Práctica MARP
Algorítmo de dijkstra con montículo sesgado

Juan Chozas Sumbera

6 de enero de 2020

UNIVERSIDAD COMPLUTENSE
MADRID

1. Implementación

He elegido Java como lenguaje para variar, ya que el año pasado hice las dos prácticas en C++. Para representar grafos he elegido listas de adyacencia: una lista de pares de enteros para cada vértice (vértice adyacente, coste de la arista).

El montículo sesgado contiene un entero (tamaño) y un nodo (raíz). La inclusión de la operación *decrecerClave* trae la necesidad de almacenar y mantener la tabla (clave: entero, valor: nodo) que almacena todos los nodos del montículo. Un nodo contiene un puntero a su nodo padre, además de a sus dos hijos. Para una implementación eficiente de *decrecerClave*, uso estos punteros para *cortar* el nodo objetivo del padre y unirlo con el montículo restante.

2. Ejecución

Incluyo también un script llamado `dijkstra` que puede usarse para compilar y ejecutar el programa. Admite varios argumentos, de los cuales solo es obligatorio `-n NUM`, usado para generar un grafo de `NUM` nodos. Se puede omitir este argumento en caso de usar la opción `-t 1` o `-t 2`, que ordena la ejecución de uno de los casos de prueba. Los grafos usados para probar el algoritmo se crearon de forma aleatoria, y se puede elegir la semilla con la opción `-s SEM`. La opción `-h` proporciona más información acerca de las opciones.

3. Medición

3.1. Métodos empleados

El grafo se genera de antemano, y el vértice inicial siempre es el 0. Los tiempos se han medido usando la función `System.nanoTime()` midiéndose únicamente el tiempo transcurrido durante la ejecución del algoritmo de la siguiente forma:

```
long t1 = System.nanoTime();
resultados = dijkstra(g, 0);
long t2 = System.nanoTime();
```

La máquina usada durante la medición siempre estaba enchufada a la corriente. El sistema operativo que usa es Ubuntu 18.04, y todas las ejecuciones se realizaron en terminales `tty`. Desactivé la conexión de la máqui-

na a Internet con las instrucciones `ip link set wlp2s0 down` y `service network-manager stop`. Las ejecuciones se hicieron usando los siguientes instrucciones:

```
for i in ‘seq 5 5 100‘; do
    ./dijkstra -n $i -a -d -i 6 > tiempos/$i;
done
for i in ‘seq 125 25 1000‘; do
   ./dijkstra -n $i -a -d -i 6 > tiempos/$i;
do ne
for i in ‘seq 1050 50 5000‘; do
    ./dijkstra -n $i -a -d -i 6 > tiempos/$i;
done
```

A partir de 5000 nodos, saltaban excepciones relacionadas al límite de memoria. Las opciones indican

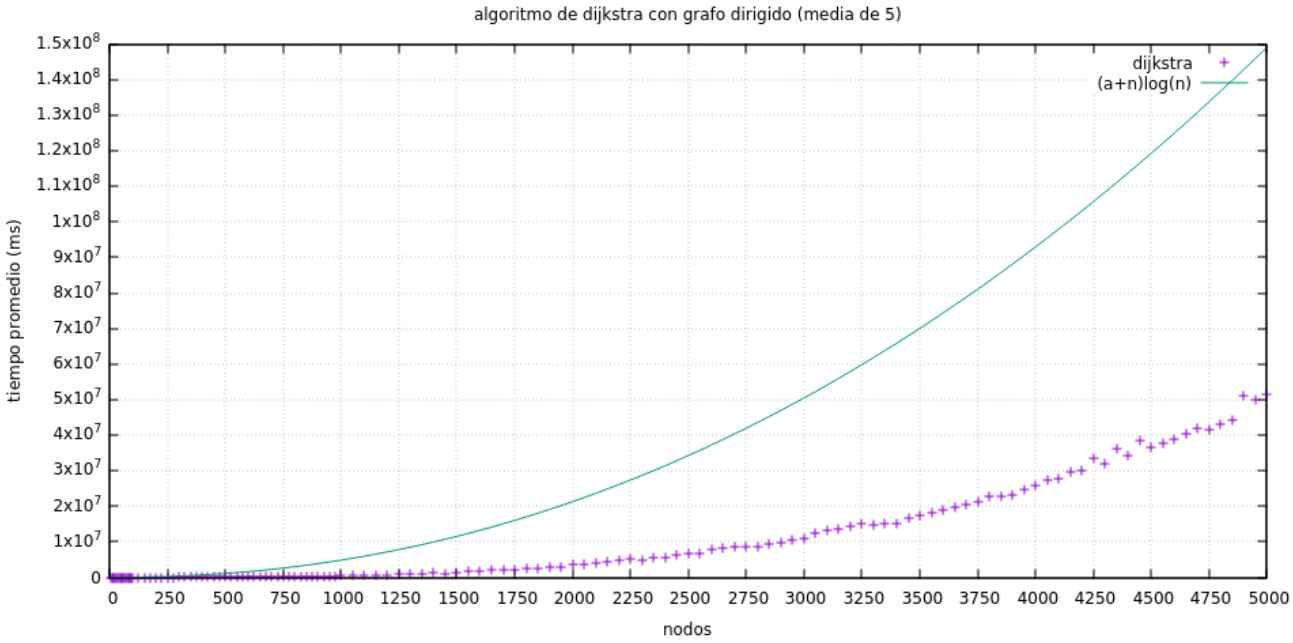
- el número de nodos (`-n $i`)
- que se imprima el número de aristas generadas (`-a`)
- que el grafo generado sea dirigido (`-d`)
- que se ejecute el algoritmo 6 veces (`-i 6`)

La última opción resulta en 6 medidas distintas, de las cuales la primera siempre tenía un valor mucho más elevado al resto (al menos en grafos con no muchos vértices). En la siguiente sección aparecen gráficas que descartan esta primera medida en el cálculo de la media, y otras que la incluyen. También hay gráficas de grafos no dirigidos para los mismos rangos de vértices (mismas instrucciones pero sin la opción `-d`).

Finalmente, procesé las medidas en la carpeta `tiempos` con dos scripts, `averages.sh` y `edges.sh`, con el fin de calcular las medias de las medidas y obtener el numero de aristas generadas para la prueba, respectivamente.

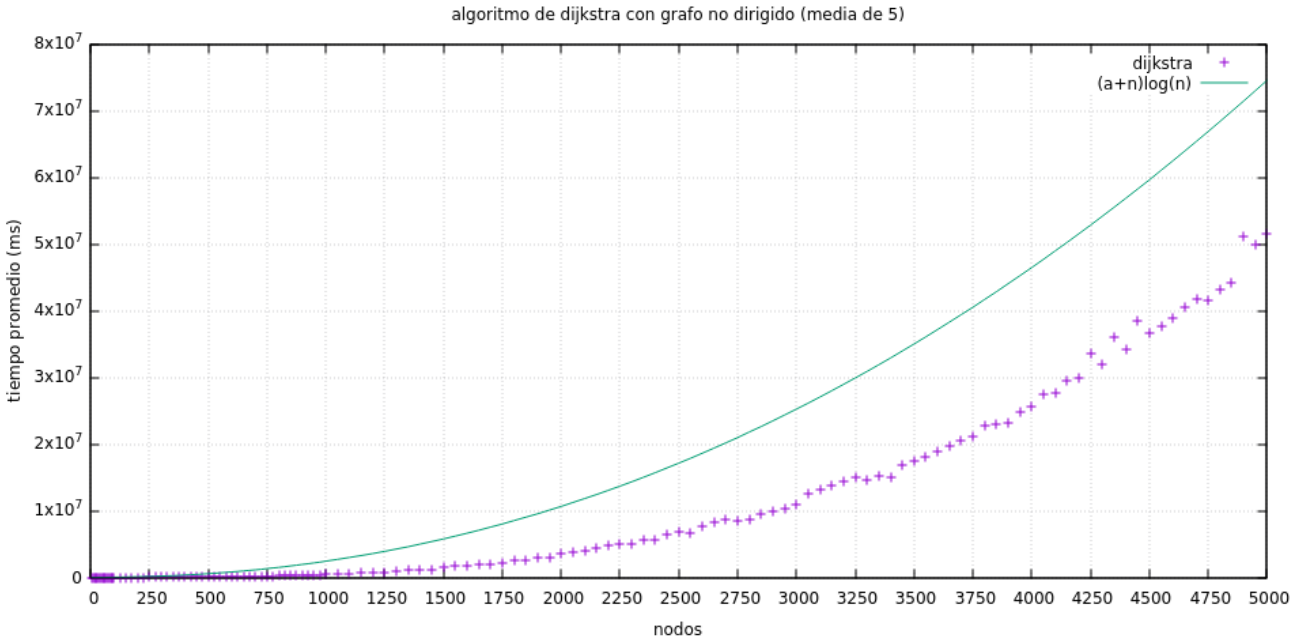
4. Resultados

A continuación aparecen las gráficas. De las 6 ejecuciones en cada numero de nodos probado, las dos primeras gráficas no tienen en cuenta la primera ejecución: el primer tiempo medido era mucho mayor que el del resto, que no variaba mucho de uno a otro. Esto lo pude comprobar observando el comportamiento en un número alto de iteraciones con la opción `-i`.



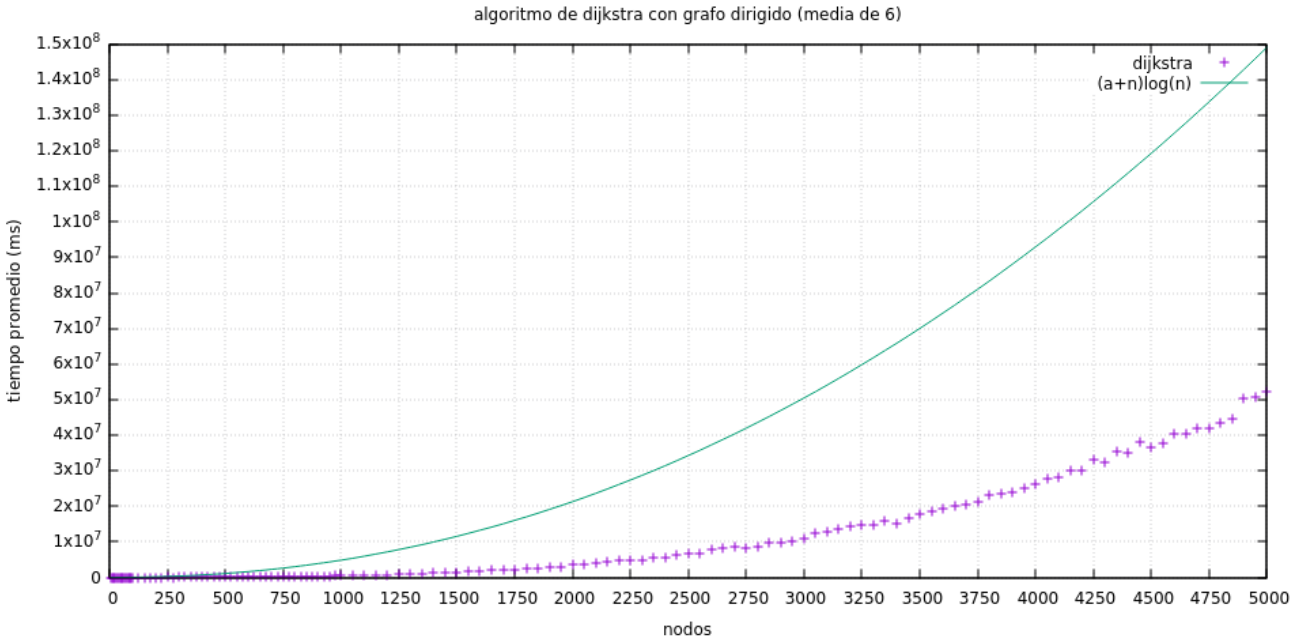
`Gráfica 1 - Grafos dirigidos sin tener en cuenta la primera medida`

La curva del tiempo es menos acentuada que la de la cota superior, pero la curva tiene la misma forma. Estimo que la curva de las medidas está en `z(a+n)log(n)`, con `z` entre 0.3 y 0.5. Puede ser porque el algoritmo termina sin visitar todos los vértices: con grafos dirigidos es más probable que haya nodos inaccesibles por haber restricciones de dirección en las aristas.



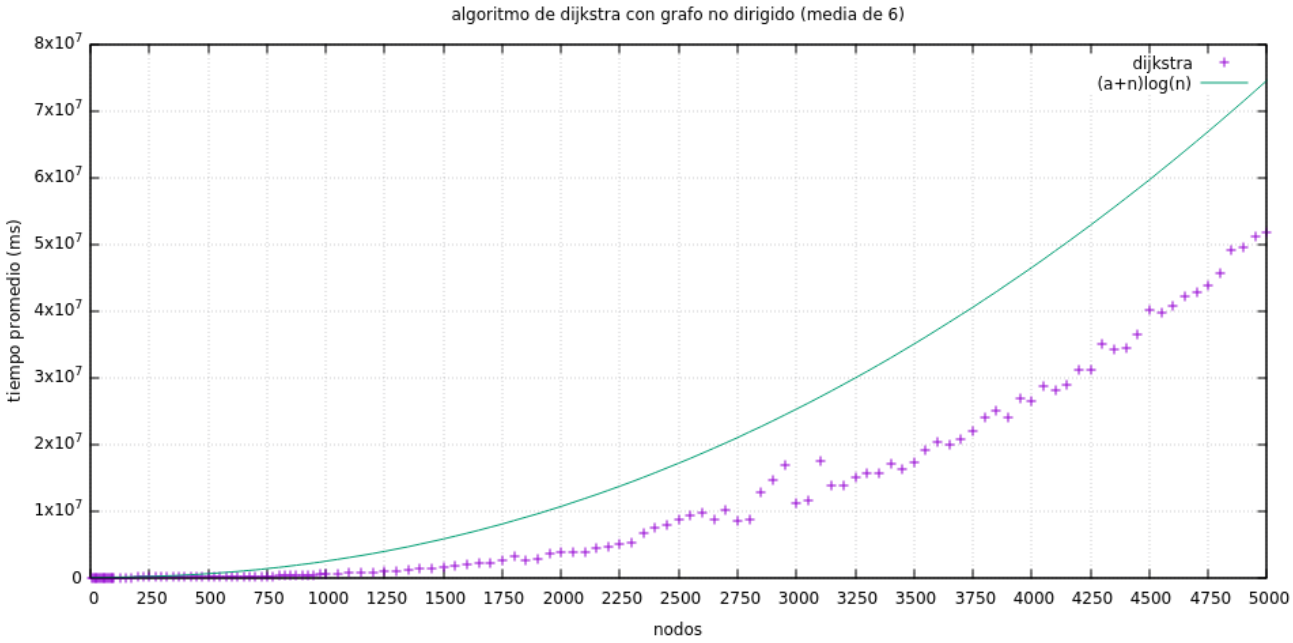
Gráfica 2 - Grafos no dirigidos sin tener en cuenta la primera medida

La curva de las medidas se asemeja más la curva de la cota superior que cuando los grafos son dirigidos. Estas medidas también parecen seguir la curva de la cota superior: diría que la curva es `z(a+n)log(n)`, con `z` entre 0.5 y 0.7. Yo creo que esto se debe al hecho de que hay más aristas que en un grafo dirigido, lo cual hace más probable que todos los nodos sean alcanzables. Es menos probable que el algoritmo termine sin haber visitado todos los nodos.



Gráfica 3 - Grafos dirigidos teniendo en cuenta la primera medida

Es muy similar a la gráfica 1. Algunos puntos suben y otros bajan, pero la diferencia es apenas notable.



Gráfica 4 - Grafos no dirigidos teniendo en cuenta la primera medida

La inclusión de la primera medida es más notable en esta gráfica que en la 3: entre 2650 y 3000 nodos hay mucha más varianza que en la gráfica 2. Creo que se le puede echar la culpa a la máquina en este subconjunto de pruebas, pues todos los tiempos son mayores. Fuera de este pequeño rango, las medidas no son muy diferentes a las de la gráfica 2. Comparando con la segunda gráfica, puedo concluir que para cada primera ejecución en el rango de nodos mencionado, el tiempo de medido es mucho más elevado al de las otras 5 ejecuciones. No sabría explicar por qué solo pasa en la primera ejecución, ni por qué hay tanta diferencia en este rango de nodos.