   
   
  **Unidad Profesional Interdisciplinaria de Ingeniería Campus Zacatecas (UPIIZ)**

**REPORTE DEL EJERCICIO TSP**

**Materia: Analisis de Algoritmos**

**Instituto Politécnico Nacional**

**Fecha:11/09/2019**

**Alumno: José Moisés Luna Montes  
 No.Boleta: 2017670761**

**Ingeniería en Sistemas Computacionales**

**Grupo:3cm1**

**INTRODUCCION**

En esta ocasión realizamos como ejercicio de la materia de análisis de algoritmo el TSP dinámicamente, después de haber hecho el de fuerza bruta en el anterior ejercicio y esto dando un desafío mayor al momento de programarlo con este método.

**MARCO TEORICO**

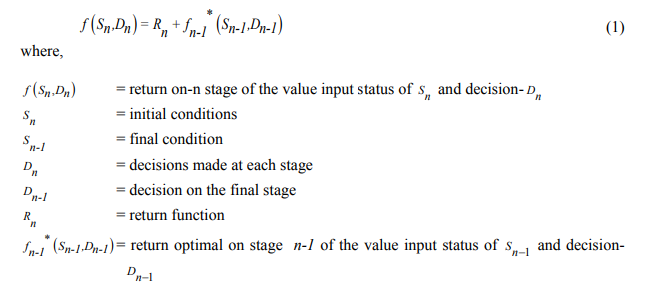
**TSP**

Es un método que se utiliza particularmente en la ciencia del transporte y puede explicarse como un vendedor que debe viajar a través de todos los ciudades separadas por las distancias más cortas, donde cada ciudad solo puede atravesarse una vez.  
La solución de este problema es el camino con la distancia más corta.

**PROGRAMACIÓN DINÁMICA**  
Es un método para resolver problemas dividiendo la solución en un conjunto de pasos o etapas para que la solución del problema se pueda ver a partir de una serie de relaciones interrelacionadas decisiones, pero arriesgando la memoria.

**PROGRAMACIÓN DINÁMICA EN TSP**

En general, el modelo de programación dinámica se puede escribir de la siguiente manera:



El código

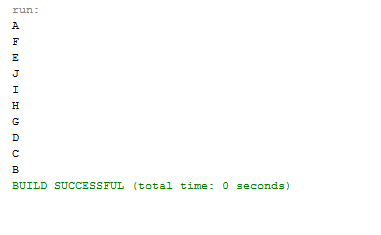
La programación del algoritmo DPTSP se resuelve utilizando el número binario 2 ^ n-1 (que es un binario consumió 4 bytes de memoria).

**DESARROLLO**

Recursos necesarios :  
\*Maquina con neatbeans, eclipse u otro entorno de programación.   
\*Tener instalado Java con sus diversas rutas en path.

**Código en java:**

|  |  |
| --- | --- |
|  | package tspd; |
|  |  |
|  | import java.util.ArrayList; |
|  |  |
|  | /\*\* |
|  | \* |
|  | \* @author Mouxes13 |
|  | \*/ |
|  | public class TSPD { |
|  |  |
|  |  |
|  | private ArrayList<Integer> salida = new ArrayList<Integer>(); |
|  | private int g[][], p[][], npow, d[][]; |
|  |  |
|  | public TSPD() { |
|  | } |
|  |  |
|  | public ArrayList<Integer> CaminoCorto(int[][] Matriz) { |
|  |  |
|  |  |
|  | npow = (int) Math.pow(2, 10); |
|  | g = new int[10][npow]; //declaramos los tamaños de la matriz |
|  | p = new int[10][npow]; //declaramos los tamaños de la matriz |
|  | d = Matriz; |
|  | int k, l, m, s; |
|  | for (int i = 0; i < 10; i++) { //llenamos las matrices de 0 |
|  | for (int j = 0; j < npow; j++) { |
|  | g[i][j] = 0; |
|  | p[i][j] = 0; |
|  | } |
|  | } |
|  | //inicializamos la matriz de g por la matriz que teniamos |
|  | for (int i = 0; i < 10; i++) { |
|  | g[i][0] = Matriz[i][0]; |
|  | } |
|  |  |
|  | int result = Tsp(0 , npow -2); //damos el resultado por lo que nos retorne Tsp |
|  | salida.add(0); // en la salida agregamos un cero |
|  | BuscarCamino(0 , npow - 2);//llamamos el metodo buscar metodo |
|  | salida.add(result); //agregamos el resultado en la matriz salida |
|  |  |
|  |  |
|  | return salida; |
|  | } |
|  |  |
|  | private int Tsp(int start, int set) { //comprabor de tsp |
|  | int masked, mask, result = 0, temp; |
|  | if (g[start][set] != 0) { //comprobamos que lo que se encuentra en la matriz sea difrente de 0 si no se retorna |
|  | return g[start][set]; |
|  | } else { |
|  | for (int x = 0; x < 10; x++) { |
|  | mask = npow - 1 - (int) Math.pow(2, x); |
|  | masked = set & mask; |
|  | if (masked != set) { |
|  | temp = d[start][x] + Tsp(x, masked); |
|  | if (result == 0 || result > temp) { |
|  | result = temp; |
|  | p[start][set] = x; |
|  | } |
|  | } |
|  | } |
|  | g[start][set] = result; |
|  | return result; |
|  | } |
|  | } |
|  |  |
|  | private void BuscarCamino(int start, int set) { //metodo recursivo para buscar el camino más corto |
|  | if (p[start][set] == 0) { |
|  | return; |
|  | } |
|  | int x = p[start][set]; |
|  | int mask = npow - 1 - (int) Math.pow(2, x); |
|  | int masked = set & mask; |
|  | salida.add(x); |
|  | BuscarCamino(x, masked); |
|  | } |
|  |  |
|  | public static void main(String[] args) { |
|  | ArrayList<Integer> resp = new ArrayList<Integer>(); |
|  | int[][] matriz = { //damos por la matriz que nos había puesto |
|  | {0, 13, 33, 28, 37, 7, 32, 40 , 80 , 26}, |
|  | {0, 0, 39, 83, 50 , 68, 16, 98, 81, 55}, |
|  | {0, 0, 0, 80 , 88, 49, 53, 75, 63, 55}, |
|  | {0, 0, 0, 0, 94, 4, 20 , 6, 59, 76}, |
|  | {0, 0, 0, 0, 0, 81, 87, 85, 4, 19}, |
|  | {0, 0, 0, 0, 0, 0, 96, 53, 40 , 37}, |
|  | {0, 0, 0, 0, 0, 0, 0, 80 , 57, 68}, |
|  | {0, 0, 0, 0, 0, 0, 0, 0, 65, 41}, |
|  | {0, 0, 0, 0, 0, 0, 0, 0, 0, 97}, |
|  | {0, 0, 0, 0, 0, 0, 0, 0, 0, 0} |
|  | }; |
|  |  |
|  | TSPD t = new TSPD(); //declaraqmos la clase |
|  | resp = t.CaminoCorto(matriz); //declaramos el metodo de la clase |
|  | for (int i = 0; i < 10; i++) { |
|  | System.out.println((char) (65 + resp.get(i))); |
|  |  |
|  | } |
|  |  |
|  | } |
|  | } |

**Resultado**

**CONCLUSION**Concluyo que en este programa aunque ya teníamos referencias de la anterior que ya habíamos hecho

**BIBLIOGRAFIA**

SOLVING TRAVELING SALESMAN PROBLEM BY DYNAMIC PROGRAMMING APPROACH IN JAVA PROGRAMMING/CE 6001 OPERATIONS MANAGEMENT AND INFRASTRUCTURE SYSTEM ADITYA NUGROHO/DEPARTMENT OF CIVIL ENGINEERING NATIONAL UNIVERSITY OF SINGAPORE 2010