

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 «Построение остова минимальной стоимости»

Выполнил обучающийся гр. ВКБ33

Минаков Даниил Сергеевич

Проверил

Доцент, Савельев Василий Александрович

Ростов-на-Дону

2021

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')  
 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 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.

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 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.

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 4

Реализовать алгоритм Прима. Создать тесты и отладить.

from sys import argv  
import graphlib as gr  
from graph\_short\_path\_find import graph\_short\_path\_find  
  
if len(argv)>1:  
 if argv[1]!='/?':  
 filename=argv[1]  
 else:  
 print('Построение остова')  
 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 prima(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=prima(filename,is\_bin)  
length=graph\_short\_path\_find(start,fin,graph)  
print(length)

Задание 5

Реализовать алгоритм Крускала. Создать тесты и отладить.

parent = dict()  
rank = dict()  
  
  
def make\_set(vertice):  
 parent[vertice] = vertice  
 rank[vertice] = 0  
  
  
def find(vertice):  
 if parent[vertice] != vertice:  
 parent[vertice] = find(parent[vertice])  
 return parent[vertice]  
  
  
def union(vertice1, vertice2):  
 root1 = find(vertice1)  
 root2 = find(vertice2)  
 if root1 != root2:  
 if rank[root1] > rank[root2]:  
 parent[root2] = root1  
 else:  
 parent[root1] = root2  
 if rank[root1] == rank[root2]: rank[root2] += 1  
  
  
def kruskal(graph):  
 for vertice in graph['vertices']:  
 make\_set(vertice)  
 minimum\_spanning\_tree = set()  
 edges = list(graph['edges'])  
 edges.sort()  
 # print edges  
 for edge in edges:  
 weight, vertice1, vertice2 = edge  
 if find(vertice1) != find(vertice2):  
 union(vertice1, vertice2)  
 minimum\_spanning\_tree.add(edge)  
  
 return sorted(minimum\_spanning\_tree)  
graph = {  
'vertices': ['A', 'B', 'C', 'D', 'E', 'F', 'G'],  
'edges': {(7, 'A', 'B'), (5, 'A', 'D'), (7, 'B', 'A'), (8, 'B', 'C'), (9, 'B', 'D'), (7, 'B', 'E'), (8, 'C', 'B'),  
 (5, 'C', 'E'), (5, 'D', 'A'), (9, 'D', 'B'), (7, 'D', 'E'), (6, 'D', 'F'), (7, 'E', 'B'), (5, 'E', 'C'),  
 (15, 'E', 'D'), (8, 'E', 'F'), (9, 'E', 'G'), (6, 'F', 'D'), (8, 'F', 'E'), (11, 'F', 'G'), (9, 'G', 'E'),  
 (11, 'G', 'F')}  
}  
  
print(kruskal(graph))