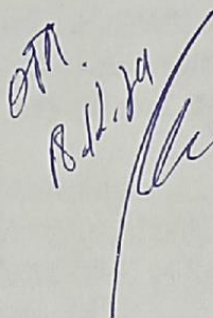


Министерство науки и высшего образования Российской Федерации
Пензенский государственный университет
Кафедра «Вычислительная техника»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе
по курсу «Логика и основы Алгоритмизации»
на тему «Реализация алгоритма Прима»



Выполнил:

студент группы 23BB2

Лисов Е.А.

Принял:

Митрохин М.А.

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет Вычислительной техники
Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ

« » 20

ЗАДАНИЕ

на курсовое проектирование по курсу

Исходные данные (технические требования) на проектирование
Студенту Лисову Егору Алексеевичу Группа 23ВВВ2
Тема проекта Реализация алгоритма Фриша

Исходные данные (технические требования) на проектирование

Разработка алгоритмов и программного обеспечения в соответствии с данными заданием курсового проекта.

Пояснительная записка должна содержать:

1. Постановку задачи
2. Теоретическую часть задания
3. Описание алгоритма поставленной задачи
4. Пример ручного расчета задачи и вычислений
5. Описание самой программы
6. Тесты
7. Список литературы
8. Результаты работы программы

Объем работы по курсу

1. Расчетная часть

Ручной расчет работы алгоритма

2. Графическая часть

Схема алгоритма в формате блок-схем

3. Экспериментальная часть

Тестирование программы

Результаты работы программы на тестовых данных

Срок выполнения проекта по разделам

- 1 Исследования теоретической части курсовой
- 2 Разработка алгоритмов программы
- 3 Разработка программы
- 4 Тестирование и завершение разработки программы
- 5 Оформление пояснительной записки
- 6
- 7
- 8

Дата выдачи задания "06" сентября 2024

Дата защиты проекта "18" декабря 2024

Руководитель Митрохин А.А.

Задание получил "06" сентября 2024 г.

Студент Лисов Е.А.

Содержание

Содержание.....	2
Реферат.....	5
Введение	6
1 Постановка задачи	8
2 Описание алгоритма	9
3 Описание программы	13
3.1 Разработка модуля инициализации графа, его перевода в матрицу смежности, а также модуль вывода этой матрицы на экран	13
3.2 Разработка модуля ручного заполнения графа	14
3.3 Разработка модуля определения вершины с минимальным весом на текущий момент и модуль самого алгоритма Прима	15
3.4 Модуль вывода результата на экран	17
3.5 Разработка модуля сохранения результата в отдельный текстовый файл	18
5 Тестирование	19
6 Ручной подсчет.....	27
Заключение	29
Список используемых источников.....	30
Приложение А Листинги программы	31
Файл «Main.cpp»	31
Приложение В	35
Окна программы.....	35

Реферат

Отчет 35 стр, 11 рисунков

Алгоритм, остов, вершина, остовное дерево, ребро.

Цель исследования – разработка программы, способная находить минимальное остовное дерево с помощью алгоритма Прима.

В работе рассмотрен алгоритм прима, его принцип. Установлено, что алгоритм Прима работает только с неориентированными графами.

В ходе выполнения работы был реализован алгоритм Прима на языке Си, в документе описаны: алгоритм программы, сама программа, разбитая на модули, проведено тестирование программы, представлены результаты работы программы и ее полный код.

Введение

Алгоритм Прима - алгоритм построения минимального остовного дерева взвешенного связного неориентированного графа. Алгоритм впервые был открыт в 1930 году чешским математиком Войцехом Ярником, позже переоткрыт Робертом Примом в 1957 году, и, независимо от них, Э. Дейкстрой в 1959 году.

На вход алгоритма подаётся связный неориентированный граф. Для каждого ребра задаётся стоимость.

Сначала берётся произвольная вершина и находится ребро, инцидентное данной вершине и обладающее наименьшей стоимостью. Найденное ребро и соединяемые им две вершины образуют дерево. Затем, рассматриваются рёбра графа, один конец которых - уже принадлежащая дереву вершина, а другой - нет; из этих рёбер выбирается ребро наименьшей стоимости. Выбираемое на каждом шаге ребро присоединяется к дереву. Рост дерева происходит до тех пор, пока не будут исчерпаны все вершины исходного графа.

Результатом работы алгоритма является остовное дерево минимальной стоимости.

Для разработки программы была выбрана среда *Visual Studio 2022*, предоставляющая разработчикам мощные инструменты для написания и отладки кода. Управление в программе осуществляется с помощью клавиатуры и мыши.

Для реализации программы было выбрано использование языка программирования C, что обусловлено его широкой популярностью, а также высокой производительностью и эффективностью. Язык C отличается скоростью выполнения кода, низкими требованиями к системным ресурсам и возможностью работы на уровне аппаратных ресурсов.

Целью данной курсовой работы является разработка программы на языке Си, который является широко используемым. Именно с его помощью в данном курсовом проекте реализуется алгоритм Прима, осуществляющий поиск минимального остовного дерева.

1 Постановка задачи

Требуется реализовать программу, которая будет находить минимальное остовное дерево используя алгоритм Прима. Исходный граф в программе должен задаваться матрицей смежности, причем при генерации данных должны быть предусмотрены граничные условия. Программа должна работать так, чтобы пользователь вводил количество вершин для генерации матрицы смежности. После обработки этих данных программа должна выводить на экран результат в виде матрицы смежности. Устройство ввода-клавиатура. Также в программе должны быть предусмотрены: Текстовое меню, дающее возможность задания пользователем размера графа выбора автоматического или ручного (с клавиатуры) задания графа возможность сохранения результатов работы программы в отдельный текстовый файл. Все эти функции должны быть реализованы с использованием языка программирования Си, соответствующих библиотек и функций.

Минимальные системные требования

Операционная система: Windows

Процессор: 1 ГГц или выше.

Оперативная память: минимум 256 МБ (рекомендуется 1 ГБ или более для больших графов).

Место на диске: 1-10 МБ.

Компилятор: GCC, MSVC, Clang или другой совместимый компилятор C.

2 Описание алгоритма

Алгоритм работает с матрицей смежности графа, которая, в свою очередь задается пользователем, либо случайно.

Работа алгоритма включает в себя 3 этапа: инициализация, рекурсивный обход, вывод результата.

Изначально остов полагается состоящим из единственной вершины (её можно выбрать произвольно).

Затем выбирается ребро минимального веса, исходящее из этой вершины, и добавляется в минимальный остов.

После этого остов содержит уже две вершины, и теперь ищется и добавляется ребро минимального веса, имеющее один конец в одной из двух выбранных вершин, а другой - наоборот, во всех остальных, кроме этих двух.

И так далее, то есть всякий раз ищется минимальное по весу ребро, один конец которого - уже взятая в остов вершина, а другой конец ещё не взятая, и это ребро добавляется в остов (если таких рёбер несколько, можно взять любое).

Для более удобной реализации программы понадобится несколько функций.

Инициализация матрицы смежности, заполнение матрицы весов вручную, заполнение матрицы весов случайным образом (в данной программе максимальный вес вершины равен 9), вывод сгенерированной матрицы на экран, алгоритм нахождения вершины с наименьшим весом, рабочий алгоритм программы (алгоритм прима), алгоритм вывода результат на экран, алгоритм сохранения результат в файл.

Для реализации каждого вышеперечисленного пункта были написаны соответствующие функции.

Алгоритм прима был реализован с помощью отдельной функции *primMST*, в которую включена еще одна функция для поиска вершины с минимальным весом *minKey*.

Функция *minKey* принимает на вход массив, который хранит минимальные веса ребер, соединяющие вершину с MST (*key[]*), массив, который отслеживает, какие вершины уже добавлены в MST (*mstSet[]*). Если *mstSet[v]* равно 0, значит, вершина *v* еще не добавлена, количество вершин в графе (*size*). Также создаются переменные *min* и *min_index*

Min инициализируется максимальным значением целого типа, для того, чтобы гарантировать, что любое значение веса ребра будет меньше

Min_index - переменная хранящая индекс вершины с минимальным ключом (возвращаемая переменная).

Цикл проходит по всем вершинам и ищет вершину, которая еще не включена в MST и имеет минимальный вес (значение в *key*). Если такая вершина найдена, обновляются значения *min* и *min_index*.

В конце функция возвращает индекс вершины с минимальным ключом.

primMST принимает на вход указатель на матрицу весов и размер этой матрицы.

В самом алгоритме выделяется память под итоговую матрицу весов (*mstGraph*), список включенных вершин (*mstSet*), минимальных весов ребер для каждой вершины (*key*), родительских вершин (*parent*).

После этого с помощью цикла *for* и оператора *calloc* итоговая матрица инициализируется нулями, все ячейки матрицы *key* заполняются максимальными значениями целочисленного формата, все ячейки *mstSet* устанавливаются в значение 0, т.к. мы не включили еще ни одной вершины, ключ первой вершины устанавливается нулевым, чтобы алгоритм начался с неё.

Далее запускается основной цикл алгоритма:

Цикл выполняется на одну итерацию меньше, чем значение кол-ва вершин графа, так как *MST* для графа с *size* вершинами будет содержать *size - 1* ребер.

На каждой итерации вызывается *minKey*, чтобы получить вершину с минимальным ключом, которая еще не включена в *MST*.

Вершина добавляется в *MST*, и для каждой соседней вершины проверяется, можно ли обновить ключ для этой вершины. Если вес ребра меньше текущего ключа, обновляются значения в *parent* и *key*.

После этого начинается заполнение итоговой матрицы весов с помощью массива родителей, для установки соответствующих ребер.

В конце, для избежания утечек памяти, освобождается ранее выделенная память на массивы, за исключением *mstGraph*.

В конечном итоге функция возвращает минимальное остовное дерево.

Псевдокод функции *primMST* и выглядит следующим образом:

minKey:

вход: список минимальных весов, список посещенных вершин,

количество вершин

primMST:

```
вход исходная матрица смежности, кол-во вершин
выделить память для списка индексов родителей
выделить память для списка минимальных весов
выделить память для списка посещенных вершин
выделить память для итоговой матрицы смежности
заполнить итоговую матрицу нулями
списку минимальных весов задать максимальные значения
списку посещенных вершин задать нулевые значения
первому элементу в списке весов поставить значение 0
первому элементу списка родителей поставить значение -1
while переменная счетчика меньше кол-ва вершин{
    минимальный индекс присвоить новой переменной
    отметить вершину как посещенную
    прибавить к переменной счетчика единицу
    for для всех элементов строки {
```

```

если в исх графе по минимальному индексу столбца и очередному индексу строки И эта
вершина не рассматривалась ранее (MSTset) И значение в этой рассматриваемой ячейке
меньше минимального пути то
значение очередного родителя будет перезаписано, индексом минимального ребра
по очередному индексу ребра будет записан вес ребра}}
for для всех вершин строки {
ячейка в итоговой матрице по индексу столбца очередного родительского по очередному
индексу строки будет равна значению по очередному родительскому индексу столбца и
очередному индексу строки
ячейка в итоговой матрице по индексу строки очередного родительского по очередному
индексу столбца будет равна значению по очередному родительскому индексу столбца и
очередному индексу строки
}
освободить память выделенной для родительского списка
освободить память выделенной для списка посещенных вершин
вернуть матрицу смежности минимального остоного дерева

```

Программа разрабатывалась с использованием различных выполняющих разные функции. Этот подход позволяет точнее отследить возможные ошибки в процессе разработки, делает саму программу более гибкой для дальнейшего дополнения функционала. Этапы пронумерованы.

Разработанная программа состоит из нескольких модулей.

1. Модуль инициализации графа, его перевода в матрицу смежности, а также модуль вывода этой матрицы на экран.
2. Модуль ручного заполнения графа
3. Модуль определения вершины с минимальным весом на текущий момент и модуль самого алгоритма Прима
4. Модуль вывода результата на экран.
5. Модуль сохранения результата в отдельный текстовый файл.

3 Описание программы

3.1 Разработка модуля инициализации графа, его перевода в матрицу смежности, а также модуль вывода этой матрицы на экран

```
int** createG(int size) {
    int** G = (int**)malloc(size * sizeof(int*));
    for (int i = 0; i < size; i++) {
        G[i] = (int*)malloc(size * sizeof(int));
    }
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            if (i == j) {
                G[i][j] = 0;
            }
            else {
                G[i][j] = rand() % 10;
            }
            G[j][i] = G[i][j];
        }
    }
    return G;
}

void printG(int** G, int size) {
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            printf("%d ", G[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}
```

Функция `createG` выделяет память на исходный граф с помощью функции `malloc` и в последствии заполняет его с помощью цикла `for` и оператора `rand`, при этом, значение элементов, находящихся на главной диагонали равно нулю, а половина выше главной диагонали «отзеркалена» относительно нижней и наоборот. В конечном итоге функция возвращает указатель на матрицу.

Функция `printG` выполняет проход по всем элементам матрицы смежности и с помощью `printf` выводит матрицу на экран.

3.2 Разработка модуля ручного заполнения графа

```
int** createGManual(int size) {
    int** G = (int**)malloc(size * sizeof(int*));
    for (int i = 0; i < size; i++) {
        G[i] = (int*)malloc(size * sizeof(int));
    }
    printf("Введите веса рёбер (0 если ребра нет):\n");
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            if (i == j) {
                G[i][j] = 0;
            }
            else {
                printf("Вес ребра %d-%d: ", i, j);
                scanf("%d", &G[i][j]);
            }
            G[j][i] = G[i][j];
        }
    }
    return G;
}
```

Эта функция похожа функцию `createG`: тут тоже выделяется память с помощью `malloc`, рассчитывается она, исходя из размерности графа. Далее как и в `createG` выполняется проход по всем ячейкам матрицы, с помощью оператора `scanf` пользователь задает значения веса ячеек матрицы, которые, в свою очередь подписаны, при этом главная диагональ не заполняется, а матрица так же «отзеркаливается» это позволяет сэкономить время, исключить ошибки со стороны пользователя.

3.3 Разработка модуля определения вершины с минимальным весом на текущий момент и модуль самого алгоритма Прима

```
int minKey(int key[], int mstSet[], int size) {
    int min = INT_MAX, min_index;

    for (int v = 0; v < size; v++) {
        if (mstSet[v] == 0 && key[v] < min) {
            min = key[v];
            min_index = v;
        }
    }
    return min_index;
}

int** primMST(int** graph, int size) {
    int* parent = (int*)malloc(size * sizeof(int));
    int* key = (int*)malloc(size * sizeof(int));
    int* mstSet = (int*)malloc(size * sizeof(int));
    int** mstGraph = (int**)malloc(size * sizeof(int*));

    // Инициализируем матрицу смежности для MST
    for (int i = 0; i < size; i++) {
        mstGraph[i] = (int*)calloc(size, sizeof(int)); // Инициализируем нулями
    }

    for (int i = 0; i < size; i++) {
        key[i] = INT_MAX;
        mstSet[i] = 0;
    }

    key[0] = 0;
    parent[0] = -1;

    for (int count = 0; count < size - 1; count++) {
        int u = minKey(key, mstSet, size);
        mstSet[u] = 1;

        for (int v = 0; v < size; v++) {
```

```

        if (graph[u][v] && mstSet[v] == 0 && graph[u][v] < key[v]) {
            parent[v] = u;
            key[v] = graph[u][v];
        }
    }
}

// Заполнение матрицы смежности для MST
for (int i = 1; i < size; i++) {
    mstGraph[parent[i]][i] = graph[parent[i]][i];
    mstGraph[i][parent[i]] = graph[parent[i]][i];
}

free(parent);
free(key);
free(mstSet);

return mstGraph; // Возвращаем матрицу смежности для MST
}

```

Функция minKey

В самом начале функция присваивает к переменной min значение, равное максимальному значению переменной целого типа. Это делается для того чтобы исключить вероятность отказа программы при введении слишком большого числа.

Далее идет цикл, сравнивающий значение каждой вершины графа, а вложенный условный оператор if сравнивает значение этой ячейки с min. если значение меньше, то в min записывается это значение.

Функция primMST

Это основная функция для вычисления минимального остовного дерева с помощью алгоритма Прима.

graph: матрица смежности графа, где graph[u][v] - вес ребра между вершинами u и v.

size: количество вершин в графе.

parent[]: хранит родительские вершины для каждой вершины в MST.

key[]: массив, хранящий минимальные веса рёбер для каждой вершины.

mstSet[]: отслеживает, включена ли вершина в MST (0 - не включена, 1 - включена).

mstGraph[][]: матрица смежности для минимального остовного дерева, которую мы и будем возвращать.

Все вершины имеют значение ключа INT_MAX, что символизирует бесконечность (не найдено подходящего ребра).

Начальная вершина (первый элемент) имеет ключ равный 0.

Родитель первой вершины устанавливается в -1 (это начальная вершина, у неё нет родителя).

Основной цикл:

В цикле for (int count = 0; count < size - 1; count++) выбираются вершины, пока не будет включена каждая вершина.

Для каждой итерации вызывается функция minKey, чтобы выбрать вершину с минимальным весом ребра, которая ещё не включена в MST.

Затем для всех смежных вершин v (для каждой вершины u, смежной с u), если вес ребра graph[u][v] меньше текущего значения ключа key[v] и вершина v ещё не включена в MST, обновляется ключ и родитель для вершины v.

После того как алгоритм завершен, заполняется матрица смежности для MST (mstGraph), где для каждой вершины устанавливаются рёбра, входящие в минимальное остовное дерево.

3.4 Модуль вывода результата на экран

Вывод результата реализован в основной части программы и реализуется посредством использования указателя lastGraph, который в свою очередь будет передаваться в остальные блоки, а в них, как было описано

ранее будет выделяться место в памяти под хранение исходных данных и результатов.

3.5 Разработка модуля сохранения результата в отдельный текстовый файл

```
void saveToFile(int** graph, int size, const char* filename) {  
    FILE* file = fopen(filename, "w");  
    if (file == NULL) {  
        printf("Ошибка открытия файла для записи.\n");  
        return;  
    }  
    fprintf(file, "%d\n", size);  
    for (int i = 0; i < size; i++) {  
        for (int j = 0; j < size; j++) {  
            fprintf(file, "%d ", graph[i][j]);  
        }  
        fprintf(file, "\n");  
    }  
    fclose(file);  
    printf("Результаты сохранены в файл %s.\n", filename);  
}
```

В этой функции используются операторы, предназначенные для работы с файлами, функция принимает указатель на граф, полученный в результате работы программе на вход, и записывает данные о нем в файл, создающийся в той же папке, где лежит программа.

5 Тестирование

В процессе отладки работы была использована программа Visual Studio 2022. Эта программа предоставляет все необходимые возможности для тестирования и отладки кода и ее модулей. Были использованы различные инструменты, такие как точки останова, выполнение кода по шагам, анализ переменных и другие, чтобы обнаружить и исправить ошибки.

Тестирование программы проводилось на разных этапах с помощью точек останова: после реализации каждой функции, а также после окончания всего программного кода. В процессе тестирования было выявлено и исправлено множество ошибок, которые были связаны с неверным определением графа, не работоспособностью функций и т.п.

Проверки работоспособности программы для матриц размерностью представлены на рисунках и соответствуют каждому пункту таблицы №1.

Таблица 1 - Описание поведения программы при тестировании

№	Описание	Предусловие	Тестирование	Ожидаемый результат
1	Работа меню	Программа запущена	Запускаем программу в <i>visual studio</i>	Вывод меню на экран
2	Выбор операции	Программа запущена	Ввод номера операции	Осуществление соответствующей операции
3	Задание кол-ва вершин графа вручную	Ввод 1 в меню	Ввод кол-ва вершин	Количество вершин графа будет равно введенному числу

Таблица 1 продолжение

№	Описание	Предусловие	Тестирование	Ожидаемый результат
4	Заполнение матрицы случайно	Ввод 2 в меню	Ввод 2 в меню	Вывод сгенерированной матрицы на экран
5	Заполнение матрицы вручную	Ввод 3 в меню	Ввод весов каждой вершины	Последовательный вывод запроса о введении значений ячеек
6	Вывод результата алгоритма	Ввод 4 в меню	Ввод 4 в меню	Результат выведется на экран
7	Сохранение результатов в отдельный файл	Ввод 5 в меню	Ввод названия файла	Создание файла в папке программы соответствующего названия
8	Выход из программы	Ввод 6 в меню	Ввод 6 в меню	Завершение работы программы

```

Меню:
1. Задать кол-во вершин графа
2. Создать граф случайно
3. Создать граф вручную
4. Вычислить минимальное остовное дерево
5. Сохранить результаты в файл
6. Выход
Выберите опцию:

```

Рисунок 1 - Меню программы

```
D:\Edu\2 курс\Логика и основы алгоритмизации\
Меню:
1. Задать кол-во вершин графа
2. Создать граф случайно
3. Создать граф вручную
4. Вычислить минимальное остовное дерево
5. Сохранить результаты в файл
6. Выход
Выберите опцию: 1
Введите кол-во вершин графа: █
```

Рисунок 2 - Выбор операции

```
D:\Edu\2 курс\Логика и основы алгоритмизации\
Меню:
1. Задать кол-во вершин графа
2. Создать граф случайно
3. Создать граф вручную
4. Вычислить минимальное остовное дерево
5. Сохранить результаты в файл
6. Выход
Выберите опцию: 1
Введите кол-во вершин графа: 6
Меню:
1. Задать кол-во вершин графа
2. Создать граф случайно
3. Создать граф вручную
4. Вычислить минимальное остовное дерево
5. Сохранить результаты в файл
6. Выход
Выберите опцию:
```

Рисунок 3 - Задание кол-ва вершин графа вручную

```
D:\Edu\2 курс\Логика и основы алгоритмизации\K
5. Сохранить результаты в файл
6. Выход
Выберите опцию: 1
Введите кол-во вершин графа: 6
Меню:
1. Задать кол-во вершин графа
2. Создать граф случайно
3. Создать граф вручную
4. Вычислить минимальное остовное дерево
5. Сохранить результаты в файл
6. Выход
Выберите опцию: 2
Сгенерированный граф:
0 5 0 7 7 9
5 0 6 6 1 9
0 6 0 4 9 8
7 6 4 0 4 1
7 1 9 4 0 6
9 9 8 1 6 0

Меню:
1. Задать кол-во вершин графа
2. Создать граф случайно
3. Создать граф вручную
4. Вычислить минимальное остовное дерево
5. Сохранить результаты в файл
6. Выход
Выберите опцию: █
```

Рисунок 4 - Заполнение матрицы случайно

```
D:\Edu\2 курс\Логика и основы алг...
6. Выход
Выберите опцию: 3
Введите веса рёбер (0 если ребра нет):
Вес ребра 0-1: 1
Вес ребра 0-2: 5
Вес ребра 0-3: 3
Вес ребра 0-4: 4
Вес ребра 0-5: 2
Вес ребра 1-0: 6
Вес ребра 1-2: 2
Вес ребра 1-3: 1
Вес ребра 1-4: 6
Вес ребра 1-5: 1
Вес ребра 2-0: 7
Вес ребра 2-1: 5
Вес ребра 2-3: 4
Вес ребра 2-4: 2
Вес ребра 2-5: 4
Вес ребра 3-0: 3
Вес ребра 3-1: 2
Вес ребра 3-2: 0
Вес ребра 3-4: 3
Вес ребра 3-5: 0
Вес ребра 4-0: 6
Вес ребра 4-1: 4
Вес ребра 4-2: 5
Вес ребра 4-3: 7
Вес ребра 4-5: 3
Вес ребра 5-0: 0
Вес ребра 5-1: 2
Вес ребра 5-2: 6
Вес ребра 5-3: 4
Вес ребра 5-4: 2
Введенный граф:
0 6 7 3 6 0
6 0 5 2 4 2
7 5 0 0 5 6
3 2 0 0 7 4
6 4 5 7 0 2
0 2 6 4 2 0
```

Рисунок 5 - Заполнение матрицы вручную


```
Меню:
1. Задать кол-во вершин графа
2. Создать граф случайно
3. Создать граф вручную
4. Вычислить минимальное остовное дерево
5. Сохранить результаты в файл
6. Выход
Выберите опцию: 4
Матрица смежности минимального остовного дерева:
0 0 0 3 0 0
0 0 5 2 0 2
0 5 0 0 0 0
3 2 0 0 0 0
0 0 0 0 0 2
0 2 0 0 2 0

Меню:
1. Задать кол-во вершин графа
2. Создать граф случайно
3. Создать граф вручную
4. Вычислить минимальное остовное дерево
5. Сохранить результаты в файл
6. Выход
Выберите опцию: _
```

Рисунок 6 - Вывод результата алгоритма

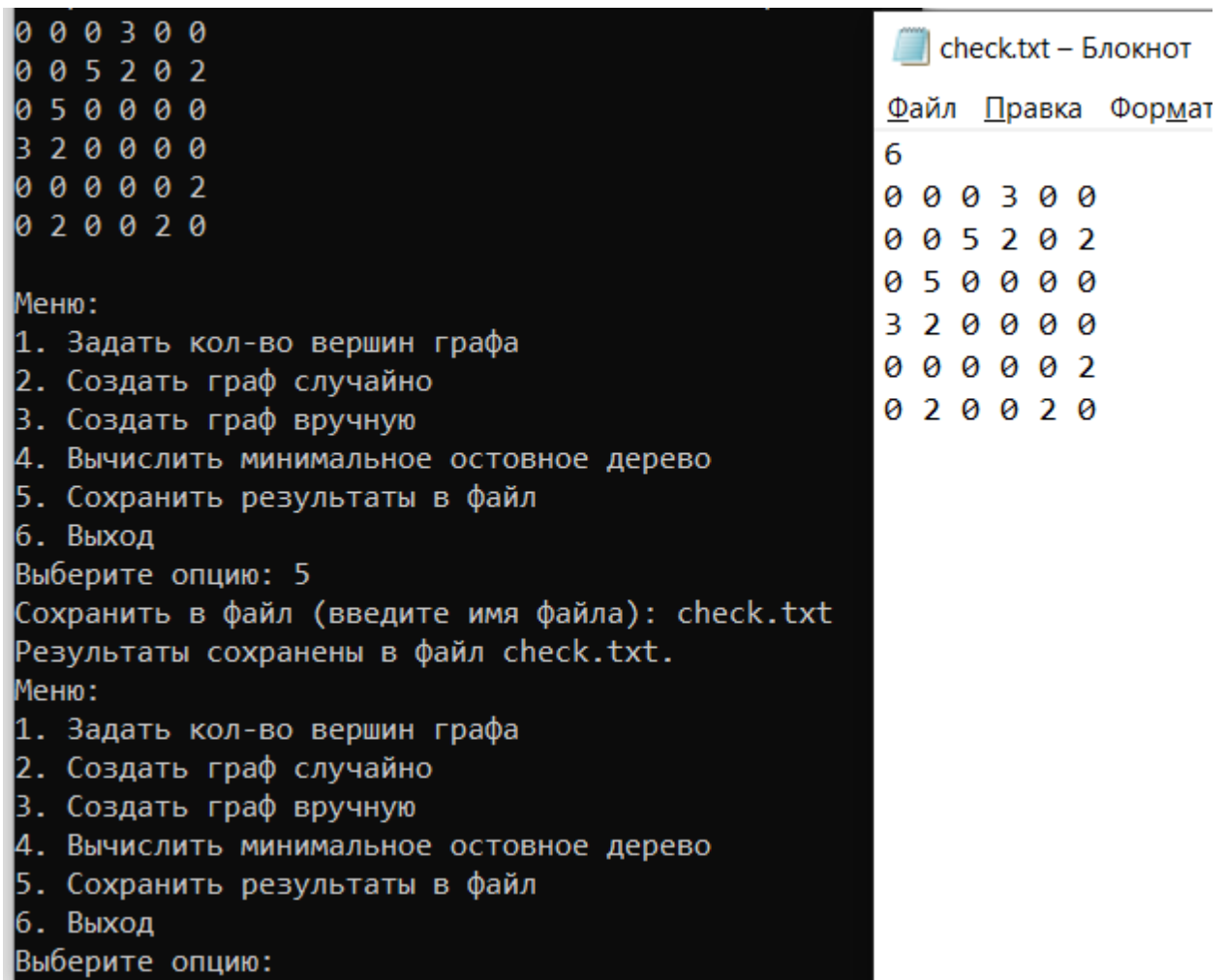


Рисунок 7 - Сохранение результатов в отдельный файл

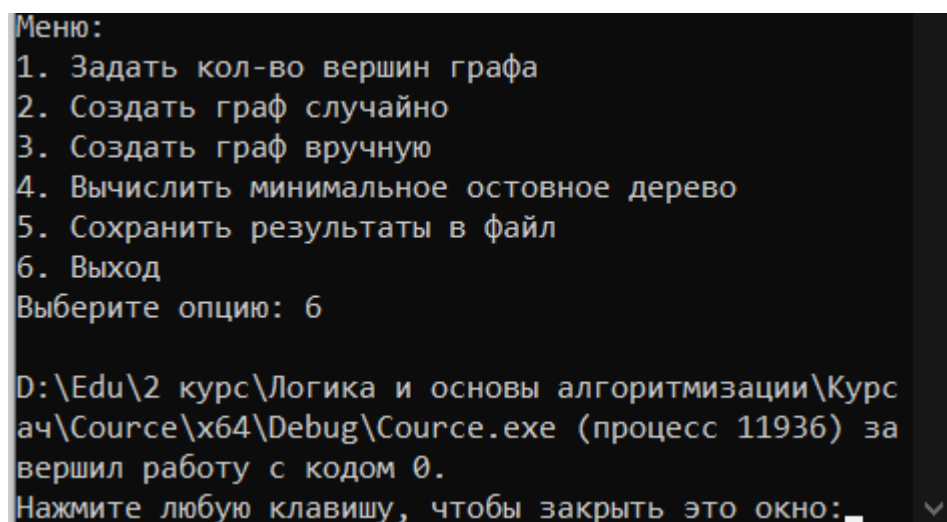


Рисунок 8 - Выход из программы

Таблица 2 – Результаты тестирования программы

Описание теста	Ожидаемый результат	Полученный результат
Работа меню	Вывод меню на экран	Верно
Выбор операции	Осуществление соответствующей операции	Верно
Задание кол-ва вершин графа вручную	Количество вершин графа будет равно введенному числу	Верно
Заполнение матрицы случайно	Вывод сгенерированной матрицы на экран	Верно
Заполнение матрицы вручную	Последовательный вывод запроса о введении значений ячеек	Верно
Вывод результата алгоритма	Результат выведется на экран	Верно
Сохранение результатов в отдельный файл	Создание файла в папке программы соответствующего названия	Верно
Выход из программы	Завершение работы программы	Верно

6 Ручной подсчет

Для проверки на верный результат была сгенерирована матрица, после был проведен ручной проход по всем вершинам методом нахождения минимальных расстояний между вершинами, как только несвязанных вершин не осталось, следует проверить результат с результатом программы. Результат ручной проверки представлен на рисунке 4. Результат работы программы с этими значениями представлен на рисунке 5.

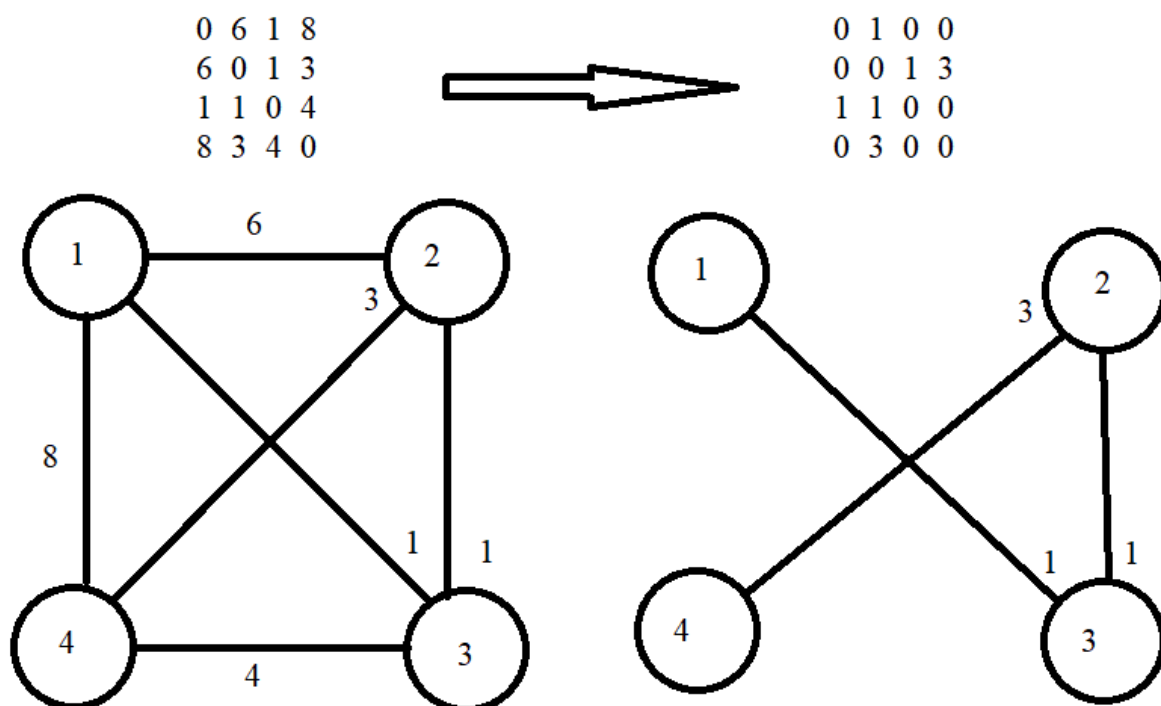


Рисунок 9 - Результат работы для проверки

```
D:\Edu\2 курс\Логика и основы алгоритмизации\Курсач\Course\x64\Debug\Course.exe
Выберите опцию: 1
Введите кол-во вершин графа: 4
Меню:
1. Задать кол-во вершин графа
2. Создать граф случайно
3. Создать граф вручную
4. Вычислить минимальное остовное дерево
5. Сохранить результаты в файл
6. Выход
Выберите опцию: 2
Сгенерированный граф:
0 6 1 8
6 0 1 3
1 1 0 4
8 3 4 0

Матрица смежности минимального остовного дерева:
0 0 1 0
0 0 1 3
1 1 0 0
0 3 0 0

Меню:
1. Задать кол-во вершин графа
2. Создать граф случайно
3. Создать граф вручную
4. Вычислить минимальное остовное дерево
5. Сохранить результаты в файл
6. Выход
Выберите опцию:
```

Рисунок 10 - Проверка вручную

Результаты совпали, следовательно программа работает верно.

Заключение

При выполнении данной курсовой работы были получены ценные навыки в разработке многомодульных программ на языке C и ассемблер. Получены базовые навыки программирования на языке C и изучены основные возможности среды программирования Visual Studio 2022. Также были получены навыки отладки и тестирования программ.

В рамках выполнения курсовой работы была разработана игра «змейка», которая представляет собой простое и удобное в использовании приложение. Она легко запускается и не требователен к ресурсам.

В ходе работы возникали некоторые трудности.

В дальнейшем развитии программы есть потенциал для улучшения. Можно добавить более интерактивный интерфейс с различными настройками и опциями. Интерактивный уровень сложности, динамический уровень сложности, улучшение графического интерфейса, введение различных разнообразий в игру, которые могут в корне изменить представления об игре.

Список используемых источников

1. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: Построение и анализ - М.: МЦНМО, 2001. - 960 с.
2. Кристофидес Н. «Теория графов. Алгоритмический подход» - Мир, 1978
3. Герберт Шилдт «Полный справочник по C++» - Вильямс, 2006
4. Уилсон Р. Введение в теорию графов. Пер. с англ. 1977. 208 с.
5. Харви Дейтел, Пол Дейтел. Как программировать на C/C++. 2009 г.
6. З. Оре О. Графы и их применение: Пер. с англ. 1965. 176 с.
7. Давыдов, В.Г. Д 13 Программирование и основы алгоритмизации: Учеб. пособие/В.Г. Давыдов. - М.: Высш. шк., 2003. - 447 с.: ил.
ISBN 5-06-004432-715:14

Приложение А

Листинги программы

Файл «Main.cpp»

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <time.h>
#include <locale.h>

int** createG(int size) {
    int** G = (int**)malloc(size * sizeof(int*));
    for (int i = 0; i < size; i++) {
        G[i] = (int*)malloc(size * sizeof(int));
    }

    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            if (i == j) {
                G[i][j] = 0; // Вес ребра до самой себя равен 0
            }
            else {
                G[i][j] = rand() % 10; // Случайный вес от 1 до 9
            }
            G[j][i] = G[i][j]; // Граф неориентированный
        }
    }
    return G;
}

int** createGManual(int size) {
    int** G = (int**)malloc(size * sizeof(int*));
    for (int i = 0; i < size; i++) {
        G[i] = (int*)malloc(size * sizeof(int));
    }

    printf("Введите веса рёбер (0 если ребра нет):\n");
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            if (i == j) {
                G[i][j] = 0; // Вес ребра до самой себя равен 0
            }
            else {
                printf("Вес ребра %d-%d: ", i, j);
                scanf("%d", &G[i][j]);
            }
            G[j][i] = G[i][j]; // Граф неориентированный
        }
    }
    return G;
}

void printG(int** G, int size) {
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            printf("%d ", G[i][j]);
        }
    }
}
```

```

        printf("\n");
    }
    printf("\n");
}

int minKey(int key[], int mstSet[], int size) {
    int min = INT_MAX, min_index;

    for (int v = 0; v < size; v++)
    {
        if (mstSet[v] == 0 && key[v] < min) //если вершина не была посещена и вес
        рассматриваемой меньше последнего(уже рассмотренной и установленной в min)
        {
            min = key[v]; //значение в мин обновится
            min_index = v; //будет выбран индекс, соответствующий этой вершине с
            наименьшим весом
        }
    }
    return min_index; //вернуть индекс вершины с минимальным весом
}

int** primMST(int** graph, int size) {
    int* parent = (int*)malloc(size * sizeof(int)); // список индексов родителей
    int* key = (int*)malloc(size * sizeof(int)); // список минимальных весов
    int* mstSet = (int*)malloc(size * sizeof(int)); // список посещенных
    (включенных) вершин
    int** mstGraph = (int**)malloc(size * sizeof(int*)); // возвращаемая матрица
    смежности

    // Инициализируем матрицу смежности для MST
    for (int i = 0; i < size; i++) {
        mstGraph[i] = (int*)calloc(size, sizeof(int)); // Инициализируем нулями
    }

    for (int i = 0; i < size; i++) {
        key[i] = INT_MAX;
        mstSet[i] = 0;
    }

    key[0] = 0; // выставляется 0 для первого элемента, так как расстояние равно 0
    parent[0] = -1; // выставляется -1, т.к. у первого элемента нет родителей

    int visitedCount = 0; // счетчик посещенных вершин

    while (visitedCount < size) { // продолжаем, пока не посетим все вершины
        int u = minKey(key, mstSet, size); // u – индекс в строке с минимальным
        весом
        mstSet[u] = 1; // выставляется 1, так как мы посетили эту вершину
        visitedCount++; // увеличиваем счетчик посещенных вершин

        for (int v = 0; v < size; v++) {
            if (graph[u][v] && mstSet[v] == 0 && graph[u][v] < key[v]) { // если в
            исходном графе по минимальному индексу столбца и очередному индексу строки И эта
            вершина не рассматривалась ранее (MstSet) И значение в этой рассматриваемой ячейке
            меньше минимального пути
                parent[v] = u; // значение очередного родителя будет перезаписано,
            индексом минимального ребра
                key[v] = graph[u][v]; // по очередному индексу ребра будет записан
            вес ребра
            }
        }
    }
}

```

```

    }

    // Заполнение матрицы смежности для MST
    for (int i = 1; i < size; i++) {
        mstGraph[parent[i]][i] = graph[parent[i]][i];
        mstGraph[i][parent[i]] = graph[parent[i]][i];
    }

    free(parent);
    free(key);
    free(mstSet);

    return mstGraph; // Возвращаем матрицу смежности для MST
}

void saveToFile(int** graph, int size, const char* filename) {
    FILE* file = fopen(filename, "w");
    if (file == NULL) {
        printf("Ошибка открытия файла для записи.\n");
        return;
    }

    fprintf(file, "%d\n", size);
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            fprintf(file, "%d ", graph[i][j]);
        }
        fprintf(file, "\n");
    }

    fclose(file);
    printf("Результаты сохранены в файл %s.\n", filename);
}

int main() {
    srand(time(NULL)); // Инициализация генератора случайных чисел
    setlocale(LC_ALL, "RU");

    int size = 0;
    char option;
    int** lastGraph = NULL; // Для хранения последнего созданного графа
    int** lastMST = NULL; // Для хранения последнего минимального остовного дерева

    while (1) {
        printf("Меню:\n");
        printf("1. Задать кол-во вершин графа\n");
        printf("2. Создать граф случайно\n");
        printf("3. Создать граф вручную\n");
        printf("4. Вычислить минимальное остовное дерево\n");
        printf("5. Сохранить результаты в файл\n");
        printf("6. Выход\n");
        printf("Выберите опцию: ");
        scanf(" %c", &option);
        {
            switch (option) {
                case '1':
                    printf("Введите кол-во вершин графа: ");
                    scanf("%d", &size);
                    break;

                case '2': {

```

```

    if (size <= 0) {
        printf("Сначала задайте кол-во вершин графа.\n");
        break;
    }
    lastGraph = createG(size);
    printf("Сгенерированный граф:\n");
    printG(lastGraph, size);
    lastMST = primMST(lastGraph, size);
    break;
}

case '3': {
    if (size <= 0) {
        printf("Сначала задайте кол-во вершин графа.\n");
        break;
    }
    lastGraph = createGManual(size);
    printf("Введенный граф:\n");
    printG(lastGraph, size);
    lastMST = primMST(lastGraph, size);
    break;
}

case '4': {
    if (lastGraph == NULL) {
        printf("Сначала создайте граф.\n");
        break;
    }
    lastMST = primMST(lastGraph, size);
    printf("Матрица смежности минимального остовного дерева:\n");
    printG(lastMST, size);
    break;
}

case '5': {
    if (lastGraph == NULL) {
        printf("Сначала создайте граф.\n");
        break;
    }
    printf("Сохранить в файл (введите имя файла): ");
    char filename[100];
    scanf("%s", filename);
    saveToFile(lastMST, size, filename);
    break;
}

case '6':
    // Освобождение памяти перед выходом
    if (lastGraph != NULL) {
        for (int i = 0; i < size; i++) {
            free(lastGraph[i]);
        }
        free(lastGraph);
    }
    if (lastMST != NULL) {
        for (int i = 0; i < size; i++) {
            free(lastMST[i]);
        }
        free(lastMST);
    }
    exit(0);
}

```

```
        default:  
            printf("Неверный выбор, попробуйте снова.\n");  
        }  
    }  
}  
return 0;  
}
```

Приложение В

Окна программы

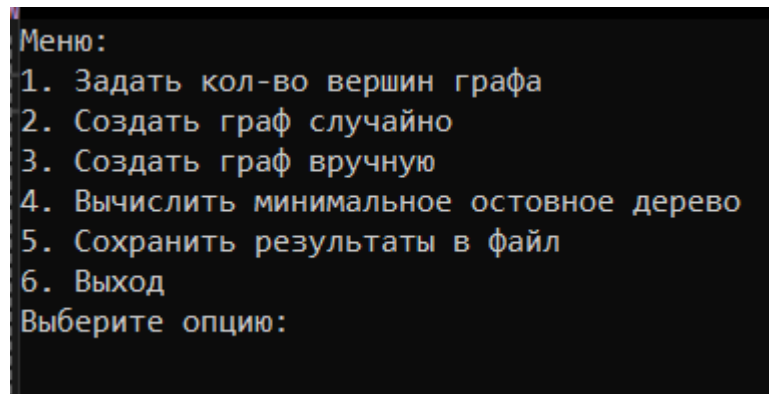


Рисунок 11 - Главное меню программы

```
D:\Edu\2 курс\Логика и основы алгор...
Меню:
1. Задать кол-во вершин графа
2. Создать граф случайно
3. Создать граф вручную
4. Вычислить минимальное остовное дерево
5. Сохранить результаты в файл
6. Выход
Выберите опцию: 1
Введите кол-во вершин графа: 6
Меню:
1. Задать кол-во вершин графа
2. Создать граф случайно
3. Создать граф вручную
4. Вычислить минимальное остовное дерево
5. Сохранить результаты в файл
6. Выход
Выберите опцию: 2
Сгенерированный граф:
0 5 7 2 0 5
5 0 8 2 0 9
7 8 0 0 4 0
2 2 0 0 9 2
0 0 4 9 0 4
5 9 0 2 4 0

Матрица смежности минимального остовного дерева:
0 0 0 2 0 0
0 0 0 2 0 0
0 0 0 0 4 0
2 2 0 0 0 2
0 0 4 0 0 4
0 0 0 2 4 0
```

Рисунок 12 - Результат работы программы