Algoritmos y Estructuras de Datos III

Trabajo Práctico 3

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Mejorando el tráfico

Informe

Integrante	LU	Correo electrónico
Braginski Maguitman, Leonel Alan	385/21	leobraginski@gmail.com
Deganis, Gaston Lucas	295/20	gastondeganis@gmail.com

1. Problema

Abordaremos el problema de buscar en un mapa de N esquinas y M calles una distancia mínima entre dos puntos críticos de una ciudad. En particular para este problema se nos da por entrada estos dos puntos críticos s y t, pero también contamos con una lista de k nuevas calles bidireccionales que podemos elegir construir en nuestra ciudad. Nuestro objetivo es encontrar cuál es la distancia mínima entre estos 2 puntos s y t teniendo en cuenta que podemos ayudarnos de a lo sumo una de estas nuevas calles bidireccionales.

Parte I

Algoritmo y demostración

2. Representación en grafos

Adaptamos este problema a una búsqueda de camino mínimo en grafos, notamos a la ciudad como el grafo dirigido pesado G donde las esquinas son representadas por los vértices y las aristas son las calles con su respectiva longitud. Las k nuevas calles bidireccionales se guardan en memoria como lista de aristas L.

3. Dijkstra

Vamos a resolver este problema utilizando el algoritmo de `Dikstra`, con el que podemos buscar y guardar las distancias mínimas entre un nodo elegido con el resto de los nodos del grafo. Usando este algoritmo dado un grafo y un vértice s, armamos un vector D_s de tamaño N donde guardamos para cada posición i cual es la distancia mínima desde s hasta i, trivialmente $D_s[s] = 0$ y si no existe camino posible desde s a algún vértice t, entonces $D_s[t] = \infty$

4. Solución del problema

Al momento de estar armando el grafo G también armamos su grafo inverso R, el cual contiene los mismos vértices y aristas pero con las direcciones opuestas. Ahora bien, iteramos sobre L y probamos para cada posible calle nueva si agregarla a la ciudad mejora la distancia mínima entre s y t. Podemos lograr esto viendo en cada iteración un "corte" del grafo de s a u y de v a t donde u y v son las aristas de la nueva calle bidireccional.

BuscarDistMinima

```
1:  D_s ← Dijkstra(G, s)                                   ▷ O((N + M)logN)
2:  D_t ← Dijkstra(R, t)                                   ▷ O((N + M)logN)
3:  dist ← D_s[t]
4:  for i < k do                                           ▷ O(k)
5:      u ← L[i].salida
6:      v ← L[i].destino
7:      l ← L[i].costo
8:      if l + min(D_s[u] + D_t[v], D_s[v] + D_t[u]) < dist then
9:          dist ← l + min(D_s[u] + D_t[v], D_s[v] + D_t[u])
10: return dist
```

Complejidad: $O(2(N + M)logN) + O(k) = O((N + M)logN)$

Usando los dos vectores de distancias, uno partiendo desde s y otro al revés desde t nos fijamos si dirigir el camino desde s hasta u, agregar el nuevo costo y dirigirlo devuelta hasta t es mas barato de lo que ya teníamos, si este es el caso marcamos esta distancia como mínima. Cabe destacar que al ser una calle bidireccional tenemos que fijarnos también si podemos minimizar incluso dando la vuelta, es decir el camino de s a v y u a t. Este algoritmo es `Greedy`

5. Correctitud

Vamos a denotar que $D\{G\}_u[v]$ es la distancia mínima calculada con `Dijkstra` desde u hasta v en el grafo G. La correctitud de `Dijkstra` esta demostrada en las clases teóricas de la materia.
Sea e la arista de la iteración i del algoritmo, y sea $G' = G + e$
Lo que queremos probar es que siempre tenemos en *dist* la distancia mínima entre s y t en cualquier grafo generado por agregar la arista e a G. En otras palabras alcanza con demostrar que vale

$$l + min(D\{G\}_s[u] + D\{G\}_t[v], D\{G\}_s[v] + D\{G\}_t[u]) < D\{G\}_s[t] \iff D\{G'\}_s[t] < D\{G\}_s[t]$$

donde u, v y l son los vértices y pesos de la arista e correspondientemente. Tener en cuenta que este algoritmo prueba con todas las posibles inserciones de rutas sin modificar el grafo por lo que el resultado final es la distancia mínima obtenible en cualquiera de los casos.

5.1. $\Rightarrow$

Si vale la primera condición, eso significa que existe un camino $s \rightsquigarrow t$ que pasa por $s \rightsquigarrow u$, $u \to v$ y $v \rightsquigarrow t$ o que pasa por $s \rightsquigarrow v$, $v \to u$ y $u \rightsquigarrow t$ tal que es menor al camino original $s \rightsquigarrow t$. Esto implica que la distancia mínima en G' va a usar esta arista ya que no existe otro camino mínimo mejor. Por lo que vale que $D\{G'\}_s[t] < D\{G\}_s[t]$

5.2. $\Leftarrow$

Tomemos e como una arista bidireccional del camino mínimo $s \rightsquigarrow t$ en G' en la que valga $D\{G'\}_s[t] < D\{G' - e\}_s[t]$. Veamos que los caminos $s \rightsquigarrow u$ o $s \rightsquigarrow v$ y $v \rightsquigarrow t$ o $u \rightsquigarrow t$ son los mismos en G y G' ya que todas las aristas entre esos caminos son las mismas en ambos grafos. Entonces $l + min(D\{G\}_s[u] + D\{G\}_t[v], D\{G\}_s[v] + D\{G\}_t[u])$ es lo mismo que calcular $D\{G'\}_s[t]$ ya que estamos agregando esta arista $u \to v$ virtualmente. Vale entonces que así como $s \rightsquigarrow t$ era mas chico en G', usarlo con la inserción virtual de e en G también es menor que $D\{G\}_s[t]$

Parte II

Demostración empírica

6. Implementaciones de Dijkstra

El algoritmo de `Dijkstra` tiene una complejidad de $O(|V|^2 + |E|)$. Esto se puede mejorar usando colas de prioridad para guardar los vértices por visitar.

`Dijkstra sin colas de prioridad` $\Rightarrow O(|V|^2 + |E|)$
`Dijkstra con Min Heap` $\Rightarrow O((|V| + |E|)log|V|)$
`Dijkstra con Fibonacci Heap` $\Rightarrow O(|V|log(|V| + |E|))$

7. Experimentación

Programamos la solución a este problema para cada una de 3 versiones de `Dijkstra` para experimentar sobre ellas. Creamos 500 muestras aleatorias de entradas para el problema y las corrimos en cada versión del programa midiendo los tiempos. Cabe aclarar que en cada muestra tomamos el promedio de tiempo entre 10 ejecuciones. Estos fueron los resultados:

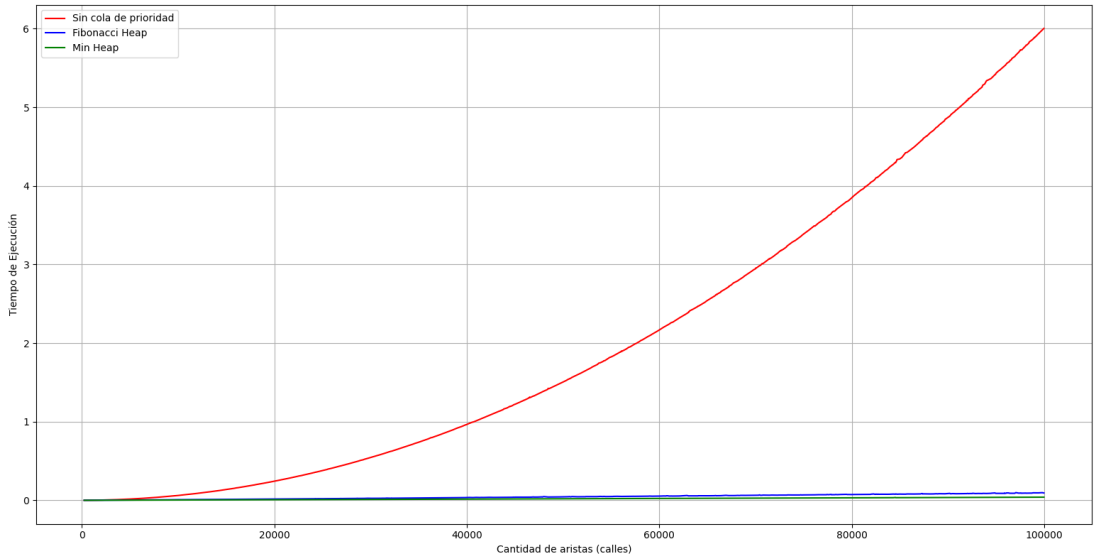

Figura 1: Comparación de las 3 implementaciones

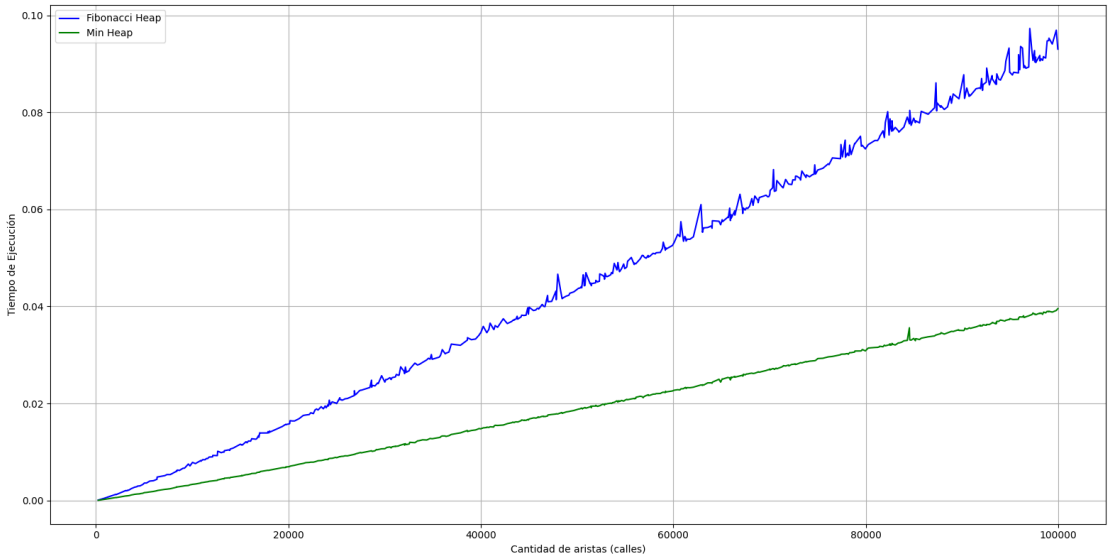

Figura 2: Acercamiento a las resoluciones Dijkstra con colas de prioridad

8. Conclusiones

- Podemos comprobar que para nuestro uso de `Dijkstra` las implementaciones usando colas de prioridad fueron mucho mejores, en particular la cola de prioridad de Min Heap tuvo mejor performance que la de Fibonacci.
- Vemos reflejadas las cotas esperadas para cada caso.
- Este problema no tiene instancias mas fáciles que otras ya que en cualquier caso estamos calculando camino mínimo sobre todo el grafo.
- Para cada muestra generada la cantidad de nodos N fue elegida al azar sin repetirse, a su vez generamos $M = 10 * N$ aristas para incrementar las probabilidades de encontrar un camino entre 2 nodos y no elegir aleatoriamente a M
- Acotando la complejidad nos queda

$$\texttt{Dijkstra sin colas de prioridad} \Rightarrow O(|V|^2)$$
$$\texttt{Dijkstra con Min Heap} \Rightarrow O(|V|log|V|)$$
$$\texttt{Dijkstra con Fibonacci Heap} \Rightarrow O(|V|log|V|)$$