

Министерство образования Российской Федерации
Пензенской государственный университет
Кафедра «Вычислительная техника»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К курсовому проектированию

По курсу «Логика и основы алгоритмизации в инженерных задачах»
на тему «Реализация алгоритма Форда-Беллмана»

20.12.23

О.И.И.И.
Ф.И.О.

Выполнил:

ст. гр. 22ВВС1

Макеева Д.Д.

Принял:

к.т.н. Юрова О.В.

Пенза 2023

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет Вычислительной техники

Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ _____

« ____ » _____ 20 ____

ЗАДАНИЕ

на курсовое проектирование по курсу

Логика и основы алгоритмизации в инженерных задачах
Студенту Мамсевой О.О. Группа 22 ВВСи
Тема проекта Реализация алгоритма Форда-Беллмана

Исходные данные (технические требования) на проектирование

Разработка алгоритмов и программного обеспечения в соответствии с данными задания курсового проекта.
Пояснительная записка должна содержать:

1. Постановку задачи;
2. Теоретическую часть задания;
3. Описание алгоритма поставленной задачи;
4. Пример ручной расчёта задачи и вычислений (на небольшом участке работы алгоритма);
5. Описание самой программы;
6. Тесты;
7. Список литературы;
8. Листинг программы;
9. 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 Исследование теоретической задачи курсового
- 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 2023г.

Дата защиты проекта " " "

Руководитель Юрова О.В. *[подпись]*

Задание получил "26" сентября 2023 г.

Студент Манеева Д.Д. *[подпись]*

Содержание

Реферат	5
Введение	6
Постановка задачи.....	7
Теоретическая часть задания	8
Описание алгоритма программы.....	10
Описание программы	13
Тестирование	18
Ручной расчёт	24
Заключение	28
Список литературы	30
ПриложениеА. Листинг программы.....	31

Реферат

Отчет 44 стр, 20 рисунков, 2 таблицы.

ГРАФ, ТЕОРИЯ ГРАФОВ, АЛГОРИТМ ФОРДА-БЕЛЛМАНА,
РАССТОЯНИЕ.

Цель исследования – разработка программы, способная найти кратчайшие пути до всех вершин от исходной, используя алгоритм Форда-Беллмана.

В работе рассмотрен алгоритм Форда-Беллмана. Установлено, что с помощью данного алгоритма можно проверять наличие отрицательного цикла и восстановление пути.

Введение

Алгоритм был разработан в 1956 году Ричардом Беллманом и Лестером Фордом. Алгоритм Беллмана-Форда - это алгоритм кратчайшего пути с одним источником, который определяет кратчайший путь между данной исходной вершиной и остальными вершинами во взвешенном графе.

Основной принцип алгоритма Беллмана-Форда заключается в том, что он начинается с одного источника и вычисляет расстояние до каждой вершины. Он основывается на пошаговом обновлении расстояний до каждой вершины итеративным способом.

Расстояние изначально неизвестно и предполагается бесконечным, но со временем алгоритм ослабляет эти пути, определяя несколько более коротких путей. Следовательно, говорят, что алгоритм Беллмана-Форда основан на “Принципе релаксации”.

В качестве среды разработки мною была выбрана среда Microsoft Visual Studio 2022, язык программирования – Си.

Целью данной курсовой работы является разработка программы на языке Си, который является широко используемым. Именно с его помощью в данном курсовом проекте реализуется алгоритм Форда-Беллмана, осуществляющий поиск кратчайших путей в орграфе.

Постановка задачи

Требуется разработать программу, которая реализует алгоритм Форда-Беллмана, находя кратчайшие пути от вершины src до остальных.

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 на экран должна выводиться матрица смежности орграфа. Необходимо предусмотреть различные исходы поиска, чтобы программа не выдавала ошибок и работала правильно.

Устройство ввода – клавиатура и мышь.

Задание выполняется в соответствии с вариантом №14.

Теоретическая часть задания

Граф G (рисунок 1) задается множеством вершин X_1, X_2, \dots, X_n и множеством ребер, соединяющих между собой определенные вершины. Ребра из множества A ориентированы, что показывается стрелкой, которая указывает достижимость данной вершины, граф с такими ребрами называется ориентированным графом.

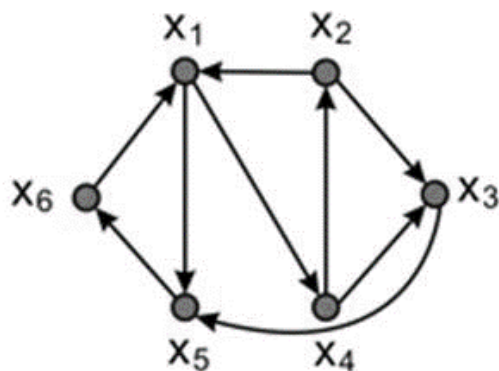


Рисунок 1 – Пример орграфа

При представлении графа матрицей смежности информация о ребрах графа хранится в квадратной матрице, где присутствие пути из одной вершины в другую обозначается весом ребра, иначе нулем.

Алгоритм Беллмана–Форда — это алгоритм, который вычисляет кратчайший путь от одной исходной вершины ко всем остальным вершинам.

Данный алгоритм применим, к графам с взвешенными рёбрами. Он допускает рёбра с отрицательным весом. Длиной пути является сумма весов рёбер, входящих в этот путь.

Основная идея алгоритма заключается в том, чтобы на каждой итерации рассматривать все ребра графа и обновлять расстояния до каждой вершины, если найден более короткий путь через текущую вершину. Алгоритм продолжает обновлять расстояния до тех пор, пока не будет достигнуто оптимальное решение.

Стоит отметить, что кратчайших путей может не существовать. Так, в графе, содержащем отрицательный цикл, существует сколь угодно короткий путь от одной вершины этого цикла до другой (каждый обход цикла уменьшает длину пути).

Алгоритм Форда-Беллмана может быть использован для решения различных задач, таких как, поиск отрицательных циклов, определение наличия пути между вершинами, восстановление пути.

Описание алгоритма программы

Функция `bellmanFord` реализует алгоритм Беллмана-Форда для поиска кратчайших путей в графе. Он принимает указатель на структуру "Graph" и исходную вершину "src" в качестве аргументов.

Сначала уменьшаем номер начальной вершины на 1, так как нумерация вершин обычно начинается с 1, а в программе используется индексация с 0. Затем создаем переменную "v" и присваиваем ей значение количества вершин в графе.

Затем функция выделяет память для массивов "dist" - массив расстояний до вершин и "path" - массив для хранения пути до каждой вершины размером V, где V - количество вершин в графе. Затем она инициализирует все элементы массива "dist" значением "INT_MAX" (бесконечность) и все элементы массива path значением -1, так как первоначально нет предшествующих вершин.

Затем функция устанавливает значение "dist[src]" равным 0.

Проходимся по всем вершинам графа, кроме начальной, и обновляем расстояние и путь, если находим более короткий путь. Вложенный цикл выполняет обход всех ребер графа и проверяет, можно ли улучшить текущее значение. Если расстояние до вершины "k" через вершину "j" меньше, чем текущее расстояние до "k", обновляем значения "dist[k]" и "path[k]" (то есть выполняется условие: $\text{graph} \rightarrow \text{Matrix}[j][k] \neq 0$ и $\text{dist}[j] \neq \text{INT_MAX}$ и $\text{dist}[j] + \text{graph} \rightarrow \text{Matrix}[j][k] < \text{dist}[k]$). Значение path[k] устанавливается равным j (предыдущая вершина на пути к k).

После завершения циклов релаксации следует проверка на наличие отрицательных циклов. Для этого снова выполняется проход по графу. Если найдено ребро (i, j) и текущее расстояние до "i" плюс вес ребра меньше, чем текущее расстояние до "j" (то есть $\text{graph} \rightarrow \text{Matrix}[i][j] \neq 0$ и $\text{dist}[i] \neq \text{INT_MAX}$ и $\text{dist}[i] + \text{graph} \rightarrow \text{Matrix}[i][j] < \text{dist}[j]$), значит, в графе существует отрицательный цикл. В этом случае выводим сообщение об обнаружении отрицательного цикла, освобождаем выделенную память для массивов dist и path и завершаем выполнение функции.

В конце функции, если отрицательных циклов не обнаружено, она заканчивается выводом кратчайших путей и запросом на сохранение результата в файл. Для недостижимых вершин расстояние остаётся равным бесконечности.

Ниже представлен псевдокод функции FordBellman ().

FordBellman ()

1. Инициализация src(стартовая вершина)=-1;
2. Инициализация v=количество вершин в графе;
3. Инициализация dist-массив расстояний до вершин размером V;
4. Инициализация path-массив для хранения пути до каждой вершины размером V;
5. Для i=0, пока i<v делать i=i+1
 6. Инициализация dist[i]=INT_MAX ("бесконечность");
 7. Инициализация path[i]=-1;
8. Конец цикла.
9. Инициализация dist[src] = 0;
10. Для i=1, пока i<v делать i=i+1
 11. Для j=0, пока j<v делать j=j+1
 12. Для k=0, пока k<v делать k=k+1
 13. Если graph->Matrix[j][k](вес ребра)!= 0 && dist[j] != INT_MAX && dist[j] + graph->Matrix[j][k] < dist[k](текущее значение)
 14. dist[k]=dist[j] + graph->Matrix[j][k];
 15. path[k]=j;
16. Конец условия и цикла.
17. Для i=0, пока i<v делать i=i+1
 18. Для j=0, пока j<v делать j=j+1
 19. Если graph->Matrix[i][j] != 0 && dist[i] != INT_MAX && dist[i] + graph->Matrix[i][j] < dist[j]
 20. Вывод "В графе обнаружен отрицательный цикл.";
 21. Освобождение памяти dist;
 22. Освобождение памяти path;
 23. Завершение функции;
24. Конец условия и цикла.

- 25. Вывод результата;
- 26. Освобождение памяти dist;
- 27. Освобождение памяти path.

Полный код программы можно увидеть в Приложении А.

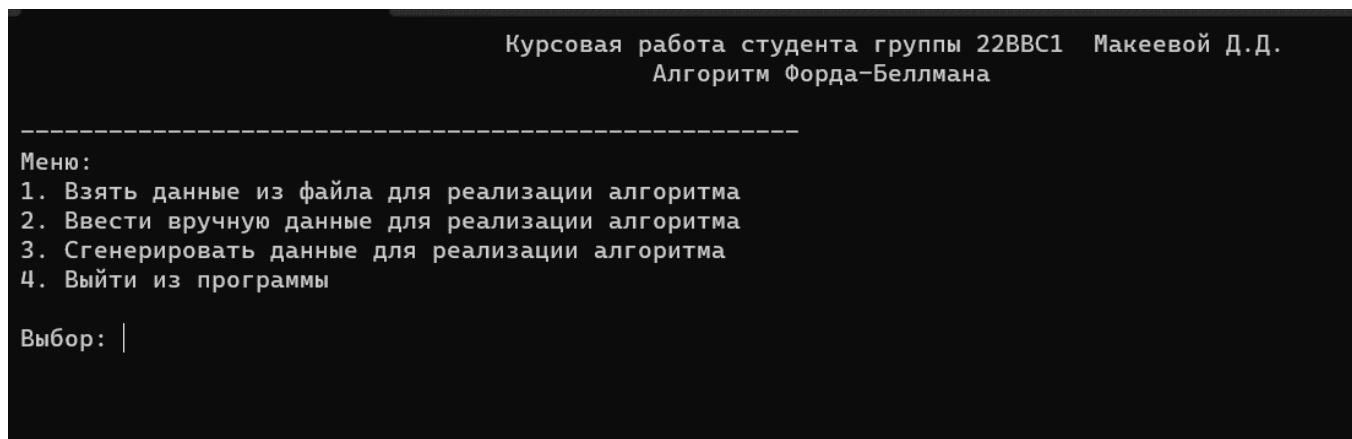
Описание программы

Для написания данной программы использован язык программирования Си. Язык программирования Си - универсальный язык программирования, который завоевал особую популярность у программистов, благодаря сочетанию возможностей языков программирования высокого и низкого уровней.

Проект был создан в виде консольного приложения Win32 (Visual C++).

Работа программы начинается с запроса генерации матрицы. Пользователю даётся на выбор 3 действия. Для выбора действия пользователю нужно нажать соответствующую клавишу на клавиатуре.

1. Ввести матрицу вручную.
2. Прочитать матрицу из файла.
3. Сгенерировать случайную матрицу.

A screenshot of a Windows console window with a black background and white text. At the top, it displays the title 'Курсовая работа студента группы 22BBS1 Макеевой Д.Д.' followed by 'Алгоритм Форда-Беллмана'. Below this is a horizontal line of dashes. The text 'Меню:' is followed by a numbered list: '1. Взять данные из файла для реализации алгоритма', '2. Ввести вручную данные для реализации алгоритма', '3. Сгенерировать данные для реализации алгоритма', and '4. Выйти из программы'. At the bottom, the text 'Выбор: |' is shown with a vertical cursor bar.

```
Курсовая работа студента группы 22BBS1 Макеевой Д.Д.  
Алгоритм Форда-Беллмана  
-----  
Меню:  
1. Взять данные из файла для реализации алгоритма  
2. Ввести вручную данные для реализации алгоритма  
3. Сгенерировать данные для реализации алгоритма  
4. Выйти из программы  
Выбор: |
```

Рисунок 2 – Выбор способа ввода матрицы

```
printf("\nВыберете какой граф будете вводить\n");  
printf("1 - неориентированный\n");  
printf("0 - ориентированный\n");  
printf("Выбор: ")
```

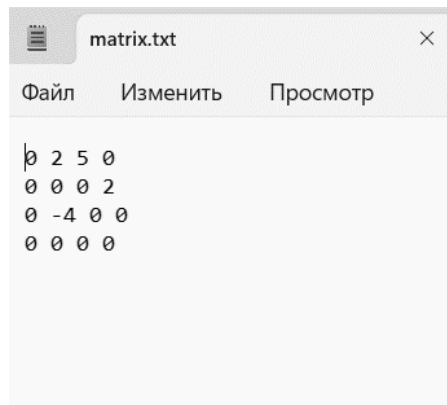


Рисунок 3 – Файл, из которого считывается матрица

```

Меню:
1. Взять данные из файла для реализации алгоритма
2. Ввести вручную данные для реализации алгоритма
3. Сгенерировать данные для реализации алгоритма
4. Выйти из программы

Выбор: 1

Предупреждение! Если в файл введён неориентированный граф, то проверьте, чтобы в нём не было отрицательных весов.
В противном случае всегда будет отрицательный цикл.

Каким файлом воспользоваться?
1 – если данные в файл введены вручную (matrix.txt)
0 – если данные в файл сохранены при генерации (matrix2.txt)
Выбор: 1

Матрица:
0  2  5  0
0  0  0  2
0 -4  0  0
0  0  0  0

```

Рисунок 4 – Ввод матрицы через файл

```

void readGraphFromFile(struct Graph* graph) {

    FILE* file;
    errno_t err;

    if ((err = fopen_s(&file, "matrix.txt", "r")) != 0) {
        perror("\nОшибка при открытии файла. Проверьте существует ли файл.\n Для
        правильного заполнения вводите значения через пробел (0 – ребра нет), количество вершин
        = количеству вершин");
        exit(0);
    }

    // Подсчет количества вершин (по числу строк в файле)
    int vertexCount = 0;
    char c;
    while ((c = fgetc(file)) != EOF) {
        if (c == '\n') {
            vertexCount++;
        }
    }
    vertexCount++;

    // Переходим к началу файла и считаем количество рёбер (ненулевых элементов в
    матрице)
    fseek(file, 0, SEEK_SET);
    int edgeCount = 0;
    int value;

```

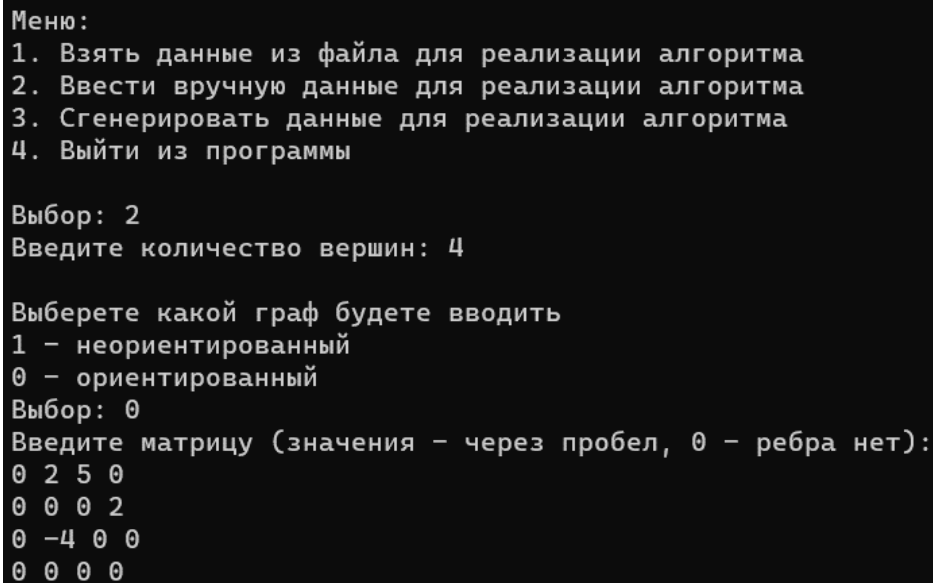


```

// Считываем матрицу смежности
graph->V = vertexCount;
graph->E = 0;
graph->Matrix = (int**)malloc(graph->V * sizeof(int*));

for (int i = 0; i < graph->V; i++) {
    graph->Matrix[i] = (int*)malloc(graph->V * sizeof(int));
    for (int j = 0; j < graph->V; j++) {
        if (fscanf_s(file, "%d", &value) == 1) {
            graph->Matrix[i][j] = value;
            if (value != 0) {
                graph->E++;
            }
        }
        // Пропускаем пробельные символы
        while ((c = fgetc(file)) != EOF && (c == ' ' || c == '\t')) {}
        // Возвращаем символ, если это не конец файла
        if (c != EOF) {
            ungetc(c, file);
        }
    }
}
// Отладочный вывод матрицы
printf("\nМатрица:\n");
for (int i = 0; i < graph->V; i++) {
    for (int j = 0; j < graph->V; j++) {
        printf("%4d ", graph->Matrix[i][j]);
    }
    printf("\n");
}
fclose(file);
}

```



```

Меню:
1. Взять данные из файла для реализации алгоритма
2. Ввести вручную данные для реализации алгоритма
3. Сгенерировать данные для реализации алгоритма
4. Выйти из программы

Выбор: 2
Введите количество вершин: 4

Выберете какой граф будете вводить
1 – неориентированный
0 – ориентированный
Выбор: 0
Введите матрицу (значения – через пробел, 0 – ребра нет):
0 2 5 0
0 0 0 2
0 -4 0 0
0 0 0 0

```

Рисунок 5 – Ручной ввод матрицы

```

printf("Введите матрицу (значения – через пробел, 0 – ребра нет):\n");

for (int i = 0; i < graph->V; i++) {
    for (int j = 0; j < graph->V; j++) {
        scanf_s("%d", &(graph->Matrix[i][j]));
    }
}
for (int i = 0; i < graph->V; i++) {
    for (int j = i + 1; j < graph->V; j++) {

```

```

        if (graph->Matrix[i][j] == graph->Matrix[j][i] && graph->Matrix[i][j] != 0) {
            printf("Граф является неориентированным. Повторите ввод\n");
            manualInput(graph);
        }
    }
}

```

```

Меню:
1. Взять данные из файла для реализации алгоритма
2. Ввести вручную данные для реализации алгоритма
3. Сгенерировать данные для реализации алгоритма
4. Выйти из программы

Выбор: 3

Выберете какой граф будете вводить
1 - неориентированный
0 - ориентированный
Выбор: 0

Граф ориентированный
Ведите количество вершин: 4

Ведите количество рёбер: 6

Матрица:
  0   0  -5   0
 -8   0  -8   0
  0   4   0  -7
  0  -9   0   0

```

Рисунок 6 – Случайная генерация матрицы

```

printf("\nГраф ориентированный\n ");
// Запрос пользователя о количестве вершин
printf("Ведите количество вершин: ");
scanf_s("%d", &graph->V);

// Запрос пользователя о количестве рёбер
printf("\nВедите количество рёбер: ");
scanf_s("%d", &value);
while (value > graph->V * (graph->V - 1) / 2) {
    printf("Количество ребер превышает максимально возможное для данного числа вершин в графе.\nПопробуйте ещё раз:");
    scanf_s("%d", &value);
}
graph->E = value;

// Инициализация генератора случайных чисел
srand(time(NULL));

// Выделение памяти для матрицы смежности
graph->Matrix = (int**)malloc(graph->V * sizeof(int*));
// Инициализация матрицы нулями
for (int i = 0; i < graph->V; i++) {
    graph->Matrix[i] = (int*)malloc(graph->V * sizeof(int));
    for (int j = 0; j < graph->V; j++) {
        graph->Matrix[i][j] = 0;
    }
}

// Заполнение рёбер
while (edgeCount < graph->E) {

    int i = rand() % graph->V;

```

```

    int j = rand() % graph->V;
    if (i != j && graph->Matrix[i][j] == 0) {
        graph->Matrix[i][j] = rand() % 21 - 10;
        edgeCount++;
    }
}

// Отладочный вывод матрицы
printf("\nМатрица:\n");
for (int i = 0; i < graph->V; i++) {
    for (int j = 0; j < graph->V; j++) {
        printf("%4d ", graph->Matrix[i][j]);
    }
    printf("\n");
}

```

Тестирование

Среда разработки Microsoft Visual Studio 2022 предоставляет все средства, необходимые при разработке и отладке многомодульной программы.

Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы. В ходе тестирования было выявлено и исправлено множество проблем, связанных с вводом данных, изменением дизайна выводимых данных, алгоритмом программы, взаимодействием функций.

Ниже продемонстрирован результат тестирования программы.

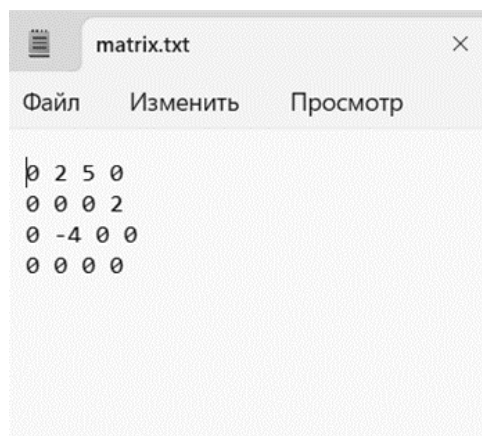


Рисунок 7 – Файл, из которого считывается матрица

```

Меню:
1. Взять данные из файла для реализации алгоритма
2. Ввести вручную данные для реализации алгоритма
3. Сгенерировать данные для реализации алгоритма
4. Выйти из программы

Выбор: 1

Предупреждение! Если в файл введён неориентированный граф, то проверьте, чтобы в нём не было отрицательных весов.
В противном случае всегда будет отрицательный цикл.

Каким файлом воспользоваться?
1 - если данные в файл введены вручную (matrix.txt)
0 - если данные в файл сохранены при генерации (matrix2.txt)
Выбор: 1

Матрица:
0  2  5  0
0  0  0  2
0 -4  0  0
0  0  0  0

Введите исходную вершину (от 1 до 4): 1

Кратчайшие пути от вершины 1:
До вершины 1: 0 (1)
До вершины 2: 1 (2 <- 3 <- 1)
До вершины 3: 5 (3 <- 1)
До вершины 4: 3 (4 <- 2 <- 3 <- 1)
Хотите ли сохранить результат в файл? (1, если да, и 0, если нет): 1
Результат сохранён в файл 'output.txt'.

```

Рисунок 8 – Тестирование считывания из файла

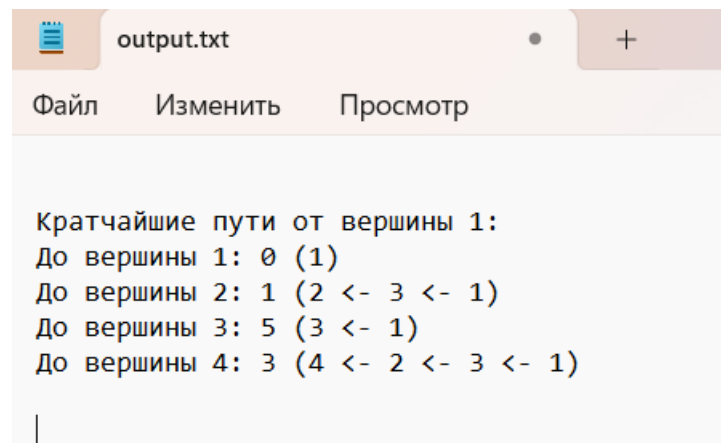


Рисунок 9 – Файл, с охранным результатом

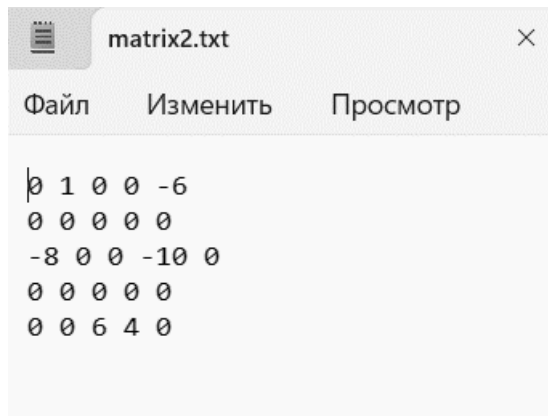


Рисунок 10 – Файл, из которого считывается случайно сгенерированная матрица

```
Меню:
1. Взять данные из файла для реализации алгоритма
2. Ввести вручную данные для реализации алгоритма
3. Сгенерировать данные для реализации алгоритма
4. Выйти из программы

Выбор: 1

Предупреждение! Если в файл введён неориентированный граф, то проверьте, чтобы в нём не было отрицательных весов.
В противном случае всегда будет отрицательный цикл.

Каким файлом воспользоваться?
1 – если данные в файл введены вручную (matrix.txt)
0 – если данные в файл сохранены при генерации (matrix2.txt)
Выбор: 0

Матрица:
0 1 0 0 -6
0 0 0 0 0
-8 0 0 -10 0
0 0 0 0 0
0 0 6 4 0

Введите исходную вершину (от 1 до 5): 2

Кратчайшие пути от вершины 2:
До вершины 1: INFINITY
До вершины 2: 0 (2)
До вершины 3: INFINITY
До вершины 4: INFINITY
До вершины 5: INFINITY
Хотите ли сохранить результат в файл? (1, если да, и 0, если нет): 0
Результат не сохранён.
```

Рисунок 11 – Тестирование считывания случайно сгенерированной матрицы

```

Меню:
1. Взять данные из файла для реализации алгоритма
2. Ввести вручную данные для реализации алгоритма
3. Сгенерировать данные для реализации алгоритма
4. Выйти из программы

Выбор: 2
Введите количество вершин: 4

Выберете какой граф будете вводить
1 – неориентированный
0 – ориентированный
Выбор: 1
Вводите положительные веса
Введите матрицу (значения – через пробел, 0 – ребра нет):
0 3 6 0
3 0 5 0
6 5 0 1
0 0 1 0
Введите исходную вершину (от 1 до 4): 1

Кратчайшие пути от вершины 1:
До вершины 1: 0 (1)
До вершины 2: 3 (2 <- 1)
До вершины 3: 6 (3 <- 1)
До вершины 4: 7 (4 <- 3 <- 1)

```

Рисунок 12 – Тестирование ручного ввода матрицы (неориентированный граф)

```

Меню:
1. Взять данные из файла для реализации алгоритма
2. Ввести вручную данные для реализации алгоритма
3. Сгенерировать данные для реализации алгоритма
4. Выйти из программы

Выбор: 2
Введите количество вершин: 4

Выберете какой граф будете вводить
1 – неориентированный
0 – ориентированный
Выбор: 0
Введите матрицу (значения – через пробел, 0 – ребра нет):
0 -3 6 0
0 0 5 0
0 0 0 10
0 1 0 0
Введите исходную вершину (от 1 до 4): 2

Кратчайшие пути от вершины 2:
До вершины 1: INFINITY
До вершины 2: 0 (2)
До вершины 3: 5 (3 <- 2)
До вершины 4: 15 (4 <- 3 <- 2)

```

Рисунок 13 – Тестирование ручного ввода матрицы (ориентированный граф)


```

Меню:
1. Взять данные из файла для реализации алгоритма
2. Ввести вручную данные для реализации алгоритма
3. Сгенерировать данные для реализации алгоритма
4. Выйти из программы

Выбор: 3

Выберете какой граф для генерации
1 – неориентированный
0 – ориентированный
Выбор: 0

Граф ориентированный
Ведите количество вершин: 5

Ведите количество рёбер: 8

Матрица:
0 0 0 -4 2
0 0 10 0 0
0 -8 0 0 0
9 0 0 0 0
0 5 7 -2 0

Сохранить ли матрицу в файл? (1 – да/ 0 – продолжить без сохранения): 1
Матрица сохранена в файл 'matrix2.txt'.

Введите исходную вершину (от 1 до 5): 1

Кратчайшие пути от вершины 1:
До вершины 1: 0 (1)
До вершины 2: 1 (2 <- 3 <- 5 <- 1)
До вершины 3: 9 (3 <- 5 <- 1)
До вершины 4: -4 (4 <- 1)
До вершины 5: 2 (5 <- 1)

```

Рисунок 14 – Тестирование случайной генерации матрицы
(ориентированный граф)

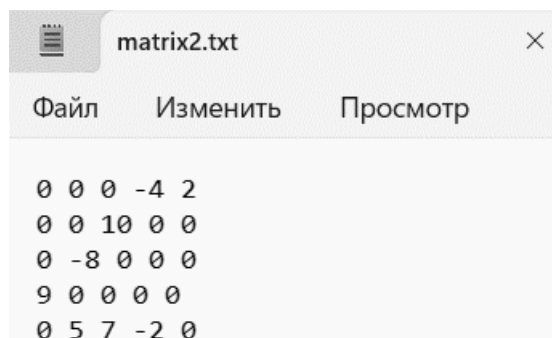


Рисунок 15 – Файл с сохранённой случайно сгенерированной матрицей

```

Меню:
1. Взять данные из файла для реализации алгоритма
2. Ввести вручную данные для реализации алгоритма
3. Сгенерировать данные для реализации алгоритма
4. Выйти из программы

Выбор: 3

Выберете какой граф для генерации
1 – неориентированный
0 – ориентированный
Выбор: 1

Граф неориентированный

Ведите количество вершин: 4

Ведите количество рёбер: 6

Матрица:
  0   3   0   8
  3   0   7   5
  0   7   0   2
  8   5   2   0

Сохранить ли матрицу в файл? (1 – да/ 0 – продолжить без сохранения): 0

Введите исходную вершину (от 1 до 4): 1

Кратчайшие пути от вершины 1:
До вершины 1: 0 (1)
До вершины 2: 3 (2 <- 1)
До вершины 3: 10 (3 <- 2 <- 1)
До вершины 4: 8 (4 <- 1)
Хотите ли сохранить результат в файл? (1, если да, и 0, если нет): 0
Результат не сохранён.

```

Рисунок 16 – Тестирование случайной генерации матрицы
(неориентированный граф)

Ниже приведена таблица, которая описывает поведение программы при тестировании.

Описание теста	Ожидаемый результат	Полученный результат
Запуск программы	Вывод сообщения о выборе: 1. Загрузить из файла 2. Ручной ввод 3. Случайная генерация	Верно
Считывание матрицы из файла	Вывод матрицы с данными из файла	Верно
Ручной ввод матрицы	Вывод меню с выбором графа. Вывод матрицы с введенными данными	Верно
Случайная генерация матрицы	Вывод меню с выбором графа. Вывод случайно сгенерированной матрицы	Верно
Сохранение случайно сгенерированной матрицы в файл	Наличие в файле случайно сгенерированной матрицы	Верно
Сохранение результата в файл	Наличие в файле результата	Верно

Таблица 1 – Описание поведения программы при тестировании

В результате тестирования было выявлено, что программа успешно проверяет данные на соответствие необходимым требованиям.

Ручной расчёт

Проведем проверку программы посредством ручных вычислений на примере взвешенного ориентированного графа с 5 вершинами (рисунок 2).

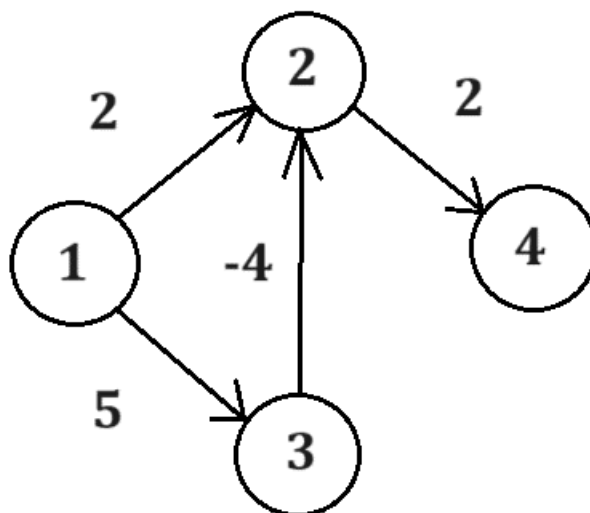


Рисунок 17 – Орграф

Мы получаем следующие расстояния, приведённые в таблице ниже.

	1	2	3	4
вес	0	∞	∞	∞
пред. верш.	0	-	-	-
вес	0	2	5	∞
пред. верш.	0	1	1	-
вес	0	1	5	4
пред. верш.	0	3	3	2
вес	0	1	5	3
пред. верш.	0	3	3	2
вес	0	1	5	3
пред. верш.	0	3	3	2

Таблица 2 – Результат (рисунок 17)

```

Матрица:
0  2  5  0
0  0  0  2
0 -4  0  0
0  0  0  0
Введите исходную вершину (от 1 до 4): 1

Кратчайшие пути от вершины 1:
До вершины 1: 0 (1)
До вершины 2: 1 (2 <- 3 <- 1)
До вершины 3: 5 (3 <- 1)
До вершины 4: 3 (4 <- 2 <- 3 <- 1)

```

Рисунок 18 – Тестирование ручного расчёта

Используя алгоритм Беллмана-Форда, мы можем определить, есть ли отрицательный цикл в графе.

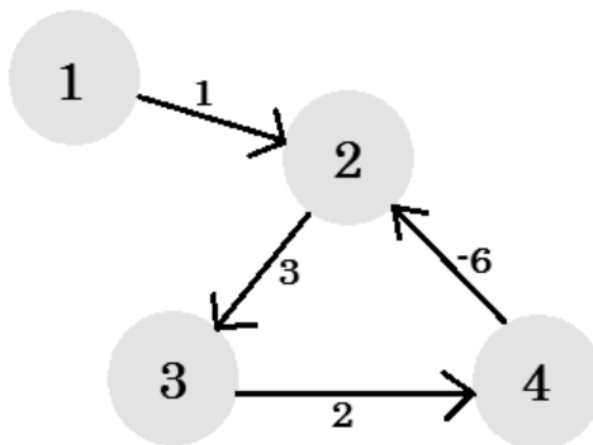


Рисунок 19 – Орграф с отрицательным циклом

Выберем вершину 1 как стартовую. После применения ранее разобранный алгоритм Беллмана-Форда к графу мы выясним расстояния от стартовой вершины до всех остальных вершин.

Вот как выглядит граф после $(N - 1) = 3$ итераций (рисунок 6). Это должно быть результатом, так как существует 4 ребра, нам нужно не более трех итераций, чтобы узнать кратчайший путь. После $(N - 1)$ итераций мы делаем еще одну заключительную итерацию, и если расстояние продолжает уменьшаться, это означает, что в графе определён есть цикл отрицательного веса.

В этом примере: если мы проверим путь 2-3, $d[2] + \text{вес}[2][3]$ даст нам 1, что меньше, чем $d[3] = 2$. Поэтому мы можем заключить, что на нашем графе есть отрицательный цикл.

```
Введите матрицу (значения – через пробел, 0 – ребра нет):  
0 1 0 0  
0 0 3 0  
0 0 0 2  
0 -6 0 0  
Введите исходную вершину (от 1 до 4): 1  
В графе обнаружен отрицательный цикл.
```

Рисунок 20 – Тестирование ручного расчёта (с отрицательным циклом)

Таким образом, можно сделать вывод, что программа работает верно.

Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм Форда-Беллмана для поиска кратчайших путей от вершины `str` до всех остальных вершин, запоминает путь до этих вершин, а также находит циклы отрицательного веса при их наличии в Microsoft Visual Studio 2022.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания матриц смежностей, основанных на теории орграфов. Приобретены навыки по осуществлению алгоритма Форда-Беллмана. Углублены знания языка программирования Си.

Недостатком разработанной программы является примитивный пользовательский интерфейс. Потому что программа работает в консольном режиме, не добавляющем к сложности языка сложность программного оконного интерфейса.

Программа имеет небольшой, но достаточный для использования функционал возможностей.

Список литературы

1. Metod_PrilTeorgrafvzi_100504_29122016.pdf
2. Рафгарден Тим: Совершенный алгоритм. Алгоритмы для NP-трудных задач. — СПб.: Питер, 2021.
3. Писарук, Н. Н.: Исследование операций / Н. Н. Писарук. — Минск : БГУ, 2015.
4. Солдатов А.И.: Основы программирования на языке СИ / А.И. Солдатов.М; Томский политехнический университет. – Томск: Изд-во Томского политехнического университета, 2015
5. https://neerc.ifmo.ru/wiki/index.php?title=Алгоритм_Форда-Беллмана.
6. З. Оре О. Графы и их применение: Пер. с англ. 1965. 176 с.

Приложение А. Листинг программы.

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

// Структура для представления графа
struct Graph {
    int V, E;
    int** Matrix;
};

//Список всех функций
void saveResultsToFile(struct Graph* graph, int* dist, int* path, int src);
void saveMatrixToFile(struct Graph* graph);
void FordBellman(struct Graph* graph, int src);
void manualInput(struct Graph* graph);
void readGraphFromFile(struct Graph* graph);
void generateRandomGraph(struct Graph* graph);

// Функция для сохранения результатов в файл
void saveResultsToFile(struct Graph* graph, int* dist, int* path, int src) {

    FILE* file;
    errno_t err;

    if ((err = fopen_s(&file, "output.txt", "a")) != 0) {
        perror("Ошибка при открытии файла");
        exit(0);
    }

    fprintf(file, "Кратчайшие пути от вершины %d:\n", src + 1);
    for (int i = 0; i < graph->V; i++) {
        if (dist[i] == INT_MAX)
            fprintf(file, "До вершины %d: INFINITY\n", i + 1);
        else {
            fprintf(file, "До вершины %d: %d (%d", i + 1, dist[i], i + 1);
```

```

        int vertex = i;
        while (path[vertex] != -1) {
            fprintf(file, " <- %d", path[vertex] + 1);
            vertex = path[vertex];
        }
        fprintf(file, "\n");
    }
    fprintf(file, "\n");
    fclose(file);
}

```

// Функция для сохранения матрицы в файл

```

void saveMatrixToFile(struct Graph* graph) {

    FILE* file;
    errno_t err;

    if ((err = fopen_s(&file, "matrix2.txt", "w")) != 0) {
        perror("Ошибка при открытии файла");
        exit(0);
    }

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

```

// Функция для выполнения алгоритма Форда-Беллмана

```

void FordBellman(struct Graph* graph, int src) {

    src -= 1;
    int V = graph->V;
    int* dist = (int*)malloc(V * sizeof(int));
    int* path = (int*)malloc(V * sizeof(int));

```

```

    // Инициализация расстояний и пути
    for (int i = 0; i < V; i++) {

```

```

        dist[i] = INT_MAX;
        path[i] = -1;
    }
    dist[src] = 0;

    // Релаксация ребер
    for (int i = 1; i < V; i++) {
        for (int j = 0; j < V; j++) {
            for (int k = 0; k < V; k++) {
                if (graph->Matrix[j][k] != 0 && dist[j] != INT_MAX && dist[j] +
graph->Matrix[j][k] < dist[k]) {
                    dist[k] = dist[j] + graph->Matrix[j][k];
                    path[k] = j;
                }
            }
        }
    }

    // Проверка наличия отрицательных циклов
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            if (graph->Matrix[i][j] != 0 && dist[i] != INT_MAX && dist[i] + graph-
>Matrix[i][j] < dist[j]) {
                printf("В графе обнаружен отрицательный цикл.\n");
                free(dist);
                free(path);
                return;
            }
        }
    }

    // Вывод результатов
    printf("\nКратчайшие пути от вершины %d:\n", src + 1);
    for (int i = 0; i < V; i++) {
        if (dist[i] == INT_MAX)
            printf("До вершины %d: INFINITY\n", i + 1);
        else {
            printf("До вершины %d: %d (%d", i + 1, dist[i], i + 1);
            int vertex = i;

            while (path[vertex] != -1) {
                printf(" <- %d", path[vertex] + 1);
                vertex = path[vertex];
            }
        }
    }

```

```

        }
        printf("\n");
    }
}

// Запрос пользователя о сохранении результатов
int choice;
printf("Хотите ли сохранить результат в файл? (1, если да, и 0, если нет): ");
scanf_s(" %d", &choice);

switch (choice) {
case 0:
    printf("Результат не сохранён.\n");
    break;
case 1:
    saveResultsToFile(graph, dist, path, src);
    printf("Результат сохранён в файл 'output.txt'.\n");
    break;
default:
    printf("Недопустимое значение. Попробуйте ещё раз.\n");
}

free(dist);
free(path);
}

// Функция для ручного ввода матрицы смежности
void manualInput(struct Graph* graph) {

    int choice;
    printf("\nВыберете какой граф будете вводить\n");
    printf("1 - неориентированный\n");
    printf("0 - ориентированный\n");
    printf("Выбор: ");
    scanf_s("%d", &choice);
    switch (choice) {
case 1:
        printf("Вводите положительные веса");

printf("\nВведите матрицу (значения - через пробел, 0 - ребра нет):\n");

        for (int i = 0; i < graph->V; i++) {
            for (int j = 0; j < graph->V; j++) {

```

```

        int value;
        scanf_s("%d", &value);
        while (value < 0){
            printf("Введено отрицательное значение. Повторите ввод:\n");
            scanf_s("%d", &value);
        }
        graph->Matrix[i][j] = value;
    }
}

for (int i = 0; i < graph->V; i++) {
    for (int j = i + 1; j < graph->V; j++) {
        if (graph->Matrix[i][j] != graph->Matrix[j][i]) {
            printf("Граф является ориентированным. Повторите ввод\n");
            manualInput(graph);
        }
    }
}
break;
case 0:
    printf("Введите матрицу (значения - через пробел, 0 - ребра нет):\n");

    for (int i = 0; i < graph->V; i++) {
        for (int j = 0; j < graph->V; j++) {
            scanf_s("%d", &(graph->Matrix[i][j]));
        }
    }
    for (int i = 0; i < graph->V; i++) {
        for (int j = i + 1; j < graph->V; j++) {
            if (graph->Matrix[i][j] == graph->Matrix[j][i] && graph->Matrix[i][j] != 0) {
                printf("Граф является неориентированным. Повторите ввод\n");
                manualInput(graph);
            }
        }
    }
}
break;

default:
    printf("Недопустимое значение. Попробуйте ещё раз.\n");

}
}

```

```

void readGraphFromFile(struct Graph* graph) {

    FILE* file;
    errno_t err;

    if ((err = fopen_s(&file, "matrix.txt", "r")) != 0) {
        perror("\nОшибка при открытии файла. Проверьте существует ли файл.\n Для
        правильного заполнения вводите значения через пробел (0 - ребра нет), количество
        вершин = количеству вершин");
        exit(0);
    }

    // Подсчет количества вершин (по числу строк в файле)
    int vertexCount = 0;
    char c;
    while ((c = fgetc(file)) != EOF) {
        if (c == '\n') {
            vertexCount++;
        }
    }
    vertexCount++;

    // Переходим к началу файла и считаем количество рёбер (ненулевых элементов в
    матрице)
    fseek(file, 0, SEEK_SET);
    int edgeCount = 0;
    int value;

    // Считываем матрицу смежности
    graph->V = vertexCount;
    graph->E = 0;
    graph->Matrix = (int**)malloc(graph->V * sizeof(int*));

    for (int i = 0; i < graph->V; i++) {
        graph->Matrix[i] = (int*)malloc(graph->V * sizeof(int));

        for (int j = 0; j < graph->V; j++) {
            if (fscanf_s(file, "%d", &value) == 1) {
                graph->Matrix[i][j] = value;
                if (value != 0) {
                    graph->E++;
                }
            }
        }
    }
}

```



```

    }
    // Пропускаем пробельные символы
    while ((c = fgetc(file)) != EOF && (c == ' ' || c == '\t')) {}
    // Возвращаем символ, если это не конец файла
    if (c != EOF) {
        ungetc(c, file);
    }
}

// Отладочный вывод матрицы
printf("\nМатрица:\n");
for (int i = 0; i < graph->V; i++) {
    for (int j = 0; j < graph->V; j++) {
        printf("%4d ", graph->Matrix[i][j]);
    }
    printf("\n");
}
fclose(file);
}

void readGraphFromFileTWO(struct Graph* graph) {

    FILE* file;
    errno_t err;

    if ((err = fopen_s(&file, "matrix2.txt", "r")) != 0) {
        perror("Ошибка при открытии файла. Проверьте существует ли файл.\n Для
        правильного заполнения вводите значения через пробел (0 - ребра нет), количество
        вершин = количеству вершин\n");
        exit(0);
    }
    fseek(file, 0, SEEK_SET);
    long filesize = ftell(file);
    if (filesize == EOF) {
        printf("Ошибка. Файл пуст. Необходимо сохранить сгенерированную
        матрицу.\n");
    }

    exit(0);
}

// Подсчет количества вершин (по числу строк в файле)
int vertexCount = 0;
char c;
while ((c = fgetc(file)) != EOF) {

```

```

        if (c == '\n') {
            vertexCount++;
        }
    }

    // Переходим к началу файла и считаем количество рёбер (ненулевых элементов в
матрице)
    fseek(file, 0, SEEK_SET);
    int edgeCount = 0;
    int value;

    // Считываем матрицу смежности
    graph->V = vertexCount;
    graph->E = 0;
    graph->Matrix = (int**)malloc(graph->V * sizeof(int*));

    for (int i = 0; i < graph->V; i++) {
        graph->Matrix[i] = (int*)malloc(graph->V * sizeof(int));
        for (int j = 0; j < graph->V; j++) {
            if (fscanf_s(file, "%d", &value) == 1) {
                graph->Matrix[i][j] = value;
                if (value != 0) {
                    graph->E++;
                }
            }
        }
        // Пропускаем пробельные символы
        while ((c = fgetc(file)) != EOF && (c == ' ' || c == '\t')) {}
        // Возвращаем символ, если это не конец файла
        if (c != EOF) {
            ungetc(c, file);
        }
    }
}

// Отладочный вывод матрицы
printf("\nМатрица:\n");

for (int i = 0; i < graph->V; i++) {
    for (int j = 0; j < graph->V; j++) {
        printf("%4d ", graph->Matrix[i][j]);
    }
    printf("\n");
}

fclose(file);
}

```

```

// Функция для генерации случайного графа
void generateRandomGraph(struct Graph* graph) {
    int choice;
    int value;
    int edgeCount = 0;
    printf("\nВыберете какой граф для генерации\n");
    printf("1 - неориентированный\n");
    printf("0 - ориентированный\n");
    printf("Выбор: ");
    scanf_s("%d", &choice);
    switch (choice) {
    case 1:
        printf("\nГраф неориентированный\n ");
        // Запрос пользователя о количестве вершин
        printf("\nВедите количество вершин: ");
        scanf_s("%d", &graph->V);

        // Запрос пользователя о количестве рёбер
        printf("\nВедите количество рёбер: ");
        scanf_s("%d", &value);
        while (value > graph->V * (graph->V - 1) / 2) {
            printf("Количество ребер превышает максимально возможное для данного
числа вершин в графе.\nПопробуйте ещё раз:");
            scanf_s("%d", &value);
        }
        graph->E = value;

        // Инициализация генератора случайных чисел
        srand(time(NULL));

        // Выделение памяти для матрицы смежности
        graph->Matrix = (int**)malloc(graph->V * sizeof(int*));

        // Инициализация матрицы нулями
        for (int i = 0; i < graph->V; i++) {
            graph->Matrix[i] = (int*)malloc(graph->V * sizeof(int));
            for (int j = 0; j < graph->V; j++) {
                graph->Matrix[i][j] = 0;
            }
        }

        // Заполнение рёбер

```

```

while (edgeCount < graph->E) {
    int i = rand() % graph->V;
    int j = rand() % graph->V;
    if (i != j && graph->Matrix[i][j] == 0) {
        graph->Matrix[i][j] = rand() % 10;
        graph->Matrix[j][i] = graph->Matrix[i][j];
        edgeCount++;
    }
}

// Отладочный вывод матрицы
printf("\nМатрица:\n");
for (int i = 0; i < graph->V; i++) {
    for (int j = 0; j < graph->V; j++) {
        printf("%4d ", graph->Matrix[i][j]);
    }
    printf("\n");
}
break;
case 0:
    printf("\nГраф ориентированный\n ");
    // Запрос пользователя о количестве вершин
    printf("Ведите количество вершин: ");
    scanf_s("%d", &graph->V);
    // Запрос пользователя о количестве рёбер
    printf("\nВедите количество рёбер: ");
    scanf_s("%d", &value);
    while (value > graph->V * (graph->V - 1) / 2) {
        printf("Количество ребер превышает максимально возможное для данного
числа вершин в графе.\nПопробуйте ещё раз:");
        scanf_s("%d", &value);
    }
    graph->E = value;

// Инициализация генератора случайных чисел
srand(time(NULL));

// Выделение памяти для матрицы смежности
graph->Matrix = (int**)malloc(graph->V * sizeof(int*));
// Инициализация матрицы нулями
for (int i = 0; i < graph->V; i++) {
    graph->Matrix[i] = (int*)malloc(graph->V * sizeof(int));
    for (int j = 0; j < graph->V; j++) {
        graph->Matrix[i][j] = 0;
    }
}

```



```

printf("-----");
printf("\nМеню:\n");
printf("1. Взять данные из файла для реализации алгоритма\n");
printf("2. Ввести вручную данные для реализации алгоритма\n");
printf("3. Сгенерировать данные для реализации алгоритма\n");
printf("4. Выйти из программы\n\n");
printf("Выбор: ");
scanf_s("%d", &choice);

switch (choice) {
case 1:
    printf("\nПредупреждение! Если в файл введён неориентированный граф,
то проверьте, чтобы в нём не было отрицательных весов. \n");
    printf("В противном случае всегда будет отрицательный цикл. \n");

    printf("\nКаким файлом воспользоваться?\n ");
    printf("1 - если данные в файл введены вручную (matrix.txt)\n ");
    printf("0 - если данные в файл сохранены при генерации (matrix2.txt)\n
");

    printf("Выбор: ");
    scanf_s("%d", &choice);
    switch (choice) {
case 1:
        readGraphFromFile(&graph);
        break;
case 0:
        readGraphFromFileTWO(&graph);
        break;
default:
        printf("Недопустимое значение. Попробуйте ещё раз.\n");
    }

    printf("Введите исходную вершину (от 1 до %d): ", graph.V);
    scanf_s("%d", &src);
    if (src < 1 || src > graph.V) {
        printf("Недопустимая исходная вершина\n");
        break;
    }

    FordBellman(&graph, src);

    for (int i = 0; i < graph.V; i++)
        free(graph.Matrix[i]);
    free(graph.Matrix);

```

```

        break;

case 2:
    printf("Введите количество вершин: ");
    scanf_s("%d", &graph.V);

    graph.E = graph.V * graph.V;
    graph.Matrix = (int**)malloc(graph.V * sizeof(int*));
    for (int i = 0; i < graph.V; i++)
        graph.Matrix[i] = (int*)malloc(graph.V * sizeof(int));

    manualInput(&graph);

    printf("Введите исходную вершину (от 1 до %d): ", graph.V);
    scanf_s("%d", &src);
    if (src < 1 || src > graph.V) {
        printf("Недопустимая исходная вершина\n");
        break;
    }

    FordBellman(&graph, src);

    for (int i = 0; i < graph.V; i++)
        free(graph.Matrix[i]);
    free(graph.Matrix);
    break;

case 3:
    generateRandomGraph(&graph);

```

43

```

    printf("Сохранить ли матрицу в файл? (1 - да/ 0 - продолжить без
сохранения): ");
    scanf_s("%d", &choice);
    switch (choice) {
    case 1:
        saveMatrixToFile(&graph);
        printf("Матрица сохранена в файл 'matrix2.txt'.\n");
        break;
    case 0:
        break;
    default:
        printf("Недопустимое значение. Попробуйте ещё раз.\n");
    }

```

```

printf("\nВведите исходную вершину (от 1 до %d): ", graph.V);
scanf_s("%d", &src);
if (src < 1 || src > graph.V) {
    printf("Недопустимая исходная вершина\n");
    break;
}

FordBellman(&graph, src);

for (int i = 0; i < graph.V; i++)
    free(graph.Matrix[i]);
free(graph.Matrix);
break;

case 4:
    printf("\nВыход из программы.\n");
    exit(0);

default:
    printf("Недопустимое значение. Попробуйте ещё раз.\n");
}

}

return 0;
}

```