DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Practico 3 - Camino Mínimo y Flujo Máximo

21 de junio de 2023 Algoritmos y Estructura de Datos III

Integrante	LU	Correo electrónico
Malena Sol, Alamo	1620/21	malusalamo@gmail.com
Laria, Jeremias	1329/21	jeremiaslaria7@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

Contents

1 Presentacion del problema

El mapa de una ciudad consiste en **n** puntos numerados y **m** calles unidireccionales que conectan dichos puntos. Se propone una lista de **k** calles bidireccionales como candidatas a ser construidas, con el fin de reducir la longitud del camino más corto entre dos puntos diferentes **s** y **t**.
El problema consta en elegir cual de las calles bidireccionales minimiza la longitud resultante del camino mas corto entre **s** y **t** (en el caso de que exista alguna), y calcular la longitud.

2 Algoritmo

2.1 Introducción

Para resolver el problema modelamos el mapa de la ciudad con un grafo dirigido pesado G=(E,V). Los vertices son los puntos, las aristas las calles unidireccionales, y el peso de la arista u $\rightarrow$ v la longitud del camino entre u y v.

El problema ahora se resume en encontrar cual de las aristas del conjunto *bid* que representan las calles a construir minimiza el camino entre el vertice **s** y el vertice **t**.

Se utilizará el concepto de arista *st-eficiente*. Decimos que una arista v $\rightarrow$ w es *st-eficiente* si pertenece a algun camino minimo de s a t.

2.2 Algoritmo

Variables:
- **G** el digrafo pesado que representa el mapa de la ciudad
- **s y t** los puntos criticos
- **bid** el conjunto de aristas que representan las calles bidireccionales propuestas para construir
- **(u,v,long)** las aristas que pertenecen a *bid*, con u y v sus puntos y long la longitud de la misma
- **distancia_s** el vector de distancias resultante de correr dijkstra(G,s)
- **distancia_t** el vector de distancias resultante de correr dijkstra(G^T,t)

```
trafico(G,s,t,bid):
  dijkstra(G,s)
  G^T = revertirAristas(G)
  dijkstra(G^T,t)

  longitudMinima = distancia[t]
  for((u,v,long) ∈ bid)
    st = min(distancia_s[u]+long+distancia_t[v],distancia_s[v]+long+distancia_t[u])
    longitudMinima=min(st,longitudMinima)

  if(longitudMinima = ∞)
    retorno 'No hay camino disponible de s a t'
  else
    retorno longitudMinima
```

3 Explicación

3.1 Propiedad de aristas *st-eficientes*

PROPIEDAD: Si definimos c:E(D) $\rightarrow \mathbb{N}$ el costo de una arista y d:V(D) $\rightarrow$ V(D) el peso del camino minimo entre dos vertices.

u $\rightarrow$ v es *st-eficiente* $\iff$ d(s,t)=d(s,u)+c(u$\rightarrow$v)+d(v,t)

DEMOSTRACIÓN
Notacion: Llamamos cam(u,v) al camino minimo entre u y v.

⇒)
d(s,t) $= \sum_e c(e)\ \forall e \in cam(s,t) =_1 \sum_e c(e)\ \forall e \in cam(s,u) + \sum_e c(e)\ \forall e \in cam(u,t)$
$=_2 \sum_e c(e)\ \forall e \in cam(s,u) + c(u,v) + \sum_e c(e)\ \forall e \in cam(v,t) \Rightarrow d(s,t) = d(s,u) + c(u,v) + d(v,t)$

1. Por PROP todo subcamino de un camino minimo es camino minimo.
2. Porque u → v es *st-eficiente*

⇐)

d(s,t)=d(s,u)+c(u,v)+d(v,t) ⇒ el camino minimo entre s y t tiene la misma longitud que el camino cam(s,u) ∪ u → v ∪ cam(v,t) ⇒ el camino cam(s,u) ∪ (u,v) ∪ cam(v,t) es camino minimo de s a t ⇒ u → v es *st-eficiente*

■

3.2 Justificacion

Tenemos como objetivo determinar si alguna de las calles propuestas para construir minimiza el camino entre s y t. Podemos utilizar la propiedad anterior para afirmar que esto es equivalente a verificar para cada arista u → v ∈ bid, si d(s,t)>d(s,u)+c(u,v)+d(v,t) o si d(s,t)>d(s,v)+c(v,u)+d(u,t) (debido a que son bidireccionales), siendo d(.,.) la longitud del camino minimo entre dos puntos en las calles actuales. Si la desigualdad se cumple, entonces u → v es *st-eficiente* y al construir su calle bidireccional correspondiente se minimiza el camino existente entre ambos puntos.

Como primer paso corremos dijkstra(G,s), almacenando todas las distancias actuales en la ciudad, incluyendo la distancia del camino que queremos minimizar.
Luego, revertimos las aritas existentes en G para crear G^T. Este paso es importante para la eficiencia del algoritmo. Esto se debe a que para poder verificar para cada arista u → v ∈ bid si es *st-eficiente*, necesitamos la distancia del camino minimo entre u y t. Para eso podriamos calcular dijkstra(u,t) para cada una de las aristas que pertenecen a bid, pero la complejidad seria O(|*bid*|)*O(dijkstra)[1]. En lugar de eso, podemos armar el grafo G^T en tiempo lineal y calcular dijkstra(G^T,t). El mismo nos retorna las distancias de todos los nodos a t, y tiene una complejidad de O(dijkstra).

Una vez calculados dijkstra(G,s) y dijkstra(G^T,t), verificamos para cada arista u → v ∈ bid si es *st-eficiente* y nos guardamos el minimo de los costos de las mismas.

En el caso de que ninguna arista en bid sea *st-eficiente*, retornamos la distancia actual de s a t. En el caso de que hayamos encontrado alguna arista *st-eficiente*, retornamos la longitud del camino si construimos esa nueva calle. Y en el caso de que no haya camino entre s y t, retornamos un mensaje de error.

[1]O(dijkstra) es equivalente a O(m $\log_n$) para grafos ralos u O(n^2) para grafos densos.

4 Estudio empírico

4.1 Implementacion Dijkstra

La mejor implementación para este problema depende de la cantidad de aristas (calles unidireccionales) que posea el mapa, ya que el cálculo del camino mínimo entre dos puntos se encuentra en la implementación utilizando el algoritmo Dijkstra.

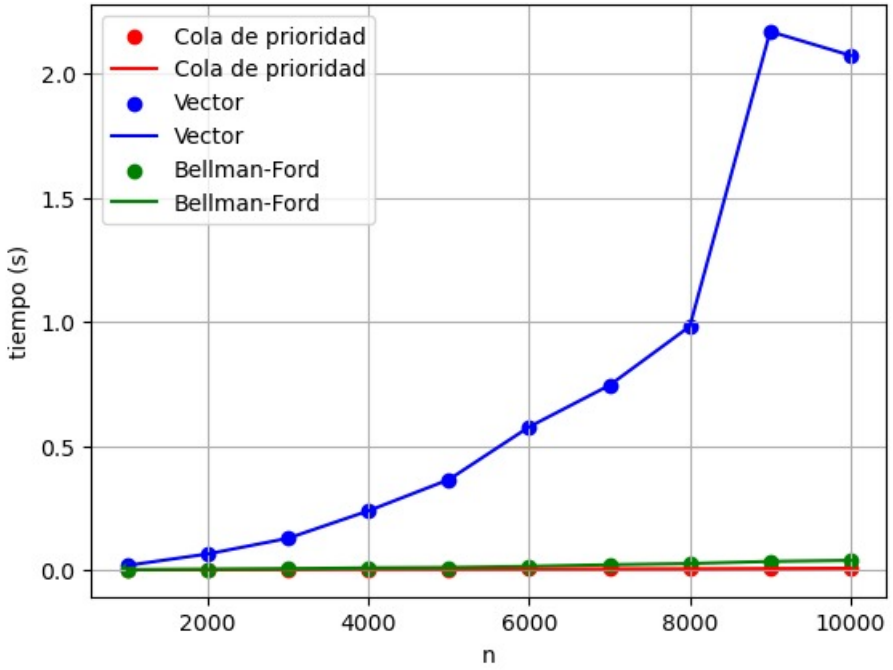
El algoritmo de Dijkstra cuenta con una complejidad de O(m $\log_n$) para grafos ralos u O(n^2) para grafos densos. Si la cantidad de calles es aproximadamente la cantidad de puntos o es menor, entonces el grafo G=(E,V) que modela el mapa cumple $|E| << n$. En ese caso, conviene la implementacion de Dijkstra O(m $\log_n$).
Si en cambio la cantidad de calles es aproximadamente el cuadrado de la cantidad de puntos, entonces el grafo G=(E,V) que modela el mapa cumple $|E| \approx O(n^2)$. Luego, conviene la implementacion de Dijkstra O(n^2).
La diferencia entre ambas implementaciones se basa en la elección de la estructura.

4.2 Bellman-Ford

Por otro lado, Bellman-Ford cuenta con una complejidad de O(mn). Esto se hace complejo cuando $|E| \approx n^2$, donde pasamos a tener una complejidad de O(n^3)

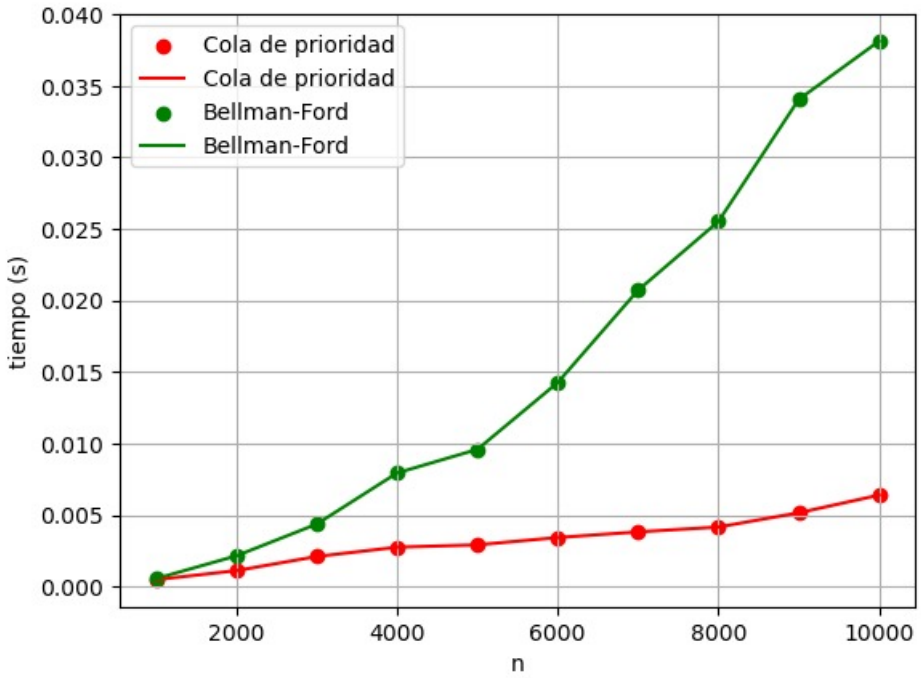
4.3 Experimentación

Realizamos una comparación empírica entre el algoritmo usando ambas implementaciones de Dijkstra y usando Bellman-Ford.



Se puede observar la diferencia entre la implementación usando un vector y usando cola de prioridad o con Bellman-Ford. Como sospechabamos, la implementacion utilizando vector es ampliamente superior, debido a que tiene complejidad $O(n^2)$.

Podemos comparar especificamente la diferencia entre Dijkstra y Bellman-Ford



De esta manera podemos observar que hacer uso de Dijkstra nos brinda mejores resultados que Bellman-Ford. Esto concuerda con la teoria: Bellman Ford $\in O(mn)$ y Dijkstra $\in O(mlog_n)$