Práctica Recorridos

Tomás Felipe Melli

Noviembre 2024

Índice

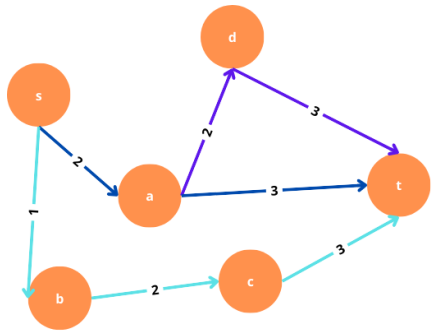
1 Algoritmo de Dijkstra

1.1 Ejercicio 1

Dado un dígrafo D con pesos $c : E(D) \to N$ y dos vértices s y t, decimos que una arista $v \to w$ es *st-eficiente* cuando $v \to w$ pertenece a algún camino mínimo de s a t. Sea $d(\cdot, \cdot)$ la función que indica el peso de un camino mínimo entre dos vértices.

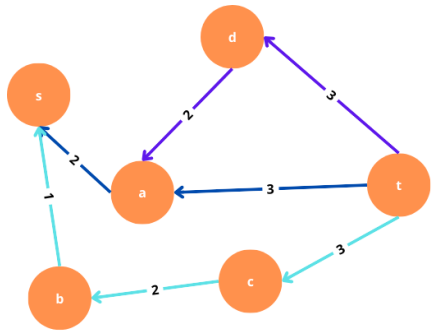
1. Demostrar que $v \to w$ es *st-eficiente* si y sólo si $d(s, v) + c(v \to w) + d(w, t) = d(s, t)$.
2. Usando el inciso anterior, proponga un algoritmo eficiente que encuentre el mínimo de los caminos entre s y t que no use aristas *st-eficientes*. Si dicho camino no existe, el algoritmo retorna $\perp$.

1. $\implies$)
   Sabemos que si $v \to w$ es *st-eficiente*, entonces $v \to w$ pertenece a algún camino mínimo de s a t. Por el **principio de optimalidad de Bellman** (cualquier subcamino $P_{u'v'} \subseteq P_{uv}$ es a su vez mínimo) podemos decir entonces que el camino que va de s a v es un camino mínimo. Análogamente el camino que va de w a t y por tanto podemos representarlos como : $c(P_{sv}) = dist(s, v)$ y análogamente $c(P_{wt}) = dist(w, t)$. En particular, como $v \to w$ pertenece a algún camino, $c(P_{st}) = dist(s, t) = dist(s, v) + c(v \to w) + dist(w, t)$ como se quería probar.
   $\impliedby$)
   $dist(s, t) = dist(s, v) + c(v \to w) + dist(w, t)$ como vale esto, queremos probar que efectivamente $v \to w$ es *st-eficiente*. Por contrarrecíproco vamos a probarlo. Si $v \to w$ no es *st-eficiente* quiere decir que no pertenece a ningún camino mínimo de s a t. Por tanto, tomamos $c(P_{sv}) = \sum_{e \in P_{sv}} c(e)$ y $c(P_{wt}) = \sum_{e \in P_{wt}} c(e)$ y el costo de la arista $c(v \to w)$ entonces podemos definir el recorrido de s a t como : $c(P_{st}) = \sum_{e \in P_{st}} c(e) = c(P_{sv}) + c(v \to w) + c(P_{wt})$ pero como $v \to w$ NO es *st-eficiente* $\implies$ $dist(s, t)! = c(P_{sv}) + c(v \to w) + c(P_{wt})$ en particular este recorrido es **mayor estricto**. Con esto en mente, podemos concluir que $dist(s, t) = dist(s, v) + c(v \to w) + dist(w, t)$ $\implies$ $v \to w$ es *st-eficiente*.
2. El algoritmo es :
   (a) Corremos DKS desde s en G.



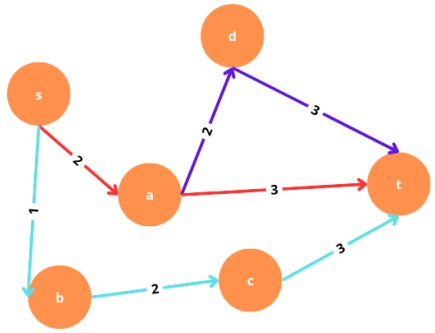
visitados = { s, b, a, c, d}
distancias = {0, 2, 1,3, 4, 5}

   (b) Procedemos a correr DKS desde t en $\overline{G}$.



visitados = {t, a,c, d, b, s}
distancias = {0, 3, 5, 3, 3, 5}

   (c) Marcamos las aristas *st-eficiente* como no elegibles y volvemos a correr DKS desde s en G.

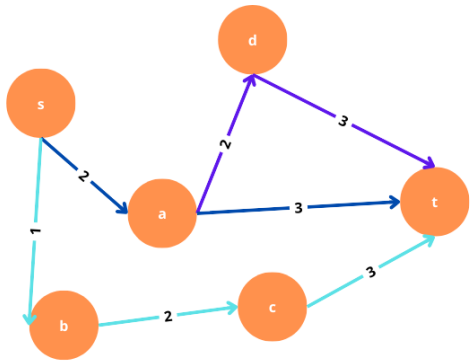


visitados = { s, b, c , t}
distancias = {0, inf, 1,3, inf, 6}

1.2 Ejercicio 2

Diseñar un algoritmo eficiente que, dado un dígrafo G con pesos no negativos, dos vértices s y t, y una cota c, determine una arista de peso máximo entre aquellas que se encuentran en algún recorrido de s a t cuyo peso (del recorrido, no de la arista) sea a lo sumo c. Demostrar que el algoritmo propuesto es correcto.

Para resolver este ejercicio vamos a tomar lo visto en el ejercicio anterior que $d(s,v) + c(v \to w) + d(w,t) = d(s,t)$. Ya sabemos que calcular $dist(s,v)$ lo logramos corriendo DKS desde s en G y $dist(w,t)$ lo logramos corriendo DKS desde t en $\overline{G}$ ahora sólo queda ver dentro de las aristas que cumplen : $c \geq dist(s,v_i) + c(e) + dist(v_j,t)$ / $e = (v_i,v_j) \in E(G)$ aquella que maximice. Eso lo podemos hacer ordenandolas de mayor a menor y vamos iterando. Veamos el ejemplo, tomamos C = 6 :



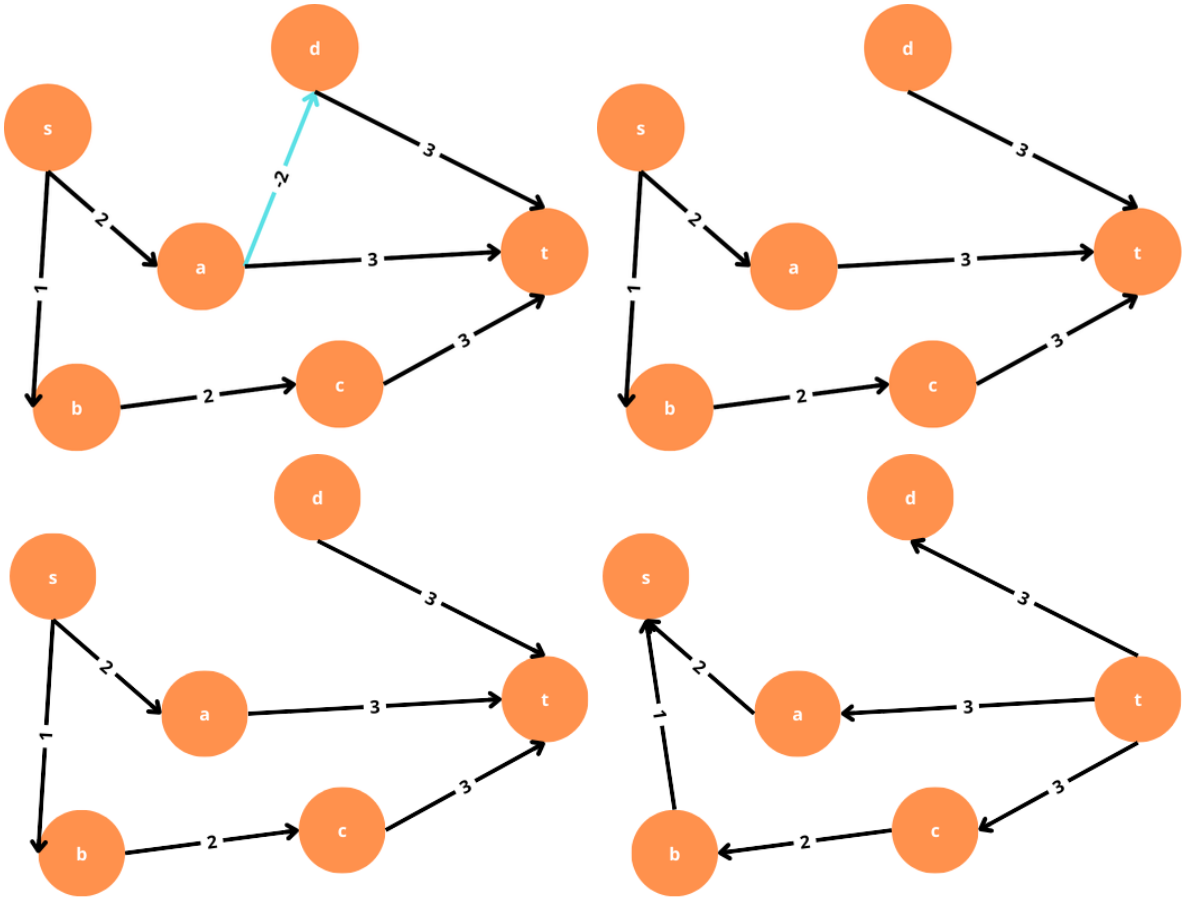
Aristas-ordenadas = { (c -> t), (a -> t),(d -> t),(b -> c), (a -> d), (s -> a),(s -> b) }
Chequamos : C >= dist(s,c) + c(c->t) + dist(t,t) = 3 + 3 + 0 = 6

La arista de mayor peso que se encuentra en algún recorrido es $c \to t$

1.3 Ejercicio 3

Diseñar un algoritmo eficiente que, dado un dígrafo pesado G y dos vértices s y t, determine el recorrido mínimo de s a t que pasa por a lo sumo una arista de peso negativo. Demostrar que el algoritmo propuesto es correcto.

Sabemos que Dijkstra no funciona bien en los casos en que hay aristas de pesos negativos. Esto se debe a que la lógica de la decisión es greedy y por tanto no se reevalúan vértices ya procesados en caso de que haya una forma de ir hacia él disminuyendo el costo (cuando existe aquella arista de peso negativo). Por ello, si queremos usar Dijkstra, vamos a tener que repensar este ejercicio de manera de tener un subgrafo de aquél que tiene la arista negativa y laburar sobre él y al final comparar los resultados : $dist(s,v_i) + c(e) + dist(v_j,t)$ = camino mínimo sin la arista negativa comparado con $dist(s,v_i) + c(e') + dist(v_j,t)$ = camino mínimo con la arista de peso negativo.



Como la arista de peso negativo es $a \to d$, queremos comparar en caso de que exista $dist(s,a)+c(a \to d)+dist(d,t)$ con el valor del camino mínimo obtenido $= 5$. Entonces, $dist(s,a) + c(a \to d) + dist(d,t) = 2 - 2 + 3 = 3 <$ camino mínimo obtenido $= 5$.

1.4 Ejercicio 4

Sea G un dígrafo con pesos positivos que tiene dos vértices especiales s y t. Para una arista $e \notin E(G)$ con peso positivo, definimos $G + e$ como el dígrafo que se obtiene de agregar e a G. Decimos que e mejora el camino de s a t cuando $d_G(s,t) > d_{G+e}(s,t)$. Diseñar un algoritmo eficiente que, dado un grafo G y un conjunto de aristas $E \notin E(G)$ con pesos positivos, determine cuáles aristas de E mejoran el camino de s a t en G. Demostrar que el algoritmo es correcto.

La idea para resolverlo es : corremos Dijkstra en G para capturar el camino mínimo que vamos a usar de referencia y luego en bucle ...

- Insertamos la arista
- Corremos Dijkstra
- Vemos si mejora o no
- La sacamos
- Repeat

La complijidad es $O(|E'| \times (|E| + |V|)logV)$ cuando $|E| \notin \Omega(|V|^2)$ esto es porque ejecutamos por cada arista en el conjunto de las potenciales optimizadoras Dijkstra y ese es el tiempo que toma Dijkstra en grafos no densos. En caso en que $|E| \in \Omega(|V|^2)$ se nos va a $O(|E'| \times |V|^2 logV)$.

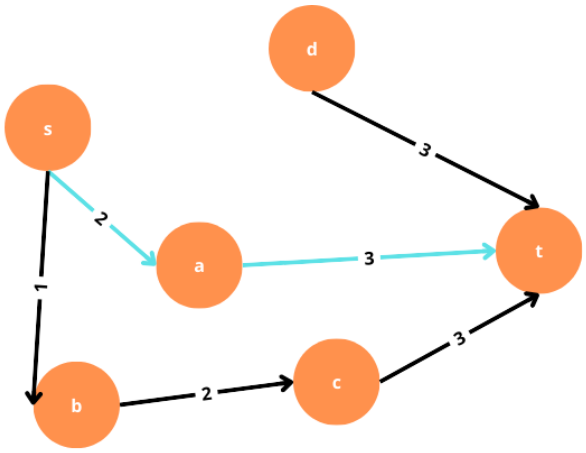
1.5 Ejercicio 5

Sea G un dígrafo con pesos positivos que tiene dos vértices especiales s y t. Decimos que una arista $e \in E(G)$ es crítica para s y t cuando $d_G(s,t) < d_{G-e}(s,t)$. Diseñar un algoritmo eficiente que, dado G, determine las aristas de G que son críticas para s y t. Demostrar que el algoritmo es correcto.

Sugerencia: Pensar en el subgrafo P de G formado por las aristas de caminos mínimos de G (el *grafo de caminos mínimos*).

$$e \in E(G) \text{ es crítica para s y t } \iff d_G(s,t) < d_{G-e}(s,t)$$

Por tanto $e \in dist_G(s,t)$, esto nos permite capturar el camino como subgrafo de G como sigue :



Las aristas del camino P serán candidatas a chequear $s \to a$ y $a \to t$. El chequeo es, sacar la arista, correr Dijkstra y hacer la comparación $d_G(s,t) < d_{G-e}(s,t)$, si se cumple, entonces la marcamos como **crítica**.

2 Bellman-Ford

2.1 Ejericio 7

Para organizar el tráfico, la ciudad de Ciclos Positivos ha decidido implementar las cabinas de peaje inverso. La idea de estas cabinas es incentivar la circulación de vehículos por caminos alternativos, estableciendo un monto que se le paga a le conductore de un vehículo cuando pasa por la cabina. Estas cabinas inversas se suman a las cabinas regulares, donde le conductore paga por pasar por la cabina. La ciudad sabe que estas nuevas cabinas pueden dar lugar al negocio del *ciclo puré*, que consiste en transitar eternamente por la ciudad a fin de obtener una ganancia que tienda a infinito. Para evitar esta situación, que genera costos y tráfico adicional, lo cual será aprovechado para desgastar a la administración ante la opinión pública, la ciudad quiere evaluar distintas alternativas antes de llevar las cabinas inversas a la práctica.

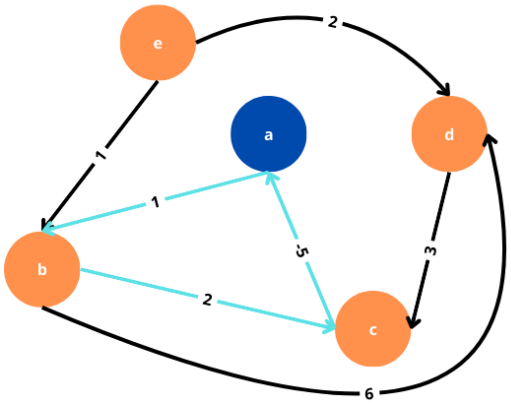
a. Modelar el problema de determinar si la ciudad permite el negocio del ciclo puré cuando el costo de transitar por cada cabina i de peaje es c_i ($c_i < 0$ si la cabina es inversa) y el costo que cuesta viajar de forma directa de cada cabina i a cada cabina j es $c_{ij} > 0$.

b. Dar un algoritmo para resolver el problema del inciso anterior, indicando su complejidad temporal.

El sistema arrojó que ninguna de las configuraciones deseadas para desincentivar el tráfico evita el negocio de los ciclos puré. Desafortunadamente, el plan se filtró a la prensa y comenzaron las peleas mediáticas. A fin de obtener cierto rédito, desde el departamento de marketing sugieren transformar la idea de cabinas inversas en cabinas mixtas. Cuando un vehículo pasa por una cabina mixta, se le paga a le conductore si se le cobró a le conductore en la cabina anterior; caso contrario, le conductore paga. Obviamente, desde marketing sugieren que se le pague a le conductore cuando la cabina mixta sea la primera cabina recorrida para bajar los malos humores.

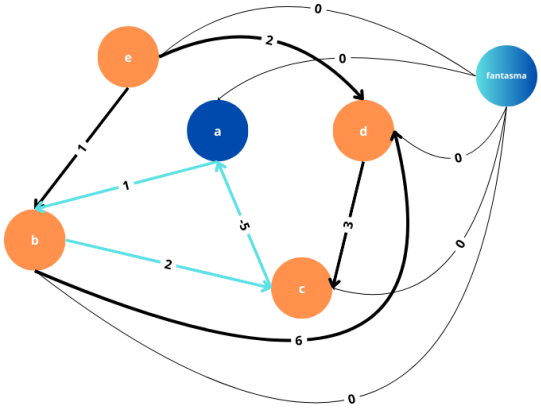
c. Modelar el problema de determinar si la ciudad permite el negocio de los ciclos puré cuando se aplica la nueva configuración para las cabinas. Además de la información utilizada para el problema original, ahora se conoce cuáles cabinas son mixtas: notar que el monto de cobro es c_i y el monto de pago es $-c_i$ para la cabina mixta i (con $c_i > 0$).

a. Nos piden modelar la siguiente situación :



Donde el ciclo negativo representa el **ciclo puré**, aquella situación que generará pérdidas para la administración, ya que los conductores circularían siempre por allí. Observemos que la **cabina inversa es a**, si se puede llegar a ella, el costo del peaje debe ser estrictamente menor a cero, en contraposición a las cabinas regulares que será estrictamente positivo el costo de transitar por ellas.

b. El algortimo utilizado será Bellman Ford para que detecte si para cierto modelado de la ciudad con el grafo G existe ciclo puré. La modificación a contemplar será el uso del **nodo fantasma** ya que por definición **el problema de recorridos mínimos se indefine $\iff$ la arista de peso negativo forma un ciclo negativo y es alcanzable desde el origen**, el vértice fantasma nos garantiza que **si existe ciclo negativo en G será alcanzable desde sí**.



Ahora sí, corremos Bellman-Ford enraizado en *fantasma*. La complejidad para grafos en que $|E| \in \Omega(|V|^2)$ se nos va a $O(|V|^3)$, sino sería la normal de Bellman-Ford $O(|E| \times |V|)$.

c. **consultar**

3 Algoritmo de Floyd-Warshall

3.1 Ejercicio 13

Decimos que una matriz cuadrada, simétrica y positiva $M \in N^2$ es de Floyd-Warshall (FW) si existe un grafo G tal que M es el resultado de aplicar FW a G. Describir un algoritmo para decidir si una matriz M es FW. En caso afirmativo, el algoritmo debe retornar un grafo G con la mínima cantidad de aristas posibles tal que el resultado de FW sobre G sea M. En caso negativo, el algoritmo debe retornar alguna evidencia que pruebe que M no es FW.

Supongamos la siguiente matriz cuadrada, simétrica, positiva:

	A	B	C	D	E
A	0	4	5	5	7
B	4	0	1	4	6
C	5	1	0	3	5
D	5	4	3	0	2
E	7	6	5	2	0

Qué propiedades cumple la matriz de FW ?
Cuando existen ciclos negativos en el grafo, el asunto se refleja en la diagonal de la matriz de distancias, en los caminos de un nodo a sí mismo. Si esta distancia es negativa, es decir,

$$\text{dist}[i][i] < 0,$$

estamos en presencia de un ciclo negativo. Esto se puede utilizar para detectar ciclos negativos o para concluir que no existe una solución bien definida para el problema.

a) Podríamos chequear si la diagonal tiene valores negativos (en dicho caso no tenemos solución definida del problema y como consecuencia no es una matriz FW). Nos dicen que es positiva : por tanto chequear esto no debería ser necesario.

Por propiedad de camino mínimo, vale la propiedad de que un camino de $i \to j$ tiene menor o igual longitud que ir de $i \to j$ pasando por cualquier nodo intermedio k

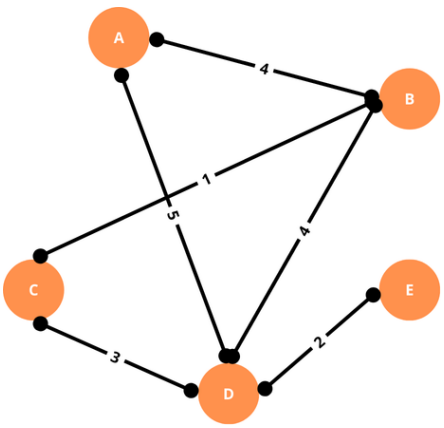
$$\text{dist}[i][j] \leq \text{dist}[i][k] + \text{dist}[k][j],$$

b) Podríamos chequear que se cumpla esta propiedad, si hay alguna entrada que no cumpla cualquiera de las propiedades anteriores, concluimos que la matriz no es FW y como nos piden evidencia, devolvemos la entrada que no cumple y los valores que demuestran que la desigualdad no vale.

En caso de encontrar una matriz FW, no podemos agregar todas las aristas $i \to j$ que sería lo más intuitivo porque nos piden mínima cantidad posible. Por ello, la idea es agregarla $\iff$ no existe un k tal que la distancia entre i y j

$$\text{dist}[i][j] = \text{dist}[i][k] + \text{dist}[k][j],$$

Reconstruyendo :



3.2 Ejercicio 14

Dado un dígrafo D con pesos $c : E(D) \to R$ que no tiene ciclos de peso negativo, queremos encontrar la arista $v \to w$ que sea st-eficiente para la mayor cantidad de pares s y t. Proponer un algoritmo eficiente y *simple de programar* para resolver este problema. **Ayuda:** verificar que la propiedad del Ejercicio 1a también es cierta en este caso.

Recordemos qué significaba que una arista sea *st-eficiente* :
$v \to w$ es *st-eficiente* $\iff d(s,v) + c(v \to w) + d(w,t) = d(s,t)$.
En este ejercicio nos piden hallar la arista forme parte de la mayor cantidad de caminos mínimos entre los pares de vértices de cierto grafo. La ayuda nos dice que miremos la propiedad que acabamos de enunciar.

- $d(s,v)$ es básicamente mirar la longitud del camino que existe en la matriz de FW desde cualquier vértice en G hasta v (el vértice cola de la arista que estamos mirando).
- $c(v \to w)$ es el costo que tiene la arista que estamos analizando
- $d(w,t)$ es básicamente mirar la longitud del camino que existe en la matriz de FW desde el vértice w (cabeza de la arista que estamos mirando) hasta cualquier otro vértice en G.

a) Corremos FW

b) Para cada arista :

   - Llevamos cuenta de a cuántos de esos caminos mínimos pertenece.
     Si se cumple : $d(s,v) + c(v \to w) + d(w,t) = d(s,t) \implies$ *st-eficiente*
   - Si es la que maximiza la cantidad de caminos mínimos a los que perteneces, entonces la asignamos en res

c) Fin