**Отчет по практике курса «Теория графов» за пятый семестр 2022-2023 учебного года**

**Работу выполнила Сухова Екатерина Владимировна 341 группа МОАИС**

Ссылка на проект на Github: <https://github.com/Ekats3529/Graph_git>

Оглавление

[Создание класса Граф 2](#_Toc121065072)

[Список смежности 18](#_Toc121065073)

[2. Список смежности Ia + 3. Список смежности Ia 18](#_Toc121065074)

[4. Список смежности Iб: несколько графов 19](#_Toc121065075)

[Обходы графа 22](#_Toc121065076)

[5. Обходы графа II (DFS) 22](#_Toc121065077)

[6. Обходы графа II (BFS) 25](#_Toc121065078)

[Построение минимального остовного дерева 28](#_Toc121065079)

[7. Каркас III (Краскал) 28](#_Toc121065080)

[Взвешенный граф 30](#_Toc121065081)

[8. Веса IV а 30](#_Toc121065082)

[9. Веса IV b 32](#_Toc121065083)

[10. Веса IV с 35](#_Toc121065084)

[Потоки 40](#_Toc121065085)

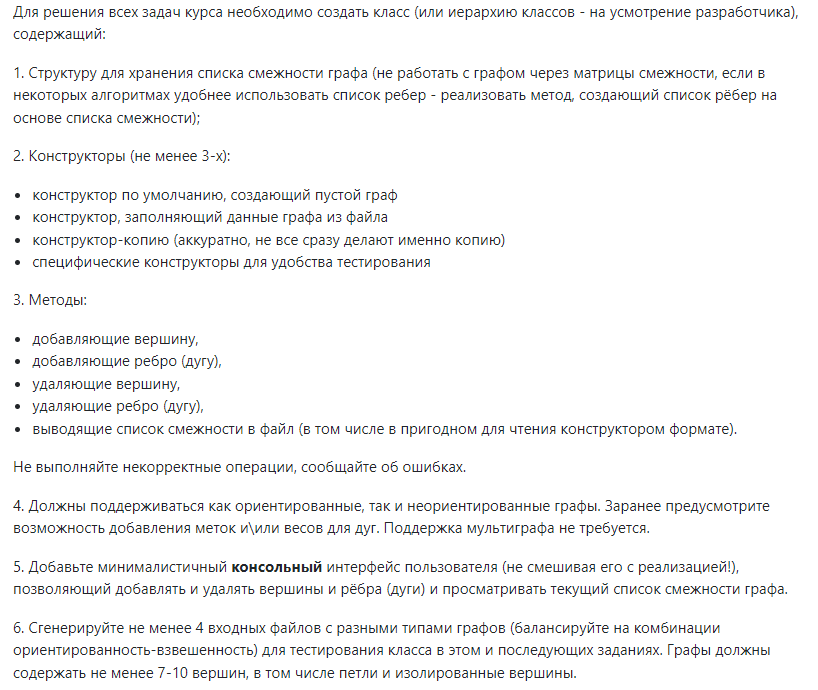
[11. Максимальный поток V 40](#_Toc121065086)

[Творческая задача 41](#_Toc121065087)

[Задание на визуализацию 41](#_Toc121065088)

Создание класса Граф

**1. Минимальные требования для класса Граф**

****

Класс Graph

from copy import deepcopy  
  
  
class Graph:  
 adj\_list = {} *# список смежности* type = "!directed" *# тип графа: ориентированный или нет* weighted = False *# тип графа: взвешенный или нет* nodes\_list = [] *# список вершин  
  
 # инициализация графа* def \_\_init\_\_(self, \*attributes):  
 self.adj\_list = {}  
 if len(attributes) > 0:  
 self.type = attributes[0]  
 self.weighted = True if attributes[1] == "weighted" else False  
  
 *# метод для создания списка ребер* def create\_edge\_list(self, dublicate):  
 edge\_list = []  
  
 if not dublicate:  
 edges = []  
 for node in self.adj\_list.keys():  
 for adj in self.adj\_list[node]:  
 if [node, adj[0]] not in edges:  
 edges.append([adj[0], node])  
 edge\_list.append([node, adj[0], adj[1]])  
 else:  
 for node in self.adj\_list.keys():  
 for adj in self.adj\_list[node]:  
 edge\_list.append([node, adj[0], adj[1]])  
  
 return edge\_list  
  
 *# метод для создания графа из файла* def create\_from\_file(self, filename):  
 adj\_list = {}  
 try:  
 fin = open(filename, encoding="utf8")  
 except FileNotFoundError:  
 print("ERROR: No such file or directory")  
 return False  
  
 self.type = fin.readline().split()[0]  
 self.weighted = True if fin.readline().split()[0] == "weighted" else False  
 n, m = map(int, fin.readline().split())  
 self.nodes\_list = fin.readline().split()  
  
 for node in self.nodes\_list:  
 adj\_list[node] = []  
  
 edges = fin.readlines()  
 for edge in edges:  
 v, u, c = edge.split()  
 try:  
 if v == u and self.type == "!directed":  
 print(f"ERROR: No loop in not directed graph")  
 return False  
 if u in self.nodes\_list:  
 if u not in adj\_list[v]:  
 adj\_list[v].append([u, c])  
 else:  
 print(f"ERROR: No multiple edges")  
 return False  
 else:  
 print(f"ERROR: No such vertex {u}")  
 return False  
 except KeyError:  
 print(f"ERROR: No such vertex {v}")  
 return False  
  
 if self.type == "!directed":  
 try:  
 if v in self.nodes\_list:  
 if u not in adj\_list[v]:  
 adj\_list[u].append([v, c])  
 else:  
 print(f"ERROR: No multiple edges")  
 return False  
 else:  
 print(f"ERROR: No such vertex {v}")  
 return False  
 except KeyError:  
 print(f"ERROR: No such vertex {u}")  
 return False  
  
 self.adj\_list = adj\_list  
 return True  
  
 *# метод для создания копии графа* def copy(self):  
 cp\_graph = Graph(self.type, "weighted" if self.weighted else "!weighted")  
 cp\_graph.adj\_list = deepcopy(self.adj\_list)  
 cp\_graph.nodes\_list = self.nodes\_list.copy()  
 return cp\_graph  
  
 *# метод для добавления вершины* def add\_node(self, node):  
 if node in self.nodes\_list:  
 print(f"ERROR: Unable to add vertex {node}. The same vertex already exist")  
 return False  
 self.adj\_list[node] = []  
 self.nodes\_list.append(node)  
 return True  
  
 *# метод для добавления ребра (дуги)* def add\_edge(self, edge):  
 if len(edge) < 2:  
 print(f"Unable to add the edge")  
 return False  
 v, u = edge[0], edge[1]  
 c = '1'  
 fl = False  
 if self.weighted:  
 try:  
 c = edge[2]  
 except (KeyError, IndexError):  
 print(f"Unable to add the edge. No weight entered")  
 return False  
 else:  
 if len(edge) == 3:  
 print("This graph is not weighted, so the weight of this edge not counted")  
 try:  
 if self.weighted:  
 if v == u and self.type == "!directed":  
 print(f"Unable to add the edge. No loop in not directed graph")  
 return False  
 nodes = [x[0] for x in self.adj\_list[v]]  
 if u in nodes:  
 print(f"Do you want to change the weight of edge {v} {u} "  
 f"from {self.adj\_list[v][nodes.index(u)][1]} to {c} "  
 f"Y/N")  
 ans = input()  
 if ans == "Y":  
 fl = True  
 self.adj\_list[v][nodes.index(u)][1] = c  
 else:  
 if u in self.nodes\_list and v in self.nodes\_list:  
 self.adj\_list[v].append([u, c])  
 else:  
 print(f"Unable to add the edge. No such vertex")  
 return False  
 else:  
 if v == u and self.type == "!directed":  
 print(f"Unable to add the edge. No loop in not directed graph")  
 return False  
 if [u, c] not in self.adj\_list[v]:  
 if u in self.nodes\_list and v in self.nodes\_list:  
 self.adj\_list[v].append([u, c])  
 else:  
 print(f"Unable to add the edge. No such vertex")  
 return False  
 else:  
 print(f"Unable to add the edge. Same edge already exists")  
 return False  
  
 except KeyError:  
 print(f"Unable to add the edge. No such vertex {v}")  
 return False  
  
 if self.type != "directed":  
 try:  
 if self.weighted:  
  
 nodes = [x[0] for x in self.adj\_list[u]]  
 if v in nodes:  
 if fl:  
 self.adj\_list[u][nodes.index(v)][1] = c  
 else:  
 if u in self.nodes\_list and v in self.nodes\_list:  
 self.adj\_list[u].append([v, c])  
 else:  
 print(f"Unable to add the edge. No such vertex")  
 return False  
 else:  
 if [v, c] not in self.adj\_list[u]:  
 if u in self.nodes\_list and v in self.nodes\_list:  
 self.adj\_list[u].append([v, c])  
 else:  
 print(f"Unable to add the edge. No such vertex")  
 return False  
 else:  
 print(f"Unable to add the edge. Same edge already exists")  
 return False  
 except KeyError:  
 print(f"Unable to add the edge. No such vertex {u}")  
 return False  
 return True  
  
 *# метод для удаления вершины* def delete\_node(self, node):  
 try:  
 for item in self.adj\_list.items():  
 nodes = [x[0] for x in item[1]]  
 if node in nodes:  
 del self.adj\_list[item[0]][nodes.index(node)]  
 del self.adj\_list[node]  
 del self.nodes\_list[self.nodes\_list.index(node)]  
 except KeyError:  
 print(f"ERROR: No such vertex {node}")  
 return False  
 return True  
  
 *# метод для удаления ребра (дуги)* def delete\_edge(self, edge):  
 v, u = edge[0], edge[1]  
  
 try:  
 nodes = [x[0] for x in self.adj\_list[v]]  
 del self.adj\_list[v][nodes.index(u)]  
 except (KeyError, ValueError):  
 print(f"ERROR: No such edge ({v}, {u})")  
 return False  
  
 if self.type != "directed":  
 try:  
 nodes = [x[0] for x in self.adj\_list[u]]  
 del self.adj\_list[u][nodes.index(v)]  
 except (KeyError, ValueError):  
 print(f"ERROR: No such edge ({u}, {v})")  
 return False  
 return True  
  
 *# метод для вывода графа в файл* def print\_to\_file(self, filename):  
 try:  
 fout = open(filename, 'w', encoding="utf8")  
 except FileNotFoundError:  
 print("ERROR: No such file or directory")  
 return False  
 lst = self.create\_edge\_list(False)  
 print(self.type, file=fout)  
 print("weighted" if self.weighted else "!weighted", file=fout)  
 print(len(self.nodes\_list), len(lst), file=fout)  
 print(" ".join(self.nodes\_list), file=fout)  
 for edge in lst:  
 print(f"{edge[0]} {edge[1]}", file=fout)  
 return True  
  
 *# метод для вывода списка смежности в консоль* def print\_to\_console(self):  
 print()  
 if self.weighted:  
 for item in self.adj\_list.items():  
 nodes = ["(" + ",".join([x[0], x[1]]) + ")" for x in item[1]]  
 line = " ".join(nodes)  
 print(f"{item[0]}: {line}")  
 else:  
 for item in self.adj\_list.items():  
 nodes = [x[0] for x in item[1]]  
 line = " ".join(nodes)  
 print(f"{item[0]}: {line}")

Файл *main.py* в котором реализован консольные интерфейс

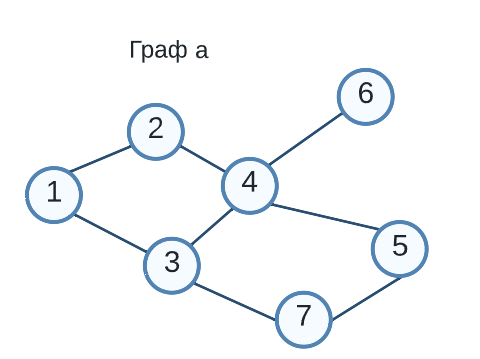
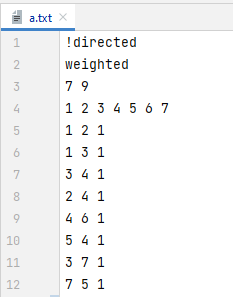
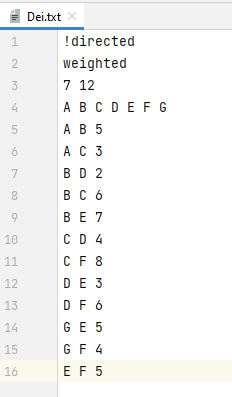
import tasks  
from graph import Graph  
  
commands\_num = {"1": "CREATE EMPTY", "2": "CREATE FILE", "3": "COPY", "4": "CHOOSE GRAPH",  
 "5": "ADD VERTEX", "6": "ADD EDGE", "7": "DELETE VERTEX", "8": "DELETE EDGE",  
 "9": "PRINT LIST EDGES FILE", "10": "PRINT LIST EDGES CONSOLE", "11": "PRINT ADJACENCY LIST CONSOLE",  
 "13": "HELP", "14": "PRINT LIST COMMANDS", "15": "PRINT LIST GRAPHS", "0": "EXIT"}  
  
commands = {"CREATE EMPTY": "Create a new empty graph",  
 "CREATE FILE": "Create a new graph from the file",  
 "COPY": "Create the copy of existing graph",  
 "CHOOSE GRAPH": "Choose the name if graph to work with",  
 "ADD VERTEX": "Add a vertex to the graph",  
 "ADD EDGE": "Add an edge to the graph",  
 "DELETE VERTEX": "Remove a vertex from the graph",  
 "DELETE EDGE": "Remove an edge from the graph",  
 "PRINT LIST EDGES FILE": "Print list of edges to the file",  
 "PRINT LIST EDGES CONSOLE": "Print list of edges to the console(there)",  
 "PRINT ADJACENCY LIST CONSOLE": "Print adjacency list to the console(there)",  
 "HELP": "Print the hint for the command",  
 "EXIT": "Exit the execution",  
 "PRINT LIST COMMANDS": "Print list of commands to the console(there)",  
 "PRINT LIST GRAPHS": "Print list of graphs to the console(there)"  
 }  
  
graphs = {}  
chosen\_graph = None  
  
  
def print\_menu():  
 print("Enter the command from the list")  
 for items in commands\_num.items():  
 print(items[0], items[1])  
  
  
def create(\*filename):  
 fl = True  
 if len(filename) == 1:  
 gr = Graph()  
 fl = gr.create\_from\_file(filename[0])  
 else:  
 print("Enter the type of graph: directed or !directed")  
 type = input()  
 if type != "directed" and type != "!directed":  
 print("ERROR: Unexpected type")  
 return  
 print("Enter the type of graph: weighted or !weighted")  
 weight = input()  
 if weight != "weighted" and weight != "!weighted":  
 print("ERROR: Unexpected type")  
 return  
 gr = Graph(type, weight)  
  
 if fl:  
 print("Enter the name for the new graph")  
 name = input()  
 while name in graphs.keys():  
 print("This name already exists. Enter another")  
 name = input()  
 else:  
 graphs[name] = gr  
  
  
def get\_graph\_by\_name(name):  
 global chosen\_graph  
 if name is None:  
 print("Enter the name of graph or enter STOP to stop entering")  
 in\_name = input()  
 if in\_name in graphs.keys():  
 chosen\_graph = in\_name  
 return graphs[chosen\_graph]  
 else:  
 while in\_name not in graphs.keys():  
 print("ERROR. Try to enter another name or STOP to stop entering")  
 in\_name = input()  
 if in\_name == "STOP":  
 chosen\_graph = None  
 return None  
 else:  
 chosen\_graph = in\_name  
  
 return graphs[chosen\_graph]  
 chosen\_graph = name  
 return graphs[name]  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 print\_menu()  
 print("\nENTER THE COMMAND: ", end="")  
 command = input()  
 cur\_graph = None  
 while command != "EXIT" and command != "0":  
 if command not in commands.keys() and command not in commands\_num.keys():  
 print("ERROR: UNKNOWN COMMAND")  
  
 else:  
 if command == "EXIT" or command == "0":  
 exit(0)  
  
 elif command == "HELP" or command == "13":  
 fl = False  
 print("To exit HELP enter STOP")  
 print("Enter the name of command")  
 name = input()  
 while name != "STOP":  
 while name not in commands.keys():  
 if name == "STOP":  
 fl = True  
 break  
 print("Wrong value. Enter name of command again")  
 name = input()  
 else:  
 print(f"{name} : {commands[name]}")  
 print("To exit HELP enter STOP")  
 if fl:  
 break  
 print("Enter the name of command")  
 name = input()  
  
 elif command == "CREATE EMPTY" or command == "1":  
 create()  
  
 elif command == "CHOOSE GRAPH" or command == "4":  
 cur\_graph = get\_graph\_by\_name(None)  
  
 elif command == "CREATE FILE" or command == "2":  
 print("Enter the name of file")  
 create(input())  
  
 elif command == "COPY" or command == "3":  
 if cur\_graph is None:  
 cur\_graph = get\_graph\_by\_name(None)  
 if cur\_graph is not None:  
 print(f"Current graph is: {chosen\_graph}")  
 print("Enter the name of the copy-graph: ", end="")  
 name = input()  
 if name not in graphs.keys():  
 graphs[name] = cur\_graph.copy()  
 else:  
 while name in graphs.keys():  
 print("ERROR. Try to enter another name or STOP to stop entering")  
 name = input()  
 if name == "STOP":  
 break  
 else:  
 graphs[name] = cur\_graph.copy()  
  
 elif command == "ADD VERTEX" or command == "5":  
 if cur\_graph is None:  
 cur\_graph = get\_graph\_by\_name(None)  
 if cur\_graph is not None:  
 print(f"Current graph is: {chosen\_graph}")  
 print("Enter the vertex to add: ", end="")  
 node = input()  
 cur\_graph.add\_node(node)  
  
 elif command == "ADD EDGE" or command == "6":  
 if cur\_graph is None:  
 cur\_graph = get\_graph\_by\_name(None)  
 if cur\_graph is not None:  
 print(f"Current graph is: {chosen\_graph}")  
 print("Enter the edge to add: ", end="")  
 edge = input().split()  
 cur\_graph.add\_edge(edge)  
  
 elif command == "DELETE VERTEX" or command == "7":  
 if cur\_graph is None:  
 cur\_graph = get\_graph\_by\_name(None)  
 if cur\_graph is not None:  
 print(f"Current graph is: {chosen\_graph}")  
 print("Enter the vertex to delete: ", end="")  
 node = input()  
 cur\_graph.delete\_node(node)  
  
 elif command == "DELETE EDGE" or command == "8":  
 if cur\_graph is None:  
 cur\_graph = get\_graph\_by\_name(None)  
 if cur\_graph is not None:  
 print(f"Current graph is: {chosen\_graph}")  
 print('Enter the edge to delete":', end="")  
 edge = input().split()  
 cur\_graph.delete\_edge(edge)  
  
 elif command == "PRINT LIST EDGES FILE" or command == "9":  
 if cur\_graph is None:  
 cur\_graph = get\_graph\_by\_name(None)  
 if cur\_graph is not None:  
 print(f"Current graph is: {chosen\_graph}")  
 print("Enter the name of file")  
 filename = input()  
 cur\_graph.print\_to\_file(filename)  
  
 elif command == "PRINT LIST EDGES CONSOLE" or command == "10":  
 if cur\_graph is None:  
 cur\_graph = get\_graph\_by\_name(None)  
 if cur\_graph is not None:  
 print(f"Current graph is: {chosen\_graph}")  
 lst = cur\_graph.create\_edge\_list(False)  
 for e in lst:  
 if cur\_graph.weighted:  
 print(\*e)  
 else:  
 print(f"{e[0]} {e[1]}")

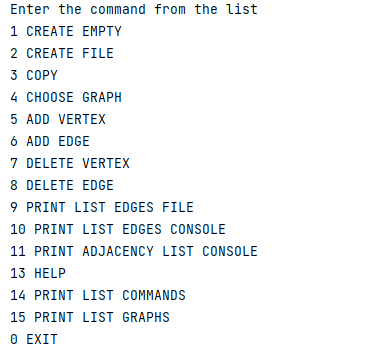
elif command == "PRINT ADJACENCY LIST CONSOLE" or command == "11":  
 if cur\_graph is None:  
 cur\_graph = get\_graph\_by\_name(None)  
 if cur\_graph is not None:  
 print(f"Current graph is: {chosen\_graph}")  
 cur\_graph.print\_to\_console()  
  
 elif command == "PRINT LIST COMMANDS" or command == "14":  
 print\_menu()  
  
 elif command == "PRINT LIST GRAPHS" or command == "15":  
 for gr in graphs.items():  
 print(f"Name: {gr[0]}\tAdjacency list: ", end="")  
 gr[1].print\_to\_console()  
 print()  
  
 print("\nENTER THE COMMAND: ", end="")  
 command = input()

Далее все задачи реализованы в виде отдельных методов в поключенном файле *tasks.py*

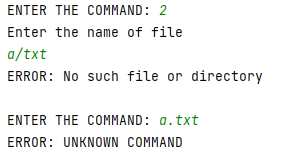
Примеры

**Создание графа из файла**

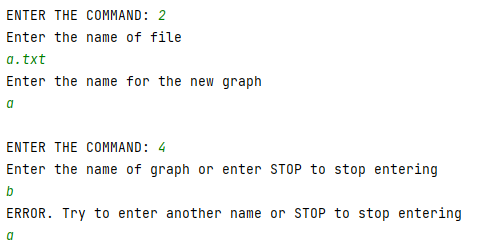
** 



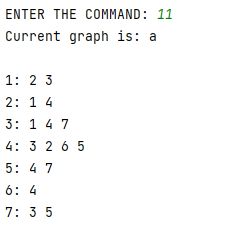
Ошибка при неправильном имени файла и при неизвестной команде



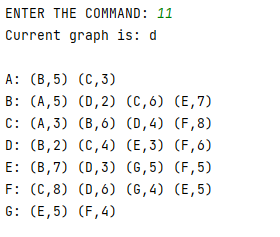
Создание и выбор графа



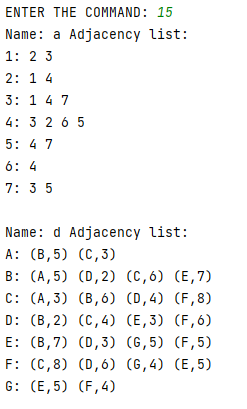
Вывод списка смежности (при невзвешенном графе)



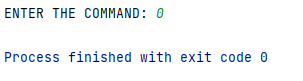
Вывод списка смежности (при взвешенном графе)



Вывод всех графов

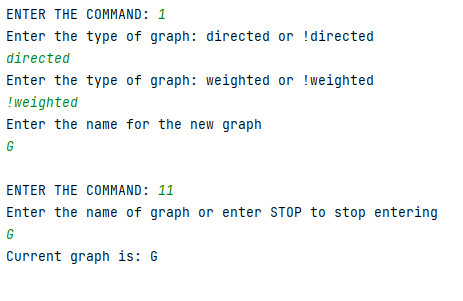


Завершение работы с программой

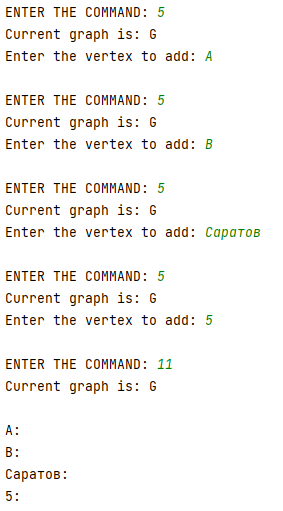


**Создание пустого графа и его наполнение**

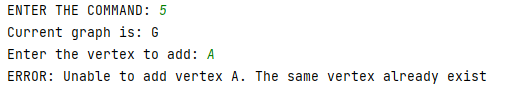
Создание графа

****

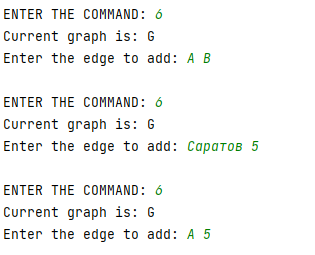
Добавление вершин

****

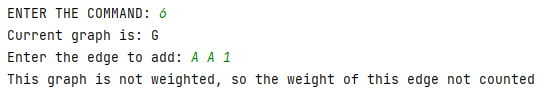
Попытка добавить существующую вершину



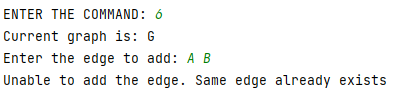
Добавление ребер



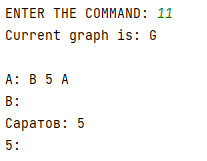
Попытка ввести в невзвешенных граф ребро с весом



Попытка добавить существующее ребро

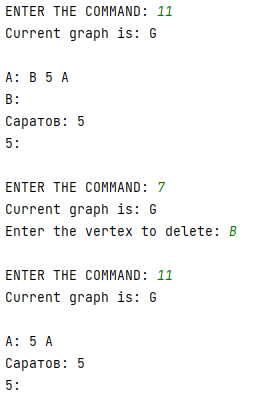


Результат

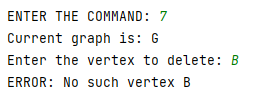


**Удаление вершин и ребер**

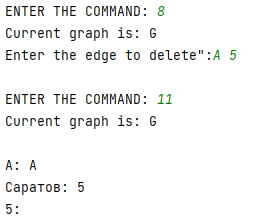
Удаление вершин



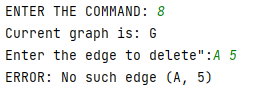
Попытка удалить несуществующую вершину



Удаление ребра (дуги)

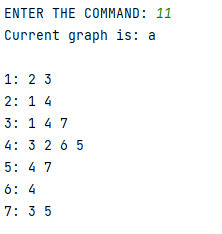


Попытка удалить несуществующее ребро

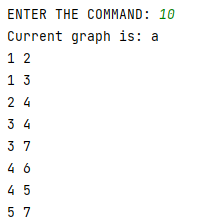


**Вывод в файл и консоль**

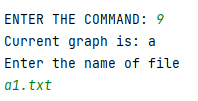
Вывод списка смежности в консоль

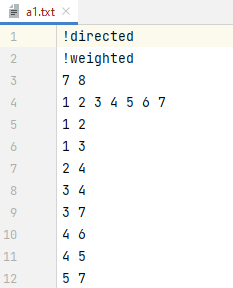


Вывод списка ребер в консоль

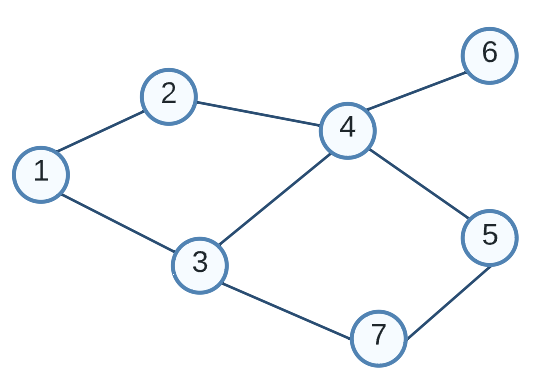
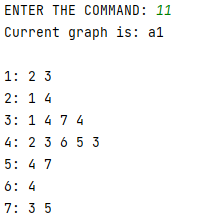


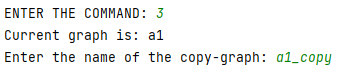
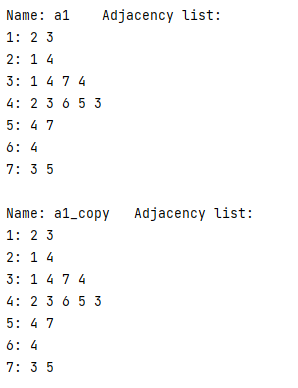
Вывод списка ребер в файл (реализовано так, чтобы потом этот же файл можно было бы считать)



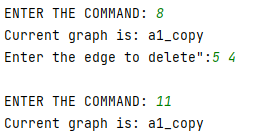


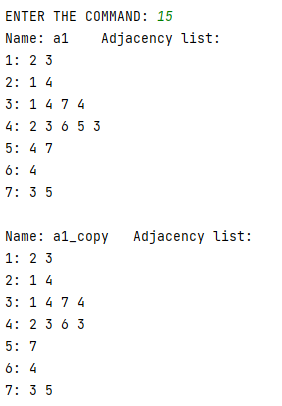
**Копия графа (+ проверка)**

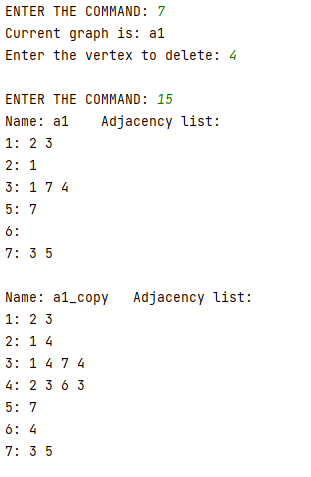
 

Проверим, удалив ребро из копии





Теперь удалим вершину из графа основы



Список смежности

2. Список смежности Ia + 3. Список смежности Ia





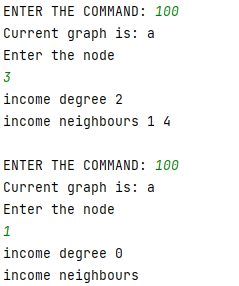
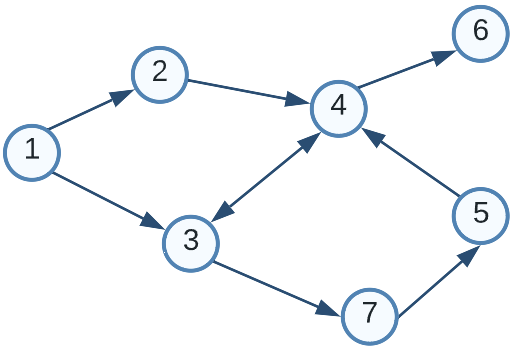
Файл *main.py*

elif command == "IN DEGREE" or command == "100":  
 if cur\_graph is None:  
 cur\_graph = get\_graph\_by\_name(None)  
 if cur\_graph is not None:  
 print(f"Current graph is: {chosen\_graph}")  
 print("Enter the node")  
 node = input()  
 ans = tasks.in\_deg(cur\_graph, node)  
 if ans != -1:  
 if cur\_graph.type == "!directed":  
 print("This graph is not directed")  
 print(f"degree {ans[0]}\nneighbours {' '.join(ans[1])}")  
 else:  
 print(f"income degree {ans[0]}\nincome neighbours {' '.join(ans[1])}")

Файл *tasks.py*

def in\_deg(graph, node):  
 if node not in graph.nodes\_list:  
 print(f"No such vertex {node}")  
 return -1  
 ans = 0  
 neib = []  
 for item in graph.adj\_list.items():  
 nodes = [x[0] for x in item[1]]  
 if node in nodes:  
 neib.append(item[0])  
 ans += 1  
 return ans, neib

Примеры



4. Список смежности Iб: несколько графов



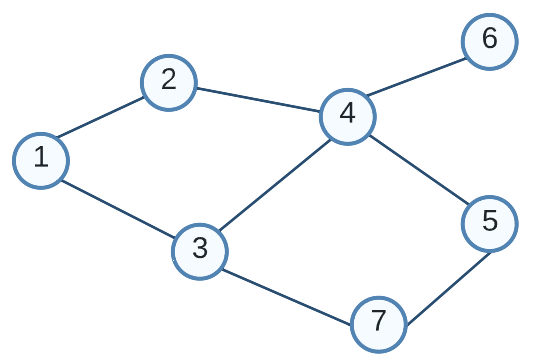
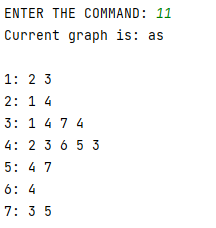
Файл *main.py*

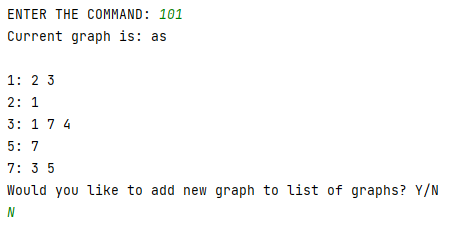
elif command == "TASK Ib 18" or command == "101":  
 if cur\_graph is None:  
 cur\_graph = get\_graph\_by\_name(None)  
 if cur\_graph is not None:  
 print(f"Current graph is: {chosen\_graph}")  
 if cur\_graph.type == "directed":  
 print(f"Unable to create the new graph from {chosen\_graph}")  
 else:  
 new\_gr = tasks.new\_graph(cur\_graph)  
 new\_gr.print\_to\_console()  
 print("Would you like to add new graph to list of graphs? Y/N")  
 ans = input()  
 if ans == "Y":  
 print("Enter the name of the new-graph: ", end="")  
 name = input()  
 if name not in graphs.keys():  
 graphs[name] = new\_gr  
 else:  
 while name in graphs.keys():  
 print("ERROR. Try to enter another name or STOP to stop entering")  
 name = input()  
 if name == "STOP":  
 break  
 else:  
 graphs[name] = new\_gr  
 elif ans != "N":  
 print("Cannot recognise the answer")

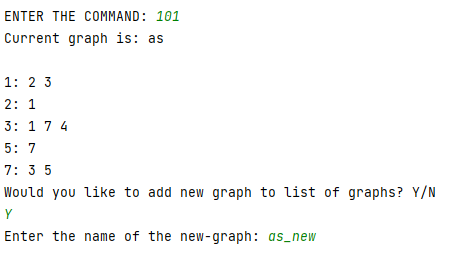
Файл *tasks.py*

def new\_graph(graph):  
 gr = graph.copy()  
 del\_nodes = []  
 for node in gr.adj\_list.items():  
 if len(node[1]) % 2 != 0:  
 del\_nodes.append(node[0])  
 for node in del\_nodes:  
 gr.delete\_node(node)  
 return gr

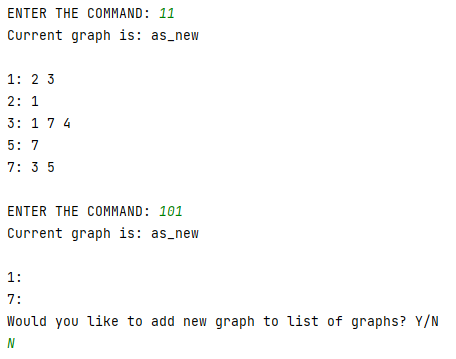
Примеры

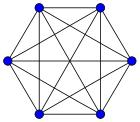
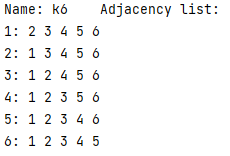


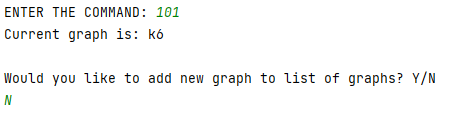


Применим метод еще раз к полученному графу

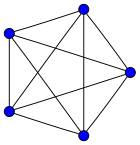
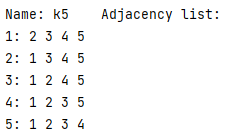


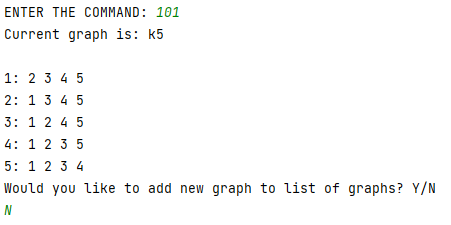
**Пример для полного графа K6 (будут удалены все вершины)**

 ****

****

**Пример для полного графа K5 (не будет удалено ни одной вершины)**

 ****

****

Обходы графа

5. Обходы графа II (DFS)



Файл *main.py*

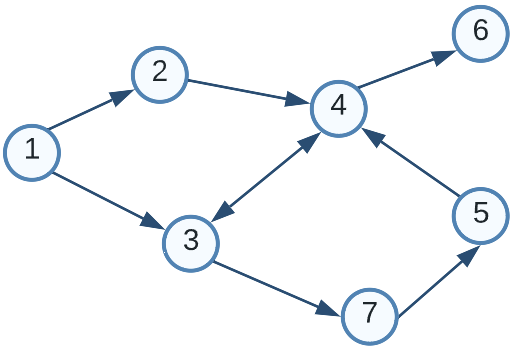
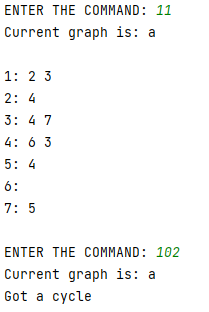
elif command == "ACYCLE" or command == "102":  
 if cur\_graph is None:  
 cur\_graph = get\_graph\_by\_name(None)  
 if cur\_graph is not None:  
 print(f"Current graph is: {chosen\_graph}")  
 res = tasks.acycle(cur\_graph)  
 if res:  
 print("Acycle")  
 elif not res:  
 print("Got a cycle")

Файл *tasks.py*

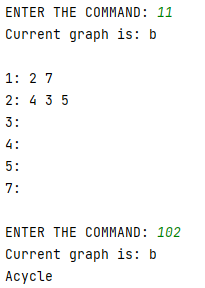
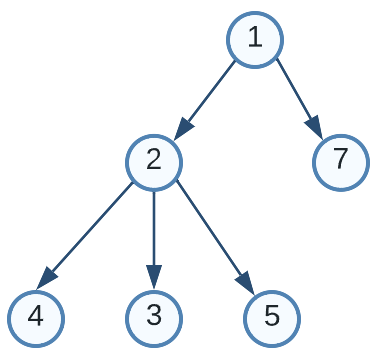
color = []  
cycle = False  
INF = 1e15  
  
  
def dfs(node, gr):  
 global color, cycle  
 color[gr.nodes\_list.index(node)] = 1  
 lst = [x[0] for x in gr.adj\_list[node]]  
 for nd in lst:  
 if color[gr.nodes\_list.index(nd)] == 0:  
 dfs(nd, gr)  
 elif color[gr.nodes\_list.index(nd)] == 1:  
 cycle = True  
  
 color[gr.nodes\_list.index(node)] = 2  
  
  
def acycle(graph):  
 if graph.type == "!directed":  
 print("Graph is not directed")  
 return  
 global color, cycle  
 gr = graph.copy()  
 color = [0 for \_ in range(len(gr.nodes\_list))]  
 for node in gr.nodes\_list:  
 if color[gr.nodes\_list.index(node)] == 0:  
 dfs(node, gr)  
 if cycle:  
 return False  
 return True

Примеры

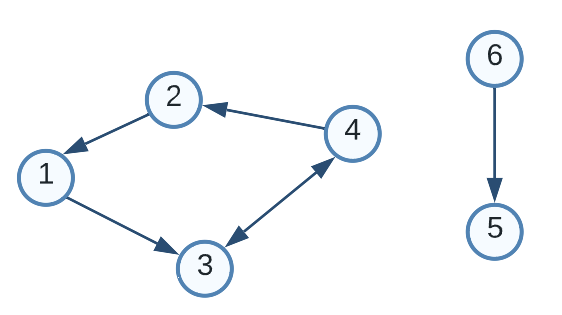
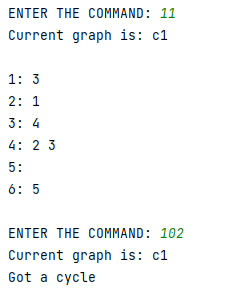
**Связный орграф с циклом**

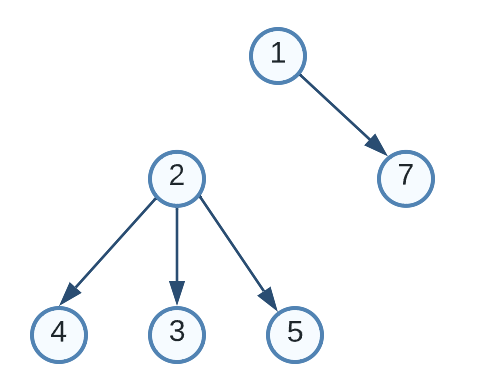
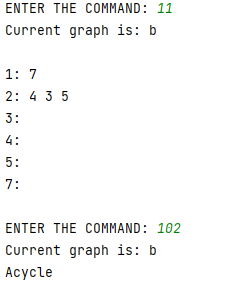
**Связный орграф без цикла**

****

**Несвязный орграф с циклом**

**** ****

**Несвязный орграф без цикла**

**** ****

6. Обходы графа II (BFS)



Файл *main.py*

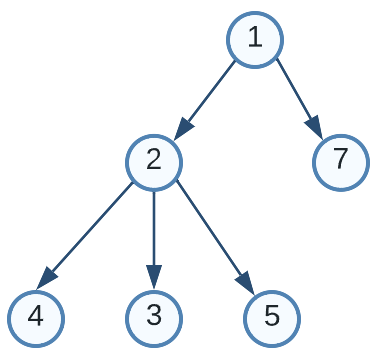
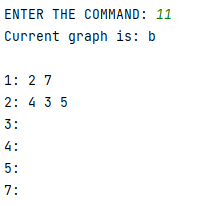
elif command == "K\_PATH" or command == "103":  
 if cur\_graph is None:  
 cur\_graph = get\_graph\_by\_name(None)  
 if cur\_graph is not None:  
 print(f"Current graph is: {chosen\_graph}")  
 print("Enter the value of k: ", end="")  
 try:  
 k = int(input())  
 tasks.k\_path(cur\_graph, k)  
 except ValueError:  
 print("ERROR Value of k is not integer")

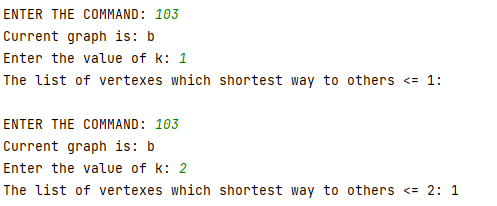
Файл *tasks.py*

def bfs(s, gr):  
 used = [False for \_ in range(len(gr.nodes\_list))]  
 d = [-1 for \_ in range(len(gr.nodes\_list))]  
 p = [None for \_ in range(len(gr.nodes\_list))]  
 used[gr.nodes\_list.index(s)] = True  
 q = Queue()  
 q.put(s)  
 while not q.empty():  
 node = q.get()  
 lst = [x[0] for x in gr.adj\_list[node]]  
 for nd in lst:  
 to = gr.nodes\_list.index(nd)  
 if not used[to]:  
 used[to] = True  
 q.put(nd)  
 d[to] = d[gr.nodes\_list.index(node)] + 1  
 p[to] = node  
 return d, p  
  
  
def k\_path(graph, k):  
 gr = graph.copy()  
 ans = []  
 for node in gr.nodes\_list:  
 d, p = bfs(node, gr)  
 if max(d) + 1 <= k and d.count(-1) == 1:  
 ans.append(node)  
 *#print(node, d, p)* print(f"The list of vertexes which shortest way to others <= {k}: {' '.join(ans)}")

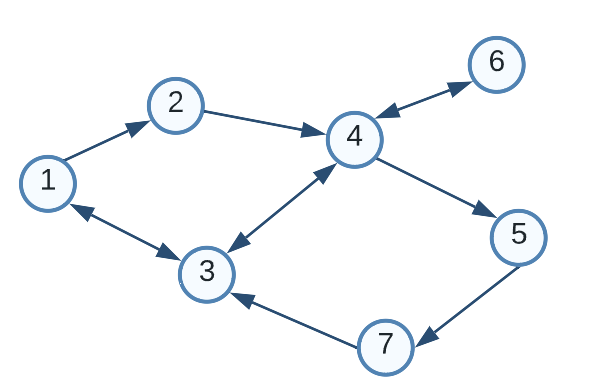
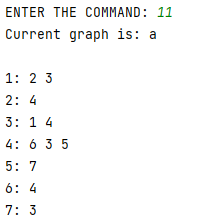
Примеры

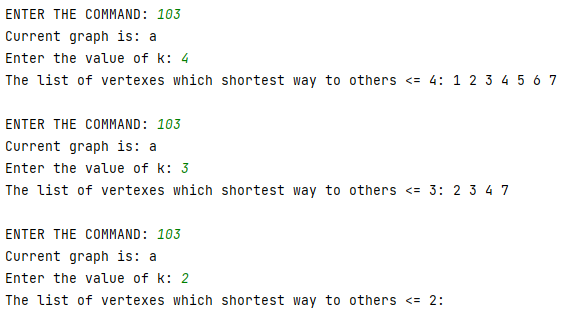
**Связный орграф**

**** 

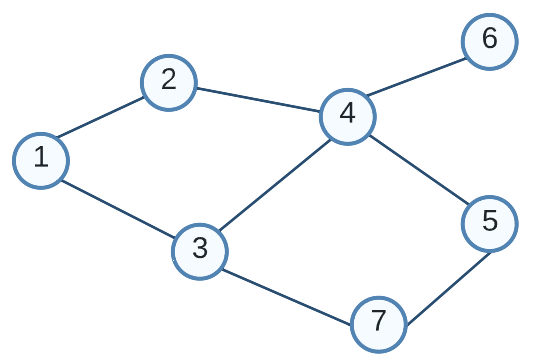
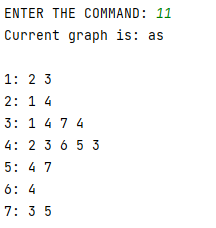


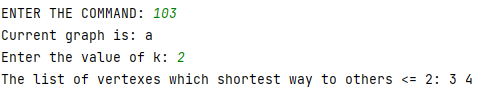
**Сильно-связный орграф**

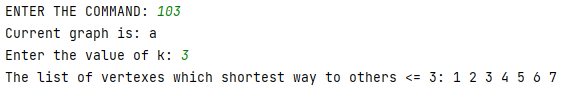
 



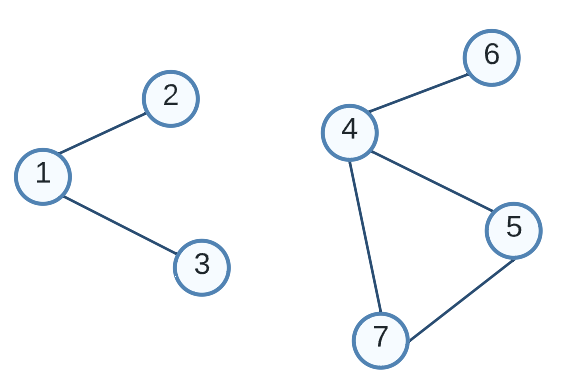
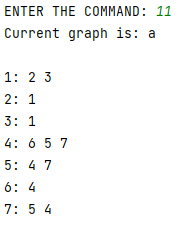
**Связный граф**

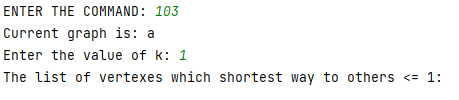
 





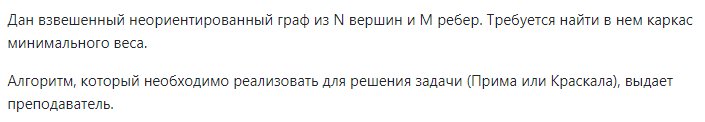
**Несвязный граф**



Построение минимального остовного дерева

7. Каркас III (Краскал)



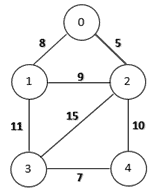
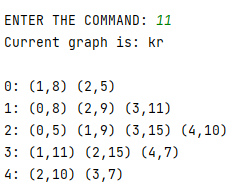
Файл *main.py*

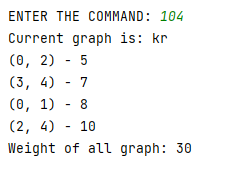
elif command == "KRUSKAL" or command == "104":  
 if cur\_graph is None:  
 cur\_graph = get\_graph\_by\_name(None)  
 if cur\_graph is not None:  
 print(f"Current graph is: {chosen\_graph}")  
 tasks.kruskal(cur\_graph)

Файл *tasks.py*

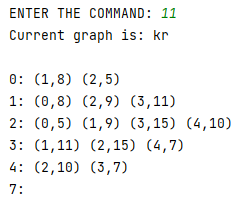
def search(gr, parent, i):  
 if parent[gr.nodes\_list.index(i)] == i:  
 return i  
 return search(gr, parent, parent[gr.nodes\_list.index(i)])  
  
  
def apply\_union(gr, parent, rank, x, y):  
 xroot = search(gr, parent, x)  
 yroot = search(gr, parent, y)  
 if rank[gr.nodes\_list.index(xroot)] < rank[gr.nodes\_list.index(yroot)]:  
 parent[gr.nodes\_list.index(xroot)] = yroot  
 elif rank[gr.nodes\_list.index(xroot)] > rank[gr.nodes\_list.index(yroot)]:  
 parent[gr.nodes\_list.index(yroot)] = xroot  
 else:  
 parent[gr.nodes\_list.index(yroot)] = xroot  
 rank[gr.nodes\_list.index(xroot)] += 1  
  
  
def kruskal(graph):  
 if graph.type == "directed":  
 print("ERROR: Incorrect type of graph")  
 return  
 gr = graph.copy()  
 n = len(gr.nodes\_list)  
 result = []  
 i, j = 0, 0  
 edge\_list = gr.create\_edge\_list(False)  
 parent = [gr.nodes\_list[i] for i in range(n)]  
 rank = [0 for \_ in range(n)]  
 try:  
 edge\_list = sorted(edge\_list, key=lambda item: int(item[2]))  
 while j < n - 1:  
 if i > len(edge\_list) - 1:  
 print("EROOR: Can't build the tree")  
 return  
 u, v, w = edge\_list[i]  
 i += 1  
 x = search(graph, parent, u)  
 y = search(graph, parent, v)  
 if x != y:  
 j += 1  
 result.append([u, v, w])  
 apply\_union(graph, parent, rank, x, y)  
 except ValueError:  
 print("ERROR: Incorrect type of weight")  
 return  
 result\_weight = 0  
 for u, v, weight in result:  
 result\_weight += int(weight)  
 print(f"({u}, {v}) - {weight}")  
 print(f"Weight of all graph: {result\_weight}")

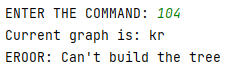
Примеры

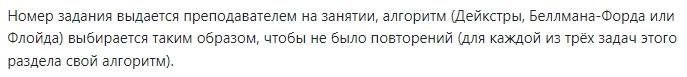


Добавим изолированную вершину





Взвешенный граф



8. Веса IV а





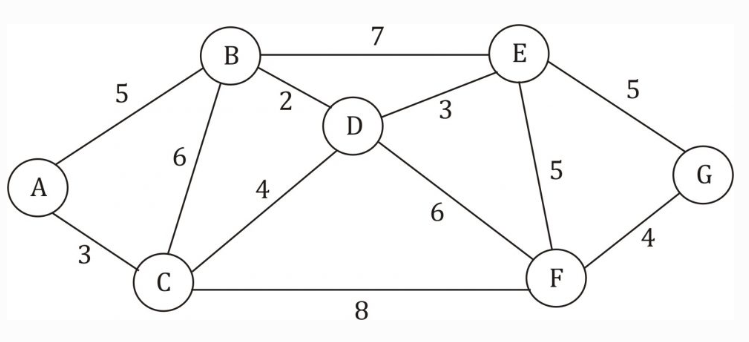
Файл *main.py*

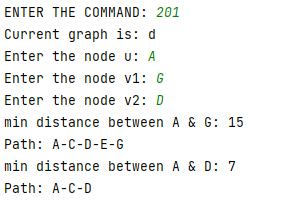
elif command == "DIJKSTRA" or command == "201":  
 if cur\_graph is None:  
 cur\_graph = get\_graph\_by\_name(None)  
 if cur\_graph is not None:  
 print(f"Current graph is: {chosen\_graph}")  
 print(f"Enter the node u: ", end="")  
 u = input()  
 print(f"Enter the node v1: ", end="")  
 v1 = input()  
 print(f"Enter the node v2: ", end="")  
 v2 = input()  
 if u not in cur\_graph.nodes\_list or v1 not in cur\_graph.nodes\_list or v2 not in cur\_graph.nodes\_list:  
 print("ERROR No such node")  
 else:  
 d, pr = tasks.dijkstra(cur\_graph, u)  
 *#print(d, pr)* if d[cur\_graph.nodes\_list.index(v1)] != tasks.INF:  
 print(f"min distance between {u} & {v1}: {d[cur\_graph.nodes\_list.index(v1)]}")  
 path = []  
 v = cur\_graph.nodes\_list.index(v1)  
 s = cur\_graph.nodes\_list.index(u)  
 while v != s:  
 path.append(cur\_graph.nodes\_list[v])  
 v = pr[v]  
 print(f"Path: {u}-{'-'.join(path[::-1])}")  
 else:  
 print(f"No path between {u} & {v1}")  
  
 if d[cur\_graph.nodes\_list.index(v2)] != tasks.INF:  
 print(f"min distance between {u} & {v2}: {d[cur\_graph.nodes\_list.index(v2)]}")  
 path = []  
 v = cur\_graph.nodes\_list.index(v2)  
 s = cur\_graph.nodes\_list.index(u)  
 while v != s:  
 path.append(cur\_graph.nodes\_list[v])  
 v = pr[v]  
 print(f"Path: {u}-{'-'.join(path[::-1])}")  
 else:  
 print(f"No path between {u} & {v2}")

Файл *tasks.py*

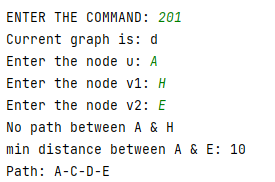
def dijkstra(graph, s):  
 gr = graph.copy()  
 n = len(gr.nodes\_list)  
 d = [INF] \* n  
 pr = [0] \* n  
 d[gr.nodes\_list.index(s)] = 0  
 used = [False for \_ in range(len(gr.nodes\_list))]  
 for i in range(n):  
 v = -1  
 for j in range(n):  
 if not used[j] and (v == -1 or d[j] < d[v]):  
 v = j  
 if d[v] == INF:  
 break  
 used[v] = True  
 lst = gr.adj\_list[gr.nodes\_list[v]]  
 for nd in lst:  
 to = gr.nodes\_list.index(nd[0])  
 ln = int(nd[1])  
 if d[v] + ln < d[to]:  
 d[to] = d[v] + ln  
 pr[to] = v  
 return d, pr

Примеры





При добавлении изолированной вершины H



9. Веса IV b





Файл *main.py*

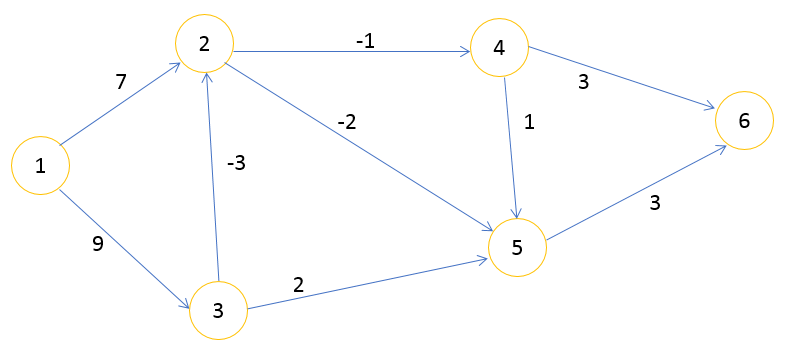
elif command == "BELLMAN\_FORD" or command == "202":  
 if cur\_graph is None:  
 cur\_graph = get\_graph\_by\_name(None)  
 if cur\_graph is not None:  
 print(f"Current graph is: {chosen\_graph}")  
 print(f"Enter the node u: ", end="")  
 u = input()  
 if u not in cur\_graph.nodes\_list:  
 print("ERROR No such node")  
 else:  
 n = len(cur\_graph.nodes\_list)  
 for v in range(n):  
 if cur\_graph.nodes\_list[v] != u:  
 d, pr = tasks.bellman\_ford(cur\_graph, cur\_graph.nodes\_list[v])  
 *#print(d, pr)* if d[cur\_graph.nodes\_list.index(u)] != tasks.INF:  
 print(f"min distance between {cur\_graph.nodes\_list[v]} & {u}: {d[cur\_graph.nodes\_list.index(u)]}")  
 path = []  
 v1 = cur\_graph.nodes\_list.index(u)  
 s = v  
 while v1 != s:  
 path.append(cur\_graph.nodes\_list[v1])  
 v1 = pr[v1]  
 print(f"Path: {cur\_graph.nodes\_list[v]}-{'-'.join(path[::-1])}")  
 else:  
 print(f"No path between {cur\_graph.nodes\_list[v]} & {u}")

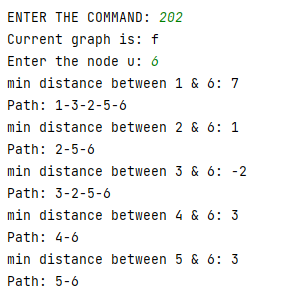
Файл *tasks.py*

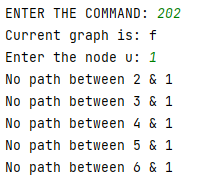
def bellman\_ford(graph, s):  
 gr = graph.copy()  
 n = len(gr.nodes\_list)  
 d = [INF] \* n  
 d[gr.nodes\_list.index(s)] = 0  
 edges = gr.create\_edge\_list(True)  
 pr = [0] \* n  
 *# print(edges)* m = len(edges)  
 while True:  
 flag = False  
 for j in range(m):  
 e = edges[j]  
 a = gr.nodes\_list.index(e[0])  
 b = gr.nodes\_list.index(e[1])  
 cost = int(e[2])  
 if d[a] < INF:  
 if d[b] > d[a] + cost:  
 d[b] = d[a] + cost  
 pr[b] = a  
 flag = True  
 if not flag:  
 break  
 return d, pr

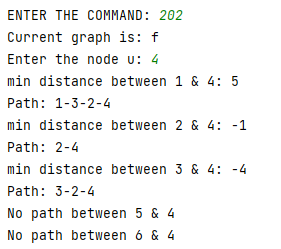
Примеры

**Граф с отрицательными ребрами**

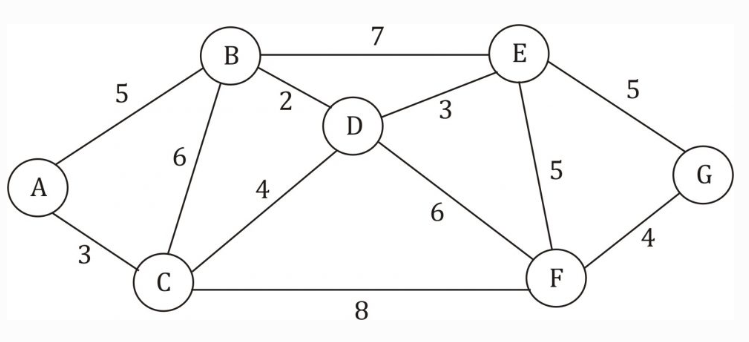


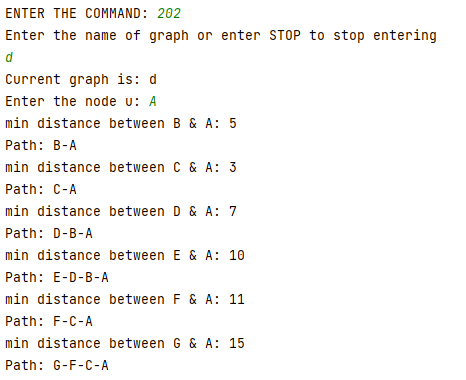






**Граф с положительными ребрами**



****

10. Веса IV с





Файл *main.py*

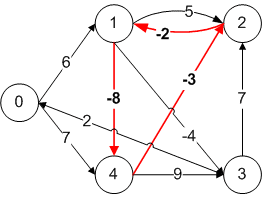
elif command == "FLOYD" or command == "203":  
 if cur\_graph is None:  
 cur\_graph = get\_graph\_by\_name(None)  
 if cur\_graph is not None:  
 print(f"Current graph is: {chosen\_graph}")  
 print(f"Enter the node u: ", end="")  
 u = input()  
 print(f"Enter the node v1: ", end="")  
 v1 = input()  
 print(f"Enter the node v2: ", end="")  
 v2 = input()  
 if u not in cur\_graph.nodes\_list or v1 not in cur\_graph.nodes\_list or v2 not in cur\_graph.nodes\_list:  
 print("ERROR No such node")  
 else:  
 A, pr, cycle = tasks.floyd(cur\_graph)  
 *# print(d, pr)* if cycle:  
 print('Negative-weight cycle found')  
 else:  
 if A[cur\_graph.nodes\_list.index(u)][cur\_graph.nodes\_list.index(v1)] != tasks.INF:  
 print(f"min distance between {u} & {v1}: {A[cur\_graph.nodes\_list.index(u)][cur\_graph.nodes\_list.index(v1)]}")  
 path = []  
 v = cur\_graph.nodes\_list.index(v1)  
 s = cur\_graph.nodes\_list.index(u)  
  
 while v != s:  
 *#print(v, pr[s][v])* path.append(cur\_graph.nodes\_list[v])  
 v = pr[s][v]  
 print(f"Path: {u}-{'-'.join(path[::-1])}")  
 else:  
 print(f"No path between {u} & {v1}")  
  
 if A[cur\_graph.nodes\_list.index(u)][cur\_graph.nodes\_list.index(v2)] != tasks.INF:  
 print(f"min distance between {u} & {v2}: {A[cur\_graph.nodes\_list.index(u)][cur\_graph.nodes\_list.index(v2)]}")  
 path = []  
 v = cur\_graph.nodes\_list.index(v2)  
 s = cur\_graph.nodes\_list.index(u)  
 while v != s:  
 *#print(v)* path.append(cur\_graph.nodes\_list[v])  
 v = pr[s][v]  
 print(f"Path: {u}-{'-'.join(path[::-1])}")  
 else:  
 print(f"No path between {u} & {v2}")

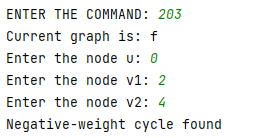
Файл *tasks.py*

def create\_matrix\_adj(gr):  
 n = len(gr.adj\_list)  
 A = [[INF if i != j else 0 for i in range(n)] for j in range (n)]  
 for x in gr.adj\_list:  
 a = gr.nodes\_list.index(x)  
 for v in gr.adj\_list[x]:  
 b = gr.nodes\_list.index(v[0])  
 cost = int(v[-1])  
 A[a][b] = cost  
 if gr.type == "!directed":  
 A[b][a] = cost  
 return A  
  
  
def create\_matrix\_pr(gr):  
 n = len(gr.adj\_list)  
 pr = [[None for i in range(n)] for j in range(n)]  
 for x in gr.adj\_list:  
 a = gr.nodes\_list.index(x)  
 for v in gr.adj\_list[x]:  
 b = gr.nodes\_list.index(v[0])  
 pr[a][b] = a  
 if gr.type == "!directed":  
 pr[b][a] = b  
 return pr  
  
  
def print\_matrix(A):  
 for x in A:  
 for y in x:  
 if y == INF:  
 print(f"{'INF': ^5}", end=" ")  
 else:  
 print(f"{'None' if y is None else y: ^5}", end=" ")  
 print()  
  
  
def floyd(graph):  
 gr = graph.copy()  
 n = len(gr.adj\_list)  
 A = create\_matrix\_adj(gr)  
 cycle = False  
 pr = create\_matrix\_pr(gr)  
 for k in range(n):  
 for v in range(n):  
 for u in range(n):  
 if A[v][k] != INF and A[k][u] != INF and A[v][k] + A[k][u] < A[v][u]:  
 A[v][u] = A[v][k] + A[k][u]  
 pr[v][u] = pr[k][u]  
 *#print(pr[v][u], pr[v][k], pr[k][u])* if A[v][v] < 0:  
 cycle = True  
 *# print('Negative-weight cycle found')  
 # return  
 # print(k)  
 # print\_matrix(A)  
 # print()  
 # print\_matrix(pr)  
 # print()  
 #print\_matrix(pr)* return A, pr, cycle

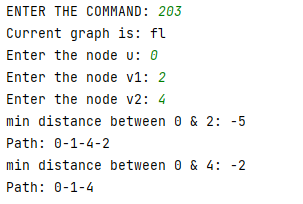
Примеры

**Граф с циклом отрицательного веса**

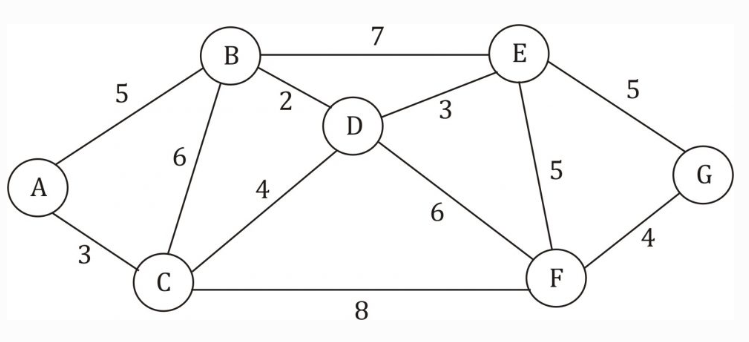


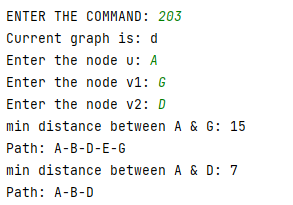


Уберем дугу (2, 1) – разорвем цикл

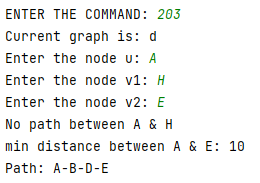


**Граф с положительными ребрами**

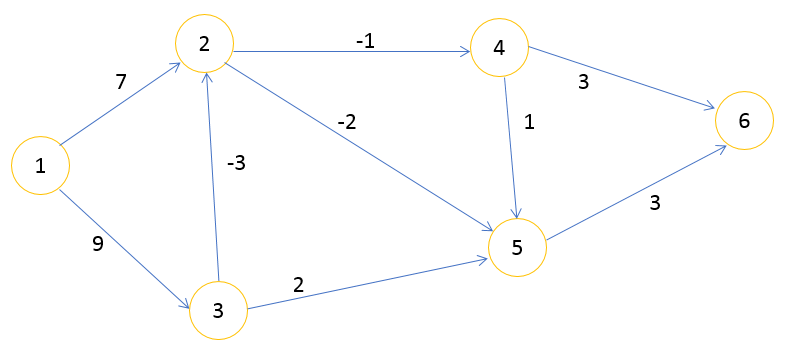


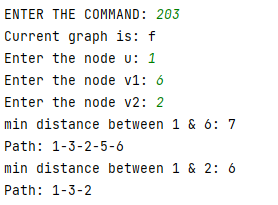


При добавлении изолированной вершины H

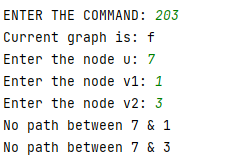


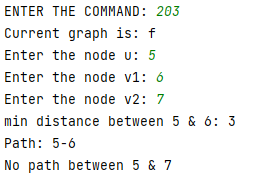
**Граф с отрицательными ребрами**





При добавлении изолированной вершины 7





Потоки

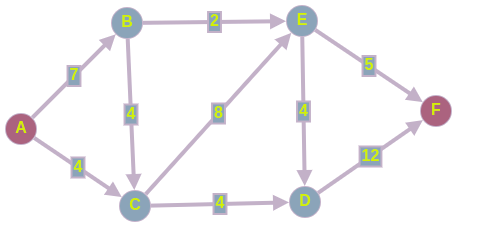
11. Максимальный поток V



Файл *main.py*

Файл *tasks.py*

Примеры



Творческая задача

Задание на визуализацию

