Resolución de Planificación de Viajes Aéreos con Programación Dinámica

Jesús Losada Arauzo Javier Gómez Moleón

May 20, 2024

1. Diseño de resolución por etapas y ecuación recurrente

El problema puede ser resuelto en etapas donde cada etapa considera la inclusión de un nodo intermedio k. La ecuación recurrente para la programación dinámica sería:

$$D[i][j] = \min(D[i][j], D[i][k] + D[k][j] + E(k))$$

donde $D[i][j]$ es el costo mínimo para viajar de i a j pasando por el nodo intermedio k.

2. Diseño de la memoria

Usamos una matriz de tamaño $n \times n$ para almacenar los costos mínimos de viaje entre todas las ciudades i y j. La matriz inicializa los valores según la matriz de tiempo T y ajusta los valores con las actualizaciones de los nodos intermedios.

3. Verificación del P.O.B. (Principio de Optimalidad de Bellman)

El principio de optimalidad de Bellman se verifica mediante la ecuación recurrente utilizada en el algoritmo de Floyd-Warshall. Esto asegura que el camino más corto entre dos nodos i y j pasando por k es óptimo.

4. Diseño del algoritmo de cálculo de coste óptimo

El algoritmo de cálculo de coste óptimo es el algoritmo de Floyd-Warshall modificado para incluir el tiempo de escala $E(k)$.

5. Diseño del algoritmo de recuperación de la solución

Para recuperar la solución (el camino más corto), necesitamos mantener una matriz de predecesores que nos permita reconstruir el camino.

6. Implementación de los algoritmos de cálculo de coste óptimo y recuperación de la solución

```cpp
// Incluyendo bibliotecas necesarias
#include <iostream>
#include <vector>
#include <climits>
#include <algorithm>

using namespace std;

const int INF = INT_MAX;

// Implementaci\'on del algoritmo de Floyd-Warshall adaptado con recuperaci\'on de camino
void floydWarshall(const vector<vector<int>>& T, const vector<int>& E, vector<vector<int>>& D, vector<vector<int>>& P) {
    int n = T.size();

    // Inicializaci\'on de la matriz de distancias y predecesores
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (i == j) {
                D[i][j] = 0;
                P[i][j] = -1; // No hay predecesor
            } else if (T[i][j] != 0) {
                D[i][j] = T[i][j];
                P[i][j] = i; // El predecesor de j es i
            } else {
                D[i][j] = INF;
                P[i][j] = -1; // No hay predecesor
            }
        }
    }

    // Actualizaci\'on de la matriz de distancias y predecesores
    for (int k = 0; k < n; ++k) {
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                if (D[i][k] != INF && D[k][j] != INF && D[i][k] + D[k][j] + E[k] < D[i][j]) {
                    D[i][j] = D[i][k] + D[k][j] + E[k];
                    P[i][j] = P[k][j]; // Actualiza el predecesor
                }
            }
        }
    }
```

```
}

// Funci\'on para recuperar el camino m\'as corto
vector<int> getPath(int i, int j, const vector<vector<int>>& P) {
    vector<int> path;
    if (P[i][j] == -1) {
        return path; // No hay camino
    }
    path.push_back(j);
    while (i != j) {
        j = P[i][j];
        path.push_back(j);
    }
    reverse(path.begin(), path.end());
    return path;
}

int main() {
    // Matriz de tiempos T y vector de tiempos de escala E
    vector<vector<int>> T = {
        {0, 1, 3, 4},
        {1, 0, 2, 3},
        {3, 2, 0, 4},
        {4, 3, 4, 0}
    };
    vector<int> E = {1, 1, 1, 1};

    int n = T.size();
    vector<vector<int>(n, vector<int>(n, INF)));
    vector<vector<int>> P(n, vector<int>(n, -1));

    floydWarshall(T, E, D, P);

    // Imprimir la matriz de distancias mÃnimas
    cout << "Matriz de costos mÃnimos:" << endl;
    for (const auto& fila : D) {
        for (const auto& valor : fila) {
            if (valor == INF) {
                cout << "INF ";
            } else {
                cout << valor << " ";
            }
        }
        cout << endl;
    }

    // Recuperar y imprimir el camino mÃ¡s corto de ejemplo
    int start = 0, end = 3;
    vector<int> path = getPath(start, end, P);
    cout << "Camino mÃ¡s corto de " << start + 1 << " a " << end +
    1 << ": ";
    for (int city : path) {
        cout << city + 1 << " ";
    }
    cout << endl;

    return 0;
```

```
98 }
```