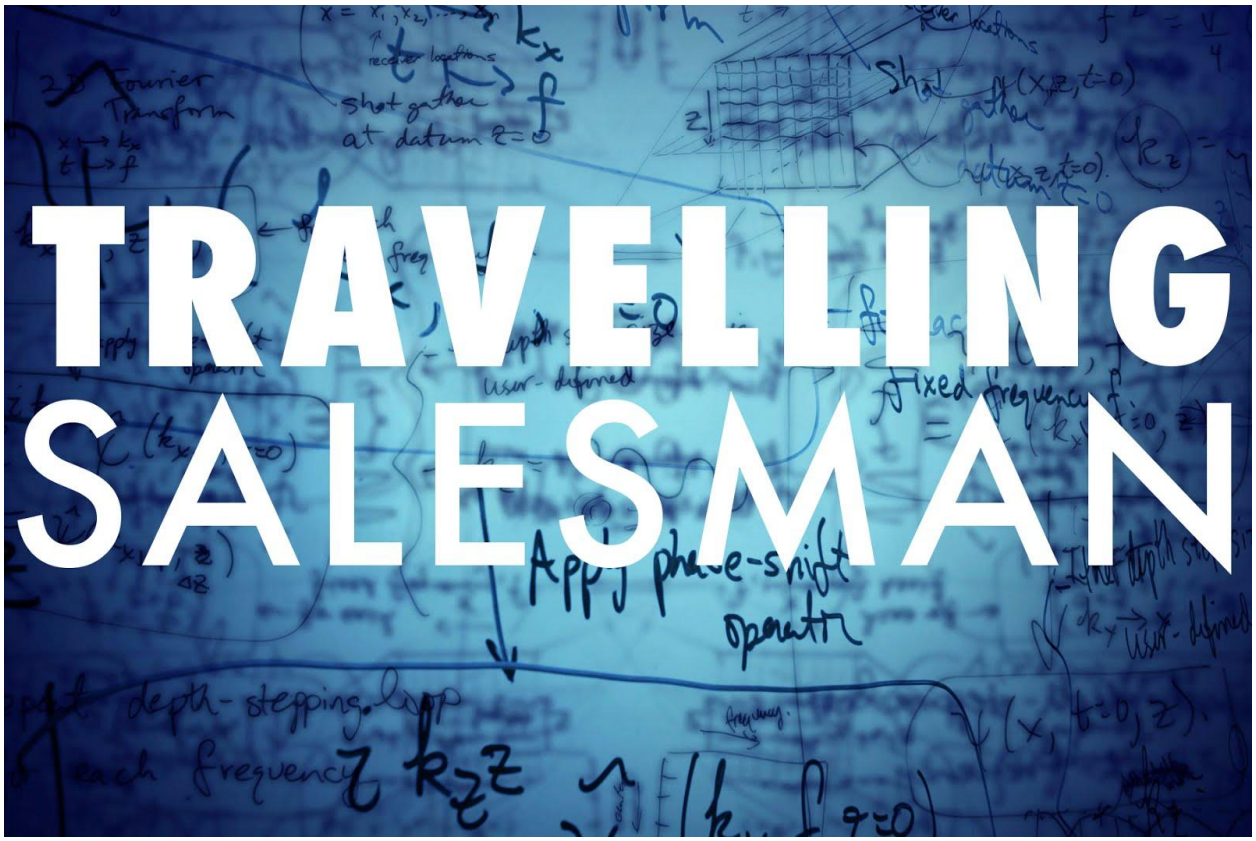
TRAVELLING SALESMAN PROBLEM



Group ID - 8

14114002

14114068

PROBLEM STATEMENT

Take the number of cities (n) from the user as input. Randomly generate a graph with n vertices and some undirected edges with randomly assigned edge-weights. Implement the dynamic programming based algorithm for solving the Travelling Salesman Problem for the weighted graph. Show the outputs for smaller size problems and compare the CPU times for varying size (n) problems.

TABLE OF CONTENTS

MEMBERS:

1. Abhishek Jaisingh (14114002)
2. Tarun Kumar (14114068)

INTRODUCTION

The **travelling salesman problem** (**TSP**) asks the following question: *Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?* It is an NP-hard problem in combinatorial optimization, important in operations research and theoretical computer science.

The problem is a famous **NP hard** problem. There is no polynomial time know solution for this problem.

ALGORITHM

Let the given set of vertices be {1, 2, 3, 4,….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 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 [cost(i) + dist(i, 1)] values. This looks simple so far. Now the question is how to get cost(i)?

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.*

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.

The subproblems are ordered by |S|. Here's the code.

$C(\{1\}, 1) = 0$
for $s = 2$ to n:
    for all subsets $S \subseteq \{1, 2, \ldots, n\}$ of size s and containing 1:
        $C(S, 1) = \infty$
        for all $j \in S, j \neq 1$:
            $C(S, j) = \min\{C(S - \{j\}, i) + d_{ij} : i \in S, i \neq j\}$
return $\min_j C(\{1, \ldots, n\}, j) + d_{j1}$

CODE

This program takes choice whether to take input, generate random input or read input from file. If we select random input, enter the number of vertices.

Main.py

```
from bitwise_manipulations import *
from math import isinf
from helper import *
import json, time

a = []
random_size = 10
def choose(n):
	global a, random_size
	if n == 1:
		a = getInputFromUser()
	if n == 2:
		print("Enter value of n:")
		random_size = int(input())
		a = generateGraph(random_size)
	if n == 3:
		a = readFromFile()


def generateSubsets(n):
	l = []
	for i in range(2**n):
		l.append(i)
	return sorted(l, key = lambda x : size(x) )


def tsp():
	global a
	n = len(a)
```

```
        l = generateSubsets(n)
        cost = [ [-1 for city in range(n)] for subset in l]
        p = [ [-1 for city in range(n)] for subset in l]

        pretty(a)
        t1 = time.time()
        count = 1
        total = len(l)
        for subset in l:
                for dest in range(n):
                        if not size(subset):
                                cost[subset][dest] = a[0][dest]
                                #p[subset][dest] = 0
                        elif (not inSubset(0, subset)) and (not inSubset(dest, subset)) :
                                mini = float("inf")
                                for i in range(n):
                                        if inSubset(i, subset):
                                                modifiedSubset = remove(i, subset)
                                                val = a[i][dest] + cost[modifiedSubset][i]

                                                if val < mini:
                                                        mini = val
                                                        p[subset][dest] = i

                                if not isinf(mini):
                                        cost[subset][dest] = mini
                #print("%f %%" % (100.0*count / total))
                count += 1
        path = findPath(p)
        t2 = time.time()
        diff = t2 - t1
        print(" => ".join(path))

        Cost = cost[2**n-2][0]
        print(Cost)
        print("Time Taken: %f milliseconds" % (diff * 1000))


if __name__ =="__main__":
        print("Enter the choice, 1-to enter Input, 2-generate random Input, 3-read from
\"input.json\" file")
        choice = int(input())
        choose(choice)
        tsp()
```

Helper.py

```
from bitwise_manipulations import *
import time
import random, json


def inSubset(i, s):
        while i > 0 and s > 0:
                s = s >> 1
                i -= 1
        cond = s & 1
```

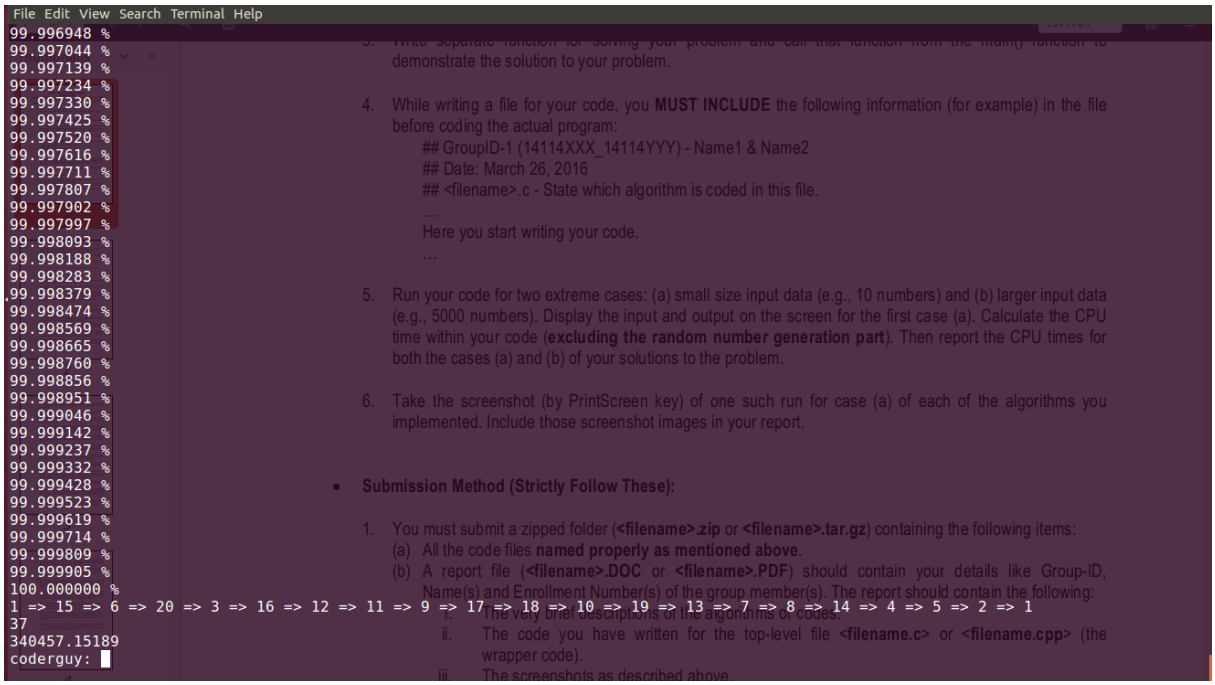
```
        return cond

def remove(i, s):
        x = 1
        x = x << i
        l = length(s)
        l = 2 ** l - 1
        x = x ^ l
        #print ( "i - %d x - %d  s - %d x&s -  %d " % (i, x, s, x & s) )
        return x & s

def findPath(p):
        n = len(p[0])
        number = 2 ** n - 2
        prev = p[number][0]
        path = []
        while prev != -1:
                path.append(prev)
                number = remove(prev, number)
                prev = p[number][prev]
        reversepath = [str(path[len(path)-i-1]+1) for i in range(len(path))]
        reversepath.append("1")
        reversepath.insert(0, "1")
        return reversepath

def pretty(a):
        print("=========================")
        for i in range(len(a)):
                for j in range(len(a[0])):
                        print "%2d"%(a[i][j]),
                print("")
        print("=========================")

def generateGraph(n):
        a = [ [-1 for i in range(n)] for j in range(n)]
        for i in range(n):
                for j in range(n):
                        rand = random.randint(0, n)
                        if a[i][j] < 0:
                                a[i][j] = rand
                                a[j][i] = rand
                        if i == j:
                                a[i][i] = 0

        #pretty(a)
        return a

def getInputFromUser():
        n = int(input("Enter number of cities:"))
        print("Enter the values like 1st row, 2nd row and so on.")
        a = [[int(input()) for i in range(n)] for j in range(n)]
        print(a)
        return a

def readFromFile():
        with open('input.json', 'r') as f:
                s = f.read()
                data  = json.loads(s)
                print(data)
                return data
```

Bitwise_manipulations.py

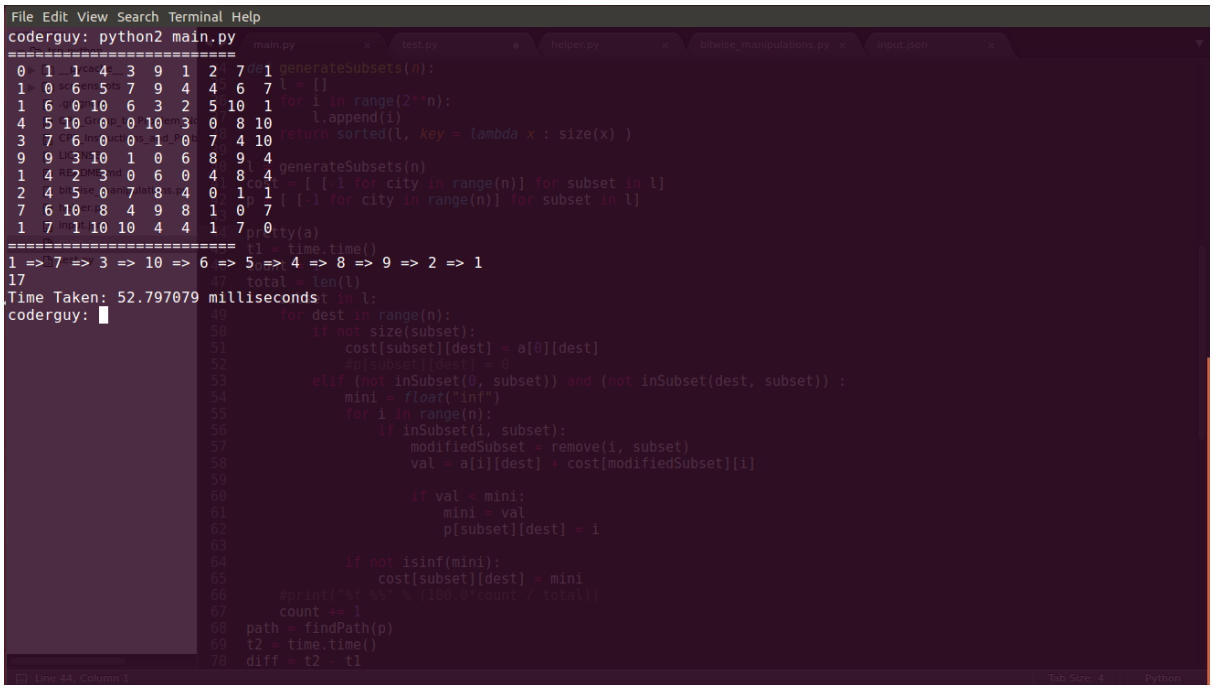
```
def size(int_type):
    length = 0
    count = 0
    while (int_type):
        count += (int_type & 1)
        length += 1
        int_type >>= 1
    return count

def length(int_type):
    length = 0
    count = 0
    while (int_type):
        count += (int_type & 1)
        length += 1
        int_type >>= 1
    return length
```

SCREENSHOTS

For large input(20 cities/vertices)



For small input(10 cities/vertices)



REFERENCES

1. https://en.wikipedia.org/wiki/Travelling_salesman_problem
2. http://www.geeksforgeeks.org/travelling-salesman-problem-set-1/
3. http://www.mafy.lut.fi/study/DiscreteOpt/tspdp.pdf