Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут ім. Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації та управління

ЗВІТ

про виконання лабораторного практикуму №9

Виконав:

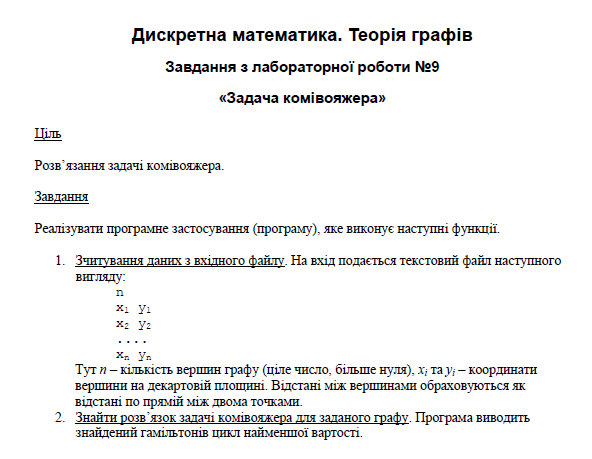
студент 1- го курсу ФІОТ

групи *ІП-91*

*Кінчур Вадим Вікторович*

Київ 2020

1. Умова лабораторної роботи



1. Програмний код (Python)

# Modules  
from math import hypot  
from copy import copy  
from math import ceil  
  
# Const  
INF = float('inf')  
  
  
# Static methods  
def get\_input(file\_name):  
 with open(file\_name, 'r') as file\_handler:  
 v\_num = int(file\_handler.readline())  
 return [tuple(map(int, line.split())) for line in file\_handler.readlines()]  
  
  
# Calculate distance between points  
def get\_distance(p1, p2):  
 return hypot(p1[0] - p2[0], p1[1] - p2[1])  
  
  
# Get n order minimal element in array  
def n\_min(arr, n):  
 return sorted(arr)[n - 1]  
  
  
# Form adjacency matrix  
def get\_adj\_matrix(points):  
 v\_num = len(points)  
 adj\_m = [[INF] \* v\_num for \_ in range(v\_num)]  
 for i in range(v\_num):  
 for j in range(v\_num):  
 if i != j:  
 adj\_m[i][j] = get\_distance(points[i], points[j])  
 return adj\_m  
  
  
# Class for solving TSP problem  
class Solver:  
 def \_\_init\_\_(self, adj\_m):  
 self.\_adj\_m = copy(adj\_m)  
 self.\_cost = INF  
 self.\_v\_num = len(adj\_m)  
 self.\_path = [-1] \* (self.\_v\_num + 1)  
 self.\_visited = [False] \* self.\_v\_num  
  
 # Save temporary solution  
 def \_\_copy\_path(self, curr\_path):  
 self.\_path[:self.\_v\_num + 1] = copy(curr\_path)  
 self.\_path[self.\_v\_num] = curr\_path[0]  
  
 # Do main work  
 def \_\_TSPRecursive(self, curr\_bound, curr\_cost, level, curr\_path):  
 # Form answer on last level  
 if level == self.\_v\_num:  
 curr\_result = curr\_cost + self.\_adj\_m[curr\_path[-1]][curr\_path[0]]  
 if curr\_result < self.\_cost:  
 self.\_\_copy\_path(curr\_path)  
 self.\_cost = curr\_result  
 return  
 # Else form new tree on this level  
 for i in range(self.\_v\_num):  
 if not self.\_visited[i]:  
 temp\_bound = curr\_bound  
 curr\_cost += self.\_adj\_m[curr\_path[level - 1]][i]  
 if level == 1:  
 curr\_bound -= (n\_min(self.\_adj\_m[curr\_path[0]], 1) + n\_min(self.\_adj\_m[i], 1)) / 2  
 else:  
 curr\_bound -= (n\_min(self.\_adj\_m[curr\_path[0]], 2) + n\_min(self.\_adj\_m[i], 1)) / 2  
 # If found new minimal  
 if curr\_bound + curr\_cost < self.\_cost:  
 self.\_visited[i] = True  
 curr\_path[level] = i  
 self.\_\_TSPRecursive(curr\_bound, curr\_cost, level + 1, curr\_path)  
 # Else backtrack and reset all changes  
 curr\_cost -= self.\_adj\_m[curr\_path[level - 1]][i]  
 curr\_bound = temp\_bound  
 self.\_visited = [False] \* self.\_v\_num  
 for index in range(self.\_v\_num):  
 if curr\_path[index] != -1:  
 self.\_visited[index] = True  
  
 # Form answer  
 def \_\_TSP(self):  
 curr\_bound = 0  
 curr\_path = [-1] \* (self.\_v\_num + 1)  
 visited = [False] \* self.\_v\_num  
  
 for row in self.\_adj\_m:  
 curr\_bound += n\_min(row, 1) + n\_min(row, 2)  
 curr\_bound = ceil(curr\_bound / 2)  
 self.\_visited[0] = True  
 curr\_path[0] = 0  
 self.\_\_TSPRecursive(curr\_bound, 0, 1, curr\_path)  
  
 # Print formed result  
 def print\_result(self):  
 self.\_\_TSP()  
 print(f'Minimal way cost is equal to {self.\_cost:.4f}')  
 print('Current way :')  
 for i in range(self.\_v\_num + 1):  
 print(self.\_path[i] + 1, end=' ')  
 if i != self.\_v\_num:  
 print('-> ', end='')  
  
  
# Main method  
def main():  
 points = get\_input('input.txt')  
 adj\_m = get\_adj\_matrix(points)  
 solver = Solver(adj\_m)  
 solver.print\_result()  
  
  
main()

1. Результати виконання

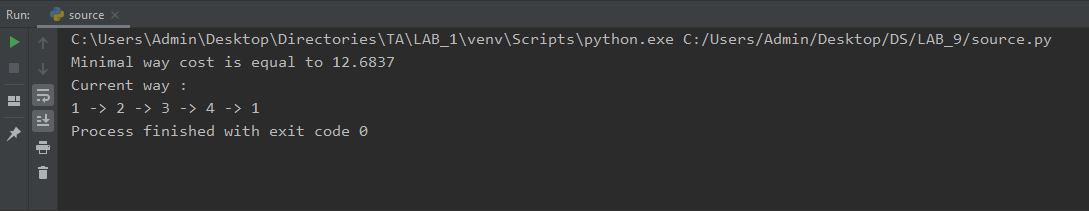
**Вхідні дані:**

4

1 2

2 5

3 7

4 3

**Вхідні дані:**

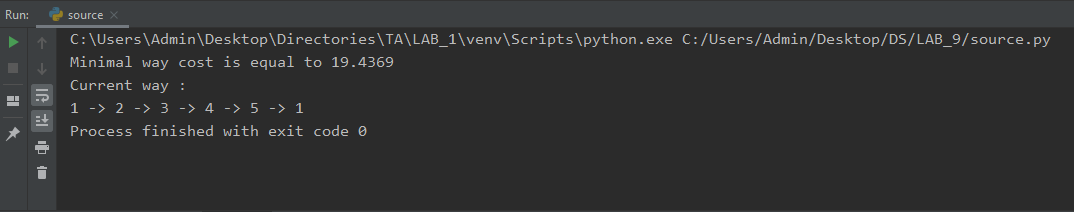
5

1 3

2 1

3 5

6 3

4 7