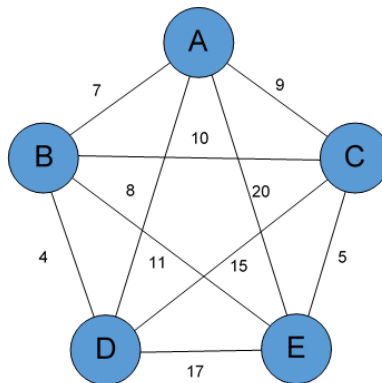




Análisis de Algoritmos

“Travelling Salesman Problem”

Solución del problema TSP (Dinámico)



***Docente:** Roberto Oswaldo Cruz Leíja.*

***Alumna:** Paulina Medrano Hurtado.*

06/11/20

Introducción

La complejidad del cálculo del problema del agente viajero ha despertado múltiples iniciativas por mejorar la eficiencia en el cálculo de rutas. El método más básico es el conocido con el nombre de fuerza bruta, y existen muchos algoritmos como Divide y vencerás, Decrece y conquista, etc. Que de igual modo dan la solución al problema. En el desarrollo de esta práctica se pretende implementar la solución del problema TSP dinámicamente (programación dinámica).

Descripción del problema

“El Problema del Agente Viajero”

La motivación detrás del problema del vendedor ambulante (también conocido como problema del vendedor ambulante) es el problema que enfrenta un vendedor que necesita visitar una cantidad de clientes ubicados en diferentes ciudades e intenta encontrar el viaje de ida y vuelta más corto para lograr esta tarea. En una formulación más general y abstracta, al TSP se le da, con un gráfico dirigido, ponderado por el borde, para encontrar una ruta cíclica más corta que visite cada nodo en este gráfico exactamente una vez.

Se tiene un número de nodos (ciudades, localidades, tiendas, empresas, etc.) que deben ser visitados por una entidad (persona, agente viajero, automotor, avión, autobús, etc.), sin visitar 2 veces el mismo nodo. Si tenemos 3 nodos (a, b y c) por visitar, entonces tendríamos una función de combinaciones sin repetición $c(3,2)$, es decir, tendríamos 6 posibles soluciones: abc, acb, bac, bca, cab, cba, para el caso de 4 nodos tendríamos 12 combinaciones, para 10 nodos tendríamos 90 combinaciones, para 100 ciudades tendríamos 9,900 combinaciones y así sucesivamente.

Programación Dinámica

La programación dinámica es un paradigma algorítmico muy poderoso en el cual un problema es resuelto mediante la identificación de una colección de subproblemas y abordarlos uno por uno, el más pequeño primero, usando las respuestas a pequeños problemas para ayudar a resolver los más grandes, hasta que todos están resueltos. En programación dinámica no se nos da un dag. El dag está implícito. Sus nodos son los subproblemas que definimos, y sus bordes son las dependencias entre los subproblemas: si para resolver el subproblema B necesitamos la respuesta al subproblema A, entonces hay un borde (conceptual) de A a B. En este caso, A se considera un subproblema más pequeño que B, y siempre lo será ser más pequeño, en un sentido obvio.

Programación dinámica: Deje que el conjunto de vértices dado sea $\{1, 2, 3, 4, \dots, n\}$. Consideremos 1 como punto de salida inicial y final. Para cada otro vértice i (que no sea 1), encontramos la ruta de costo mínimo con 1 como punto de partida, i como punto final y todos los vértices aparecen exactamente una vez. Supongamos que el costo de esta ruta es el costo (i) , el costo del ciclo correspondiente sería el costo $(i) + \text{dist}(i, 1)$ donde $\text{dist}(i, 1)$ es la distancia de i a 1. Finalmente, devolvemos el mínimo de todos los valores $[\text{costo}(i) + \text{dist}(i, 1)]$. Esto parece simple hasta ahora. Ahora la pregunta es cómo obtener el costo (i) .

Para calcular el costo (i) usando la programación dinámica, necesitamos tener alguna relación recursiva en términos de subproblemas. Definamos un término $C(S, i)$ será el costo de la ruta de costo mínimo que visita cada vértice en el conjunto S exactamente una vez, comenzando en 1 y terminando en i .

Comenzamos con todos los subconjuntos de tamaño 2 y calculamos $C(S, i)$ para todos los

subconjuntos donde S es el subconjunto, luego calculamos $C(S, i)$ para todos los subconjuntos S de tamaño 3 y así sucesivamente. Tenga en cuenta que 1 debe estar presente en cada subconjunto.

Descripción de implementación: Se utiliza un método para reducir el tiempo de ejecución de un algoritmo mediante la división en subproblemas y resolver recordando todas las soluciones por si en las siguientes iteraciones fueran necesarias nuevamente.

Código

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package TSPDinamico;

import java.util.ArrayList;

/**
 *
 * @author HP
 */
//Camino más cortos en días, revisados

public class TSP {
    //Matriz
    int matriz[][];
    int INF = 100000000;
    public TSP(){
        ArrayList<Integer> S = new ArrayList<Integer>();
        Tokenizador.leerDatos();
        ArrayList<int[]> distancias = Tokenizador.instancias;
        int n = distancias.size();
        //Creación de la matriz
        matriz = new int[n][n];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                matriz[i][j] = distancias.get(i)[j];
            }
        }
        for(int i = 1; i < n; i++) S.add(i);
        System.out.println("Costo mínimo: "+C(0,S));
    }
    public int C(int i, ArrayList<Integer> S){
        //S no tiene datos se hace la búsqueda desde 0
        if(S.size() == 0) return matriz[i][0];
    }
}
```

```

        else if(S.size()>0){
            //Si tiene datos se retorna el minimo de la consulta en
la matriz mas
            //la funcion con el conjunto sin el valor que se esta
consultando
            int minx = INF;
            for(Integer k : S){
                ArrayList<Integer> X =
(ArrayList<Integer>)S.clone();
                X.remove(k);
                minx = Math.min(minx,matriz[i][k.intValue()] +
C(k.intValue(),X));
            }
            return minx;
        }
        return 0;
    }
    public static void main(String[] args) {
        TSP tsp = new TSP();
    }
}

```

```

/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package TSPDinamico;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.StringTokenizer;
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;
/**
 *
 * @author HP
 */
public class Tokenizador {
    public static ArrayList<int[]> instancias;

    public static void leerDatos(){
        instancias = new ArrayList<>();
        String texto, aux;
        LinkedList<String> lista = new LinkedList();
        //ArrayList<Patron> patrones = new ArrayList<>();
        try {
            //llamamos el metodo que permite cargar la ventana
            JFileChooser file = new JFileChooser();
            file.showOpenDialog(file);
            //abrimos el archivo seleccionado
            File abre = file.getSelectedFile();

            //recorremos el archivo y lo leemos
            if (abre != null) {
                FileReader archivos = new FileReader(abre);
                BufferedReader lee = new BufferedReader(archivos);

                while ((aux = lee.readLine()) != null) {
                    texto = aux;
                    lista.add(texto);
                }
                lee.close();
                //System.out.println(lista.size());
            }
        }
    }
}

```

```

        ArrayList<String> lista2 = new ArrayList<>();
        String clase = "";
        for (int i = 0; i < lista.size(); i++) {
            StringTokenizer st = new
StringTokenizer(lista.get(i), ",");

            while (st.hasMoreTokens()) {
                lista2.add(st.nextToken());
            }

            int[] vector = new int[lista2.size()];

            for (int x = 0; x < lista2.size(); x++) {
                vector[x] = Integer.parseInt(lista2.get(x));
            }

            // a la coleccion de patrones se agrega un nuevo
patron

            instancias.add(vector);
            // patrones.add();
            lista2.clear();

        }

    } catch (IOException ex) {
        JOptionPane.showMessageDialog(null, ex + ""
            + "\nNo se ha encontrado el archivo",
            "ADVERTENCIA!!!", JOptionPane.WARNING_MESSAGE);
    }

}

}

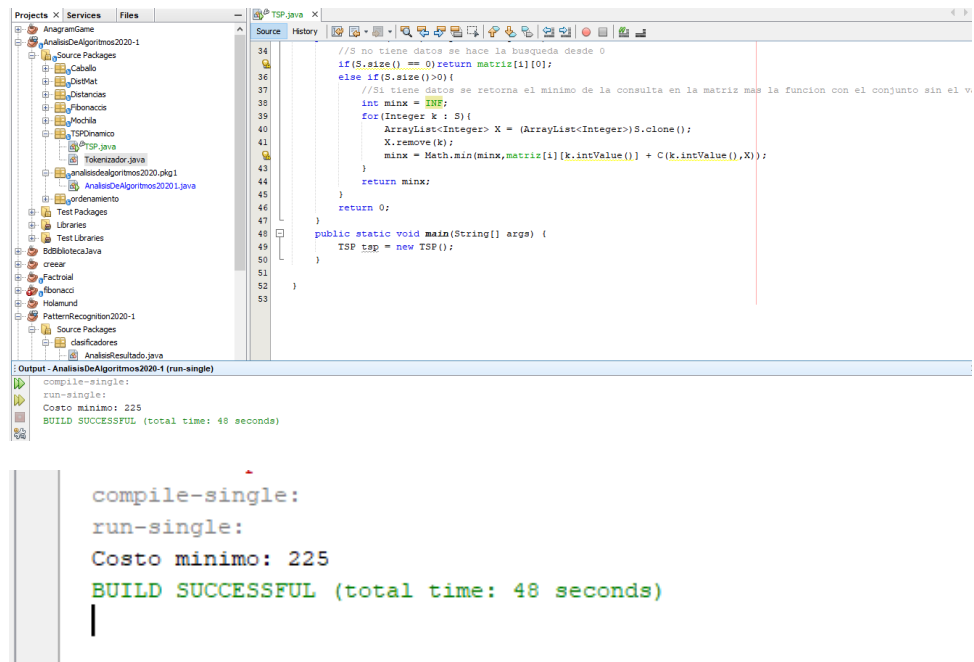
```

Resultados

Utilizando esta matriz de caminos

```
distancias Bloc de notas
Archivo Edición Formato Ver Ayuda
0,13,33,28,37,7,32,40,80,26
13,0,39,83,50,68,16,98,81,55
33,39,0,80,88,49,53,75,63,55
28,83,80,0,94,4,20,6,59,76
37,50,88,94,0,81,87,85,4,19
7,68,49,4,81,0,96,53,40,37
32,16,53,20,87,96,0,80,57,68
40,98,75,6,85,53,80,0,65,41
80,81,63,59,4,40,57,65,0,97
26,55,55,76,19,37,68,41,97,0
```

Se obtuvo.



The screenshot shows an IDE with a project named 'AnalisisDeAlgoritmos2020-1'. The code in the editor is a Java program for a Traveling Salesman Problem (TSP). It includes a recursive method to find the minimum cost path by visiting all cities exactly once. The output window shows the results of running the program.

```
Source History
34 //Si no tiene datos se hace la búsqueda desde 0
35 if(S.size() == 0) return matrix[i][0];
36 else if(S.size() > 0){
37     //Si tiene datos se retorna el minimo de la consulta en la matriz mas la funcion con el conjunto sin el v
38     int minx = INF;
39     for(Integer k : S){
40         ArrayList<Integer> X = (ArrayList<Integer>)S.clone();
41         X.remove(k);
42         minx = Math.min(minx, matrix[i][k.intValue()] + C(k.intValue(), X));
43     }
44     return minx;
45 }
46
47 return 0;
48
49 public static void main(String[] args) {
50     TSP tsp = new TSP();
51 }
52
53 }
```

Output - AnalisisDeAlgoritmos2020-1 (run-single)

```
compile-single:
run-single:
Costo minimo: 225
BUILD SUCCESSFUL (total time: 48 seconds)
```

```
compile-single:
run-single:
Costo minimo: 225
BUILD SUCCESSFUL (total time: 48 seconds)
```

Conclusiones

En esta práctica el desarrollo de la solución del TSP dinámico, fue algo retador ya que se sabía ya del problema y ya se había dado una solución que fue con fuerza bruta, pero esta vez sería distinto y para poder lograrlo estuve investigando al respecto para tener una idea más clara.

Implemente la solución con la matriz que antes ya habíamos trabajado con fuerza bruta todo el grupo y los resultados si fueron favorables o bien correctos.

Otra solución que bien es igual a la solución ya planteada es con el algoritmo Held y Karp o bien mejor conocido como algoritmo Held y Karp para TSP y este implementado con mascara de bits.

Bibliografía

<https://www.sciencedirect.com/topics/computer-science/travelling-salesman-problem>

<https://www.ingenieriaindustrialonline.com/herramientas-para-el-ingeniero-industrial/investigaci%C3%B3n-de-operaciones/problema-del-agente-viajero-tsp/>