Національний технічний університет України «КПІ ім. Ігоря Сікорського»

Факультет Інформатики та Обчислювальної Техніки

Кафедра Автоматизованих Систем Обробки Інформації та Управління

Лабораторна робота № 9

з дисципліни «Проектування та аналіз обчислювальних алгоритмів»

на тему

«Жадібні алгоритми»

Виконав:

студент гр. ІС-91

Хмелiнiн Андрiй

Викладач:

[ст.вик. Халус О. А.](http://rozklad.kpi.ua/Schedules/ViewSchedule.aspx?v=318f7459-ddf0-47ee-b882-5650e756cf2a)

Київ – 2020

Постановка задачi

Завдання

У даній роботі необхідно запропонувати жадібний алгоритм для задачі комівояжера.

Задача комівояжера формулюється для повного графу. Для зваженого повного графу G з n вершинами задані відстані між усіма парами вершин (i, j). Необхідно знайти найкоротший маршрут, який проходить через всі вершини графу та заходить в кожну вершину лише один раз.

В даній роботі розглядається симетричний варіант задачі, коли відстань від міста *i* до міста *j* дорівнює відстані від *j* до *і* (відстані між (i, j) та (j, i) рівні).

Задача комівояжера відноситься до NP-повних задач і для неї не існує оптимального алгоритму, який би працював за поліноміальний час. Тому для її розв’язання часто використовуються евристичні алгоритми, до яких відносяться і жадібні алгоритми.

Вам необхідно запропонувати ідею жадібного алгоритму та перевірити його роботу на декількох екземплярах задач.

Код програми:

import math as m

# длинна между точками

def func(a, b):

return m.sqrt((a[0] - b[0]) \*\* 2 + (a[1] - b[1]) \*\* 2)

def greedy(coordinates):

# массив точек

way = [coordinates[0]]

# длинна пути

size = 0

first = coordinates.pop(0)

# проход по всем точкам

while len(coordinates) != 0:

# временная переменная для сохранения минимальной длинны

min\_len = func(way[-1], coordinates[0])

j = 0

# нахождение минимальной длинны

for i in range(len(coordinates)):

# создать переменную, чтобы временно хранить длину пути между двумя вершинами

tmp = func(way[-1],coordinates[i])

if tmp < min\_len:

min\_len = tmp

j = i

# увеличение пройденного пути

size += min\_len

# обмен текущего и первого элементов

coordinates[0], coordinates[j] = coordinates[j], coordinates[0]

way.append(coordinates.pop(0))

# увеличение пройденного пути между первым и вторым

size += func(way[-1], first)

# добавление первой вершины

way.append(first)

return [i[2] for i in way], size

def main():

# чтение из файла

file = open("Inputs\input\_08.txt", 'r')

file.readline()

points = [[int(j) for j in i.split()] for i in file.readlines()]

for i in range(len(points)):

points[i].append(i)

file.close()

# запись в файл

file = open("Outputs\khmelinin\_output\_08.txt", 'w')

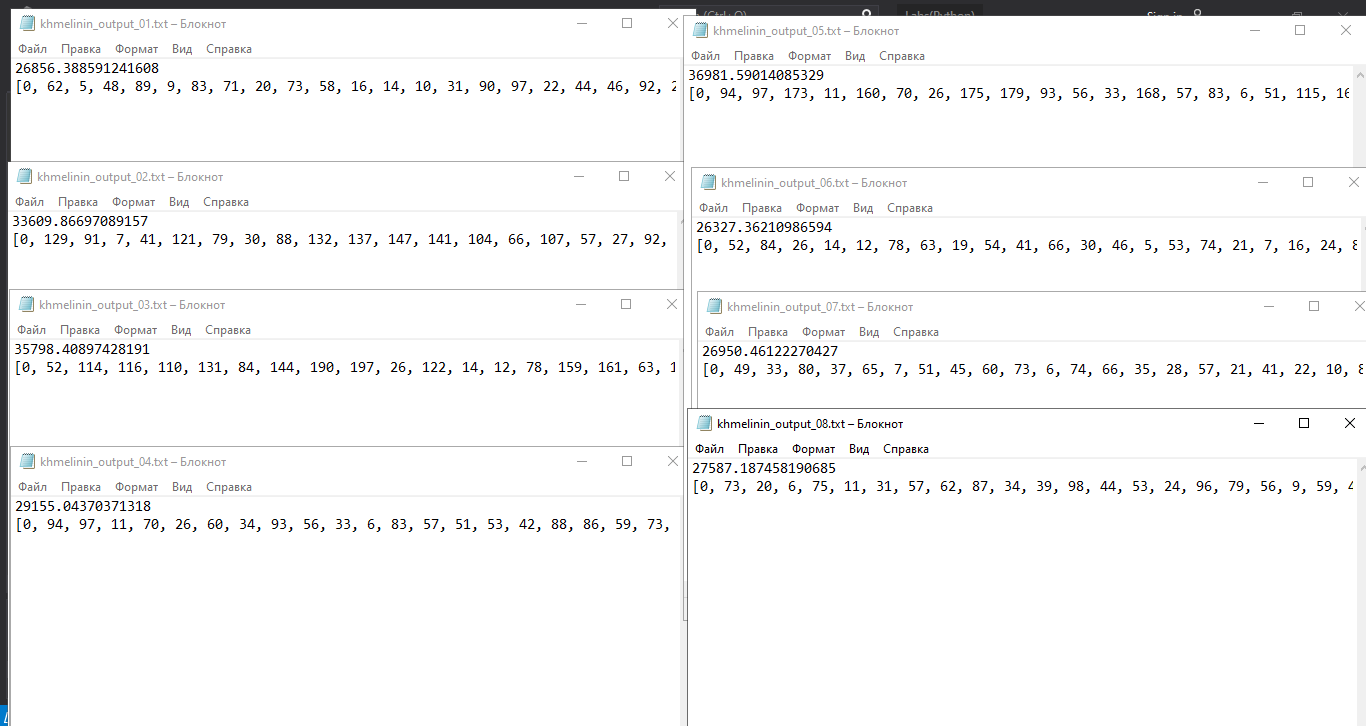
way, size = greedy(points)

file.write(str(size) + '\n')

file.writelines(str(way) + '\n')

if \_\_name\_\_ == '\_\_main\_\_':

main()



|  |  |  |  |
| --- | --- | --- | --- |
| Вхідний файл | Вартість знайденого шляху | Вартість еталонного шляху | Різниця у % |
| input\_01.txt | 26856.38 | 21282 | 20.8 |
| input\_02.txt | 33609.86 | 26524 | 21 |
| input\_03.txt | 35798.40 | 29368 | 17.9 |
| input\_04.txt | 29155.04 | 22141 | 24 |
| input\_05.txt | 36981.59 | 29437 | 20.4 |
| input\_06.txt | 26327.36 | 20749 | 21.2 |
| input\_07.txt | 26950.46 | 21294 | 20.1 |
| input\_08.txt | 27587.18 | 22068 | 20 |

Висновок:

Я створив жадібний алгоритм для розв’язання задачі Комівояжера, який працює на основі вибору найкоротшого шляху від поточного міста. На невеликих наборах міст, вага знайденого шляху майже не відрізняється, але на більших наборах міст, алгоритм допускає неточності і знаходить шлях, в середньому, на ~21 відсотк більше за еталонний.