Equipo 3

Proyecto Final EDA II

Nadia González Fernández Arrastre
Bryan Machin García C-212
Jose Alejandro Solís Fernández C-212

Facultad de Matemática y Computación
Universidad de La Habana

1. Problema 1

Sea una red de flujo $G =< V, E, c >$, con fuente y receptor $s, t \in V$ respectivamente y para toda arista $e \in E$ se cumple que $c(e) = 1$. Se desea eliminar k aristas de la red de flujo de tal forma que el valor del flujo máximo disminuya lo más posible. Diseñe un algoritmo que encuentre un conjunto F de k aristas ($|F| = k$) de tal forma que el valor del flujo máximo en la red de flujo $G' =< V, E - F, c >$ es lo más pequeño posible. La complejidad temporal de su algoritmo debe ser de $O(|E| * k)$.

2. Problema 2

Sean n ciudades, una de ellas la capital y m carreteras. Cada carretera conecta unidireccionalmente a un par de ciudades. Diseñe un algoritmo que devuelva una lista de carreteras a construir tal que luego de construidas sea posible llegar desde la capital al resto de las ciudades El tamaño de la lista tiene que ser lo menor posible incluso vacía en caso de que inicialmente todas las ciudades sean alcanzables desde la capital. La complejidad temporal del algoritmo debe ser $O(n + m)$.

2.1. Propuesta de Solución

Para hallar la menor cantidad de carreteras que deben ser construidas para llegar desde la capital al resto de las ciudades, modelemos el ejercicio como un grafo dirigido $G =< V, E >$. Sea V el conjunto de vértices formado por las n ciudades, incluida la capital, y E el conjunto de aristas, que representan las m carreteras, donde la arista $< n_1, n_2 > \in E$ si y solo si existe una carretera de la ciudad n_1 hasta n_2.
Luego, se hallan todas las componentes fuertemente conexas y se crea un grafo reducido de dichas componentes fuertemente conexas.
Después se hace $DFS\text{-}Visit$ tomando como raíz el vértice que representa a la componente fuertemente conexa a la cual pertenece la capital.
Cada vértice tiene la propiedad *in-degree* (cantidad de aristas que inciden sobre él).
Luego, todos los vértices que no forman parte del árbol creado por $DFS\text{-}Visit$, es decir, todos las componentes fuertemente conexas que no son alcanzables desde la componente fuertemente conexa que contiene a la capital. Se analizan los *in-degree* y por cada vértice con *in-degree* $= 0$ se añade la arista formada por el vértice capital, y cualquier vértice de la componente fuertemente conexa donde en el grafo de componentes tiene *in-degree* $= 0$.

2.2. Correctitud

Sea $G^{SCC} = \{V^{SCC}, E^{SCC}\}$ un grafo reducido de G. Cada vértice tiene la propiedad *in-degree*. Esta se actualiza en el vértice w cuando se descubre la arista $< v, w >$. Esto no modifica el funcionamiento del algoritmo.
Sin pérdida de generalidad, sea C_i la componente fuertemente conexa que contiene al vértice de la capital. No es necesario añadir aristas dentro de C_i ya que existe un camino del vértice capital al resto de los vértices de la componente.

Entonces se debe analizar si existe un camino que conecte un vértice de C_i con cualquier otro vértice de alguna componente C_j tal que $i \neq j$. Para ello, se hace $DFS\text{-}Visit$ en el grafo reducido G^{SCC}, tomando como raíz el vértice de la componente donde se encuentra la capital. Durante la ejecución, se actualiza una lista global que contiene todos los vértices visitados. Para cada vértice v visitado en este recorrido existe un camino desde la raíz (la capital) hasta v. Ningún vértice no visitado en el $DFS\text{-}Visit$ es alcanzable desde la capital.
Luego de este analisis, se hallan las aristas que deben ser añadir tal que exista un camino desde la capital a todos los vértices.
Demostremos que solo es necesario añadir aquellas aristas formadas por el vértice capital y cualquier vértice dentro de una componente fuertemente conexa C representada por vértices no visitados tal que $in\text{-}degree(C) = 0$.

Lema 1: Para cada vértice v de $in\text{-}degree > 0$ existe un camino que conecta a un vértice u de $in\text{-}degree = 0$ a v.

Demostración: Sea $c = v_0, v_1, ..., v_k$ el camino más largo que contiene al vértice v_k. Supongamos que $in\text{-}degree(v_0) > 0$, entonces existe un vértice w que cumple:

- $w \notin c \Rightarrow \exists c'$ tal que c' tiene mayor longitud que c. *Contradicción.*
- $w \in c \Rightarrow$ existe un ciclo en c $\Rightarrow v_0$ y w pertenecen a la misma componene fuertemete conexa. *Contradicción*

Por lo tanto, añadiendo las aristas que van desde el vértice que contiene a la capital a los vétices que tiene $in\text{-}degree = 0$, la capital quedaría conectada con todas las ciudades.
Demostremos que se han añadido la cantidad mínima de aristas necesarias. Sea $< u, v >$ una de las aristas añadidas. Por la forma en que se añadió esta arista, v tiene $in\text{-}degree = 1$. Al quitarla, v tendría $in\text{-}degree = 0$ y por lo antes expuesto queda desconectado de la capital.

2.3. Pseudocódigo:

Algorithm 1 Carreteras necesarias

```
1: G ← BuildGraph < V, E >          ▷ grafo formado por las n ciudades y m carreteras
2: G' ← Strongly-Connected-Components(G)
3: G^SCC ← ReducedComponentGraph                                   ▷ grafo reducido
4: visited ← DFS-Visit(G^SCC, capital)
5: edges ← list
6: for each vertex in G^SCC do
7:     if not visited[vertex] and vertex.in-degree == 0 then
8:         edges.add(<capital,vertex>)
9:     end if
10: end for
11: return edges
```

2.4. Análisis de la complejidad temporal

Se construyen los grafos G, G' y G^{SCC} utilizando listas de adyacencia, lo cual tiene un costo $O(V + E)$. La complejidad temporal en la construcción del grafo G^{SCC} no se ve afectada por la actualización de la propiedad *in-degree*, ya que es $O(1)$. La ejecución del $DFS\text{-}Visit$ es $O(V + E)$ porque se utiliza un grafo implementado en una lista de adyacencia. En las líneas $6 - 9$ se recorre a lo sumo una vez cada vértice del grafo. Por lo tanto, el ciclo es $O(V)$. Luego, por regla de la suma, la complejidad temporal del algoritmo es $O(V + E)$.

3. Problema 3

Sea $G =< V, E >$ un grafo dirigido y ponderado, $s \in V(G)$, $t \in V(G)$, $P =< e_i, w_i >$ donde $e_i \in E(G)$, $w_i \in N$, y $0 \leq k \leq |P|$. Se desea calcular el camino de costo mínimo de s a t en G pudiendo cambiar a lo sumo k aristas del grafo G por aristas del conjunto P. La complejidad temporal de su algoritmo debe ser $O(k|E|log|V|)$.

3.1. Propuesta de Solución

Sean $G_1, G_2, \ldots, G_{(k+1)}$ grafos idénticos a G. Sea G' el grafo formado por la unión de los anteriores. Cada arco de P será colocado entre todo par consecutivo de estos grafos, de manera que, si el arco $< a, b >\in P$ entonces $< a_i, b_{(i+1)} >\in G$, para todo $1 \leq i \leq k$. Luego se calculan las distancias de los caminos de costo mínimo de s_1 hacia t_i para todo $1 \leq i \leq k$ y, la menor de estas, es la respuesta al problema.

3.2. Correctitud

Sea x_i el vértice $x \in V(G)$ en el i-ésimo de los grafos iguales. Un camino que vaya desde el vértice s_1 hacia el vértice t_i , pasa exactamente por i arcos de P debido a que no existen arcos que vayan desde el grafo $G_{(i+1)}$ hacia el grafo G_i. Sea j el índice para el cual el costo del camino de costo mínimo de s_1 a t_i, para todo $1 \leq k \leq 1$, sea mínimo.
Dicho camino p, que va desde s_1 a t_j, no puede contener un par de vértices iguales pero que pertenezcan a un G_i distinto. O sea, v_i y $v_{(i+k)}$ no pertenecen a p simultáneamente para ningún $v \in G$, pues al saber que en cada G_i contamos con las mismas aristas, podemos asegurar que tomando el mismo recorrido que se tomó para llegar de $v_{(i+k)}$ a t_j, se puede llegar a un t_k a partir de v_i. Pero como todas las aristas tienen costo positivo, el costo del camino de s_1 a t_k es menor que el costo de p lo cual es imposible.
Luego, en este camino p tampoco existirán aristas que conecten a vértices iguales, pero de distintos grafos G_i, por lo que si a dicho camino pertenece una arista del conjunto P, no pertenecerá su arista homóloga del grafo original.
Por tanto, al hallar el camino de costo mínimo que va desde s_1 hacia uno de los t_i se sabe que a lo sumo se utilizan k aristas del conjunto P y que de utilizarse estas, es porque fueron intercambiadas por las originales.
Por último, como se calculan los caminos de costo mínimo desde s_1 hacia todos los t_i, se garantiza que todas las $j-$combinaciones de aristas a intercambiar, para todo $j \leq k$, fueron comparadas por lo que esta solución es correcta.

3.3. Pseudocódigo

```
1: n ← |V|
2: G' ← vacío de tamaño n(k + 1)                    ▷ Construcción del grafo
3: for do i = 0 to k do
4:     for do u, v ∈ E(G) do
5:         G'[i * n + u].append(i * n + v)
6:     end for
7: end for                                           ▷ Dijkstra
8: d ← Dijkstra(G', s)                               ▷ Encontrar la respuesta
9: return min_{0≤i≤k} d[i * n + t]
```

3.4. Análisis de la complejidad temporal

Construir G' es $O(|V'| + |E'|)$. Aplicar Dijkstra es $O(|E'|log|V'|)$.
Ya que cada vértice del grafo original se repite en cada uno de los $(k+1)$ grafos idénticos sabemos que $|V'| = (k+1)|V|$.
Por cada uno de estos $(k+1)$ grafos idénticos se repiten también las aristas del grafo original y, además entre todo par de grafos consecutivos se encuentran las aristas de P que los enlaza por lo que $|E'| = (k+1)|E| + k|P|$. Como sabemos que $|P| \leq |E|$ entonces $|E"| \leq (2k+1)|E|$
Luego construir G' es $O((k+1)|V| + (2k+1)|E|) = O(k(|V| + |E|))$ y aplicar Dijkstra es $O((2k+1)|E|log(k+1)|V|) = O(k|E|log|V|)$.
Por la regla de la suma la complejidad temporal del algoritmo es $O(k|E|log|V|)$.

4. Problema 4

Sea $< T = V >$ un árbol, con raź $r \in V$, que se comporta como un circuito lógico. Cada hoja (excluyendo la raíz) representa una entrada del circuito y el resto de los vértices son elementos lógicos, incluyendo la raíz que es la única salida del circuito. Las entradas reciben un bit de información y la salida retorna un bit. Hay 4 tipos de elementos lógicos *AND* (2 entradas), emphOR (2 entradas), emphXOR (2 entradas) y emphNOT (1 entrada). Los elementos lógicos toman los valores de sus descendientes directos (las entradas) y devuelven el resultado de su operación. Se conoce la estructura de T , el tipo de elemento lógico en sus vértices, la configuración inicial de los bits en las entradas y que hay exactamente una entrada rota. Para arreglar dicha entrada hay que cambiar el valor en ella. Como se desconoce la ubicación de la entrada rota se requiere saber para cada entrada I cuál sería la salida si cambiara el valor de I. Diseñe un algoritmo que resuelva esta problemática con complejidad temporal $O(|V|)$.