UNIVERSIDAD SIMÓN BOLÍVAR
DEPARTAMENTO DE COMPUTACIÓN Y
TECNOLOGÍA DE INFORMACIÓN

PROBLEMAS DE PROGRAMACIÓN

Elaborado por:
Wilmer Bandres
Juan A. Escalante

Sartenejas, 23 de enero de 2015

1. Problema 1:

   (a) Descripción del algoritmo implementado:
   El problema nos pide buscar un bosque mínimo cobertor con M arboles.

   El algoritmo utilizado para la solución del problema se basa en el algoritmo de Kruskal para encontrar un Árbol mínimo cobertor, en donde se modela un grafo cuyos nodos son los puntos dados por el problema y cuyos arcos son todas las posibles conexiones entre estos nodos.

   Cada arco tiene un costo dependiendo de si su distancia es mayor a R (la variable dada) o no. Si la distancia entre un par de nodos excede R, entonces el peso de ese arco es de V-veces el valor de la distancia entre esos nodos y en caso contrario el peso de ese arco es de U-veces el valor de la distancia entre los nodos. [1]

   Inicialmente cada nodo es un árbol. Luego, se va a ejecutar el algoritmo de kruskal para ir uniendo cada par de nodos que esten en componentes (arboles) distintas (ya que si estan en la misma componente no hace falta agregar ese arco para que haya una cionexión entre ellas), es decir que vamos a ir uniendo componentes y estas uniones se ejecutaran un máximo de N-M veces ya que al hacer N-M uniones nos quedan M componentes a las que asignamos 1 modem a cada una para que las oficinas de cada componente tenga conexión a internet.

[1] La distancia es la distancia euclideana.

(b) Pseudocodigo

Input: N, M, R, U, V, $\overrightarrow{X}$, $\overrightarrow{Y}$
Output: Una tupla indicando el costo regular y especial

1: $costoRegular \leftarrow 0$
2: $costoEspecial \leftarrow 0$
3: $arcos \leftarrow \{\}$
4: **for all** $x_i \in \overrightarrow{X}$ **do**
5: **for all** $x_j \in \overrightarrow{X}$ **do**
6: **if** $i \neq j$ **then**
7: $arcos \leftarrow arcos \cup ((x_i, y_i), (x_j, y_j))$
8: **end if**
9: **end for**
10: **end for**

11: $arcos \leftarrow ordenar\_arcos\_por\_precio(arcos)$
12: $M \leftarrow N - M$
13: **for all** $arco \in arcos$ **do**
14: **if** $N = 0$ **then**
15: *break*
16: **end if**
17: **if** *Los nodos del arco pertenecen a arboles distintos* **then**
18: *Unir los arboles de los nodos*
19: **if** *Distancia de los nodos* $\leq R$ **then**
20: $costoRegular \leftarrow costoRegular + costo(arco)$
21: **else**
22: $costoEspecial \leftarrow costoEspecial + costo(arco)$
23: **end if**
24: $M \leftarrow M - 1$
25: **end if**
26: **end for**
27: **return** $(costoRegular, costoEspecial)$

Algorithm 1: Problema 1

(c) Analisis de tiempo y espacio:

i. Tiempo: En el algoritmo primero se crean $E = \frac{V \times (V-1)}{2}$ arcos, se ordentan lo que toma un tiempo $O(E \times \log E)$, y por ultimo se recorren los lados en orden de costo y se van uniendo los arboles a los que pertenecen los nodos, que al estar implementado con una estructura de datos de disjoint-set union, el costo de unir dos arboles es de $\alpha(N, E)$, donde α es la inversa de la función de Ackerman, que crece muy lentamente y es casi constante para fines practicos,por lo tanto el tiempo total de corrida es:

$$O(E + E \times \log(E) + E \times \alpha(N, E)) = O(E \times \log E)$$

ii. Espacio: El espacio ocupado por el algoritmo esta acotado por $O(N^2)$, ya que esa es la cota de la cantidad de arcos que existen en el grafo previamente explicado y de resto solo hay que guardar los nodos dados como entrada que es O(N), por lo tanto el espacio total utilizado en el algoritmo viene dado por:

$$O(N^2 + N) = O(N^2).$$

2. Problema 2:

   (a) Descripción del algoritmo implementado:
   El algoritmo implementado para este problema hace uso de la estructura del algoritmo de kruskal (disjoint set) esta vez será utilizado para llevar un tipo de conteo muy específico y para unir árboles.

   Supongamos que en vez de ir reventando arcos (como lo índica el problema), vamos a ir uniendo árboles cada vez (cosntruyendo arcos). Es decir, el analisis utilizado empieza con todos los nodos en árboles distintos donde cada nodo es un árbol. Luego, en vez de ir respondiendo los queries en la manera dada por el problema, se responderan en forma inversa, y cada vez que el problema índica que se reviente un arco, en su lugar se unirán los árboles de los nodos de ese arco, ya que ya se respondieron todos los queries con la ausencia de ese arco, de lo contrario no se hubiese llegado a esa parte en el input del problema (en su orden inverso).

   Incialmente la cantidad de pares de nodos que estan en distintas componentes es $\frac{N \times (N-1)}{2}$. Cuando unimos dos arboles lo que se debe hacer es actualizar esta cuenta y restarle la siguiente cantidad:

   Digamos que K_1 es la cantidad de nodos en el árbol del primer nodo del arco.

   Digamos que K_2 es la cantidad de nodos en el árbol del segundo nodo del arco.

   Entonces el total de pares en componentes distintas al unir estos dos árboles se reduce en una cantidad de: $K_1 \times K_2$, ya que ahora los K_1 nodos y los K_2 nodos se encuentran en el mismo árbol.

   Para responder los queries en forma inversa se hace uso de dos pilas auxiliares y para la unión de los árboles se utiliza la estructura de disjoint-set union, asi como un arreglo que lleva el conteo de cuántos nodos hay en un árbol.

   Por último, aquellos arcos que no van a desaparecer nunca por la entrada del problema, son los primeros a empezar a unir ya que todos los queries poseen estos arcos, es decir que estos arcos tienen que ser los que estan en el tope de las pilas auxiliares a utilizar.

(b) Pseudocodigo

Input: N, C, $\overrightarrow{X}$, $\overrightarrow{Y}$
Output: Nada

1: $s1 \leftarrow newStack()$
2: $s2 \leftarrow newStack()$
3: $total \leftarrow \frac{N \times (N-1)}{2}$
4: **for** $0 \leq i \leq C$ **do**
5: $read(mandato)$
6: **if** $mandato = R$ **then**
7: $s1.push(1)$
8: read(arco)
9: $s2.push(arco)$
10: **else**
11: $s1.push(0)$
12: **end if**
13: **end for**
14: **for all** $0 \leq i \leq N-1$ *and* i *no esta en* $s2$ **do**
15: $s2.push(i)$
16: $s1.push(1)$
17: **end for**
18: **while** $s1$ *no este vacia* **do**
19: $tope \leftarrow s1.top()$
20: $s1.pop()$
21: **if** $tope = 0$ **then**
22: $print(total)$
23: **else**
24: $arco \leftarrow s2.top()$
25: $s2.pop()$
26: $total = total - UnirArbolesDeNodosDelArco(arco)$
27: **end if**
28: **end while**

Algorithm 2: Problema 2

Input: arco
Output: Cantidad de nodos unidos

1: $(nodo\_1, nodo\_2) \leftarrow arco$
2: $K_1 \leftarrow cantidadDeNodosEnArbol(nodo\_1)$
3: $K_2 \leftarrow cantidadDeNodosEnArbol(nodo\_2)$
4: *Se unen los arboles de* $nodo_1$ *y* $nodo_2$
5: *return* $K_1 \times K_2$

Algorithm 3: UnirArbolesDeNodosDelArco

(c) Analisis de tiempo y espacio:

i. Tiempo:
Cada inserción en cada una de las pilas es O(1), al igual que el pop de las mismas. Por otro lado, tenemos a lo sumo $\min(C, N-1)$ inserciones y pop's en la pila s1, y a lo sumo N-1 inserciones y pop's en la pila s2, y como por cada pop a la pila s2 se realiza una unión de árboles que tiene una complejidad de $\alpha(N, N)$ (donde *alpha* es la inversa de la funcion de Ackerman), entonces la complejidad en tiempo es la siguiente:

$$O(2 \times \max(C, N-1) + 2 \times (N-1) + \alpha(N, N) \times (N-1)) = O(\max(C, N-1) + \alpha(N, N) \times N)$$

ii. Espacio:
Dado que los elementos a almacenar son solamente los arcos, y los queries, y un arreglo donde se almacene el conteo de la cantidad de nodos por árbol y todos con una cota superior de N-1, entonces el espacio total requerido es de:

$$O(3 \times (N-1)) = O(N)$$

3. Problema 3:

   (a) Descripción del algoritmo implementado:

   El algoritmo implementado fue el algoritmo de activity scheduling, ya que es un algoritmo greedy que nos maximiza la cantidad de actividades a realiza. Lo que hace este algoritmo es ordenar las actividades por su tiempo de finalización e ir agarrando la siguiente actividad válida a partir a partir del ultimo tiempo escogido por el mismo, esto nos garantiza que la cantidad de actividades realizadas es máxima.

   (b) Pseudocodigo:

Input: $\overrightarrow{X}$, $\overrightarrow{Y}$
Output: Número máximo de actividades a realizar

1: $total \leftarrow 0$
2: $last\_activity \leftarrow -1$
3: $OrdenarActividadesPorMenorY(\overrightarrow{X}, \overrightarrow{Y})$
4: **for all** $x_i \in \overrightarrow{X}$ **do**
5: **if** $x_i \geq last\_activity$ **then**
6: $last\_activity \leftarrow y_i$
7: $total \leftarrow total + 1$
8: **end if**
9: **end for**
10: **return** *total*

Algorithm 4: Problema 3

   (c) Analisis de tiempo y espacio:

      i. Tiempo: El tiempo de este algoritmo vienedado por el tiempo que toma ordenar las actividades y luego recorrerlas es lineal en el numero de actividades. Por lo tanto si el número de actividades es N la complejidad en tiempo es de:

$$O(N\ \log(n) + N) = O(N\ \log(N))$$

      ii. Espacio: Dado que en el algoritmo solo tiene que guardar las coordenadas x's y y's de cada intervalo, el espacio esta acotado por:

$$O(2 \times N) = O(N)$$