

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 обучающийся гр. ВКБ32

Шелехов Александр Константинович

Проверил

Доцент, Савельев Василий Александрович

Ростов-на-Дону

2022

Задание 1

R = [(13, 1, 2), (18, 1, 3), (17, 1, 4), (14, 1, 5), (22, 1, 6),

     (26, 2, 3), (22, 2, 5), (3, 3, 4), (19, 4, 6)]

Rs = sorted(R, key=lambda x: x[0])

U = set()

D = {}

T = []

for r in Rs:

    if r[1] not in U or r[2] not in U:

        if r[1] not in U and r[2] not in U:

            D[r[1]] = [r[1], r[2]]

            D[r[2]] = D[r[1]]

        else:

            if not D.get(r[1]):

                D[r[2]].append(r[1])

                D[r[1]] = D[r[2]]

            else:

                D[r[1]].append(r[2])

                D[r[2]] = D[r[1]]

        T.append(r)

        U.add(r[1])

        U.add(r[2])

for r in Rs:

    if r[2] not in D[r[1]]:

        T.append(r)

        gr1 = D[r[1]]

        D[r[1]] += D[r[2]]

        D[r[2]] += gr1

print(T)

Задание 2

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

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)

        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.

import random

from string import ascii\_uppercase

def prima(W, city\_labels = None):

    \_ = float('inf')

    cities\_count = len(W)

    for weights in W:

        assert len(weights) == cities\_count

    if not city\_labels:

        city\_labels = [ascii\_uppercase[x] for x in range(cities\_count)]

    assert cities\_count <= len(city\_labels)

    free\_vertexes = list(range(0, len(city\_labels)))

    starting\_vertex = random.choice(free\_vertexes)

    tied = [starting\_vertex]

    free\_vertexes.remove(starting\_vertex)

    print('Начало с %s' % city\_labels[starting\_vertex])

    road\_length = 0

    while free\_vertexes:

        min\_link = None

        overall\_min\_path = \_

        for current\_vertex in tied:

            weights = W[current\_vertex]

            min\_path = \_

            free\_vertex\_min = current\_vertex

            for vertex in range(cities\_count):

                vertex\_path = weights[vertex]

                if vertex\_path == 0:

                    continue

                if vertex in free\_vertexes and vertex\_path < min\_path:

                    free\_vertex\_min = vertex

                    min\_path = vertex\_path

            if free\_vertex\_min != current\_vertex:

                if overall\_min\_path > min\_path:

                    min\_link = (current\_vertex, free\_vertex\_min)

                    overall\_min\_path = min\_path

        try:

            path\_length = W[min\_link[0]][min\_link[1]]

        except TypeError:

            print('Unable to find path')

            return

        print('Просматривается дорога от %s до %s [%s]' % (city\_labels[min\_link[0]], city\_labels[min\_link[1]], path\_length))

        road\_length += path\_length

        free\_vertexes.remove(min\_link[1])

        tied.append(min\_link[1])

    print('Длина дороги: %s' % road\_length)

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

    \_ = float('inf')

    W = [

      [0, 3, \_, \_, \_, 2, \_],

        [3, 0, 3, \_, \_, \_, 6],

        [\_, 3, 0, 3, 2, \_, 1],

        [\_, \_, 3, 0, 5, \_, \_],

        [\_, \_, 2, 5, 0, 4, 3],

        [2, \_, \_, \_, 4, 0, 3],

        [\_, 6, 1, \_, 3, 3, 0],

    ]

    prima(W)

Задание 5 Реализовать алгоритм Крускала. Создать тесты и отладить.

class Graph:

    def \_\_init\_\_(self, vertex):

        self.V = vertex

        self.graph = []

    def add\_edge(self, u, v, w):

        self.graph.append([u, v, w])

    def search(self, parent, i):

        if parent[i] == i:

            return i

        return self.search(parent, parent[i])

    def apply\_union(self, parent, rank, x, y):

        xroot = self.search(parent, x)

        yroot = self.search(parent, y)

        if rank[xroot] < rank[yroot]:

            parent[xroot] = yroot

        elif rank[xroot] > rank[yroot]:

            parent[yroot] = xroot

        else:

            parent[yroot] = xroot

            rank[xroot] += 1

    def kruskal(self):

        result = []

        i, e = 0, 0

        self.graph = sorted(self.graph, key=lambda item: item[2])

        parent = []

        rank = []

        for node in range(self.V):

            parent.append(node)

            rank.append(0)

        while e < self.V - 1:

            u, v, w = self.graph[i]

            i = i + 1

            x = self.search(parent, u)

            y = self.search(parent, v)

            if x != y:

                e = e + 1

                result.append([u, v, w])

                self.apply\_union(parent, rank, x, y)

        for u, v, weight in result:

            print("Графы:",u, v,end =" ")

            print("-",weight)

g = Graph(5)

g.add\_edge(0, 1, 8)

g.add\_edge(0, 2, 5)

g.add\_edge(1, 2, 9)

g.add\_edge(1, 3, 11)

g.add\_edge(2, 3, 15)

g.add\_edge(2, 4, 10)

g.add\_edge(3, 4, 7)

g.kruskal()