UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Titulo: Indexado de Wikipedia a través de una arquitectura Map-Reduce
Autores: José Luis Larroque
Director: Alicia Diaz
Codirector: Diego Torres
Asesor profesional: -
Carrera: Licenciatura en Informática, plan 2003

Resumen

El objetivo de este trabajo de tesis es el desarrollo de un algoritmo que permite generar un índice de caminos entre dos artículos cualesquiera de Wikipedia, lo cual se logró a través de la adaptación de un algoritmo BFS preexistente. Este algoritmo fue desarrollado adaptando Wikipedia para ser procesada como un grafo en Giraph, un framework de procesamiento de grafos utilizado por grandes compañías como Facebook, Twitter, Yahoo, etc. La arquitectura de base utilizada fue Hadoop, a través de su modelo de procesamiento Map Reduce, en el cual Giraph se basa de soporte para la ejecución de algoritmos de procesamiento de grafos.
La plataforma de cómputo utilizada para ejecutar este trabajo fue Amazon Web Services, a través de la funcionalidad Elastic Map Reduce. Al ser pago dicho entorno, se usó el mismo a través de una subvención para investigación.

Palabras Claves

Giraph – Java – Map Reduce – Cloud Computing- Amazon Web Services

Conclusiones

Se logró el objetivo de poder generar los caminos posibles entre dos artículos cualesquiera de Wikipedia. Se debe continuar con el desarrollo para adaptar el algoritmo a contextos más desafiantes, como grafos más densos.

Trabajos Realizados

Se investigó cómo construir localmente un cluster en Hadoop, así como desarrollar algoritmos que corran en el. Se investigó el funcionamiento de Giraph,de forma de poder realizar algoritmos en este framework. Se desarrolló un algoritmo capaz de buscar caminos en un grafo del tamaño de Wikipedia (versión en español), usando como tecnologías principales el framework Giraph (el cual está preparado para correr en arquitecturas Map Reduce). Se probó el mismo en la plataforma de Cloud Computing Amazon Web Services. Se documentaron los resultados de estas evaluaciones y se los analizó.

Trabajos Futuros

Optimización del algoritmo adaptando el mismo a Giraph 1.2, implementación de nuevas versiones del algoritmo de búsqueda, como la búsqueda entre un inicio y todos los destinos y/o la búsqueda entre múltiples orígenes y múltiples destinos en un grafo.

Fecha de la presentación: Marzo del 2017

Indexado de Wikipedia a través de una arquitectura Map Reduce

José Luis Larroque

10 de Marzo del 2017

Resumen

Este trabajo consiste la búsqueda de distintos caminos entre distintos puntos del grafo de Wikipedia, con el fin de generar información nueva en esta enciclopedia a partir de información existente en su enciclopedia hermana semántica, DBPedia. Para este fin se diseñó un algoritmo capaz de utilizar Map Reduce en un ambiente Cloud Computing, el cual en esencia es un BFS optimizado. Este algoritmo reemplazará una parte de un algoritmo pre-existente, BlueFinder.

Índice general

Resumen....2
Índice general....3
Capítulo 1 - Introducción....5
1.1 - Objetivo....5
1.2 - Motivo....5
1.3 - Introducción....5
1.4 - Organización del informe....7
Capítulo 2 - Marco teórico....10
2.1 - Wikipedia....11
2.1.1. Qué es Wikipedia....11
2.1.2 Artículos....11
2.1.3 - Gráfico conceptual....26
2.2 - Grafos....28
2.2.1 - Recorridos de Grafos....29
2.2.2 - Alternativas tecnológicas para procesamiento de grafos en entornos Cloud 34
2.3 - Hadoop....35
2.4 - Map Reduce....35
2.5 - BlueFinder....37
2.6 - Conclusiones del capítulo....38
Capítulo 3 - BlueFinder y el origen de esta tesina....39
3.1 - DBPedia....39
3.1.1 - Qué es DBPedia....39
3.1.2 - Comparación de diferencias entre Wikipedia y DBPedia....40
3.2 - Descripción del problema a resolver....41
3.3 - Elección del tipo de recorrido de grafo a usar – BFS o DFS....42
3.4 - Giraph....43
3.4.1 - Explicación de su funcionamiento....43
3.4.2 - El método Compute....44
3.4.3 - Sincronización entre superpasos y su relación con el final del procesamiento en Giraph....46
3.4.4 - Modelo computacional de Giraph y su relación con el modelo de BSP....46
3.5 - Conclusiones del capítulo....47
Capítulo 4 - Solución Algorítmica....49
4.1 - Construcción del grafo de entrada....50
4.1.1 - Graphipedia....52
4.1.2 - Algoritmo para la construcción del grafo de entrada....54
4.2 - Algoritmo para encontrar caminos entre dos vértices del grafo de Wikipedia....57
4.2.1 - Versión con un vértice de origen y un vértice de destino....58

4.2.2 - Versión con un vértice de origen y múltiples destinos....64
4.2.3 - Explicación gráfica del funcionamiento del algoritmo....73
4.3 - Construcción del archivo de salida....82
4.3.1 - Versión con un vértice de origen y uno de destino....82
4.3.2 - Versión con un vértice de origen y múltiples destinos....84
4.4 – Enlace con BlueFinder - Construcción de base MySQL a partir del resultado del algoritmo de búsqueda de caminos navegacionales....85
4.5 - Conclusiones del capítulo....87
Capítulo 5 - Evaluación de resultados....89
5.1 - Aspectos generales de la evaluación de resultados....89
5.1.1 - Variables para la evaluación....90
5.1.2 - Métricas usadas, en qué consisten y cómo fueron calculadas....91
5.2 - Evaluaciones en Wikiquotes....92
5.3 – Evaluaciones en Wikipedia....100
5.4 - Conclusiones del capítulo....108
Capítulo 6 - Conclusiones y trabajos futuros....109
6.1 - Conclusiones de este informe final....109
6.2 - Mejoras pendientes / Posibles extensiones del trabajo....112
6.1.2 - Extensión para trabajar únicamente con vértices de origen, y abarcar todos los destinos....112
6.1.3 - Extensión para poder procesar múltiples vértices de origen a la vez....113
6.1.4 – Uso de un Combiner para reducir la cantidad de mensajes total a enviar entre vértices....113
6.1.5 - Uso de Giraph 1.2....113
Glosario....115
Apéndice....117
1 - Script guardar-logs-local.sh....117
2 - Script armar-dataset-wikipedia.sh....118
3 – Script de construcción de base MySQL bluefinderdb.sql....118
Referencias bibliográficas....120

Capítulo 1 - Introducción

1.1 - Objetivo

El objetivo de esta tesina de grado es indexar caminos navegacionales de Wikipedia a través de una arquitectura Map Reduce, usando una plataforma de Cloud Computing, partiendo desde el algoritmo BlueFinder.

1.2 - Motivo

Se buscó optimizar la forma en que el algoritmo BlueFinder encontraba caminos navegacionales, aprovechando el potencial de un framework específico de procesamiento de grafos basado en una tecnología como Map Reduce, y una plataforma de procesamiento en la nube, dado que la potencia necesaria para realizar esto a nivel local no se podía alcanzar con un cluster al que se pudiera tener acceso.

1.3 - Introducción

Wikipedia es una enciclopedia construida por una comunidad activa de múltiples usuarios a lo largo del mundo. Esto significa que continuamente se modifica su información. Al ser continuamente modificada, existe la posibilidad de que se pueda perder información. Por [5] sabemos que está situación no es una posibilidad, sino una realidad. Al comparar información existente en Wikipedia con información existente en DBPedia (la cual es un espejo semántico de Wikipedia), se puede detectar una brecha de información.

Con el fin de corregir esta brecha de información en Wikipedia, fue diseñado BlueFinder [2], el cual es un algoritmo que se encarga de recomendar enlaces faltantes en Wikipedia a través de propiedades semánticas existentes en DBPedia, generando múltiples caminos navegacionales con el fin de elegir el que mejor se corresponda. Para poder generar los múltiples caminos navegacionales, BlueFinder considera a Wikipedia como un grafo, tomando a las páginas como nodos y a los enlaces entre ellas como aristas, y realiza

un recorrido de tipo DFS para encontrar los distintos caminos existentes. Con los distintos caminos, BlueFinder construía un índice, etapa que se denomina *indizado*, para luego usar este índice para generar el mejor camino posible entre dos artículos distintos, en una etapa que se llama *recomendación*. Sin embargo, BlueFinder tiene una falencia, que es que el tiempo de cómputo que necesita la etapa de indizado para una propiedad semántica de DBPedia en particular dificulta en cierta medida su uso. En promedio, tarda semanas , por lo tanto, sería importante optimizar el tiempo que tarda esta etapa.

Al revisar la estructura de Wikipedia, en este trabajo se mostrará que la misma se asemeja a la forma de un grafo dirigido. Esta analogía es posible si se asemeja los vértices de un grafo, a los artículos de Wikipedia, y las conexiones a través de enlaces o categorías de un artículo, como aristas de dicho grafo.

Al unir la falencia a nivel de performance de BlueFinder con la forma en que Wikipedia se asemeja a una estructura de datos de un grafo, se entiende que, si se encontrase una forma más eficiente para generar los caminos en un grafo que la utilizada en BlueFinder, tendríamos una mejora de performance en dicho algoritmo, así como una nueva manera de generar la información faltante en Wikipedia.

Existiendo en la actualidad la tecnología Hadoop Map Reduce, la cual permite procesar grandes cantidades de información en un tiempo muy corto, este trabajo usará una arquitectura basada en esta tecnología para diseñar una solución algorítmica que, basándose en Map Reduce para la búsqueda de caminos en el grafo de Wikipedia, se obtenga una mejora de performance de la etapa de indizado original de BlueFinder. Esto permitirá disminuir los tiempos que tarda el algoritmo BlueFinder original, dado que se optimizará su etapa más lenta, permitiendo ejecutar esta tarea en forma eficiente en un grafo de grandes dimensiones.

Esta trabajo de tesis requirió desarrollar un conocimiento de la plataforma Hadoop, del modelo de computación paralelo Map Reduce, y del framework de procesamiento de grafos Giraph, elegido entre otras opciones por razones que más adelante se aclararán. Una vez adquiridos estos conocimientos, fue posible desarrollar un algoritmo que fuera capaz de solucionar la tarea requerida.

Una vez desarrollado el algoritmo y evaluado a nivel local su funcionamiento, se procedió a configurar un entorno de pruebas en una plataforma de Cloud Computing, principalmente por que la cantidad de datos a procesar hacían que la demanda de recursos a nivel de hardware sea difícil de alcanzar. Con este objetivo, se investigó en profundidad la forma de construir la arquitectura necesaria para ejecutar el algoritmo en la plataforma

Amazon Web Services, con el fin de obtener las mejores prestaciones posibles en dicho entorno de Cloud Computing.

Se realizaron diferentes evaluaciones de los resultados con distintas configuraciones de hardware, con el fin de justificar la eficiencia de la solución algorítmica desarrollada. También se plantearon posibles extensiones en caso de que se desee extender este trabajo en un futuro.

1.4 - Organización del informe

Esta tesina está organizada en varios capítulos. Se le informa al lector que en cada capítulo de este trabajo se han usado distintos recursos para enriquecer el contenido, como por ejemplo notas al pie de página, usadas para referenciar a sitios web o para hacer aclaraciones menores que sean pertinentes; así como un **glosario**, al cual se hará referencia cada vez que una palabra o frase se remarque con un subrayado, de modo que el lector pueda, en caso de que así lo desee, entender en mayor profundidad lo que significa. Cada capítulo puede contener una serie de imágenes, las cuales serán numeradas **por capítulo** del número 1 en adelante, y cada una de estas tendrá una descripción explicando en que consiste la misma y la fuente de donde se obtuvo dicha imagen. Cada capítulo contendrá un resumen a su inicio, y una conclusión a su final, de modo de introducir al lector en los conceptos que va a explicar, así como cerrar los temas de una forma correcta. Todos los capítulos contendrán a su inicio un índice con las secciones más importantes.

Los capítulos que conforman el presente informe final son los siguientes:

El **Capítulo 1** consiste en la introducción, el cual tiene un objetivo concreto y específico acerca de lo que se buscó realizar con este trabajo, los motivos para hacer el mismo, una introducción acerca del contenido de este trabajo y una descripción acerca de cómo está organizado.

El **Capítulo 2** contiene el marco teórico, es decir, el marco a partir del cual se empezó a desarrollar la tesina. Consiste en una explicación acerca de la forma en que que está estructurada Wikipedia, introduciendo la idea de ver a Wikipedia como un grafo. A partir de este enfoque, se explican recorridos de grafos típicos, se detallan distintas tecnologías de procesamiento de grafos, y se fundamenta la elección de una de ellas, Giraph, para poder realizar la búsqueda de caminos navegacionales. Se explica en que consiste el

modelo de procesamiento MapReduce, base de la arquitectura a utilizar en este informe, así como la plataforma Open Source que nos permitirá usarlo, Hadoop. Finalmente se introduce BlueFinder y se explica brevemente por que a partir de este algoritmo surgió el motivo para la realización de esta tesina.

El **Capítulo 3** explica el problema a resolver en esta tesina: Generar caminos navegacionales entre distintos artículos del grafo de Wikipedia utilizando tecnología Map Reduce para lograr una solución performante, a través de la utilización de dicha tecnología en una plataforma de Cloud Computing. Se inicia explicando en que consiste DBPedia, contrastándola para entender de que trata con Wikipedia. Posteriormente, se explica en profundidad el problema a resolver, dado que hasta ahora solo había sido explicado brevemente, lo cual está estrechamente relacionado con el algoritmo BlueFinder y el contexto en el que surgió, así como por que sus resultados son importantes y de que forma pueden ser utilizados. También se remarcan la falencia a nivel de rendimiento que BlueFinder tiene.

Una vez que se ha explicado el problema, se retoma la decisión pendiente del capítulo 2, en donde se debía decidir si usar DFS o BFS, y se justifica la decisión efectuada.

Se explica técnicamente en que consiste Giraph, un framework de procesamiento de grafos, para poder realizar la búsqueda de caminos navegacionales en una arquitectura Map Reduce, en mejores tiempos y con mayor cantidades de datos que el algoritmo BlueFinder original. Se explica como realizar un algoritmo de camino mínimo con este framework, para poder explicar mejor su funcionamiento.

El **Capítulo 4** explica en profundidad el desarrollo algorítmico realizado. Se explicará cómo se adaptó la información de Wikipedia para ser procesada en Giraph, se detallará el algoritmo paso a paso (en sus dos versiones), así como los problemas más relevantes encontrados en el desarrollo de los mismos, y por último se describirá que tipo de información resultante produce este algoritmo.

El **Capítulo 5** consiste en la evaluación de resultados, tanto usando Wikiquotes como Wikipedia. Se analizaran las dos versiones del algoritmo, con distintos conjuntos de datos según se corresponda con Wikiquotes o Wikipedia. Se analizarán distintas métricas, como por ejemplo cantidad de caminos navegacionales encontrados, tiempo promedio en realizar una búsqueda de caminos navegacionales, cantidad de búsquedas fallidas, etc. Este análisis se hará a medida que se incrementa la envergadura de los recursos utilizados, como pueden ser cantidad de memoria RAM por unidad de procesamiento, cantidad de unidades de procesamiento, etc.

El **Capítulo 6** contiene las conclusiones acerca del trabajo realizado y posibles extensiones del mismo.

Capítulo 2 - Marco teórico

Capítulo 2 - Marco teórico....10
2.1 - Wikipedia....11
2.1.1. Qué es Wikipedia....11
2.1.2 Artículos....11
2.1.2.1 - Enlaces....13
2.1.2.1.1 - Enlaces Internos....13
2.1.2.1.2 - Enlaces Externos....14
2.1.2.1.2.1 - Enlaces externos directos....14
2.1.2.1.2.2 Enlaces externos indirectos....15
2.1.2.2 - Categorías....17
2.1.2.2.1 - Jerarquía de categorías....18
2.1.2.3 - Estructura de un artículo....20
2.1.2.4 - Creación de un artículo....23
2.1.2.5 - Edición de un artículo....23
2.1.3 - Gráfico conceptual....26
2.2 - Grafos....28
2.2.1 - Recorridos de Grafos....29
2.2.1.1 - Depth First Search....29
2.2.1.2 - Breadth First Search....31
2.2.2 - Alternativas tecnológicas para procesamiento de grafos en entornos Cloud 34
2.3 - Hadoop....35
2.4 - Map Reduce....35
2.5 - BlueFinder....37
2.6 - Conclusiones del capítulo....38

Este capítulo se inicia introduciendo al lector con una explicación acerca de Wikipedia, así como la forma en que está organizada la información que en ella reside. Se introduce la razón qué asemeja la información presente en Wikipedia a la estructura de un Grafo dirigido.

Luego se explican recorridos de grafos, fundamentales debido a que necesitamos recorrer la estructura de Wikipedia, y al asemejarse dicha enciclopedia a uno de ellos, se presentarán distintas formas de recorrerla, justificando la decisión de cúal algoritmo de

recorrido de grafos es el mejor para esta tarea en otro capítulo. Se explicaran que tecnologías de grafos existen en el mercado, y se fundamentará la decisión de por que se elige a una de ellas.

Habiendo explicado estos conceptos, se explicará que es Map Reduce, concepto fundamental en este informe, dado que es la base de la arquitectura desde donde se buscará una solución algorítmica al problema que intenta resolver este trabajo, es decir, el indexado de caminos navegacionales en Wikipedia.

Por último, se introducirá el algoritmo BlueFinder, y se explicará por que este algoritmo dio origen a esta tesina.

2.1 - Wikipedia

2.1.1. Qué es Wikipedia

Wikipedia es un sitio web, el cual se define como una “Enciclopedia **libre** y **gratuita**, que todos pueden **editar**”. Se considera que su acceso es libre porque puede accederse desde cualquier explorador de internet, gratuita dado que no se cobra por su uso, y la información que se presenta es editable por cualquiera por que todos pueden editar su contenido, más información en la sección *Edición de un artículo*.

Desde su página principal [1], se puede ver que el contenido de esta enciclopedia está dividido acorde al idioma en que fue escrito. Cada idioma se considera una **Wikipedia**, es decir, una enciclopedia por si misma. Existen trece idiomas con más de un millón de artículos y 295 idiomas en total[2]. El idioma inglés es el que más artículos tiene.

A continuación, se explica en que consiste un Artículo.

2.1.2 Artículos

Cada Wikipedia ,es decir, cada idioma en la cual está escrita esta enciclopedia, consta de **artículos**[3], los cuales son “Páginas con contenido enciclopédico” [4] dentro del mismo sitio de Wikipedia. Los artículos pueden ser vistos como la unidad a partir de la cual

1 Página principal, Wikipedia, 1/11/2016 <https://www.wikipedia.org/>
2 Language Editions, Wikipedia, 24/11/2016 <https://en.wikipedia.org/wiki/Wikipedia#Language_editions>
3 También llamados páginas en la diversa documentación de la misma Wikipedia, sinonimia que también será válida a lo largo de todo este informe final.
4 Página de ayuda, explicación de en que consiste un articulo, Wikipedia, 1/11/2016 <https://es.wikipedia.org/wiki/Ayuda:Artículo>

fue construida, poco a poco, cada Wikipedia. Son fuente de información, al igual que sucede con los mismos en una enciclopedia tradicional.

El contenido de los artículos debe tener un “punto de vista neutral”, y estar basado en “conocimiento humano existente” [5]. No puede ser una página de auto-promoción o publicidad, ensayos o investigaciones personales[6].

Los artículos deben ser escritos en un lenguaje de marcas especial llamado Wikitexto[7], el cual no dispone de un estándar que lo defina, sino que depende del software utilizado[8]. En el caso de Wikipedia, es gestionada por el software libre MediaWiki[9]. Este lenguaje nos permite enriquecer el contenido de cada página de está enciclopedia, para que su contenido no sea solo texto plano.

En todo momento un artículo puede ser visto de dos formas, una es la vista de edición, explicada en la sección Edición de un Artículo, y la otra es la vista normal, en la cual se ven reflejados los cambios realizados en la vista de edición.

El uso de Wikitexto se logra a través de una sintaxis especial, la cual es la que nos permite indicar como se enriquecerá el texto. Por ejemplo, si se desea poner un texto plano **cursiva**, se debe ubicar entre dos comillas simples de cada lado, usando Wikitexto de la siguiente forma:

‘’argentum’’

de forma de obtener el resultado que indica la Imagen 2-1.

El nombre «Argentina» proviene del latín *argentum* (‘plata’) y está asociado a la leyenda de la Sierra de la Plata, común entre los primeros exploradores europeos de la región, tanto españoles como portugueses. Fueron estos quienes denominaron Rio da Prata (‘Río de la Plata’) al gran estuario descubierto por la expedición portuguesa de 1502 en la que participaba Américo Vespucio, y al que luego llegó Juan Díaz de Solís en 1516, llamándolo Mar Dulce.

Imagen 2-1: Uso de cursiva en un texto

Fuente: Wikipedia, Artículo de Argentina, con edición propia

5 Página de ayuda, “Pero por favor no escribas…”, Wikipedia, 2/11/2016 <https://es.wikipedia.org/wiki/Ayuda:Tu_primer_artículo#Pero_por_favor.2C_no_escribas...>
6 Página de ayuda, Wikipedia “Pero por favor no escribas…” , 2/11/2016 <https://es.wikipedia.org/wiki/Ayuda:Tu_primer_artículo#Pero_por_favor.2C_no_escribas...>
7 Wikitexto, Wikipedia, 10/12/2016 <https://es.wikipedia.org/wiki/Wikitexto>
8 Software para Wikis, Wikipedia, 10/12/2016 <https://es.wikipedia.org/wiki/Software_para_wikis>

Este es un ejemplo muy simple del uso de Wikitexto. Hay ejemplos mucho más complejos de su uso, como por ejemplo los Enlaces de un artículo, uno de los conceptos que más importan de Wikipedia en lo que tiene que ver al objetivo de este informe, y que es explicado a continuación.

2.1.2.1 - Enlaces

Un enlace nos permite navegar de un sitio a otro, tanto en Internet como en cualquier red que disponga de distintos sitios. En el caso de Wikipedia, este nos permite ir de un Artículo a otro, así como de un Artículo a un sitio externo.

En palabras de la misma Wikipedia[10], "*Es muy importante enlazar entre sí los artículos de Wikipedia. Estos enlaces permiten a los usuarios acceder a información relacionada al artículo que están leyendo.*"

El artículo de Argentina, así como cualquier otro artículo de Wikipedia, contiene múltiples enlaces a diversos lugares, los cuales permiten enriquecer el contenido del mismo, permitiéndole al lector profundizar en los aspectos que crea conveniente.

En Wikipedia, hay dos tipos de enlaces: internos y externos. Ambos serán explicados a continuación, usando de ejemplo el artículo de Argentina referenciado previamente.

2.1.2.1.1 - Enlaces Internos

Permiten enlazar a los artículos de Wikipedia entre sí, de modo que el usuario pueda ampliar su conocimiento a través del acceso a otros artículos de la misma enciclopedia.

Para crear un enlace interno, se debe usar la siguiente notación de lenguaje de Wikitexto:

[[Título de la página | Título del enlace]]

Se debe escribir en *Título de la página* el nombre de otra página de Wikipedia. Si se quiere que el enlace se muestre con palabras diferentes a las del título del artículo al que se

9 MediaWiki, 10/12/2016, <https://www.mediawiki.org/wiki/MediaWiki/es>
10 Tutorial de Enlaces Internos, Wikipedia, 9/9/2015 <https://es.wikipedia.org/wiki/Ayuda:Tutorial_(enlaces_internos)>

quiere enlazar, se puede escribir el nombre alternativo en *Título del enlace* (el cual siempre estará después del símbolo de una línea vertical: |).

Una vez que se haya insertado el enlace en el artículo, Wikipedia comprobará si existe o no. Si se quiere insertar un enlace a un artículo inexistente en Wikipedia, este se convertirá en un **red link**[11], es decir, un enlace de color rojo**,** una vez que se haya grabado la edición que se está realizando del artículo[12]. Si el artículo existe, el enlace será de un **color azul**. Estos dos colores le permiten a Wikipedia graficar de una forma muy precisa si el enlace direcciona a un artículo existente, o a uno inexistente.

Por ejemplo, en el artículo de Argentina, un enlace a *Estado Nación*, está escrito de la siguiente manera:

[[Estado nación|país soberano]]

Y figura en Wikipedia como indica la Imagen 2-2.

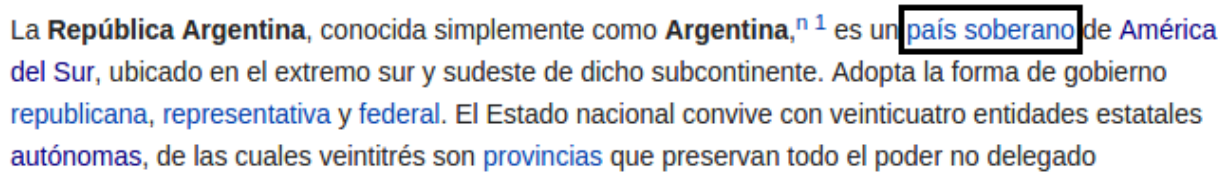

Imagen 2-2: Ejemplo de enlace interno
Fuente: Wikipedia, Artículo de Argentina (editada)

2.1.2.1.2 - Enlaces Externos

Dentro de los enlaces externos, es decir, enlaces a contenido que no forma parte de Wikipedia, se consideran dos tipos, *Directos* e *Indirectos.* A continuación, se explican sus diferencias.

2.1.2.1.2.1 - Enlaces externos directos

Al escribir directamente una dirección web de una página que se desea enlazar, Wikipedia detectará que empieza con el prefijo http://, lo reconocerá automáticamente como un enlace externo, y mostrará la dirección completa. Por ejemplo, si se escribe:

11 Red Links, Wikipedia (inglés), 17/12/2016 <https://simple.wikipedia.org/wiki/Wikipedia:Red_link>
12 Los “Red Links” nos permite crear un Artículo. Más información en la sección *Creación de un Artículo.*

http://www.economist.com/node/21548229

Se verá como indica la Imagen 2-3.

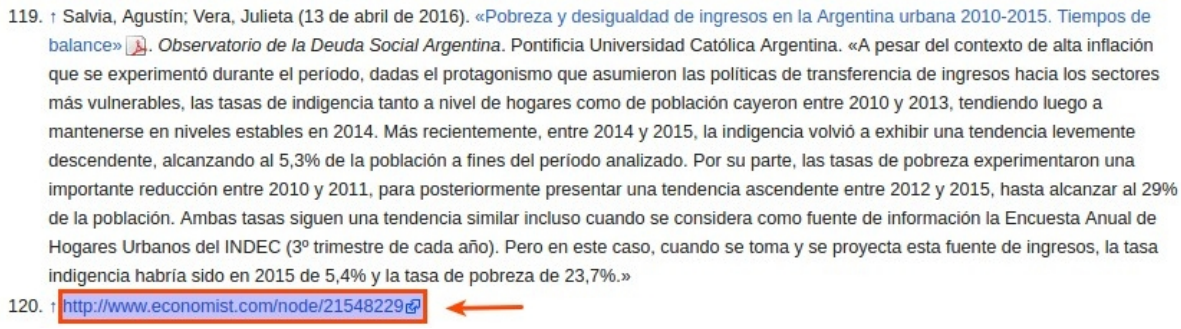
119. ↑ Salvia, Agustín; Vera, Julieta (13 de abril de 2016). «Pobreza y desigualdad de ingresos en la Argentina urbana 2010-2015. Tiempos de balance». *Observatorio de la Deuda Social Argentina*. Pontificia Universidad Católica Argentina. «A pesar del contexto de alta inflación que se experimentó durante el período, dadas el protagonismo que asumieron las políticas de transferencia de ingresos hacia los sectores más vulnerables, las tasas de indigencia tanto a nivel de hogares como de población cayeron entre 2010 y 2013, tendiendo luego a mantenerse en niveles estables en 2014. Más recientemente, entre 2014 y 2015, la indigencia volvió a exhibir una tendencia levemente descendente, alcanzando al 5,3% de la población a fines del período analizado. Por su parte, las tasas de pobreza experimentaron una importante reducción entre 2010 y 2011, para posteriormente presentar una tendencia ascendente entre 2012 y 2015, hasta alcanzar al 29% de la población. Ambas tasas siguen una tendencia similar incluso cuando se considera como fuente de información la Encuesta Anual de Hogares Urbanos del INDEC (3º trimestre de cada año). Pero en este caso, cuando se toma y se proyecta esta fuente de ingresos, la tasa indigencia habría sido en 2015 de 5,4% y la tasa de pobreza de 23,7%.»
120. ↑ http://www.economist.com/node/21548229

Imagen 2-3: Ejemplo de enlace externo directo y sin nombre
Fuente: Wikipedia, Artículo de Argentina (editada)

Dado que de esta forma no se puede ofrecer una descripción adecuada del sitio enlazado, esta opción está desaconsejada para su uso.

2.1.2.1.2.2 Enlaces externos indirectos

La segunda forma de crear enlaces externos es incluir una descripción que indique el contenido de la dirección web de la página. Esta descripción aparecerá en reemplazo de la dirección web. Se deberá usar la siguiente notación:

[url_del_sitio_externo nombre_dado_al_sitio_externo]

Donde url_del_sitio_externo seria una dirección web, como puede ser la de Google por ejemplo (http://www.google.com) y donde *nombre_dado_al_sitio_externo* sería el nombre que verá el lector, por ejemplo *Motor de búsqueda de Google*. Si se aplican estos ejemplos al formato de un enlace externo en Wikipedia, la notación sería:

[http://www.google.com Motor de búsqueda de Google]

El cual se vería visualmente de forma muy parecida a un Enlace Interno, salvo por una flecha a su derecha, la cual puede ser visualizada en la Imagen 2-4. Un ejemplo en el artículo de Argentina sería el indicado por la Imagen 2-5.

Imagen 2-4: Flecha - Enlace externo indirecto

Fuente: Wikipedia, Artículo de Argentina

Portal oficial del Estado argentino

Imagen 2-5: Enlace externo con texto

Fuente: Wikipedia, Artículo de Argentina

Es importante que se deje el enlace en en la sección de Enlaces externos, tal cual muestra la Imagen 2-6. El resultado gráfico de la Imagen 2-6 se puede ver en la Imagen 2-7.

```
== Enlaces externos ==
{{commons |Argentina}}
{{wikiatlas |Argentina}}
{{wikcionario |Argentina}}
{{wikisource |Argentina}}
{{Wikinoticias |Argentina}}
{{wikiquote}}
{{wikiviajes |Argentina}}

* {{web oficial |www.argentina.ar}}
* [http://www.argentina.gob.ar/ Portal oficial del Estado argentino]
* [http://www.casarosada.gob.ar/ Sitio oficial de la Presidencia de la Nación]
* [http://www.turismo.gob.ar/ Sitio oficial de la Secretaría de Turismo]
* [http://www.fao.org/countryprofiles/index.asp?lang=es&iso3=ARG&subj=1&paia= Perfiles de países: Argentina]

{{Ubicación geográfica
| Noroeste = {{bandera|Chile}} [[Chile]] / {{bandera|Bolivia}} [[Bolivia]]
| Norte = {{bandera|Bolivia}} [[Bolivia]] / {{Bandera|Paraguay}} [[Paraguay]]
| Noreste = {{Bandera|Paraguay}} [[Paraguay]] / {{Bandera|Brasil}} [[Brasil]]
| Oeste = {{Bandera|Chile}} [[Chile]]
| Centro = Argentina
| Este = {{Bandera|Brasil}} [[Brasil]]<br />{{bandera|Uruguay}} [[Uruguay]]
| Suroeste = {{bandera|Chile}} [[Chile]]
| Sur = [[Canal de Beagle]] / [[Pasaje de Drake]] o [[Mar de Hoces]] (sector del océano Atlántico Sur).
| Sureste = [[Océano Atlántico]]}}
```

Imagen 2-6: Formato de la zona de Enlaces Externos

Fuente: Wikipedia, Artículo de Argentina, Wikitexto (editada)

Imagen 2-7: Sección de enlaces externos

Fuente: Wikipedia, Artículo de Argentina, sección Enlaces Externos (editada)

Por ejemplo, en la Imagen 2-7 se puede ver que hay múltiples enlaces externos, entre ellos, el del Portal oficial del Estado argentino[13].

En adición a los Enlaces, existe otro elemento en Wikipedia que nos permite enlazar dos páginas entre si, este son las Categorías, las cuales son el segundo concepto más importante que tiene Wikipedia en sus páginas en base a los objetivos de este informe[14]. Este concepto, el de las Categorías, es explicado a continuación.

2.1.2.2 - Categorías

Según la propia Wikipedia “Una **categoría** es una agrupación de páginas que comparten algún tema en común” [15]. Es decir, una categoría es un método que nos brinda Wikipedia para agrupar contenido (enlaces) que tiene algo en común. Un sencillo ejemplo es el artículo de Aristóteles[16], el cual tiene una categoría llamada Filósofos de la antigua Grecia[17], categoría que agrupa los distintos filósofos que hubo en ese país en la antigüedad.

13 Portal oficial del Estado Argentino, 04/11/2016 <http://www.argentina.gob.ar/>
14 El primer concepto eran los enlaces, explicados en la sección inmediatamente anterior.
15 Wikipedia, página de ayuda: Categoría, 2/11/2016 https://es.wikipedia.org/wiki/Ayuda:Categoría
16 Aristóteles, Wikipedia, 20/12/2016 <https://es.wikipedia.org/wiki/Aristóteles>
17 Filósofos de la antigua Grecia, Categoría de Wikipedia , 20/12/2016, <https://es.wikipedia.org/wiki/Categoría:Filósofos_de_la_Antigua_Grecia>

Otro ejemplo, en línea con el resto de los ejemplos de este informe, es el artículo de Argentina, el cual está incluido en la categoría Argentina[18]. Se debe notar que ambas cosas son distintas, mientras el artículo de Argentina da una explicación detallada de este país, la categoría Argentina (Categoría:Argentina) es una categoría que engloba a muchas otras subcategorías, que obviamente tienen que ver con este país, tales como Economía de Argentina[19] y Política de Argentina[20] . Además, la categoría Argentina referencia al artículo principal de la misma, dado que este artículo es su **artículo de cabecera**. Por ende, se puede ver que a través de la Categoría Argentina, se puede agrupar información relativa a la categoría en sí, mientras que en el artículo Argentina se encuentra información relacionada con el país sudamericano.

Todo artículo debería pertenecer al menos a una categoría (salvo escasas excepciones), y también puede pertenecer a múltiples categorías a la vez.

El proceso de incluir en un artículo a una categoría se denomina *categorización*, y se realiza editando la página que se desea categorizar.

Según la misma Wikipedia[21]*, "las categorías permiten ver los artículos contenidas en cada una de ellas, o bien navegar a través de ellas, gracias a su* ordenación jerárquica*. Una página no categorizada se considera tan* ***huérfana*** *como una sin enlaces internos".*

En toda Categoría, Wikipedia automáticamente nos brinda enlaces a través de los cuales podemos navegar hacia las páginas que están categorizadas con ella, por ejemplo si accedemos a la página de la Categoría: Argentina, tendremos un enlace hacia el Artículo de Argentina. También nos permite navegar fácilmente a las categorías de las cuales es subcategoría, por ejemplo retomando el ejemplo de Categoría: Argentina, esta nos permite navegar hacia la Categoría Miembros del Mercosur[22].

A continuación, se explica en profundidad el concepto de la ordenación jerárquica de las categorías.

2.1.2.2.1 - Jerarquía de categorías

18 Wikipedia, categoría del artículo Argentina, 2/11/2016 <https://es.wikipedia.org/wiki/Categoría:Argentina>
19 Economía argentina, Wikipedia, 2/11/2016 <https://es.wikipedia.org/wiki/Categoría:Economía_de_Argentina>
20 Política argentina, Wikipedia, 2/11/2016 <https://es.wikipedia.org/wiki/Categoría:Política_de_Argentina>
21 Página de ayuda: Categoría, Wikipedia, 2/11/2016 <https://es.wikipedia.org/wiki/Ayuda:Categoría>
22 Categoría Miembros del Mercosur, Wikipedia, 10/12/2016 <https://es.wikipedia.org/wiki/Categoría:Miembros_del_Mercosur>

Todas las categorías existentes pertenecen a una jerarquía de categorías, la cual se inicia a partir del “Índice de categorías”[23] .

Dicha categoría incluye las siguientes 5 subcategorías:

- Administración de Wikipedia
    - Clasifica los artículos relacionados con el desarrollo y la administración de Wikipedia
- Anexos
    - Es la categoría base para todas las páginas que ofrecen información de soporte enciclopédico y que están alojadas en ese espacio de nombres.
- Artículos
    - Lleva al esquema de clasificación básico de Wikipedia, organizado en cuatro grandes áreas: Ciencia, Arte, Naturaleza y Sociedad.
- Ayuda
    - Clasifica los manuales escritos para conocer mejor el funcionamiento de la Wikipedia.
- Categorías
    - Categorías que agrupan a otras categorías por características de las categorías en sí, más que de los artículos que contienen.

A su vez, cada una de estas categorías se extiende en otras, tal cual muestra el siguiente organigrama de los niveles más altos de la estructura jerárquica de Wikipedia en español:

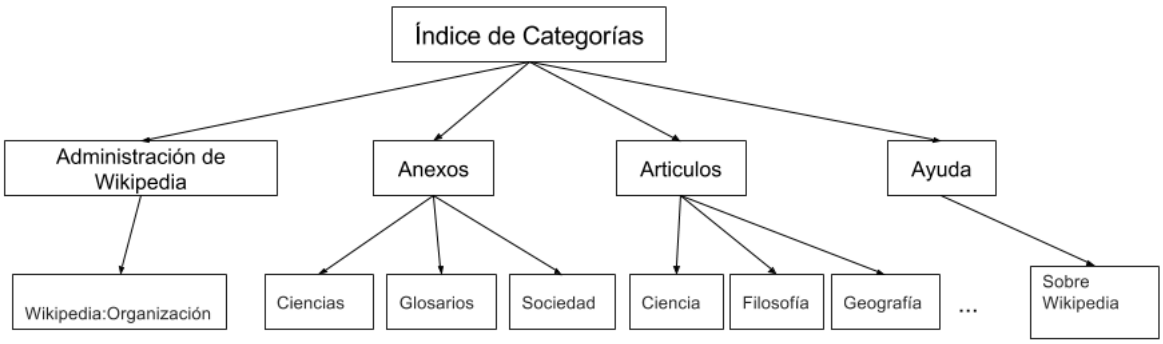

Imagen 2-8: Estructura jerárquica de categorías

Fuente: Elaboración propia

23 Índice de Categorías, Wikipedia, 5/12/2016 <https://es.wikipedia.org/wiki/Categoría:Índice_de_categorías>

Habiéndose explicado los conceptos más importantes para este informe que tiene Wikipedia en cada uno de sus Artículos, se explicará a continuación la estructura de dichos Artículos en si.

2.1.2.3 - Estructura de un artículo

Las principales secciones en las cuales está estructurado un artículo son las siguientes:

- Título

El título de un artículo se ubica al principio del mismo . Permite indicar acerca de qué trata el artículo y nos sirve para poder distinguir un artículo por sobre el resto. Según la política oficial[24] de la misma Wikipedia: "*deben ser formulados de tal manera que puedan ser localizados con la mayor facilidad posible por las personas hispanohablantes que consultan Wikipedia en español* "

- Sección introductoria:

Esta sección se ubica a continuación del *Título, está compuesta por no más de cinco párrafos, y debe contener los siguientes puntos:*

  - Definición lo más concisa posible del tema del artículo, generalmente con una mención en negrita del título.
  - Algún/os párrafo/s para poner en contexto el tema del artículo, ofrecer la información más relevante de este y mostrar por qué el tema del artículo es notable e interesante.
  - Índice
  - Fichas informativas (Opcional)
  - Imágenes (Opcional)

- Cuerpo principal del artículo

Cada artículo tiene un cuerpo principal, el cual está dividido en secciones, para organizar su contenido y ayudar al lector a encontrar lo que busca. Cada sección puede contener enlaces (ver sección *Enlaces*) a otras páginas de Wikipedia.

- Véase también

Esta sección está compuesta de *Enlaces* que puedan ser de interés para el lector, hacia temas relacionados al artículo previamente leído.

---

24 Convenciones de Títulos, Wikipedia, 9/9/2015: <https://es.wikipedia.org/wiki/Wikipedia:Convenciones_de_títulos>

- Notas

Referencias textuales tomadas de algún lugar, tal como un sitio web, la página de un libro, un artículo académico, etc.

- Referencias
    - Bibliografía utilizada

Lista con la bibliografía principal utilizada para redactar el artículo.

    - Bibliografía adicional

Lista con bibliografía adicional, que no haya aparecido en la sección *Notas* o *Bibliografía utilizada* y que pueda servir para ampliar los contenidos del artículo en sí.

    - Enlaces externos

Sitios externos, ajenos a Wikipedia, utilizados para el desarrollo del artículo.

- Categorías: Se indica una por una las categorías a las que pertenece el artículo, sin mucho más detalle.

Imagen 2-9: Estructura de un Artículo

Fuente: Wikipedia, Artículo de Argentina, con edición propia

Para explicar la estructura de un artículo, se usará de ejemplo el artículo de Argentina [25] que tiene como título *Argentina*. En la Imagen 2-9 se pueden ver las partes recientemente especificadas [26] [27]

A continuación, se explicará como se construye un Artículo para que respete correctamente la Estructura de un Artículo, es decir, se explicará en que consiste el proceso de Creación de un Artículo y como se realiza Edición de un Artículo existente.

25 República Argentina, Wikipedia, 09/09/2015 <https://en.wikipedia.org/wiki/Argentina>
26 Estructura de un artículo, Wikipedia, 09/09/2015 <https://es.wikipedia.org/wiki/Wikipedia:Estructura_de_un_artículo>
27 Manual de estilo, Wikipedia, 09/09/2015 <https://en.wikipedia.org/wiki/Wikipedia:Manual_of_Style/Layout>

2.1.2.4 - Creación de un artículo

Existen dos formas para poder crear un Artículo en Wikipedia:

1. Mientras se edita el Wikitexto de una página existente, se puede poner texto entre dobles corchetes como, por ejemplo:

[[mi página]]

(ver sección de *Enlaces* para más detalles). Cuando se guarde esta página, aparecerá un enlace azul si la página ya existe, o rojo si aún no. En el segundo caso, al pulsar sobre él, se irá a una nueva página que puede ser editada. Se debe reemplazar el texto «Escribe aquí el contenido de la página» con el texto que se desee introducir en el nuevo artículo.

2. Teclear el título del nuevo articulo precedido de «http://es.wikipedia.org/wiki/» (en el caso de la Wikipedia en español) en un navegador. Wikipedia nos enviará a una página con el título que se eligió para el artículo y se empezará a editarla.

El modo recomendado es el primero. Se recomienda dicho modo para que los artículos queden enlazados entre ellos y sean más fácil de ser encontrados. Al crear artículos, se debe comprobar antes que realmente no existía y se debe asegurar que el nuevo artículo cumple con políticas oficiales de Wikipedia acerca del tema [28].

2.1.2.5 - Edición de un artículo

En Wikipedia, todos los Artículos pueden ser cambiados, o editados, por sus propios usuarios. En cada artículo, existe una solapa, arriba a la izquierda, que permite editarlo, inclusive sin estar registrado. Tan solo basta con presionar el botón **Editar** situado en la esquina superior derecha de la página de Wikipedia, ver Imagen 2-10 para más detalle.

28 Lo que Wikipedia no es, Wikipedia, 9/9/2015 <https://es.wikipedia.org/wiki/Wikipedia:Lo_que_Wikipedia_no_es>

Imagen 2-10: Edición de un Artículo

Fuente: Wikipedia, Artículo de Argentina, con edición propia

Para poder editar un artículo correctamente, es necesario conocer como es el formato con en el que se escribe en Wikipedia, dado que de otro modo no se podrá lograr dicho cometido. Al presionar el botón Editar, se abre la sección de Edición, la cual tiene algunos accesos directos a herramientas básicas de edición cómo escribir texto en negrita, escribir en cursiva, insertar enlaces, etc. También tiene una sección en donde el usuario puede escribir, usando el formato mencionado más arriba. Por ejemplo, para escribir texto en **Negrita** se debe usar tres veces una comilla simple, de cada lado de la palabra o frase que se desee poner en negrita. Si se quiere que la frase *República Argentina* salga en negrita, debe escribirse así:

'''República Argentina'''

Este ejemplo, en la página de Wikipedia, se ve como indica la Imagen 2-11:

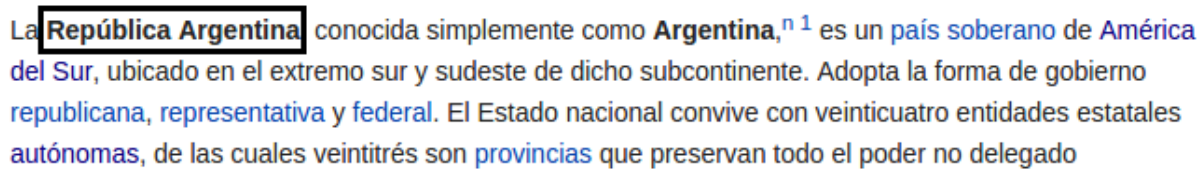

Imagen 2-11: Uso de negrita en un texto

Fuente: Wikipedia, Artículo de Argentina, con edición propia

Wikipedia nos permite previsualizar cómo quedará el artículo antes de terminar su edición, para conocer lo antes posible como se verá el artículo que se está editando. Para

poder realizar esto, se debe presionar el botón *Mostrar previsualización*, tal como indica la Imagen 2-12:

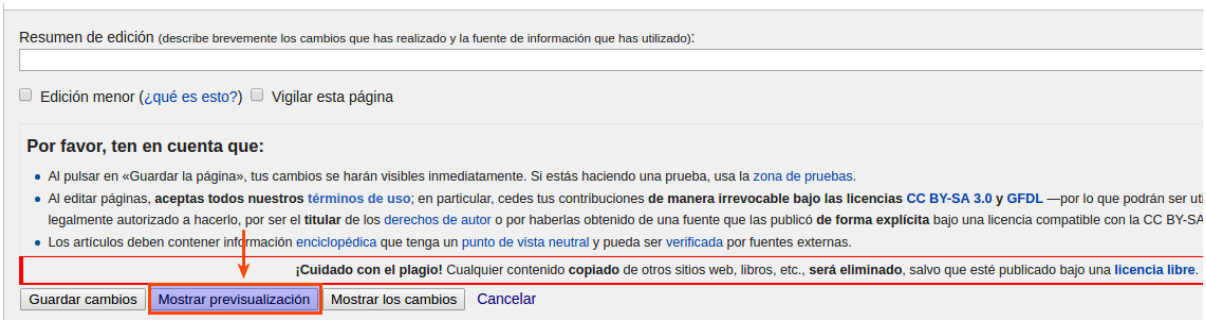

Imagen 2-12: Previsualización de una edición

Fuente: Wikipedia, junto con edición propia

También se pueden insertar viñetas, imágenes, títulos de distinto tamaño, referencias, etc (No se detallan cada uno de estos elementos dado que no es intención de este informe ser un manual de Wikipedia).

El último paso de una edición se da al guardar los cambios, de modo que el artículo que se está modificando, quede modificado definitivamente. Esto se realiza al apretar el botón *Guardar cambios*, ver Imagen 2-13.

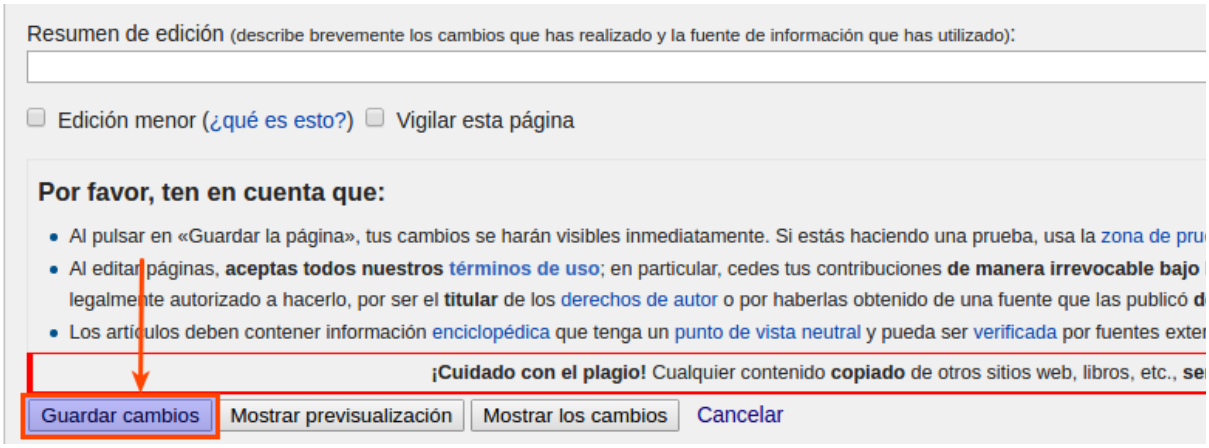

Imagen 2-13: Guardado de cambios al editar

Fuente: Wikipedia, junto con edición propia

En resumen, los pasos básicos para editar un artículo son los siguientes:

1. Hacer clic en editar en la parte superior de un artículo.
2. Escribir o modificar un texto.

3. Hacer clic en Mostrar previsualización para probar cómo quedarían los cambios.
4. Hacer clic en **Guardar cambios** en la parte inferior para guardar los cambios.

2.1.3 - Gráfico conceptual

El gráfico de la Imagen 2-14, es la conceptualización de los conceptos más importantes acerca de Wikipedia para este trabajo. Nos permite ver que en Wikipedia existen artículos y categorías, ambos representados como rectángulos. Las líneas representan enlaces o categorías, en cada una se aclara a que corresponde.

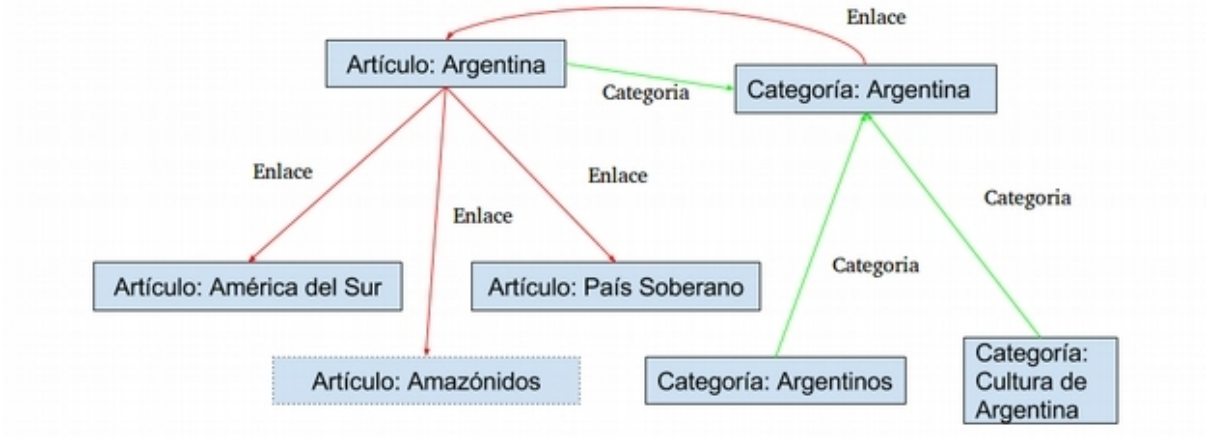

Imagen 2-14: Gráfico conceptual con los distintos elementos de Wikipedia

Fuente: Elaboración propia

Se observa que el artículo *Argentina* tiene enlaces a los artículos de *América del Sur* [29] y *País Soberano* [30]. También se puede notar que el artículo *Argentina* tiene un enlace a un artículo *Amazónidos* el cual no fue creado aun, lo cual fue representado con un rectángulo de contorno de línea de puntos, en lugar de una línea sólida. Las categorías Argentinos y *Cultura de Argentina*[31] son subcategorías de *Categoría:Argentina*. La categoría Argentina tiene como artículo principal al artículo *Argentina*, y este artículo tiene como categoría a la *Categoría: Argentina*.

29 Wikipedia, artículo de América del Sur, accedido el 02/11/2016 <https://es.wikipedia.org/wiki/América_del_Sur>
30 Wikipedia, artículo de País Soberano, accedido el 02/11/2016 <https://es.wikipedia.org/wiki/Soberano>
31 Categoría Cultura de Argentina, Wikipedia, 02/11/06 <https://es.wikipedia.org/wiki/Categoría:Cultura_de_Argentina>

Se puede observar que la naturaleza de la información almacenada en Wikipedia, únicamente teniendo en cuenta los conceptos representados en el gráfico, se asemeja a lo que una estructura de datos particular. A continuación, se explica cual estructura de datos es y por que este dato es importante en relación a este trabajo.

2.2 - Grafos

Un **grafo** G = (V, E) está formado por un conjunto de vértices V, y un conjunto de aristas, E. Cada arista es un par (v, w), donde v y w son vértices pertenecientes a V[32]. Por ejemplo, en el grafo de la Imagen 2-15 se puede observar que tiene 4 vértices A, B, C y D. Y que tiene aristas entre A y B, B y C, A y C, y C con D.

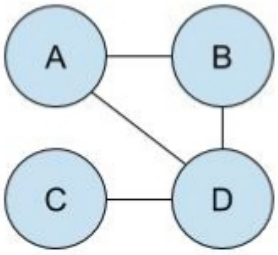

Imagen 2-15: Grafo no dirigido
Fuente: Edición propia

Este tipo de grafos se conoce, en la teoría de grafos, como **grafo no dirigido**, dado que los pares de vértices (v, w) que conforman sus aristas no tienen orden[33] . Es decir, al unir A y C a través de una arista, se une ambos nodos al mismo tiempo, y nos permite, si tuviéramos que recorrerlo, tanto *ir* desde A hasta C, como desde C hasta A, por ejemplo.

Si las aristas (v, w) del grafo tienen un orden definido, entonces se debe referir a este como un **grafo dirigido[34]**. Esto significa que una arista tiene un único sentido definido, y para que haya navegabilidad completa entre dos vértices distintos, deben existir dos aristas distintas entre dichos vértices.

Habiendo introducido estas definiciones acerca de los grafos, el lector podrá observar que Wikipedia se asemeja a la estructura de un grafo dirigido, si tomamos a los cuadrados de la Imagen 2-14 como nodos, y a las líneas existentes entre estos como aristas. Dado que se necesita poder recorrer todo el grafo de Wikipedia, se necesitará usar algoritmos de recorridos de grafos que nos brinden esta posibilidad. Dichos algoritmos se explican a continuación.

32 Capítulo de Algoritmos de grafos, sección Definiciones, página 359 de [11].
33 Ídem pie de página 32
34 Ídem pié de página 32

2.2.1 - Recorridos de Grafos

En la literatura asociada a esta estructura de datos, existen diversidad de algoritmos que nos permiten recorrer el grafo[35], pero son pocos los que están preparados para recorrer el grafo en su totalidad, sin buscar algún resultado en particular (como puede ser, el camino mínimo entre dos nodos).

Los recorridos de grafos más simples que permiten recorrer todo el grafo, son el Depth First Search[36] y el Breadth First Search[37], se explicarán ambos a continuación.

2.2.1.1 - Depth First Search

El algoritmo **DFS**, o **Depth First Search** (búsqueda en profundidad) nos permite recorrer todos los nodos de un grafo, recorriendo en profundidad cada nodo del grafo, antes de seguir con el vértice adyacente. A través de la Imagen 2-16 se aclarará como trabaja este algoritmo:

```
void dfs( Vertex v )
{
        v.visited = true;
        for each Vertex w adjacent to v
                if ( !w.visited )
                        dfs( w );
}
```

Imagen 2-16: DFS en pseudo-código
Fuente: [11], pagina 400

El algoritmo empieza en un vértice v, el cual debe ser elegido previamente. Una vez que se ha procesado v, lo cual de acuerdo a las necesidades de cada implementación en particular de este algoritmo puede tomar muchos significados distintos, se recorren todos los vértices adyacentes a v. Cada vez que se procesa un vértice, este se marca como

35 Capítulo 9, de [11]
36 Página 399, “Aplicaciones de la búsqueda Depth First Search”, capítulo 9, de [11]
37 Página 368, capítulo 9 de [11]

visitado, con el fin de evitar los ciclos. Al detectar un vértice adyacente que no ha sido visitado, se lo recorre de la misma forma, es decir, se recorre en profundidad este nodo, antes de continuar con un adyacente a el. Este funcionamiento se repite hasta recorrer todos los nodos del grafo.

A continuación, se usará el grafo de la Imagen 2-17 de ejemplo para mostrar su funcionamiento

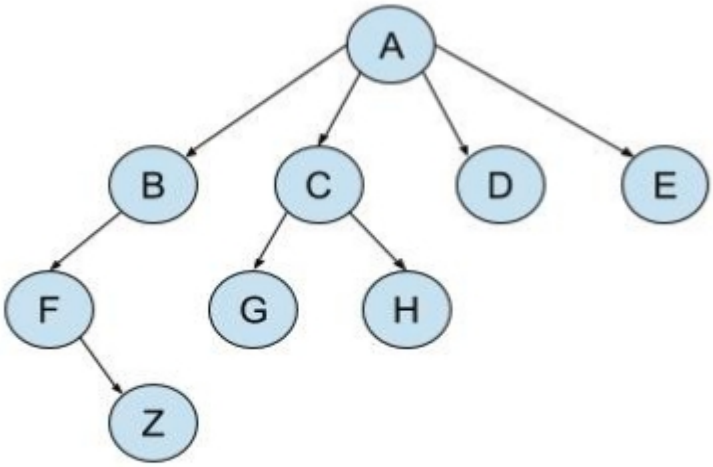

Imagen 2-17: Grafo de ejemplo
Fuente: Creación propia

1. Se elige a A como inicio del algoritmo DFS. Se lo marca como visitado (es decir, se le da el valor true a la propiedad *visited*), y se analizan los vértices adyacentes, es decir, B, C, D y E. Como el vértice B no ha sido visitado aún, se inicia un DFS a partir de ese vértice.
2. Se marca como visitado B. Se analizan sus vértices adyacentes, en este caso solo hay uno, F. Se lo marca como visitado, y se revisan entre sus nodos adyacentes, con cual seguir. En este caso, solamente Z está adyacente a F, por lo cual, como aún no ha sido visitado, se inicia un recorrido DFS a partir de Z.
3. Se marca como visitado Z. Al no haber vértices adyacentes, se termina el recorrido DFS de este subgrafo.
4. Se retorna al recorrido DFS a partir de F. Como todos los vértices adyacentes a este vértice han sido visitados, se termina el DFS iniciado a partir de F.
5. Se retorna al recorrido DFS a partir de B. Como este vértice no tiene más vértices adyacentes que queden sin visitar, se continúa con el recorrido DFS a partir de A.

6. A elige a C como vértice de inicio para un nuevo recorrido DFS.
7. A partir de C, se inicia un recorrido DFS en G. Como no tiene vértices adyacentes, se marca como visitado G y se continúa desde C.
8. C elige hacer un recorrido DFS a partir de H.
9. Se marca H como visitado, y se vuelve al recorrido DFS a partir de C.
10. Como C no tiene más vértices adyacentes sin recorrer, se vuelve a recorrer A.
11. Desde A, se inicia un recorrido en DFS en los vértices adyacentes restantes, primero en D y luego en E. En ambos subgrafos, lo único que se hace es marcar como visitado el vértice, es decir D o E según sea el caso, dado que ni D o E tienen vértices adyacentes..
12. Se retorna al recorrido DFS desde A. Como desde A no quedan vértices adyacentes sin recorrer, se termina el recorrido DFS, habiendo recorrido todos los vértices del grafo.

2.2.1.2 - Breadth First Search

El algoritmo **BFS**, o **Breadth First Search** (búsqueda en anchura), también nos permite recorrer todos los vértices de un grafo, pero el recorrido es radicalmente distinto al algoritmo DFS previamente explicado.

En el algoritmo BFS, el grafo es recorrido por capas. Es decir, primero se recorre el vértice de inicio, el cual podemos decir que conforma la primer capa y del cual diremos que tiene una distancia 0 (dado que es el origen). Una vez recorrido el vértice de inicio, se recorren los vértices adyacentes a el, es decir, los vértices con distancia 1, hasta el origen. Y así sucesivamente. Podemos ver una implementación de este algoritmo en pseudo-código en la Imagen 2-18

```
void unweighted( Vertex s )
{
        Queue<Vertex> q = new Queue<Vertex>( );
        for each Vertex v
                v.dist = INFINITY;
        s.dist = 0;
        q.enqueue( s );
        while( !q.isEmpty( ) )
        {
                Vertex v = q.dequeue( );
                for each Vertex w adjacent to v
                        if( w.dist == INFINITY )
                        {
                                w.dist = v.dist + 1;
                                w.path = v;
                                q.enqueue( w );
                        }
        }
}
```

Imagen 2-18: BFS en pseudo-código

Fuente: [11], pag. 372

Siguiendo el grafo de ejemplo de la Imagen 2-17, el resultado de aplicar el algoritmo de la Imagen 2-18 es el siguiente:

1. Se inicia el recorrido BFS en A. Este vértice pasa a tener una distancia 0 (*dist*) y es agregado a una cola.
2. Se saca de la cola un elemento, el cual es el vértice A, y se recorren los vértices adyacentes a el, es decir, los vértices B, C, D y E. Los cuatro son configurados con una distancia de valor 1, y se los agrega a la cola, en el mismo orden en que estos fueron mencionados.

3. Se saca un vértice de la cola, en este caso, es el vértice B. Se recorren sus vértices adyacentes, configurándolos con una distancia de 2 (la distancia de su predecesor más uno) y se los encola. Como resultado de este paso, se encola F y se continúa con el siguiente vértice.
4. Se desencola un vértice de la cola, en este paso, desencolamos a C. Este vértice se corresponde con la capa que está recorriendo el algoritmo, que es la capa con los vértices de distancia 1. Se recorren sus vértices adyacentes, G y H, se les configura una distancia 2 y se los encola a ambos en la cola, para continuar con otro vértice.
5. Se desencola un nuevo vértice de la cola, en este caso, el vértice D. Como este no tiene vértices adyacentes, se termina su procesamiento sin agregar ningún vértice a la cola.
6. Se desencola un nuevo vértice de la cola, en este caso, es el vértice E. Como no tiene vértices adyacentes, no se agrega ningún vértice nuevo a la cola. Al terminar de procesar este vértice, terminamos con los vértices de la capa con distancia 1 en el grafo.
7. Se desencola un nuevo vértice, este es el vértice F. Este vértice es el primero de la capa de vértices con distancia 2, por lo que, a partir de su procesamiento, se procesarán todos los vértices de esta capa. Es el primer vértice procesado en su nivel, debido a que es el primer vértice de este nivel que fue agregado a la cola. Para procesar F, se recorren sus vértices adyacentes y se los encola, en el caso de F, se encola el vértice Z, antes habiendo configurado su distancia como 3.
8. Se desencola un nuevo vértice, G. Como no tiene vértices adyacentes, no se encola ningún vértice.
9. Se desencola H, y su procesamiento es similar a G.
10. Se desencola el primer y único vértice con distancia 3, Z. Como no tiene vértices adyacentes, no se encola ningún vértice.
11. El algoritmo termina, dado que la cola de vértices no dispone de ningún vértice restante para ser procesado

2.2.2 - Alternativas tecnológicas para procesamiento de grafos en entornos Cloud

Como existen distintas tecnologías para el procesamiento de grafos, se debe elegir una entre todas ellas. Por lo tanto, a continuación se exponen distintas tecnologías que nos permiten procesar grafos existentes en la industria, con sus ventajas y desventajas.

- Pregel [3] : Es un sistema de procesamiento de grafos a gran escala desarrollado por Google. Se descartó por qué, además de ser un un software propietario de Google, no está a la venta en el caso de que una persona u empresa quisiera usarlo.

- GraphLab [4] : Es un framework de procesamiento distribuido de grafos de alto desempeño. Actualmente cuenta con una API desarrollada en python para que el desarrollador de software que la utilice interaccione con el framework. Al no tener una API para Java, este factor es el que permite descartar a esta opción como válida 31.

- Hive [38]: Permite procesamiento a través de una arquitectura Map/Reduce, aunque no fue inicialmente desarrollado con el fin específico de procesar grafos. Este es uno de los factores que permite establecer que no es la alternativa más eficiente [1].

- Giraph: Diseñado como una alternativa a Pregel, pero Open-Source, en 2011.
  Es la alternativa más eficiente disponible, comparando con Hive y GraphLab, de acuerdo a estudios realizados por el equipo de ingeniería de Facebook, con grafos de 25 millones de aristas [1].

Por ser la más eficiente posible entre las alternativas a las que se dispone de acceso entre todas las disponibles, se eligió **Giraph**. A continuación se explica en que consiste Hadoop , debido a que sobre esta plataforma basa su funcionamiento Giraph.

38 Sitio oficial de Hive, accedido el 11/11/2015 <https://hive.apache.org/>

2.3 - Hadoop

Hadoop es una plataforma de software que nos permite el procesamiento distribuido de grandes cantidades de datos a través de un cluster de computadoras usando modelos de programación simples. Está diseñado para ser escalable y tolerante a fallos [7], tanto en un nivel de software como en el de hardware.

Es Open Source, y su desarrollo está a cargo de la fundación Apache[39]. Fue diseñado por Doug Cutting y Mike Cafarella mientras trabajaban para Yahoo! en 2005[40], con el objetivo del procesamiento y almacenamiento de acumulaciones masivas de datos o Big Data. Actualmente es usado por grandes empresas como Facebook, Twitter, eBay, entre otras.

Un modelo de programación que permite procesar grandes cantidades de datos y que puede ser usado en la plataforma de software Hadoop, es **Map Reduce**, el cual es explicado a continuación.

2.4 - Map Reduce

Map Reduce empezó siendo un modelo de programación de tareas en procesamiento paralelo, presentado como un paper en 2008 por Jeffrey Dean y Sanjay Ghemawat [9], dos empleados en Google.

Su funcionamiento se basa en la reducción de los datos en partes más pequeñas, para su mejor y más eficiente procesamiento. El mismo consiste en funciones Map y Reduce.

La función Map recibe cada registro de los datos de entrada, los cuales pueden ser líneas de un archivo o filas de una base de datos por nombrar algunos ejemplos, como pares clave-valor o key-value y produce una salida del mismo tipo. Debido a su diseño, cada función Map es invocada independientemente del resto, permitiendo al framework usar la técnica divide-y-conquistarás[41] con el fin de paralelizar el cómputo total.

39 Apache Software Foundation, pagina oficial, 26/11/2016: <https://www.apache.org/>
40 Historia de Hadoop, Wikipedia, 27/12/2014
<http://en.wikipedia.org/wiki/Apache_Hadoop#History>
41 Algoritmo Divide y Vencerás, Wikipedia, 6/12/2016
<https://es.wikipedia.org/wiki/Algoritmo_divide_y_vencerás>

Se creará una tarea Map para cada bloque HDFS[42] de datos de entrada o input data, y dentro de cada tarea, se invoca múltiples veces a una función Map. El número de veces que se invoca a una función Map dentro de cada tarea es igual al número de registros correspondiente al bloque HDFS asignado. Esta etapa se denomina *Map Phase*.

Al finalizar las funciones Map, se agrupan los registros clave-valor resultantes, output, a través de las claves y los distribuye a las tareas Reduce. Esta fase, la de distribución y transmisión de datos a las tareas Reduce se denomina *Shuffle Phase*.

Los datos de entrada o *input data* de las tareas Reduce, los cuales se debe recordar que provienen de la salida de las tareas Map, también serán ordenados y agrupados por las claves o keys. Las funciones Reduce serán invocadas por cada una de dichas claves. Esta etapa se denomina *Reduce Phase.*

Al usar Map Reduce, implementando las funciones Map y Reduce es suficiente para que se realice la planificación y ejecución de los mismos en paralelo. Automáticamente se volverán a ejecutar las tareas que hayan fallado, así como mitigar los efectos de una computación desbalanceada.

La Imagen 2-19 nos permite comprender todo el proceso de Map Reduce, observando de forma gráfica cada una de las etapas previamente explicadas de este proceso.

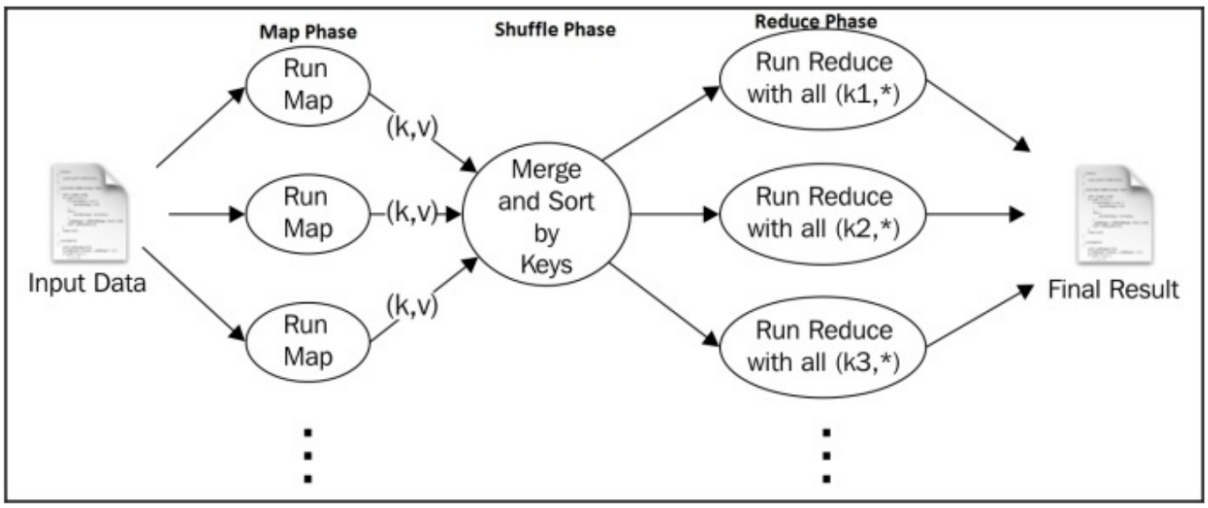

Imagen 2-19: Explicación gráfica de Map Reduce

Fuente: [10], página 65

42 HDFS quiere decir Hadoop Filesystem, y es el filesystem utilizado en Hadoop. Ver glosario para más información.

A continuación, se introducirá el algoritmo BlueFinder y se explicará por qué es necesario usar Map Reduce, para poder comprender correctamente hacia donde va este informe final.

2.5 - BlueFinder

El algoritmo BlueFinder [2] detecta y recomienda la mejor manera de representar en un sitio de la *Web Social*, información que es extraída de la Web Semántica. En particular, BlueFinder recomienda la manera de representar una propiedad semántica de DBpedia en Wikipedia, respetando las convenciones de la comunidad de usuarios de esta última.

Consta de dos etapas fundamentales, el **indexado** y la **recomendación**:

En la etapa de **indexado**, se recibe como entrada un conjunto de pares de artículos, y a partir de dicho conjunto y una copia local de Wikipedia (dump), se procesa el grafo de Wikipedia y se obtienen los distintos caminos que existen para cada uno de dichos pares. Por ejemplo, si queremos saber los caminos existentes entre los artículos del par *Rosario - Lionel Messi*, el algoritmo se posiciona en el artículo de *Rosario* y va navegando por los enlaces como si fuera un usuario, hasta conseguir todos los caminos existentes hasta llegar al artículo *Lionel Messi*. Al finalizarse esta etapa, se generará como salida un índice, en donde para cada par de artículos, se almacenarán los distintos caminos encontrados de la forma previamente descrita en el grafo de Wikipedia.

En la etapa de **recomendación**, se estima, para un par de artículos desconectados entre sí, la mejor forma de unirlos a través de un enlace, tratando de que la utilidad de este enlace sea la máxima posible. Implementa un algoritmo KNN (K nearest neighbors [16]) usando la información generada en la etapa anterior, para lograr cumplir su cometido.

BlueFinder nos permite establecer nuevas relaciones entre artículos de Wikipedia, basándose en los caminos navegacionales encontrados entre cada par de artículos, pero para esto es necesario hacer un indexado de todos los caminos navegacionales entre cada par de artículos, con el fin de encontrar el camino navegacional de mayor utilidad.

Aunque cumple la función con la que fue diseñado, la principal falencia del algoritmo BlueFinder es que la etapa de indexado es lenta, dado que el grafo de datos que conforma Wikipedia es increíblemente denso. Por ejemplo, la versión en español de Wikipedia tiene 1.295.015 artículos y dispone de 4.402.855 usuarios usuarios registrados[43]. Se agregan por

43 Estadísticas, Wikipedia, 05/11/2016 <https://es.wikipedia.org/wiki/Wikipedia:Estadísticas>

día más de 400 artículos y está entre los 10 sitios más visitados del mundo[44]. Debido a la magnitud de datos descripta, aún ejecutando el algoritmo BlueFinder en una máquina de altas prestaciones, suele tardar mucho, incluso semanas. Y lo que se quiere obtener no son solamente los caminos navegacionales entre un par de artículos sino hacerlo para todos los artículos de Wikipedia. **Este es el origen de esta tesina**, mejorar la etapa de indexado del algoritmo BlueFinder para poder realizar la misma función pero en menos tiempo y de una manera más eficiente.

Por esta razón, se decidió usar una tecnología especializada en el procesamiento de grafos como **Giraph**, dado que Wikipedia puede ser vista como un grafo, para poder llevar este objetivo a la realidad.

2.6 - Conclusiones del capítulo

En este capítulo se introdujo al lector en gran parte de los contenidos básicos que debe saber para comprender el resto de la tesina. Se pudo explicar en gran detalle la composición de Wikipedia, y por que puede considerarse como un grafo a esta misma. Debido a que es un grafo, se explicaron brevemente formas de recorrerlo, dejando la definición de cual de los dos usar para posteriores capítulos. Se introdujo al lector en distintas tecnologías que nos permiten realizar procesamientos de grafos, y se justificó por que, entre todas ellas, se eligió Giraph. Se introdujo al lector en Map Reduce, componente fundamental de la solución algorítmica definitiva de este informe final.

En el final de este capítulo, se introdujo brevemente en que consiste el algoritmo BlueFinder y cual es el motivo de la realización de esta tesina.

44 Lista de websites más populares, Wikipedia, 05/11/2016 <http://en.wikipedia.org/wiki/List_of_most_popular_websites>

Capítulo 3 - BlueFinder y el origen de esta tesina

Capítulo 3 - BlueFinder y el origen de esta tesina.....39
3.1 - DBPedia.....39
3.1.1 - Qué es DBPedia.....39
3.1.2 - Comparación de diferencias entre Wikipedia y DBPedia.....40
3.2 - Descripción del problema a resolver.....41
3.3 - Elección del tipo de recorrido de grafo a usar – BFS o DFS.....42
3.4 - Giraph.....43
3.4.1 - Explicación de su funcionamiento.....43
3.4.2 - El método Compute.....44
3.4.3 - Sincronización entre superpasos y su relación con el final del procesamiento en Giraph.....46
3.4.4 - Modelo computacional de Giraph y su relación con el modelo de BSP.....46
3.5 - Conclusiones del capítulo.....47

El **Capítulo 3** se inicia con una explicación acerca de DBPedia[45], que función tiene este sitio, el porque de su importancia y su relación con Wikipedia. A continuación y en función de lo explicado en la sección de DBPedia, se explica claramente cual es el problema a resolver, es decir, la razón u origen de esta tesina. Al haber explicado el problema a resolver, queda se debe elegir que tipo de recorrido de grafos se realizará, por ende se explica cual recorrido se usará para realizar el algoritmo, es decir, BFS o DFS. Por último, se introduce en profundidad el framework Giraph[46], explicándolo en profundidad a través de ejemplos y casos reales de uso en los que fue exitoso.

3.1 - DBPedia

3.1.1 - Qué es DBPedia

45 Sitio oficial de DBpedia, 11/11/2015 <http://wiki.dbpedia.org/>
46 Sitio oficial de Giraph, 12/12/2016 <http://giraph.apache.org/>

DBpedia es un proyecto realizado por la Universidad de Leipzig, la Universidad Libre de Berlín y la compañía OpenLink Software, y consiste en la construcción de una base a gran escala, consistente en extraer información estructurada en 111 distintas, de distintos lenguajes de Wikipedia. Esta información queda disponible para que pueda ser accedida desde la web, en forma de una base de datos semántica[47]. El acceso a esta información se puede realizar a través de un endpoint SPARQL[48] , ya sea en forma manual es decir, accediendo al endpoint y haciendo una consulta específica sobre lo que se quiera averiguar, o en forma programática a través de una API. Fue introducida a la comunidad científica a través de [15].

DBPedia es en la actualidad uno de los nodos centrales (debido a la gran cantidad de proveedores de datos que han empezado a conectar su información hacia este sitio) en el entramado actual de la emergente ***Web de Datos*** [49]. La Web de Datos surgió de la iniciativa de **Datos Abiertos** [50] o Open Data, en Inglés, la cual pregona que “determinados tipos de datos estén disponibles de forma libre para todo el mundo, sin restricciones de derechos de autor, de patentes o de otros mecanismos de control” .

3.1.2 - Comparación de diferencias entre Wikipedia y DBPedia

Si se analiza *DBPedia*, se podrán encontrar múltiples relaciones entre artículos, que en la Wikipedia original ya no existen como caminos navegacionales entre ellas.

Por ejemplo, es posible hacer una consulta en DBPedia y consultar todos los nacidos en Boston. Esta consulta produce una serie de pares de artículos (Boston, nombre de persona) que son relacionados por la propiedad semántica de DBPedia *is birthplace of*. Sorprendentemente, la cantidad de personas que están incluida en el resultado de esta consulta, es mayor que la cantidad obtenida navegando desde el artículo Boston en Wikipedia.

La pregunta, frente a esto, es la siguiente ¿Es necesario agregar esta información a Wikipedia? En algunos casos, esta información está escondida intencionalmente, para simplicidad de la misma página, debido a un exceso de enlaces, por ejemplo. Pero en otros casos, puede llegar a ser útil.

47 DBpedia, artículo de Wikipedia, 5/11/2016 <http://es.wikipedia.org/wiki/DBpedia>
48 DBPedia SPARQL Endpoint, 5/11/2016 <http://es.dbpedia.org/sparql>
49 Sitio oficial de DBpedia, sección “Acerca de”, 11/11/2015: <http://dbpedia.org/about>
50 Datos Abiertos, Wikipedia, 11/11/2015 <https://es.wikipedia.org/wiki/Datos_abiertos>

Agregar estos enlaces perdidos a Wikipedia no es fácil, dado que este sitio tiene múltiples convenciones para agregar contenido[51] y no siempre es tan evidente de que manera debería hacerse.

De acuerdo a las convenciones más arriba explicadas, para describir relaciones de uno a muchos, se debe usar categorías.

A simple vista, parece correcto de acuerdo a las convenciones de Wikipedia, representar la relación “is birthplace of” a través de una categoría. En el caso del artículo “Donna Summer”[52], por ejemplo, se representa mediante la categoría *Bostonianos*[53]. Por ende, sería deseable que dependiendo de qué enlaces faltantes existan, tengamos una forma de sugerir automáticamente la categoría que corresponda para corregir esa pérdida de información. Aquí surge el problema, que es como sugerir dicha categoría. Dicho problema se detalla a continuación.

3.2 - Descripción del problema a resolver

El problema que se propone resolver esta tesis, es la generación de caminos navegacionales para cada uno de los pares de artículos que se realiza en la etapa de indexado de BlueFinder, dado que esta no es eficiente. Este trabajo propone utilizar un framework de procesamiento de grafos como Giraph, el cual basa su funcionamiento en una arquitectura Map Reduce, para una generación de caminos navegacionales eficiente, que reemplace la etapa de indexado de BlueFinder, y usando los resultados de este trabajo directamente en la siguiente etapa de BlueFinder, que como se mencionó anteriormente, es la etapa de recomendación.

Por este motivo, se necesita desarrollar un algoritmo que consiga todos los caminos navegacionales entre un par de artículos dados de Wikipedia. Al ejecutar este algoritmo para distintos pares de artículos, se podrá construir un índice que indique, para cada par, que caminos navegacionales le corresponden. Con este índice, se podrá identificar los distintos caminos que existen entre dos artículos y a partir de este punto, el problema de recomendar que enlace será el correcto para completar la información faltante en Wikipedia recaerá en la etapa de **recomendación** de BlueFinder, previamente explicada al final del Capítulo 2. Aunque el desarrollo de este algoritmo no es trivial, dado que la magnitud de

51 Manual de estilo, Wikipedia, 5/11/2016 <https://es.wikipedia.org/wiki/Wikipedia:Manual_de_estilo>
52 Donna Summer, Wikipedia, 5/11/2016 <https://es.wikipedia.org/wiki/Donna_Summer>
53 Bostonianos, Wikipedia, 5/11/2016 <https://es.wikipedia.org/wiki/Categoría:Bostonianos>

datos de Wikipedia es grande, y debe encontrar en forma eficiente sus resultados, de otra forma no estaríamos obteniendo mejores resultados.

La búsqueda de caminos a realizar con el algoritmo implica la necesidad de recorrer el grafo de Wikipedia, yendo por todos los caminos posibles de un artículo A a otro artículo B. Esto significa que se debe desarrollar un algoritmo que recorra un grafo por completo. Como se explicó en el Capítulo 2, los recorridos de grafos para cumplir este objetivo pueden ser el BFS o el DFS. Se debe analizar cuál recorrido usar, antes de comenzar con el diseño del algoritmo que implemente dicho recorrido. Dicho análisis se realiza a continuación.

3.3 - Elección del tipo de recorrido de grafo a usar – BFS o DFS

Para la definición del tipo de recorrido que se realizará sobre el grafo de Wikipedia, debemos tener en cuenta distintos factores.

Si se evalúa esta decisión **en función de la profundidad de la longitud de los caminos** a generar, se debe tener en cuenta que los caminos en general tendrán una longitud promedio de 5 vértices[54], por lo que claramente la búsqueda en profundidad no será de particular uso, dado que necesitamos hacer muchas búsquedas en distancias relativamente cortas.

Si se decide **en función de los recursos de hardware a los que se tiene acceso**, se debe notar que al tener acceso a una cantidad considerable de recursos por disponer de acceso a un entorno de Cloud Computing, este motivo nos acerca más al uso de un BFS, dado que al poder disponer del hardware necesario, podemos optimizar la generación de caminos haciendo múltiples búsquedas de caminos en paralelo[55]. Aunque esto también incremente sustancialmente la cantidad de memoria requerida y la potencia de cómputo necesaria, esto no representa un obstáculo, ya que en este informe final se buscará en todo momento reducir el tiempo que debe tardar la solución algorítmica, siendo la disminución de los requerimientos de memoria y poder de cómputo un objetivo de índole secundario. La definición establecida en el Capítulo 2 acerca del uso de Map Reduce como arquitectura, también nos acerca inevitablemente al uso de un BFS que se ejecute en paralelo.

54 De acuerdo a [13] y [14], un camino navegacional de una longitud mayor que 5 es inalcanzable por un usuario común. Por esta razón, se considera que artículos unidos por un camino navegacional de distancias mayores a 5 no están conectados.

55 Se debe recordar del Capítulo 2, que Map Reduce es un modelo de computación enfocado al paralelismo, por lo que hablar en esta instancia de este enfoque computacional no debería ser extraño para el lector.

Por estas razones, **el algoritmo de recorrido del grafo de Wikipedia elegido como base para la solución algorítmica es el BFS o Breadth First Search**.

Habiendo definido que tipo de recorrido de grafos se realizará, se introduce a continuación el framework elegido para el procesamiento del grafo de Wikipedia, Giraph.

3.4 - Giraph

3.4.1 - Explicación de su funcionamiento

Giraph, o mejor llamado Apache Giraph, es un framework de procesamiento iterativo de grafos, construido encima de *Hadoop*. Es actualmente usado en Facebook [7] para analizar el grafo formado por sus usuarios y las conexiones entre estos. Originado como la contraparte Open-Source de Pregel [3], *a*mbos sistemas fueron inspirados por BSP [17], el modelo de computación distribuida introducido por Leslie Valiant.

En Giraph, cada vértice realiza el procesamiento en forma local y se comunica con otros vértices a través de mensajes enviados en forma asíncrona. También Giraph nos permite asignarles un peso a cada vértice o arista del grafo,

La Imagen 3-1 nos permitirá conceptualizar las distintas etapas en las cuales Giraph realiza para poder procesar grafos:

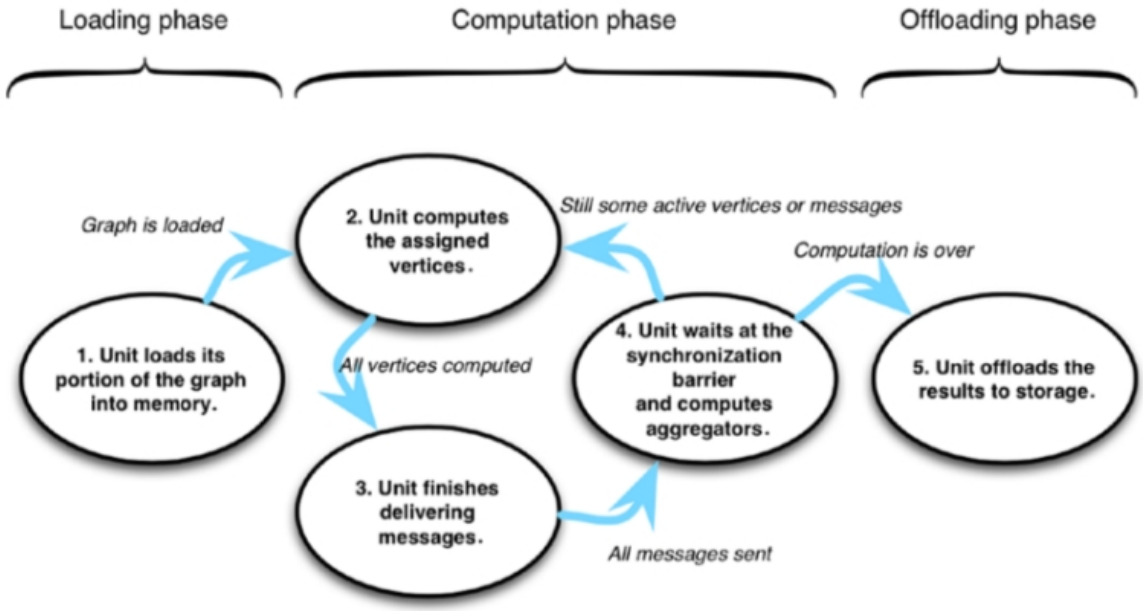

Imagen 3-1: Esquema de como se realiza el procesamiento en Giraph

Fuente:[12], figura 3-11, pagina 58

El procesamiento en Giraph debe ser pensado desde los vértices , dado que es, justamente, un modelo de procesamiento de grafos orientado a los **vértices**. Es decir, en todo momento se estará pensando con qué datos cuenta cada vértice, que procesamiento realizará cada uno de ellos, que mensajes enviará y/o recibirá, etc.

Giraph realiza el procesamiento de grafos de la siguiente manera:

La **primer etapa** es la etapa de carga de datos, se la puede ver en la Imagen 3-1 como *Loading Phase*, y consiste en el procesamiento de los datos de entrada. En esta fase se recibe un archivo, el cual contiene la información de un grafo compuesto de vértices y aristas dirigidas, en un formato apto para ser procesado por Giraph fin. A través de este archivo de entrada se configura la topología del grafo así como los valores asociados a cada uno de sus elementos, dado que Giraph nos permite asociarles un valor tanto a los vértices como a las aristas.

La **segunda etapa** es la etapa de procesamiento, se la puede ver en la Imagen 3-1 como *Computation Phase*, y es en donde se realiza el procesamiento de cada vértice del grafo. Esta etapa se realiza a través de una serie de iteraciones, llamadas superpasos o *supersteps* en BSP [17]. Inicialmente, todo vértice está activo[56]. En cada superpaso, cada vértice activo invoca al método **compute**. Esta etapa se repetirá hasta que no queden vértices activos en todo el grafo, lo que significa que se finalizó el procesamiento a realizar.

La **tercer etapa** es la del procesamiento de los datos de salida, se la puede ver en la Imagen 3-1 como *Offloading Phase* Dado que cada vértice debe saber como imprimir su información de salida, el conjunto de la información de salida de todos los vértices se convertirá en la salida resultante del algoritmo.

A continuación se explicará el método Compute previamente mencionado, el cual es fundamental para entender el funcionamiento de cualquier algoritmo desarrollado con Giraph.

3.4.2 - El método Compute

Este método deberá implementar el algoritmo que será ejecutado sobre el grafo de entrada. Al ejecutarse este método, el vértice recibe mensajes que se le hayan enviado en el superpaso previo y realiza el procesamiento usando dichos mensajes, el valor del vértice y los valores de sus aristas. Esto puede resultar en modificaciones en los valores del mismo

56 Solo los vértices activos realizan procesamiento. Al entrar en estado inactivo, no realizan actividad alguna.

vértice y/o de las aristas y/o de la topología del grafo, así como en el envío de nuevos mensajes para el superpaso posterior.

Este método no tiene un acceso directo a los valores de otros vértices, así como a las aristas de otros vértices. La comunicación entre vértices se realiza a través del *envío de mensajes únicamente*.

En un **algoritmo de búsqueda del camino mínimo**, por ejemplo, este método hará lo siguiente:

1. Encontrar el valor mínimo entre todos los valores recibidos a través de un mensaje
2. Si el valor es menor que el mínimo actual
    a. Se adopta el valor recibido como el nuevo mínimo
    b. Se envía la sumatoria del nuevo mínimo más el valor de la arista, por cada arista que tenga el vértice.

A continuación se dispone de un código simplificado de este algoritmo, en la Imagen 3-2:

```
public void compute(Iterable<DoubleWritable> messages) {
  double minDist = Double.MAX_VALUE;
  for (DoubleWritable message : messages) {
    minDist = Math.min(minDist, message.get());
  }
  if (minDist < getValue().get()) {
    setValue(new DoubleWritable(minDist));
    for (Edge<LongWritable, FloatWritable> edge : getEdges()) {
      double distance = minDist + edge.getValue().get();
      sendMessage(edge.getTargetVertexId(), new DoubleWritable(distance));
    }
  }
  voteToHalt();
}
```

Imagen 3-2: Método Compute, búsqueda del camino mínimo

Fuente: http://giraph.apache.org/intro.html , Fig. 2

Este método primero determina la distancia mínima al arribar los mensajes , y en caso de que este valor consista en un nuevo mínimo, lo envía a través de cada una de sus aristas a través del método *sendMessage*.

3.4.3 - Sincronización entre superpasos y su relación con el final del procesamiento en Giraph

Dado que en cada superpaso se envían mensajes desde un vértice a otros, es necesario tener una coordinación que nos asegure que los mensajes de un superpaso llegarán exactamente en el siguiente, tal como dice el estándar de Pregel [3], en el cual está basado Giraph. Esta sincronización se logra a través de barreras de sincronización (synchronization barrier, concepto proveniente de BSP [17]) entre superpasos consecutivos. Dichas barreras nos aseguran dos cosas:

1. Los mensajes enviados en cualquier superpaso son enviados a sus vértices de destino recién en el superpaso siguiente
2. Los vértices inician procesamiento de un superpaso, una vez que todos los vértices han finalizado el procesamiento del superpaso anterior.

Los valores de vértices y aristas son retenidos a través de las barreras de sincronización. Esto quiere decir que el valor de cualquier vértice o arista al iniciarse un nuevo superpaso, es idéntico al valor que tenía dicho vértice o arista al finalizar el superpaso anterior.

Cualquier vértice puede detener su procesamiento en cualquier superpaso. El vértice simplemente declara que no quiere estar activo más tiempo, a través del método *voteToHalt()* . Sin embargo, todo mensaje entrante activará el vértice de nuevo.

El procesamiento en Giraph finaliza cuando todos los vértices han sido detenidos y cuando no hay más mensajes en tránsito.

3.4.4 - Modelo computacional de Giraph y su relación con el modelo de BSP

Tanto en Giraph como en BSP[57], se dispone de N unidades de procesamiento (las cuales en Giraph se denominan Workers), los cuales pueden comunicarse a través de una red o bus. Se divide la entrada a través de todas las unidades de procesamiento, y cada unidad esta encargada de realizar el procesamiento correspondiente en forma local.

57 Esta sección está basada en “Understanding the Bulk Synchronous Parallel Model”, página 67, [12]

Cuando este procesamiento ha finalizado, estas unidades intercambian los resultados entre si y esperan a recibir los resultados del resto de las unidades, lo que previamente se ha denominado barrera de sincronización. Una vez que el envío de los resultados ha terminado, se inicia un nuevo superpaso.

El equilibrio en el procesamiento de cada Worker, así como la cantidad de comunicación que cada uno de ellos realiza son claves para optimizar un algoritmo en Giraph, por lo que cada Worker debería utilizar los recursos de hardware que disponga de la forma más eficiente posible, de modo de no demorar al resto en cada superpaso ejecutado.

3.5 - Conclusiones del capítulo

Se explicó el funcionamiento de Giraph en profundidad en forma abstracta así como con un ejemplo de un algoritmo concreto, la búsqueda de caminos mínimos en un grafo.

Se explicaron conceptos básicos de DBPedia, con el fin de poder entender la relación de estos conceptos con el trabajo realizado por BlueFinder. A partir de estos conceptos, se pudo explicar cual era exactamente el problema que deberá resolver este trabajo final.

Por último, se hace una amplia explicación de que es Giraph y como funciona, para que a partir de este punto, el lector esté debidamente capacitado en la tecnología con el fin de poder después usarla. Se lo explica a través de ejemplos para que sea muy fácil la explicación.

Capítulo 4 - Solución Algorítmica

Índice de contenidos

Capítulo 4 - Solución Algorítmica....49
- 4.1 - Construcción del grafo de entrada....50
  - 4.1.1 - Graphipedia....52
    - 4.1.1.1 - Cambios introducidos en el proyecto Graphipedia....53
  - 4.1.2 - Algoritmo para la construcción del grafo de entrada....54
    - 4.1.2 .1- Filtrado mediante expresiones regulares....57
- 4.2 - Algoritmo para encontrar caminos entre dos vértices del grafo de Wikipedia....57
  - 4.2.1 - Versión con un vértice de origen y un vértice de destino....58
    - 4.2.1.1 – Funcionamiento del algoritmo....61
    - 4.2.1.2 - Problemas encontrados en esta versión....64
  - 4.2.2 - Versión con un vértice de origen y múltiples destinos....64
    - 4.2.2.1 – Funcionamiento del algoritmo....71
    - 4.2.2.2 - Problemas encontrados en esta versión....72
  - 4.2.3 - Explicación gráfica del funcionamiento del algoritmo....73
    - 4.2.3.1 - Explicación de búsqueda de caminos en un grafo sin ciclos....73
    - 4.2.3.2 - Explicación de búsqueda de caminos en un grafo con ciclos....77
- 4.3 - Construcción del archivo de salida....82
  - 4.3.1 - Versión con un vértice de origen y uno de destino....82
  - 4.3.2 - Versión con un vértice de origen y múltiples destinos....84
- 4.4 – Enlace con BlueFinder - Construcción de base MySQL a partir del resultado del algoritmo de búsqueda de caminos navegacionales....85
- 4.5 - Conclusiones del capítulo....87

En este capítulo se detallara todo lo concerniente al algoritmo en si. Se comenzará explicando como es el proceso para poder convertir los datos de Wikipedia en un archivo apto para ser procesado por Giraph, se continuará con la explicación de cómo funciona el algoritmo y por último se detallara en que forma el algoritmo produce sus resultados.

4.1 - Construcción del grafo de entrada

El lector debe recordar que, para que Giraph procese los datos del archivo de entrada, los cuales representan el grafo de Wikipedia, este archivo debe tener un formato apto para tal fin. Por eso, esta sección detallara la forma en que se se construirá este archivo respetando este **formato de entrada**. Para construir este archivo de entrada, se necesita disponer de la información presente en Wikipedia (es decir, artículos con sus respectivos enlaces y categorías). Dicha información se obtendrá de los dumps disponibles en el repositorio de dumps de Wikimedia[58]. De este repositorio es deseable usar el dump más reciente que haya disponible, asegurándonos de bajar el archivo que finaliza con el sufijo and*s-articles-multistream.xml.bz2*[59].

Una vez obtenido el dump de Wikipedia correspondiente, se puede estudiar que estructura tiene. Un extracto de esta estructura se muestra en la siguiente Imagen 4-1

58 Repositorio de dumps de Wikimedia, 26/12/2016, <https://dumps.wikimedia.org/eswiki/>
59 Este dump contiene las referencias de cada artículo a sus categorías, así como las categorías en si.

```
...
<page>
    <title>Andorra</title>
    <ns>0</ns>
    <id>7</id>
    <revision>
      ...
;Política
* [http://www.govern.ad/ Gobierno de Andorra] (en catalán)
* [http://www.consellgeneral.ad Consell General] (en catalán)
* [http://www.coprince-fr.ad/ Web del Copríncipe francés] (en catalán y francés)

[[Categoría:Andorra| ]]
[[Categoría:Instituciones patrocinadoras de equipos ciclistas]]</text>
      ...
    </revision>
  </page>
  <page>
    <title>Argentina</title>
    <ns>0</ns>
    <id>10</id>
    <revision>
  ...
...
</mediawiki>
```

Imagen 4-1: Extracto del dump de Wikipedia en español del 01-06-2016

Fuente: Extracto de un dump de Wikipedia

Se puede observar que un artículo es representado mediante una etiqueta xml *page*, dentro de la cual están la etiqueta <*title*>, correspondiente al nombre del artículo en Wikipedia. Cada artículo también posee el contenido de Wikipedia en la etiqueta *text*, en la cual se puede encontrar las categorías y enlaces que cada artículo posee, usando el mismo formato de Wikitexto que se detalló en el capítulo 2. Habiendo definido a partir de que información se debe construir el archivo de entrada de Giraph, se analizó las herramientas existentes de conversión de archivos XML, y se encontró un proyecto, denominado **Graphipedia**, que podía procesar un dump de Wikipedia para convertirlo en un archivo con un formato distinto. Dicho proyecto se explica a continuación.

4.1.1 - Graphipedia

El proyecto **Graphipedia**[60] fue desarrollado por Mirko Nasato[61] con el fin de generar un archivo XML a partir de un dump de Wikipedia. Este XML generado contendrá todos los artículos que existan en el dump de Wikipedia y los Enlaces que cada uno de estos artículos contengan. Los pasos a realizar son los siguientes (fueron probados en una computadora con Ubuntu 14.04):

- Descargamos el proyecto Graphipedia desde GitHub
    - wget https://github.com/mirkonasato/graphipedia/archive/master.zip
- Descomprimimos el zip a través del comando **unzip**
    - unzip master.zip
- Accedemos al directorio donde está la herramienta
    - cd graphipedia-master
- Construimos a través de **maven**
    - mvn package
- Ejecutamos a través de la línea de comandos
    - java -classpath graphipedia-dataimport.jar org.graphipedia.dataimport.ExtractLinks wikipedia-dump.xml salida-resultante.xml

Esta ejecución[62] dará como resultado un archivo xml con un formato similar al indicado en la Imagen 4-2:

60 Proyecto Graphipedia en GitHub, 29/12/2016 <https://github.com/mirkonasato/graphipedia>
61 Perfil *LinkedIn* de Mirko Nasato, 29/12/2016 <https://www.linkedin.com/in/mirkonasato>
62 Algoritmo ExtractLinks, Mirko Nasato, 29/12/2016 <https://github.com/mirkonasato/graphipedia/blob/master/graphipedia-dataimport/src/main/java/org/graphipedia/dataimport/ExtractLinks.java>

```
<?xml version='1.0' encoding='UTF-8'?>
<d>
<p>
        <t>tituloArticulo1</t>
        <l>Enlace1Articulo1</l>
        <l>Enlace2Articulo1</l>
</p>
<p>
        <t>tituloArticulo2</t>
        <l>...</l>
        <l>...</l>
</p>
</d>
```

Imagen 4-2: Formato del archivo XML resultante de ExtractLinks

Fuente: elaboración propia

Se puede observar que, como no se está mostrando ninguna Categoría de ningún Artículo, dado que esto no era necesario en el algoritmo desarrollado por Mirko Nasato, no es exactamente lo que se necesita en este informe. Como se necesita también las categorías de cada artículo, además de las categorías listadas como artículos, se debió extender esta funcionalidad. Por esta razón, se renombró la clase ExtractLinks, y se modificó su funcionamiento para que también extraiga las categorías de cada artículo. A continuación, se explica este cambio.

4.1.1.1 - Cambios introducidos en el proyecto Graphipedia

Se renombró la clase ExtractLinks, para que su nombre contenga el nuevo funcionamiento a agregar, bajo el nombre **ExtractLinksAndCategories**.

Esta clase también muestra las categorías en el archivo XML resultante tal cual ilustra la Imagen 4-3:

```
<d>
        <p>
            <t>tituloArticulo1</t>
            <l>Enlace1Artículo1</l>
            <l>Enlace2Artículo1</l>
            <c>Categoría:CategoríaArtículo1</>
        </p>
        <p>
            <t>tituloArticulo2</t>
            <l>Enlace1Artículo2</l>
            <l>Enlace2Artículo2</l>
            <c>Categoría:CategoríaArtículo2</>
        </p>
</d>
```

Imagen 4-3: Formato del archivo XML resultante de ExtractLinks
Fuente: elaboración propia

Aunque se estaba más cerca de lo necesario, no fue suficiente, dado que esta información no tiene un formato apto para ser cargado en Giraph. Se decidió hacer algo más específico aún. Se creo una nueva clase, que extraerá la información que ya extrae ExtractLinksAndCategories, pero en esta versión se colocará la información en un formato apto para Giraph definido específicamente para este trabajo, dado que ninguno de los formatos de entrada pre-existentes en Giraph era similar a lo que se necesitaba en este trabajo. Dicha clase es explicada a continuación.

4.1.2 - Algoritmo para la construcción del grafo de entrada

Se construyó una clase Java nueva , la cual se denominó **ExtractGraph**[63]. Esta clase tiene un funcionamiento similar a **ExtractLinksAndCategories**, solo que el formato de la información producida es apta para un formato Giraph especialmente definido para

63 ExtractGraph, referencia a la clase en el repositorio BitBucket, 29/30/2016, <https://bitbucket.org/ambartsumian/lectura_dump_wikiquote/src/3b1dcddd174206db0931e973f6cb127dfbf57f6b/lectura_dump_wikiquote/generacion_grafo_wikiquote/src/main/java/tesina/ExtractGraph.java?at=master&fileviewer=file-view-default>

este trabajo[64]. Por cada uno de los Artículos y Categorías existentes en el dump, este algoritmo generará una línea correspondiente en el archivo resultante, y cada una de estas líneas representará un vértice al ser cargado en Giraph. Estas líneas contendrán el título del Artículo o Categoría, así como sus Enlaces y Categorías, los cuales se convertirán en aristas de los vértices previamente mencionados. Los Artículos, Enlaces y Categorías se identificarán a través de su atributo *title* del dump.

El algoritmo ExtractGraph es configurable a través de los siguientes parámetros:

- Archivo de entrada: dump XML descargado del repositorio de Wikipedia
- Nombre que contendrá el archivo de salida generado
- Archivo con expresiones regulares, parámetro opcional que nos permite indicar una serie de expresiones regulares, a través de las cuales le indicamos al programa que enlaces no son válidos y deben ser descartados.

El archivo de salida resultante tendrá un formato como el que indica la Imagen 4-4 (el lector debe notar que **/t** implica un carácter TAB). En la Imagen 4-5 podemos ver un extracto de un archivo generado con este formato.

```
Vertice1/tpesoVert1/tenl1/tpesoEnl1/tenlace2/tpesoEnl2/tcategoria1/tpesoCat1
Vertice2/tpesoVert2/tenl1/tpesoEnl1
```

Imagen 4-4: Formato del archivo XML resultante de ExtractGraph

Fuente: elaboración propia

Como generalmente en Giraph se le da un peso a cada vértice y arista, se definió, en función de esto, que el peso de cada vértice será **0.0**, el de los enlaces **1.0** y el de las categorías **2.0**, lo cual puede ser verificado en el extracto de la Imagen 4-5.

64 Formato de entrada diseñado a medida de este trabajo, 28/12/2016, <https://bitbucket.org/ambartsumian/tesina/src/d30e02f07cfeac0e2d0f687e2c340617a5fb0b65/generacion_entrada_giraph/procesamiento_grafo_wikiqoutes/src/main/java/ar/edu/info/unlp/tesina/vertice/estructuras/IdTextWithComplexValueInputFormat.java?at=master&fileviewer=file-view-default>

Demografía de Andorra 0.0 Portugal 1.0 San Julián de Loria 1.0 Años 1950 1.0 Años 60 1.0 Lengua materna 1.0 Ordino 1.0 Apátrida 1.0 Idioma catalán 1.0 Paraíso fiscal 1.0 España 1.0 Idioma español 1.0 Sindicato 1.0 Universidad de Andorra 1.0 Encamp 1.0 Derecho al voto 1.0 Idioma francés 1.0 Derechos constitucionale s 1.0 La Massana 1.0 Católica 1.0 Andorra 1.0 Idioma portugués 1.0 Escaldes-Engordany 1.0 Francia 1.0 Canillo 1.0 The World Factbook 1.0 Estac ión de esquí 1.0 Andorra la Vieja 1.0 Categoría:Demografía de Andorra| 2.0

Comunicaciones de Andorra 0.0 Radio Tele-Taxi 1.0 Ràdio Principat 1.0 Flaix FM 1.0 El Periòdic d'Andorra 1.0 La Poste (Francia) 1.0 CIA World Factbook 1.0 Máxim a FM 1.0 Sociedad Estatal Correos y Telégrafos (España) 1.0 Cadena Dial 1.0 Los 40 Principales 1.0 M80 Radio 1.0 Andorra 1.0 Diari d'Andorra 1.0 Radio Nacional de RNE 1.0 Correo 1.0 Ràdio 4 1.0 Francia 1.0 Radio 3 1.0 France Inter 1.0 Onda Rambla 1.0 Códigos de países 1.0 Catalunya Ràdio 1.0 España 1.0 Cadena SER 1.0 Ràdio Nacional d'Andorra 1.0 Categoría:Medios de comunicación de Andorra 2.0

Imagen 4-5: Extracto del archivo de entrada generado

Fuente: elaboración propia

En el ejemplo de la Imagen 4-5, se puede ver que hay dos vértices listados: *Demografía de Andorra* y *Comunicaciones de Andorra, los cuales tienen un peso de 0.0*. Ambos tienen múltiples Enlaces, cada uno con su título correspondiente, y el valor 1.0, como por ejemplo el vértice *Demografía de Andorra* que tiene un enlace a *Portugal*.

Además se puede notar que cada uno de los vértices de la Imagen 4-5 tiene una Categoría, la cual tiene un peso de 2.0, además de tener el prefijo *Categoría*. En el extracto de ejemplo de la Imagen 4-5, el vértice *Comunicaciones de Andorra* tiene como categoría a *Categoría:Medios de comunicación de Andorra.*

Para ejecutar este pequeño programa, debemos usar la siguiente línea

- java -classpath graphipedia-dataimport.jar org.graphipedia.dataimport.ExtractGraph wikipedia-multistream.xml grafo-giraph.txt

Como detalle adicional, queda por mencionar que este programa generará un archivo con todos los vértices que formarán parte del grafo, en un archivo **vertices.txt**, con un mero fin informativo.

4.1.2 .1- Filtrado mediante expresiones regulares

La clase ExtractGraph está preparada para recibir un parámetro opcional, el cual es un archivo con líneas de texto. Este archivo permite eliminar en forma masiva Enlaces, de forma que no aparezcan en el archivo generado.

Por ejemplo, en caso de que carguemos este archivo con una línea que tenga la expresión regular **^[0-9]{0,4}$**, esto hará que al algoritmo compare esta expresión con cualquier texto formada por 4 dígitos, y se descartarán todos los enlaces que coincidan con esta expresión, en los vértices que correspondan.

Habiendo explicado cómo es la generación del archivo de entrada con un formato apto para ser procesado por Giraph, se procederá en la siguiente sección, a explicar en qué consiste el algoritmo realizado.

4.2 - Algoritmo para encontrar caminos entre dos vértices del grafo de Wikipedia

Con el fin de encontrar los caminos navegacionales que existen entre dos vértices, se desarrolló un algoritmo que sea apto para ser ejecutado en Giraph. Este algoritmo estará basado en el formato de entrada previamente definido en la sección anterior.

El algoritmo consiste en un BFS, pero con modificaciones que no son tan usuales en algoritmos característicos de los recorridos de este tipo. Se usó como base el algoritmo *SimpleBFSStructureComputation*[65] desarrollado por Marco Lotz[66] y se creó el algoritmo BusquedaDeCaminosNavegacionalesWikiquote. Este algoritmo se encarga en determinar, entre dos vértices dados y una distancia dada, que caminos existen entre dos vértices distintos del grafo, sin tener en cuenta los ciclos. Para realizar esta función, se basa en el contenido de los mensajes que circulan entre los vértices, por esta razón muchas veces se

65 SimpleBFSStructureComputation, BFS en Giraph, desarrollado por Marco Lotz, 28/12/2016 <https://github.com/MarcoLotz/GiraphBFSSO/blob/master/src/uk/co/qmul/giraph/structurebfs/SimpleBFSStructureComputation.java>

66 Marco Lotz, perfil LinkedIn, 28/12/2016 <https://www.linkedin.com/in/marco-aurelio-lotz-3b7a4176>

tomara casi como sinónimos los conceptos de mensajes y caminos, dado que un mensaje se puede convertir en un camino válido, o no, dependiendo de lo que ocurra en la ejecución del algoritmo.

Dado que Giraph está escrito en este lenguaje[67], se prefirió usar el lenguaje **Java** para escribir el algoritmo.

Se desarrollaron dos soluciones algorítmicas distintas, estas son: una solución **entre un vértice de origen y un vértice de destino, y otra solución** entre un **vértices de origen y múltiples vértice de destino**. A continuación se explica como funcionan ambas versiones de este algoritmo.

4.2.1 - Versión con un vértice de origen y un vértice de destino

El pseudo-código de esta versión del algoritmo[68] es el siguiente:

```
1 public class BusquedaDeCaminosNavegacionalesWikiquote {
2       public String SOURCE_ID = "A";
3       public String DEST_ID = "B";
4       public Integer MAX_SUPERSTEPS = 5;

5       public boolean esInicio(Vertex vertex) {
6               return vertex.getId().toString().equals(SOURCE_ID.toString());
7       }

8       public boolean esDestino(Vertex vertex) {
9               return vertex.getId().toString()
10                              .equals(DEST_ID).toString());
11      }
```

67 Url del repositorio de código de Giraph, donde se puede ver que el código fuente esta escrito en el lenguaje JAVA, 28/12/2016 <https://github.com/apache/giraph>

68 BusquedaDeCaminosNavegacionalesWikiquote, un vértice de origen y un vértice de destino, referencia BitBucket, 28/12/2016, <https://bitbucket.org/ambartsumian/tesina/src/97703b704707029d3757f16f96c685e2f01793bb/generacion_entrada_giraph/procesamiento_grafo_wikiqoutes/src/main/java/ar/edu/info/unlp/tesina/lectura/grafo/BusquedaDeCa

```
12      private int maximaCantidadDeSuperpasosPosibles() {
13              return MAX_SUPERSTEPS;
14      }

15      public void BFSMessages( Vertex vertex) {
16              if (!esDestino(vertex)) {
17                      for (String mensajeDeUnVerticePredecesor : vertex
18                                      .getMensajesVerticesPredecesores()
19                                      .split(SEPARADOR_VALORES_EN_LINEA)) {
20                              sendMessageToAllEdges(vertex, new Text(
21                                              mensajeDeUnVerticePredecesor));
22                      }

                        // Ponemos en blanco los mensajes recibidos, para no mandar
un
23                      // mensaje dos veces
24                      vertex.setMensajesVerticesPredecesores("");
25              } else {
26                      //Vertice destino, no se envian mensajes
                }
        }

        @Override
27      public void compute( Vertex vertex, Iterable<Text> messages) {
28              // Se fuerza a que los vertices converjan en el superpaso máximo
29              if (getSuperstep() < maximaCantidadDeSuperpasosPosibles()) {
30                      if (getSuperstep() == 0) {
31                              if (esInicio(vertex)) {
32                                      vertex.setMensajesVerticesPredecesores(
33                                                      vertex.getId().toString());
```

minosNavegacionalesWikiquote.java?at=v1.0&fileviewer=file-view-default>

```
34                                  BFSMessages(vertex);
35                          }
36                  }
37                  else {
38                          // Procesamos los mensajes que llegan desde los
vertices
39                          // predecesores
40                          configurarMensajesDePredecesores(vertex, messages);
41
42                          // Continuar con el resto de la estructura del grafo
43                          BFSMessages(vertex);
44                  }
45          }
46          vertex.voteToHalt();
47  }

48  private void configurarMensajesDePredecesores(
49                  Vertex vertex, Iterable<Text> messages) {
 50         Iterator<Text> iteradorDeMensajes = messages.iterator();

           // Se configuran los predecesores al vertice en caso de que
           // haya
51         if (iteradorDeMensajes.hasNext()) {
52                  // Obtenemos los predecesores hasta el momento (puede
haber
53                  // de un procesamiento previo)
54                  String mensajesDeVerticesPredecesores = vertex
55                                  .getMensajesVerticesPredecesores();
56
57                  while (iteradorDeMensajes.hasNext()) {
58                          Text next = iteradorDeMensajes.next();
```

```
59                          mensajesDeVerticesPredecesores += \t
60
61                          // Si el mensaje viene de un vertice predecesor (es
decir,no
62                          // hicimos un camino ciclico), lo aceptamos
63                          if (!contieneCiclos(vertex, next)) {
64                                  mensajesDeVerticesPredecesores +=
next.toString()
65                                                  + " "   + vertex.getId() + \t;
66                          } else {
67                                  // No se agrega, es un ciclo
68                          }
69                  }
70                  vertex.setMensajesVerticesPredecesores(
71                                  mensajesDeVerticesPredecesores);
              }
      }
}
```

Imagen 4-6: Pseudo-código del algoritmo en su versión con un vértice de inicio y uno de destino.

Fuente: Elaboración propia

Esta versión del algoritmo necesita de los siguientes parámetros para su funcionamiento: *Vértice de origen*, *Vértice de destino* y *Longitud máxima de caminos navegacionales buscados.* Estos parámetros son declarados como constantes para simplificar la explicación de esta versión del algoritmo *.*

4.2.1.1 – Funcionamiento del algoritmo

El funcionamiento de esta versión, suponiendo una longitud máxima de N para los caminos navegacionales, es el siguiente:

Luego de cargar el grafo con todos sus vértices y aristas en el superpaso -1, funcionamiento estándar de Giraph para cualquier algoritmo, se inicia el algoritmo en si de búsqueda de caminos navegacionales.

El algoritmo inicia su funcionamiento en el método Compute (linea 27), tal cual se explicó en la Sección 3.4.2. Podemos observar que el método *compute* tiene un parámetro de tipo Vertex (el cual es la clase con la cual Giraph representa un vértice a nivel de código) y un parámetro de tipo Iterable<Text> de mensajes representados por la colección messages, los cuales son de tipo Text (el cual es lo más parecido a un String que puede encontrarse en Giraph).

Se debe recordar de la sección 3.4.1, que la ejecución de un algoritmo en Giraph se da a través de una serie de iteraciones o superpasos. Por lo que, el lector deberá estar familiarizado con este concepto en orden de comprender esta explicación.

En la linea 29, se puede observar un IF que engloba a casi la totalidad del contenido del método compute, el cual controla que el superpaso actual, es decir el resultado del método getSuperstep(), sea menor que la longitud máxima de caminos navegacionales, la cual es determinada por el resultado del método maximaCantidadDeSuperpasosPosibles(). Por ejemplo, si estamos en el superpaso 1, y la longitud máxima de caminos navegacionales es 2, el if sera evaluado como verdadero. Solo dará falso en caso de que hayan transcurrido más superpasos que los permitidos por la longitud máxima de caminos navegacionales, el cual es un parámetro que este algoritmo recibe.

En el caso de que el IF de la linea 29 se haya evaluado como True, se procederá a determinar en que superpaso está el algoritmo. Pueden ocurrir dos casos muy distintos, que el superpaso sea el 0, es decir, que se esté ante la ejecución del primer superpaso, o que estemos ante la ejecución de un superpaso mayor al 0.

En el caso que se esté en el superpaso **0**, el IF de la linea 30, el cual controla que el superpaso sea 0, dará un resultado positivo. Habiendo determinado esta condición, el algoritmo buscara entre todos los vértices existentes en el grafo[69] al vértice de inicio, a través del IF de la linea 31 con el método esInicio(vertex). En el caso de que el vértice de inicio no sea encontrado, el algoritmo se da por finalizado. En el caso que si sea encontrado, el vértice agregará a través del método setMensajesVerticesPredecesores("camino"), un camino. Este camino estará compuesto únicamente por el identificador del vértice (recordar que es la propiedad <title> del dump de

69 Se debe recordar de la Sección 3.4.1 que todos los vértices están activos y por ende, todos los vértices ejecutaran la consulta de la linea 30 para determinar si es el superpaso 0 y la de la linea 31, para determinar si cada uno de ellos es el vértice de inicio.

Wikipedia). En adición a esto, el vértice de inicio enviará a todos los vértices adyacentes un mensaje, a través del método que se encuentra en la linea 15 llamado BFSMessages(). El contenido de este mensaje es el valor devuelvo por el método getMensajesVerticesPredecesores() aplicado al vértice de inicio. Este mensaje, hasta este momento y hasta que finalice el algoritmo, conforma un camino, el cual, como todos los caminos encontrados (es decir, todos los caminos que contenga que devuelva el método getMensajesVerticesPredecesores() aplicado a un vértice particular), será aceptado como válido si y solo si logra llegar al vértice de destino en la cantidad de superpasos indicada (o en menos). En caso de que no se pueda encontrar el vértice de origen, el algoritmo termina.

Del superpaso **1** hasta el último, todos los vértices del algoritmo están preparados para recibir y procesar mensajes. Al ser el superpaso distinto de 0, siempre el IF de la linea 30 dará como resultado False, por ende, todos los vértices ejecutarán el código de las lineas 40 y 43, sin excepción. La linea 40 se encarga de procesar los mensajes que un vértice recibió de sus predecesores, es decir, de todos los vértices que lo tenían como vértice adyacente a través de sus aristas. Este procesamiento se realiza a través del método configurarMensajesDePredecesores(vertex,messages), el cual se encarga de armar un String con todos los caminos (es decir, con todos los mensajes), encontrados hasta este punto, agregando su identificador al final de cada uno de los caminos (en la linea 65 se puede observar que es concatenado el identificador del vértice actual) para así especificar que los caminos lo contienen, y separando los distintos caminos con un TAB, además de descartar cualquier ciclo que se detecte; al terminarse de ejecutar el método configurarMensajesDePredecesores(vertex,messages) , este almacena el valor de todos los caminos en el vértice que se esta procesando en la linea 70, a través del método setMensajesVerticesPredecesores(mensajesDeVerticesPredecesores).

Una vez finalizado el método configurarMensajesDePredecesores(vértice) de la linea 40, se ejecuta el método BFSMessages(vertex) en la linea 43, el cual determinará si es necesario enviar mensaje a vértices adyacentes, con el fin de continuar con la búsqueda de caminos navegacionales. Este método controla si se está en el vértice de destino a través de la linea 16 y el método esDestino(vertice). Si esto es cierto, no se continúa con el envío de mensajes, dado que cualquier camino hallado entre los vértices de origen y destino a partir de este punto del recorrido del grafo, implicaría encontrar un ciclo. Si eso es falso, es decir el vértice no es el de destino, se envían los caminos del vértice que esta siendo actualmente procesado, hacia los vértices adyacentes, con el fin de intentar encontrar el destino de la búsqueda de caminos en los nodos adyacentes. Este envió se realiza

haciendo un split (linea 19) a través del delimitador TAB, a partir del resultado de vertex .getMensajesVerticesPredecesores() y de esta forma, separamos los caminos para que sean enviados en distintos mensajes a todos los vértices adyacentes, a través del método sendMessageToAllEdges(vertex, new Text(mensajeDeUnVerticePredecesor)) de la linea 20. El método sendMessageToAllEdges() es proporcionado por Giraph para simplificar el envió de mensajes desde un vértice a todos sus adyacentes.

En el caso de que se llegue al superpaso **N + 1**, todos los vértices se detienen instantáneamente, dado que el IF de la linea 29 envía a que se detengan a la fuerza a través del método de Giraph voteToHalt() (linea 46), dado que se deben encontrar caminos válidos entre el vértice de origen y destino bajo una cantidad máxima de N pasos.

4.2.1.2 - Problemas encontrados en esta versión

Esta solución tardaba un tiempo demasiado alto, 40 segundos de promedio, en encontrar los caminos navegacionales entre dos artículos, fundamentalmente por una constante de tiempo que no se podía disminuir, la cual estaba relacionada a el tiempo que tarda Giraph en iniciar un procesamiento.

Se decidió ir por una versión mejor, buscando múltiples destinos desde un mismo origen. Dicha versión del algoritmo se explica a continuación.

4.2.2 - Versión con un vértice de origen y múltiples destinos

El pseudo-código de esta versión[70] del algoritmo es el siguiente:

```
1 public class BusquedaDeCaminosNavegacionalesWikiquote{

2   public String SOURCE_ID = "A";
3    public String DESTINATION_ID = "B-@-C";
```

70 BusquedaDeCaminosNavegacionalesWikiquote, un vértice de origen y múltiples vértices de destino, referencia BitBucket, 28/12/2016,<https://bitbucket.org/ambartsumian/tesina/src/d30e02f07cfeac0e2d0f687e2c340617a5fb0b65/generacion_entrada_giraph/procesamiento_grafo_wikiqoutes/src/main/java/ar/edu/info/unlp/tesina/lectura/grafo/BusquedaDeCaminosNavegacionalesWikiquote.java?at=master&fileviewer=file-view-default>

```
4   public Integer MAX_SUPERSTEPS = 5;
5   public String SEPARADOR_VERTICES = "-@-";
6   public String LINE_TOKENIZE_VALUE_DEFAULT = "\t";
7   public String getSeparadorDeVertices() {
8      return SEPARADOR_VERTICES;
9   }
10
11   public boolean esInicio(Vertex<Text, ?, ?> vertex) {
12      return vertex.getId().toString().equals(SOURCE_ID.toString());
13   }
14
15   public boolean contieneVertice(String[] coleccionDeVertices,String
verticeBuscado) {
16      return Arrays.asList(coleccionDeVertices).contains(verticeBuscado);
17   }
18   private int maximaCantidadDeSuperpasosPosibles() {
19      return MAX_SUPERSTEPS;
20   }
21
22   public void BFSMessages(Vertex vertex, List<String> mensajesPorEnviar) {
23
24      List<String> caminos =
caminosNavegacionalesParaReenviarAVerticesVecinos(vertex);
25      List<String> resultadosAImprimirComoSalida = new ArrayList<String>();


26      if (esDestino(vertex)) {
27         resultadosAImprimirComoSalida.addAll(caminos));
28      }

29      // En caso de que sea un vertice destino, debemos guardar los mensajes
```

```
30        // resultantes (sin perder resultados anteriores)
31        vertex.setMensajesVerticesPredecesores(

32        armarMensajesSeparadosPorTAB(resultadosAImprimirComoSalida));

33        // Como puede haber pares de vertices cuyo destino no es el vertice
34        // actual, reenviamos los mensajes que se corresponden con vertices de
35        // destino que no han sido aún encontrados, solo en el caso que que no
36        // estemos en el ultimo superpaso (si lo estamos, nadie recibira los
37        // mensajes en el superpaso siguiente, dado que se detendran los
38        // vertices sin importar nada mas)
39        if (getSuperstep() < maximaCantidadDeSuperpasosPosibles()) {

40            for (String mensajeDeUnVerticePredecesor : mensajesPorEnviar) {
41                sendMessageToAllEdges(vertex, new
Text(mensajeDeUnVerticePredecesor));
42            }
43        }

44    }

45  private List<String>
caminosNavegacionalesParaReenviarAVerticesVecinos(Vertex vertex) {
46        return
vertex.getMensajesVerticesPredecesores().split(SEPARADOR_VALORES_EN_LINEA))
{
47    }

48    private String getVerticeDeInicio(String mensaje) {
49        return mensaje.split(getSeparadorDeVertices())[0];
50    }
```

```
51  @Override
52  public void compute(Vertex vertex,Iterable<Text> messages) {
 53
54      // Se fuerza a que los vertices converjan en el superpaso máximo
55      if (getSuperstep() <= maximaCantidadDeSuperpasosPosibles()) {

56          List<String> mensajesPorEnviar = new ArrayList<String>();

57          // Solo el vertice de inicio y los de destino deberia trabajar en el
58          // primer superpaso
59          // Todos los demas deberian detenerse y esperar por mensajes
60          if (getSuperstep() == 0) {
61              if (esInicio(vertex)) { // Es vertice de inicio
62                  vertex.getValue().setMensajesVerticesPredecesores(
63                  vertex.getId().toString());
64                  mensajesPorEnviar.add(vertex.getId().toString());
65                  BFSMessages(vertex, mensajesPorEnviar);
66              } else {
67                  if (esDestino(vertex)) { // Es vertice de destino
68                      vertex.setVerticeDestino(Boolean.TRUE);
69                      vertex.setIdVerticeInicio(getVerticeInicio());

70                  }
71              }
72          } else {
73              if ((getSuperstep() < maximaCantidadDeSuperpasosPosibles() ||
(getSuperstep() == maximaCantidadDeSuperpasosPosibles() &&
esDestino(vertex)))) 74 {

75                  // Procesamos los mensajes que llegan desde vertices predecesores
```

```
76
mensajesPorEnviar.addAll(configurarMensajesDePredecesores(vertex, messages));

77              // Continuar con el resto de la estructura del grafo
78                  BFSMessages(vertex, mensajesPorEnviar);
79              } else {
80                  // Vaciamos mensajes de predecesores, de otro modo seran listados
como caminos navegac.
81                  vertex.getValue().setMensajesVerticesPredecesores("");
82              }
83          }
84      }
85  }

86  private String armarMensajesSeparadosPorTAB(List<String> mensajes) {
87      String separador = "";
88      StringBuilder mensajesArmadosStringBuilder = new StringBuilder();
89          for (String mens : mensajes) {
90              mensajesArmadosStringBuilder.append(separador).append(mens);
91              separador = SEPARADOR_VALORES_EN_LINEA;
92          }
93      return mensajesArmadosStringBuilder.toString();
94  }

95  private String getVerticeInicio() {
96      return SOURCE_ID;
97  }

98  public boolean esDestino(Vertex vertex) {
99      String[] coleccionDeVertices =
DESTINATION_ID.split(getSeparadorDeVertices());
```

```
100         return contieneVertice(coleccionDeVertices, vertex.getId().toString());
101     }

102     private List<String> configurarMensajesDePredecesores(Vertex
vertex,Iterable<Text> messages) {

103         Iterator<Text> iteradorDeMensajes = messages.iterator();
104         List<String> mensajesAEnviar = new ArrayList<String>();

105         // Se configuran los predecesores al vertice en caso de que haya
106         if (iteradorDeMensajes.hasNext()) {

107             // Obtenemos los predecesores hasta el momento (puede haber de un
procesamiento previo)
108             String mensajesDeVerticesPredecesores =
vertex.getMensajesVerticesPredecesores();

109             // Dado que pueden llegar a hacerse muchas concatenaciones, es
necesario usar un StringBuilder
110             StringBuilder mensajesDeVerticesPredecesoresStringBuilder = new
StringBuilder(mensajesDeVerticesPredecesores);

111             String mensajeAEnviar;
112             while (iteradorDeMensajes.hasNext()) {

113                 Text next = iteradorDeMensajes.next();

114                 // Si el mensaje viene de un vertice predecesor (es decir, no hicimos
un camino ciclico), lo aceptamos
115                 if (!contieneCiclos(vertex, next)) {
```

```
116                 mensajeAEnviar = next.toString() + getSeparadorDeVertices() +
vertex.getId();
117
mensajesDeVerticesPredecesoresStringBuilder.append(separador).append(mensaj
eAEnviar);

118                 mensajesAEnviar.add(mensajeAEnviar);
119                 separador = SEPARADOR_VALORES_EN_LINEA;
120             } else {
121                 // Es un ciclo, no se realiza ninguna acción
122             }
123         }

124         mensajesDeVerticesPredecesores =
mensajesDeVerticesPredecesoresStringBuilder.toString();

125         // Configuramos nuevamente los mensajes de vertices pred.
126         // agregando los nuevos mensajes (exceptuando los ciclos)
127
vertex.setMensajesVerticesPredecesores(mensajesDeVerticesPredecesores);
128       }
129       return mensajesAEnviar;
130   }

131   private Boolean contieneCiclos(Vertex vertex,Text mensajeDeUnPredecesor)
{
131       return
mensajeDeUnPredecesor.toString().contains(vertex.getId().toString());
132   }
133 }
```

Imagen 4-7: Pseudo-código del algoritmo en su versión con un vértice de inicio y múltiples destinos.
Fuente: Elaboración propia

Esta versión del algoritmo necesita los siguientes datos para poder realizar el procesamiento: Un *conjunto de pares de vértices origen-destino*, en los cuales hay que buscar caminos (el vértice de origen en cada par debe ser el mismo) y *Longitud máxima de caminos buscados*. El lector debe tomar nota que la razón por la cual no se detallan los vértices de destino, es por que el destino depende del vértice de origen, razón por la cual se piden *pares* de vértices origen-destino (a diferencia de la versión anterior)

4.2.2.1 – Funcionamiento del algoritmo

El funcionamiento de esta versión del algoritmo, suponiendo una longitud máxima de N para los caminos, es el siguiente:

En el superpaso **0** (cuando en la linea 60 se evalúa la condición del IF como verdadera), se dispara desde el vértice de origen el recorrido BFS. Esto se realiza enviando un mensaje desde el vértice de origen a todos sus adyacentes (linea 65). El contenido de este mensaje es el identificador del vértice (recordar que es la propiedad <title> del dump de Wikipedia). Este mensaje, hasta este momento y hasta que finalice el algoritmo, conforma un camino, el cual, como todos los caminos encontrados, será aceptado como válido si y solo si logra llegar a un vértice de destino (recordar que la versión anterior solo poseía uno, pero ahora se dispone de varios destinos) en la cantidad de superpasos indicada (o en menos). En caso de que no se pueda encontrar el vértice de origen, el algoritmo termina.

En esta versión del algoritmo, también se inicializan todos los vértices de destino, a través de la configuración de una marca (de tipo Boolean) para indicar que estamos, justamente, ante un vértice de destino. Para configurar el valor de esta marca, se compara el identificador de cada vértice con con el conjunto de identificadores de destinos buscados (linea 66), y si el identificador del vértice se halla incluido en dicho conjunto, la marca se configura con el valor True (lineas 67, 68 y 69).

Del superpaso **1** hasta el anteúltimo (cuando en la linea 60 se evaluá la condición del IF como falsa), todos los vértices del algoritmo están preparados para recibir y procesar mensajes. Al recibir un mensaje en estos superpasos, se controla que no implique haber recorrido el grafo en forma cíclica. Para realizar esto, cada vértice busca su identificador dentro del contenido del mensaje (linea 115). Si el identificador del vértice está dentro del

camino, quiere decir que se está ante la presencia de un ciclo, por ende se descarta el mensaje.

Habiendo descartado los ciclos, se toma al mensaje como un camino válido (linea 115) y el vértice agrega su propio identificador al final de cada uno de los mensajes (lineas 116, 117, 118 y 119). Una vez que el vértice realizó esta acción, se controla si se está en el vértice de destino, dado que en todo momento disponemos de múltiples vértices de destino. Si se está en el vértice de destino (linea 26), todos los caminos obtenidos en este punto son informados como caminos válidos para este destino (lineas 27, 31 y 32). Aunque se almacene dicho mensaje o no como camino resultante valido, el total de los mensajes se envía a los vértices adyacentes (linea 39, 40 y 41), dado que aunque hallamos encontrado un vértice de destino, esto no quiere decir que en algún otro lugar del grafo, el mismo camino se agrande y termine siendo válido, con más vértices en su haber, como camino para un par de vértices origen-destino distinto. He aquí otra diferencia con el algoritmo anterior, al tener múltiples destinos, no podemos en ningún momento dejar de enviar un mensaje, dado que un mensaje, a lo largo del recorrido que realiza en el grafo y a medida que adquiere nuevo contenido a través de nuevos identificadores, puede conformarse como camino válido para múltiples destinos distintos. Esto incrementa la cantidad total de mensajes a enviar sustancialmente comparado con la anterior versión.

En el último superpaso, se descartan los caminos que impliquen un ciclo y luego se almacenan los caminos que signifiquen un camino válido para el par origen-destino dado, pero bajo ninguna razón se envían mensajes (es decir, la linea 39 dara False), dado que este es el último superpaso que se ejecutará, y cualquier mensaje enviado solo enlentecerá el tiempo que tarda el algoritmo, pero sin aportar nuevos resultados. La razón del cambio del comportamiento en el último superpaso del algoritmo es que, los únicos que deberían controlar si hay caminos navegacionales válidos en dicho momento es el vértice de destino, dado que en cualquier otro caso, el pasar por un vértice que no es de destino, no aportará nada nuevo en este punto, es decir, cuando el algoritmo está por terminar el recorrido. Está diferenciación de no enviar ningún mensaje en el último superpaso también es una mejora respecto a la anterior versión.

Si por alguna razón algún vértice se ejecuta en un superpaso N + 1, dichos vértices se detienen sin realizar acción alguna(voteToHalt()).

4.2.2.2 - Problemas encontrados en esta versión

Esta versión del algoritmo aumenta la cantidad de mensajes enviados, dado que al haber múltiples destinos, siempre se debe reenviar los mensajes a los vértices adyacentes aunque ya hayamos encontrados el destino[71], debido a que puede haber un vértice destino que no haya sido alcanzado aún. La cantidad de mensajes que se llegaban a generar en los peores casos hacía que el algoritmo colapse por completo, produciendo un error bien conocido en la comunidad de Giraph, que es que cuando se envían muchos mensajes pequeños en un instante reducido de tiempo, el framework no siempre funciona bien y termina fallando. Esto tiene que ver con una funcionalidad de Giraph denominada out-of-core, la cual se activa cuando la cantidad de mensajes supera a la máxima que puede manejar un algoritmo de acuerdo a la memoria disponible, y empieza a paginar[72] memoria al disco con parte de los mensajes. Este problema debería verse resuelto en la versión de Giraph 1.2, la cual debería estar disponible pronto.

4.2.3 - Explicación gráfica del funcionamiento del algoritmo

Al ser complejas de explicar ambas versiones del algoritmo, se introducen a continuación algunos ejemplos simples con imágenes, para poder entender cómo funcionan en una forma muy básica y/o genérica.

4.2.3.1 - Explicación de búsqueda de caminos en un grafo sin ciclos

Suponiendo que debemos buscar los caminos entre los vértices A y F, en un grafo idéntico al de la Imagen 4-8, en la cual podemos ver que hay un grafo sin ciclos, se detallara paso a paso como el algoritmo buscará los caminos válidos entre el vértice de origen y el de destino.

71 En la versión de un inicio y un destino, al encontrar el destino, no se enviaban los mensajes a los vértices adyacentes, dado que en todos los casos se hubiera incurrido en un camino cíclico
72 Paginado de memoria, Wikipedia, 15/02/2017 <https://es.wikipedia.org/wiki/Paginación_de_memoria>

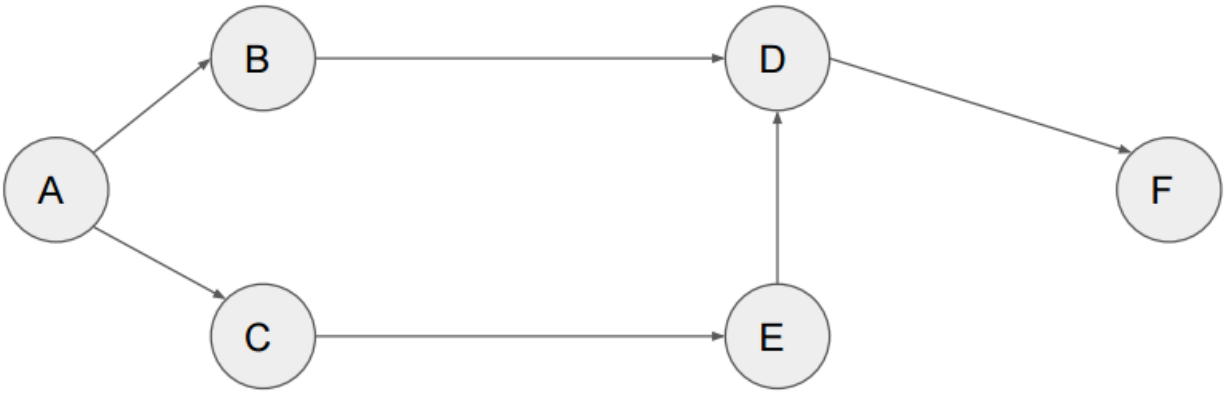

Imagen 4-8: Grafo sin ciclos - Búsqueda de caminos
Fuente: Elaboración propia

El algoritmo buscara en el primer superpaso, es decir, en el superpaso 0, el vértice de origen A. Si el vértice de origen no existiese, se da por terminado el algoritmo, sin haber encontrado ningún camino. Como en este caso, el vértice A existe, se envía desde A, un mensaje a través de todas sus aristas. Este mensaje tendrá el mismo contenido, y sera el valor A, indicando que hasta el momento, los caminos que van desde A hasta F están formados, justamente, por A, dado que es el el único vértice por el cual pasó el algoritmo hasta el momento. Se puede observar que en la Imagen 4-9 solo A esta con un color distinto al resto, dado que es el único en que se realizo actividad alguna.

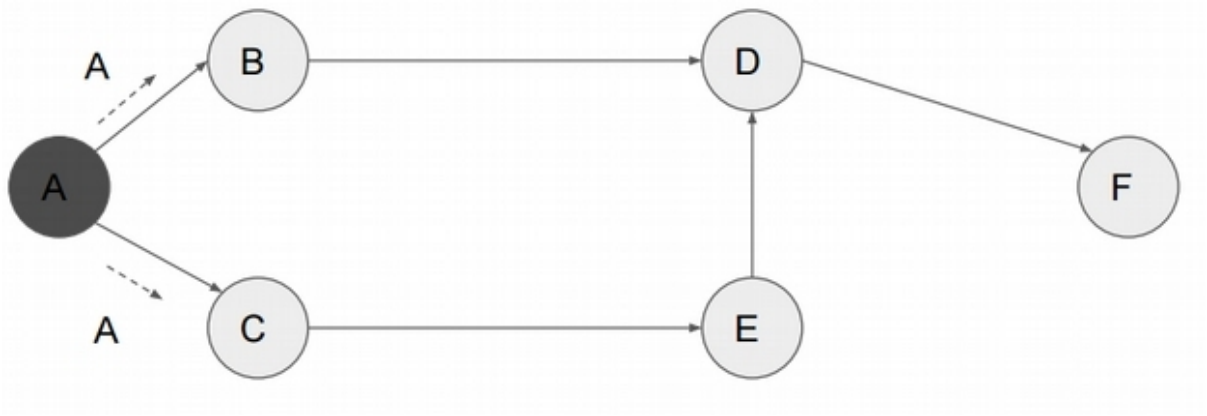

Imagen 4-9: Grafo sin ciclos - Búsqueda de caminos – Superpaso 0

Al terminar el superpaso 0 y iniciar el superpaso 1, se puede observar en la Imagen 4-10 que tanto los vértices B como C se activan, dado que recibieron un mensaje

proveniente de A. Tanto B como C, intentarán continuar en la búsqueda del vértice destino F, por ende, se agregarán a si mismos en los caminos hasta ahora encontrados, y enviarán dichos mensajes a sus vértices adyacentes. Por ejemplo, C, al recibir un mensaje con contenido A, agrega su identificación, es decir C al contenido del mensaje y lo envía a sus adyacentes, es decir a E. El funcionamiento es idéntico con B y el envío de mensaje que realiza a su vértice adyacente D.

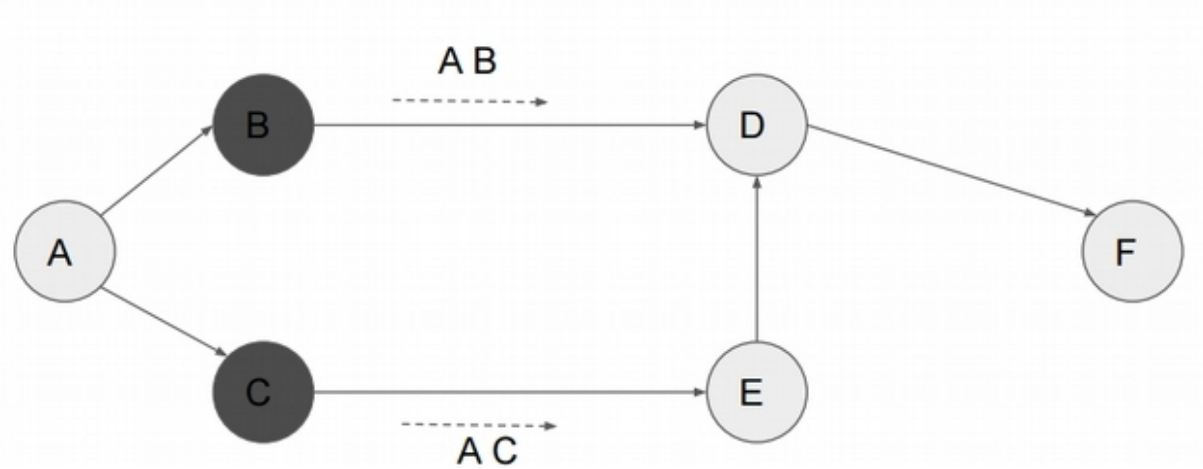

Imagen 4-10: Grafo sin ciclos - Búsqueda de caminos – Superpaso 1

Fuente: Elaboración propia

Al iniciarse el superpaso 2, se puede comprender, mirando la Imagen 4-11, que tanto D como E están activos, como consecuencia de haber recibido un mensaje enviado desde el superpaso anterior desde B y C, respectivamente. Se puede notar que E, al recibir un mensaje con el contenido A C, enviará un mensaje a sus vértices adyacentes, el cual contendrá el valor A C E, indicando que también E es parte del posible camino a encontrar entre A y F (hasta no haber encontrado el vértice de destino, no tenemos la seguridad que sea un camino válido entre A y F), y dicho mensaje será recién entregado a D en el superpaso 3, tal cual indica el modelo BSP [17]. El vértice D, por su parte, recibió un mensaje con el contenido A B, y le agrego su identificador al mismo y envió el mensaje A B D a sus vértices adyacentes, es decir, a F.

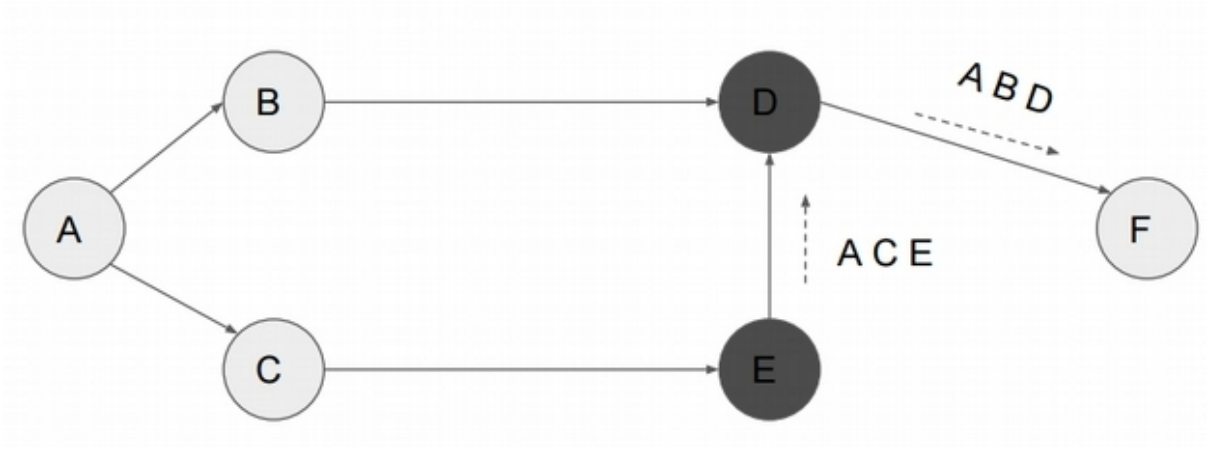

Imagen 4-11: Grafo sin ciclos - Búsqueda de caminos – Superpaso 2

Fuente: Elaboración propia

Al iniciarse el superpaso 3, el vértice D se activará por el mensaje enviado desde E en el superpaso anterior, y enviará hacia sus vértices adyacentes, el contenido del mensaje original (A C E) con la adición de D al final, es decir, enviará a F el mensaje A C E D. El vértice de destino F recibirá en este superpaso el mensaje enviado por el vértice D en el superpaso 2, y al detectarse como un vértice de destino, este no enviará ningún mensaje hacia ningún otro vértice adyacente, sino que se guardara como camino válido entre A y F al mensaje A B D F. Esto quiere decir que, al momento de producir los resultados de salida, aparecerá el camino A B D F como resultado del algoritmo. Todo esto puede ser observado en la Imagen 4-12.

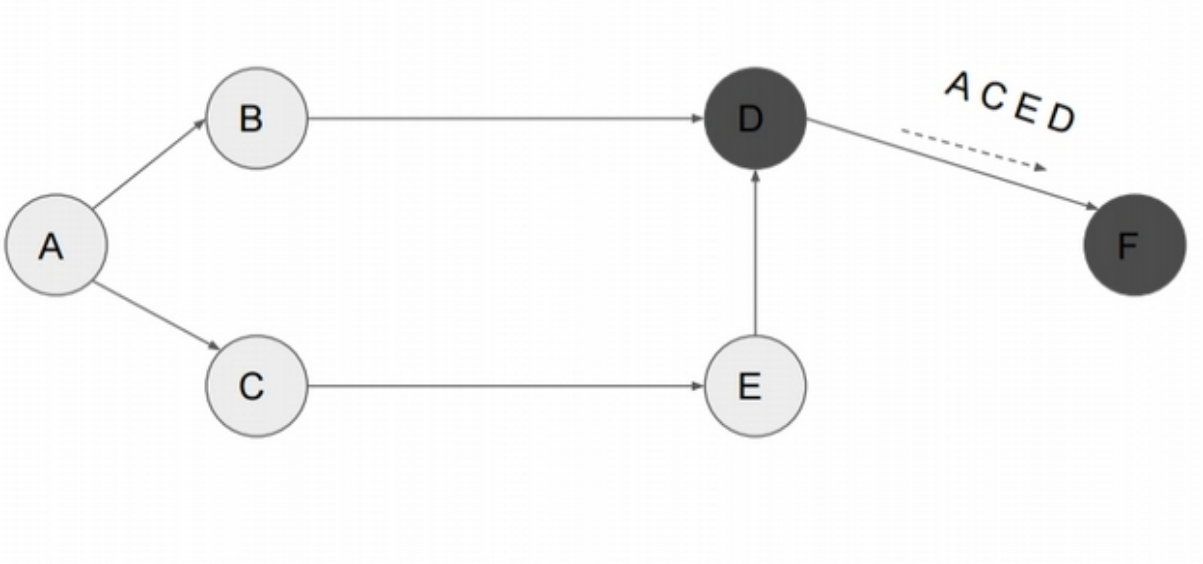

Imagen 4-12: Grafo sin ciclos - Búsqueda de caminos – Superpaso 3

Fuente: Elaboración propia

Al iniciarse el superpaso 4, el único vértice que realizará procesamiento alguno es F, tal como indica la Imagen 4-13, dado que se activó por el mensaje enviado desde D en el superpaso anterior. Como este mensaje no contiene un ciclo, y también se inició en A y tenía como vértice de destino a F, este mensaje con contenido A C E D F pasará a ser un nuevo camino válido y será almacenado por F como resultado válido de este algoritmo.

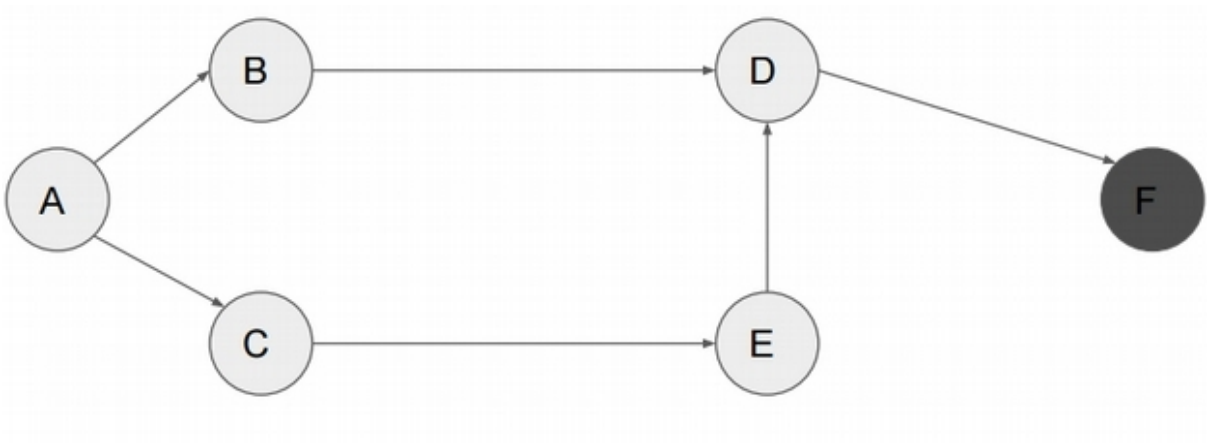

Imagen 4-13: Grafo sin ciclos - Búsqueda de caminos – Superpaso 4
Fuente: Elaboración propia

4.2.3.2 - Explicación de búsqueda de caminos en un grafo con ciclos

Suponiendo que debemos buscar los caminos entre los vértices A y E, en un grafo idéntico al de la Imagen 4-14, en la cual podemos ver que hay un grafo sin ciclos, se detallara paso a paso como el algoritmo buscará los caminos válidos entre el vértice de origen y el de destino.

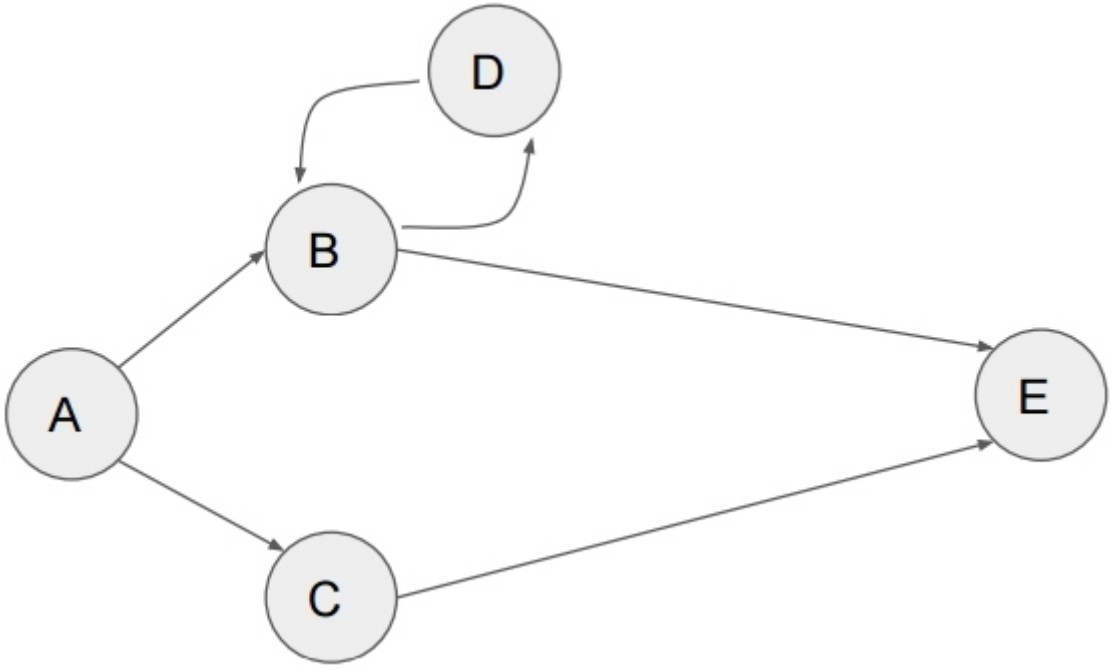

Imagen 4-14: Explicación de búsqueda de caminos en un grafo con ciclos

Fuente: Elaboración propia

Al iniciar el algoritmo, se buscará el vértice de inicio A. En caso de que no exista, el algoritmo finaliza sin encontrar ningún camino. En el ejemplo de la Imagen 4-14, este vértice existe, por ende, en el superpaso 0, este vértice envía a sus vértices adyacentes un mensaje que tiene como contenido A, tal cual indica la Imagen 4-15.

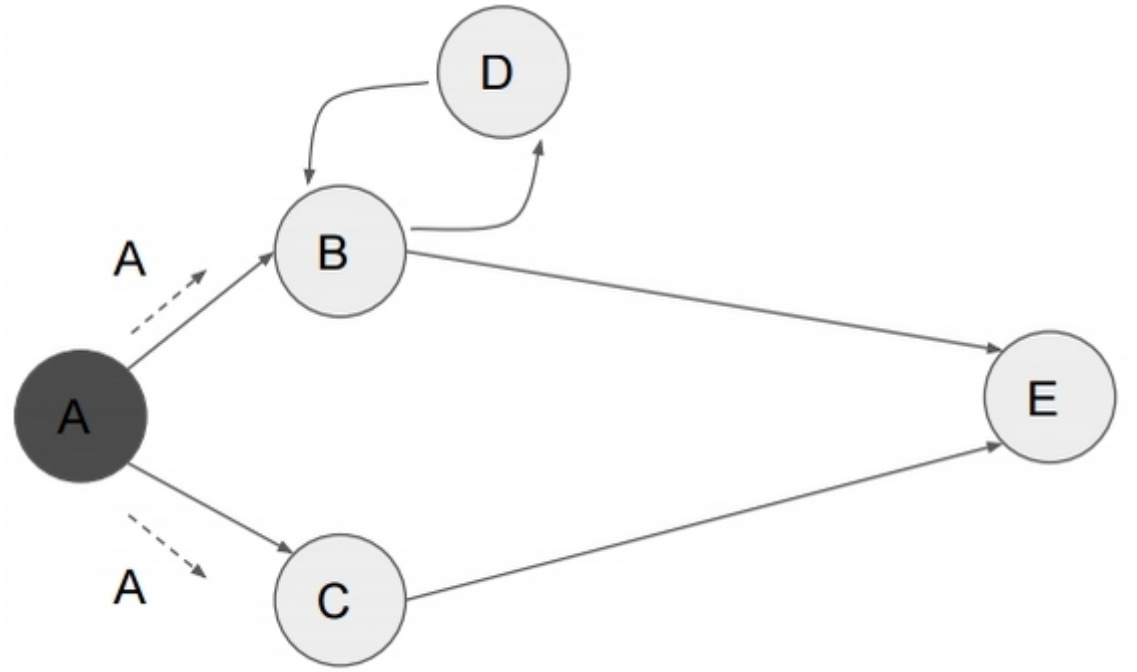

Imagen 4-15: Explicación de búsqueda de caminos en un grafo con ciclos – Superpaso 0

Fuente: Elaboración propia

En el superpaso 1, ilustrado en la Imagen 4-16, se activan los vértices B y C cuando reciben un mensaje con el contenido A. Al activarse el vértice B por recibir un mensaje con contenido A, le agrega su identificador al mismo, y enviá a los vértices adyacentes el mensaje A B, mensaje que sera recibido en el superpaso 2. El vértice C en cambio, recibió el mensaje A, y envió a sus adyacentes el mensaje A C.

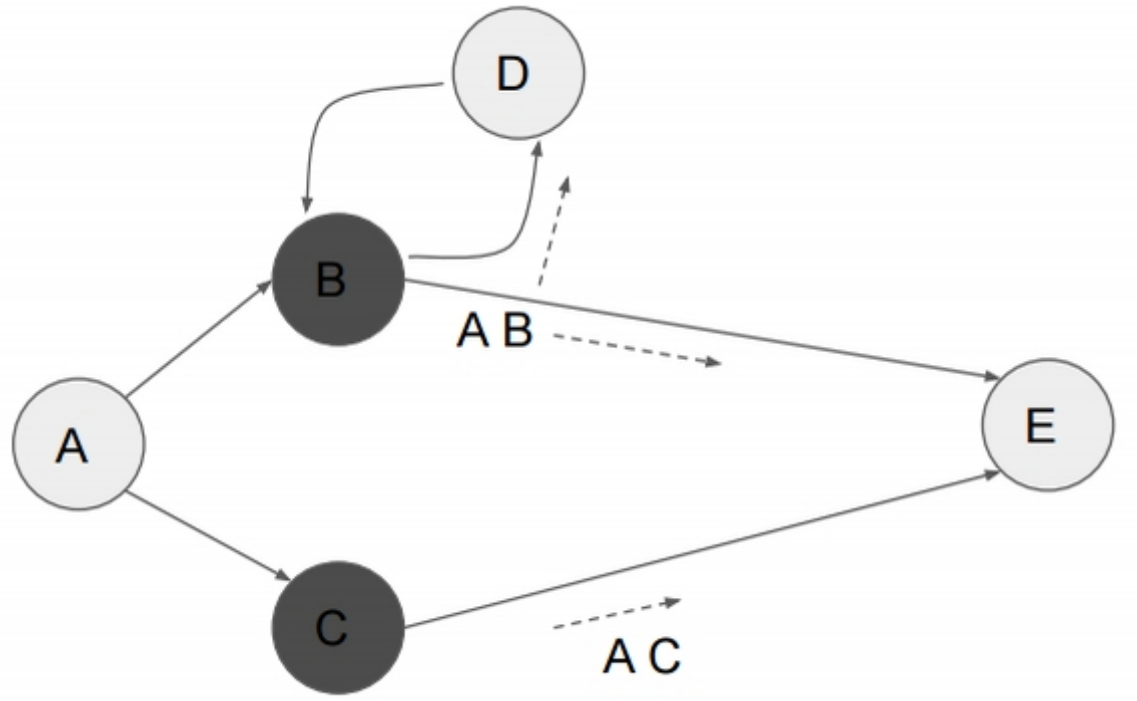

Imagen 4-16: Explicación de búsqueda de caminos en un grafo con ciclos – Superpaso 1

Fuente: Elaboración propia

En el superpaso 2, indicado por la Imagen 4-17, se activan los vértices D y E. El vértice D recibió el mensaje con contenido A B, por ende este vértice enviará a todos sus adyacentes, el mensaje A B D. E detectará que es un vértice de destino respecto a A, por ende, almacenará el recorrido A C E como camino válido entre A y E, y no enviará ningún mensaje a sus vértices adyacentes.

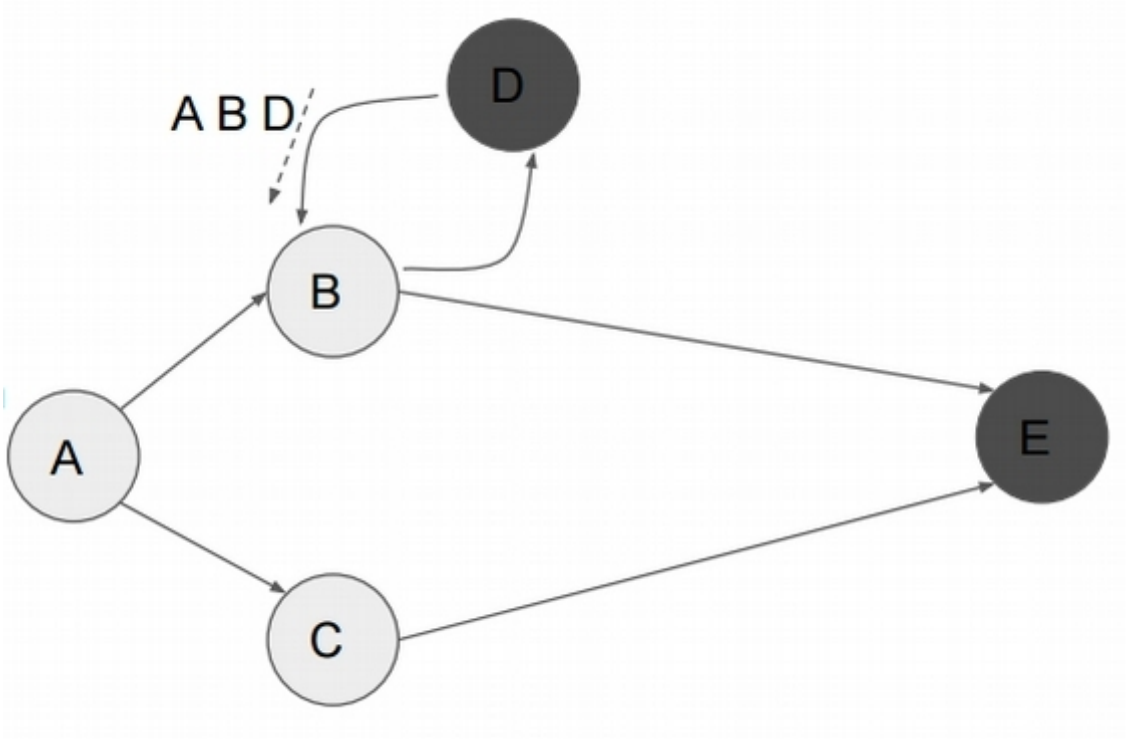

Imagen 4-17: Explicación de búsqueda de caminos en un grafo con ciclos – Superpaso 2

Fuente: Elaboración propia

En el superpaso 3, indicado por la Imagen 4-18 , se activa nuevamente el vértice B, al recibir el mensaje A B D. El vértice revisa si el mensaje implica que hay un ciclo, al buscar el contenido de su propio identificador en el mismo, es decir, al buscar el valor B dentro del mensaje A B D. Como efectivamente B está contenido en el mensaje A B D, el algoritmo descarta este camino, por ende B no vuelve a enviar ningún mensaje, y el algoritmo termina.

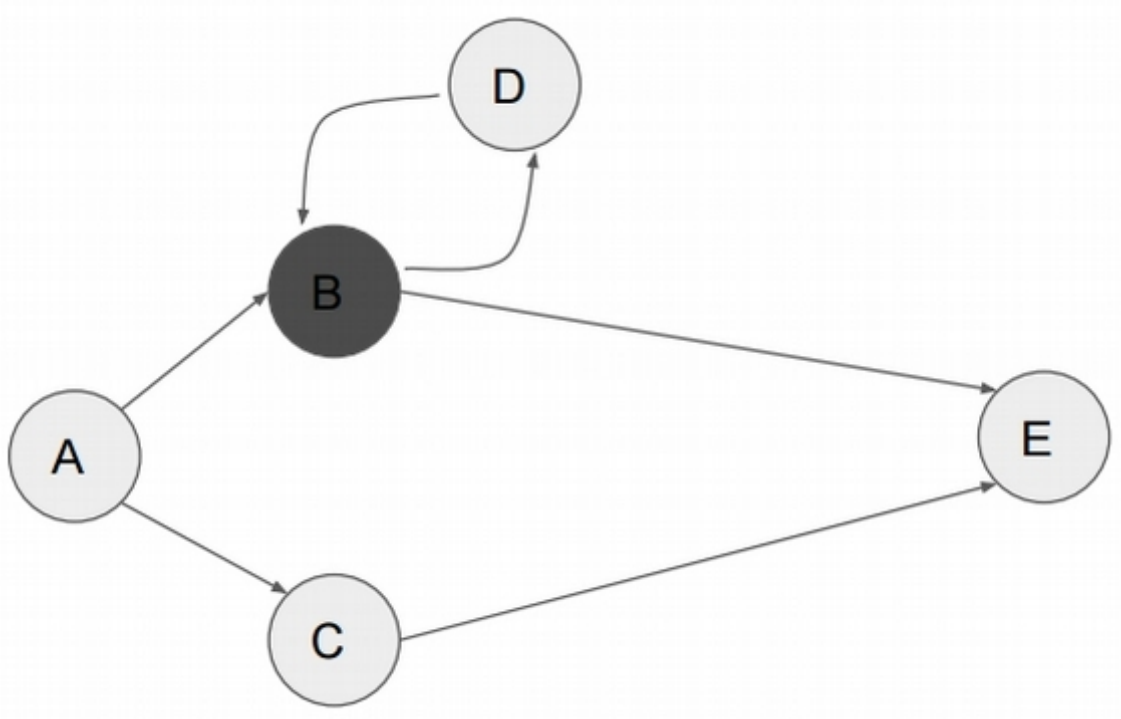

Imagen 4-18: Explicación de búsqueda de caminos en un grafo con ciclos – Superpaso 3

Fuente: Elaboración propia

Habiendo explicado en que consiste el grafo de entrada que usa el algoritmo, y cómo trabaja el algoritmo en sus dos versiones, queda por explicar el formato en que produce sus resultados. Esta información se introduce a continuación.

4.3 - Construcción del archivo de salida

Así como la información en Giraph debe tener un formato de entrada para poder ser interpretada correctamente por un algoritmo de este framework, también debe tener un **formato de salida**, de forma que el resultado de la ejecución de cada uno de los vértices pueda ser almacenado en HDFS para su posterior utilización.

4.3.1 - Versión con un vértice de origen y uno de destino

En la versión de un solo vértice de origen y un solo vértice de destino, la información de salida consiste, literalmente, en los mensajes que almacenó el vértice de destino al terminar el recorrido de algoritmo. El lector debe recordar de la explicación que precede a esta sección, que cuando un vértice almacenaba un camino internamente, es por que dicho camino consistía en un camino navegacional válido entre el vértice de origen y el de destino. Cada uno de estos caminos contiene vértices del grafo separados por un espacio en blanco, y dichos vértices consisten en la secuencia de pasos que se debe repetir para recorrer el grafo a través de un camino válido entre el vértice de origen y el de destino, este es el simple **formato de salida** que deben respetar los resultados de cada vértice.

La información resultante del algoritmo se almacena en HDFS, y posteriormente se almacena en forma consecutiva en un archivo, al cual nos referiremos como **archivo de salida**, que tendrá los resultados de múltiples búsquedas de caminos navegacionales usando la versión algorítmica de un origen y un destino.

A continuación, en la Imagen 4-19 se muestra el resultado de una búsqueda de caminos entre el vértice de origen *1984 (novela)* y el de destino *The Matrix, y* en la Imagen 4-20 se muestra el conjunto de resultados de múltiples búsquedas de la primer versión del algoritmo, al guardarse en un archivo de salida, a través del script guardar-logs-local.sh[73].

1984 (novela) Londres Angela Carter Sueño The Matrix
1984 (novela) Sol Mark Twain Sueño The Matrix

Imagen 4-19: Ejemplo del formato de salida - Resultado de ejecutar una búsqueda de caminos entre 1984 (novela) y The Matrix
Fuente: Ejecución del algoritmo – Versión de un solo vértice de inicio y uno de destino

73 Ver apéndice.

1984 (novela) Londres Angela Carter Sueño The Matrix

1984 (novela) Sol Mark Twain Sueño The Matrix

Aborto Vida Mark Twain Sueño The Matrix

Abraham Lincoln Felicidad Mark Twain Sueño The Matrix

Aburrimiento Oscar Wilde Bigamia Sueño The Matrix

A. C. Bhaktivedānta Swami Prabhupāda Ceguera Anhelo Sueño The Matrix

A. C. Bhaktivedānta Swami Prabhupāda Ceguera Anhelo The Matrix

A. C. Bhaktivedānta Swami Prabhupāda Ceguera Soñar Sueño The Matrix

Imagen 4-20: Extracto de salida de múltiples búsquedas de caminos navegacionales – Algoritmo de un origen y un destino

Fuente: Información resultante de la ejecución del algoritmo – Versión un origen y un destino

4.3.2 - Versión con un vértice de origen y múltiples destinos

El algoritmo, en esta versión, genera información de salida usando el vértice de inicio y el de destino, separando el vértice de origen del vértice de destino de la búsqueda realizada por el delimitador -@@@-. A continuación de este símbolo, se listaran en lineas distintas los caminos encontrados, separando los distintos vértices que conforman cada camino por -@- (este valor puede ser configurado)

Por ejemplo, el resultado de la búsqueda entre *Virginia Auber Noya* y *Dificultades* es el que indica la Imagen 4-21, y el conjunto de múltiples búsquedas que podría conformar un archivo de salida, es el indicado en la Imagen 4-22 . Notar que se listan los caminos para cada par de vértices origen-destino encontrado, y en caso de no encontrar caminos para un par dado, se lista de todas formas el mismo.

Virginia Auber Noya-@@@--Dificultades-@@@-

Virginia Auber Noya-@-España-@-Vida-@-Día-@-Ralph Waldo Emerson-@-Dificultades

Virginia Auber Noya-@-España-@-Muerte-@-Día-@-Ralph Waldo Emerson-@-Dificultades

Imagen 4-21: Ejemplo del formato de salida - Resultado de ejecución del algoritmo – Versión un inicio y múltiples destinos

Fuente: Ejecución del algoritmo – Versión un inicio y múltiples destinos

Virginia Auber Noya-@@@--Dificultades-@@@-

Virginia Auber Noya-@-España-@-Vida-@-Día-@-Ralph Waldo Emerson-@-Dificultades

Virginia Auber Noya-@-España-@-Muerte-@-Día-@-Ralph Waldo Emerson-@-Dificultades

Virginia Auber Noya-@@@-El compromiso-@@@-

Virginia Auber Noya-@@@-Amin Maalouf-@@@-

Virginia Auber Noya-@@@-Elizabeth Heaphy de Murray-@@@-

Virginia Auber Noya-@@@-Faustina Sáez de Melgar-@@@-

Virginia Auber Noya-@@@-El camino de los sabios-@@@-

Virginia Auber Noya-@@@-Canek-@@@-

Virginia Auber Noya-@@@-Cartagena-@@@-

Virginia Auber Noya-@-España-@-Arturo Pérez-Reverte-@-Cartagena

Virginia Auber Noya-@-España-@-Idioma español-@-Arturo Pérez-Reverte-@-Cartagena

Virginia Auber Noya-@-España-@-Nacionalismo-@-Arturo Pérez-Reverte-@-Cartagena

Imagen 4-22: Extracto de salida de múltiples búsquedas de caminos navegacionales – Algoritmo de un origen y múltiples destinos

Fuente: Información resultante de la ejecución del algoritmo – Versión un origen y múltiples destinos

No son listados los pares de vértices que contienen vértices de origen o destino que no existen en el grafo, dado que al no existir el vértice de inicio o el de destino, es imposible que exista un camino entre ambos.

4.4 – Enlace con BlueFinder - Construcción de base MySQL a partir del resultado del algoritmo de búsqueda de caminos navegacionales

A partir de la ejecución del algoritmo de búsquedas de caminos navegacionales se genera un archivo con caminos navegacionales generados. Este archivo es generado a través del script **guardar-logs-local.sh**[74]

A partir de este archivo, se deja al lector la posibilidad de construir una base de datos MySQL, para tener un medio de consulta ágil de los resultados, así como una forma eficiente de unir los resultados de este trabajo con el algoritmo BlueFinder.

Este programa produce una base MySQL con el esquema que nos muestra la Imagen 4-23:

74 Ver Apéndice.

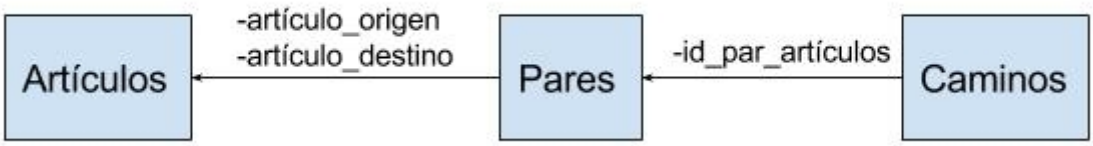

Imagen 4- 23: Esquema de base de datos que permite almacenar el archivo resultante de caminos navegacionales.

Fuente: Elaboración propia

Este programa[75] fue desarrollado bajo la versión Java 1.8, con MySQL Server instalado en su versión 5.5.54-0ubuntu0.14.04.1, ojdbc7 y un sistema operativo Ubuntu 14.04 LTS.

Antes de la ejecución de este algoritmo, se debe construir la base de datos que este programa usará, a través de la importación en MySQL del archivo bluefinderdb.sql[76]. Esto permitirá generar todas las tablas necesarias para que este programa funcione.

El funcionamiento de este pequeño programa es muy simple, y está particularmente basado en el formato del archivo generado por el script de recolección de los resultados del algoritmo de búsqueda de caminos navegacionales, guardar-logs-local.sh, recientemente mencionado. Se debe recordar que el archivo resultante, el cual es la acumulación secuencial de los resultados de cada una de las búsquedas de caminos navegacionales, contiene dos tipos de lineas, tal como indica la imagen 4-22:

- Un par de vértices para el cual hubo una búsqueda de caminos navegacionales
- Un camino navegacional, resultado de una búsqueda de caminos navegacionales para un par particular de vértices.

Por ende, el programa recorre secuencialmente el archivo, insertando los vértices en la tabla *artículos*, los pares en la tabla *pares* y los caminos navegacionales en la tabla *caminos*. Todas las inserciones se hicieron a través de JDBC en la base MySQL.

Esto permite consultar rápidamente, para un par de artículos dado, que caminos navegacionales existen, permitiendo una forma de integración entre el algoritmo desarrollado en esta tesina y el algoritmo BlueFinder.

75 ConstruirIndiceMySQL.java, referencia BitBucket, 8/03/2017,<https://bitbucket.org/ambartsumian/indice_bluefinder/src/d71654cf5a3b6ff183434eeaaa5348d285614058/src/main/java/construccion_base_datos/ConstruirIndiceMySQL.java?at=master&fileviewer=file-view-default>

76 Ver Apéndice.

Este programa se evaluó con un archivo resultante de caminos navegacionales de 1.800.000 caminos, y para este archivo, tardó menos de 11 minutos.

Corremos el programa a través del siguiente comando, previa descarga del programa via git y su compilación vía maven:

java -cp "repositorio_git_indice_bluefinder-0.0.1-SNAPSHOT.jar:mysql-connector-java-6.0.5.jar:ojdbc14-0.2.0.4.0.jar:commons-io-1.3.2.jar" construccion_base_datos.ConstruirIndiceMySQL <archivo-local-resultados-busqueda-caminos-navegacionales> <url-database> <usuario-bd> <clave-bd> <delimitador-pares-vertices>

Por ejemplo, para la ejecución local de este algoritmo, se uso el siguiente comando:

java -cp "repositorio_git_indice_bluefinder-0.0.1-SNAPSHOT.jar:mysql-connector-java-6.0.5.jar:ojdbc14-0.2.0.4.0.jar:commons-io-1.3.2.jar" construccion_base_datos.ConstruirIndiceMySQL /tmp/caminos jdbc:mysql://localhost/bluefinderdb bluefinder bluefinderdb8453 -@@@-

4.5 - Conclusiones del capítulo

En este capítulo se explicó el algoritmo desarrollado en profundidad para encontrar los caminos navegacionales existentes entre dos artículos de Wikipedia. Se inició explicando de que forma se debe construir un grafo de entrada con los datos de Wikipedia, para que dichos datos puedan ser correctamente cargados en Giraph, a través del uso de un archivo que respete el formato de datos de entrada. Una vez explicado esto, se explicó como funcionaba el algoritmo, con ambas versiones del mismo, con distintos ejemplos y a través de imágenes que relataban paso a paso que sucede, de modo que el lector no pierda el hilo de lo descrito. Al terminar estos temas, se comento de que forma se pueden recuperar el resultado del algoritmo, es decir, cómo a partir de los caminos navegacionales resultantes del algoritmo, podremos producir una base de datos MySQL que permita una sencilla interacción entre este algoritmo y BlueFinder.

Capítulo 5 - Evaluación de resultados

Índice de contenidos

Capítulo 5 - Evaluación de resultados....89
5.1 - Aspectos generales de la evaluación de resultados....89
5.1.1 - Variables para la evaluación....90
5.1.2 - Métricas usadas, en qué consisten y cómo fueron calculadas....91
5.2 - Evaluaciones en Wikiquotes....92
5.2.1 - Elección de los pares de vértices a utilizar....93
5.2.2 - Prueba 0....93
5.2.3 - Prueba 1-1....94
5.2.4 - Prueba 1-2....95
5.2.5 - Prueba 2-1....96
5.2.6 - Prueba 2-2....97
5.2.7 - Prueba 3-1....98
5.2.8 - Prueba 3-2....99
5.3 – Evaluaciones en Wikipedia....100
5.3.1 - Elección de los pares de vértices a utilizar....101
5.3.2 - Prueba 4....103
5.3.2 – Prueba 5-1....104
5.3.3 - Prueba 5-2....105
5.3.3 - Prueba 5-3....106
5.3.4 – Prueba 6-1....106
5.3.5 - Prueba 6-2....107
5.4 - Conclusiones del capítulo....108

Este capítulo permitirá analizar los resultados obtenidos a partir de distintas pruebas realizadas con el algoritmo. Se inicia con un análisis acerca de los resultados del algoritmo en una Wiki pequeña como es Wikiquotes, para luego analizar el algoritmo en una Wiki de mayor envergadura como es Wikipedia. Se analizaran distintas estadísticas con el fin de evaluar el funcionamiento del propio algoritmo, evaluando las ventajas y desventajas del mismo. Por último, se detallarán las conclusiones de este capítulo.

5.1 - Aspectos generales de la evaluación de resultados

Todas las evaluaciones de este capítulo fueron realizadas en un entorno Cloud Computing como Amazon Web Services, específicamente a través de un servicio provisto

por esta plataforma denominado Elastic Map Reduce. Para poder usar este entorno, se desarrollaron scripts que configuren correctamente Giraph, los cuales no son explicados aquí dado que no tienen que ver directamente con el objetivo de este trabajo, sino con la consecuencia de usar Giraph en un entorno que no posee soporte nativo para su uso.

Respecto de las wikis elegidas para ser las evaluaciones de ambas versiones del algoritmo desarrollados en este informe, se debe remarcar que se usó la version en español, tanto de Wikiquotes como de Wikipedia. El uso de otras versiones de estas wikis como por ejemplo la versión en inglés, debe estar sujeta a nuevas evaluaciones del algoritmo, dado por ejemplo que las versiones en inglés de ambas wikis son superiores en tamaño a sus contrapartes en español, lo cual podría llevar a determinar que se deben realizar optimizaciones extras al algoritmo que no pueden ser predeterminadas bajo las pruebas realizadas en este informe.

Cada prueba tiene un identificador único, que permite en todo momento saber de qué prueba se está hablando.

A continuación, se detallan las distintas variables que se usarán para evaluar los resultados.

5.1.1 - Variables para la evaluación

Para todas las pruebas realizadas en este capítulo, se usarán las siguientes variables de evaluación:

- Cantidad de vértices destino: 1,50,100.
- Cantidad de Workers: 4,7,10.
- Longitud máxima de caminos navegacionales: 3,4,5.
- Tipos de Máquina Virtual utilizadas y características técnicas de las mismas[77]

Maquina virtual	Cantidad de vCPU	Cantidad de memoria disponible (GB)
m3.xlarge	4	15
m3.2xlarge	8	30

A continuación, se establecen qué métricas se usarán al momento de evaluar los resultados obtenidos en este capítulo.

77 Especificaciones de Máquinas Virtuales, Amazon Web Services, 15/02/2017 <https://aws.amazon.com/ec2/instance-types/>

5.1.2 - Métricas usadas, en qué consisten y cómo fueron calculadas

Se usarán las siguientes métricas para evaluar los resultados obtenidos:

- **Duración completa de la prueba**: Este es el tiempo total que conlleva la totalidad de las aplicaciones en Giraph. Se calcula como la diferencia entre la fecha de inicio de la primer aplicación Giraph ejecutada y la fecha de fin de la última aplicación Giraph ejecutada.
- **Cantidad de caminos navegacionales generados**: Es la cantidad total de caminos navegacionales encontrados en una prueba en particular. Cada una de las pruebas contiene un archivo con todos los caminos navegacionales obtenidos en dicha prueba, y ese archivo es al que se referirá esta métrica.
- **Promedio de caminos generados por aplicación exitosa**: Es la relación entre la métrica anterior y la cantidad de aplicaciones exitosas en una prueba particular. Sería deseable que en promedio se genere uno o más caminos navegacionales por aplicación exitosa, de otra forma la cantidad de caminos generados sera muy baja y por lo tanto el algoritmo no habrá servido de mucho.
- **Cantidad mínima de caminos generados por aplicación exitosa**: Cada aplicación exitosa generará 0 o más caminos navegacionales. Para determinar esta métrica, se analizarán todas las aplicaciones exitosas, y se determinará en cual de ellas se dió la menor cantidad de caminos navegacionales generados. En general es 0.
- **Cantidad máxima de caminos generados por aplicación exitosa**: Ídem la métrica anterior, salvo que en este caso se buscara la cantidad máxima de caminos navegacionales generados entre todas las aplicaciones exitosas.
- **Cantidad de aplicaciones exitosas**: Es la cantidad de aplicaciones Giraph que terminan con estado SUCCEDED. Esto quiere decir que dichas aplicaciones fueron ejecutadas exitosamente en el cluster correspondiente.
- **Cantidad de aplicaciones fallidas**: Es la cantidad de aplicaciones Giraph que terminan con estado FAILED o KILLED. Esto quiere decir que dichas aplicaciones no pudieron ser ejecutadas correctamente, significando que se detuvieron por algún error (FAILED) o fueron interrumpidas (KILED) para continuar con las otras aplicaciones Giraph y no desperdiciar poder de cómputo.
- **Promedio de tiempo de ejecución de aplicaciones exitosas**: Es una relación entre la sumatoria del tiempo de ejecución de aplicaciones exitosas contra la

cantidad de aplicaciones exitosas. Es deseable que este promedio se mantenga entre 30 y 50 segundos, los cuales son tiempos bajos en función de todas las pruebas que se realizaron para hacer este informe final.

- **Tiempo de ejecución combinado de aplicaciones exitosas y fallidas**: Sumatoria entre el tiempo que fue necesario para ejecutar las distintas aplicaciones Giraph.
- **Sobrecarga debido a tiempos externos a las aplicaciones Giraph en si**: Es la diferencia entre el tiempo de ejecución específico de las aplicaciones, y el tiempo total necesario para ejecutar una prueba en particular. Es importante remarcar este tiempo, dado que Giraph no calcula el tiempo de una aplicación desde que mandamos a ejecutar una aplicación en el cluster, sino que le lleva cierto tiempo iniciar una aplicación. Este es el tiempo que representa ese tiempo extra necesario que, de otra forma, quedaría por fuera del análisis.

A continuación se detallan las pruebas realizadas en Wikiquotes.

5.2 - Evaluaciones en Wikiquotes

El primer conjunto de evaluaciones se realizará sobre la wiki Wikiquotes, en su versión en español. Para estas evaluaciones se uso una copia offline de Wikiquotes de Junio del 2016[78]. Este dataset tiene 17357 vértices, los cuales tienen 90144 aristas en total. Dividiendo la cantidad de aristas total por la cantidad de vértices, cada vértice tiene un promedio de 5,19 aristas. El lector debe recordar que cada arista puede tener como destino un Artículo o una Categoría de Wikiquotes, dado que se tratan de la misma forma, es decir, ambos representan un vértice.

Los pares de vértices usados para el desarrollo de las pruebas fueron elegidos al azar. Se tomaron 10 vértices en forma aleatoria, y a cada uno de estos vértices se los escogió como vértice de inicio, luego se escogieron 100 vértices aleatorios adicionales, y se los seleccionó como vértice de destino de cada uno de los vértices de inicio escogidos previamente, consiguiendo 1000 pares de vértices para realizar este conjunto de pruebas en Wikiquotes.

Las evaluaciones realizadas para el dataset Wikiquotes se pueden resumir en la Tabla 1:

78 Dump de Wikiquotes de Junio del 2016, 15/07/2016 <https://dumps.wikimedia.org/eswikiquote/20160601/eswikiquote-20160601-pages-articles-multistream.xml.bz2>

Id	# Vértices destino	Workers	Máquina virtual	Longitud máxima del camino
0	1	4	m3.xlarge	5
1-1	50	4	m3.xlarge	5
1-2	100	4	m3.xlarge	5
2-1	100	4	m3.xlarge	7
2-2	100	7	m3.xlarge	7
3-1	100	4	m3.2xlarge	7
3-2	100	7	m3.2xlarge	7

Tabla 1: Pruebas realizadas con el grafo de Wikiquotes

5.2.1 - Elección de los pares de vértices a utilizar

Para la elección de los pares de vértices a utilizar en este conjunto de evaluaciones, se seleccionaron 100 vertices al azar, y

A continuación, se detallarán los resultados obtenidos en cada una de las pruebas, conjuntamente con el análisis que corresponda al caso.

5.2.2 - Prueba 0

Esta prueba se realizó con la versión del algoritmo de **un** vértice de inicio y **un** destino. Se utilizaron 4 workers m3.xlarge, con una longitud de caminos navegacionales máxima de 5. Los resultados fueron los siguientes:

- Duración completa de la prueba: 9 horas, 9 minutos y 47 segundos
- Cantidad de caminos navegacionales generados: 89
- Promedio de caminos generados por aplicación exitosa: 0,0954
- Cantidad mínima de caminos generados por aplicación exitosa: 0

- Cantidad máxima de caminos generados por aplicación exitosa: 26
- Cantidad de aplicaciones exitosas: 932
- Cantidad de aplicaciones fallidas: 1
- Promedio de tiempo de ejecución de aplicaciones exitosas: 33.77 segundos
- Tiempo de ejecución combinado de aplicaciones exitosas y fallidas: 8 horas y 45 minutos
- Sobrecarga debido a tiempos externos a las aplicaciones Giraph en si: 24 minutos con 47 segundos

Se puede observar que, aunque cada búsqueda de caminos navegacionales tarda en promedio 33.77 segundos, el tiempo total de realizar las búsquedas es de 9 horas, 9 minutos y 47 segundos.

Al procesar un par de vértices de inicio-destino por aplicación, se tendrá un tiempo significativamente mayor en total, que si se procesaran varios a la vez, como sucede en la Prueba 1-1, explicada a continuación..

5.2.3 - Prueba 1-1

Esta prueba se realizó con la versión del algoritmo de **un** vértice de inicio y **múltiples** destinos, en este caso particular se eligió que la cantidad de vértices destino sea 50. Se utilizaron 4 workers m3.xlarge, con una longitud de caminos navegacionales máxima de 5.

- Duración completa de la prueba: 11 minutos, 2 segundos
- Cantidad de caminos navegacionales generados: 3540
- Promedio de caminos generados por aplicación exitosa: 177
- Cantidad mínima de caminos generados por aplicación exitosa: 0
- Cantidad máxima de caminos generados por aplicación exitosa: 1308
- Cantidad de aplicaciones exitosas: 20
- Cantidad de aplicaciones fallidas: 0
- Promedio de tiempo de ejecución de aplicaciones exitosas: 30,85 segundos
- Tiempo de ejecución combinado de aplicaciones exitosas y fallidas: 10 minutos y 17 segundos

- Sobrecarga debido a tiempos externos a las aplicaciones Giraph en si: 45 segundos

Esta prueba, así como todas las restantes que se realizaron con Wikiquotes, se ejecutó con la version de un inicio y múltiples destinos, eligiendo en 50 la cantidad de destinos utilizados por aplicación Giraph, lo cual quiere decir que los pares de vértices fueron procesados de a 50 a la vez. El lector debe notar que el tiempo promedio de 30,85 es muy parecido al tiempo promedio de la Prueba 0, la cual nos daba un promedio de 33,77 por aplicación.

La cantidad de caminos navegacionales se esperaba que no sufriera modificaciones sustanciales, aunque finalmente sí las tuvo, en la Prueba 0 se encontraron 89 caminos navegacionales, en cambio en esta prueba se encontraron 3540. Se controló que la diferencia fue por caminos que en la primer versión del algoritmo no estaban siendo encontrados, dado que los caminos existentes en la Prueba 0 son también encontrados en esta solución, pero a su vez se encontraron múltiples caminos que no habían sido encontrados en la Prueba 0. Esto demuestra que la primer versión del algoritmo, además de tener la importante particularidad de tardar más tiempo, también generaba caminos en forma defectuosa, algo que es bastante probable dado que al seguir desarrollando el algoritmo se fueron solucionando pequeños problemas que contribuyeron a la mejor calidad de resultados. Con esto se prueba que la última versión del algoritmo es capaz de reemplazar en forma fiable a la versión anterior.

5.2.4 - Prueba 1-2

Esta prueba se realizó con la **versión del algoritmo de un vértice de inicio y múltiples destinos**, en este caso particular se eligió que la **cantidad de vértices destino sea 100**. Se utilizaron **4 workers m3.xlarge**, con una **longitud de caminos navegacionales máxima de 5**.

- Duración completa de la prueba: 5 minutos con 32 segundos
- Cantidad de caminos navegacionales generados: 3540 (ídem Prueba 1-1)
- Promedio de caminos generados por aplicación exitosa: 177 (Ídem Prueba 1-1)
- Cantidad mínima de caminos generados por aplicación exitosa: 0 (Ídem Prueba 1-1)
- Cantidad máxima de caminos generados por aplicación exitosa: 1308 (Ídem Prueba 1-1)

- Cantidad de aplicaciones exitosas: 10
- Cantidad de aplicaciones fallidas: 0
- Promedio de tiempo de ejecución de aplicaciones exitosas: 30,5 segundos
- Tiempo de ejecución combinado de aplicaciones exitosas y fallidas: 5 minutos con 2 segundos.
- Sobrecarga debido a tiempos externos a las aplicaciones Giraph en si: 27 segundos

Esta prueba es idéntica a la Prueba 1-1, pero con el detalle de que la cantidad de destinos elegidos para esta prueba fue 100, es decir, hay 50 destinos adicionales en cada búsqueda de caminos realizada en esta prueba, con respecto a la Prueba 1-1. Se debe notar que esta diferencia no incrementó la cantidad de aplicaciones fallidas, dado que sigue siendo 0 al igual que en la Prueba 1-1, y mantuvo el tiempo de ejecución de una aplicación promedio exitosa, dado que la prueba 1-1 tenía un promedio de 30,85 segundos y en esta prueba se obtuvo un tiempo promedio de 30,5 segundos. Se logró acortar el tiempo de ejecución de aplicaciones a la mitad en comparación, dado que con la Prueba 1-1 la duración completa de la prueba fue de 11 con 2 segundos, cuando en esta prueba el tiempo de la duración completa fue de 5 minutos con 32 segundos (prácticamente la mitad).

Los caminos navegacionales resultantes encontrados son exactamente los mismos, comparando esta prueba con la Prueba 1-1, lo que nos asegura que el incrementar la cantidad de destinos para la búsqueda de caminos navegacionales no influye en los resultados obtenidos y por lo tanto se lo considera seguro.

5.2.5 - Prueba 2-1

Esta prueba se realizó con la **versión del algoritmo de un vértice de inicio y múltiples destinos**, en este caso particular se eligió que la **cantidad de vértices destino sea 100**. Se utilizaron **4 workers m3.xlarge**, con una **longitud de caminos navegacionales máxima de 7**.

- Duración completa de la prueba: 10 minutos, 13 segundos
- Cantidad de caminos navegacionales generados: 2714
- Promedio de caminos generados por aplicación exitosa: 339,25
- Cantidad mínima de caminos generados por aplicación exitosa: 0

- Cantidad máxima de caminos generados por aplicación exitosa: 1308
- Cantidad de aplicaciones exitosas: 8
- Cantidad de aplicaciones fallidas: 2
- Promedio de tiempo de ejecución de aplicaciones exitosas: 31,62 segundos
- Tiempo de ejecución combinado de aplicaciones exitosas y fallidas: 8 minutos, 28 segundos
- Sobrecarga debido a tiempos externos a las aplicaciones Giraph en si: 1 minuto, 45 segundos

Esta prueba es idéntica a la Prueba 1-2, salvo por la diferencia en la longitud máxima de los caminos buscados. En esta prueba la longitud máxima se incrementa a 7, longitud para la cual no había sido probado el algoritmo. Al usar esta longitud, vemos que el algoritmo falla dos aplicaciones de un total de 10. Esto se debe a que la cantidad extra de mensajes generados en la búsqueda de caminos, termina haciendo que la aplicación falle en su totalidad, al quedarse sin memoria física disponible. Esto nos permite probar en forma empírica[79] cuales son los límites del algoritmo.

5.2.6 - Prueba 2-2

Esta prueba se realizó con la **versión del algoritmo de un vértice de inicio y múltiples destinos**, en este caso particular se eligió que la **cantidad de vértices destino sea 100**. Se utilizaron 7 **workers m3.xlarge**, con una **longitud de caminos navegacionales máxima de 7**.

- Duración completa de la prueba: 8 minutos, 31 segundos
- Cantidad de caminos navegacionales generados: 1
- Promedio de caminos generados por aplicación exitosa: 0,125
- Cantidad mínima de caminos generados por aplicación exitosa: 0
- Cantidad máxima de caminos generados por aplicación exitosa: 1
- Cantidad de aplicaciones exitosas: 8
- Cantidad de aplicaciones fallidas: 2
- Promedio de tiempo de ejecución de aplicaciones exitosas: 31,87 segundos

79 Es importante la diversidad de las pruebas, dado que las aplicaciones Giraph suelen probarse con prueba y error hasta que se va refinando su funcionamiento en distintas situaciones.

- Tiempo de ejecución combinado de aplicaciones exitosas y fallidas: 8 minutos, 18 segundos
- Sobrecarga debido a tiempos externos a las aplicaciones Giraph en sí: 13 segundos

Prueba idéntica a la Prueba 2-1, con la diferencia de que se usan 7 workers en lugar de 4. Esto significa que se buscan los mismos caminos y de la misma forma que en la prueba anterior, salvo con el poder de cómputo adicional que agrega el uso de 3 máquinas virtuales adicionales. Aunque esto era esperable que solucione el problema de la generación de mensajes extra en el uso de este algoritmo bajo una longitud máxima de 7, es decir, el problema de la prueba anterior, el lector puede observar que sigue habiendo 2 aplicaciones que han fallado, las cuales son las mismas que ya fallaron previamente. Esto implica que no necesariamente haciendo más grande un cluster, se va a obtener una mejora directa en los resultados del algoritmo.

Se puede observar que no se obtienen mejoras significativas en las métricas usadas, por ejemplo el promedio de ejecución de aplicaciones exitosas en esta prueba es de 31.87 segundos, contra los 31,62 de la prueba anterior. La duración completa de la prueba tiene una diferencia sustancial de dos minutos, pero esto es exclusivamente debido a que se tardó se les asignó más tiempo a las aplicaciones que fallaron para que intenten terminar correctamente. El lector de este trabajo no debería interpretar esto como una métrica negativa de esta prueba.

5.2.7 - Prueba 3-1

Esta prueba se realizó con la **versión del algoritmo de un vértice de inicio y múltiples destinos**, en este caso particular se eligió que la **cantidad de vértices destino sea 100**. Se utilizaron 4 **workers m3.2xlarge**, con una **longitud de caminos navegacionales máxima de 7**.

- Duración completa de la prueba: 9 minutos con 53 segundos
- Cantidad de caminos navegacionales generados: 1
- Promedio de caminos generados por aplicación exitosa: 0,125
- Cantidad mínima de caminos generados por aplicación exitosa: 0
- Cantidad máxima de caminos generados por aplicación exitosa: 1
- Cantidad de aplicaciones exitosas: 8

- Cantidad de aplicaciones fallidas: 2
- Promedio de tiempo de ejecución de aplicaciones exitosas: 29 segundos
- Tiempo de ejecución combinado de aplicaciones exitosas y fallidas: 9 minutos, 28 segundos
- Sobrecarga debido a tiempos externos a las aplicaciones Giraph en si: 25 segundos

Prueba idéntica a Prueba 2-1, salvo que se utilizan máquinas m3.2xlarge, lo que significa que se dispone casi del doble de memoria RAM en cada uno de los workers. A pesar de la mayor disponibilidad de RAM, esto no permitió solucionar el problema de fondo, y terminaron fallando las mismas aplicaciones que fallaron en la Prueba 2-1. Ya en este punto de las evaluaciones podemos observar un patrón, que consiste en que, al haber una cantidad de mensajes entre los vértices importante, las máquinas que integran el cluster se quedan sin memoria, y la aplicación en su totalidad falla.

5.2.8 - Prueba 3-2

Esta prueba se realizó con la **versión del algoritmo de un vértice de inicio y múltiples destinos**, en este caso particular se eligió que la **cantidad de vértices destino sea 100**. Se utilizaron 7 **workers m3.2xlarge**, con una **longitud de caminos navegacionales máxima de 7**.

- Duración completa de la prueba: 7 minutos, 15 segundos
- Cantidad de caminos navegacionales generados: 1
- Promedio de caminos generados por aplicación exitosa: 0,125
- Cantidad mínima de caminos generados por aplicación exitosa: 0
- Cantidad máxima de caminos generados por aplicación exitosa: 1
- Cantidad de aplicaciones exitosas: 8
- Cantidad de aplicaciones fallidas: 2
- Promedio de tiempo de ejecución de aplicaciones exitosas: 30,125 segundos
- Tiempo de ejecución combinado de aplicaciones exitosas y fallidas: 6 minutos, 34 segundos
- Sobrecarga debido a tiempos externos a las aplicaciones Giraph en si: 41 segundos

Prueba idéntica a Prueba 3-1, salvo que se usaron 7 workers m3.2xlarge en vez de 4. Los resultados fueron muy similares, fallaron la misma cantidad de aplicaciones, el promedio de ejecución de aplicaciones exitosas es similar entre ambas pruebas, etc. Se vuelve a demostrar que solo aumentando recursos a nivel de hardware no necesariamente se consiguen mejores resultados en las evaluaciones realizadas.

5.3 – Evaluaciones en Wikipedia

El segundo y último conjunto de evaluaciones se realizará sobre Wikipedia, en su versión en español. Para estas evaluaciones se usó una copia offline de Wikipedia de Junio del 2016[80]. Este dataset tiene 3152922 vértices, los cuales tienen 40106890 aristas en total. Dividiendo la cantidad de aristas total por la cantidad de vértices, cada vértice tiene un promedio de 12,72 aristas. El lector debe recordar que cada arista puede tener como destino un Artículo o una Categoría de Wikipedia, dado que se tratan de la misma forma, es decir, ambos representan un vértice.

Para entender la envergadura de la prueba, se debe recordar que en Wikiquotes se disponían de 17357 vértices y 90144 aristas, esto quiere decir que al usar Wikipedia se está probando ambas versiones del algoritmo con un grafo que **es 180 veces más grande que Wikiquotes**.

Debido a el tamaño del dataset con el que se va a probar, se debió reducir la longitud máxima de los caminos navegacionales buscados, en este conjunto de pruebas se usará el valor 5 como máximo. También se redujo el nivel de log al mínimo, dado que el nivel estándar de logging de Giraph (INFO) incrementaba sustancialmente los tiempos de ejecución de las aplicaciones, por lo que se lo redujo al nivel más bajo (ERROR). También se incrementó hasta 10 la cantidad de workers, dado que el aumento en la cantidad de vértices y la cantidad adicional de mensajes a enviar hacía que un cluster de 4 workers quede sobreexigido.

La cantidad de vértices destino elegida en la segunda versión del algoritmo fue de 50, dado que al analizar los pares de vértices a utilizar en este conjunto de evaluaciones, el 99% de los vértice de destino, al contarlos agrupados por el vértice de inicio, no llegaban a superar ese valor, por lo que utilizar un valor mayor no hubiera traído aparejado modificaciones importantes en los resultados, como sí sucedía en Wikiquotes.

80 Dump de Wikipedia de Junio del 2016, 15/07/2016 <https://dumps.wikimedia.org/eswiki/20160601/eswiki-20160601-pages-articles-multistream.xml.bz2>

Las evaluaciones realizadas para el dataset Wikipedia se pueden resumir en la Tabla 2:

Id	# Vértices destino	Workers	Máquina virtual	Longitud máxima del camino
4	1	10	m3.xlarge	3
5-1	50	10	m3.xlarge	3
5-2	50	10	m3.xlarge	4
5-3	50	10	m3.xlarge	5
6-1	50	10	m3.2xlarge	4
6-2	50	10	m3.2xlarge	5

Tabla 2: Pruebas realizadas con el grafo de Wikipedia

5.3.1 - Elección de los pares de vértices a utilizar

Para la elección de los pares de vértices a utilizar en esta prueba, se escogieron 4 propiedades semánticas distintas de DBPedia: birthplace, deathplace, party, recordLabel. A través de estas propiedades, se construyeron 4 consultas SPARQL distintas:

- Birthplace
  - select ?from ?to where
    {?wto a <http://dbpedia.org/ontology/Person>. ?wto
    <http://dbpedia.org/property/birthPlace> ?wfrom. ?wfrom a
    <http://dbpedia.org/ontology/Place>. ?wto
    <http://www.w3.org/2002/07/owl#sameAs> ?to. ?wfrom
    <http://www.w3.org/2002/07/owl#sameAs> ?from.
    filter regex(str(?from), "es.dbpedia.org")
    filter regex(str(?to), "es.dbpedia.org")}
- Deathplace

- select ?from ?to
  where {?wto a <http://dbpedia.org/ontology/Person>. ?wto <http://dbpedia.org/property/deathPlace> ?wfrom. ?wfrom a <http://dbpedia.org/ontology/Place>. ?wto <http://www.w3.org/2002/07/owl#sameAs> ?to. ?wfrom <http://www.w3.org/2002/07/owl#sameAs> ?from. filter regex(str(?from), "es.dbpedia.org") filter regex(str(?to), "es.dbpedia.org")}

- Party
  - select ?from ?to
    where {?wto a <http://dbpedia.org/ontology/Person>. ?wto <http://dbpedia.org/property/party> ?wfrom. ?wfrom a <http://dbpedia.org/ontology/PoliticalParty>. ?wto <http://www.w3.org/2002/07/owl#sameAs> ?to. ?wfrom <http://www.w3.org/2002/07/owl#sameAs> ?from. filter regex(str(?from), "es.dbpedia.org") filter regex(str(?to), "es.dbpedia.org")}
- recordLabel
  - select ?wfrom ?wto
    where {
    ?wto a <http://dbpedia.org/ontology/Person>.
    ?wto <http://dbpedia.org/ontology/recordLabel> ?wfrom.
    ?wfrom a <http://dbpedia.org/ontology/Company>.
    ?wto <http://www.w3.org/2002/07/owl#sameAs> ?to. ?wfrom <http://www.w3.org/2002/07/owl#sameAs> ?from. filter regex(str(?from), "es.dbpedia.org") filter regex(str(?to), "es.dbpedia.org") }

Estas consultas fueron ejecutadas en el endpoint SPARQL correspondiente a DBPedia en español[81]. Cada una de estas consultas generó un máximo de 10000

81 SPARQL endpoint de DBPedia en español, 05/03/2017, <http://es.dbpedia.org/sparql>

resultados. Una vez ejecutadas las consultas, se guardaron los resultados de cada una como archivos separados en forma local. Se eligió para la descarga de estos archivos el formato CSV. Estos archivos CSV fueron adaptados al formato que necesita el algoritmo de búsqueda de caminos navegacionales a través del script **armar-dataset-wikipedia.sh**[82], el cual esta en el DVD que se presenta junto a esta tesina. Este script se encarga de preparar un archivo del formato previamente especificado, para que pueda ser consumido por el algoritmo, y de esta forma, el algoritmo obtenga desde que pares de vértices debe buscar caminos navegacionales.

Al usar pares de vértices obtenidos desde DBPedia en vez de usar vértices al azar, adaptamos las pruebas para que produzcan resultados de acuerdo a lo que el algoritmo BlueFinder necesita.

5.3.2 - Prueba 4

Esta prueba se realizó con la **versión del algoritmo de un vértice de inicio y un destino**. Se utilizaron 10 **workers m3.xlarge**, con una **longitud de caminos navegacionales máxima de 3**.

- Duración completa de la prueba: 49 horas, 34 minutos
- Cantidad de caminos navegacionales generados: 5647
- Promedio de caminos generados por aplicación exitosa: 1,44 caminos
- Cantidad mínima de caminos generados por aplicación exitosa: 0
- Cantidad máxima de caminos generados por aplicación exitosa: 37
- Cantidad de aplicaciones exitosas: 3963
- Cantidad de aplicaciones fallidas: 0
- Promedio de tiempo de ejecución de aplicaciones exitosas: 43 segundos
- Tiempo de ejecución combinado de aplicaciones exitosas y fallidas: 47 horas, 56 minutos
- Sobrecarga debido a tiempos externos a las aplicaciones Giraph en si: 1 hora, 38 minutos

Al igual que en la Prueba 0, podemos ver en funcionamiento la primer versión del algoritmo, ahora con el grafo de Wikipedia. Se puede observar que, al realizar búsquedas

82 Ver apéndice.

de caminos de longitud 3 como máximo, el tiempo de ejecución promedio de aplicaciones exitosas es 43 segundos, lo cual es 10 segundos por encima del promedio obtenido en la Prueba 0, aunque esto es un tiempo esperable. Es importante remarcar que en esta primer prueba, hay 0 aplicaciones fallidas, lo que quiere decir que con caminos de longitud 3 como máximo el algoritmo, en su primera versión, funciona correctamente.

5.3.2 – Prueba 5-1

Esta prueba se realizó con la **versión del algoritmo de un vértice de inicio y múltiples destinos**. Se eligió que la cantidad de vértices destinos sea 50. Se utilizaron 10 **workers m3.xlarge**, con una **longitud de caminos navegacionales máxima de 3**.

- Duración completa de la prueba: 10 horas, 24 minutos, 28 segundos
- Cantidad de caminos navegacionales generados: 141052
- Promedio de caminos generados por aplicación exitosa: 159,2 caminos
- Cantidad mínima de caminos generados por aplicación exitosa: 0
- Cantidad máxima de caminos generados por aplicación exitosa: 2102
- Cantidad de aplicaciones exitosas: 886
- Cantidad de aplicaciones fallidas: 0
- Promedio de tiempo de ejecución de aplicaciones exitosas: 40,69 segundos
- Tiempo de ejecución combinado de aplicaciones exitosas y fallidas: 10 horas, 1 minuto
- Sobrecarga debido a tiempos externos a las aplicaciones Giraph en si: 23 minutos, 28 segundos

En esta prueba podemos reconocer, al igual como sucedió en la Prueba 1-1 al compararla con Prueba 0, la reducción sustancial en el tiempo de duración total de la prueba si se la compara con la Prueba 4. Se puede notar que esta prueba tardó 10 horas y 24 minutos, cuando la Prueba 4 tardó 49 horas y 34 minutos. Este buen desempeño se produjo incluso bajando el tiempo promedio de ejecución de aplicaciones exitosas en casi 3 segundos, y sin una sola aplicación fallida.

Se puede observar que la cantidad de caminos encontrados en esta prueba son más del doble que los encontrados en la Prueba 4, lo cual sucede, como se comentó en la

Prueba 5-1, a defectos de funcionamiento de la primera versión del algoritmo, que fueron subsanados en la última versión del mismo

.

5.3.3 - Prueba 5-2

Esta prueba se realizó con la **versión del algoritmo de un vértice de inicio y múltiples destinos**. Se eligió que la cantidad de vértices destinos sea 50. Se utilizaron 10 **workers m3.xlarge**, con una **longitud de caminos navegacionales máxima de 4**.

- Duración completa de la prueba: 20 horas, 28 minutos, 2 segundos
- Cantidad de caminos navegacionales generados: 1192156
- Promedio de caminos generados por aplicación exitosa: 1938,46
- Cantidad mínima de caminos generados por aplicación exitosa: 0
- Cantidad máxima de caminos generados por aplicación exitosa: 25546
- Cantidad de aplicaciones exitosas: 615
- Cantidad de aplicaciones fallidas: 271
- Promedio de tiempo de ejecución de aplicaciones exitosas: 48,29 segundos
- Tiempo de ejecución combinado de aplicaciones exitosas y fallidas: 20 horas, 1 minuto, 10 segundos
- Sobrecarga debido a tiempos externos a las aplicaciones Giraph en si: 26 minutos, 52 segundos

En esta prueba se puede observar que hay un incremento de la duración completa de la prueba en casi el doble al compararla con la Prueba 5-1. Esto se debe principalmente en la aparición de múltiples aplicaciones fallidas en una cantidad proporcionalmente muy superior al 20% máximo que había en Wikiquotes y llegando a casi el 31% del total de aplicaciones. Dichas aplicaciones fallidas incrementan sustancialmente el tiempo total, dado que para determinar si se debe interrumpir una aplicación por considerarla defectuosa, se debe esperar un tiempo promedio que es sustancialmente mayor al que tarda una aplicación exitosa.

Por otro lado, se rescata como positivo el incremento en casi 10 veces en la cantidad de caminos navegacionales obtenidos en esta prueba en comparación con la Prueba 4, aún a pesar de la cantidad de aplicaciones fallidas.

5.3.3 - Prueba 5-3

Esta prueba se realizó con la **versión del algoritmo de un vértice de inicio y múltiples destinos**. Se eligió que la cantidad de vértices destinos sea 50. Se utilizaron 10 **workers m3.xlarge**, con una **longitud de caminos navegacionales máxima de 5**.

- Duración completa de la prueba: 1 día, 4 horas, 9 minutos, 50 segundos
- Cantidad de caminos navegacionales generados: 985 caminos
- Promedio de caminos generados por aplicación exitosa: 75,76 caminos
- Cantidad mínima de caminos generados por aplicación exitosa: 0
- Cantidad máxima de caminos generados por aplicación exitosa: 739
- Cantidad de aplicaciones exitosas: 13
- Cantidad de aplicaciones fallidas: 873
- Promedio de tiempo de ejecución de aplicaciones exitosas: 41 segundos
- Tiempo de ejecución combinado de aplicaciones exitosas y fallidas: 1 día, 3 horas, 45 minutos, 40 segundos
- Sobrecarga debido a tiempos externos a las aplicaciones Giraph en si: 24 minutos, 10 segundos

En esta prueba se puede ver el impacto de incrementar la longitud máxima de caminos a buscar hasta 5, dado que en la prueba anterior, es decir la Prueba 5-2, este máximo era de 4. La cantidad de aplicaciones fallidas se incrementó del 31% de la prueba anterior, hasta un 98%. Este incremento en la cantidad de aplicaciones fallidas hizo incrementar la duración total de la prueba en varias horas, dado que, como se explicó en la prueba anterior, cuando una aplicación no es exitosa, se debe esperar un tiempo mayor para interrumpirla que el tiempo que tarda una aplicación que si es exitosa.

5.3.4 – Prueba 6-1

Esta prueba se realizó con la **versión del algoritmo de un vértice de inicio y múltiples destinos**. Se eligió que la cantidad de vértices destinos sea 50. Se utilizaron 10 **workers m3.2xlarge**, con una **longitud de caminos navegacionales máxima de 4**.

- Duración completa de la prueba: 15 horas, 15 minutos, 19 segundos
- Cantidad de caminos navegacionales generados: 1762506
- Promedio de caminos generados por aplicación exitosa: 2.528,70
- Cantidad mínima de caminos generados por aplicación exitosa: 0
- Cantidad máxima de caminos generados por aplicación exitosa: 92580
- Cantidad de aplicaciones exitosas: 697
- Cantidad de aplicaciones fallidas: 189
- Promedio de tiempo de ejecución de aplicaciones exitosas: 46,57 segundos
- Tiempo de ejecución combinado de aplicaciones exitosas y fallidas: 14 horas, 52 minutos, 50 segundos
- Sobrecarga debido a tiempos externos a las aplicaciones Giraph en si: 22 minutos, 29 segundos

Esta prueba nos permite evaluar el algoritmo usando máquinas m3.2xlarge, las cuales tienen casi el doble de RAM que las utilizadas en las anteriores pruebas de Wikipedia.

En esta prueba podemos observar un ligero incremento en la cantidad de aplicaciones exitosas y un decremento en la cantidad de aplicaciones fallidas, en comparación con la otra prueba de caminos de longitud 4, es decir, la Prueba 5-2. La cantidad de aplicaciones fallidas bajó un 10%, dado que la cantidad de aplicaciones fallidas en la Prueba 5-2 era de 31% y en esta evaluación es del 21%. La cantidad de caminos navegacionales encontrados aumento, dado que, al haber más aplicaciones exitosas, surgieron nuevos caminos navegacionales. El tiempo de ejecución promedio de aplicaciones exitosas se mantuvo en un valor similar al de Prueba 5-2.

5.3.5 - Prueba 6-2

Esta prueba se realizó con la **versión del algoritmo de un vértice de inicio y múltiples destinos**. Se eligió que la cantidad de vértices destinos sea 50. Se utilizaron 10 **workers m3.2xlarge**, con una **longitud de caminos navegacionales máxima de 5**.

- Duración completa de la prueba: 1 día, 1 hora, 54 minutos, 10 segundos
- Cantidad de caminos navegacionales generados: 985

- Promedio de caminos generados por aplicación exitosa: 75,76 caminos
- Cantidad mínima de caminos generados por aplicación exitosa: 0
- Cantidad máxima de caminos generados por aplicación exitosa: 739
- Cantidad de aplicaciones exitosas: 13
- Cantidad de aplicaciones fallidas: 873
- Promedio de tiempo de ejecución de aplicaciones exitosas: 39,23 segundos
- Tiempo de ejecución combinado de aplicaciones exitosas y fallidas: 23 horas, 34 minutos, 14 segundos
- Sobrecarga debido a tiempos externos a las aplicaciones Giraph en si: 2 horas, 19 minutos, 56 segundos

Esta es la última evaluación de este informe. La diferencia con la anterior evaluación es que en esta prueba la longitud máxima de los caminos es 5. Algunos resultados son iguales en comparación a los de la Prueba 5-3, la cantidad de aplicaciones fallidas trepó al 98%, y la duración completa de la prueba fue de más de un día, coincidente con el tiempo de más de un día de la Prueba 5-3. Aunque parezcan muy negativos los resultados de la Prueba 5-3 y de esta prueba, nos permiten reafirmar cual es el punto débil del algoritmo, y como se debe mejorarlo en futuras continuaciones de este trabajo.

5.4 - Conclusiones del capítulo

Se probó el algoritmo en sus dos versiones, tanto con Wikiquotes como con Wikipedia y se documentaron los resultados. Se explicaron los resultados obtenidos, tanto los resultados exitosos como los no exitosos, planteando las bases para las futuras extensiones de este trabajo.

Capítulo 6 - Conclusiones y trabajos futuros

Índice de contenidos

Capítulo 6 - Conclusiones y trabajos futuros.....109
6.1 - Conclusiones de este informe final.....109
6.2 - Mejoras pendientes / Posibles extensiones del trabajo.....112
6.1.2 - Extensión para trabajar únicamente con vértices de origen, y abarcar todos los destinos.....112
6.1.3 - Extensión para poder procesar múltiples vértices de origen a la vez.....113
6.1.4 – Uso de un Combiner para reducir la cantidad de mensajes total a enviar entre vértices.....113
6.1.5 - Uso de Giraph 1.2.....113

6.1 - Conclusiones de este informe final

Este trabajo se inicio con un estudio de la tecnología Hadoop, la cual tiene como modelo de procesamiento a Map Reduce, dado que al ser este último el componente esencial de la arquitectura a utilizar, era necesario introducirse en este tema para poder empezar a hacer este trabajo. Se desarrollaron múltiples pruebas en forma local, experimentando con algoritmos como WordCount y PageRank, los cuales son algoritmos que ya vienen listos para ser probados en cualquier versión de Hadoop Map Reduce. Esto permitió conocer en profundidad aspectos de configuración de un cluster Hadoop y el funcionamiento del modelo de procesamiento Map Reduce..

Al haber entendido en forma básica el funcionamiento de Map Reduce y como construir un cluster para su ejecución, se estudiaron distintas alternativas para el procesamiento de grafos. Entre las distintas alternativas disponibles en el mercado, se eligió Giraph, y se inició la investigación de este framework, un framework desarrollado siguiendo el modelo de Pregel [3] y utilizado por grandes compañías a lo largo del mundo. Se adaptó el cluster local pre-existente para soportar la ejecución de aplicaciones Giraph, dado que, aunque un programa Giraph tiene una arquitectura Map Reduce subyacente, el soporte de Giraph no está dado en forma nativa en Hadoop. Se iniciaron las pruebas con el algoritmo Shortest Path, de forma de entender como Giraph se comporta al ejecutar aplicaciones.

Disponiendo de estas bases, se inició el desarrollo del algoritmo de búsqueda de caminos navegacionales, objetivo máximo de esta tesina. Se buscó si había algún algoritmo que cumpliera esta función, pero no se encontró uno similar. El más parecido fue un algoritmo desarrollado por Marco Lotz, que consistía en un BFS que se detenía al encontrar el nodo buscado. Lentamente, se lo fue adaptando, y se lo fue probando en un IDE, convirtiéndolo en un BFS que generara caminos navegacionales, entre un vértice de inicio y uno de destino, y bajo la premisa de que los caminos no excedan una longitud máxima. Esta fue la primer versión exitosa del algoritmo de búsqueda de caminos navegacionales desarrollado en esta tesina.

Se continuó mejorando el algoritmo, dado que se descubrió una forma importante de aumentar su performance, que fue usar un vértice de origen, pero usar múltiples destinos, con el fin de hacer varías búsquedas simultaneas entre ese mismo origen y los distintos destinos. De esta forma, se consiguió la segunda y definitiva versión de este algoritmo, la cual permite escoger un vértice de inicio, uno o varios vértices de destinos, y una longitud máxima de caminos navegacionales.

En paralelo a el desarrollo de este algoritmo, se desarrollo un programa para construir el grafo de entrada, el cual estuvo basado en Wikiquotes en un principio, y en Wikipedia al final, usando de ambas enciclopedias la versión en español. Este programa estuvo basado en un programa similar realizado por Mirko Nasato llamado Graphipedia. Este programa parte de una copia offline de Wikipedia/Wikiquotes, para generar un archivo txt con un formato que pueda ser leído correctamente por el algoritmo de búsqueda de caminos navegacionales.

Al empezar a tener resultados positivos con el algoritmo de búsqueda de caminos, se inicio con la investigación acerca de Amazon Web Services, plataforma de Cloud Computing escogida para la prueba de esta tesina, dado que tiene un poder de computo muy superior a cualquier cluster al que pueda tener acceso el autor de este trabajo. Se hicieron múltiples pruebas con el fin de entender como replicar la configuración del cluster local en un cluster creado en Amazon Web Services, y poco a poco se profundizó acerca de como poder configurar un cluster de mayor potencia.

Una vez habiendo finalizado de modificar el algoritmo, se diseño un conjunto de evaluaciones, para poder medir los distintos resultados arrojados por el algoritmo. Se hicieron evaluaciones tanto como en Wikipedia como en Wikiquotes, y bajo distintas métricas se evaluaron sus resultados, de ambas versiones de este algoritmo.

Por último, se desarrolló un programa que permite cargar los caminos navegacionales resultantes de la ejecución del algoritmo, en una base MySQL.

Resumiendo hasta este punto, se puede afirmar que este trabajo final obtuvo un índice de caminos navegacionales, usando una arquitectura Map Reduce en la nube (Cloud Computing). Para este fin se desarrolló un algoritmo BFS el cual fue diseñado con el propósito de ser ejecutado e a través del framework Giraph. Y los resultados de este algoritmo pueden ser accedidos a través de una base MySQL con el fin de facilitar la integración de este algoritmo con otros que usen sus resultados, como BlueFinder.

Fueron varios los escollos sorteados en la realización de este trabajo. Inicialmente, la bibliografía acerca del framework Giraph era escasa, y muy orientada a versiones antiguas de Hadoop. A lo largo de la realización del trabajo, fueron publicados dos libros distintos acerca de Giraph, los cuales son material de consulta permanente de la comunidad, uno de los cuales fue citado múltiples veces como referencia en este trabajo. Tampoco es tan útil la comunidad de este framework, dado que es algo pequeña y no muy activa.

Como critica de este trabajo se pueden mencionar que, bajo ciertas longitudes, el algoritmo falla. Se lo reviso exhaustivamente y se comprobó que esta importante falla se da en casos en que se generan muchos mensajes en muy poco tiempo, como por ejemplo, cuando un vértice tiene muchas aristas a otros artículos. Esto es debido a que, al querer enviar muchos mensajes, cada uno de los Workers de los cuales se compone una aplicación Giraph debe enviar muchos mensajes pequeños a otros workers. El problema es que al querer generar para poder enviar dichos mensajes, necesita crear múltiples conexiones distintas desde cada worker, y esto nos da como resultado que la cantidad de memoria se dispare y la aplicación falla, justamente por que se queda sin memoria. Al usar Wikiquotes como grafo, se pudo usar una longitud máxima de caminos navegacionales de 5 para hacer búsquedas de caminos navegacionales, aunque en Wikipedia la longitud máxima para hacer búsquedas sin problemas fue de 3. Se intento solucionar este problema con distintos configuraciones de parámetros que Giraph provee, pero no se obtuvo los resultados esperados y no fue posible obtener una configuración paramétrica que nos ayude con este problema. También se intento modificar el algoritmo para contemplar este caso a través de una compresión y descompresión manual de mensajes antes del envió y recibo de los mismos, respectivamente, pero no fue exitosa. En adición a estos intentos, se intentó conseguir con mayor poder de computo una solución a este problema, pero no fue posible, aunque se agregaran muchas más, las mejoras eran muy poco significativas.

También se puede criticar que el algoritmo no soporte cualquier tamaño de grafo por igual, aunque esto está fuertemente relacionado al manejo de “prueba y error” que se recomienda realizar con Giraph hasta obtener los resultados esperados.

Por último, hubiera sido interesante que la versión de un origen y varios destinos también funcione para varios orígenes con varios destinos, pero dado que esta versión generaba muchos más mensajes, dado que en lugar de hacer un BFS, realizaba múltiples BFS desde distintos vértices de origen, esto no fue posible. Para que esta versión funcione, primero debería corregirse la versión de un vértice de inicio y muchos destinos.

El principal problema que tiene este algoritmo es un problema que es bien conocido en la comunidad de Giraph, y por esta razón fue rediseñado en forma integral desde la versión Giraph 1.1 que se utilizó en este trabajo, a la versión 1.2 recientemente lanzada, a la que el autor de este trabajo no pudo tener acceso.

De todo esto, el lector podrá apreciar que se llego a hacer un algoritmo útil pero que necesita ser extendido con el fin de solucionar los problemas que el autor de este trabajo no pudo corregir, principalmente por falta de tiempo. Las principales extensiones posibles de este trabajo son detalladas en la siguiente sección.

6.2 - Mejoras pendientes / Posibles extensiones del trabajo

Esta sección se encarga de listas las distintas extensiones que podría tener el trabajo, en caso de que se extienda el mismo para mejorarlo. Las mejoras son listadas a continuación:

6.1.2 - Extensión para trabajar únicamente con vértices de origen, y abarcar todos los destinos

Una posibilidad de mejora de este trabajo, que se extrae de la posibilidad de manejar múltiples destinos para un mismo vértice de inicio, sería posibilitar que se busquen simultáneamente caminos para todos los destinos, es decir, que se busquen caminos hacia todos los vértices a los que el algoritmo tuvo alcance. Es decir, supongamos que se buscan caminos para los pares A-C y A-D. Esto quiere decir que se buscarán caminos entre los vértices A y C, y A y D, pero el algoritmo puede (y seguramente lo hará) pasar por múltiples vértices más antes de llegar a los vértices de destino C y D. Lo que esta extensión plantea

es calcular los caminos generados para cada uno de los vértices a partir de A, sin tener que especificar cuales son destinos, dado que todos los vértices por los que pasa el algoritmo se considerarían destinos. Esta mejora significaría un incremento sustancial de la cantidad de caminos generados, pero no sería difícil de de realizar.

6.1.3 - Extensión para poder procesar múltiples vértices de origen a la vez

Se descubrió un problema al intentar tener múltiples vértices de origen en lugar de uno solo, al buscar caminos entre un conjunto de vértices de destino y un conjunto de vértices de destino. Este problema radicaba en que un vértice terminaba siendo procesado desde dos lugares al mismo tiempo, lo cual no debería ser posible dado que Giraph se basa en el modelo teórico de Pregel, en el cual esto no puede suceder. Si se descubriera la razón de esta anomalía, se podría corregir el código del algoritmo para involucrar esta importante mejora. De todos modos, al permitir múltiples vértices de origen, esto incrementaría dramáticamente varias veces la cantidad total de mensajes enviados entre los vértices, lo cual traería aparejado una prueba exhaustiva para saber si este incremento en la cantidad de mensajes puede ser soportado por el algoritmo.

6.1.4 – Uso de un Combiner para reducir la cantidad de mensajes total a enviar entre vértices.

Se podría adaptar el funcionamiento para que comprima antes de enviar todos los mensajes en uno, y los descomprima al recibirlo, a través del uso de un Combiner o combinador, el cual es una forma que tiene Giraph de comprimir múltiples mensajes en uno, de forma que su envió y recepción se simplifique. Por ende, la cantidad de mensajes a enviar sería reducida drásticamente, y se podría conseguir una importante mejora en el funcionamiento del algoritmo, dado que se enviaría un solo mensaje grande en lugar de múltiples chicos, reduciendo la cantidad de conexiones entre workers drásticamente.

6.1.5 - Uso de Giraph 1.2

En octubre de 2016, es decir, cuando se estaban haciendo las pruebas más avanzadas a nivel de performance de este algoritmo, finalmente se lanzó la versión 1.2 de Giraph. Esta nueva versión del framework incluye mejoras en el manejo del envió de

múltiples millones de mensajes, dado que se redefinió por completo el manejo de mensajes cuando no hay memoria suficiente (out-of-core). Sería recomendable revisar la performance del algoritmo con esta nueva versión, dado que seguramente se tendrían mejoras significativas, principalmente la versión de múltiples destinos.

Glosario

API (Application Programming Interface)
La interfaz de programación de aplicaciones, abreviada como API del inglés: Application Programming Interface es el conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción (https://es.wikipedia.org/wiki/Interfaz_de_programación_de_aplicaciones)

Big Data
Big data es un concepto que hace referencia al almacenamiento de grandes cantidades de datos y a los procedimientos usados para encontrar patrones repetitivos dentro de esos datos.
(https://es.wikipedia.org/wiki/Big_data)

Caminos Navegacionales
Un camino navegacional puede ser definido como una sucesión de enlaces, que nos permiten llegar desde un artículo hasta otro (ver definición 3.1, pag 2, de [5]).

Cloud Computing
La computación en la nube, conocida también como servicios en la nube, informática en la nube, nube de cómputo o nube de conceptos (del inglés Cloud Computing), es un paradigma que permite ofrecer servicios de computación a través de una red, que usualmente es Internet.
(https://es.wikipedia.org/wiki/Computación_en_la_nube**)**

Endpoint
Punto de entrada a un servicio o proceso.

HDFS
Sigla que quiere decir “Hadoop Distributed File System”. Sistema de archivos distribuido, escalable y portátil escrito en Java para el framework Hadoop.
(https://es.wikipedia.org/wiki/Hadoop)

IDE
Un entorno de desarrollo integrado1 2 o entorno de desarrollo interactivo, en inglés Integrated Development Environment (IDE), es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software.
(https://es.wikipedia.org/wiki/Entorno_de_desarrollo_integrado)

Lenguaje de Marcas

Un lenguaje de marcado o lenguaje de marcas es una forma de codificar un documento que, junto con el texto, incorpora etiquetas o marcas que contienen información adicional acerca de la estructura del texto o su presentación.

Map, Reduce
Funciones de Programación funcional, utilizadas en Apache Hadoop Map Reduce.
SPARQL
SPARQL es un acrónimo recursivo del inglés SPARQL Protocol and RDF Query Language. Se trata de un lenguaje estandarizado para la consulta de grafos RDF, normalizado por el RDF Data Access Working Group (DAWG) del World Wide Web Consortium (W3C). Es una tecnología clave en el desarrollo de la Web Semántica.
(https://es.wikipedia.org/wiki/SPARQL)

Web Semántica, o Semantic Web
La web semántica (del inglés Semantic web) es un conjunto de actividades desarrolladas en el seno de World Wide Web Consortium con tendencia a la creación de tecnologías para publicar datos legibles por aplicaciones informáticas (máquinas en la terminología de la Web semántica).
(https://es.wikipedia.org/wiki/Web_semántica)

Web Social, o Social Web
La web social incluye servicios que determinan la tendencia en la forma de compartir información digital, en este caso es importante definir que internet, comprendería la red de redes que hacen posible que existan servicios como la web, web 2.0.
(https://es.wikipedia.org/wiki/Web_social)

YARN
Framework que nos posibilita la ejecución planificada de procesos y el control de los recursos de un cluster, dentro de la plataforma de software Hadoop. Surgió para separar el control de recursos de un cluster Hadoop, del modelo de procesamiento en el cual se escribían los programas.

Apéndice

Este apéndice consiste de distintos scripts utilizados durante el trabajo, y adjuntos tanto a BitBucket como al DVD anexo a este informe. Los mismos son explicados a continuación.

1 - Script guardar-logs-local.sh

- Nombre del script
  - guardar-logs-local.sh
- Ubicación
  - https://bitbucket.org/ambartsumian/archivos_configuracion_giraph/commits/1a40dc6f3ec724a01d3a4e986acb222c1fc692ff#chg-cluster-de-clusters/guardar-logs-local-useast.sh
  - https://bitbucket.org/ambartsumian/archivos_configuracion_giraph/commits/1a40dc6f3ec724a01d3a4e986acb222c1fc692ff#chg-cluster-de-clusters/guardar-logs-local-saopaulo.sh
- Función
  - Este script bash se encarga de guardar los resultados de la ejecución del algoritmo de búsqueda de caminos navegacionales, para un cluster AWS particular, y en un directorio especifico. Esto se logra conectándose via SSH a través del id del cluster, y recorriendo HDFS para recolectar los distintos caminos navegacionales generados por el algoritmo. Todos los caminos navegacionales generados por la ejecución del algoritmo son guardados en un archivo llamado "caminos-navegacionales-resultantes" el cual es almacenado en el directorio especificado en la invocación de este script.
- Invocación, con parámetros correspondientes:
  - guardar-logs-local <cluster-id> <directorio-para-guardar-resultados-por-cluster>
- Explicación de los parámetros
  - Cluster-id: Cluster id de AWS (cada cluster tiene un ID particular).
  - Directorio-para-guardar-resultados-por-cluster: Directorio en S3 donde se guardaran los resultados del cluster

- Resultado generado
  - Archivo con todos los caminos navegacionales resultantes, separando los mismos por el par de vértices para el que fueron generados.

2 - Script armar-dataset-wikipedia.sh

- Nombre del script
  - armar-dataset-wikipedia.sh
- Ubicación
  - https://bitbucket.org/ambartsumian/archivos_configuracion_giraph/commits/1a40dc6f3ec724a01d3a4e986acb222c1fc692ff#chg-cluster-de-clusters/guardar-logs-local-useast.sh
  - https://bitbucket.org/ambartsumian/archivos_configuracion_giraph/commits/1a40dc6f3ec724a01d3a4e986acb222c1fc692ff#chg-cluster-de-clusters/guardar-logs-local-saopaulo.sh
- Función
  - Este script bash se encarga de construir el archivo de pares de vértices que necesita el algoritmo de busqueda de caminos navegacionales para ser ejecutado correctamente.
- Invocación, con parámetros correspondientes:
  - armar-dataset-wikipedia.sh <archivo-con-consulta-sparql1> <archivo-con-consulta-sparql2> ...
- Explicación de los parámetros
  - Cada uno de los parametros es un archivo CSV resultante de ejecutar una consulta SPARQL en el endpoint es.dbpedia.org
- Resultado generado
  - Archivo con pares de vertices, listo para ser ejecutado en la busqueda de caminos navegacionales

3 – Script de construcción de base MySQL bluefinderdb.sql

- Nombre del script
  - bluefinderdb.sql
- Ubicación
  - URL Bitbuckett: https://bitbucket.org/ambartsumian/indice_bluefinder/src/8cb188717d716f80dcefdfcffe2e0006f0bfcd64/bluefinderdb.sql?at=master&fileviewer=file-view-default
  - Archivo en DVD tesina_cd/Algoritmos y scripts/Programa para convertir salida del algoritmo en una base MySQL
- Función
  - Este script SQL construye en forma integral la base de datos usada por el programa ConstruirIndiceMySQL.java, explicado en la Sección 4.4 de este informe
- Invocación, con parámetros correspondientes:
  - Importar en MySQL database de forma estándar
- Explicación de los parámetros
  - No posee
- Resultado generado
  - Base de datos construida y lista para su uso por parte de ConstruirIndiceMySQL.java

Referencias bibliográficas

[1] "Scaling Apache Giraph to a trillion edges - Facebook." Accedido el 12/11/2015. 2013 <https://www.facebook.com/notes/facebook-engineering/scaling-apache-giraph-to-a-trillion-edges/10151617006153920>

[2] D. Torres, H. Skaf-Molli, P. Molli, y A. Diaz. BlueFinder: Recommending Wikipedia Links Using DBpedia Properties. En ACM Web Science Conference 2013 (WebSci 13). París, Francia, Mayo 2013 <http://dl.acm.org/citation.cfm?id=2464515>

[3] Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, Grzegorz Czajkowski. Pregel: a system for large-scale graph processing, Actas de la Conferencia Internacional ACM SIGMOD de 2010 sobre la gestión de datos. Indianapolis, Indiana, USA. Junio 2010 <https://kowshik.github.io/JPregel/pregel_paper.pdf>

[4] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein. Graphlab: A new parallel framework for machine learning. “Conference on Uncertainty in Artificial Intelligence”. 2010 <http://www.select.cs.cmu.edu/publications/paperdir/uai2010-low-gonzalez-kyrola-bickson-guestrin-hellerstein.pdf>

[5] Diego Torres, Pascal Molli, Hala Skaf-Molli, Alicia Diaz. Improving Wikipedia with DBpedia. SWCS - Semantic Web Collaborative Spaces Workshop 2012 in 21st WWW Conference 2012,Apr 2012, Lyon, France. <hal-00688145>

[6] IBM - ¿Qué es MapReduce ?, accedido el 28/11/2014: <http://www-01.ibm.com/software/data/infosphere/hadoop/mapreduce/>

[7] Sitio oficial de Hadoop, accedido el 28/11/2014: <http://hadoop.apache.org/>

[8] D. Torres, H. Skaf-Molli, P. Molli, y A. Diaz. Có-evolución entre la Web Social y la Web Semántica. Conferencia online desde Argentina hacia distintos lugares de Argentina y Francia. Octubre, 2014. http://hdl.handle.net/10915/41223

[9] Jeff Dean, Sanjay Ghemawat. Google, Inc. MapReduce: simplified data processing on large clusters. Communications of the ACM - 50th anniversary issue: 1958 - 2008, Volume 51 Issue 1, January 2008. Pages 107-113 <http://dl.acm.org/citation.cfm?id=1327492>

[10] Gunarathne, Thilina. Hadoop MapReduce v2 Cookbook Second Edition. Packt Publishing Ltd. Birmingham, UK. February 2015

[11] Weiss, Mark Allen. Data Structures and Algorithm Analysis in Java, third edition. Pearson. United States of America, November, 2011.

[12] Roman Shaposhnik, Claudio Martella, and Dionysios Logothetis. 2015. *Practical Graph Analytics with Apache Giraph* (1st ed.). Apress, Berkely, CA, USA.

[13] T. K. Landauer and D. W. Nachbar. Selection from alphabetic and numeric menu trees using a touch screen: breadth, depth, and width. SIGCHI Bull., 16(4):73–78, April 1985. 85, 91

[14] Kevin Larson and Mary Czerwinski. Web page design: implications of memory, structure and scent for information retrieval. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI ’98, pages 25–32, New York, NY, USA, 1998. ACM Press/Addison-
Wesley Publishing Co. 85, 91

[15] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. DBpedia: a nucleus for a web of open data. In *Proceedings of the 6th international The semantic web and 2nd Asian conference on Asian semantic web conference* (ISWC'07/ASWC'07), Karl Aberer, Philippe Cudré-Mauroux, Key-Sun Choi, Natasha Noy, Dean Allemang, Kyung-Il Lee, Lyndon Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, and Guus Schreiber (Eds.). Springer-Verlag, Berlin, Heidelberg, 722-735.

[16] Zhang, M., and Zhou, Z. A k-nearest neighbor based algorithm for multi-label classification. In Granular Computing, 2005 IEEE International Conference on, vol. 2, IEEE (2005), 718–721

[17] McColl, W. F. (1995). Scalable computing. In Computer Science Today (pp. 46-61). Springer Berlin Heidelberg.