Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра “Вычислительная техника”

**Отчет**

по лабораторной работе № 8

по курсу “Логика и основы алгоритмизации в инженерных задачах”

на тему “Обход графа в ширину”

**Выполнил студент группы 22ВВС1:**

Пилюгин А.Э

**Приняли:**

Акифьев И. В.

Юрова О.В

Пенза 2023

**Лабораторное задание:**

### **Задание 1**

1. Сгенерируйте (используя генератор случайных чисел) матрицу смежности для неориентированного графа *G*. Выведите матрицу на экран.
2. Для сгенерированного графа осуществите процедуру обхода в ширину, реализованную в соответствии с приведенным выше описанием. При  реализации алгоритма в качестве очереди используйте класс **queue** из стандартной библиотеки С++.

**3.**\* Реализуйте процедуру обхода в ширину для графа, представленного списками смежности.

### Задание 2\*

1. Для матричной формы представления графов реализуйте алгоритм обхода в ширину с использованием очереди, построенной на основе структуры данных «список», самостоятельно созданной в лабораторной работе № 3.

2. Оцените время работы двух реализаций алгоритмов обхода в ширину (использующего стандартный класс **queue** и использующего очередь, реализованную самостоятельно) для графов разных порядков.

**Задание №1.1.2**

import random  
  
  
  
# Функция для генерации случайной матрицы смежности  
def generate\_random\_adjacency\_matrix(num\_vertices, num\_edges):  
 matrix = [[0 for \_ in range(num\_vertices)] for \_ in range(num\_vertices)]  
  
 edge\_count = 0  
 while edge\_count < num\_edges:  
 vertex1 = random.randint(0, num\_vertices - 1)  
 vertex2 = random.randint(0, num\_vertices - 1)  
  
 if vertex1 != vertex2 and matrix[vertex1][vertex2] == 0:  
 matrix[vertex1][vertex2] = 1  
 matrix[vertex2][vertex1] = 1  
 edge\_count += 1  
  
 return matrix  
  
  
# Функция для обхода в ширину  
def breadth\_first\_search(graph, start\_vertex):  
 visited = [False] \* len(graph)  
 queue = [0] \* len(graph)  
 front, rear = 0, 0  
  
 visited[start\_vertex] = True  
 print(f"Посещенная вершина: {start\_vertex}")  
 queue[rear] = start\_vertex  
 rear += 1  
  
 while front != rear:  
 current\_vertex = queue[front]  
 front += 1  
 for neighbor in range(len(graph)):  
 if graph[current\_vertex][neighbor] == 1 and not visited[neighbor]:  
 visited[neighbor] = True  
 print(f"Посещенная вершина: {neighbor}")  
 queue[rear] = neighbor  
 rear += 1  
  
  
# Главная функция  
def main():  
 random.seed()  
 num\_vertices = 6 # Желаемое количество вершин  
 num\_edges = 8 # Желаемое количество рёбер  
 adjacency\_matrix = generate\_random\_adjacency\_matrix(num\_vertices, num\_edges)  
  
 print("Матрица смежности:")  
 for row in adjacency\_matrix:  
 print(" ".join(map(str, row)))  
  
 start\_vertex = int(input(f"Введите начальную вершину для обхода (от 0 до {num\_vertices - 1}): "))  
 if start\_vertex < 0 or start\_vertex >= num\_vertices:  
 print(f"Недопустимая начальная вершина. Пожалуйста, введите вершину между 0 и {num\_vertices - 1}.")  
 return 1 # Выход с кодом ошибки  
  
 print(f"Обход в ширину, начиная с вершины {start\_vertex}:")  
 breadth\_first\_search(adjacency\_matrix, start\_vertex)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Задание №1.3  
  
# Функция для добавления ребра между вершинами u и v  
def add\_edge(adjacency\_list, u, v):  
 adjacency\_list[u][v] = 1  
 adjacency\_list[v][u] = 1 # Для ненаправленного графа  
  
# Процедура для вывода списка смежности графа  
def print\_adjacency\_list(adjacency\_list, num\_vertices):  
 print("Список смежности графа:")  
 for i in range(num\_vertices):  
 print(f"Вершина {i}: ", end='')  
 for j in range(num\_vertices):  
 if adjacency\_list[i][j] == 1:  
 print(f"{j} ", end='')  
 print()  
  
# Процедура обхода в ширину  
def breadth\_first\_search(adjacency\_list, start\_vertex, num\_vertices):  
 visited = [False] \* num\_vertices  
 queue = []  
 front, rear = 0, 0  
  
 visited[start\_vertex] = True  
 print(f"Посещенная вершина: {start\_vertex}")  
 queue.append(start\_vertex)  
 rear += 1  
  
 while front < rear:  
 current\_vertex = queue[front]  
 front += 1  
 for neighbor in range(num\_vertices):  
 if adjacency\_list[current\_vertex][neighbor] == 1 and not visited[neighbor]:  
 visited[neighbor] = True  
 print(f"Посещенная вершина: {neighbor}")  
 queue.append(neighbor)  
 rear += 1  
  
# Главная функция  
def main():  
 num\_vertices = 6  
 adjacency\_list = [[0 for \_ in range(MAX\_VERTICES)] for \_ in range(MAX\_VERTICES)]  
  
 # Добавление рёбер между вершинами (пример)  
 add\_edge(adjacency\_list, 0, 1)  
 add\_edge(adjacency\_list, 0, 2)  
 add\_edge(adjacency\_list, 1, 3)  
 add\_edge(adjacency\_list, 2, 3)  
 add\_edge(adjacency\_list, 3, 4)  
 add\_edge(adjacency\_list, 4, 5)  
  
 print\_adjacency\_list(adjacency\_list, num\_vertices)  
  
 start\_vertex = int(input(f"Введите начальную вершину для обхода (от 0 до {num\_vertices - 1}): "))  
 if start\_vertex < 0 or start\_vertex >= num\_vertices:  
 print(f"Недопустимая начальная вершина. Пожалуйста, введите вершину между 0 и {num\_vertices - 1}.")  
 return 1 # Выход с кодом ошибки  
  
 print(f"Обход в ширину, начиная с вершины {start\_vertex}:")  
 breadth\_first\_search(adjacency\_list, start\_vertex, num\_vertices)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**Задание 2.**

import random  
import time  
  
class Queue:  
 def \_\_init\_\_(self):  
 self.items = []  
  
 def is\_empty(self):  
 return len(self.items) == 0  
  
 def enqueue(self, item):  
 self.items.append(item)  
  
 def dequeue(self):  
 if not self.is\_empty():  
 return self.items.pop(0)  
  
 def size(self):  
 return len(self.items)  
  
# Генерация случайной матрицы смежности  
def generate\_random\_adjacency\_matrix(num\_vertices):  
 matrix = [[random.randint(0, 1) for \_ in range(num\_vertices)] for \_ in range(num\_vertices)]  
 for i in range(num\_vertices):  
 matrix[i][i] = 0 # На главной диагонали будем всегда иметь нули  
 for j in range(i + 1, num\_vertices):  
 matrix[i][j] = matrix[j][i] # Симметричная матрица для неориентированного графа  
 return matrix  
  
# Обход в ширину для графа на основе матрицы смежности  
def breadth\_first\_search(graph, start):  
 num\_vertices = len(graph)  
 visited = [False] \* num\_vertices  
 queue = Queue()  
  
 visited[start] = True  
 print(f"Посещенная вершина: {start}")  
 queue.enqueue(start)  
  
 start\_time = time.time() # Начало замера времени  
  
 while not queue.is\_empty():  
 current\_vertex = queue.dequeue()  
 for neighbor in range(num\_vertices):  
 if graph[current\_vertex][neighbor] == 1 and not visited[neighbor]:  
 visited[neighbor] = True  
 print(f"Посещенная вершина: {neighbor}")  
 queue.enqueue(neighbor)  
  
 end\_time = time.time() # Окончание замера времени  
 execution\_time = end\_time - start\_time  
 print(f"Время выполнения: {execution\_time} секунд")  
  
# Ввод размерности матрицы  
print('Введите размерность матрицы:')  
num\_vertices = int(input()) # размерность матрицы  
adjacency\_matrix = generate\_random\_adjacency\_matrix(num\_vertices)  
  
# Вывод матрицы смежности на экран  
print("Матрица смежности:")  
for row in adjacency\_matrix:  
 print(' '.join(map(str, row)))  
  
# Выбор вершины для обхода  
print('Выберите вершину для обхода:')  
start\_vertex = int(input()) # Начальная вершина для обхода  
print(f"Обход в ширину, начиная с вершины {start\_vertex}:")  
breadth\_first\_search(adjacency\_matrix, start\_vertex)

**Результаты работы программ**

Изображение выглядит как текст, снимок экрана, программное обеспечение, компьютер

Автоматически созданное описание

**Рисунок 1 - Результат работы программы lab8.12**

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

**Рисунок 2 - Результат работы программы lab8.1.3**

**Оценка времени выполнения.**

**Сравнение времени исполнения (для deque и queue):**

**для 100:**

****

**Для 200:**

****

**Для 500:**

****

**Для 1000:**

****

**Вывод:** Встроенная реализация очереди оказалась быстрее.

**Вывод**

В результате выполнения лабораторной работы были успешно реализованы алгоритмы построения матрицы смежности, усвоены и практически проверены навыки в алгоритме реализации обхода графа в ширину.