МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. Н. Э. Баумана
КАФЕДРА ПРОЕКТИРОВАНИЕ И ТЕХНОЛОГИЯ ПРОИЗВОДСТВА ЭЛЕКТРОННОЙ АППАРАТУРЫ

Отчет о выполнении практического задания №5

«Удалить ребра, которые смежны с заданной вершиной, и в модифицированном графе отсортировать вершины по убыванию степени вершины»

по курсу «Функциональная логика и теория алгоритмов»

Выполнил: студент каф. ИУ4-21Б
Назаренко Денис Игоревич
Проверил:

Москва 2024

Цель работы

Целью данного кода является реализация базовых операций с графом, таких как инициализация, добавление ребер, удаление ребер, получение степени вершины, сортировка вершин по убыванию степени.

1. Исходные данные

Данные, подающиеся на вход (вводимые с клавиатуры):

Количество вершин графа и список ребер, которые необходимо добавить в граф.

Данные, которые необходимо вывести:

- Исходный граф
- Граф после удаления ребер, смежных с заданной вершиной
- Вершины, отсортированные по убыванию степени

2. Выполнение

1. initGraph(Graph* graph, int numVertices): Инициализирует граф, заполняя матрицу смежности нулями.
2. addEdge(Graph* graph, int src, int dest): Добавляет ребро между вершинами src и dest, устанавливая соответствующие значения в матрице смежности.
3. removeEdgesConnectedToVertex(Graph* graph, int vertex): Удаляет ребра, смежные с заданной вершиной, обнуляя соответствующие значения в матрице смежности.
4. getDegree(Graph* graph, int vertex): Возвращает степень вершины - количество инцидентных ей ребер.
5. sortVerticesByDegreeDesc(Graph* graph, int* sortedVertices): Сортирует вершины по убыванию степени, используя функцию getDegree.

3. Плюсы и минусы:

Плюсы:

- Простая и понятная реализация базовых операций с графом.
- Код легко читается и понимается.

Минусы:

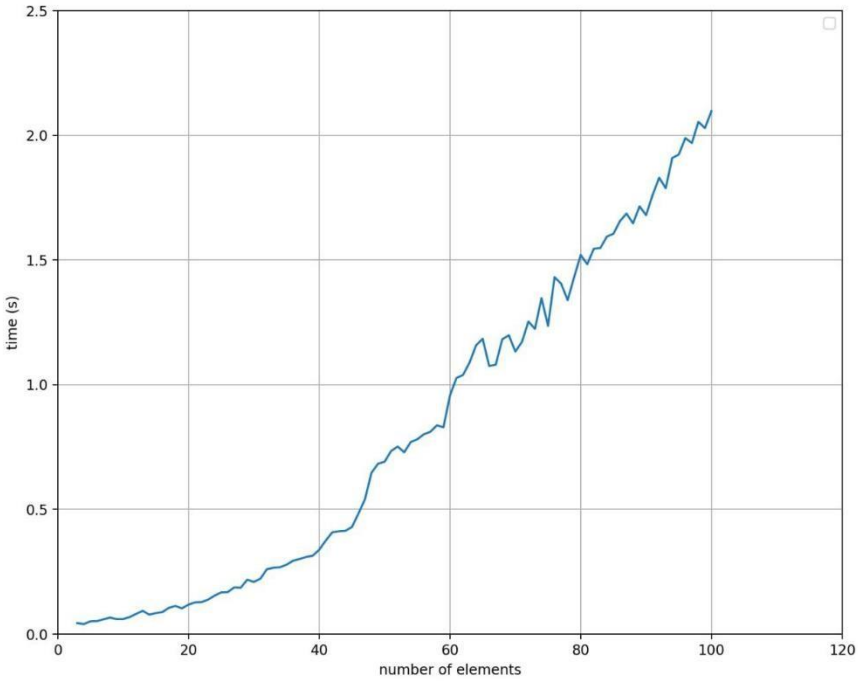
- Нет обработки ошибок при вводе данных, что может привести к сбоям программы при неправильном вводе.
- Код не оптимизирован для работы с большими графами, так как используется матричное представление графа, что может быть неэффективно для хранения больших графов.

4. Сложность:

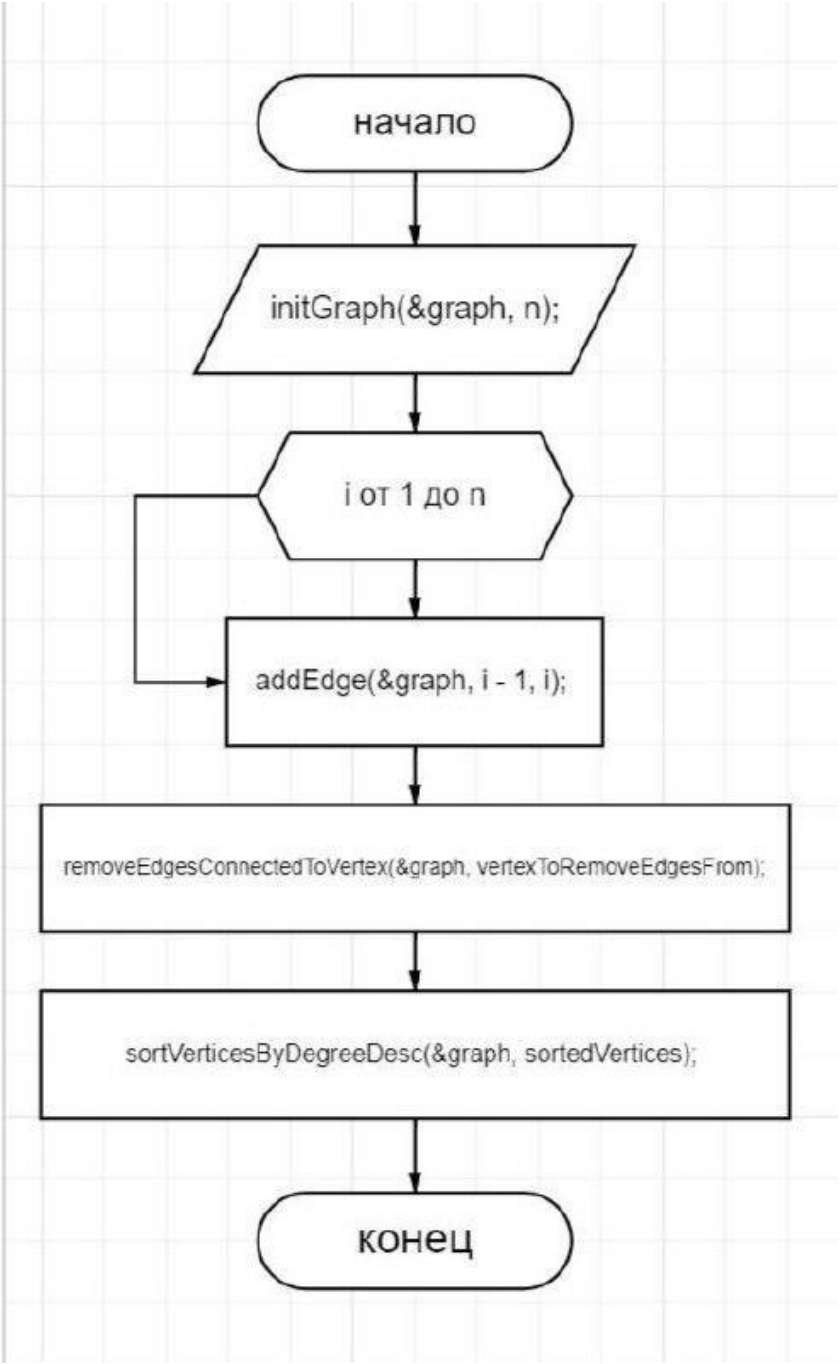
- Инициализация графа: O(V^2), где V - количество вершин.
- Добавление/удаление ребра: O(1).
- Получение степени вершины: O(V), где V - количество вершин.

- Сортировка вершин: O(V^2), где V - количество вершин.

5. График эффективности:



6. Блок-схема:


начало
initGraph(&graph, n);
i от 1 до n
addEdge(&graph, i - 1, i);
removeEdgesConnectedToVertex(&graph, vertexToRemoveEdgesFrom);
sortVerticesByDegreeDesc(&graph, sortedVertices);
конец

7. **Код:**

```c
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>

#define MAX_VERTICES 100

// Структура для представления графа
typedef struct {
    int matrix[MAX_VERTICES][MAX_VERTICES];
    int numVertices;
} Graph;

// Инициализация графа
void initGraph(Graph* graph, int numVertices) {
    graph->numVertices = numVertices;
    for (int i = 0; i < numVertices; i++) {
        for (int j = 0; j < numVertices; j++) {
            graph->matrix[i][j] = 0;
        }
    }
}

// Добавление ребра в граф
void addEdge(Graph* graph, int src, int dest) {
    graph->matrix[src][dest] = 1;
    graph->matrix[dest][src] = 1;
}

// Удаление ребер, смежных с заданной вершиной
void removeEdgesConnectedToVertex(Graph* graph, int vertex) {
    if (vertex >= graph->numVertices) {
        printf("Ошибка: Недопустимая вершина\n");
        exit(EXIT_FAILURE);
    }

    for (int i = 0; i < graph->numVertices; i++) {
        if (graph->matrix[vertex][i] == 1) {
            graph->matrix[vertex][i] = 0;
            graph->matrix[i][vertex] = 0;
        }
    }
}

// Получение степени вершины
int getDegree(Graph* graph, int vertex) {
    int degree = 0;
    for (int i = 0; i < graph->numVertices; i++) {
        if (graph->matrix[vertex][i] == 1) {
            degree++;
        }
    }
    return degree;
}

```

```
// Сортировка вершин по убыванию степени
void sortVerticesByDegreeDesc(Graph* graph, int* sortedVertices) {
    // Инициализация массива отсортированных вершин
    for (int i = 0; i < graph->numVertices; i++) {
        sortedVertices[i] = i;
    }

    // Сортировка массива отсортированных вершин по убыванию степени
    for (int i = 0; i < graph->numVertices - 1; i++) {
        for (int j = 0; j < graph->numVertices - i - 1; j++) {
            if (getDegree(graph, sortedVertices[j]) < getDegree(graph, sortedVert
                int temp = sortedVertices[j];
                sortedVertices[j] = sortedVertices[j + 1];
                sortedVertices[j + 1] = temp;
            }
        }
    }
}

int main() {
    setlocale(LC_ALL, "rus");
    printf("Введите количество ребер: ");
    int n = 0;
    scanf("%d", &n);
    Graph graph;
    initGraph(&graph, n);

    addEdge(&graph, 0, 2);
    for (int i = 1; i <= n; i++) {
        addEdge(&graph, i - 1, i);
    }

    printf("Исходный граф:\n");
    for (int i = 0; i < graph.numVertices; i++) {
        for (int j = 0; j < graph.numVertices; j++) {
            printf("%d ", graph.matrix[i][j]);
        }
        printf("\n");
    }

    printf("Какую вершину хотите удалить?\n ");
    int vertexToRemoveEdgesFrom = 0;
    scanf("%d", &vertexToRemoveEdgesFrom);
    removeEdgesConnectedToVertex(&graph, vertexToRemoveEdgesFrom);

    printf("\nГраф после удаления ребер, смежных с вершиной %d:\n", vertexToRemov
    for (int i = 0; i < graph.numVertices; i++) {
        for (int j = 0; j < graph.numVertices; j++) {
            printf("%d ", graph.matrix[i][j]);
        }
        printf("\n");
    }

    // Получаем массив отсортированных вершин по убыванию степени
    int sortedVertices[MAX_VERTICES];
    sortVerticesByDegreeDesc(&graph, sortedVertices);

    printf("\nВершины, отсортированные по убыванию степени:\n");
    for (int i = 0; i < graph.numVertices; i++) {
        printf("%d ", sortedVertices[i]);
    }

    return 0;
}
```

_CRTIMP int __cdecl __MINGW_NOTHROW printf(const char*, ...)

8. Результаты работы.

Удачная попытка:

```
Введите количество ребер: 9
Исходный граф:
0 1 1 0 0 0 0 0 0
1 0 1 0 0 0 0 0 0
1 1 0 1 0 0 0 0 0
0 0 1 0 1 0 0 0 0
0 0 0 1 0 1 0 0 0
0 0 0 0 1 0 1 0 0
0 0 0 0 0 1 0 1 0
0 0 0 0 0 0 1 0 1
0 0 0 0 0 0 0 1 0
Какую вершину хотите удалить?
 5

Граф после удаления ребер, смежных с вершиной 5:
0 1 1 0 0 0 0 0 0
1 0 1 0 0 0 0 0 0
1 1 0 1 0 0 0 0 0
0 0 1 0 1 0 0 0 0
0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0 1
0 0 0 0 0 0 0 1 0

Вершины, отсортированные по убыванию степени:
2 0 1 3 7 4 6 8 5
```

Неудачная попытка:

```
Введите количество ребер: 6
Исходный граф:
0 1 1 0 0 0
1 0 1 0 0 0
1 1 0 1 0 0
0 0 1 0 1 0
0 0 0 1 0 1
0 0 0 0 1 0
Какую вершину хотите удалить?
 9
Ошибка: Недопустимая вершина
```

9. Вывод

Данный код представляет простую реализацию операций с графом. Он хорош для обучения основам работы с графами, но требует доработок для улучшения эффективности и обработки возможных ошибочных ситуаций.