Laboratorio Nro. 4: Algoritmos voraces o codiciosos

Isabela Muriel Roldán
Universidad Eafit
Medellín, Colombia
imurielr@eafit.edu.co

Mateo Flórez Restrepo
Universidad Eafit
Medellín, Colombia
mflorezr@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos

1. Para resolver el problema del ejercicio 1.1 utilizamos un algoritmo voraz, el cual calcula los sucesores del nodo actual y a partir de este se elige el siguiente nodo teniendo en cuanta cual es el arco que tiene menor costo hasta finalmente regresar al nodo inicial.
2. Para que el algoritmo voraz funcione el grafo debe tener al menos un circuito hamiltoniano, ya que de lo contrario el algoritmo no podría regresar al nodo inicial.
3. Para el ejercicio 2.1 utilizamos algoritmos voraces, los cuales consisten en elegir la opción más optima en cada uno de los pasos hasta llegar a la solución general más óptima. El algoritmo que implementamos para darle una solución óptima al problema funciona de la siguiente manera: Por entrada por teclado recibe primero tres parámetros: número de rutas, duración de la ruta y tarifa por hora (n, d, r); en la siguiente línea según el número de rutas n, se solicita la duración de cada ruta en la mañana en horas, y en la tercera línea lo mismo, pero con las rutas de la tarde. El algoritmo lo que hará será sumar las duraciones de cada ruta en la mañana y en la tarde, luego hallara la diferencia entre la duración total de las dos rutas y la duración total esperada(d), y de acuerdo a las horas que haya de más, se multiplicara el excedente de cada ruta por la tarifa por hora(r) y luego se sumaran los resultados de la tarifa excedente de las dos rutas, y se devuelve el resultado de la suma, hallando así la cantidad mínima que debe pagar la empresa, el algoritmo seguirá solicitando casos de prueba hasta que los tres primeros parámetros sean 0 (n=0, d=0, r=0) luego retornara 0.
4. **Complejidad:**

```
private static int ejercicio2()throws IOException{
   Scanner sc = new Scanner(System.in);                          //C1
   int n=0;                                                      //C2
   int d=0;                                                      //C3
   int r=0;                                                      //C4
    System.out.print(" ");                                       //C5
   while((n=sc.nextInt())!=0 &&(d=sc.nextInt())!=0 && (r=sc.nextInt())!=0){ //C6*m
    InputStreamReader isr = new InputStreamReader(System.in);              //C7*m
    BufferedReader br = new BufferedReader (isr);                          //C8*m

    String rutaM =br.readLine();                                 //C9*m
    String duracionRutaM[] = new String [n];                     //C10*m
    duracionRutaM= rutaM.split(" ");                             //C11*m
    String rutaT =br.readLine();                                 //C12*m
```

```
String duracionRutaT[] = new String [n];                    //C13*m
duracionRutaT= rutaT.split(" ");                            //C14*m
int duraciontotalM=0;                                       //C15*m
int duraciontotalT=0;                                       //C16*m
for(int i=0; i< n;i++){                                     //C17*m*n
    duraciontotalM+= Integer.parseInt(duracionRutaM[i]); //C18*m*n
    duraciontotalT+= Integer.parseInt(duracionRutaT[i]);  //C19*m*n
  }
duraciontotalM=(duraciontotalM-d)*r;                        //C20*m
duraciontotalT=(duraciontotalT-d)*r;                        //C21*m
System.out.println(duraciontotalM+duraciontotalT);          //C22*m
}
return 0;                                                   //C23
}
```

T(n)= C+C*m+C*m*n
T(n)=O(C+C*m+C*m*n)
T(n)= O(C*m*n)
T(n)= O(m*n)

5. **n y m en la complejidad:** En los cálculos de la complejidad anterior, estas variables se refieren a la cantidad de veces que el algoritmo tiene que ejecutar algún paso o acción al momento de compilarse para llegar al resultado esperado. En el ejercicio anterior, hicimos uso de las dos variables ya que nuestro algoritmo trabaja con más de una entrada, tiene m y n que diferencian los alcances de complejidad que puede llegar a tener nuestro algoritmo según el valor que tomen las variables al momento de compilarse.

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	20,A	40,F	70,F	∞	30,B	90,A	∞
4	C	20,A	40,F	50,C	∞	30,B	90,A	60,C
5	D	20,A	40,F	50,C	∞	30,B	70,D	60,C
6	H	20,A	40,F	50,C	∞	30,B	70,D	60,C
7	G	20,A	40,F	50,C	∞	30,B	70,D	60,C
8	E	20,A	40,F	50,C	∞	30,B	70,D	60,C

b).

Vértice	Costo	Anterior	Camino desde el vértice A

G	70	D	A,B,F,C,D,G

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