Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительной техники»

**ОТЧЕТ**

по лабораторной работе №7

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

на тему: «ОБХОД ГРАФА В ГЛУБИНУ.»

Выполнили**:**

студенты группы 22ВВВ2

Коробкин В. Ю.

Горбатов К. В.

Проверили:

Митрохин М. А.,

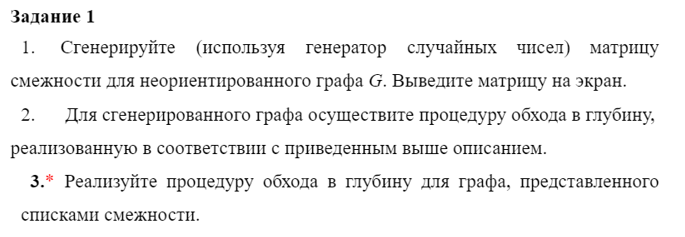
Акифьев И. В.

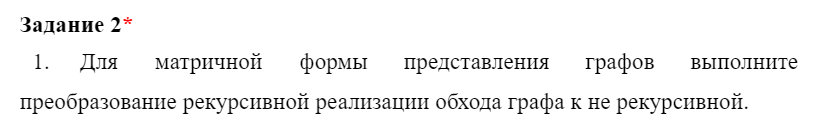
Пенза 2023

**Цель работы**

Научиться реализовывать алгоритм поиска в глубину.

**Лабораторное задание**





Листинг

7.1 log.py – Задание 1 (1 и 2 пункт)

import random

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):

# Генерируем случайное число в диапазоне [0, 1]

random\_value = random.random()

# Если случайное число меньше или равно p, то устанавливаем ребро

if random\_value <= p:

adjacency\_matrix[i][j] = 1

adjacency\_matrix[j][i] = 1

return adjacency\_matrix

def dfs(graph, visited, vertex):

if not visited[vertex]:

print("Посещаем вершину:", vertex)

visited[vertex] = True

for neighbor, has\_edge in enumerate(graph[vertex]):

if has\_edge and not visited[neighbor]:

dfs(graph, visited, neighbor)

#вероятность ребра #####################################

p = 0.5 #####################################

while True:

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

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

if n == -1:

break

# Генерируем матрицу смежности

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

# Определяем размер графа

edge\_count = sum(sum(row) for row in adjacency\_matrix) // 2

# Определяем количество вершин

graph\_size = len(adjacency\_matrix)

# Выводим информацию о графе

print("Матрица смежности:")

for row in adjacency\_matrix:

print(row)

# Инициализация массива посещенных вершин

visited = [False] \* graph\_size

# Начало обхода в глубину с первой вершины

start\_vertex = 0

dfs(adjacency\_matrix, visited, start\_vertex)

7.2 log.py – Задание 1 (3 пункт)

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):

# Генерируем случайное число в диапазоне [0, 1]

random\_value = random.random()

# Если случайное число меньше или равно p, то добавляем ребро

if random\_value <= p:

adjacency\_list[i].append(j)

adjacency\_list[j].append(i)

return adjacency\_list

def dfs(graph, visited, vertex):

if not visited[vertex]:

print("Посещаем вершину:", vertex)

visited[vertex] = True

for neighbor in graph[vertex]:

if not visited[neighbor]:

dfs(graph, visited, neighbor)

p = 0.5

while True:

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

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

if n == -1:

break

# Генерируем список смежности

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

# Определяем размер графа

edge\_count = sum(len(neighbors) for neighbors in adjacency\_list.values()) // 2

# Определяем количество вершин

graph\_size = len(adjacency\_list)

# Выводим информацию о графе

print("Список смежности:")

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

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

# Инициализация массива посещенных вершин

visited = {vertex: False for vertex in range(graph\_size)}

# Начало обхода в глубину с первой вершины

start\_vertex = 0

dfs(adjacency\_list, visited, start\_vertex)

7.3 log.py – Задание 2

import random

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):

# Генерируем случайное число в диапазоне [0, 1]

random\_value = random.random()

# Если случайное число меньше или равно p, то устанавливаем ребро

if random\_value <= p:

adjacency\_matrix[i][j] = 1

adjacency\_matrix[j][i] = 1

return adjacency\_matrix

def dfs\_non\_recursive(graph, start\_vertex):

stack = [start\_vertex]

visited = [False] \* len(graph)

while stack:

vertex = stack.pop()

if not visited[vertex]:

print("Посещаем вершину:", vertex)

visited[vertex] = True

stack.extend(neighbor for neighbor, has\_edge in enumerate(graph[vertex]) if has\_edge and not visited[neighbor])

p = 0.5

while True:

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

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

if n == -1:

break

# Генерируем матрицу смежности

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

# Определяем размер графа

edge\_count = sum(sum(row) for row in adjacency\_matrix) // 2

# Определяем количество вершин

graph\_size = len(adjacency\_matrix)

# Выводим информацию о графе

print("Матрица смежности:")

for row in adjacency\_matrix:

print(row)

# Начало не-рекурсивного обхода в глубину с первой вершины

start\_vertex = 0

dfs\_non\_recursive(adjacency\_matrix, start\_vertex)

Результаты работы кодов программ

7.1 log.py

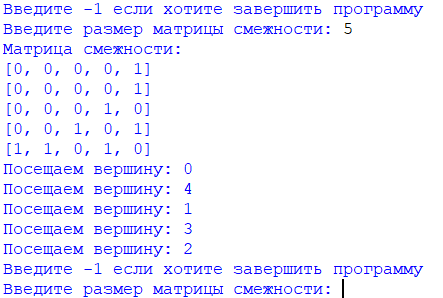


Рисунок 1. – Результат работы первого кода

7.2 log.py

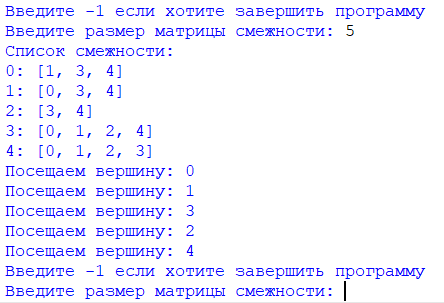


Рисунок 2. – Результат работы второго кода

7.3 log.py

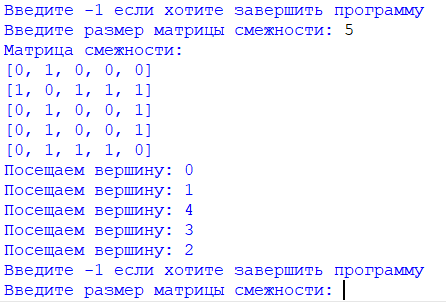


Рисунок 3. – Результат работы третьего кода

Вывод: в ходе выполнения лабораторной работы были получены навыки разработки программы на языке Python, реализующую алгоритм обхода графа в глубину.