Given Graph Data:
Vertices: San Francisco (SF), Los Angeles (LA), Denver (DEN), Chicago (CHI), Boston (BOST), New York (NY), Miami (MIA), New Orleans (NO).

Distances:

- SF -> LA: 300
- SF -> DEN: 800
- SF -> CHI: 1200
- SF -> BOST: ∞
- SF -> NY: ∞
- SF -> MIA: ∞
- SF -> NO: ∞
- LA -> DEN: 1000
- LA -> NO: 1700
- DEN -> CHI: 1000
- DEN -> NO: 1400
- CHI -> BOST: 1500
- CHI -> NY: 1000
- BOST -> NY: 250
- NY -> MIA: 900
- NO -> MIA: 1000

Initial Setup:
Initialize distances to all vertices as infinity, except the source (Boston) which is set to 0.
Mark all vertices as unvisited.

Steps:
Select the vertex with the smallest tentative distance (initially Boston, distance 0).
Update distances to all neighbors of the selected vertex.
Mark the selected vertex as visited.
Repeat until all vertices are visited or the smallest tentative distance among the unvisited vertices is infinity.

Implementation Table:

Iteration	Vertex Selected	S	LA	SF	DEN	CHI	BOST	NY	MIA	NO
Initial	4 (Boston)	{4}	∞	∞	∞	1500	0	250	∞	∞
1	5 (New York)	{4, 5}	∞	∞	∞	1250	0	250	1150	∞
2	6 (Miami)	{4, 5, 6}	∞	∞	∞	1250	0	250	1150	1650
3	3 (Chicago)	{4, 5, 6, 3}	∞	∞	2250	1250	0	250	1150	1650
4	7 (New Orleans)	{4, 5, 6, 3, 7}	3350	∞	2250	1250	0	250	1150	1650
5	2 (Denver)	{4, 5, 6, 3, 7, 2}	3350	3250	2250	1250	0	250	1150	1650
6	1 (San Francisco)	{4, 5, 6, 3, 7, 2, 1}	3350	3250	2250	1250	0	250	1150	1650

Data Structures

- **Adjacency Matrix (graph[V][V]):**
  - A 2D array where graph[i][j] holds the weight of the edge between vertices i and j. If i and j are not directly connected, graph[i][j] should ideally be INT_MAX (to represent infinity), but in your matrix, it is 0, which is not correct for Dijkstra unless 0 is meant to imply no direct connection (which Dijkstra's algorithm doesn't naturally handle as it treats 0 as a valid path of no cost).
- **Distance Array (dist[V]):**
  - An array where dist[i] will hold the shortest distance from the source vertex to vertex i after the algorithm has been executed. Initially, all distances are set to INT_MAX (infinity), except the source vertex, which is set to 0.
- **Shortest Path Tree Set (sptSet[V]):**
  - A boolean array where sptSet[i] is true if the vertex i is included in the shortest path tree, or the shortest distance from the source to i has been finalized.

Functions and Their Roles

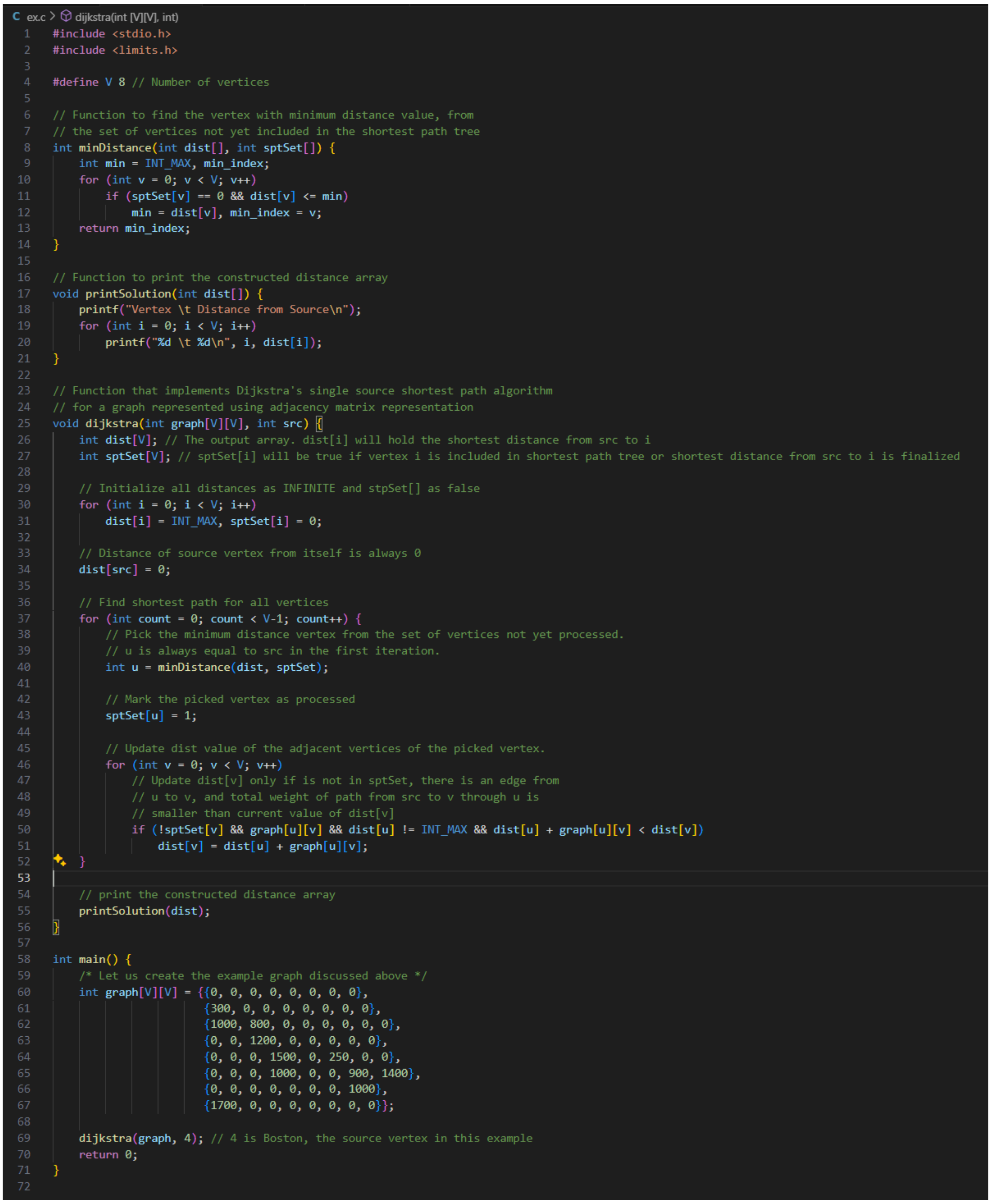
- **minDistance(int dist[], int sptSet[]):**
  - Finds the vertex with the minimum distance value from the set of vertices that have not yet been included in the shortest path tree. It scans the dist array to find a vertex v that has the smallest dist[v] and is not yet in sptSet.
- **printSolution(int dist[]):**
  - Prints the vertices and their distance from the source vertex. This function is called after all shortest paths are found.
- **dijkstra(int graph[V][V], int src):**
  - **Implements Dijkstra's algorithm**:
    - **Initialization**: Sets all distances to INT_MAX and all sptSet entries to false. The distance from the source to itself is set to 0.
    - **Core Algorithm**:
      - Repeatedly finds the vertex u with the minimum distance (which hasn't been included in sptSet).
      - Includes u in sptSet.
      - Updates the distance of all adjacent vertices of u. For each neighbor v of u, if v is not in sptSet, there is an edge from u to v, and the path from the source to v through u offers a shorter distance than the current value of dist[v], then it updates dist[v].

Algorithm

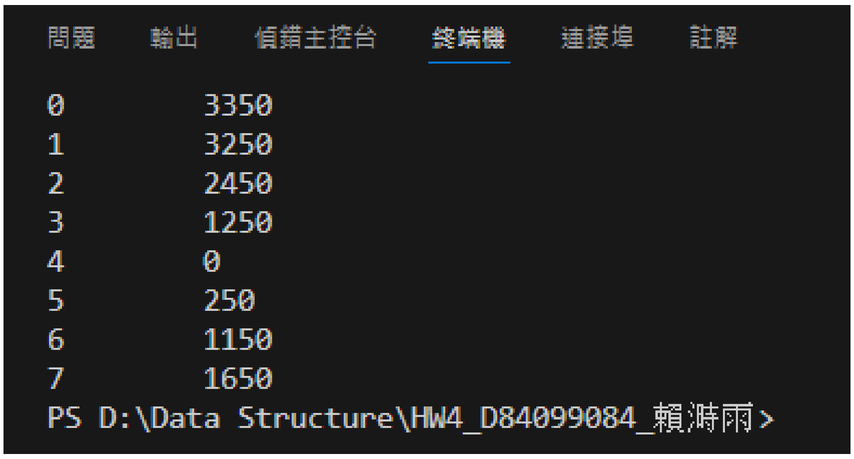
- **Dijkstra's Algorithm**: A greedy algorithm that finds the shortest paths from a single source node to all other nodes in a graph with non-negative weights. It builds up the shortest path step by step, expanding outwards from the source and using a priority mechanism (min-distance) to decide the next vertex to add to the path tree.

Errors and Improvements

- **Graph Initialization**: The graph should correctly represent non-edges with INT_MAX instead of 0, unless there's a specific handling in your graph setup. This is because 0 could be misinterpreted by the algorithm as a zero-cost edge.
- **Edge Case Handling**: Ensure handling of graphs with no possible paths, cyclic paths in directed scenarios, and vertices isolated from the source.



```c
#include <stdio.h>
#include <limits.h>

#define V 8 // Number of vertices

// Function to find the vertex with minimum distance value, from
// the set of vertices not yet included in the shortest path tree
int minDistance(int dist[], int sptSet[]) {
    int min = INT_MAX, min_index;
    for (int v = 0; v < V; v++)
        if (sptSet[v] == 0 && dist[v] <= min)
            min = dist[v], min_index = v;
    return min_index;
}

// Function to print the constructed distance array
void printSolution(int dist[]) {
    printf("Vertex \t Distance from Source\n");
    for (int i = 0; i < V; i++)
        printf("%d \t %d\n", i, dist[i]);
}

// Function that implements Dijkstra's single source shortest path algorithm
// for a graph represented using adjacency matrix representation
void dijkstra(int graph[V][V], int src) {
    int dist[V]; // The output array. dist[i] will hold the shortest distance from src to i
    int sptSet[V]; // sptSet[i] will be true if vertex i is included in shortest path tree or shortest distance from src to i is finalized

    // Initialize all distances as INFINITE and stpSet[] as false
    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX, sptSet[i] = 0;

    // Distance of source vertex from itself is always 0
    dist[src] = 0;

    // Find shortest path for all vertices
    for (int count = 0; count < V-1; count++) {
        // Pick the minimum distance vertex from the set of vertices not yet processed.
        // u is always equal to src in the first iteration.
        int u = minDistance(dist, sptSet);

        // Mark the picked vertex as processed
        sptSet[u] = 1;

        // Update dist value of the adjacent vertices of the picked vertex.
        for (int v = 0; v < V; v++)
            // Update dist[v] only if is not in sptSet, there is an edge from
            // u to v, and total weight of path from src to v through u is
            // smaller than current value of dist[v]
            if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX && dist[u] + graph[u][v] < dist[v])
                dist[v] = dist[u] + graph[u][v];
    }

    // print the constructed distance array
    printSolution(dist);
}

int main() {
    /* Let us create the example graph discussed above */
    int graph[V][V] = {{0, 0, 0, 0, 0, 0, 0, 0},
                       {300, 0, 0, 0, 0, 0, 0, 0},
                       {1000, 800, 0, 0, 0, 0, 0, 0},
                       {0, 0, 1200, 0, 0, 0, 0, 0},
                       {0, 0, 0, 1500, 0, 250, 0, 0},
                       {0, 0, 0, 1000, 0, 0, 900, 1400},
                       {0, 0, 0, 0, 0, 0, 0, 1000},
                       {1700, 0, 0, 0, 0, 0, 0, 0}};

    dijkstra(graph, 4); // 4 is Boston, the source vertex in this example
    return 0;
}
```



```
0        3350
1        3250
2        2450
3        1250
4        0
5        250
6        1150
7        1650
PS D:\Data Structure\HW4_D84099084_賴澔雨>
```