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»

Выполнили:
студенты группы 24ВВВ3
Азаров М.С.
Кукушкин А.А.

Приняли:
к.т.н., доцент Юрова О.В.
к.т.н., Деев М.В.

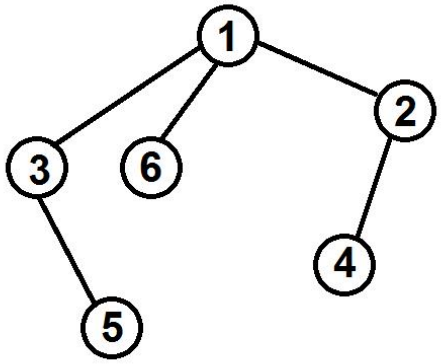
Пенза 2025

Цель работы – Изучение и практическая реализация алгоритма обхода графа в ширину (BFS) на примере различных структур данных для представления графа (матрица смежности, списки смежности).

Общие сведения.

Обход графа в ширину – еще один распространенный способ обхода графов.

Основная идея такого обхода состоит в том, чтобы посещать вершины по уровням удаленности от исходной вершины. Удалённость в данном случае понимается как количество ребер, по которым необходимо прейти до достижения вершины. Например, если для графа на рисунке 1 начать обход из первой вершины, то вершины 3, 6 и 2 будут находиться на уровне удаленности в 1 ребро, а вершины 5 и 4 на уровне удаленности в 2 ребра.



$$G = \begin{Bmatrix} 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{Bmatrix}$$

Рисунок 1 – Граф

Тогда при обходе этого графа в ширину, мы сначала посетим вершины первого уровня удаленности (с номерами 2, 3 и 6), и только после того, как закончатся не посещенные вершины на этом уровне, мы перейдем к следующему. На втором уровне мы посетим все вершины, которые удалены от исходной на 2 ребра (вершины 4 и 5).

Так, алгоритм обхода в ширину продолжает осматривать уровень за уровнем, пока не пройдет все доступные вершины.

Чтобы не заходить повторно в уже пройденные вершины, они помечаются, как и в алгоритме обхода в глубину.

Для того, чтобы проход осуществлялся по уровням необходимо хранить информацию о требуемом порядке посещения вершин. Вершины, которые являются ближайшими соседями исходной вершины (из которой начат обход) должны быть посещены раньше, чем соседи соседей и т.д. Такой порядок позволяет задать структура данных «очередь». Просматривая строку матрицы смежности (или список смежности) для текущей вершины мы помещаем всех её ещё не посещенных соседей в очередь. На следующей итерации текущей вершиной становится та, которая стоит в очереди первой и уже её не посещенные соседи будут помещены в очередь. Но место в очереди они займут после тех вершин, которые были помещены туда на предыдущих итерациях.

Таким образом, можно предложить следующую реализацию алгоритма обхода в ширину.

Вход: G – матрица смежности графа.

Выход: номера вершин в порядке их прохождения на экране.

Алгоритм ПОШ

1.1. для всех i положим NUM[i] = False пометим как "не посещенную";

1.2. **ПОКА** существует "новая" вершина v

1.3. **ВЫПОЛНЯТЬ** BFS (v).

Алгоритм BFS(v):

2.1. Создать пустую очередь Q = {};

2.2. Поместить v в очередь Q.push(v);

2.3. пометить v как "посещенную" NUM[v] = True;

2.4. **ПОКА** Q != ∅ очередь не пуста **ВЫПОЛНЯТЬ**

2.5. v = Q.front() установить текущую вершину;

2.6. Удалить первый элемент из очереди Q.pop();

2.7. вывести на экран v;

2.8. **ДЛЯ** i = 1 **ДО** size_G **ВЫПОЛНЯТЬ**

2.9. **ЕСЛИ** G(v,i) = = 1**И** NUM[i] = = False

2.10. **ТО**

2.11. Поместить i в очередь Q.push(i);

2.12. пометить v как "посещенную" NUM[v] = True;

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 C++.

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.

Код программы

C

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <time.h>
#include <queue>

using namespace std;

void DFS(int** G, int numG, int* visited, int current) {
	visited[current] = 1;
	printf("%3d", current);

	for (int i = 0; i < numG; i++) {
		if (G[current][i] == 1 && visited[i] == 0) {
			DFS(G, numG, visited, i);
		}

	}
}

void BFS(int** G, int numG, int* visited, int s) {
	queue<int> q;
	int v;

	visited[s] = 1;
	q.push(s);

	while (!q.empty()) {

		v = q.front();
		q.pop();
		printf("%3d", v);

		for (int i = 0; i < numG; i++) {
			if (G[v][i] == 1 && visited[i] == 0) {
				q.push(i);
				visited[i] = 1;
			}

		}
	}
}



int main() {
	int** G;
```

```
	int* visited;
	int numG, current;

	printf("Input number of vershini: ");
	scanf("%d", &numG);

	visited = (int*)malloc(numG * sizeof(int));
	G = (int**)malloc(numG * sizeof(int*));
	for (int i = 0; i < numG; i++)
		G[i] = (int*)malloc(numG * sizeof(int));

	srand(time(NULL));

	for (int i = 0; i < numG; i++) {
		visited[i] = 0;
		for (int j = i; j < numG; j++) {
			G[i][j] = G[j][i] = (i == j ? 0 : rand() % 2);

		}
	}

	for (int i = 0; i < numG; i++) {
		for (int j = 0; j < numG; j++) {
			printf("%3d", G[i][j]);

		}
		printf("\n");
	}

	printf("input start vershinka: ");
	scanf("%d", &current);


	printf("\nPath: ");
	BFS(G, numG, visited, current);
	printf("\n\n");

	free(visited);
	for (int i = 0; i < numG; i++)
		free(G[i]);
	free(G);

	return 0;

}
```

Результаты работы программы

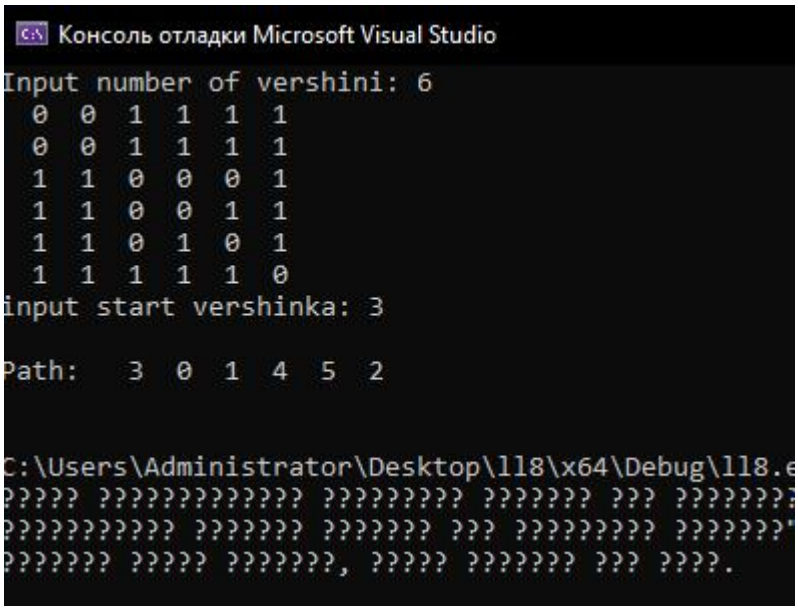


Рисунок 1 - Результат работы программы

Вывод: В ходе выполнения лабораторной работы был успешно изучен и реализован алгоритм обхода графа в ширину для различных структур данных: матрицы смежности и списков смежности.