|  |
| --- |
| МИНОБРНАУКИ РОССИИ  Федеральное государственное  бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |

Институт информационных технологий

Кафедра корпоративных информационных систем

|  |  |  |
| --- | --- | --- |
| **ОТЧЕТ**  **по лабораторной работе №1** | | |
| **по дисциплине** | | |
| **«Структуры и алгоритмы обработки данных»**  **Тема лабораторной работы: «**Графовые структуры данных**»** | | |
| Студент группы | ИКБО-07-18 | Зейналов М.Г. |
| Принял | ассистент кафедры КИС | Габриелян Г.А. |
|  |  |  |
| Выполнено | «\_\_\_» \_\_\_\_\_\_ 202\_\_ г. | *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* |
|  |  | *(подпись студента)* |
| Зачтено | «\_\_\_» \_\_\_\_\_\_ 201\_\_ г. | *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* |
|  |  | *(подпись преподавателя)* |

Москва 2020

1. **Задача №1**
   1. **Постановка задачи**

Необходимо добраться на самолете из города А в город В при условии, что между ними нет прямого авиационного сообщения, затратив при этом минимальные средства. Заданы возможные промежуточные аэропорты. Для каждой пары аэропортов известно, существует ли между ними прямой маршрут, и если да, то известна минимальная стоимость перелета по этому маршруту.

**Описание используемых структур данных**

Граф — абстрактный математический объект, представляющий собой множество вершин графа и набор рёбер, то есть соединений между парами вершин. Например, за множество вершин можно взять множество аэропортов, обслуживаемых некоторой авиакомпанией, а за множество рёбер взять регулярные рейсы этой авиакомпании между городами.

* 1. **Пользовательский интерфейс**

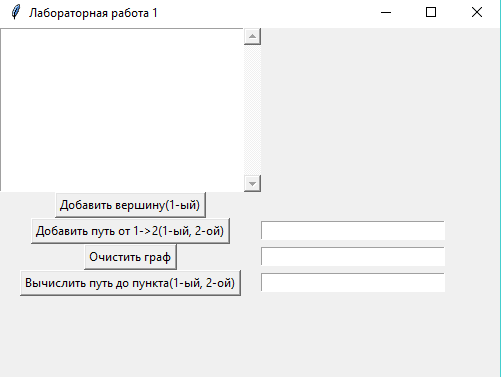


Рисунок 1.1 – «Пользовательский интерфейс»

* 1. **Описание алгоритма**

Создается класс Graph, который имеет 3 поля: все вершины, маршруты и цены. В этом классе есть также 2 метода: добавить вершину и добавить путь.

class Graph:

def \_\_init\_\_(self):

self.nodes = set()

self.edges = {}

self.distances = {}

Метод add\_node() добавляет вершину в множество.

def add\_node(self, value: str) -> None:

self.nodes.add(value)

Метод add\_edge() добавляет путь в словарь.

def add\_edge(self, from\_node, to\_node, cost):

self.\_add\_edge(from\_node, to\_node, cost)

self.\_add\_edge(to\_node, from\_node, cost)

def \_add\_edge(self, from\_node, to\_node, cost):

self.edges.setdefault(from\_node, [])

self.edges[from\_node].append(to\_node)

self.distances[(from\_node, to\_node)] = cost

Для нахождения минимальной стоимости используется алгоритм Дейкстры. Мы последовательно обходим вершины, проверяя все пути из этой вершины. Для каждого пути считаем стоимость, и если она меньше (или вообще единственный вариант), то мы записываем новое значение и помечаем обход, удаляя из множества вершин. Обход заканчивается, когда в множестве не остается вершин.

Восстановление пути находится обратным обходом уже готового словаря с минимальными ценами из вершин. Мы обходим словарь до того момента, пока не встретим начальную вершину, попутно складывая цену до этого города. Весь код реализации алгоритма будет на Github и ниже в приложении.

* 1. **Тестирование**

При тестировании мы заполняем граф городами и ценами

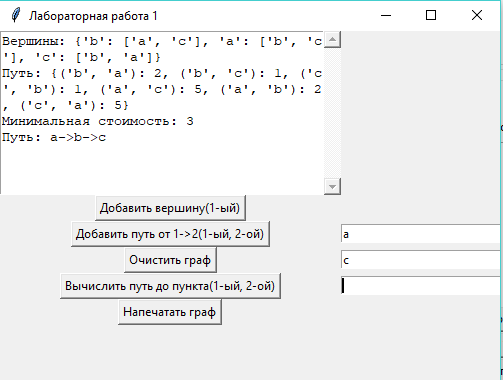


Рисунок 1.2 – «Тестирование задания»

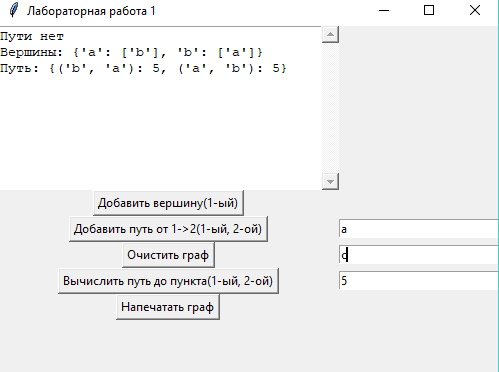


Рисунок 1.3 – «Тестирование задания»

* 1. **Листинг программы**

from tkinter import \*

from tkinter import scrolledtext

class Graph:

def \_\_init\_\_(self):

self.nodes = set()

self.edges = {}

self.distances = {}

def add\_node(self, value: str) -> None:

self.nodes.add(value)

def add\_edge(self, from\_node, to\_node, cost):

self.\_add\_edge(from\_node, to\_node, cost)

self.\_add\_edge(to\_node, from\_node, cost)

def \_add\_edge(self, from\_node, to\_node, cost):

self.edges.setdefault(from\_node, [])

self.edges[from\_node].append(to\_node)

self.distances[(from\_node, to\_node)] = cost

def print\_graph(self) -> list:

return [str(self.edges), str(self.distances)]

def find\_min\_road(graph: Graph, init: str) -> (dict, dict):

costs = {init: 0}

path = {}

nodes = set(graph.nodes)

while nodes:

min\_node = None

for node in nodes:

if node in costs:

if min\_node is None:

min\_node = node

elif costs[node] < costs[min\_node]:

min\_node = node

if min\_node is None:

break

nodes.remove(min\_node)

cur\_wt = costs[min\_node]

for edge in graph.edges[min\_node]:

wt = cur\_wt + graph.distances[(min\_node, edge)]

if edge not in costs or wt < costs[edge]:

costs[edge] = wt

path[edge] = min\_node

return costs, path

def get\_road(path: dict, from\_node: str, to\_node: str) -> str:

result = to\_node + ' '

cur\_node = to\_node

while path[cur\_node] != from\_node:

cur\_node = path[cur\_node]

result += cur\_node + ' '

result += from\_node

result = result[::-1].replace(' ', '->')

return result

graph = Graph()

# GUI

def btn\_print\_graph() -> None:

global graph

scr.insert(INSERT, "Вершины: " + str(graph.print\_graph()[0])+"\n")

scr.insert(INSERT, "Путь: " + str(graph.print\_graph()[1])+"\n")

def btn\_add\_node() -> None:

graph.add\_node(txt1.get())

def btn\_add\_edge() -> None:

graph.add\_edge(txt1.get(), txt2.get(), int(txt3.get()))

def btn\_clear\_graph() -> None:

global graph

graph = Graph()

def btn\_road() -> None:

from\_node = txt1.get()

to\_node = txt2.get()

vis, path = find\_min\_road(graph, from\_node)

if to\_node in vis.keys() and from\_node in vis.keys():

scr.insert(INSERT, "Минимальная стоимость: " + str(vis[to\_node]) + '\n')

scr.insert(INSERT, "Путь: " + get\_road(path, from\_node, to\_node) + '\n')

else:

scr.insert(INSERT, "Пути нет\n")

window = Tk()

window.title("Лабораторная работа 1")

window.geometry('500x350')

txt1 = Entry(window, width=30)

txt1.grid(column=2, row=2)

txt2 = Entry(window, width=30)

txt2.grid(column=2, row=3)

txt3 = Entry(window, width=30)

txt3.grid(column=2, row=4)

scr = scrolledtext.ScrolledText(window, width=40, height=10)

scr.grid(column=1, row=0)

btn\_append\_node = Button(window, text="Добавить вершину(1-ый)", command=btn\_add\_node)

btn\_append\_node.grid(column=1, row=1)

btn\_clear\_edge = Button(window, text="Добавить путь от 1->2(1-ый, 2-ой)", command=btn\_add\_edge)

btn\_clear\_edge.grid(column=1, row=2)

btn\_clear = Button(window, text="Очистить граф", command=btn\_clear\_graph)

btn\_clear.grid(column=1, row=3)

btn\_get\_road = Button(window, text="Вычислить путь до пункта(1-ый, 2-ой)", command=btn\_road)

btn\_get\_road.grid(column=1, row=4)

btn\_print = Button(window, text="Напечатать граф", command=btn\_print\_graph)

btn\_print.grid(column=1, row=5)

window.mainloop()