Laboratorio Nro. 4: Algoritmos voraces o codiciosos (Greedy Algorithms)

Santiago Castrillón Galvis
Universidad Eafit
Medellín, Colombia
scastrillg@eafit.edu.co

Luis Javier Palacio Mesa
Universidad Eafit
Medellín, Colombia
ljpalaciom@eafit.edu.co

Alejandro Arroyave Bedoya
Universidad Eafit
Medellín, Colombia
aarroyaveb@eafitedu.co

3) Simulacro de preguntas de sustentación de Proyectos

1. Para resolver el problema es necesario usar diversas listas, ya que estas nos permiten llevar un conteo de los vertices que hemos visitado, saber cuales vertices son los que hacen parte de la solucion, y poder guardar vertices sucesores para ser comparados y dar continuidad al camino, esto se basa en que la lista nos permite ir agregando los elementos uno a uno, y en determinado momento, mirar si algun elemento esta contenido dentro de esta. El algoritmo comienza designando un nodo deposito que será el inicio del camino y el final(objetivo al que queremos llegar), despues de esto el nodo deposito será nuestro nodo actual y lo agregamos a la lista que llevará el camino resultante. Luego de esto comenzamos a buscar el camino mediante un ciclo en el que tenemos una pareja menor, que estará compuesta por nuestro nodo actual y con un peso infinito, marcaremos el nodo actual como visitado y guardaremos sus sucesores en una lista. Si hay sucesores entonces compararemos cada uno de los pesos entre ellos y el nodo actual y seleccionaremos el que sea menor, de manera que nuestra pareja menor cambiara a estar compuesta por el sucesor y su peso con respecto al actual, a continuacion sumaremos este peso al acumulador que llevará la cuenta del peso total que tengael camino, cambiaremos nuestro nodo actual por el nodo sucesor que escogimos y lo agregaremos a la lista del camino. Este proceso lo repetiremos siempre que el nodo actual sea diferente que el nodo contenido en menor, pues si estos son iguales significa que ya visitamos todos los nodos, por lo que solo faltaria agregar al acumulador el peso desde el actual hacia el nodo deposito y retornar el camino contenido dentro de lista.
2. Debe cumplir que sea un grafo completo, debido a que si no lo fuera, podría darse el caso de que no se visiten todos los nodos, tambien, que al final no haya camino entre el ultimo nodo seleccionado y el primero, por lo que no podriamos volver al inicio y no completariamos el camino.
3. Se usan dos listas (ArrayList), una que representa las longitudes de rutas en la mañana y otra en la tarde. Se ordenan de menor a mayor con el método Collections.sort() y se busca asignar a cada conductor al principio la ruta máxima en la mañana que no haya sido asignada en el pasado, y luego intentar asignarle la ruta máxima no asignada en la tarde sin pasarse de d, en cuyo caso se le asigna la segunda más larga o así sucesivamente siempre evitando que se pase de d. Así se hace una mejor distribución de los turnos y rutas Y se minimiza el costo extra a pagar.

4.

```
public class ejercicioEnLinea {
  public static void main(String[] args) {
     BufferedReader lector = new BufferedReader(new InputStreamReader(System.in));
     ArrayList<Integer> qManana = new ArrayList<>();
     ArrayList<Integer> qTarde = new ArrayList<>();
     try {
        while (true) {
           String lineaPartida[] = lector.readLine().split(" ");
           int n = Integer.parseInt(lineaPartida[0]);
           int d = Integer.parseInt(lineaPartida[1]);
           int r = Integer.parseInt(lineaPartida[2]);
           if (n == 0) {
              break;
           }
           lineaPartida = lector.readLine().split(" ");
           for (String hora : lineaPartida) {
              qManana.add(Integer.parseInt(hora));
           }
           lineaPartida = lector.readLine().split(" ");
           for (String hora : lineaPartida) {
              qTarde.add(Integer.parseInt(hora));
           }
           int dineroExtra = 0;
           int acum;
           Collections.sort(qManana); nLog(n)
           Collections.sort(qTarde ); nLog(n)
           for (int i = 0; i < n; i++) { C1n
              acum = qManana.remove(qManana.size() - 1); C2n
              boolean entre = false; C3n
              int next = 0; C4n
              for (int j = qTarde.size() - 1; j >= 0; j--) { C5n*n
                 next = qTarde.get(j);C6n*n
                 if (acum + next <= d) {C7n*n
                    qTarde.remove(j); C8*n
                    entre = true; C9*n
                    break;
                 }
              }
              if (!entre) {
                 qTarde.remove(0);
                 acum += next - d;
                 dineroExtra += acum * r;
              }
           }
           System.out.println(dineroExtra);
        }
     } catch (Exception e) {
     }
  }
```

DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co

}
T(n) = 2nlogn + Cn + C'n*n
T(n) es O(2nlogn + Cn + C'n*n)
T(n) es O(C'n*n)
T(n) es O(n*n)
T(n) es O(n^2)
Esta complejidad no tiene en cuenta el tiempo que se toma ingresar los datos, sino la solución con los mismos.

5. La variable n se refiere a las n rutas del bus, n en la mañana, n en la tarde, y a los n conductores del bus.

4) Simulacro de Parcial

1. *i = j*
2. *min > adjacencyMatrix[element][i]*
3. A.

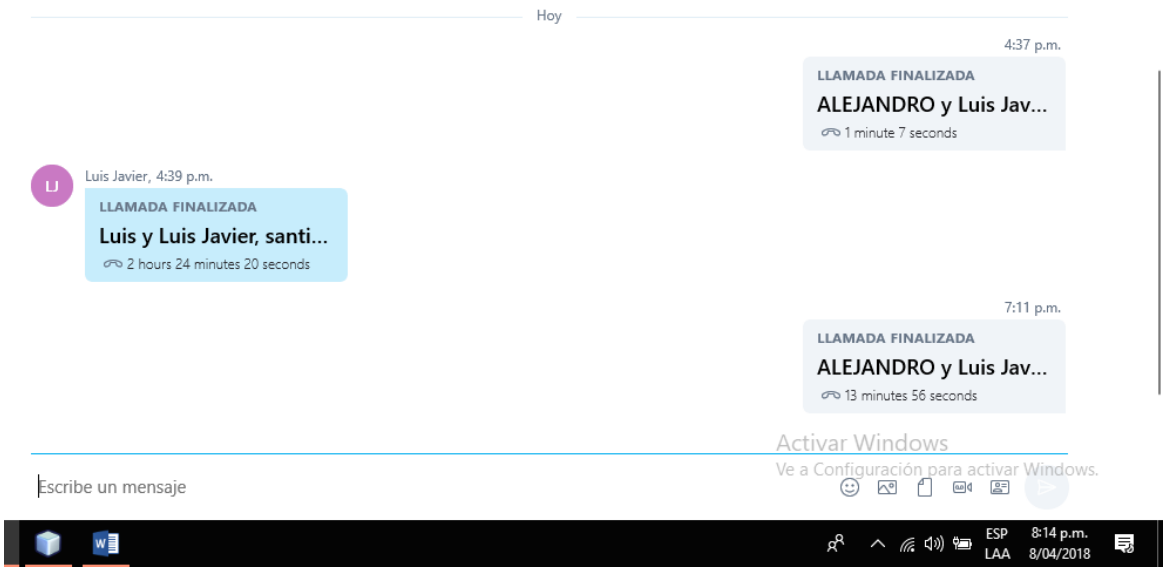
Paso	a	B	C	D	E	F	G	H
1	A	20,A	∞	80,A	∞	∞	90,A	∞
2	B	20,A	∞	80,A	∞	30,B	90,A	∞
3	F	*	40,F	70,F	∞	30,B	90,A	∞
4	C	*	40,F	50,C	∞	*	90,A	60,C
5	D	*	*	50,C	∞	*	70,D	60,C
6	H	*	*	*	∞	*	70,D	60,C
7	G	*	*	*	∞	*	70,D	*
8		*	*	*	∞	*	*	*

B. A,B,F,C,D,G.

6) Trabajo en Equipo y Progreso Gradual (Opcional)

a)

DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co



DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co