Proyecto 1-Diseño de Algoritmos I

Fabio Castro 10-10132, Leopoldo Pimentel 06-40095

23/01/2015

1 Problema 1

1.1 Análisis de Complejidad

Para el analisis de este algoritmo, lo podemos dividir en varias subpartes. La primera parte, seria la lectura de cada una de las pruebas. Esta lectura, es de orden $O(n)$, que que presenta un ciclo de tamaño N La segunda parte es la creacion de cada uno de los arcos posibles Esta, creacion, es de orden $O(n^2)$, ya que recorre la lista en orden cuadratico para conseguir generar todos los arcos posibles Se usa un algorimo de ordenamiento que viene con el lenguaje. Este algoritmo su complejidad es de $O(nlogn)$ La tercera parte y mas compleja es una busqueda de las componentes conexas. Esta trabaja con una lista y su recorrido de orden $O(n)$ y adicionalmente, unas consultas. Y una busqueda en otra lista. Cada busqueda es de orden $O(componentes_conexas)$. Es decir que la busqueda es sobre un grafo mucho mas pequeño que N, por consiguiente, podriamos considerar que esta tercera parte es de orden $O(n)$

1.2 Explicación de la solución dada

Lo primero que se hizo fue una lectura de las posibles cantidad de pruebas Luego, por cada prueba, leemos las coordenadas de la oficina y la colocamos dentro de una lista en forma de pares ordenados. Luego, sacamos todos los posibles arcos que pueden haber dadas las oficionas leidas, cada arco, tiene entonces un peso o costo y esto es calculado por la distancia entre oficinas en un plano Luego, ordenamos la lista por el costo de los arcos. Y empezamos a agregar a una nueva lista. Nuestro grafo final. A dicho grafo solo vamos a agregar los primeros N arcos, hasta obtener un grafo, Este grafo, tiene que cumplir que solo existe una cantidad de componentes disjuntas como modem hay en la prueba efectuada. Para verificar si se agrega o no un arco, tenemos que verificar si una coordenada ya ha sido agregada a nuestro grafo. Si es asi, entonces estamos uniendo dos componentes que antes estaban desunidas. Pero, si los dos puntos ya estan en nuestro nuevo grafo, entonces quiere decir que estamos haciendo un arco ya formado o un circuito. Por consiguiente este tipo de puntos no es agregado.

Algoritmo 1 Calcular maxima cantidad de cable normal y cable especial para que sea mínimo en costo y todos los nodos se encuentren conectados

Entrada: C Conjunto de Pares que contienen las coordenadas de los nodos, M Maxima cantidad de metros a tarifa1, $T1$ Tarifa 1, $T2$ Tarifa 2, MO Cantidad de Modems

Salida: Par que contiene en la primera posicion la cantidad de cable normal, y en la segunda cantidad de cable especial.

1: entero $componentesconexas = C$.size() - M
2: $par < entero, entero > salida$
3: entero $tarifa = 0$
4: entero $costototalt1 = 0$
5: entero $costototalt2 = 0$
6: pila $q =$ **DistanciasYCostos**(C) {La cola esta ordena por menor costo}
7: $par < par < entero, entero >, entero > elementoe$
8: **para** $entero\ x = 0$; $entero < q.size()$ **y** $componentesconexas > 0$ **;** $x = x + 1$ **hacer**
9: $elemento = q.pop$
10: **si** $find(elemento.primero.primero)$ **diferente** $find(elemento.primero.segundo)$ **entonces**
11: $tarifa = elemento.segundo$
12: **si** $distancia(elemento.primero.primero, elemento.primero.segundo) > M$ **entonces**
13: $costototalt1 = costototalt1 + tarifa$
14: **si no**
15: $costototalt2 = costototalt2 + tarifa$
16: **fin si**
17: $componentesconexas = componentesconexas - 1$
18: **fin si**
19: **fin para**
20: **devolver** $< costototalt1, costototalt2 >$

2 Problema 2

2.1 Análisis de Complejidad

Este Algoritmo se parece mucho al primero, este tiene un while de orden $O(n)$, pero es depreciado porque este N representa la cantidad de pruebas y no deberia ser un numero muy grande. Despues, internamente exiten 3 ciclos, cada uno de orden $O(n)$ Cada uno de estos ciclos, son ciclos de lectura y de acomodacion de datos. Su complejidad no es muy grande para números pequeños, pero se podría complicar para números muy grandes o para pruebas con números muchos mas grandes. El siguiente ciclo, es un for externo que llama a una funcion Unir internamente El ciclo es de orden $O(n)$, luego la funcion de interna, tiene una llamada a otra funcion auxiliar que es recursiva. Esta funcion recursiva tambien tiene un comportamiendo de orden $O(n)$ ya que se sabe que nunca sera mayor que N Podriamos decir entonces, que este ciclo, y por consiguiente la funcion interna tienen orden $O(n^2)$, ya que hay un ciclo interno y otro externo que estan ciclando y uno depende el otro. Por ultimo, quedan dos ciclos, cada uno de orden $O(n)$, son ciclos de recorrido de los vectores. Tomando en consideracion todos estos ordenes internos a nuestro algoritmo, podemos decir que nuestro algoritmo se comporta en el peor caso como un algoritmo de orden $O(n^2)$

Algoritmo 2 Calcular cantidad de pares desconectados

Entrada: N Conjunto de Nodos, Q Lista de Querys, C Cantidad de Arcos Originales, G Lista de Arcos, H Arcos a Eliminar
Salida: Lista de respuesta de los Querys

```
1: entero total = C * (C - 1)/2
2: lista < entero > salida
3: para cada elemento ∈ G y elemento ∉ H hacer
4:    unir(elemento.primero, elemento.segundo)
5: fin para
6: para cada query ∈ L hacer
7:    si query.primero ==' Q' entonces
8:       salida.push(total)
9:    si no
10:      total−=unir(G[query.segundo.primero].primero ,
         G[query.segundo.primero].segundo)
11:   fin si
12: fin para{Recorre la lista en orden inverso}
13: devolver salida
```

2.2 Explicación de la solución dada

El problema fue resuelto usando la idea que hacer unión de conjuntos es menos cosotosa que eliminar arcos de un grafo. En base a esto, armamos el grafo con todos los nodos menos los que se iban a eliminar y contamos a partir de esto las cantidad de conexiones que causaría que este arco se colocase, así le restabamos al total de conexiones existentes en dicho momento y así según cada

query solicitado. Usamos una implementacion basada kruskal usando cojuntos disjuntos para obtener el resultado deseado.

3 Tercer problema

3.1 Análisis de Complejidad

Primera parte es una lectura de la cantidad de pruebas que tiene el caso. Esta primera parte es $O(n)$, ya que es un ciclo de tamaño N. Luego, tenemos para cada empleado un ciclo por el numero de actividades que este realizara. Este ciclo es de tamaño n, por consiguiente, el proceso es $O(n)$ Se usa un algorimo de ordenamiento que viene con el lenguaje. Este algoritmo su complejidad es de $O(nlogn)$ Por ultimo, se realiza un ciclo de tamaño N, este ciclo tiene una complejidad de $O(n)$ Como tenemos el primer ciclo por afuera, y todos los demas internos, podriamos pensar que entonces la complejidad del problema es de $O(n^2logN)$. Pero, al ser el primer ciclo un ciclo de iteraciones de pruebas, donde el numero no sera muy grande, podriamos pensar entonces que la complejidad de todo el algoritmo es la complejidad mas grande. Esta seria $O(nlogN)$

Algoritmo 3 Calcular *total*, que es el total de actividades que puede realizar el empleado

Entrada: C Cantidad de tareas, A Lista de tuplas de actividades
Salida: Entero que indica la cantidad maxima de actividades que puede realizar

```
1: entero ultimo = −1
2: entero total = 0
3: ordenar(A)
4: para cada elemento de A hacer
5:    si ultimo > elemento.primero entonces
6:       total = total + 1
7:    fin si
8:    ultimo = elemento.segundo
9: fin para
10: devolver total
```

3.2 Explicación de la solución dada

Primero leemos la cantidad de empleados que exite Luego, vemos cuales son las actividades que tiene dicho empleado, luego ordenamos las actividades del empleado de forma ascendete. Luego, vamos agregando actividades realizadas o posibles a un contador, mientras existan posiciones libres o espacios libres para hacer un trabajo. Haciendo esto, estamos usando la estrategia Greedy. Garantizando asi que sea la solucion mas optima al problema.