DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico II

Recorridos y Árbol Generador Mínimo

28 de mayo de 2023 — Algoritmos y Estructuras de Datos III

Integrante	LU	Correo electrónico
Teplizky, Gonzalo Hernán	201/20	gonza.tepl@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2610 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (++54 +11) 4576-3300
http://www.exactas.uba.ar

Resumen

Para el segundo trabajo práctico de la materia, el enfoque será puesto en trabajar con distintos algoritmos de Grafos. Entre ellos, se encuentran la busqueda en profundidad - Depth First Search o DFS - donde el grafo se recorre transversalmente con distintas implementaciones. En este caso, pudimos hallar puentes, puntos de articulación, y también componentes fuertemente conexas, en el caso de los digrafos.
Tratamos con grafos sin pesos en sus ejes, así como también con aquellos con pesos en sus aristas, siempre que el enunciado lo requiere, como es el caso al hallar un Árbol Generador Mínimo.
En cuanto a los ejercicios de este trabajo, el desempeño de las implementaciones será testeada por un *juez online*[1].
Este informe se centrará en el tercer ejercicio del trabajo práctico, donde se nos pide modelar un problema que se resuelve con Árbol Generador Mínimo, implementado con distintas versiones del algoritmo de Kruskal. La estructura es la siguiente:

- **Introducción - Presentación del problema**: Se da un vistazo inicial del problema a resolver.
- **Desarrollo - Algoritmo. Correctitud y Complejidad del mismo**: Se explica el algoritmo desarrollado para la resolución del problema, justificando la complejidad computacional y la correctitud del mismo.
- **Experimentación y resultados**: Se compara empíricamente el funcionamiento del algoritmo para distintas implementaciones del algoritmo de Kruskal considerando grafos de diversas formas y tamaños.

Palabras clave: *Grafos*, *Recorridos*, *DFS*, *Orden Topológico*, *DAG*, *Componentes Fuertemente Conexas*, *Kosaraju*, *Árbol Generador Mínimo*, *Kruskal*, *Disjoint Set Union*.

Índice

[1] `https://www.spoj.com`

1. Ejercicio 3

1.1. Introducción - Presentación del problema

En este ejercicio se nos presenta un conjunto de N oficinas. Cada una de ellas se representa mediante dos posiciones (x, y) siendo estas las coordenadas de la oficina i-ésima en un eje cartesiano del espacio donde ellas se encuentran ubicadas.
Se desea proveer a estas oficinas de conexión a internet. Para ello, contamos con W modems que podemos instalar en cualquiera de ellas. Como W < N, sabemos de antemano que no podremos colocar un módem por oficina. Por lo tanto, si queremos garantizar que todas ellas tengan conexión, se requiere comprar cables que conecten a las oficinas entre sí, de forma que toda oficina tenga un módem, o esté conectada a alguna que lo tenga. Y esto sin importar de si en el medio de esa conexíon hay otras oficinas que también están conectadas.
Hay dos tipos de cables que pueden comprarse:

- Cables UTP: tienen costo por centímetro U. Sólo pueden utilizarse entre dos oficinas que se encuentran a distancia de hasta R centímetros, una de la otra.
- Cables de fibra óptica: tienen costo por centímetro V. Pueden utilizarse para cualquier distancia.

Se sabe que el precio de los cables de fibra óptica es por lo menos el precio de los cables UTP. A su vez, el precio de los cables UTP es a lo sumo el precio de los cables de fibra óptica. En otras palabras, $U \leq V$.
A partir de estos 5 parámetros mencionados, y las coordenadas de las N oficinas, debemos encontrar el mínimo costo que debe gastarse en cables UTP y en cables de fibra óptica.

1.2. Algoritmo

Se nos pide resolver el problema mediante un modelo eficiente basado en el algoritmo de Kruskal. Este permite encontrar un árbol generador mínimo en un grafo conexo cualquiera. En otras palabras, permite seleccionar aquellas $N-1$ conexiones pesadas de entre todas las conexiones entre cada par de nodos del grafo, cuya suma de costos es la menor posible que puede obtenerse de entre todos los árboles posibles que tiene este grafo.
Notar que puede existir más de un árbol que cumpla esta propiedad. Lo que se busca es encontrar uno de ellos, cualquiera sea, ya que el foco está puesto en el costo final que van a tener las conexiones.
En nuestra situación contamos con oficinas. En particular con sus ubicaciones, en el espacio que les corresponde en el pabellón. A partir de estas ubicaciones podemos calcular, para cada oficina, su distancia hacia todas las demás $N-1$ oficinas del conjunto. Esta distancia, medida en centímetros, está directamente relacionada al costo de extender un cable entre ambas y así conectarlas. Así logramos que si una oficina tiene conexión a internet, la otra también pueda contar con este servicio.
De esta forma, en el modelo que proponemos, nuestro grafo tendrá un nodo por cada oficina, y una arista por cada posible conexión entre cada par de ellas. Es decir, N vértices, y $N(N-1)/2$ aristas. El costo de cada conexión, de cada arista, será la distancia euclidiana entre cada par de oficinas. Es decir, si consideramos una oficina $v_1 = (x_1, y_1)$ y otra oficina $v_2 = (x_2, y_2)$, el costo $c = w(v_1, v_2)$ de la arista que representa la conexión posible entre ambas oficinas, se define como:

$$c = \sqrt{|x_1 - x_2|^2 + |y_1 - y_2|^2}$$

Notar que, en este modelo, el grafo resultante es completo. Todo nodo está conectado a los demás, cada oficina está conectada a las demás. Por lo tanto, al correr Kruskal sobre este grafo, serán seleccionadas $N-1$ conexiones de menor costo, de todas aquellas conexiones que conectan

cada par de oficinas.
Hasta este punto, contamos con las conexiones mínimas para que toda oficina esté conectada a las demás. Cada una de ellas se encuentra conectada a la oficina que tiene más cerca (una de las tantas). Pero notemos que no todas estas conexiones serán necesarias en todos los casos. Todo depende de la cantidad de modems con los que se cuenta para equipar a las oficinas. Esta cantidad es conocida: el W que nos ingresaron previamente.
Las $N-1$ aristas serán necesarias únicamente si la cantidad de modems es 1: no importa que oficina tenga el modem, si quiero que todas las demás también tengan conexión, debo extender un cable a cada una de ellas, para así unirlas a la estructura en la que el modem se halla. Supongamos W > 1, como contamos con más de un modem, si se siguen manteniendo las $N-1$ aristas que conectan cada oficina, estaíamos poniendo cables de más, ya que dos oficinas que tienen conexiones separadas a internet, cada uno por su lado, se encuentran conectadas entre ellas.
¿De cuántas conexiones entre oficinas, de las $N-1$ totales, podemos prescindir, si contamos con W modems? Con un módem necesitamos $N-1$ uniones entre oficinas distintas. Con dos, notemos que nos alcanza con $N-2$. Si tenemos 3 modems, $N-3$ uniones parecen suficientes:
Proposición: Si contamos con W modems, para mantener a todas las oficinas con conexión a internet, son necesarias $N-W$ uniones entre oficinas.
La cuestión ahora es, ¿Qué $N-W$ conexiones tomamos, de las $N-1$ conexiones totales entre oficinas que nos devuelve el algoritmo de Kruskal? Deseamos hallar el mínimo costo que debe invertirse en cables UTP y cables de fibra óptica para conectar estas N oficinas. Por lo tanto, basta con considerar las $N-W$ conexiones de menor costo, de las $N-1$ devueltas por Kruskal. Concretamente, aquellas conexiones cuyo costo sea menor o igual a R, serán de cable UTP, y aquellas cuyo costo sea mayor, serán de fibra óptica. Los costos por conexion, es de costo del cable (arista) multiplicada por costo del tipo de cable que corresponda, según de si el costo del cable es menor o igual a R, o no.

1.3. Correctitud del Algoritmo

Queremos devolver los costos mínimos que deben invertirse en cables UTP y fibra óptica para que toda oficina tenga conexión a internet, si contamos con W modems con los que podemos equiparlas.
En el modelo planteado, los conjuntos de oficinas conectadas entre sí, pueden verse como un bosque dentro del grafo original, donde los nodos representan las oficinas, y aristas las distancias entre cada par de ellas. Cada oficina se encuentra conectada a la única oficina que se encuentra a distancia mínima de ella, dentro de una de las componente conexa del grafo.
Para que un conjunto de oficinas tenga internet, requerimos que haya un módem al cual puedan alcanzar mediante un camino de cables y oficinas intermedias. Podemos pensarlo de la siguiente forma: si tengo T oficinas, $T \leq N$, conectadas entre ellas utilizando la cantidad mínima de cables $(T-1)$, si ubicamos un único módem en cualquiera de las T oficinas, las $T-1$ oficinas restantes pasarán a tener conexión a internet. Ahora, como queremos que los costos de cables UTP y fibra óptica sean los mínimos, debemos comprar la cantidad mínima de cada uno para pooder garantizar esto. ¿Cuál será esta mínima cantidad? Aquella que genere que haya exactamente W grupos de oficinas. En estos W grupos, cada oficina debe encontrarse conectada a las demás, mediante un camino de oficinas y cables intermedio. Notemos que estos grupos de oficinas podemos pensarlos como componentes conexas del grafo original que modela las N oficinas conectadas todas con todas mediante un único cable. Ahora, como debe invertirse el menor costo posible, esto se traslada a conectar cada oficina a una de estas componentes que se van formando, a aquella que se encuentre a menor distancia de ella: de esta forma, el costo del cable será el más barato posible.
Notemos que estos requerimientos son satisfechos en su totalidad por el algoritmo de Kruskal: se pretende encontrar el bosque de tamaño exactamente W del grafo, que contenga únicamente las $N-W$ aristas de menor costo. Partimos del ordenamiento de las aristas por su peso (distancia entre las oficinas cuya conexión representan), en forma ascendente. Por el algoritmo de Kruskal, sabemos en la iteración i-ésima, $i < N-W$ tendremos exactamente $N-i$ componentes conexas en nuestro bosque parcial. Las $N-i$ aristas elegidas para conectar algunas oficinas entre sí, son aquellas

de menor costo. Entonces, luego de $N - W$ iteraciones del algoritmo, tendremos conformado un bosque de tamaño W. En el bosque, estarán únicamente las $N - W$ aristas de menor costo, que permiten que cada vértice (oficina) sea parte de una única componente conexa, de las W totales. Por cada componente conexa formada, tenemos un camino entre cada par de oficinas de ella, en caso de que su cardinal fuese mayor a 1. Entonces, no importa en cuál de todas las oficinas de cada subárbol coloquemos el módem, puesto que todas las oficinas igualmente tendrán acceso a la conexión que este provee. Por lo tanto, si ubicamos un módem por cada componente conexa formada, podemos darle conexión a internet a cada oficina de las N totales.
La sumatoria de los pesos de las $N - W$ aristas que prevalecen en el bosque, será la mínima que puede obtenerse, de agarrar cualquier subconjunto de $N - W$ aristas del grafo original, y sumar sus costos. Luego, para calcular cuáles son los costos mínimos que deben invertirse en cables, basta con devolver la sumatoria de todas las aristas cuyo peso es menor o igual a R, y multiplicar este resultado por U, en el caso del costo de cables UTP, y hacer algo similar con todas las aristas restantes, de esas $N - W$ aristas del bosque, multiplicándolas en este caso por V, en el caso del costo de cables de fibra óptica.

1.4. Complejidades del Algoritmo

Consideramos la complejidad para una única instancia de entrada. La complejidad total será la constante C (cantidad de casos de test que van a ingresarse), multiplicada por la complejidad máxima de entre las C complejidades de cada una de las instancias procesadas. En realidad, la complejidad de las C instancias es siempre la misma. Más concretamente, consideraremos como máxima, la que mayor tamaño de entrada tenga.
Representamos el grafo que modela las oficinas y las distancias dos a dos entre ellas, mediante un conjunto de aristas pesadas, siendo la distancia entre cada par de ellas, el costo de la arista que las une en el grafo. Obtenidas las coordenadas de las N oficinas, armar el grafo constituido por las aristas que representan la distancia entre cada par de oficinas, podemos hacerlo en $O(N(N-1)/2)$ $= O(N^2)$ pasos. Cada uno de ellos lo realizamos en tiempo constante, $O(1)$, asumiendo que la distancia euclidiana entre dos oficinas es calculable en este tiempo.
Armado el grafo, corremos Kruskal en su versión para grafos densos que utiliza la estructura de Disjoint Set Union para representar las uniones entre los vértices que finalmente conformarán el Árbol Generador Mínimo que buscamos. Este algoritmo requiere tener ordenadas las aristas, lo que podemos hacer en $O(N^2 Log(N))$ pasos. Luego la estructura del DSU nos permite obtener el AGM en $O(N^2 Log(N))$ pasos en el peor caso. Esto se debe a que tenemos que recorrer las M aristas, M del orden $O(N^2)$ debido a la densidad del grafo, para encontrar las $N - 1$ aristas que conformaran el árbol. Debido a las optimizaciones de Path Compression y Union by Rank utilizadas, la operación de find es realizada en tiempo $Log(N)$ en peor caso, por lo que la union de un vértice con otro se realiza en $Log(N)$ pasos. La complejidad total de Kruskal es entonces $O(N^2 Log(N))$.
A partir de este AGM, también representado como conjunto de aristas pesadas, tomamos las primeras $N - W$ aristas, puesto que ya vienen ordenadas en forma ascendente. Esto podemos realizarlo en $O(N)$ pasos, puesto que sabemos que un árbol tiene $N - 1$ aristas, y que $W < N$.
Estas aristas recordamos que representan las conexiones mínimas necesarias para garantizar que toda oficina tenga conexión a internet. Debemos devolver el costo de este cableado, medido en centímetros de cables UTP y fibra óptita según su tamaño (distancia) con respecto a R. Entonces por cada arista chequeamos su tamaño y, dependiendo de este, incrementamos el costo del cable correspondiente en la cantidad de centímetros de la distancia, multiplicada por el costo del cable adecuado. Esto puede realizarse en tiempo $O(N)$ si nuevamente asumimos que las operaciones suma y multiplicación pueden realizarse en tiempo constante cada una.
Finalmente, devolvemos los costos mínimos de cableado UTP y fibra óptica que calculamos en el paso previo, en $O(1)$. El algoritmo completo puede realizarse en $O(N^2 Log(N))$ pasos totales.

1.5. Experimentación

Para finalizar con el informe, nos propusimos hacer un análisis de la performance de distintas implementaciones del algoritmo de Kruskal. En particular, nuestro grafo es siempre denso, ya que asumimos conectadas a todo par de oficinas del edificio. Para encontrar un Árbol Generador Mínimo para el mismo, utilizamos la estructura de **Disjoint Sets**, la cual nos permite separar a los vértices del grafo según el árbol en el que se encuentran. El nombre se debe justamente a que un nodo puede estar solamente en una única componente, haciendo que todo conjunto sea disjunto.
Como sabemos, la interfaz de la estructura nos provee dos operaciones: *Find y Unite*. Por un lado, queremos conocer el padre o representante de un vértice en el árbol en el que se encuentra, y por el otro, queremos poder unir a dos nodos cuyos valores de Find difieren según nos sea conveniente.

Es aquí donde entran en juego las optimizaciones, que resultan cruciales para mejorar el rendimiento del algoritmo, y más aún al tener una cantidad muy grande de aristas que procesar.
En principio, tomaremos las dos versiones de DSU que utilizan optimizaciones, buscando ver si existe alguna diferencia considerable entre las mismas.
El **objetivo de ambas versiones** es minimizar el tiempo de unión (*Unite*) entre los subárboles parciales del bosque, producto del invariante del algoritmo de Kruskal para hallar el AGM del grafo original.
Estas versiones a las que nos referimos son:

- *Union By Size*
- *Union By Rank*

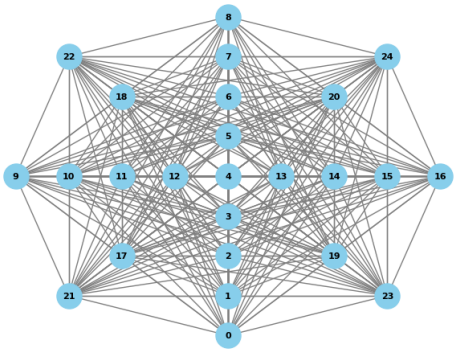
Lo que hace **óptimas** a estas versiones, es que usan lo que llamamos *Path Compression*, donde se retroalimentan las operaciones de *Find y Unite*. Al unir por **tamaño**, se mantiene un vector de tamaños de la componente conexa de cada nodo, y las uniones se hacen según el vértice que este en la componente predominante. Por su parte, al unir por **rank**, la raíz del vértice cuyo árbol sea de menor altura se agarra del de mayor altura.
Estas formas de realizar la unión permiten hacer logarítmico el *Find*, debido al balanceo de los árboles que se van armando, el cual se implementa también con optimizaciones, buscando que las próximas veces que se llama al *Find* del mismo vértice, sea en un tiempo lo más parecido a constante.

Dado que las complejidades en estas dos versiones de DSU optimizadas son a nivel teórico las mismas, con el *Find* en $O(Log(N))$, lo que pensamos es que ambas van a ser **igual o prácticamente igual de óptimas**, y **mejores** que la naive, con el Find en peor caso $O(N)$, sin optimizar, donde unimos sin hacer consideraciones, poniendo de padre de un nodo al otro nodo. Queremos ver si el hecho de tener grafos de tantas aristas **genera algún efecto particular o no**.
Lo que haremos será tomar grafos K_n de distintas formas, y para cada uno de ellos, analizaremos los tiempos de ejecución con las 3 implementaciones que mencionamos previamente: las dos optimizadas, y la que es naive, sin considerar *Rank, Size o Path Compression.* Luego, uniremos en un único grafo todos los casos, y le iremos agregando más oficinas, para ver si sigue sin haber diferencia en los tiempos si es que no la hay originalmente, o si ésta es más marcada a medida que crece el input.

Aclaración: todos los grafos generados a continuación fueron armados proponiendo una secuencia de posiciones acorde para que las oficinas queden distribuidas según la forma especificada. Los dibujamos a mano, y luego, con un pequeño script de Python que incluiremos en la entrega, los graficamos. Se nos ocurrió la idea de armar una experimentación así y recibimos el visto bueno en clase.

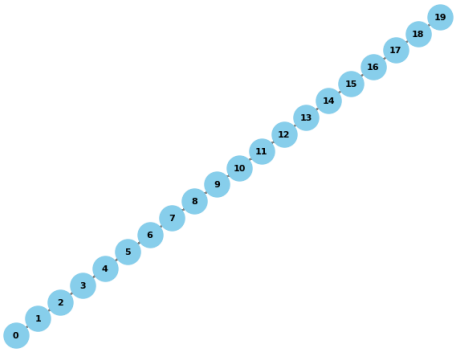
1.5.1. Grafo tipo En Capas

El primer grafo con el que experimentamos es uno con una forma similar a una espiral, por el hecho de que está como autocontenido en capas cada vez más chicas. En particular, es un K_{25}, es decir, el grafo completo de 25 vértices. Así se ve el grafo:



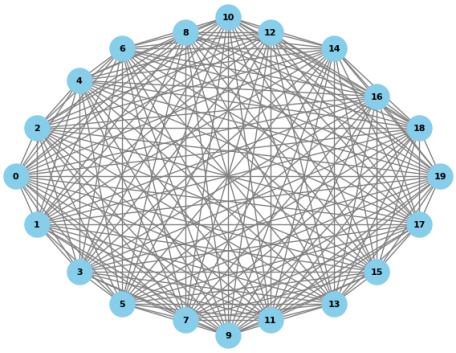
1.5.2. Grafo tipo Linea Recta

En este grafo, representamos al K_{20}, el completo de 20 vértices, como una línea recta. Así se ve el grafo (nos quedó como árbol pero deberían estar las aristas entre todos):



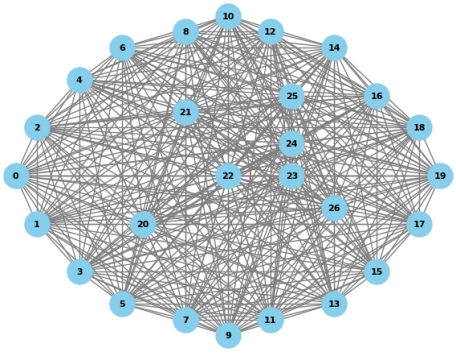
1.5.3. Grafo tipo Circulo

Tomamos otro K_{20}, pero con una forma parecida a un círculo. Así se ve el grafo:



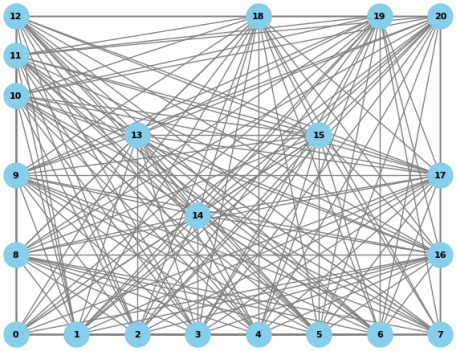
1.5.4. Grafo tipo Circulo recargado

Este grafo es de la misma forma que el anterior, aunque es un K_{27}, de 27 vértices. Así se ve:



1.5.5. Grafo tipo Cuadrado

Por último, representamos un K_{21} con forma de cuadrado, aunque no cerrado del todo. Así se ve el grafo:



1.5.6. Casos mayores

Además de hacer la experimentación con estos grafos por separado, procedemos a unirlos en uno único.

Una vez unidos, les agregamos de forma aleatoria una cierta cantidad de posiciones de modo que lleguemos a $n = 150$, $n = 200$ y $n = 500$, con n el total de oficinas.

1.6. Resultados

Procedemos a mostrar el total en segundos de cada test, es decir, del tiempo de ejecución del algoritmo de Kruskal en cada una de las 3 implementaciones mencionadas, para los 5 grafos que construimos. Estos tiempos los calculamos agregando al código de cada implementación la librería *chrono* de C++, guardando el tiempo de inicio y fin de ejecución y mostrando en segundos el resultado de su resta.

- Grafo I
  DSU con Union By Rank: **0.00047** segundos | *DSU con Union By Size*: **0.00046** segundos | *DSU Naive*: **0.00049** segundos
- Grafo II
  DSU con Union By Rank: **0.00035** segundos | *DSU con Union By Size*: **0.00033** segundos | *DSU Naive*: **0.00039** segundos
- Grafo III
  DSU con Union By Rank: **0.00030** segundos | *DSU con Union By Size*: **0.00029** segundos | *DSU Naive*: **0.00030** segundos
- Grafo IV
  DSU con Union By Rank: **0.00051** segundos | *DSU con Union By Size*: **0.00051** segundos | *DSU Naive*: **0.00054** segundos
- Grafo V
  DSU con Union By Rank: **0.00032** segundos | *DSU con Union By Size*: **0.00036** segundos | *DSU Naive*: **0.00036** segundos

Lo que podemos notar es que esencialmente, para cada grafo, en cada una de las implementaciones de Kruskal, el tiempo de ejecución **es el mismo.** Lo que sí sucede es que, aunque sea igual, sí que es mayor a medida que el grafo es más grande, como es el IV, que era de 27 oficinas.
Esto nos dio la pauta de que, al pasar a la segunda parte de la experimentación, donde generamos instancias ya **sin importarnos la forma, si no el tamaño**, el n, el tiempo iba a ir aumentando, pero ya veíamos posible que la duración pudiese ser igual en todas las implementaciones.

Presentamos los tiempos obtenidos en las 3 implementaciones para los grafos de n = 150, $n = 200$ y $n = 500$.

- $n = 150$
  DSU con Union By Rank: **0.01662** segundos | *DSU con Union By Size*: **0.01577** segundos | *DSU Naive*: **0.01640** segundos
- $n = 200$
  DSU con Union By Rank: **0.02797** segundos | *DSU con Union By Size*: **0.02549** segundos | *DSU Naive*: **0.02678** segundos
- $n = 500$
  DSU con Union By Rank: **0.15312** segundos | *DSU con Union By Size*: **0.14532** segundos | *DSU Naive*: **0.15557** segundos

Concluimos en que ni la forma del grafo, ni el total de oficinas o el hecho de ser pesado, generaron un efecto que lleve a que los tiempos entre las distintas implementaciones difieran. Tanto las dos optimizadas como la que no, incluso con grafos grandes, con 500 oficinas y completo, es decir, de mas de 100.000 aristas, computan en igual tiempo. El tiempo crece con estos grafos más grandes, como habíamos visto levemente en la primer parte de la experimentación. Finalmente, la teoría inicial de que la versión naive tendría distancia significativa en eficiencia, no se cumplió.