

### Experiment No. – 2.4

**Student Name: Deepak Saini**

**UID: 20BCS4066**

**Branch: 20BCC1**

**Section/Group: A**

**Semester: 5<sup>th</sup>**

**Date of Performance: 24/09/2022**

**Subject Name: ADVANCED PROGRAMMING LAB**

**Subject Code: 20CSP-334**

**1. Aim/Overview of the practical:**

Implement Travelling Salesperson problem using Dynamic programming

**2. Task to be done:**

Implement Travelling Salesperson problem using Dynamic programming

**3. Steps for practical:**

**Approach:**

**Naive Solution:**

- 1) Consider city 1 as the starting and ending point.
- 2) Generate all  $(n-1)!$  Permutations of cities.
- 3) Calculate the cost of every permutation and keep track of the minimum cost permutation.
- 4) Return the permutation with minimum cost.

**Dynamic Programming:**

Let the given set of vertices be  $\{1, 2, 3, 4, \dots, n\}$ . Let us consider 1 as starting and ending point of output. For every other vertex  $I$  (other than 1), we find the minimum cost path with 1 as the starting point,  $I$  as the ending point, and all vertices appearing exactly once. Let the cost of this path cost  $(i)$ , and the cost of the corresponding Cycle would cost  $(i) + \text{dist}(i, 1)$  where  $\text{dist}(i, 1)$  is the distance from  $I$  to 1. Finally, we return the minimum of all  $[\text{cost}(i) + \text{dist}(i, 1)]$  values. This looks simple so far.

Now the question is how to get  $\text{cost}(i)$ ? To calculate the  $\text{cost}(i)$  using Dynamic Programming, we need to have some recursive relation in terms of sub-problems. Let us define a term  $C(S, i)$  be the cost of the minimum cost path visiting each vertex in set  $S$  exactly once, starting at 1 and ending at  $i$ . We start with all subsets of size 2 and calculate  $C(S, i)$  for all subsets where  $S$  is the subset, then we calculate  $C(S, i)$  for all subsets  $S$  of size 3 and so on. Note that 1 must be present in every subset. Below is the dynamic programming solution for the problem using top down recursive+memoized approach:-

For maintaining the subsets we can use the bitmasks to represent the remaining nodes in our subset. Since bits are faster to operate and there are only few nodes in graph, bitmasks is better to use.

For example: –

10100 represents node 2 and node 4 are left in set to be processed

010010 represents node 1 and 4 are left in subset.

NOTE:- ignore the 0th bit since our graph is 1-based

#### 4. Code:

```
#include <iostream>
```

```
using namespace std;
```

```
// there are four nodes in example graph (graph is 1-based)
```

```
const int n = 4;
```

```
// give appropriate maximum to avoid overflow
```

```
const int MAX = 1000000;

// dist[i][j] represents shortest distance to go from i to j
// this matrix can be calculated for any given graph using
// all-pair shortest path algorithms
int dist[n + 1][n + 1] = {
    { 0, 0, 0, 0, 0 }, { 0, 0, 10, 15, 20 },
    { 0, 10, 0, 25, 25 }, { 0, 15, 25, 0, 30 },
    { 0, 20, 25, 30, 0 },
};

// memoization for top down recursion
int memo[n + 1][1 << (n + 1)];

int fun(int i, int mask)
{
    // base case
    // if only ith bit and 1st bit is set in our mask,
    // it implies we have visited all other nodes already
    if (mask == ((1 << i) | 3))
        return dist[1][i];
    // memoization
    if (memo[i][mask] != 0)
        return memo[i][mask];

    int res = MAX; // result of this sub-problem

    // we have to travel all nodes j in mask and end the
    // path at ith node so for every node j in mask,
    // recursively calculate cost of travelling all nodes in
    // mask except i and then travel back from node j to
    // node i taking the shortest path take the minimum of
```

```
// all possible j nodes

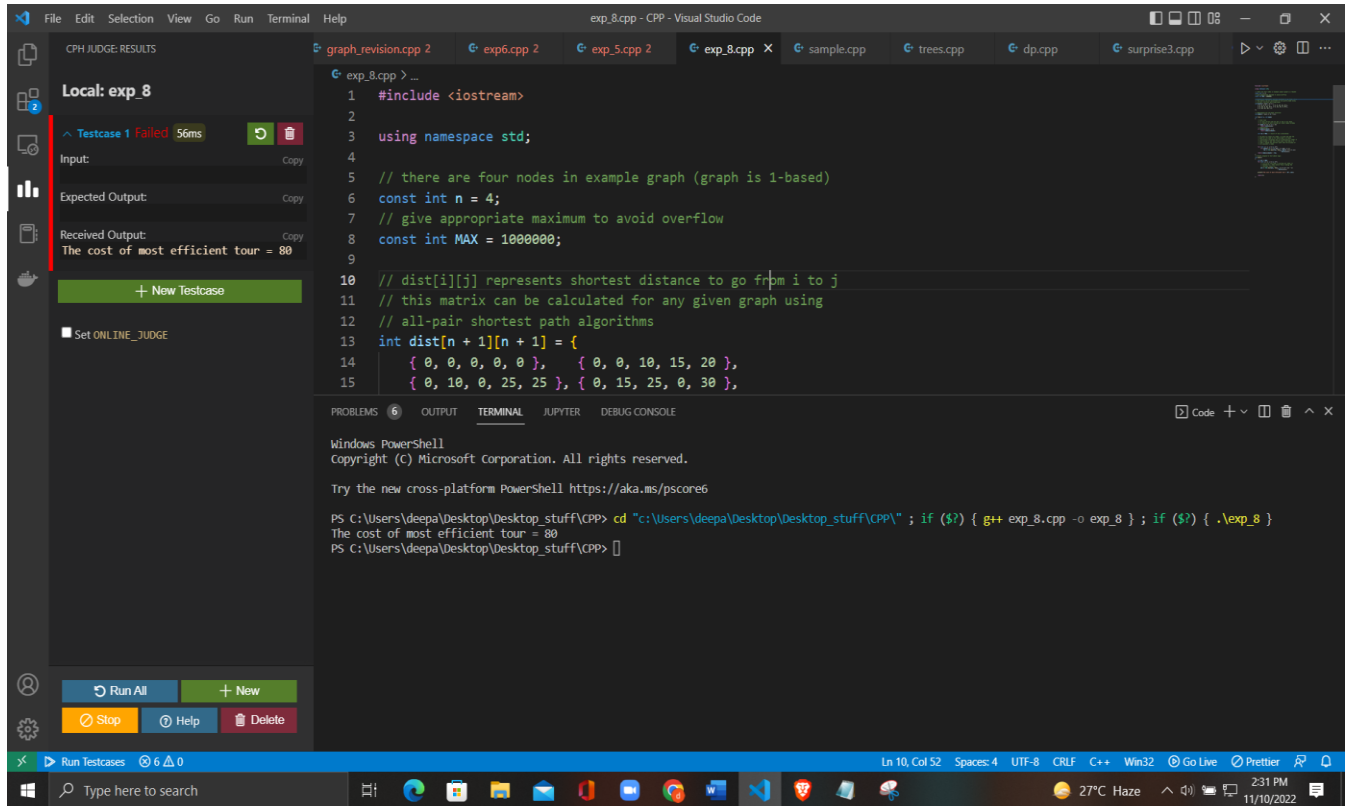
for (int j = 1; j <= n; j++)
    if ((mask & (1 << j)) && j != i && j != 1)
        res = std::min(res, fun(j, mask & ~(1 << i)))
            + dist[j][i]);
return memo[i][mask] = res;
}
// Driver program to test above logic
int main()
{
    int ans = MAX;
    for (int i = 1; i <= n; i++)
        // try to go from node 1 visiting all nodes in
        // between to i then return from i taking the
        // shortest route to 1
        ans = std::min(ans, fun(i, (1 << (n + 1)) - 1)
            + dist[i][1]);

    printf("The cost of most efficient tour = %d", ans);

    return 0;
}
```

## 5. Output:

a)



```

exp_8.cpp - CPP - Visual Studio Code
graph_revison.cpp 2 exp_6.cpp 2 exp_5.cpp 2 exp_8.cpp X sample.cpp trees.cpp dp.cpp surprise3.cpp
CPH JUDGE: RESULTS
Local: exp_8
Testcase 1 Failed 56ms
Input:
Expected Output:
Received Output:
The cost of most efficient tour = 80
+ New Testcase
Set ONLINE_JUDGE

exp_8.cpp > ...
1 #include <iostream>
2
3 using namespace std;
4
5 // there are four nodes in example graph (graph is 1-based)
6 const int n = 4;
7 // give appropriate maximum to avoid overflow
8 const int MAX = 1000000;
9
10 // dist[i][j] represents shortest distance to go from i to j
11 // this matrix can be calculated for any given graph using
12 // all-pair shortest path algorithms
13 int dist[n + 1][n + 1] = {
14     { 0, 0, 0, 0 }, { 0, 0, 10, 15 },
15     { 0, 10, 0, 25 }, { 0, 15, 25, 0 }
16 };
17
18 int main() {
19     // ...
20 }

PROBLEMS 6 OUTPUT TERMINAL JUPYTER DEBUG CONSOLE
Windows PowerShell
Copyright (c) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\deepa\Desktop\Desktop_stuff\CPP> cd "c:\Users\deepa\Desktop\Desktop_stuff\CPP" ; if ($?) { g++ exp_8.cpp -o exp_8 }; if ($?) { .\exp_8 }
The cost of most efficient tour = 80
PS C:\Users\deepa\Desktop\Desktop_stuff\CPP>
  
```



```

PROBLEMS 6 OUTPUT TERMINAL JUPYTER DEBUG CONSOLE
Code + v [ ] [ ] ^ X

Windows PowerShell
Copyright (c) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\deepa\Desktop\Desktop_stuff\CPP> cd "c:\Users\deepa\Desktop\Desktop_stuff\CPP" ; if ($?) { g++ exp_8.cpp -o exp_8 }; if ($?) { .\exp_8 }
The cost of most efficient tour = 80
PS C:\Users\deepa\Desktop\Desktop_stuff\CPP>
  
```

## 6. Learning Outcomes:

- To learn the basics of Graph to how to take inputs.
- To learn the approach to how to solve problems related to graph.
- To learn about how to use stack data structure.
- To solve the problems using Dynamic Programming.

**Evaluation Grid (To be created as per the SOP and Assessment guidelines by the faculty):**

Sr. No.	Parameters	Marks Obtained	Maximum Marks
1.			
2.			
3.			