

# Лабораторная работа 1 по Теории Конечных графов

Студент

Киреев Данил Алексеевич

Группа

НФИБД-01-17

Преподаватели:

к.ф.-м.н., доцент Зарипова Э.Р.,

преп. Гольская А.А.

Тема лабораторной работы 1:

Алгоритм Краскала

Количество баллов:

\_\_\_\_\_ баллов из 10 б.

## **1. Теоретические сведения по алгоритму Краскала**

Для лабораторной работы использовались источники [1, 2].

## **2. Реализация алгоритма Краскала на примере**

Для реализации алгоритма Краскала построения максимального покрывающего дерева на примере был выбран граф на рис. 1.

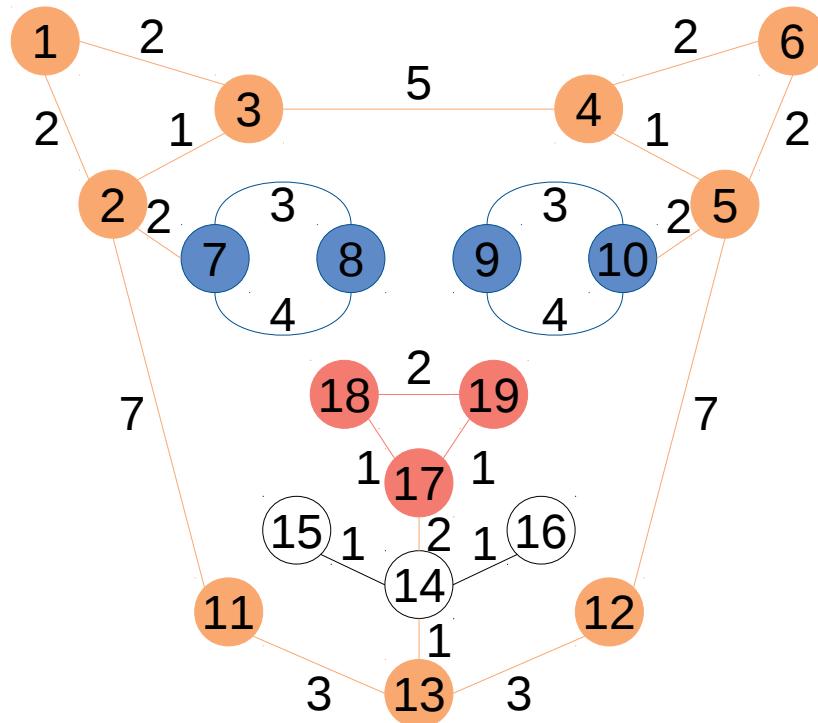


Рис.1. Исходный граф

Для данного графа применён представленный ниже исходный код построения максимального покрывающего дерева алгоритмом Краскала, выполненный на языке программирования Python3 (<https://www.python.org/download/releases/3.0/>).

Примечания:

Эта программа имеет 2 режима ввода информации о графе: заранее и вручную, выбор предлагается при старте программы. Если выбрать з(заранее), программа прочитает заранее заданные данные о графе на рисунке 1 из этой работы. Если выбрать в(вручную), программа попросит ввести данные о графе. Так же, есть опция показать полученный граф, отредактированный(без петель и параллельных рёбер) и полученное покрывающее дерево. Чтобы выбрать её в конце работы программы нужно выбрать д(да), чтобы закончить работу программу - н(нет).

Исходный код:

```
def user_input():
    """
    функция ввода графа пользователем,
    граф должен быть связанным
    function for inputting graph information by user,
    graph must be connected
    :return: list of dicts{cords=tuple(c1, c2), weight=int(w)}
    """

    res_e = list()
    print("введите запрошенные данные СВЯЗАННОГО графа")
    ui_inp = int(input("введите количество рёбер: "))
    while ui_inp <= 0: # check if input is correct
        ui_inp = int(input("введите положительное число"))
    for i in range(ui_inp): # input set amount of edges
        c = list(input("введите крайние вершины ребра " +
                       "{:^3} через запятую: ".format(i + 1)))
        if ',' in c:
            c.remove(',')
        if ' ' in c:
            c.remove(' ')
        c.sort()
        c = tuple(c)
        w = int(input("введите вес ребра {:^3}: ".format(i + 1)))
        res_e.append(dict(cords=c, weight=w))
    return res_e

def pre_input():
```

```

"""
функция, возвращающая заранее заданный граф из текстовой
части ЛР
function, returning previously mentioned graph in LW
:return: list of dicts{cords=tuple(c1, c2), weight=int(w)}
"""

res_e = list() # setup for inputting predetermined graph
res_e.append(dict(cords=(1, 2), weight=2))
res_e.append(dict(cords=(1, 3), weight=2))
res_e.append(dict(cords=(2, 3), weight=1))
res_e.append(dict(cords=(2, 7), weight=2))
res_e.append(dict(cords=(2, 11), weight=7))
res_e.append(dict(cords=(3, 4), weight=5))
res_e.append(dict(cords=(4, 5), weight=1))
res_e.append(dict(cords=(4, 6), weight=2))
res_e.append(dict(cords=(5, 6), weight=2))
res_e.append(dict(cords=(5, 10), weight=2))
res_e.append(dict(cords=(5, 12), weight=7))
res_e.append(dict(cords=(7, 8), weight=3))
res_e.append(dict(cords=(7, 8), weight=4))
res_e.append(dict(cords=(9, 10), weight=3))
res_e.append(dict(cords=(9, 10), weight=4))
res_e.append(dict(cords=(11, 13), weight=3))
res_e.append(dict(cords=(12, 13), weight=3))
res_e.append(dict(cords=(13, 14), weight=1))
res_e.append(dict(cords=(14, 15), weight=1))
res_e.append(dict(cords=(14, 16), weight=1))
res_e.append(dict(cords=(14, 17), weight=2))
res_e.append(dict(cords=(17, 18), weight=1))
res_e.append(dict(cords=(17, 19), weight=1))
res_e.append(dict(cords=(18, 19), weight=2))
return res_e
def special_sort(list_of_points):
"""
сортировка точек графа:
    удаление петель
    удаление паралельных рёбер с меньшим весом
    (т.к. надо найти максимальное покрывающее дерево)
sorting graph:
    removing loops
    removing parallel edges with lower weight
    (bc we need to find max covering tree)
:param list_of_points: list of dicts{cords=tuple(c1, c2),
weight=int(w)}
:return: list of dicts{cords=tuple(c1, c2), weight=int(w)}

```

```

"""
p_lop = list()
for i, i_el in enumerate(list_of_points):
    n = -1
    m = -1
    f = False
    if i_el["cords"][0] != i_el["cords"][1]:
        # if edge isn't a loop
        for j, j_el in enumerate(p_lop):
            if i_el["cords"] == j_el["cords"]:
                f = True
                n = j
                m = i
        # try to find element with the same cords
        # and if successful raise flag and remember place
    if not f:
        p_lop.append(i_el)
    else:
        p_lop[n]["weight"] = max(p_lop[n]["weight"],
                                  list_of_points[m]
                                  ["weight"])
p_lop = sorted(p_lop, key=lambda k: k["weight"],
reverse=True)
# sort graph edges by weight
return p_lop
def alg_Kraskala(list_of_points):
"""
    имплементация алгоритма Краскала
    implementation of Kruskal's algorithm
    :param list_of_points: list of dicts{cords=tuple(c1, c2),
    weight=int(w)}
    :return: list of dicts{cords=tuple(c1, c2), weight=int(w)}
    (описывающий покрывающее дерево)
"""

    res = list()
    buckets = list()
    for l_i, l_el in enumerate(list_of_points): # check through
graphs edges
        c0_b = False
        c1_b = False
        c0_i = -1
        c1_i = -1
        for b_i, b_el in enumerate(buckets): # check through
buckets(букеты)
            if l_el["cords"][0] in b_el:

```

```

        # memorize edges first end as in bucket(букует)
        c0_b = True
        c0_i = b_i
    if l_el["cords"][1] in b_el:
        # memorize edges second end as in bucket(букует)
        c1_b = True
        c1_i = b_i
    if (not c0_b) and (not c1_b): # if none are in the
bucket
        buckets.append([l_el["cords"][0], l_el["cords"][1]])
        res.append(l_el)
    elif (not c0_b) and c1_b: # if only one is in the
bucket
        buckets[c1_i].append(l_el["cords"][0])
        res.append(l_el)
    elif c0_b and (not c1_b): # if only one is in the
bucket
        buckets[c0_i].append(l_el["cords"][1])
        res.append(l_el)
    elif c0_b and c1_b and (c0_i != c1_i):
        # if both are in different buckets
        buckets[c0_i] += buckets[c1_i]
        buckets.pop(c1_i)
        buckets[c0_i].sort()
        res.append(l_el)
    return res
def graph_print(list_of_points):
"""
вывод информации о графе
print information about graph
:param list_of_points: list of dicts{cords=tuple(c1, c2),
weight=int(w)}
:return: nothing / ничего
"""

    sum = int()
    for i in list_of_points:
        print("ребро:", i["cords"], "\t, вес: ", i["weight"])
        sum += i["weight"]
    print("вес данного графа: ", sum)
def graph_draw(list_of_points, graph_num):
"""
создание рисунка графа используя библиотеку graphviz
draw graph using graphviz library
:param graph_num: int()

```

```

:param list_of_points: list of dicts{cords=tuple(c1, c2),
weight=int(w)}
:return: nothing / ничего
"""

import graphviz as gv
g = gv.Graph()
for i in list_of_points:
    # add all of the edges to special structure
    g.edge(str(i["cords"])[0]), str(i["cords"])[1]),
        weight=str(i["weight"]),
        label=str(i["weight"]))
g.render("Graph_{}".format(graph_num), view=True)
return

def main():
    E = list()
    print("выберите способ задания графа одной из предложенных
булев: " +
          "\n\t-заранее: з\n\t-вручную: в")
    # input for way to input graph
    inp = input("введите выбранный способ: ")
    # python's case equivalent_
    inp_options = {
        'з': pre_input,
        'в': user_input
    }
    try:
        E = inp_options[inp]()
    except KeyError as e:
        print("Ошибка ввода")
        raise ValueError('Undefined unit: {}'.format(e.args[0]))
    # _python's case equivalent
    print("введённый график: ") # print graph
    graph_print(E)
    # remove unnecessary data to simplify and fasten algorithm
    p_E = special_sort(E)
    print("подготовленный график: ") # print graph
    graph_print(p_E)
    # use implementation of algorithm to find max covering graph
    E_res = alg_Kraskala(p_E)
    print("полученный график: ") # print graph
    graph_print(E_res)
    # input for graph visualisation
    inp = input("показать введённый и полученный способ?(д/н) ")
    if inp == "д":
        graph_draw(E, 1)

```

```

graph_draw(p_E, 2)
graph_draw(E_res, 3)
elif inp == "H":
    pass
else:
    print("Ошибка ввода")
    raise ValueError('Undefined unit: {}'.format(inp))
if __name__ == "__main__":
    main()

```

(так же исходный код программы есть на github:  
[https://github.com/randomunrandom/LW\\_TKG\\_3\\_1](https://github.com/randomunrandom/LW_TKG_3_1), различия только в форматировании)

В результате применения алгоритма получен следующий результат:

максимальное покрывающее дерево  $T_{\max} = \{(2, 11), (5, 12), (3, 4), (7, 8), (9, 10), (11, 13), (12, 13), (1, 2), (1, 3), (2, 7), (4, 6), (5, 10), (14, 17), (18, 19), (13, 14), (14, 15), (14, 16), (17, 18)\}$

и подсчитан вес дерева  $W_{T_{\max}} = 7+7+5+4+4+3+3+2+2+2+2+2+2+1+1+1 = 51$

и получено покрывающее дерево (рис. 2)

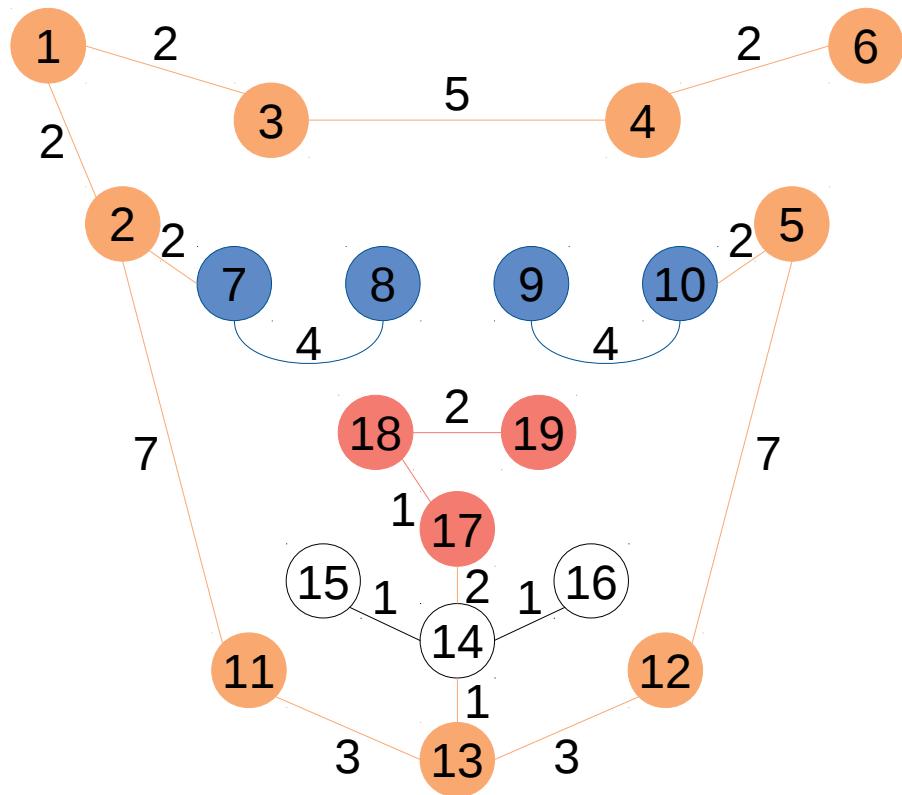


Рис. 2. Конечный граф.

### Источники

1. Лекции по теории графов, лектор Зарипова Э.Р.
2. Зарипова Э.Р., Кокотчикова М.Г., Лекции по дискретной математике: Теория графов. М., РУДН, 2013 – 162 с.: ил.