Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительной техники»

**ОТЧЕТ**

по лабораторной работе №9

по курсу «Логика и основы алгоритмизации в инженерных задачах.»

на тему: «Поиск расстояний в графе.»

Выполнили**:**

студенты группы 22ВВВ2

Коробкин В. Ю.

Горбатов К. В.

Проверили:

Митрохин М. А.

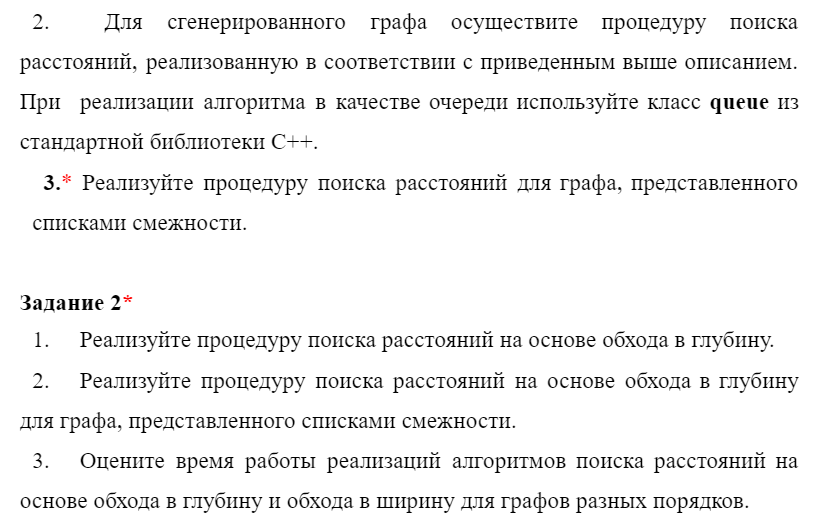
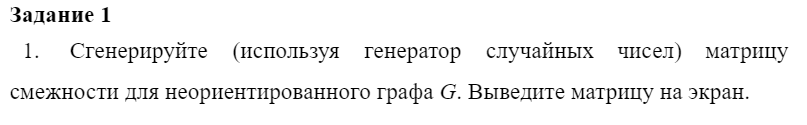
Акифьев И.В.

Пенза 2023

**Цель работы**

Научиться реализовывать алгоритм поиска расстояний в графе.

**Лабораторное задание**

****

**Листинг**

Программа 9.1 log – Задание 1 (1,2)

from collections import deque

import random

import time

def generate\_random\_adjacency\_matrix(n, p):

adjacency\_matrix = [[0] \* n for \_ in range(n)]

for i in range(n):

for j in range(i + 1, n):

random\_value = random.random()

if random\_value <= p:

adjacency\_matrix[i][j] = 1

adjacency\_matrix[j][i] = 1

return adjacency\_matrix

def bfs(adj\_matrix, start):

visited = [False] \* len(adj\_matrix)

distances = [-1] \* len(adj\_matrix)

queue = deque()

queue.append(start)

visited[start] = True

distances[start] = 0

while queue:

current\_vertex = queue.popleft()

for neighbor, is\_connected in enumerate(adj\_matrix[current\_vertex]):

if is\_connected and not visited[neighbor]:

queue.append(neighbor)

visited[neighbor] = True

distances[neighbor] = distances[current\_vertex] + 1

return distances

p = 0.5

while True:

print("Введите -1 если хотите завершить программу ")

n = int(input("Введите размер матрицы смежности: "))

if n == -1:

break

adjacency\_matrix = generate\_random\_adjacency\_matrix(n, p)

print("Матрица смежности:")

for row in adjacency\_matrix:

print(row)

start\_vertex = int(input("Введите стартовую вершину для поиска расстояний: "))

start\_time = time.time()

distances = bfs(adjacency\_matrix, start\_vertex)

end\_time = time.time()

print(f"Расстояния от вершины {start\_vertex}:")

for vertex, distance in enumerate(distances):

print(f"До вершины {vertex}: {distance}")

print(f"Время выполнения: {end\_time - start\_time} секунд")

Программа9.2 log – Задание 1 (3)

from collections import deque

import random

def generate\_random\_adjacency\_list(n, p):

adjacency\_list = {i: [] for i in range(n)}

for i in range(n):

for j in range(i + 1, n):

random\_value = random.random()

if random\_value <= p:

adjacency\_list[i].append(j)

adjacency\_list[j].append(i)

return adjacency\_list

def bfs(adj\_list, start):

visited = [False] \* len(adj\_list)

distances = [-1] \* len(adj\_list)

queue = deque()

queue.append(start)

visited[start] = True

distances[start] = 0

while queue:

current\_vertex = queue.popleft()

for neighbor in adj\_list[current\_vertex]:

if not visited[neighbor]:

queue.append(neighbor)

visited[neighbor] = True

distances[neighbor] = distances[current\_vertex] + 1

return distances

p = 0.5

while True:

print("Введите -1 если хотите завершить программу ")

n = int(input("Введите размер графа (количество вершин): "))

if n == -1:

break

adjacency\_list = generate\_random\_adjacency\_list(n, p)

print("Списки смежности:")

for vertex, neighbors in adjacency\_list.items():

print(f"{vertex}: {neighbors}")

start\_vertex = int(input("Введите стартовую вершину для поиска расстояний: "))

distances = bfs(adjacency\_list, start\_vertex)

print(f"Расстояния от вершины {start\_vertex}:")

for vertex, distance in enumerate(distances):

print(f"До вершины {vertex}: {distance}")

Программа 9.3 log – Задание 2 (1)

from collections import deque

import random

import time

def generate\_random\_adjacency\_matrix(n, p):

adjacency\_matrix = [[0] \* n for \_ in range(n)]

for i in range(n):

for j in range(i + 1, n):

random\_value = random.random()

if random\_value <= p:

adjacency\_matrix[i][j] = 1

adjacency\_matrix[j][i] = 1

return adjacency\_matrix

def dfs(adj\_matrix, start):

visited = [False] \* len(adj\_matrix)

distances = [-1] \* len(adj\_matrix)

def dfs\_recursive(vertex, distance):

visited[vertex] = True

distances[vertex] = distance

for neighbor, is\_connected in enumerate(adj\_matrix[vertex]):

if is\_connected and not visited[neighbor]:

dfs\_recursive(neighbor, distance + 1)

dfs\_recursive(start, 0)

return distances

p = 0.5

while True:

print("Введите -1 если хотите завершить программу ")

n = int(input("Введите размер матрицы смежности: "))

if n == -1:

break

adjacency\_matrix = generate\_random\_adjacency\_matrix(n, p)

print("Матрица смежности:")

for row in adjacency\_matrix:

print(row)

start\_vertex = int(input("Введите стартовую вершину для поиска расстояний: "))

start\_time = time.time()

distances\_dfs = dfs(adjacency\_matrix, start\_vertex)

end\_time = time.time()

print(f"Расстояния DFS от вершины {start\_vertex}:")

for vertex, distance in enumerate(distances\_dfs):

print(f"До вершины {vertex}: {distance}")

print(f"Время выполнения DFS: {end\_time - start\_time} секунд")

Программа 9.4 log – Задание 2 (2)

from collections import deque

import random

def generate\_random\_adjacency\_list(n, p):

adjacency\_list = {i: [] for i in range(n)}

for i in range(n):

for j in range(i + 1, n):

random\_value = random.random()

if random\_value <= p:

adjacency\_list[i].append(j)

adjacency\_list[j].append(i)

return adjacency\_list

def dfs(adj\_list, start):

visited = [False] \* len(adj\_list)

distances = [-1] \* len(adj\_list)

def dfs\_recursive(vertex, distance):

visited[vertex] = True

distances[vertex] = distance

for neighbor in adj\_list[vertex]:

if not visited[neighbor]:

dfs\_recursive(neighbor, distance + 1) # Увеличиваем расстояние на 1 при переходе к соседней вершине

dfs\_recursive(start, 0)

return distances

p = 0.5

while True:

print("Введите -1 если хотите завершить программу ")

n = int(input("Введите размер графа (количество вершин): "))

if n == -1:

break

adjacency\_list = generate\_random\_adjacency\_list(n, p)

print("Списки смежности:")

for vertex, neighbors in adjacency\_list.items():

print(f"{vertex}: {neighbors}")

start\_vertex = int(input("Введите стартовую вершину для поиска расстояний: "))

distances = dfs(adjacency\_list, start\_vertex)

print(f"Расстояния DFS от вершины {start\_vertex}:")

for vertex, distance in enumerate(distances):

print(f"До вершины {vertex}: {distance}")

**Результаты выполнения программ:**

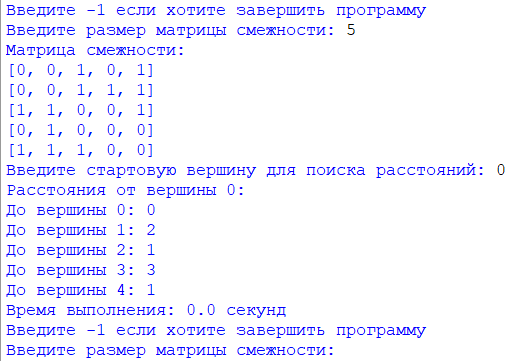


Рисунок 1. – Результат выполнения программы 9.1 log.py

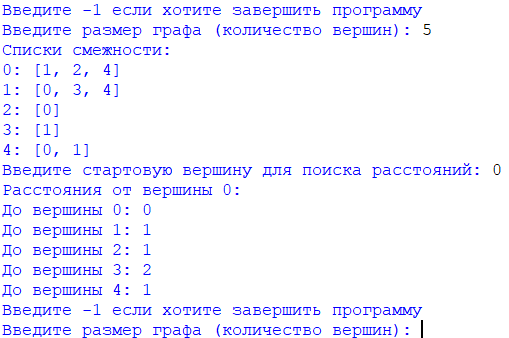


Рисунок 2. – Результат выполнения программы 9.2 log.py

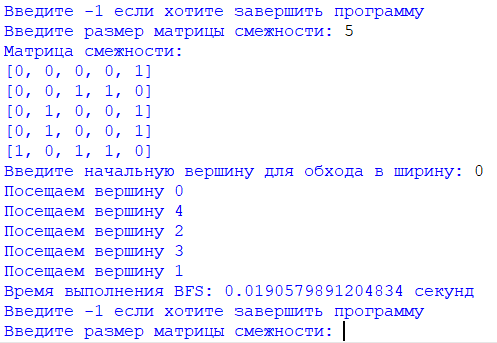


Рисунок 3. – Результат выполнения программы 9.3 log.py

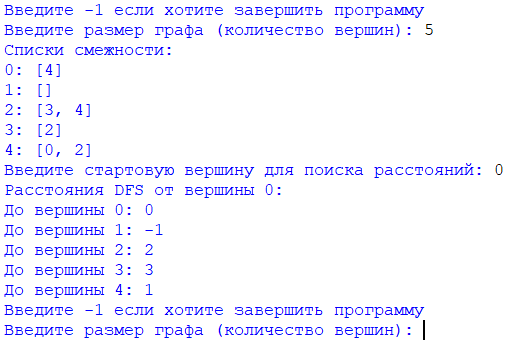


Рисунок 4. – Результат выполнения программы 9.4 log.py

**Задание 2 (3)**

Оценка времени работы реализаций алгоритмов поиска расстояний на основе обхода в глубину и обхода в ширину для графов разных порядков производилась с помощью функции time в программах 9.1 log.py и 9.3 log.py.

Зависимость времени работы алгоритма от размера графа:

***Алгоритм поиска расстояний на основе обхода в ширину****:*

100х100 - 0.001 секунд

500x500 - 0.006 секунд

1000x1000 - 0.027 секунд

Матрицы большего размера генерации не подлежат

***Алгоритм поиска расстояний на основе обхода в глубину****:*

100х100 – 0.001 секунд

500x500 - 0.008 секунд

1000x1000 - 0.046 секунд

Матрицы большего размера генерации не подлежат

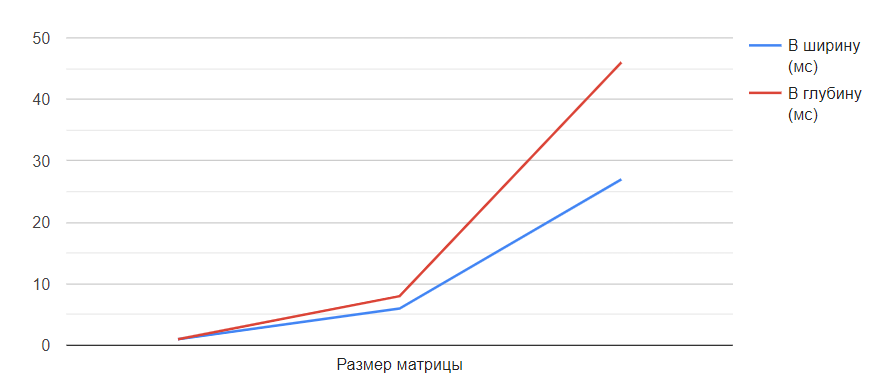


Рисунок 4. – Графики зависимости времени работы функций обхода от размера матриц

**Из этих данных можно понять, что:**

Алгоритм поиска расстояний на основе обхода в ширину, справляется быстрее, нежели алгоритм поиска расстояний на основе обхода в глубину.

Вывод: в ходе выполнения лабораторной работы были получены знания реализации алгоритма поиска расстояний в графе, а также оценки их работы.