REVIEW – 2

COURSE CODE : CSE2003

DATA STRUCTURES AND ANALYSIS

TRAVELLING SALESMAN PROBLEM

VANDIT JAIN-18BCE2135

PIYUSH BAJAJ-18BCE2146

PALIMAR ABHISHEK RAO-18BCE2204

ABSTRACT

Computers have become indispensable part of our lives. Computers, logical devices are present everywhere computing the shortest and fastest method to solve the problem. With the growing population, it has become vital to operate effectively in order to full fill the demands of more people in less time. In 21st century, we can't afford to use a lot of time, expend a lot of energy and space to solve a problem. We have to be change our approach.

To solve this traveling salesman problem we have to find the sequence of traveling places such that the traveller returns to starting place by shortest route.

The Traveling salesman problem(TSP) is considered as one of the most well-known combinatorial optimization tasks and researchers have paid so much attention to the TSP for many years. In this problem, a salesman starts to move from an arbitrary place called depot and after visits all of the nodes where he should deliver, finally comes back to the depot. The objective is to minimize the total distance travelled by the salesman.

AIM-Comparing the performances of greedy algorithm, backtracking, dynamic programming, and brute-force approach to solve Traveling Salesman Problem. Given a set of cities and distance between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point.

OBJECTIVE- We are designing a greedy algorithm in this program. We'll follow 3 simple steps to complete the project which are Input, operation and output. The steps are:

1) Get the inputs which comprises of names of places and then run a loop to collect the individual distance between every pair of cities.
2) Store the input data in the list/ structs for ease of operating and for instant outputs.
3) After recording all the city names and the distances between them operate on them.
4) Design algorithms to achieve all the possible routes to travel all the Places.
5) Find all the sequences of traveling the places such that the route is the shortest.
6) Make the sequence user-friendly and print the sequence.
7) Compare the performances of greedy algorithm, backtracking, dynamic programming and brute force approach with the help of time complexity, spatial complexity and space and time complexity.
8) Print the conclusion

APPLICABILITY- Delivery companies like amazon, flipkart and etc. have a lot of trucks each weekday each truck starts at a depot, make N stops and returns to the depot. I don't know how large n would be for a typical truck, but I would guess probably somewhere between 20 and 50. Suppose that these companies could compute a matrix of costs to get the truck between every pair of the n stops. Then, given this matrix, these companies would like to solve the traveling-salesman problem for that truck, so that its route has the lowest overall cost.

INTRODUCTION:

In the traveling salesman problem, a map of cities is given to the salesman and he has to visit all the cities only once and return to his starting point to complete the tour in such a way that the length of the tour is the shortest among all possible tours for this map. Clearly starting from a given city, the salesman will have a total of (n-1)! Different sequences. If n = 2, A and B, there is no choice. If n = 3, i.e. he wants to visit three cities inclusive of the starting point, he has 2! Possible routes and so on. Dynamic programming (usually referred to as DP) is a very powerful technique to solve a particular class of problems. It demands very elegant formulation of the approach and simple thinking and the coding part is very easy. The idea is very simple, If you have solved a problem with the given input, then save the result for future reference, so as to avoid solving the same problem again. If the given problem can be broken up in to smaller sub-problems and these smaller subproblems are in turn divided in to still-smaller ones, and in this process, if you observe some over-lappping subproblems, then its a big hint for DP. Also, the optimal solutions to the subproblems contribute to the optimal solution of the given problem.

EXISTING METHODS AND DRAWBACKS WITH THE EXISTING METHOD:

There have been many approaches to solving the Traveling Salesman Problem (TSP).These approaches range from a simple heuristic algorithm to algorithms based on the physical workings of the human mind to those based on ant colonies. These algorithms all have the same ultimate goal: in a graph with weighted edges, find the shortest (the path through all nodes with the smallest sum of edge weights). Unfortunately, this goal is very hard to achieve. The algorithms therefore settle for trying to accomplish two smaller goals:

(1) To more quickly find a good solution.

(2) To find a better good solution. A good solution is one that is close to being optimal and the best of these good solutions is, of course, the optimal solution itself.

Drawbacks with the existing methods:

1)Huge memory requirements.

2)Time complexity.

3)Difficult to manage everything as everything used in the program is very old.

PROPOSED METHOD AND ADVANTAGES OF PROPOSED METHOD:

Dynamic programming : Let the given set of vertices be {1, 2, 3, 4,upto n numbers}. 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 be cost(i), the cost of corresponding Cycle would be cost(i) + dist(i, 1) where dist(i, 1) is the distance from i to 1. Finally, we return the minimum of all values.

To calculate 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.

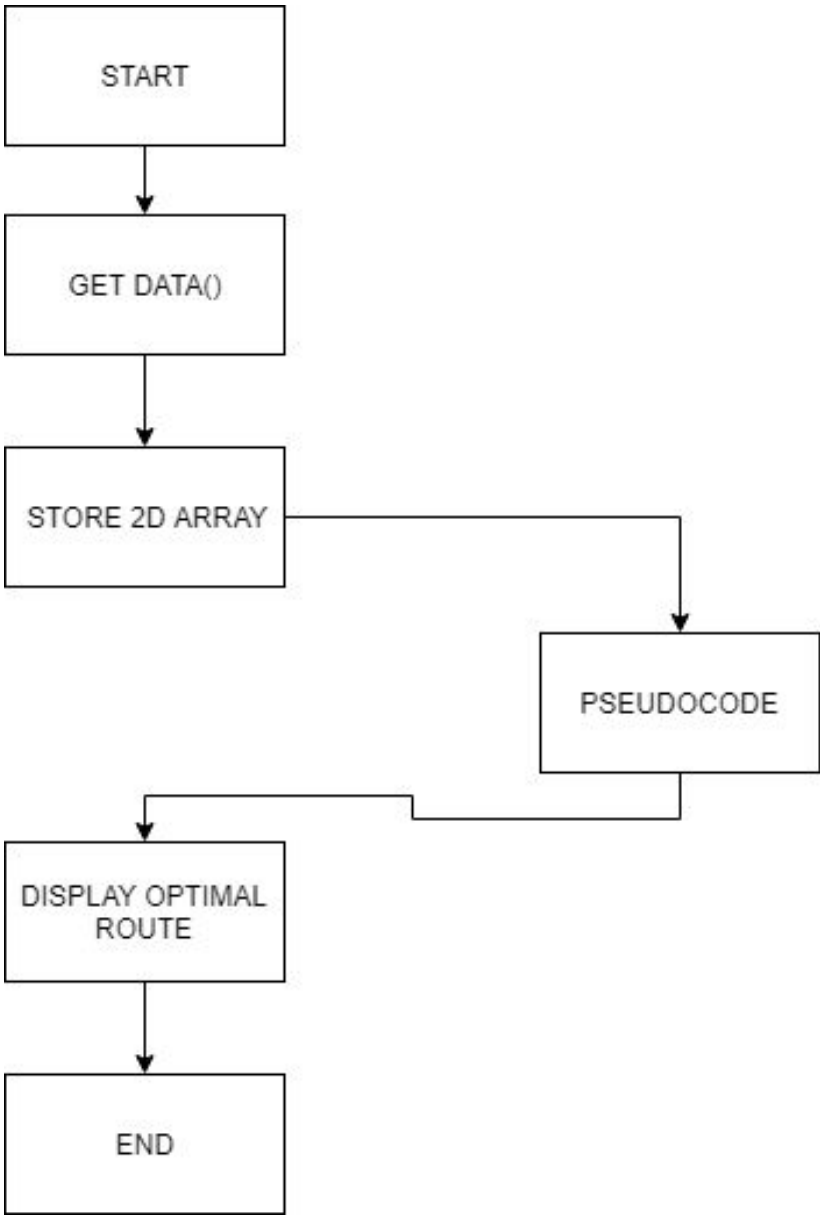
Using the above recurrence relation, we can write dynamic programming based solution.

Brute Force If you think about it, representing a TSP tour is not too hard: it is just a *permutation* of the cities, with the added restriction that the first city must always be 0.

Examine all possible permutations of cities, and keep the one that is shortest.

It turns out that there are exactly $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-1) \cdot n$ different permutations of the numbers from 0 to $n-1$. Since we only care about permutations that start with 0, to solve an n-city TSP instance with brute force requires that we look at exactly $(n-1)! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-2) \cdot (n-1)$ different permutations.

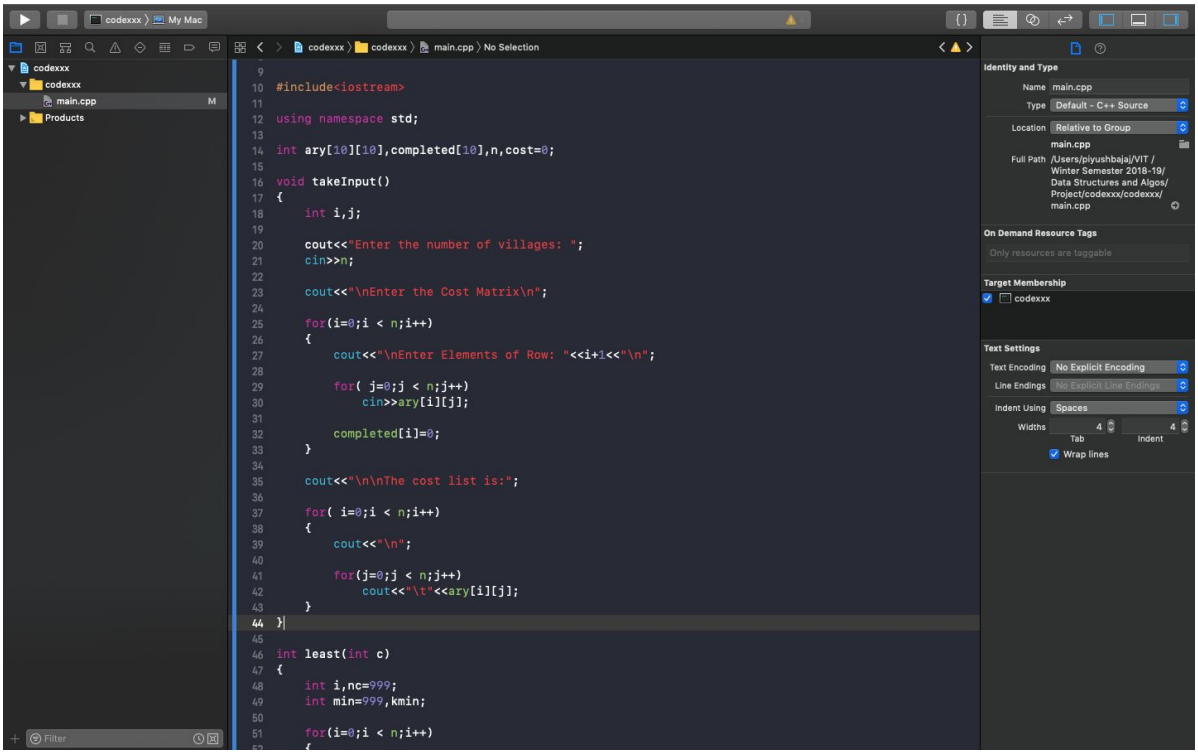
PROPOSED SYSTEM ARCHITECTURE:



IMPLEMENTATION DETAILS WHICH DESCRIBES THE MODULES IN YOUR PROJECT:

We made this program using Data Structures which was new for us so that we got a chance to write and implement our codes in a new manner. It was really exciting and was really easy to handle it. Firstly, it was new to us so we had difficulties in running the code but later on we watched different videos on YouTube due to which later on it was good. Firstly, we prepared the backbone of the project that is we created the data structure in which we stored all the distance between each town and we processed that information to minimize the distance travelled by the salesman and we used program to find the shortest path in the figure below.

TESTING AND SAMPLE SCREENSHOTS WITH OUTPUT:



```
#include<iostream>

using namespace std;

int ary[10][10],completed[10],n,cost=0;

void takeInput()
{
    int i,j;

    cout<<"Enter the number of villages: ";
    cin>>n;

    cout<<"\nEnter the Cost Matrix\n";

    for(i=0;i < n;i++)
    {
        cout<<"\nEnter Elements of Row: "<<i+1<<"\n";

        for( j=0;j < n;j++)
            cin>>ary[i][j];

        completed[i]=0;
    }

    cout<<"\n\nThe cost list is:";

    for( i=0;i < n;i++)
    {
        cout<<"\n";

        for(j=0;j < n;j++)
            cout<<"\t"<<ary[i][j];
    }
}

int least(int c)
{
    int i,nc=999;
    int min=999,kmin;

    for(i=0;i < n;i++)
```

```
public:
  // add a city at point p
  void add(const Point& p) {
    cities.push_back(p);
  }

  // add a city at (x, y)
  void add(double x, double y) {
    Point p(x, y);
    add(p);
  }

  // # of cities
  int size() const {
    return cities.size();
  }

  // calculate the distance between cities i and j
  double dist_between(int i, int j) const {
    return dist(cities[i], cities[j]);  // dist is from Point.h
  }

  // calculate the score of the given tour
  double score(const vector<int>& tour) const {
    double result = dist_between(tour[0], tour[size() - 1]);
    for(int i = 1; i < size(); ++i) {                // i starts at 1, not 0!
      result += dist_between(tour[i - 1], tour[i]);
    }
    return result;
  }
```

```
Last login: Wed Feb 27 22:29:35 on ttys000
/Users/piyushbajaj/VIT\ /Winter\ Semester\ 2018-19/Data\ Structures\ and\ Algos/Project/code ; exit;
Piyushs-MacBook-Air:~ piyushbajaj$ /Users/piyushbajaj/VIT\ /Winter\ Semester\ 2018-19/Data\ Structures\ and\ Algos/Project/code ; exit;
Enter the number of villages: 3

Enter the Cost Matrix

Enter Elements of Row: 1
12
23
5

Enter Elements of Row: 2
23
11
14

Enter Elements of Row: 3
45
3
28


The cost list is:
       12      23      5
       23      11      14
       45      3       28

The Path is:
1--->2--->3--->1

Minimum cost is 82logout
Saving session...
...copying shared history...
...saving history...truncating history files...
...completed.

[Process completed]
```

```
Enter Elements of Row: 5
76
66
54
89
91


The cost list is:
       12      23      33      26      5
       14      10      9       8       6
       5       8       67      56      43
       31      54      34      48      96
       76      66      54      89      91

The Path is:
1--->2--->3--->4--->5--->1

Minimum cost is 260logout
Saving session...
...copying shared history...
...saving history...truncating history files...
...completed.

[Process completed]
```

CONCLUSION AND FUTURE ENHANCEMENTS:

In this way we created the project using data structures. We got opportunity to learn different algorithms. We enjoyed a lot in this project. It was challenging and we were scared. We were really worried about the project but slowly due to involvement of each member in our group it was really interesting at the end we were collected knowledge and implemented our project. This travelling salesman problem has several applications such as planning, scheduling, logistics and packing. The project has a very vast scope in future.Project can be updated in near future asand when requirement for the same arises, as it is very flexible in terms of expansion. The following are the future scope for the project.

1) It can be used in army war plans to conquer a state.
2) It can be used by school buses and college buses.
3) It can be used for making travelling packages.

REFERENCES:

https://people.sc.fsu.edu/~jburkardt/cpp_src/tsp_brute/tsp_brute.html
http://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/materials/cppdoc/point-h.html
http://www.cs.sfu.ca/CourseCentral/125/tjd/tsp_example.html
http://lcm.csa.iisc.ernet.in/dsa/node186.html
https://github.com/topics/travelling-salesman-problem?l=c%2B%2B&o=asc&s=stars
http://www.martinbroadhurst.com/traveling-salesman-problem-using-backtracking-in-c.html