|  |  |
| --- | --- |
|  | МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ  федеральное государственное бюджетное образовательное учреждение высшего образования  «Самарский государственный технический университет» (ФГБОУ ВО «СамГТУ»)  Институт автоматики и информационных технологий |

Кафедра «Вычислительная техника»

Дисциплина «Технологии мультисервисных сетей»

## Отчёт о выполнении лабораторной работы №1,2,3,4

## на тему «Представление топологии сети в терминах теории графов»

|  |  |
| --- | --- |
| Выполнили | Студенты гр. 1-ИАИТ-21ИАИТ-103М Красильников Э.В,  Гукасян Д.В,  Головизов Д.И,  Аршинов Р.А |
|  |  |
| Проверил | к.т.н., доцент Гавлиевский С.Л. |

Самара 2022

**Задание:**

На базе модулей *NetworkX* и *MatPlotLib* разработать тренажер работы с графами, который должен обеспечивать ввод, корректировку, отображение, сохранение топологии графа. Тренажер должен обеспечить следующие возможности:

1. Добавлять и исключать узлы по как одному, так и по несколько узлов одновременно;
2. Добавлять и исключать ребра по как одному, так и по несколько ребер одновременно;
3. Сохранять в файл топологию в графическом виде;
4. Сохранять текущую топологию в файл и считывать ее из файла;
5. Выводить на экран топологию сети в графическом виде;
6. Выводить на экран информацию об общем числе узлов и ветвей сети, соседях конкретного узла.
7. Всё вышеперечисленное со взвешенными графами
8. Поиск кратчайшего пути методом BFS и Алгоритмом Дейкстры

Программа должна обеспечивать защиту от некорректных действий (ошибок ввода, работы с файлами)

Выполнение:

Код программы представлен в листинге 1

**import** matplotlib.pyplot **as** plt  
**import** networkx **as** nx  
**import** tkinter **as** tk  
**from** tkinter **import** filedialog  
**from** PIL **import** Image, ImageTk  
  
**class** Programm(tk.Tk):  
 **def** \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 *# Создадим экземпляр класса nx.Graph*  
self.G = nx.Graph()  
 self.geometry(**'1200x600+50+50'**)  
 self.widgets()  
 self.nodesList = []  
 self.edgesList = []  
 self.weightList=[]  
 self.pathList = []  
 self.f = plt.Figure(figsize=(5, 5), dpi=100)  
 self.plot = tk.Label(self,image=**""**)  
 self.test()  
  
 **def** widgets(self):  
 tk.Label(self, text=**"input tab"**).grid(row=0, column=1)  
 tk.Label(self, text=**"add nodes sep by space"**).grid(row=1)  
 tk.Label(self, text=**"add edges in format n,n"**).grid(row=2)  
 tk.Label(self, text=**"add weights in format n,n,w"**).grid(row=3)  
 self.addNodesBox = tk.Entry(self)  
 self.addNodesBox.grid(row=1, column=1)  
 self.addEdgesBox = tk.Entry(self)  
 self.addEdgesBox.grid(row=2, column=1)  
 self.addWeightBox = tk.Entry(self)  
 self.addWeightBox.grid(row=3, column=1)  
 tk.Button(self,text=**'add'**, command=self.addNodes).grid(row=1, column=2)  
 tk.Button(self, text=**'add'**, command=self.addEdges).grid(row=2, column=2)  
 tk.Button(self, text=**'add'**, command=self.addWeight).grid(row=3, column=2)  
 tk.Label(self, text=**"del tab"**).grid(row=0,column=4)  
 tk.Label(self, text=**"del nodes sep by space"**).grid(row=1,column=3)  
 tk.Label(self, text=**"del edges in format n,n"**).grid(row=2,column=3)  
 tk.Label(self, text=**"del weights in format n,n,w"**).grid(row=3, column=3)  
 self.removeNodesBox = tk.Entry(self)  
 self.removeNodesBox.grid(row=1, column=4)  
 self.removeEdgesBox = tk.Entry(self)  
 self.removeEdgesBox.grid(row=2, column=4)  
 self.removeWeightsBox = tk.Entry(self)  
 self.removeWeightsBox.grid(row=3, column=4)  
 tk.Button(self, text=**'del'**, command=self.removeNodes).grid(row=1, column=5)  
 tk.Button(self, text=**'del'**, command=self.removeEdges).grid(row=2, column=5)  
 tk.Button(self, text=**'del'**, command=self.removeWeights).grid(row=3, column=5)  
 tk.Button(self, text=**'load graph'**, command=self.loadGraph).grid(row=1, column=7)  
 tk.Button(self, text=**'sefe graph'**, command=self.saveGraph).grid(row=2, column=7)  
 self.pathFrame = tk.Frame(self)  
 self.pathFrame.grid(row=0,column=6,rowspan=4)  
 tk.Label(self.pathFrame, text=**"path tab"**).grid(row=0, column=0, columnspan=2)  
 tk.Label(self.pathFrame, text=**"path in format n n"**).grid(row=1, column=0,columnspan=2)  
 self.pathBox=tk.Entry(self.pathFrame)  
 self.pathBox.grid(row=2,column=0,columnspan=2)  
 tk.Button(self.pathFrame,text=**'find path Dijkstra’s'**,command=self.path).grid(row=3,column=0)  
 tk.Button(self.pathFrame, text=**'find path BFS'**, command=self.pathBFS).grid(row=3, column=1)  
 tk.Button(self,text=**'reload'**,command=self.vis).grid(row=1,column=8)  
  
 self.info = tk.Text(self,width=20)  
 self.info.grid(row=4,column=0)  
 **def** test(self):  
 self.addNodesBox.insert(0,**'1 2 3 4 5 6'**)  
 self.addNodes()  
 self.addEdgesBox.insert(0,**'1,2 2,3 3,4 4,5 5,6 6,1 1,4'**)  
 self.addEdges()  
 self.addWeightBox.insert(0,**'1,2,5 2,3,4 3,4,6 4,5,9 5,6,5 6,1,1 1,4,2'**)  
 self.addWeight()  
 **def** path(self):  
 nodesList = self.pathBox.get().split(**" "**)  
 self.addNodesBox.delete(0, **"end"**)  
 **if** self.checkNode(nodesList[0]) **or** self.checkNode(nodesList[1]): **return False**  
 **else**:  
 predecessors, \_ = nx.floyd\_warshall\_predecessor\_and\_distance(self.G)  
 self.pathList = nx.reconstruct\_path(nodesList[0], nodesList[1], predecessors)  
 self.vis()  
 **def** pathBFS(self):  
 nodesList = self.pathBox.get().split(**" "**)  
 self.addNodesBox.delete(0, **"end"**)  
 **if** self.checkNode(nodesList[0]) **or** self.checkNode(nodesList[1]): **return False**  
 **else**:  
 explored=[]  
 queue = [[nodesList[0]]]  
 **if** nodesList[0] == nodesList[1]:  
 self.addNodesBox.insert(0, **"same nodes"**)  
 **return**  
 **while** queue:  
 path = queue.pop(0)  
 node = path[-1]  
 **if** node **not in** explored:  
 neighbours = self.G[node]  
 **for** neighbour **in** neighbours:  
 new\_path = list(path)  
 new\_path.append(neighbour)  
 queue.append(new\_path)  
 **if** neighbour == nodesList[1]:  
 self.pathList = [str(i) **for** i **in** new\_path]  
 self.vis()  
 **return**  
explored.append(node)  
 **return**  
 **def** saveGraph(self):  
 f = filedialog.asksaveasfile(initialdir=**"/"**,mode=**'w'**, defaultextension=**".grf"**)  
 **if** f **is None**: **return**  
f.write(str(self.nodesList)+**"\n"**+str(self.edgesList))  
 f.close()  
 **def** loadGraph(self):  
 file = filedialog.askopenfilename(initialdir=**'/'**,filetypes=((**"graph"**, **".grf"**), (**"All Files"**, **"\*.\*"**)))  
 *# inserting data to text editor*  
content = open(file)  
 data = content.readlines()  
 self.G.remove\_edges\_from(self.edgesList)  
 self.G.remove\_nodes\_from(self.nodesList)  
 **for** i **in** data[0]:  
 **if** self.checkNode(i):  
 self.G.add\_node(i)  
 self.nodesList.append(i)  
 **for** i **in** data[1]:  
 **if** self.checkEdge(i):  
 self.G.add\_edge(i[0],i[1])  
 self.edgesList.append(i)  
 self.vis()  
 **def** vis(self):  
 plt.figure()  
 **if** len(self.weightList)==0:  
 nx.draw\_circular(self.G, node\_color=**'red'**, node\_size=1000, with\_labels=**True**)  
 **elif** len(self.pathList)==0:  
 wei=nx.get\_edge\_attributes(self.G,**'weight'**)  
 pos = nx.spring\_layout(self.G)  
 nx.draw(self.G,pos,with\_labels=**True**,font\_weight=**'bold'**)  
 nx.draw\_networkx\_edge\_labels(self.G,pos,edge\_labels=wei)  
 **else**:  
 edges = [(a, b) **for** a, b **in** zip(self.pathList, self.pathList[1:])]  
 weights = nx.get\_edge\_attributes(self.G, **'weight'**)  
 pos = nx.circular\_layout(self.G)  
 nx.draw\_networkx(self.G, pos=pos)  
 nx.draw\_networkx\_edge\_labels(self.G, pos, edge\_labels=weights)  
 nx.draw\_networkx\_edges(self.G, pos=pos, edgelist=edges, edge\_color=**"r"**, width=3)  
 title = **"Shortest path between [{}] and [{}]: {}"** \  
 .format(**"s"**, **"v"**, **" -> "**.join(self.pathList))  
 plt.savefig(**"tmpplot.png"**)  
 image = Image.open(**"tmpplot.png"**)  
 display = ImageTk.PhotoImage(image)  
 self.plot = tk.Label(self,image=display)  
 self.plot.grid(row=4, rowspan=6, column=1,columnspan=6)  
 notes = self.G.nodes(data=**False**)  
 j = 0  
 self.info.insert(0.0,**"a"**)  
 self.info.delete(0.0,tk.END)  
 **for** i **in** notes:  
 nei = [n **for** n **in** self.G.neighbors(i)]  
 self.info.insert(tk.END,**f'neighbors of {**i**} ({**len(nei)**}): {**nei**} \n'**)  
 *#tk.Label(self, text=f"neighbors of {i} ({len(nei)}): {nei}").grid(row=4+j, column=0)*  
j+=1  
 self.info.insert(tk.END,**f"number of nodes: {**len(notes)**}"**)  
 *#tk.Label(self, text=f"number of nodes: {len(notes)}").grid(row=4+len(notes)+1, column=0)*  
**def** checkNode(self,node): **return** node != **'' and** node != **" " and** node **not in** self.nodesList  
 **def** checkEdge(self,edge):  
 **return** len(edge) == 2 **and** (edge **not in** self.edgesList **or** (edge[1], edge[0]) **not in** self.edgesList) **and** \  
 edge[0] **in** self.nodesList **and** edge[1] **in** self.nodesList  
 **def** checkWEdge(self,edge):  
 **return** ((edge[0],edge[1]) **in** self.edgesList **or** (edge[1], edge[0]) **in** self.edgesList)  
 **def** addNodes(self):  
 nodesList = self.addNodesBox.get().split(**" "**)  
 self.addNodesBox.delete(0,**"end"**)  
 **for** i **in** nodesList[:]:  
 **if** self.checkNode(i):self.nodesList.append(i)  
 **else**: nodesList.remove(i)  
 **if** len(self.nodesList)>0:  
 self.G.add\_nodes\_from(nodesList)  
 self.vis()  
 **def** addEdges(self):  
 edgesList = self.addEdgesBox.get().split(**" "**)  
 self.addEdgesBox.delete(0, **"end"**)  
 edgeList = []  
 **for** i **in** edgesList[:]:  
 edge = tuple(i.split(**","**))  
 **if** self.checkEdge(edge):  
 self.edgesList.append(edge)  
 edgeList.append(edge)  
 **else**: edgesList.remove(i)  
 **if** len(edgeList) > 0:  
 self.G.add\_edges\_from(edgeList)  
 self.vis()  
 **def** addWeight(self):  
 weightsList=self.addWeightBox.get().split(**" "**)  
 self.addWeightBox.delete(0,**"end"**)  
 weightList=[]  
 **for** i **in** weightsList[:]:  
 edge = tuple(i.split(**","**))  
 **if** (edge[0],edge[1],float(edge[2])) **not in** self.weightList **and** \  
 (edge[1], edge[0], float(edge[2])) **not in** self.weightList:  
 **if** self.checkEdge((edge[0],edge[1])):  
 self.weightList.append((edge[0],edge[1],float(edge[2])))  
 weightList.append((edge[0],edge[1],float(edge[2])))  
 self.edgesList.append((edge[0],edge[1]))  
 **elif** self.checkWEdge(edge):  
 self.removeEdge((edge[0],edge[1]))  
 self.weightList.append((edge[0],edge[1],float(edge[2])))  
 weightList.append((edge[0],edge[1],float(edge[2])))  
 self.edgesList.append((edge[0],edge[1]))  
 **if** len(weightList) > 0:  
 self.G.add\_weighted\_edges\_from(weightList)  
 self.vis()  
 **def** removeNode(self,nodeIndex):  
 self.G.remove\_node(nodeIndex)  
 self.vis()  
 **def** removeNodes(self):  
 nodesList = self.removeNodesBox.get().split(**" "**)  
 self.removeNodesBox.delete(0, **"end"**)  
 **for** i **in** nodesList[:]:  
 **if** i == **'' or** i == **" " or** i **not in** self.nodesList: nodesList.remove(i)  
 **else**: self.nodesList.remove(i)  
 **if** len(nodesList)>0:  
 self.G.remove\_nodes\_from(nodesList)  
 self.vis()  
 **def** removeEdge(self, edge):  
 **if** edge **in** self.edgesList: self.edgesList.remove(edge)  
 **else**: self.edgesList.remove((edge[1], edge[0]))  
 self.G.remove\_edge(edge[0],edge[1])  
 **def** removeEdges(self):  
 edgesList = self.removeEdgesBox.get().split(**" "**)  
 self.addEdgesBox.delete(0, **"end"**)  
 edgeList = []  
 **for** i **in** edgesList[:]:  
 edge = tuple(i.split(**","**))  
 **if** edge **not in** self.edgesList **and** (edge[1], edge[0]) **not in** self.edgesList: edgesList.remove(i)  
 **elif** edge[0] **not in** self.nodesList **or** edge[1] **not in** self.nodesList: edgesList.remove(i)  
 **else**:  
 **if** edge **in** self.edgesList:  
 self.edgesList.remove(edge)  
 **else**:  
 self.edgesList.remove((edge[1], edge[0]))  
 edgeList.append(edge)  
  
 **if** len(edgeList) > 0:  
 self.G.remove\_edges\_from(edgeList)  
 self.vis()  
 **def** removeWeights(self):  
 edgesList = self.removeWeightsBox.get().split(**" "**)  
 self.addEdgesBox.delete(0, **"end"**)  
 edgeList = []  
 print(self.edgesList,self.weightList)  
  
 **for** i **in** edgesList[:]:  
 edge = tuple(i.split(**","**))  
 flag=**False**  
 **for** j **in** self.weightList[:]:  
 **if** edge==(j[0],j[1]) **or** edge==(j[1],j[0]):  
 self.weightList.remove(j)  
 flag=**True**  
 **if** flag:  
 **if** (edge[0],edge[1]) **in** self.edgesList:  
 self.edgesList.remove((edge[0],edge[1]))  
 **else**:  
 self.edgesList.remove((edge[1],edge[0]))  
 edgeList.append(edge)  
  
 **if** len(edgeList) > 0:  
 self.G.remove\_edges\_from(edgeList)  
 self.vis()  
  
**if** \_\_name\_\_==**'\_\_main\_\_'**:  
 app = Programm()  
 app.mainloop()

Примеры работы:

1. Пример добавления и исключения узлов показан на рисунках 1-4;

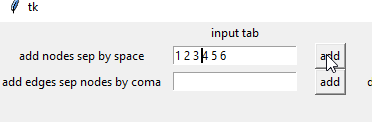


Рисунок 1 — Ввод данных

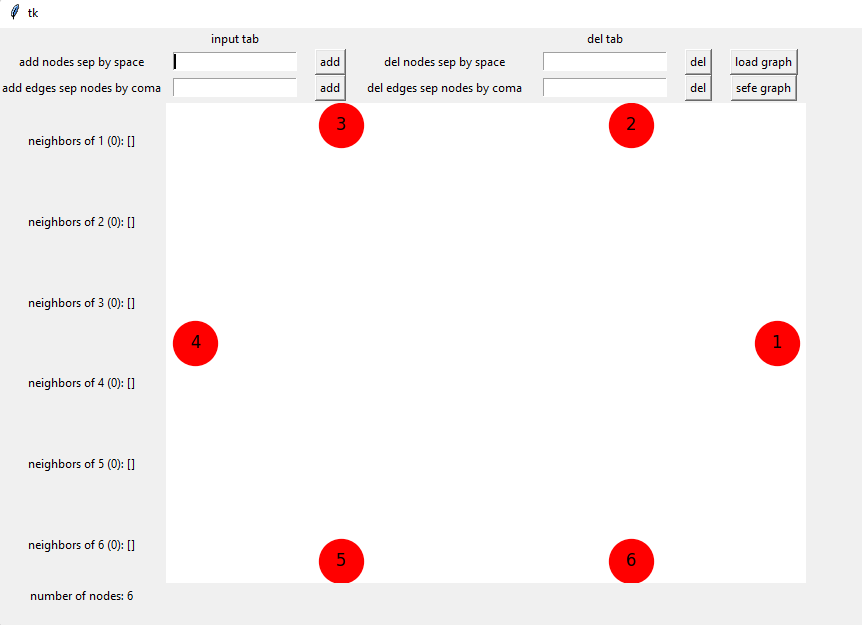


Рисунок 2 — отображение данных

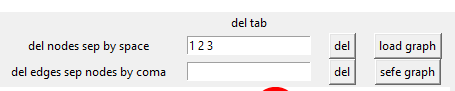


Рисунок 3 - Ввод данных

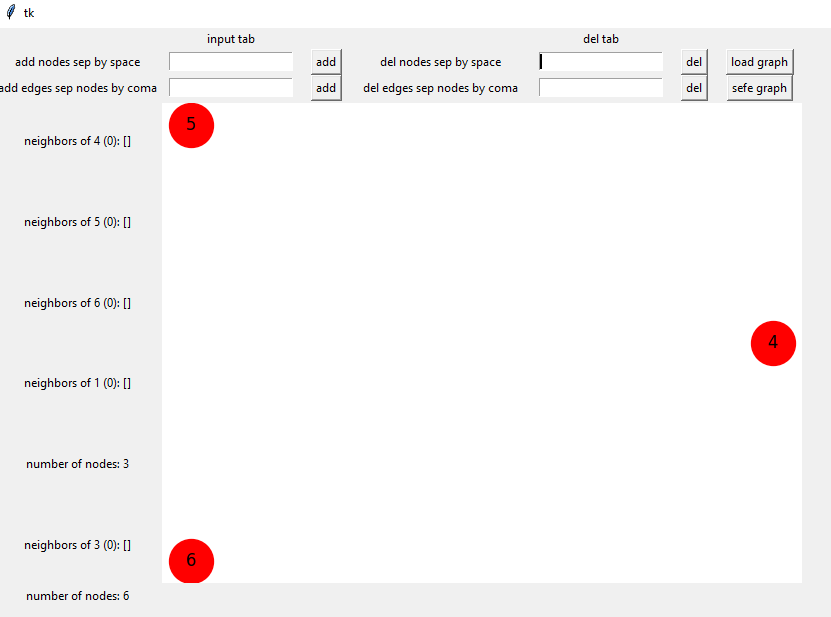


Рисунок 4 — Отображение данных

1. Пример добавления и исключения ребер представлен на рисунках 5 - 8;

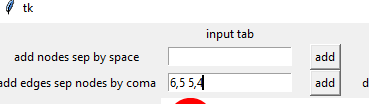


Рисунок 5 — Ввод данных

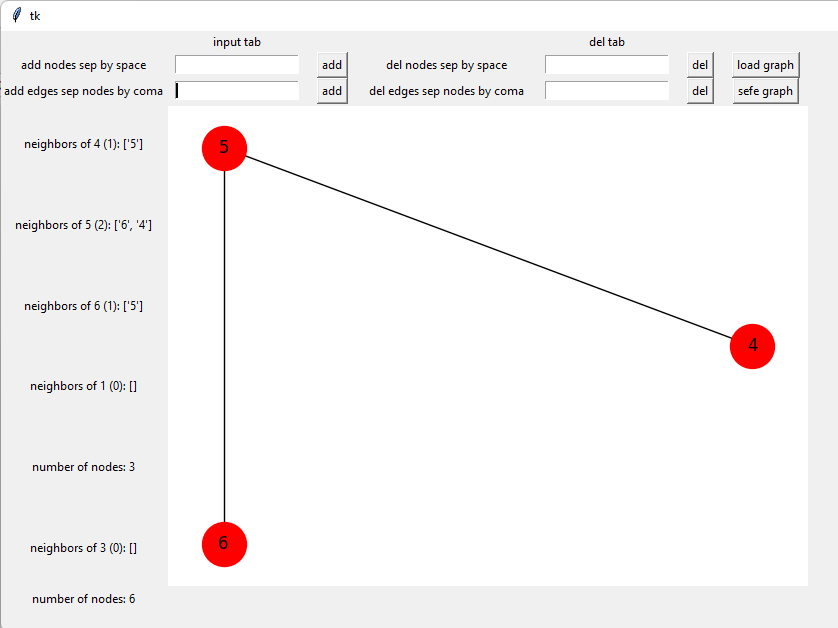


Рисунок 6 — Вывод данных

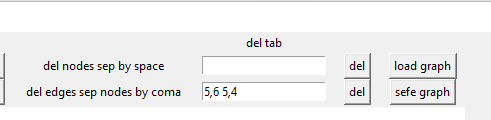


Рисунок 7 — Ввод даных

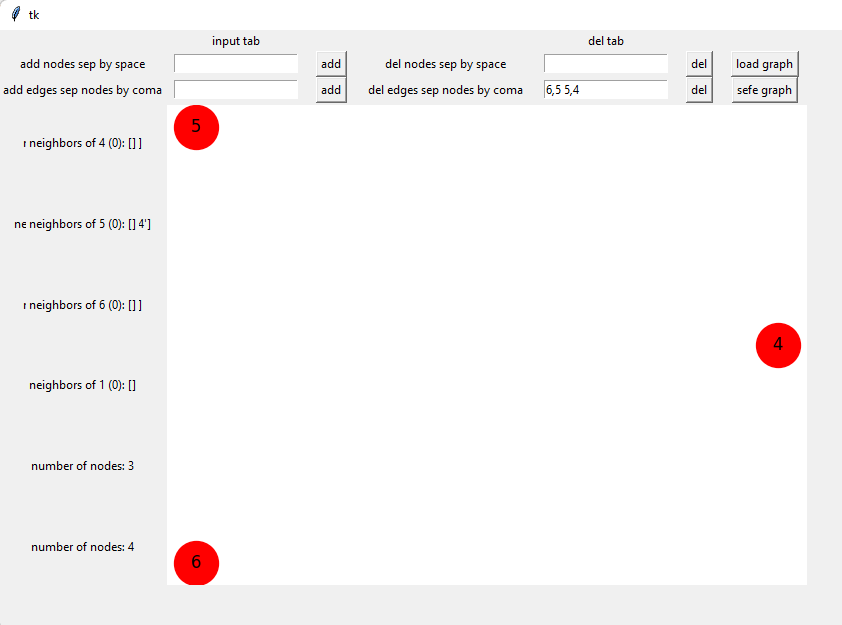


Рисунок 8 — Вывод данных

1. Сохранение в файл графа представлено на рисунке 9;



Рисунок 9 — Пример сохранённого графа

1. Сохранение и загрузка графа представлена на рисунках 10-11;

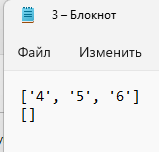


Рисунок 10 — Сохранённый граф

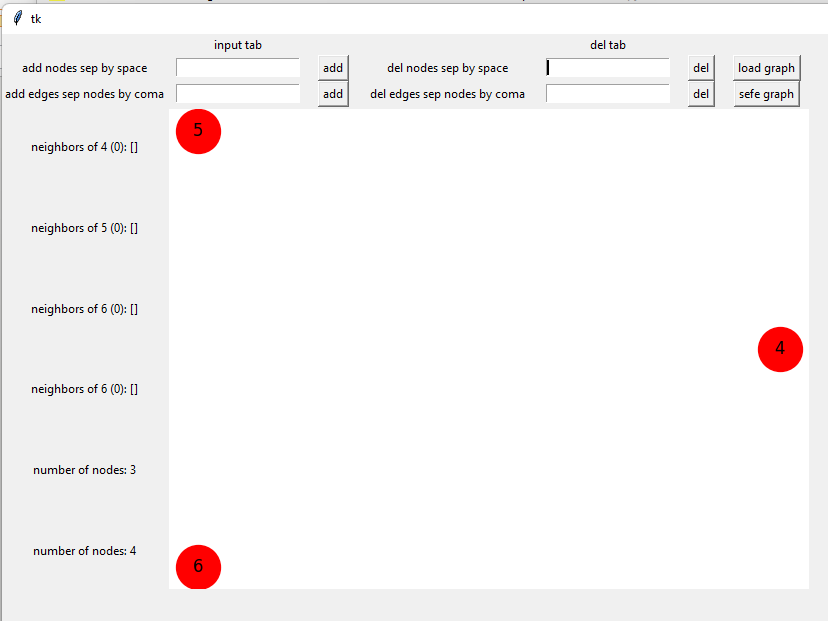


Рисунок 10 — Загруженный граф

1. Взвешенный граф представлен на рисунке 11

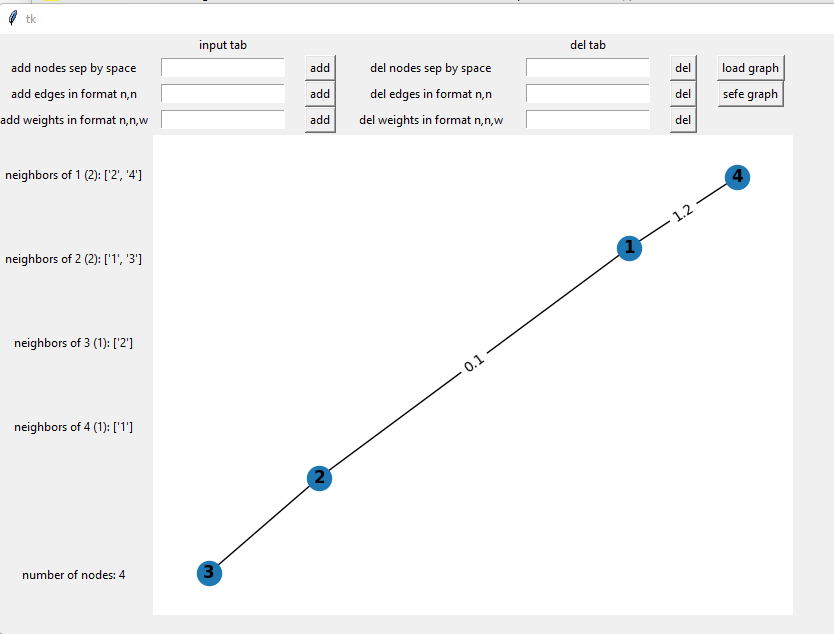


Рисунок 11 — Взвешенный граф

1. Удаление взвешенной грани представлено на рисунке 12

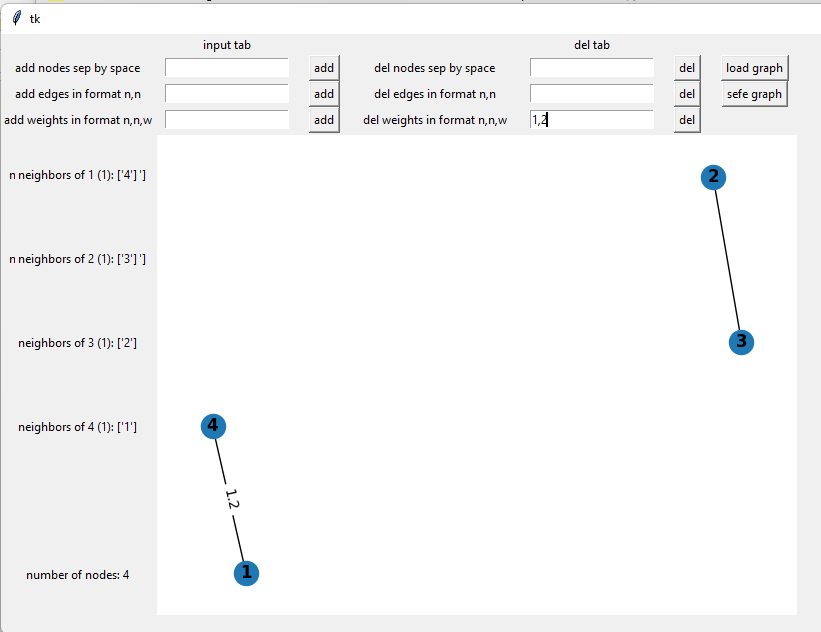


Рисунок 12 — Пример удаления грани из взвешенного графа

7. Поиск кратчайшего пути представлен на рисунке 13.

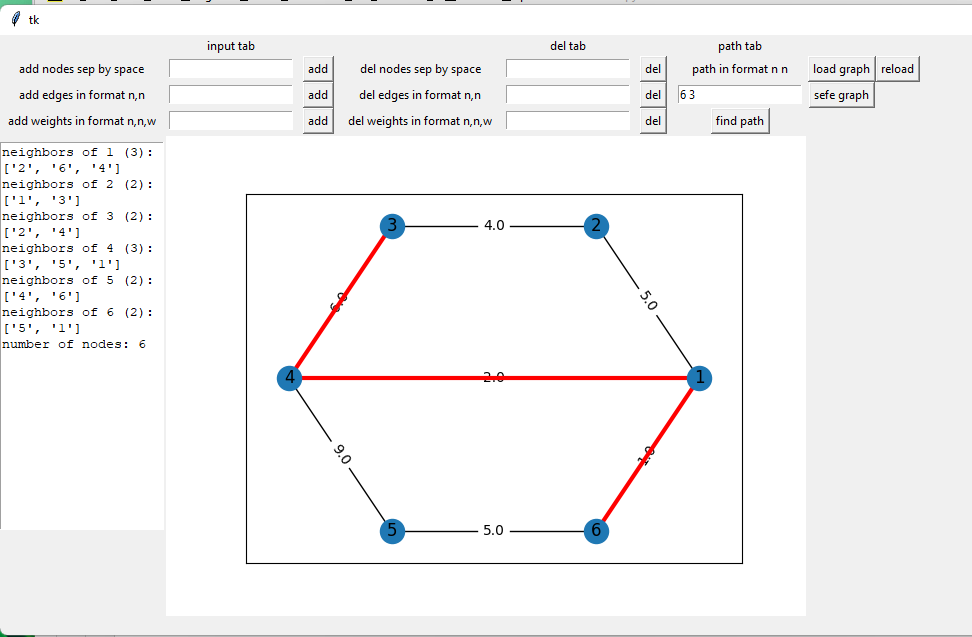


Рисунок 13 — Поиск кратчайшего пути

Поиск кратчайшего пути по алгоритму BFS представлен на рисунке 14

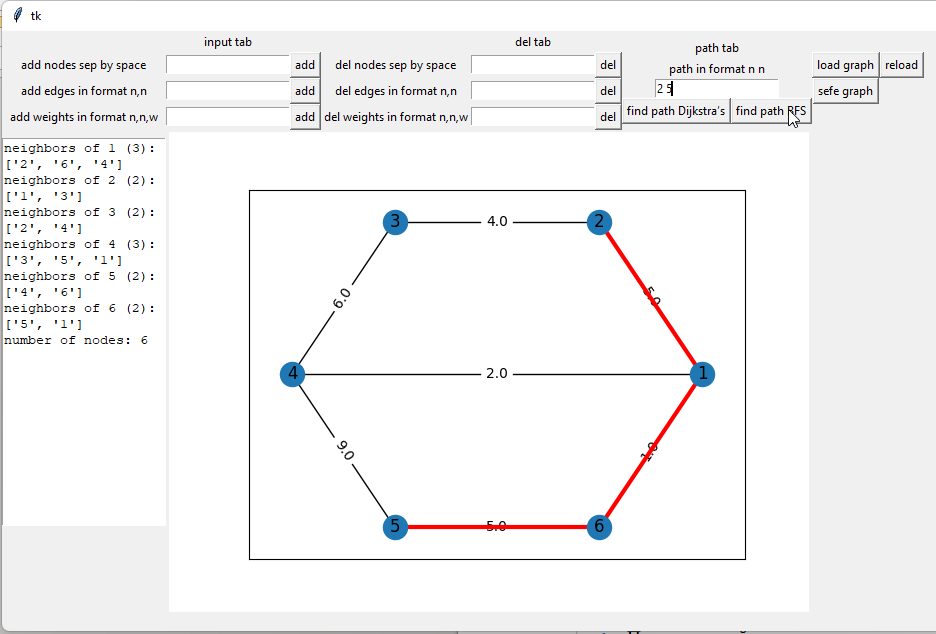


Рисунок 14 — Поиск кратчайшего пути по алгоритму BFS.

**Вопросы**

1. Приведите фрагмент телекоммуникационной (компьютерной, мультисервисной, корпоративной) сети и соответствующий ей граф.



Рисунок 11 — Фрагмент компьютерной сети дома

1. Дайте определение графа с использованием описания его в терминах теории множеств. Пример.

Граф - это графическое представление набора объектов, в котором некоторые пары объектов соединены ссылками.

1. Смежные вершины - определение. Покажите на графе смежные вершины.

Две вершины графа, соединенные ребром, называются **смежными.**

1. Кратные ребра – определения. Пример.

Кратные рёбра (также называемые параллельными рёбрами или мультирёбрами) — это **два и более рёбер, инцидентных одним и тем же двум вершинам**.

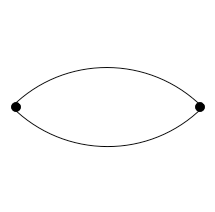


Рисунок 12 — Пример кратных рёбер

1. Петля на графе. Пример.

**Пе́тля́** в графе — ребро, инцидентное одной и той же вершине.

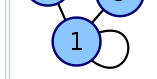


Рисунок 13 — Пример петли

1. Степень вершины – определение. Пример.

**Степень** (**валентность**) **вершины** графа — количество рёбер графа G {\displaystyle G} , инцидентных вершине x.

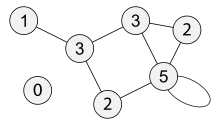


Рисунок 14 — пример степеней вершин

1. Как связаны между собой сумма степеней вершин всех ребер графа и число ветвей (ребер) графа. Покажите на примере.

сумма степеней вершин любого графа равна удвоенному числу его рёбер.

1. Какой граф называется простым? Пример.

Простой граф – конечный граф без петель и кратных ребер.

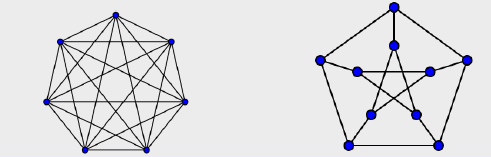


Рисунок 15 — Примеры простого графа

1. Что такое инцидентность? Пример.

Вершины, соединяемые ребром, называются **инцидентными** этому ребру.

1. Что такое висячая вершина? Лист. Пример.

Вершина степени 0 называется изолированной, степени 1 – висячей.

1. Что такое подграф? Пример.

**Подграфом** G’=(V’,U’) исходного графа G=(V,U) называется граф, такой что множества его вершин и дуг являются подмножествами исходного графа

1. Что такое пустой граф? Пример.

Пустой граф ( дерево) - это граф с пустым множеством вершин.

1. Что такое маршрут? Как его построить?

Маршрут в графе — это чередующаяся последовательность вершин и рёбер в которой любые два соседних элемента инцидентны.

1. Что такое цепь? Простая цепь? Примеры.

Маршрут, в котором все рёбра попарно различны, называется *цепью*

1. Что такое путь? Простой путь? Примеры.

Путь — последовательность рёбер (в неориентированном графе) и/или дуг (в ориентированном графе), такая, что конец одной дуги (ребра) является началом другой дуги (ребра).

1. Что такое изоморфизм? Примеры.

Два графа называются изоморфными, если у  
них одинаковое число вершин и вершины каждого из них соединены ребром тогда и только тогда, когда эти вершины во втором графе соединены

1. Какой граф называется планарным? Плоским? Примеры.

**Плана́рный граф** — граф, который можно изобразить на плоскости без пересечений рёбер не по вершинам. Какое-либо конкретное изображение планарного графа на плоскости без пересечения рёбер не по вершинам называется **плоским графом**.

1. Что такое цикл? Примеры.

Замкнутый маршрут, являющийся цепью, называется *циклом*.

1. Что такое мост? Пример.

Мост — **ребро в теории графов, удаление которого увеличивает число компонент связности**.

1. Что такое точка сочленения? Примеры.

**Точка сочленения** графа *G* — вершина, при удалении которой в *G* увеличивается число компонент связности.

1. Что такое дерево? Примеры.

Дерево – связный граф без циклов.

1. Остовное дерево. Пример.

**Остовное дерево** (англ. *spanning tree*) графа *G*=(*V*,*E*) — ациклический связный подграф данного связного неориентированного графа, в который входят все его вершины.

1. Взвешенный граф. Пример.

граф, содержащий взвешенные рёбра, взвешенным.

Вопросы к использованию модуля *NetworkX*

1. Как подключитьмодуль  *NetworkX?*

Pip install networkx

1. Как создать пустой граф?

*G = nx.Graph()*

1. Как добавить в существующий граф один узел?

*G.add\_node(1).*

1. Как добавить в существующий граф несколько узлов?

*G.add\_nodes\_from([2, 3])*

1. Как добавить ребро в существующий граф?

*G.add\_edge(1, 2)*

1. Как добавить несколько ребер в существующий граф?

*G.add\_edges\_from([(1, 2), (1, 3)])*

1. Как определить количество узлов (вершин) и ребер (дуг) в графе?

*G.nodes(),G.edges()*

1. Как просмотреть узлы-соседей для заданного узла? Пример.

*G.neighbors('C')*

1. Как просмотреть все ветви графа? Пример. : *G.edges()*