

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 № 2**

на тему «Алгоритмы построения остова минимальной стоимости»

Выполнил студент:

уч. группа ВКБ32

Ф.И.О.: Зуев

Вячеслав Юрьевич

№ зачетки: 1954182

Проверил: Савельев В.А.

Ростов-на-Дону

2022

В данном графе построить остов минимальной стоимости.

Задание 1 Реализовать алгоритм построения остовного дерева минимальной стоимости. Создать тесты и отладить.

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')

Задание 2 Использовать алгоритм построения остова минимальной стоимости для вывода графа в виде XML-документа. Использовать атрибуты спецификации XLINK для представления ребер графа не вошедших в остов. Создать тесты и отладить. Оформить в библиотеку. Сравнить по используемой памяти и времени работы с реализованными в первой лабораторной работе способами вывода графа.

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 Реализовать ввод графа из XML-документа. Создать тесты и отладить. Оформить в библиотеку. Сравнить по используемой памяти и времени работы с реализованными в первой лабораторной работе способами ввода графа.

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=[],lens=[],path=''):

if length==0:

lens=[]

path+=str(cpoint)+'-'

if cpoint==tpoint:

lens.append(length)

#print('Найден путь ',path[:-1],':',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)

lens=''

return otv

graph=gr.load\_graph(filename,is\_bin)

length=graph\_short\_path\_find(start,fin,graph)

print(length)

Задание 4 Реализовать алгоритм Прима. Создать тесты и отладить.

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()

Задание 5 Реализовать алгоритм Крускала. Создать тесты и отладить.

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:

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()