East West University

Algorithm (CSE246)

Project Title : Traveling Salesman Problem

(Project Report)

Submitted by

Name	ID
Md. Naeemur Rahman	2022-1-60-167
Md. Sadat Ahmed Rafi	2022-1-60-261
Ahsan Aftab	2019-1-60-118
Rajia Afrin Rima	2021-2-60-029

Section: 02

Semester: Fall 2023

Submitted to

Jesan Ahammed Ovi

Senior Lecturer

Department of Computer Science and Engineering

Date of Submission: 23 December 2023

TRAVELING SALESMAN PROBLEM

INTRODUCTION

The presented project addresses the Traveling Salesman Problem (TSP), a classic optimization problem in graph theory. The goal is to find the most efficient route that visits each city exactly once and returns to the starting city, minimizing the total travel cost.

SYSTEM DESIGN

Dynamic Programming and Backtracking :

- Dynamic programming and backtracking work together to optimize the TSP solution.
- DP memoizes subproblem solutions, and backtracking navigates through the computed values to find the shortest path

SYSTEM REQUIREMENTS:

Processor: Any modern multi-core processor, preferably dual-core or higher.

RAM: Minimum 4 GB RAM for small to medium-sized instances; additional RAM for larger instances.

Operating System: Compatible with Windows, Linux, macOS.

IDE: Supports standard C++ development, e.g., Visual Studio, Code::Blocks, Dev-C++, Xcode, Visual Studio Code.

CODE IMPLEMENTATION

```
#include <bits/stdc++.h>

using namespace std;

struct node {int vertex, weight;};

vector<vector<int>> matrix;  // Declare matrix as a global variable

vector<struct node> ad[100] = {};

int travel = 0,n, e, u, v, w;;

void G() {

    struct node temp;
```

```
    cout << "\nEnter weight between " << u << " and " << v << " paths : ";
    cin >> w;
                cout<<
"\n_____________________________________________________\n";
                cout<<
"\n_____________________________________________________\n";



        temp.vertex = v;
        temp.weight = w;
        ad[u].push_back(temp);
        temp.vertex = u;  // Reverse the direction for an undirected graph
        ad[v].push_back(temp);
}
void printMatrix(int n){
    for (int i = 1; i <= n; i++) {
        for (int j = 0; j < ad[i].size(); j++) {
            int v = ad[i][j].vertex;
            int weight = ad[i][j].weight;
            matrix[i - 1][v - 1] = weight;
            matrix[v - 1][i - 1] = weight;  //An undirected graph so same
weight in both index
        }
    }
    // Print matrix
    cout << "\nAdjacency Matrix Path Weight Table:\n";
    for (int i = 0; i < n; i++) {
            cout << "|\t";
        for (int j = 0; j < n; j++) {
            cout <<matrix[i][j] << "\t|\t\t";
        }
                cout<<
"\n_____________________________________________________
_____________________________________________\n";
```

```
        cout << endl;
    }
}
int tsp(int dp[101][101]){
    for(int i = 0; i < n; i++){
        for(int j = 0 ; j < ad[i].size();j++){
             if(dp[i][j] != 0){
                return dp[i][j];
            }
            int ans = 999999;
            if(dp[i][j] == 0){
                travel = tsp(dp);
                int newans = ad[i][j].weight + travel;
                ans  = min(newans, ans);
                return dp[i][j] = ans;
}}}}
void printPath(int i, int j, int dp[101][101]) {
    while (j != 0 && i != j) {
        cout << i << " ";
        i = dp[i][j];
    }
    cout << i;
}
int main() {
int dp[101][101];
        cout<< "\t\t
_____\n"
;
        cout<< "\t\t|\t\t\t\t\t\t\t\t\t\t\t|\n";
        cout << "\t\t|\t\t Welcome to Travelling Salesman
Problem\t\t\t|\n";
        cout<<
"\t\t|_____
___|\n\n";
```

```
        cout<< "_____________________________________________________\n";
        cout << "Enter number of cities : ";
        cin >> n;
        cout<< "_____________________________________________________\n";
        cout << "Enter number of paths : ";
        cin >> e;
        cout<< "_____________________________________________________\n";
        matrix.resize(n, vector<int>(n, 0));
        memset(dp, -1, sizeof(dp));

    for (int i = 1; i <= e; i++) {
        cout << "Enter city  " << i  << " : "  ;
        cin >> u;
        cout<< "\n";
        cout << "Enter city  " << i+1  << " : "  ;
        cin >> v;
        G();
    }
    printMatrix(n);
     cout << "\nMinimum Cost is: ";
    int res = tsp(dp);
    cout << "\n" << res;
      cout << "\nShortest Path is: ";
       printPath(1, n,dp); // Print the path starting from city 1
    return 0;
}
```

CODE STRUCTURE

Global Variables:

matrix: A global 2D vector representing the adjacency matrix of the graph.

ad[100]: An array of vectors to store the adjacency list of the graph.

travel: A variable to track the total travel cost during TSP calculations.

Functions:

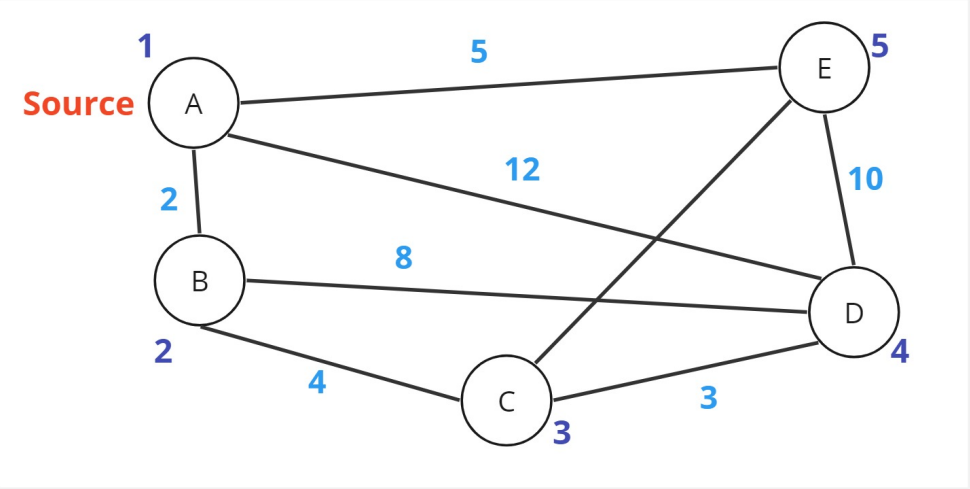
G(): Accepts user input for the weight between two cities and updates the adjacency list.

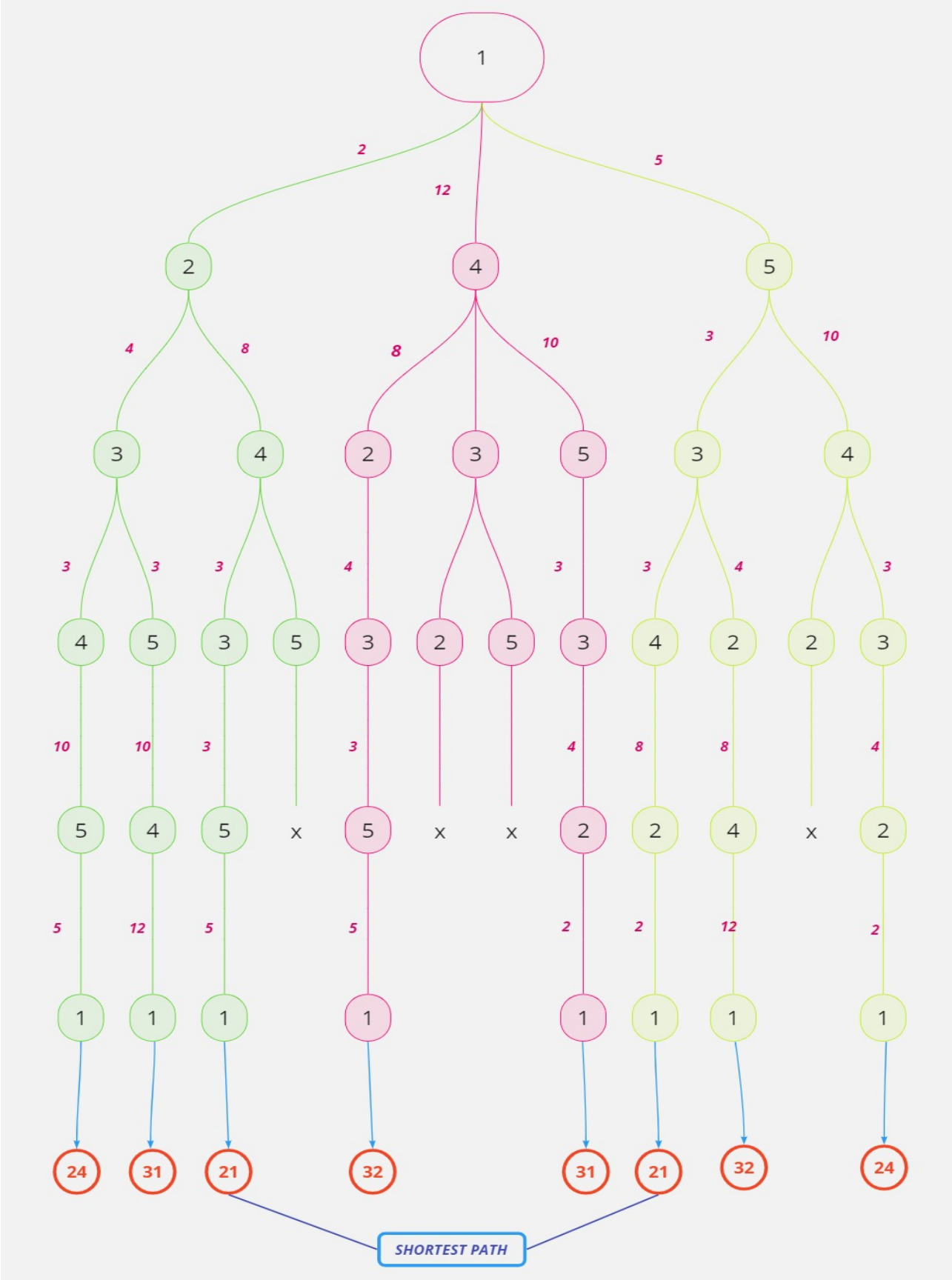
printMatrix(int n): Constructs and prints the adjacency matrix path weight table.

tsp(int dp[101][101]): Implements the Traveling Salesman Problem using dynamic programming to find the minimum cost.

printPath(int i, int j, int dp[101][101]): Prints the shortest path between two cities.

DEMO GRAPH & TREE




1
2
12
5
2
4
5
4
8
8
10
3
10
3
4
2
3
5
3
4
3
3
3
3
4
3
3
4
5
3
5
3
2
5
3
4
2
2
3
10
10
3
3
4
8
8
4
5
4
5
x
5
x
x
2
2
4
x
2
5
12
5
5
2
2
12
2
1
1
1
1
1
1
1
1
24
31
21
32
31
21
32
24
SHORTEST PATH

RECURSIVE ANALYSIS :

$g(1, \{2, 3, 4,5\}) = \min \{C_{12} + g(2,\{3, 4,5\}), C_{12} + g(2,\{3,5,4\}), C_{12} + g(2,\{ 4,3,5\}),$

$C_{14} + g(4,\{2, 3,5\}), C_{14} + g(4,\{5, 3,1\}), C_{15} + g(5,\{3, 4,2\}),$

$C_{15} + g(5,\{3, 2,4\}), C_{15} + g(5,\{4,3,2\})\}$

g (2, Ø) =2

g (3, Ø) =0

g (4, Ø) = 12

g (5, Ø) =5

g (2, {4}) =20

g (3, {2}) =6

g (3,{5}) =8

g (4, {2}) =10

g (4,{5}) =15

g (5,{4}) =22

g (2, {3,5}) =12

g (3, {2,4}) = 24

g (3, {4,2}) =13

g (3, {5,4}) =25

g (3, {4, 5)} =18

g (2, {3,4,5}) =22

g (2, {3,5,4}) =26

g (2, {4,3,5}) =19

g (4, {2,3,5}) =20

g (4, {5,3,2}) = 19

g (5, {3,4,2}) =16

g (5, {3,2,4}) = 27

g (5, {4,3,2}) = 19

g (1, {2,3,4,5}) =24

g (1, {2,3,5,4}) =31

g (1, {2,4,3,5}) =21

g (1, {4,2,3,5}) =32

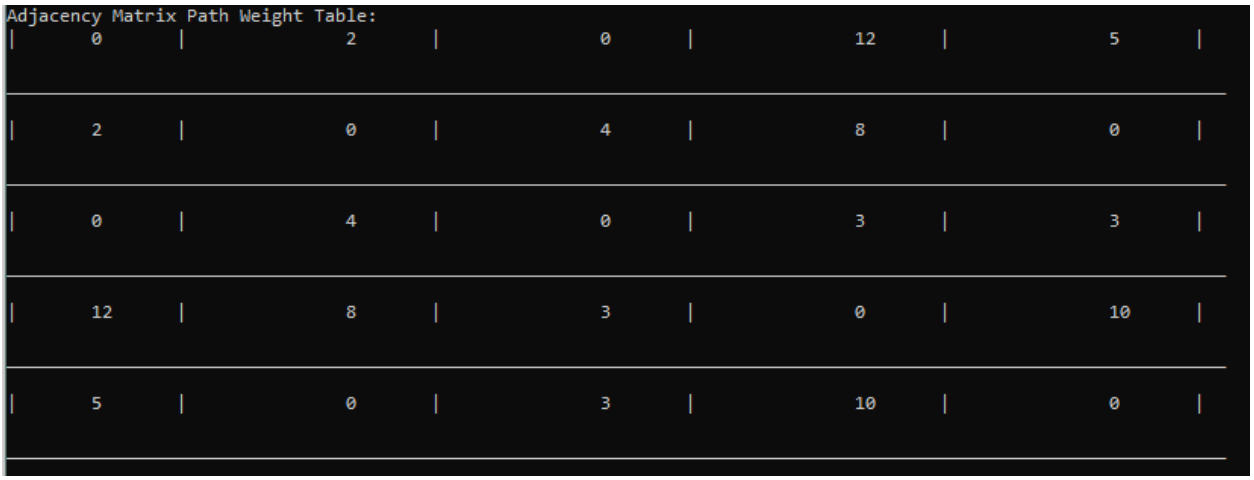
g (1, {4,5,3,2}) =31

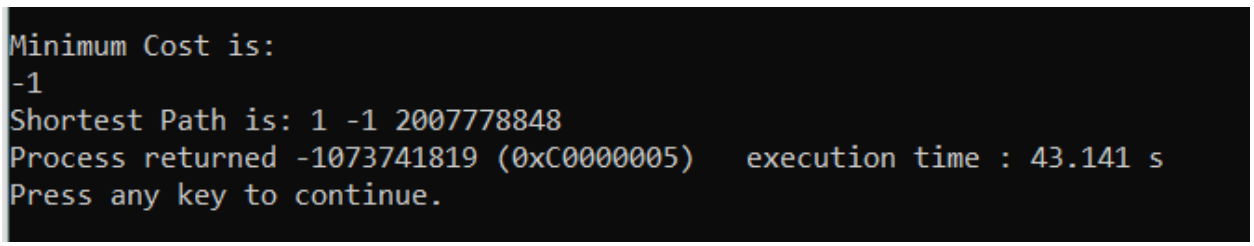
g (1, {5,3,4,2}) = 21

g (1, {5,3,2,4}) =32

g (1, {5,4,3,2}) =24

TESTING RESULTS





FUTURE SCOPE

Limitations:

- The current implementation may face scalability issues for extremely large instances of the Traveling Salesman Problem (TSP) due to the exponential nature of the solution space.
- The dynamic programming approach involves storing solutions in a 2D array (dp[101][101]). For very large graphs, this may consume a significant amount of memory.

Future Scope:

- Implement and integrate more advanced algorithms and heuristics to handle larger instances of the TSP efficiently.
- Extend the project for practical applications, such as route optimization for delivery services, logistics planning, or tour planning in the tourism industry.

OPTIMIZATION:

The program employs dynamic programming to optimize the solution to the Traveling Salesman Problem.

CONCLUSION:

The implemented program successfully solves the Traveling Salesman Problem, providing a valuable tool for finding the optimal route for a salesman visiting a set of cities.

REFERENCE:

The code implementation is based on standard C++ practices and algorithms. No external libraries or references were used in the development of this project.