

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. El algoritmo del punto 1.2 funciona encontrando inicialmente el conjunto potencia de los vértices del grafo, sin incluir el inicial ; para cada uno de los subconjuntos unitarios se calcula el costo de alcanzarlo desde el vértice inicial y desde los demás vértices , teniendo en cuenta el trayecto que ya se tiene desde el inicial al vértice en el que se itera. Para los subconjuntos con mas de 1 elemento se analiza el mínimo costo entre el vértice con el que se esta comparando con el subconjunto unitario de uno de los elementos del subconjunto , y así mismo con todos los subconjuntos de conjunto potencia hasta comparar el inicial con el subconjunto que contiene a todos los demás vértices eligiendo el mínimo costo de las comparaciones anteriores.
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. El ejercicio en linea funciona agregando todos los residuos (a modo de Pareja , que contiene las coordenadas "x" y "y") en un ArrayList, y a partir del punto inicial se calcula el residuo más cercano para recoger perteneciente al ArrayList, una vez encontrado se elimina de este, se toma esta coordenada como el punto inicial y se agrega la diferencia en "x" y "y" respecto al punto inicial al total del recorrido, y así hasta recoger todos los residuos , al final simplemente se agrega el costo del recorrido desde el ultimo residuo hasta la posición inicial y se retorna el costo total.
- 4.

```

public static int distance( ArrayList<Pair<Integer,Integer>> recoger, int inix O(n2)
, int iniy){
    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()-                  C*n
recoger.get(indexMasCercano).getKey()+
        Math.abs(inicial.getValue()-
recoger.get(indexMasCercano).getValue());
        inicial=recoger.get(indexMasCercano);                   C*1
        recoger.remove(recoger.get(indexMasCercano));           C*1
        distMin=Integer.MAX_VALUE;                               C*1
        indexMasCercano =0;                                       C*1
    }
    return totes+ Math.abs(inicial.getKey()-                      C
inicialg.getKey()+Math.abs(inicial.getValue()-inicialg.getValue());

}

```

```

private static double distancia(Pair<Integer,Integer> O(1)
a,Pair<Integer,Integer> b){

```

```

        double d = Math.hypot(Math.abs(a.getKey()-b.getKey()),
Math.abs(a.getValue()-b.getValue()));

        return d;
    }

    public static void main(String[] args) throws IOException{
        BufferedReader sc = new BufferedReader(new
InputStreamReader(System.in));
        System.out.println (" ");
        String esc = sc.readLine();
        int escenarios = Integer.parseInt(esc);
        for(int i=0;i<escenarios;i++){
            String tamaño = sc.readLine();
            String[] lxa = tamaño.split(" ");
            esc = sc.readLine();
            String[] aux = esc.split(" ");
            esc = sc.readLine();
            int desechos = Integer.parseInt(esc);
            ArrayList<Pair<Integer,Integer>> vertices = new ArrayList();
            for(int j=0;j<desechos;j++){
                String posicion = sc.readLine();
                String [] partido = posicion.split(" ");
                Pair<Integer,Integer> a = new
Pair(Integer.parseInt(partido[0]),Integer.parseInt(partido[1]));
                vertices.add(a);
            }
            System.out.println("The shortest path has length:
"+distance(vertices,Integer.parseInt(aux[0]),Integer.parseInt(aux[1])));
        }
    }

```

C
C


O(n²*e)
C

C
C
C
C*e
C*e
C*e
C*e
C*e
C*e
C*e
C*e*d
C*e*d
C*e*d
C*e*d

C*e*d

C*e*(n²)

Complejidad: O(n²*e)

	<p style="text-align: center;">UNIVERSIDAD EAFIT ESCUELA DE INGENIERÍA DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS</p>	<p>Código: ST245</p>
		<p>Estructura de Datos 1</p>

5. En el cálculo de la complejidad del ejercicio en línea hay dos variables: 'n' y 'e'. 'n' representa el número de residuos a recoger y 'e' es el número de escenarios que serán probados.

4) Simulacro de parcial**1.1)**

	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

1.2)

	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) $O(\text{lenx} * \text{leny})$
- 2) `return table[lenx][leny];`

3.

- 1) a.
- 2) c.

4.

- 1) c.