Informe TP3

Facundo Gonzalez y Joaquín Polonuer

Descripción del problema

El problema consiste en elegir de una lista de calles bidireccionales, la que minimice la distancia entre dos puntos. Vamos a tener n puntos, m calles unidireccionales, s y t puntos críticos. Queremos decidir el costo de llegar de s a t pudiendo agregar solo una de las calles.
Para resolver este problema, modelamos un grafo con las calles unidireccionales y buscamos el camino mínimo entre s y t. Luego recorremos la lista de calles propuestas y calculamos el nuevo costo de s a t pudiendo usar esa nueva calle en ambos sentidos.

Algoritmo y justificación

Recordemos que, para cada arista $(u,\ v)$, se tiene que

$$d_G(s,u) \ + \ c(u,v) \ + d_G(v,t) \ = d_G(s,t) \ \Leftrightarrow (u,v)\ pertenece\ a\ CM\ en\ G$$

Vamos a querer verificar está condición. Para eso necesitamos precalcular las distancias $d_G(s,\bullet)$ y $d_G(\bullet,t)$ y lo podemos hacer con:

$$dijkstra(G,\ t)$$
$$dijkstra(G^T,\ t)$$

Luego, la longitud del camino mínimo es $d_G(s,\ t)$ según obtuvimos con dijkstra. Sin embargo, como podemos agregar aristas externas, podría pasar que, agregando una arista candidata e

$$d_G(s,u) \ + \ c(e) \ + d_G(v,t) \ < d_G(s,t)$$

Notar que cuando usamos la arista $(u \rightarrow v)$ para acortar el camino podemos escribir $d_G(s,u) \ = \ d_{G+e}(s,u)\ y\ d_G(v,t) \ = \ d_{G+e}(v,t)$, ya que no usaremos esta arista para llegar de s a u, ni de v a t. Además, usar $(u \rightarrow v)$ en nuestro nuevo camino mínimo nos garantiza que no vamos a usar $(v \rightarrow u)$ en ese camino, porque en ese caso tendríamos un ciclo negativo. Entonces, sucede que

$$d_{G+e}(s,t) \ = \ min\{d_G(s,t),\ d_G(s,u) \ + \ c((u \rightarrow v)) \ + d_G(v,t),\ d_G(s,v) \ + \ c((v \rightarrow u)) \ + d_G(u,t)\}$$

Esto vale, ya que o bien

- La distancia es la que calculamos previamente en G
- Nos conviene usar, en algún camino mínimo, la arista nueva que une u y v en alguno de los dos sentidos.

En el segundo caso, la distancia en el grafo $G + e$ será $d_G(s, u) + c(e) + d_G(v, t)$ (o cambiando u por v, dependiendo del sentido) porque, como notamos previamente, la distancias $d_G(s, u)$ y $d_G(v, t)$ se mantienen y podemos usar la propiedad de relajación para disminuir el camino de s a t usando la nueva arista.

Ahora, de todas las aristas e que podemos usar para aplicar relajación, queremos usar aquella que minimice la distancia en $G + e$, es decir $min_{e \in Candidatas}(d_{G+e}(s, t))$, que será la respuesta al problema.

De está forma, el algoritmo resulta ser

```
dijkstra(G, s)  //Tenemos las distancias de s a todos
dijkstra(G^T, t) //Tenemos las distancias de todos a t
distancia_minima = d(s,t)
for e=(u,v,w) in Candidatas:
     if d(s,u)+w+d(v,t) < distancia_minima:
          distancia_minima = d(s,u)+w+d(v,t)
     if d(s,v)+w+d(u,t) < distancia_minima:
          distancia_minima = d(s,u)+w+d(v,t)
return distancia_minima
```

Experimentación

Para la experimentación tuvimos en cuenta dos implementaciones de Dijksta. La primera utiliza un min-heap como cola de prioridad, y la segunda utiliza un array. La complejidad teórica para la implementación con min-heap es $O(m * log(n))$, mientras que la complejidad para la implementación con array es $O(n^2)$.

Hicimos experimentos con dos tipos de grafos distintos. Uno con grafos ralos, en los cuales $m \in O(n)$, y otro con grafos densos, en los que $m \in O(n^2)$.

Experimentando con grafos ralos, observamos que la implementación con min-heap no parecía corresponderse con su complejidad teórica. Haciendo pruebas concluimos que esto se debía a que, en general, no había camino de s a t en estos grafos ralos y por eso la ejecución terminaba antes de tiempo (Fig 1). En base a esto, decidimos hacer la experimentación con grafos ralos asegurando que existiera en ellos un camino de s a t.

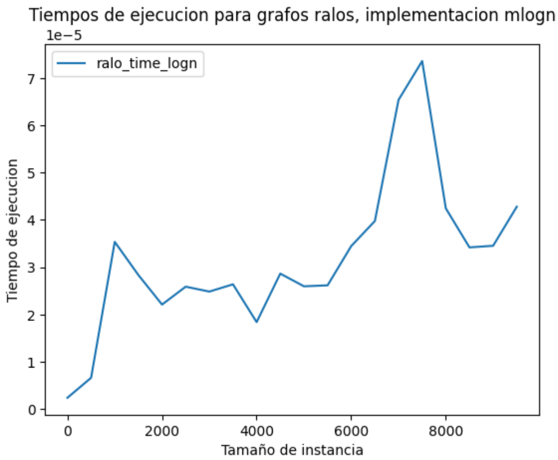

Fig 1: Tiempos de ejecución medidos para la implementación con min-heap en grafos donde no necesariamente existe camino de s a t. Notar la orden de magnitud del eje Y.

Luego de este análisis sobre cómo crearíamos las instancias, medimos los tiempos de ejecución para cada tipo de grafo en cada algoritmo y realizamos ajustes para corroborar que se correspondiera con la complejidad teórica (Figs 2 a 5).

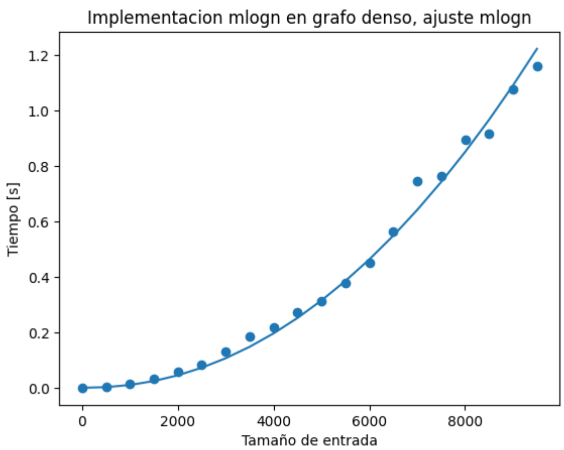

Fig 2: Ajuste del tipo $a * n^2 * log(n)$ *realizado a los tiempos obtenidos con la implementación con min-heap en grafos densos.*

Fig 3: Ajuste del tipo $a * n * log(n)$ *realizado a los tiempos obtenidos con la implementación con min-heap en grafos ralos.*

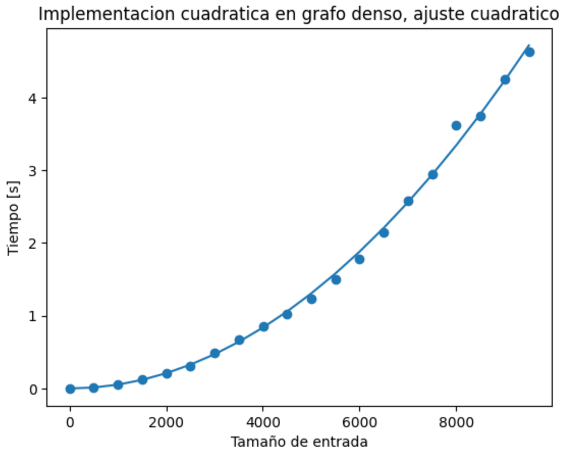

Fig 4: Ajuste del tipo $a * n^2$ realizado a los tiempos obtenidos con la implementación con array en grafos densos.

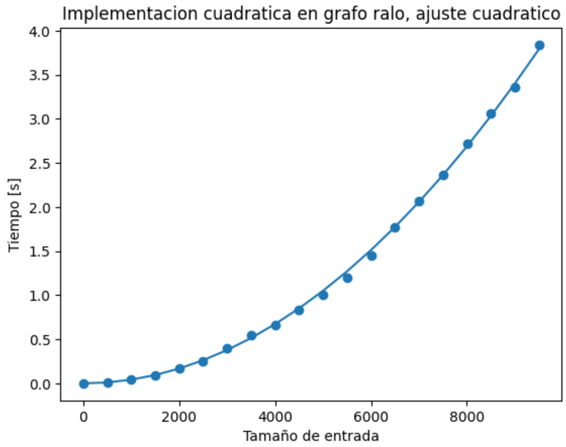

Fig 4: Ajuste del tipo $a * n^2$ realizado a los tiempos obtenidos con la implementación con array en grafos ralos.

Finalmente, comparamos ambas implementaciones para cada tipo de grafo (Figs 6 y 7). Esperábamos que para grafos densos, la implementación cuadrática sea más efectiva, ya que $O(m * log(n)) = O(n^2 * log(n))$ en estos casos. Sin embargo, en la práctica terminó siendo más efectivo el algoritmo con min-heap. También hay que tener en cuenta que $n \leq 10000$, por lo que $log(n) < 14$. Por lo tanto, $log(n)$ no llega a afectar mucho la complejidad.

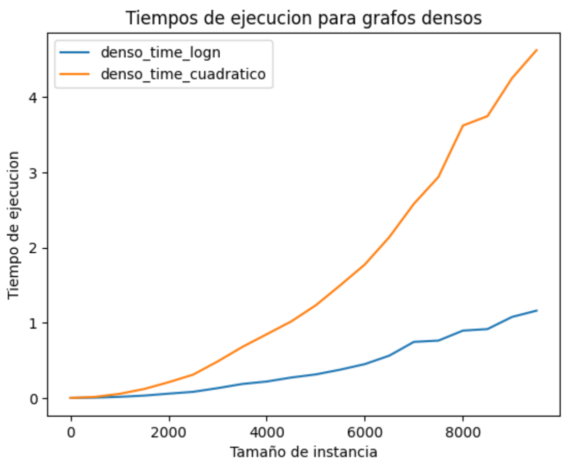

Fig 6: Comparación entre los tiempos de ejecución de cada implementación en grafos densos.

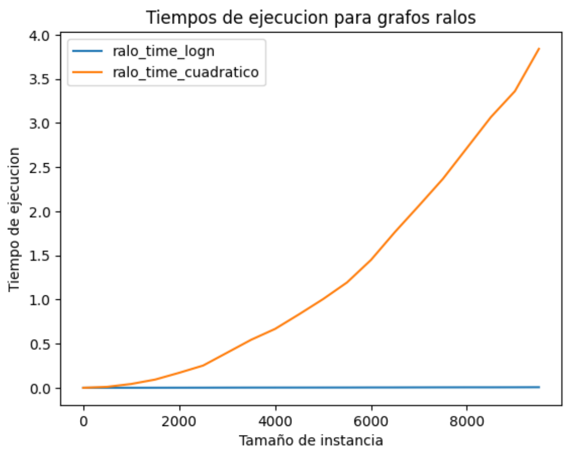

Fig 7: Comparación entre los tiempos de ejecución de cada implementación en grafos ralos.