Práctica 4: Programación Dinámica

Irene Huertas González
Javier Alcántara García
César Muñoz Reinoso

Índice

Objetivo de la practica

El objetivo de esta práctica es apreciar la utilidad de la programacion dinámica para resolver problemas de forma muy eficiente obteniendo soluciones óptimas en algunos casos y aproximaciones en otros.
Para ello se nos ha pedido que desarrollemos un problema usando dicha programación dinámica y la compararemos con algoritmos voraces.

Problema elegido: El problema del viajante de comercio

Enunciado: Dado un conjunto de ciudades y una matriz con las distancias entre todas ellas, un viajante debe recorrer todas las ciudades exactamente una vez, regresando al punto departida, de forma tal que la distancia recorrida sea mínima. Mas formalmente, dado un grafo G, conexo y ponderado, se trata de hallar el ciclo hamiltoniano de mínimo peso de ese grafo.
Nos piden diseñar y desarrollar un algoritmo basado en Programación dinámica para el problema del viajante de comercio. Además debemos comparar la solución y la eficiencia del algoritmo con los elaborados en la práctica 3, es decir, los algoritmos voraces.

Problema elegido: El problema del viajante de comercio

Nos piden diseñar y desarrollar un algoritmo basado en Programación dinámica para el problema del viajante de comercio. Además debemos comparar la solución y la eficiencia del algoritmo con los elaborados en la práctica 3, es decir, los algoritmos voraces.

Generación de datos

Los datos que se tienen que usar para evaluar esta tarea son los que se nos proporcionan en PRADO, los de la carpeta TSPLIB.

Implementación

```
int tsp(const vector<vector<int>>& cities, int pos,
        int visited, vector<vector<int>>& state)
{
    if(visited == ((1 << cities.size()) - 1))
        return cities[pos][0];

    if(state[pos][visited] != INT_MAX)
        return state[pos][visited];
```

Implementación

```
        for(int i = 0; i < cities.size(); ++i)
        {
            if(i == pos || (visited & (1 << i)))
                continue;

            int distance = cities[pos][i] +
            tsp(cities, i, visited | (1 << i), state);
            if(distance < state[pos][visited])
                state[pos][visited] = distance;
        }

        return state[pos][visited];
    }
```

Ecuación de recurrencia

Para la ecuacion Recurrente del Algoritmo hemos propuesto 2 casos diferentes.

$$TSP(n, p_k) = \begin{cases} 0 & si\ la\ ciudad\ ya\ ha\ sido\ recorrida \\ \min\{TSP(n-1, p_k+1) + d_k, TSP(n, p_k+1)\} & en\ otro\ caso \end{cases} \quad (1)$$

Donde n es el numero de cuidades, p_k es la ciudad k y d_k es la distancia hasta la ciudad k.

Ejecucciones y Comparación

Hemos ejecutado los archivos a280.tsp y att48.tsp comparandolo con el Algoritmo Voráz y este es el resultado:
./ejercicio a280.tsp
25 ciudades a280->
P.Dinamica: 358
Insercion : 635

./ejercicio att48.tsp
15 ciudades attt48->
P.Dinamica: 20347
Insercion : 33191

Conclusión

En conclusion vemos que el Algoritmo Voráz es mas rapido que el Algoritmo de Programación Dinámica aunque apreciamos que este ultimo es el más eficiente.