Práctica MARP
Algorítmo de dijkstra con montículo sesgado

Juan Chozas Sumbera

3 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 la representación mediante listas de adyacencia, que contienen de pares de enteros para almacenar el vértice adyacente y el coste de la arista.

Para el montículo sesgado se almacena un entero (tamaño), un nodo (raíz), y una tabla (clave: entero, valor: nodo). La inclusión de la operación *decrecerClave* trae la necesidad de almacenar y mantener la tabla que almacena todos los nodos del montículo. Un nodo almacena un puntero a su nodo padre, además de a sus dos hijos. Para una implementación eficiente de *decrecerClave*, uso el puntero al padre del nodo objetivo para *cortar* el nodo objetivo del padre (en caso de que la clave decrecida sea menor que la del padre).

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 cargando. El sistema operativo que usa es Ubuntu 18.04, y todas las ejecuciones se realizaron

en terminales `tty`. Desactivé la conexión de la máquina 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. 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 `(a + n)*log(n)` respectivamente.

4. Resultados

Gráfica 1