

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Факультет Информатика и вычислительная техника

Кафедра Кибербезопасность информационных систем

**ЛАБОРАТОРНАЯ РАБОТА № 1**

на тему «Алгоритмы нахождения кратчайших путей в графе»

Выполнил обучающийся гр. ВКБ33

Минаков Даниил Сергеевич

Проверил

Доцент, Савельев Василий Александрович

Ростов-на-Дону

2021

Реализовать нахождение кратчайшего пути в графе между двумя заданными вершинами. Внутренняя структура графа не задается, но рекомендуется ориентироваться на решения эффективные и по памяти, и по времени реализации алгоритма. Также в задание входит создание вспомогательных библиотек, обеспечивающих ввод/вывод графа.

***Задание 1***

Создать библиотеку, обеспечивающую ввод и вывод графа в текстовой форме. Граф в текстовом файле задается списком ребер — одно ребро на строке, ребро задается номером начальной вершины, номером конечной вершины и весом ребра. Все три значения целые, разделяются пробельными символами.

Пример:

0 1 7

0 2 9

1 2 10

1 3 15

2 3 11

Написать тесты и отладить эту библиотеку.

Измерить скорость ввода-вывода

***Задание 2***

Создать библиотеку, обеспечивающую ввод и вывод графа в бинарной форме. Использовать для сохранения значений 32-разрабные целые поля в сетевом порядке байтов. Написать тесты и отладить библиотеку. Измерить скорость ввода/вывода.

Код к заданию 1 и 2 (библиотека для бинарного/небинарного ввода-вывода графов):

import os.path as osp  
  
  
def load\_graph(file, is\_bin=False):  
 if is\_bin:  
 graph = open(file, 'rb')  
 fsize = osp.getsize(file)  
 reblist = []  
 for e in range(int(fsize / 12)):  
 temp = []  
 for k in range(3):  
 temp.append(int.from\_bytes(graph.read(4), 'little'))  
 reblist.append(temp)  
 else:  
 graph = open(file, 'r')  
 reblist = [[int(k) for k in e.split()] for e in graph.readlines()]  
 graph.close()  
 return reblist  
  
  
def save\_graph(graph, file\_name, is\_bin=False):  
 if is\_bin:  
 newf = open(file\_name, 'wb')  
 for e in graph:  
 newf.write(e[0].to\_bytes(4, byteorder='little'))  
 newf.write(e[1].to\_bytes(4, byteorder='little'))  
 newf.write(e[2].to\_bytes(4, byteorder='little'))  
 else:  
 newf = open(file\_name, 'w')  
 for e in graph:  
 newf.write('{} {} {}\n'.format(e[0], e[1], e[2]))  
 newf.close()

***Задание 3.***

Написать программу, способную прочесть граф из указанного файла (как в текстовой, так и бинарной форме, выбирается ключом командной строки), получающую также в командной строке номера начальной и конечной вершин. Выводится кратчайший путь между этими вершинами. Если пути нет, выводится одна строка с номерами начальной и конечной вершин и длиной пути -1. Если конечная вершина не указана, выводится длина кратчайшего пути из начальной вершины во все остальные вершины.

Написать тесты и отладить программу.

Программа:

from sys import argv  
import graphlib as gr  
  
if len(argv) > 1:  
 if argv[1] != '/?':  
 filename = argv[1]  
 else:  
 print(  
 'Поиск в графе кратчайшего пути между заданными вершинами.\nlaba1.py [имя\_файла\_графа] (тип\_файла /b-бинарный файл или /n -небинарный) (номер\_начальной\_вершины) (номер\_конечной\_вершины)')  
 exit()  
 if len(argv) > 4:  
 if argv[2] == '/b':  
 is\_bin = True  
 else:  
 is\_bin = False  
 start = int(argv[3])  
 fin = int(argv[4])  
 else:  
 is\_bin = False  
 start = 0  
 fin = 3  
else:  
 is\_bin = False  
 start = 0  
 fin = 3  
 filename = 'input.txt'  
  
  
def graph\_short\_path\_find(cpoint, tpoint, rebrs, length=0, dellst=None, lens=None, path=''):  
 if lens is None:  
 lens = []  
 if dellst is None:  
 dellst = []  
 if length == 0:  
 lens = []  
 path += str(cpoint) + '-'  
 if cpoint == tpoint:  
 lens.append(length)  
 return None  
 for num in dellst:  
 rebrs.pop(num)  
 dellst = []  
 for c, d in enumerate(rebrs):  
 if (cpoint in rebrs[c]):  
 dellst.append(c)  
 dellst.reverse()  
 for num in dellst:  
 if rebrs[num][0] != cpoint:  
 nextp = rebrs[num][0]  
 else:  
 nextp = rebrs[num][1]  
 graph\_short\_path\_find(nextp, tpoint, rebrs[:], length + rebrs[num][2], dellst, lens, path)  
 if not lens:  
 otv = '{} {} -1'.format(cpoint, tpoint)  
 else:  
 otv = min(lens)  
 return otv  
  
  
graph = gr.load\_graph(filename, is\_bin)  
length = graph\_short\_path\_find(start, fin, graph)  
print(length)

Тест:

class TestShortPath(unittest.TestCase):  
 def test\_1(self):  
 self.assertEqual(graph\_short\_path\_find(0, 3, graph), 20)  
  
 def test\_2(self):  
 self.assertEqual(graph\_short\_path\_find(2, 3, graph), 11)  
  
 def test\_3(self):  
 self.assertEqual(graph\_short\_path\_find(3, 0, graph), 20)  
  
 def test\_4(self):  
 self.assertEqual(graph\_short\_path\_find(3, 2, graph), 11)  
  
 def test\_5(self):  
 self.assertEqual(graph\_short\_path\_find(2, 4, graph), '2 4 -1')  
  
 def test\_6(self):  
 self.assertEqual(graph\_short\_path\_find(4, 2, graph), '4 2 -1')  
  
 def test\_7(self):  
 self.assertEqual(graph\_short\_path\_find(5, 6, graph), '5 6 -1')  
  
 def test\_8(self):  
 self.assertEqual(graph\_short\_path\_find(6, 5, graph), '6 5 -1')  
  
 def test\_9(self):  
 self.assertEqual(graph\_short\_path\_find(1, 3, graph), 15)  
  
 def test\_10(self):  
 self.assertEqual(graph\_short\_path\_find(3, 1, graph), 15)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 graph = gr.load\_graph(filename, is\_bin)  
 unittest.main()

Итог прохождения теста:

