**Задание 4**

import networkx as nx # импорт библиотеки для работы с графами

import matplotlib.pyplot as plt # импорт библиотеки для визуализации графов

class Graph: # создание класса Graph

def \_\_init\_\_(self, vertices, edges): # определение конструктора класса с параметрами вершин и ребер

self.vertices = vertices # инициализация атрибута vertices

self.edges = edges # инициализация атрибута edges

self.graph = nx.Graph() # создание пустого графа

self.graph.add\_nodes\_from(vertices) # добавление вершин в граф

self.graph.add\_edges\_from(edges) # добавление ребер в граф

def dfs(self, start\_vertex): # определение метода для обхода в глубину с параметром начальной вершины

visited = {v: False for v in self.vertices} # создание словаря для отслеживания посещенных вершин

stack = [start\_vertex] # создание стека с начальной вершиной

while stack: # запуск цикла, пока стек не пуст

current\_vertex = stack.pop() # извлечение вершины из стека

if not visited[current\_vertex]: # проверка, не посещена ли вершина

print(current\_vertex, end=' ') # вывод текущей вершины

visited[current\_vertex] = True # отметка вершины как посещенной

for neighbor in self.graph.neighbors(current\_vertex): # цикл по соседям текущей вершины

if not visited[neighbor]: # проверка, не посещен ли сосед

stack.append(neighbor) # добавление соседа в стек

# Пример использования

vertices = [1, 2, 3, 4, 5,6,8,7,9] # список вершин

edges = [(1, 2), (1, 3),(1,9), (2, 4), (3, 4), (4, 5),(5,6),(5,7),(6,8)] # список ребер

g = Graph(vertices, edges) # создание объекта графа

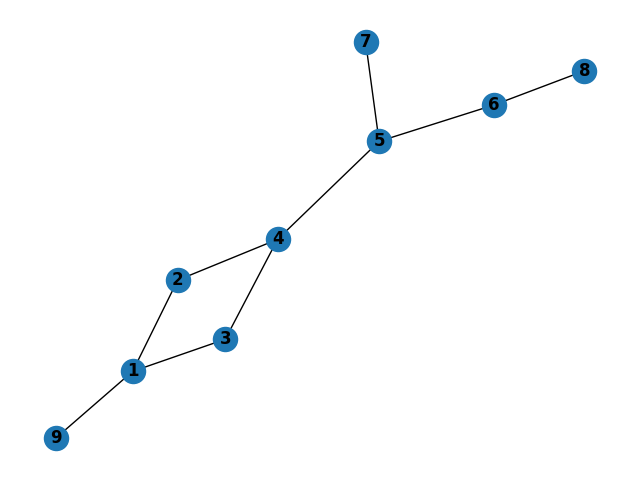
print("DFS: ")

g.dfs(9) # начинаем обход с вершины 9

# Визуализация графа

nx.draw(g.graph, with\_labels=True, font\_weight='bold') # визуализация графа с метками и жирным шрифтом

plt.show() # отображение визуализации графа



import networkx as nx # Импорт библиотеки networkx под псевдонимом nx

import matplotlib.pyplot as plt # Импорт библиотеки matplotlib.pyplot под псевдонимом plt

def dfs(graph, start, visited=None): # Объявление функции dfs с параметрами graph, start, visited

if visited is None: # Проверка, если visited не определен, то

visited = set() # Создание пустого множества visited

visited.add(start) # Добавление узла start в множество visited

print(start, end=' ') # Вывод узла start на экран с пробелом в конце

for next in graph[start] - visited: # Цикл по всем соседним узлам узла start, которые не находятся в множестве visited

dfs(graph, next, visited) # Рекурсивный вызов функции dfs для каждого соседнего узла

return visited # Возврат множества visited

# Пример графа в виде словаря списками соседей

graph = {

'1': {'2', '3'},

'2': {'7', '4', '5'},

'3': {'1', '6'},

'4': {'6'},

'5': {'1', '3'},

'6': {'2', '5'},

'7': {'2', '1'}

}

# Создание графа с помощью библиотеки networkx

G = nx.Graph(graph) # Создание графа G с помощью библиотеки networkx и передача ему словаря graph

# Визуализация графа с помощью библиотеки matplotlib.pyplot

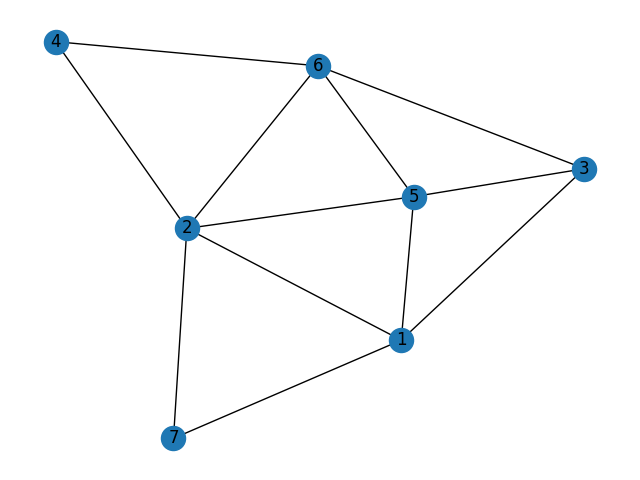
nx.draw(G, with\_labels=True) # Визуализация графа G с помощью функции draw из библиотеки networkx с отображением меток узлов

plt.show() # Отображение графа с помощью библиотеки matplotlib.pyplot

# Вызов метода DFS

print("DFS: ")

dfs(graph, '1') # Вызов метода dfs для обхода графа, начиная с узла '1'



**BFS**

import networkx as nx # импорт библиотеки networkx под псевдонимом nx.

import matplotlib.pyplot as plt # импорт модуля pyplot из библиотеки matplotlib под псевдонимом plt.

def bfs(graph, start\_node): # определение функции bfs с параметрами graph и start\_node.

visited = set() # создание пустого множества visited для отслеживания посещенных узлов.

queue = [start\_node] # создание очереди queue с начальным узлом start\_node.

while queue: # начало цикла, который выполняется, пока очередь не пуста.

node = queue.pop(0) # извлечение первого элемента из очереди.

if node not in visited: # проверка, был ли узел посещен.

print(node, end=' ') # вывод узла на экран.

visited.add(node) # добавление узла в множество посещенных.

for neighbor in graph[node]: # перебор всех соседних узлов текущего узла.

if neighbor not in visited: # проверка, был ли соседний узел посещен.

queue.append(neighbor) # добавление соседнего узла в очередь.

G = nx.Graph() # создание пустого графа.

G.add\_nodes\_from([1, 2, 3, 4, 5]) # добавление узлов в граф.

G.add\_edges\_from([(1, 2),(1,9), (2, 4), (3, 4),(5,6),(5,7),(6,8), (9,9), (10,2), (7,10), (8,10)]) # добавление ребер между узлами в граф.

pos = nx.spring\_layout(G) # расчет координат узлов для визуализации.

nx.draw(G, pos, with\_labels=True, node\_color='lightblue', node\_size=1500, font\_weight='bold', font\_size=15, edge\_color='gray') # визуализация графа с помощью библиотеки networkx.

plt.show() # отображение визуализации графа.

start\_node = 1 # задание начального узла для поиска в ширину.

print("Результат обхода в ширину:") # вывод сообщения о начале обхода в ширину.

bfs(G, start\_node) # вызов функции bfs для графа G с начальным узлом start\_node.

