**Laboratorio Nro. 5: Programación Dinámica**

|  |  |
| --- | --- |
| **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. **LUISA ☺**
2. Aparte de la fuerza bruta o algoritmos voraces, el problema del agente viajero se puede solucionar por medio de la programación dinámica, dividiendo en sub problemas el problema mayor. Una solución con programación dinámica para este problema es el algoritmo de Held-Karp. Este algoritmo consiste en encontrar la distancia mínima a los nodos cercanos a medida que los va recorriendo, y guardándola, para así saber la distancia que debe recorrer al pasar por estos nuevamente.
3. **LUISA ☺**

|  |  |
| --- | --- |
|  |  |
| public static int distance( ArrayList<Pair<Integer,Integer>> recoger,int[][] mapa, int inix , int iniy){ | **O(n2)** |
| Pair<Integer,Integer> inicial = new Pair(inix,iniy); | C |
| Pair<Integer,Integer> inicialg = new Pair(inix,iniy); | C |
| double distMin=Integer.MAX\_VALUE; | C |
| int indexMasCercano =0; | C |
| int totes=0; | C |
| while(!recoger.isEmpty()){ | C\*n |
| for(Pair<Integer,Integer> a : recoger){ | C\*n\*n |
| double d =distancia(inicial,a); | C\*n\*n |
| if(d<distMin){ | C\*n\*n |
| distMin=d; | C\*n\*n |
| indexMasCercano = recoger.indexOf(a); | C\*n\*n |
| } |  |
| } |  |
| totes=totes+ Math.abs(inicial.getKey()-recoger.get(indexMasCercano).getKey())+ | C\*n |
| Math.abs(inicial.getValue()-recoger.get(indexMasCercano).getValue()); |  |
| inicial=recoger.get(indexMasCercano); | C\*l |
| recoger.remove(recoger.get(indexMasCercano)); | C\*l |
| distMin=Integer.MAX\_VALUE; | C\*l |
| indexMasCercano =0; | C\*l |
| } |  |
| return totes+ Math.abs(inicial.getKey()-inicialg.getKey())+Math.abs(inicial.getValue()-inicialg.getValue()); | C |
|  |  |
| } |  |
|  |  |
| private static double distancia(Pair<Integer,Integer> a,Pair<Integer,Integer> b){ | **O(1)** |
| double d = Math.hypot(Math.abs(a.getKey()-b.getKey()), Math.abs(a.getValue()-b.getValue())); | C |
| return d; | C |
| } |  |
|  |  |
| public static void main(String[] args) throws IOException{ | **O(n2\*e)** |
| BufferedReader sc = new BufferedReader(new InputStreamReader(System.in)); | C |
| System.out.println (" "); | C |
| String esc = sc.readLine(); | C |
| int escenarios = Integer.parseInt(esc); | C |
| for(int i=0;i<escenarios;i++){ | C\*e |
| String tamaño = sc.readLine(); | C\*e |
| String[] lxa = tamaño.split(" "); | C\*e |
| int[][] mapa= new int [Integer.parseInt(lxa[0])][Integer.parseInt(lxa[1])]; | C\*e |
| esc = sc.readLine(); | C\*e |
| String[] aux = esc.split(" "); | C\*e |
| esc = sc.readLine(); | C\*e |
| int desechos = Integer.parseInt(esc); | C\*e |
| ArrayList<Pair<Integer,Integer>> vertices = new ArrayList(); | C\*e |
| for(int j=0;j<desechos;j++){ | C\*e\*d |
| String posicion = sc.readLine(); | C\*e\*d |
| String [] partido = posicion.split(" "); | C\*e\*d |
| Pair<Integer,Integer> a = new Pair(Integer.parseInt(partido[0]),Integer.parseInt(partido[1])); | C\*e\*d |
| mapa[Integer.parseInt(partido[0])][Integer.parseInt(partido[1])]= 1; | C\*e\*d |
| vertices.add(a); | C\*e\*d |
| } |  |
| System.out.println("The shortest path has length: "+distance(vertices,mapa,Integer.parseInt(aux[0]),Integer.parseInt(aux[1]))); | C\*e\*O(n2) |
|  |  |
| } |  |
| } |  |
|  |  |

**Complejidad: O(n2\*e)**

1. En el cálculo de la complejidad del ejercicio en línea hay dos variables: ‘n’ y ‘e’. ‘n’ representa el número de **LUISA ☺** y ‘e’ es el número de escenarios que serán probados.

**4) Simulacro de parcial**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **C** | **A** | **L** | **L** | **E** |
| **C** | 0 | 1 | 2 | 3 | 4 |
| **A** | 1 | 0 | 1 | 2 | 3 |
| **S** | 2 | 1 | 1 | 2 | 3 |
| **A** | 3 | 2 | 2 | 2 | 3 |

**1.1)**

**1.2)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **M** | **A** | **D** | **R** | **E** |
| **M** | 0 | 1 | 2 | 3 | 4 |
| **A** | 1 | 0 | 1 | 2 | 3 |
| **M** | 2 | 1 | 1 | 2 | 3 |
| **A** | 3 | 2 | 2 | 2 | 3 |

**2.**

1) O(x\*y)

2) return table[i][j];

**3.**

1) a

2) a | c

**4.**

1) b