**Laboratorio Nro. 4: Algoritmos voraces**

|  |  |
| --- | --- |
| **Luisa María Vásquez Gómez**  Universidad Eafit  Medellín, Colombia  lmvasquezg@eafit.edu.co | **Juan José Parra Díaz**  Universidad Eafit  Medellín, Colombia  jjparrad@eafit.edu.co |

**3) Simulacro de preguntas de sustentación de Proyectos**

1. El algoritmo del punto 1 funciona buscando en el nodo de inicio el sucesor más cercano a este ,al hallarlo, este es agregado al camino y se realiza simultáneamente el mismo proceso moviéndose a través del grafo en aquellos vértices que no han sido visitados; una vez recorrido todo el grafo, se busca un camino hacia el inicio de nuevo, si este es sucesor inmediato del ultimo vértice del camino, simplemente vuelve al origen, de lo contrario, busca un camino a este con el mismo método usado anteriormente, eso sí, con el costo de que tendría que visitar ciudades ya visitadas, todo esto depende de cómo este estructurado el grafo.
2. Para poder dar solución al problema del agente viajero es indispensable que exista al menos un arco que llegue al origen , ya que si no lo hay, se vuelve imposible encontrar un camino que termine en este; además , todas las ciudades deben tener arcos de entrada y salida, ya que a falta de estos se puede llegar a un estado en el que no se pueden visitar mas ciudades , o la ciudad en cuestión nunca podría ser visitada.

**HASTA AQUI LAB4**

1. El ejercicio en línea 2.1 funciona leyendo por medio de un BufferedReader, la información de la entrada, con la cual va creando un grafo, línea por línea, hasta que no haya más arcos que agregar. Con el grafo creado, se comienza a explorar por medio de un algoritmo DFS recursivo. Este recorre todos los sucesores de un nodo, sin repetir, pero antes de ello, revisa que el peso del camino acumulado al nodo que va a visitar sea inferior al peso del camino con menor peso encontrado. Si decide que el peso es menor, guarda el camino en un entero y pasa al siguiente nodo, pero si es mayor, se devuelve hasta donde fue menor por última vez y retira el camino agregado del entero. Al hallar un camino, se sabe que fue uno con menor peso que el anterior, entonces lo guarda y junto a este, el peso que consumió. Al final, se imprime el camino más corto encontrado.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

1. En el cálculo de la complejidad, la variable ‘n’ representa el número de nodos, de tal manera que ésta se puede leer como el factorial del número total de nodos de un grafo.

**4) Simulacro de parcial**

* 1. n-a,a,b,c
  2. res,solucionar(n-b,a,b,c)+1
  3. res,solucionar(n-c,a,b,c)+1
  4. graph.length-1
  5. v,graph,pat,pos
  6. graph,path,pos+1
  7. DFS

0 = 0 3 7 4 2 1 5 6

1 = 1 0 3 7 2 4 6 5

2 = 2 1 0 3 7 5 4 6

3 = 3 7

4 = 4 2 1 0 3 7 5 6

5 = 5

6 = 6 2 1 0 3 7 4 5

7 = 7

* 1. BFS

0 = 0 3 4 7 2 1 6 5

1 = 1 0 2 5 3 4 6 7

2 = 2 1 4 6 0 5 3 7

3 = 3 7

4 = 4 2 1 6 0 5 3 7

5 = 5

6 = 6 2 1 4 0 5 3 7

7 = 7

**4.**

|  |
| --- |
|  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**5.**