

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

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе
по курсу «Логика и основы алгоритмизации
в инженерных задачах»
на тему «Реализация алгоритма Форда-Беллмана»

24.12.24
хорошо
оценка

Выполнил:

студент группы 23BBB2
Скалдин В. С.

Принял:

к.т.н. доцент
Юрова О.В.
д.т.н. проф.
Митрохин М. А.

Пенза 2024

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 Скалдину Вадиму Сергеевичу Группа 23ВВВх
Тема проекта Реализация алгоритма Форда - Беллмана

Исходные данные (технические требования) на проектирование

Разработка алгоритмов и программного обеспечения в соответствии с данным заданием курсового проекта. Исходные данные должны содержать:

1. Постановку задачи;
2. Теоретическую часть задания;
3. Описание алгоритма поставленной задачи;
4. Пример ручного расчета задачи и вычислений (на небольшом участке работы алгоритма);
5. Описание самой программы;
6. Тесты;
7. Список литературы;
8. Листинг программы;
9. Результаты выполнения работы программы

Объем работы по курсу

1. Расчетная часть

Вручной расчет работы алгоритма

2. Графическая часть

Схема алгоритма в формате блок-схем

3. Экспериментальная часть

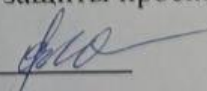
Тестирование программы;
Результаты работы программы на тестовых данных

Срок выполнения проекта по разделам


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

Дата выдачи задания "17" сентября 2024

Дата защиты проекта " " "

Руководитель Юрова О.В. 

Задание получил "17" сентября 2024 г.

Студент Скалдин В.С. 

Содержание

| | |
|---------------------------------------|-------------------------------------|
| Реферат | Error! Bookmark not defined. |
| Введение | Error! Bookmark not defined. |
| 1. Постановка задачи | Error! Bookmark not defined. |
| 2. Теоретическая часть..... | 8 |
| 3. Описание алгоритма программы..... | 9 |
| 4. Описание программы..... | 12 |
| 5. Тестирование..... | 20 |
| Заключение..... | 26 |
| Список литературы | 27 |
| Приложение А. Листинг программы. | 28 |

Реферат

Отчет 31стр, 13 рисунков, 1 приложение

ГРАФ,ТЕОРИЯ ГРАФОВ,КРАТЧАЙШИЙ ПУТЬ, АЛГОРИТМ ФОРДА-БЕЛЛМАНА

Цель исследования – разработка программы на языке C++, которая выполняет поиск кратчайшего пути из одной вершины графа до всех остальных с помощью алгоритма Форда-Беллмана.

В работе рассмотрены алгоритм Форда-Беллмана, опираясь на использования динамического программирования, функции ввода и генерации рёбер.

Введение

Алгоритм Форда-Беллмана — это алгоритм поиска кратчайшего пути во взвешенном графе. Он находит кратчайшие пути от одной вершины графа до всех остальных, даже для графов, в которых веса рёбер могут быть отрицательными. Принцип действия алгоритма основан на принципе релаксации. Он начинается с предположения, что кратчайшее расстояние до всех вершин от исходной вершины равно бесконечности. Затем, посредством серии итераций, он обновляет все эти расстояния, ослабляя рёбра (означает поиск более коротких путей, когда это возможно).

Алгоритм Форда-Беллмана является важным инструментом для решения задач оптимизации, таких как нахождение кратчайшего маршрута, методах отмены цикла, а также в протоколах маршрутизации, связанных с векторами расстояний, например, в Протоколе информации о маршрутизации (RIP). В качестве среды разработки мною была выбрана среда Microsoft Visual Studio 2022, язык программирования — C++.

Целью данной курсовой работы является разработка программы на языке C++, который является широко используемым. Именно с его помощью в данном курсовом проекте реализуется алгоритм Форда-Беллмана.

1. Постановка задачи

Требуется разработать программу, которая найдет кратчайший путь из одной вершины графа до всех остальных с помощью алгоритма Форда-Беллмана.

Исходный граф в программе должен задаваться матрицей смежности, причем должна быть возможность задать ее различными способами, как случайно, так и с помощью ручного ввода. Пользователь должен вводить в программу количество вершин. После обработки этих данных на экран выводится матрица смежности и результат алгоритма Форда-Беллмана. Необходимо предусмотреть различные исходы поиска, чтобы программа не выдавала ошибок и работала правильно. Устройство ввода – клавиатура и мышь. Необходимо меню для удобной работы пользователя с алгоритмом.

2. Теоретическая часть задания

Алгоритм Форда-Беллмана — это алгоритм для нахождения кратчайших путей от одной вершины графа до всех остальных вершин. Он может работать с графами, содержащими ребра с отрицательными весами, что отличает его от алгоритма Дейкстры, который не может корректно обрабатывать такие случаи. Алгоритм работает с ненаправленными графами, где ребра могут иметь как положительные, так и отрицательные веса.

Начальная вершина (источник) инициализируется значением 0 (расстояние до самой себя). Все остальные вершины инициализируются бесконечностью (или очень большим числом), что означает, что они недоступны на начальном этапе.

Алгоритм проходит по всем ребрам графа и обновляет расстояния до вершин. Если текущее расстояние до вершины A плюс вес ребра (A, B) меньше текущего известного расстояния до вершины B , то расстояние до B обновляется. Этот процесс повторяется $V-1$ раз, где V — количество вершин в графе. Это делается потому, что в самом худшем случае кратчайший путь может содержать $V-1$ ребер. Этот процесс называется релаксацией.

После $V-1$ итераций алгоритм выполняет еще одну итерацию релаксации. Если на этой итерации удастся улучшить расстояние до какой-либо вершины, это означает, что в графе присутствует отрицательный цикл.

Временная сложность алгоритма составляет $O(V * E)$, где V — количество вершин, а E — количество ребер в графе. Это делает его менее эффективным по сравнению с алгоритмом Дейкстры для графов с положительными весами, но более универсальным.

Таким образом, алгоритм Форда-Беллмана позволяет эффективно находить кратчайшие пути даже в сложных условиях.

3. Описание алгоритма программы

Алгоритм программы `fordBellman` реализует поиск кратчайших путей от одной вершины графа до всех остальных вершин.

Параметры функции:

- `int** G`: Матрица смежности графа, где `G[u][v]` представляет вес ребра от вершины `u` к вершине `v`. Если `G[u][v] == 0`, это означает, что ребро отсутствует.
- `int size`: Количество вершин в графе.
- `int start`: Индекс начальной вершины, от которой мы будем искать кратчайшие пути.

Локальные переменные:

- `int* distance`: Массив, который хранит расстояния от начальной вершины до каждой другой вершины. Изначально все значения устанавливаются в "бесконечность" (`INT_MAX`), кроме начальной вершины, для которой расстояние равно 0.
- `bool* inNegativeCycle`: Массив, который помечает вершины, связанные с отрицательными циклами. В данной реализации он не используется, но обычно его применяют для отслеживания вершин, которые могут быть достигнуты через отрицательные циклы.

Порядок работы алгоритма:

Сперва мы инициализируем массив расстояний до всех вершин как бесконечность, а для начальной вершины устанавливаем расстояние равным 0. Далее внешний цикл (`for (int k = 0; k < size - 1; k++)`) выполняется `size - 1` раз. Это количество итераций необходимо для того, чтобы гарантировать нахождение кратчайших путей, так как в графе с `n` вершинами максимальная длина пути без циклов не может превышать `n - 1`. Затем первый вложенный цикл (`for (int u = 0; u < size; u++)`) проходит по всем вершинам `u`. После второй вложенный цикл (`for (int v = 0; v < size; v++)`) проходит по всем вершинам `v`, проверяя наличие ребра от `u` к `v`.

Затем проверяются условия внутри второго вложенного цикла:

- $G[u][v] \neq 0$: Проверяет, существует ли ребро от u к v .
- $distance[u] \neq INT_MAX$: Проверяет, достигнута ли вершина u .
- $distance[u] + G[u][v] < distance[v]$: Проверяет, является ли найденный путь до вершины v через вершину u более коротким, чем ранее известный путь. Если все условия выполняются, обновляется расстояние до вершины v .

После выполнения основного цикла массив `distance` будет содержать кратчайшие расстояния от начальной вершины до всех других вершин.

Ниже представлен псевдокод функции `fordBellman()`.

fordBellman(G, size, start):

1. Объявить массив расстояний `distance[size]`
2. Объявить массив `inNegativeCycle[size]`
3. Для i от 0 до $size - 1$:
 4. $distance[i] \leftarrow \infty$
 5. `inNegativeCycle[i] ← Ложь`
6. $distance[start] \leftarrow 0$
7. Для k от 0 до $size - 2$:
 8. Для u от 0 до $size - 1$:
 9. Для v от 0 до $size - 1$:
 10. Если $G[u][v] \neq 0$ И $distance[u] \neq \infty$ И $distance[u] + G[u][v] < distance[v]$:
 11. $distance[v] \leftarrow distance[u] + G[u][v]$
 12. Для u от 0 до $size - 1$:
 13. Для v от 0 до $size - 1$:
 14. Если $G[u][v] \neq 0$ И $distance[u] \neq \infty$ И $distance[u] + G[u][v] < distance[v]$:
 15. `inNegativeCycle[v] ← Истина`
 16. Для k от 0 до $size - 1$: // Повторяем $size$ раз
 17. Для u от 0 до $size - 1$:
 18. Для v от 0 до $size - 1$:
 19. Если $G[u][v] \neq 0$ И `inNegativeCycle[u] = Истина`:
 20. `inNegativeCycle[v] ← Истина`

21. Вывести "Кратчайшие расстояния от вершины " + start + ":"
22. Для i от 0 до size - 1:
 23. Если inNegativeCycle[i] = Истина:
 24. Вывести "До вершины " + i + ": недоступно (отрицательный цикл)"
 25. Иначе если distance[i] = ∞ :
 26. Вывести "До вершины " + i + ": " + 0
 27. Иначе:
 28. Вывести "До вершины " + i + ": " + distance[i]

Полный код программы можно увидеть в Приложении А.

4. Описание программы

Эта программа представляет собой консольное приложение на языке C, которое позволяет работать с графами, включая их создание, ввод рёбер, отображение и поиск кратчайших путей с использованием алгоритма Форда-Беллмана. Проект был создан в виде консольного приложения Win32 (Visual C++). Основные принципы работы программы:

1) Создание графа:

- Граф представлен в виде матрицы смежности, где каждая ячейка хранит вес ребра между двумя вершинами.
- Функция `createG` создает пустую матрицу заданного размера и инициализирует её нулями.

2) Программа поддерживает два способа ввода рёбер:

- Ручной ввод (`manualInp`): Пользователь задаёт количество рёбер, а затем вводит начальную и конечную вершины каждого ребра, а также его вес.
- Случайная генерация (`randomInp`): Рёбра генерируются случайным образом с заданным пользователем максимальным весом.

3) Отображение графа:

- Функция `printG` выводит матрицу смежности графа в удобном формате.

4) Алгоритм Форда-Беллмана:

- Реализован для поиска кратчайших путей от заданной стартовой вершины до всех остальных.
- Алгоритм учитывает возможность наличия отрицательных весов рёбер.
- Проверяется наличие отрицательных циклов, и если вершина связана с таким циклом, программа помечает её как недоступную.
- Результаты выводятся в виде расстояний от стартовой вершины до каждой другой вершины.

5) Программа предоставляет пользователю интерактивное меню с такими опциями:

- Установить размер графа.
- Ввести рёбра вручную.
- Сгенерировать рёбра случайным образом.
- Отобразить граф.
- Найти кратчайшие расстояния с использованием алгоритма Форда-Беллмана.
- Выйти из программы.

6) Управление памятью:

- Для хранения графа используется динамическая память. Функция `deleteG` освобождает память, выделенную под матрицу смежности, чтобы избежать утечек памяти.\

7) Обработка ошибок:

- Программа проверяет корректность ввода данных (например, размер графа, количество рёбер, номера вершин) и выводит соответствующие сообщения об ошибках.

8) Главная функция:

- Функция `menu` организует работу программы, предоставляя пользователю доступ к функционалу через меню.
- Основная функция `main` запускает меню и завершает выполнение программы.

Работа программы начинается с вывода меню и ожидания дальнейшего ввода (Рисунок 1).

```
Меню:  
1. Установить размер графа  
2. Ввести рёбра вручную  
3. Сгенерировать рёбра случайно  
4. Отобразить граф  
5. Найти кратчайшие расстояния (алгоритм Форда-Беллмана)  
6. Выход  
Введите ваш выбор:
```

Рисунок 1 – Меню программы

Пользователь должен выбрать способ генерации матрицы смежности. Если выбрана случайная генерация, пользователь указывает количество вершин и максимальный вес ребра (Рисунок 2). Для вывода матрицы на экран в меню следует выбрать соответствующий пункт (Рисунок 3).

```
Меню:  
1. Установить размер графа  
2. Ввести рёбра вручную  
3. Сгенерировать рёбра случайно  
4. Отобразить граф  
5. Найти кратчайшие расстояния (алгоритм Форда-Беллмана)  
6. Выход  
Введите ваш выбор: 1  
Введите размер графа: 6  
Размер графа - 6.  
  
Меню:  
1. Установить размер графа  
2. Ввести рёбра вручную  
3. Сгенерировать рёбра случайно  
4. Отобразить граф  
5. Найти кратчайшие расстояния (алгоритм Форда-Беллмана)  
6. Выход  
Введите ваш выбор: 3  
Введите максимальный вес для рёбер: 6  
Граф успешно сгенерирован.
```

Рисунок 2 – Случайная генерация матрицы

```

Меню:
1. Установить размер графа
2. Ввести рёбра вручную
3. Сгенерировать рёбра случайно
4. Отобразить граф
5. Найти кратчайшие расстояния (алгоритм Форда-Беллмана)
6. Выход
Введите ваш выбор: 4
  0    0    0   -2    0    0
  0    0    0    0   -1    0
  0    0    0    0    0    0
-2    0    0    0   -3   -1
  0   -1    0   -3    0    2
  0    0    0   -1    2    0

```

Рисунок 3 – Сгенерированная матрица

После создания матрицы смежности она сохраняется в памяти. Далее вызывается алгоритм Форда-Беллмана, в котором пользователь вводит стартовую вершину. После того как алгоритм будет выполнен, пользователь сможет посмотреть результат в консоли (рисунок 4).

```

Меню:
1. Установить размер графа
2. Ввести рёбра вручную
3. Сгенерировать рёбра случайно
4. Отобразить граф
5. Найти кратчайшие расстояния (алгоритм Форда-Беллмана)
6. Выход
Введите ваш выбор: 5
Введите стартовую вершину: 3
Кратчайшие расстояния от вершины 3:
До вершины 0: недоступно (отрицательный цикл)
До вершины 1: недоступно (отрицательный цикл)
До вершины 2: 0
До вершины 3: недоступно (отрицательный цикл)
До вершины 4: недоступно (отрицательный цикл)
До вершины 5: недоступно (отрицательный цикл)

```

Рисунок 4 – Результат работы алгоритма

Подключение библиотек:

```
#include <iostream>
#include <locale.h>
#include <iomanip>
#include <climits>
```

```
using namespace std;
```

Создание матрицы:

```
int** createG(int size) {
    int** G = new int* [size];
    for (int i = 0; i < size; i++) {
        G[i] = new int[size];
        for (int j = 0; j < size; j++) {
            G[i][j] = 0;
        }
    }
    return G;
}
```

Вывод матрицы:

```
void printG(int** G, int size) {
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++)
            cout << setw(5) << G[i][j];
        cout << endl;
    }
}
```

Случайная генерация рёбер:

```
void randomInp(int** G, int size) {

    int maxWeight;
    cout << "Введите максимальный вес для рёбер: ";
    cin >> maxWeight;

    srand(time(NULL));

    for (int i = 0; i < size; i++) {
        for (int j = i; j < size; j++) {
            int weight = rand() % (maxWeight + 1) - maxWeight / 2;
            G[i][j] = (rand() % 2 == 0) ? weight : 0; // 50% вероятность ребра
            if (i == j)
                G[i][j] = 0;
            G[j][i] = G[i][j];
        }
    }
}
```

Ручная генерация рёбер:

```
void manualInp(int** G, int size) {
    int u, v, weight;
    int edges;
    cout << "Введите количество рёбер: ";
    while (!(cin >> edges) || edges < 0 || edges > size * (size - 1) / 2) {
        cout << "Такого количества ребер быть не может, попробуйте снова: ";
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
    }
}
```



```

for (int i = 0; i < edges; i++) {
    cout << "Введите начало, конец и вес ребра (u v weight): ";
    cin >> u >> v >> weight;
    if (u >= size && v >= size) {
        cout << "Некорректные узлы, попробуйте снова." << endl;
        i--;
        continue;
    }
    G[u][v] = weight;
    G[v][u] = weight;
}
}

```

Нахождение кратчайших путей от заданной стартовой вершины до всех остальных:

```

void fordBellman(int** G, int size, int start) {
    int* distance = new int[size];
    bool* inNegativeCycle = new bool[size];

    for (int i = 0; i < size; i++) {
        distance[i] = INT_MAX;
        inNegativeCycle[i] = false;
    }
    distance[start] = 0;
}

```

Релаксация рёбер:

```

for (int k = 0; k < size - 1; k++) {
    for (int u = 0; u < size; u++) {
        for (int v = 0; v < size; v++) {
            if (G[u][v] != 0 && distance[u] != INT_MAX && distance[u] + G[u][v] < distance[v])
                distance[v] = distance[u] + G[u][v];
        }
    }
}
}

```

Проверка на наличие отрицательных циклов:

```

for (int u = 0; u < size; u++) {
    for (int v = 0; v < size; v++) {
        if (G[u][v] != 0 && distance[u] != INT_MAX && distance[u] + G[u][v] < distance[v]) {
            inNegativeCycle[v] = true;
        }
    }
}
for (int k = 0; k < size; k++) {
    for (int u = 0; u < size; u++) {
        for (int v = 0; v < size; v++) {
            if (G[u][v] != 0 && inNegativeCycle[u]) {
                inNegativeCycle[v] = true;
            }
        }
    }
}
}

```

Вывод кратчайших расстояний:

```

cout << "Кратчайшие расстояния от вершины " << start << ": " << endl;
for (int i = 0; i < size; i++) {
    if (inNegativeCycle[i]) {
        cout << "До вершины " << i << ": недоступно (отрицательный цикл)" << endl;
    }
    else if (distance[i] == INT_MAX) {
        cout << "До вершины " << i << ": " << 0 << endl;
    }
}

```

```

    }
    else {
        cout << "До вершины " << i << ": " << distance[i] << endl;
    }
}

```

Освобождение памяти:

```

delete[] distance;
delete[] inNegativeCycle;
}

void deleteG(int** G, int size) {
    for (int i = 0; i < size; i++) {
        delete[] G[i];
    }
    delete[] G;
}

```

В отдельной функции menu() реализовано текстовое меню для работы с графом в виде динамической матрицы смежности. Она позволяет пользователю выполнять различные операции с графом, такие как установка размера, ввод рёбер, их генерация, отображение графа и поиск кратчайших расстояний с помощью алгоритма Форда-Беллмана до тех пор, пока пользователь не выберет пункт 6.

Элементы меню:

1. Установить размер графа:

- Проверяет, существует ли граф (если да, то освобождает память).
- Запрашивает у пользователя размер графа и создает новую матрицу смежности.

смежности.

2. Ввести рёбра вручную:

- Проверяет, установлен ли размер графа. Если нет, выводит сообщение об ошибке.
- Вызывает функцию manualInp(G, size), которая должна обрабатывать ввод рёбер пользователем.

3. Сгенерировать рёбра случайно:

- Также проверяет наличие установленного размера графа.
- Вызывает функцию randomInp(G, size) для генерации рёбер случайным образом.

4. Отобразить граф:

- Проверяет наличие графа и вызывает функцию `printG(G, size)` для отображения матрицы смежности.

5. Найти кратчайшие расстояния (алгоритм Форда-Беллмана):

- Проверяет наличие графа и запрашивает стартовую вершину.
- Вызывает функцию `fordBellman(G, size, startVertex)` для выполнения алгоритма.

6. Выход:

- Завершает выполнение программы.

Конец программы.

5. 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.

Таблица 1 – Описание поведения программы при тестировании

| № | Описание | Предусловие | Тестирование | Ожидаемый результат |
|---|--------------------------|---|---|--------------------------------------|
| 1 | Работа меню | Программа запущена | Запускаем программу с помощью Visual Studio | Вывод в консоли меню программ |
| 2 | Выбор функции | Программа запущена | Вводим номер функции | Переход к выполнению функции |
| 3 | Ручное создание рёбер | Ввод 1 и 2 в меню | Ввод начала, конца и вес ребра | Создание рёбер и запись в памяти |
| 4 | Случайное создание рёбер | Ввод 1 и 3 в меню | Ввод максимального веса ребёр | Создание рёбер и запись в памяти |
| 5 | Вывод матрицы на экран | Наличие матрицы в памяти | Ввод 4 в меню | Вывод матрицы на экран |
| 6 | Работа алгоритма | Создание любым способ матрицы смежности | Ввод 5 в меню | Вывод в консоли результата алгоритма |

Тест №1

Запускаем программу из Microsoft Visual Studio 2022 (Рисунок 5).

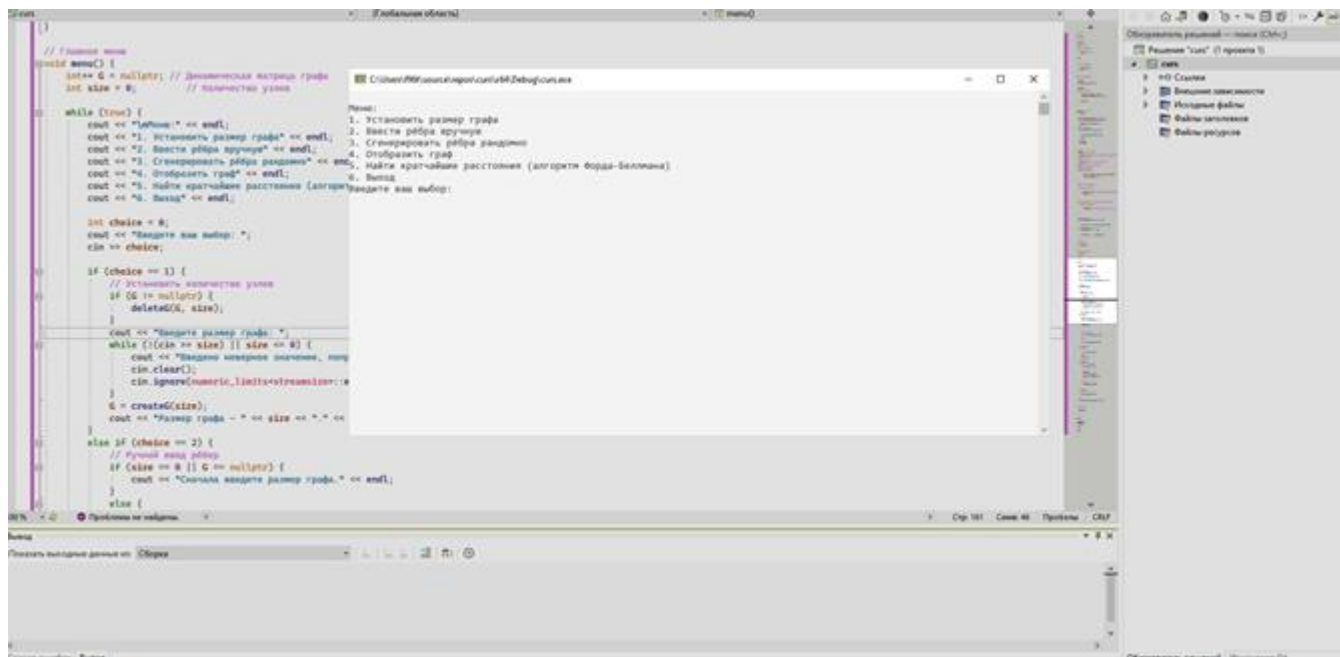


Рисунок 5 –Запуск программы

Программа успешно запустилась. Ожидаемые результаты совпали с реальными.

Тест №2

Ввели в консоли 1(Рисунок 6).

```
Меню:
1. Установить размер графа
2. Ввести рёбра вручную
3. Сгенерировать рёбра случайно
4. Отобразить граф
5. Найти кратчайшие расстояния (алгоритм Форда-Беллмана)
6. Выход
Введите ваш выбор: 1
Введите размер графа: 3
Размер графа - 3.

Меню:
1. Установить размер графа
2. Ввести рёбра вручную
3. Сгенерировать рёбра случайно
4. Отобразить граф
5. Найти кратчайшие расстояния (алгоритм Форда-Беллмана)
6. Выход
Введите ваш выбор:
```

Рисунок 6 – Проверка ввода

Программа успешно выполнилась и ожидает дальнейшего ввода.

2.3 Ввели в консоль неправильный символ (Рисунок 7).

```
Меню:
1. Установить размер графа
2. Ввести рёбра вручную
3. Сгенерировать рёбра рандомно
4. Отобразить граф
5. Найти кратчайшие расстояния (алгоритм Форда-Беллмана)
6. Выход
Введите ваш выбор: 7
Некорректный выбор. Попробуйте снова.

Меню:
1. Установить размер графа
2. Ввести рёбра вручную
3. Сгенерировать рёбра рандомно
4. Отобразить граф
5. Найти кратчайшие расстояния (алгоритм Форда-Беллмана)
6. Выход
Введите ваш выбор:
```

Рисунок 7 – Проверка ввода

У программы предусмотрена защита от неправильного ввода.

Тест №3

Выбрали в меню пункт 2 и ввели начало, конец и вес рёбер (Рисунок 8).

```
Меню:
1. Установить размер графа
2. Ввести рёбра вручную
3. Сгенерировать рёбра рандомно
4. Отобразить граф
5. Найти кратчайшие расстояния (алгоритм Форда-Беллмана)
6. Выход
Введите ваш выбор: 2
Введите количество рёбер: 3
Введите начало, конец и вес ребра (u v weight): 1
3
-1
Введите начало, конец и вес ребра (u v weight): 1
2
3
Введите начало, конец и вес ребра (u v weight): 3
1
2

Меню:
1. Установить размер графа
2. Ввести рёбра вручную
3. Сгенерировать рёбра рандомно
4. Отобразить граф
5. Найти кратчайшие расстояния (алгоритм Форда-Беллмана)
6. Выход
Введите ваш выбор:
```

Рисунок 8 – Проверка ручного ввода рёбер

Выбрали в меню пункт 4 и вывели матрицу на экран (Рисунок 9).

```
меню:
1. Установить размер графа
2. Ввести рёбра вручную
3. Сгенерировать рёбра рандомно
4. Отобразить граф
5. Найти кратчайшие расстояния (алгоритм Форда-Беллмана)
6. Выход
Введите ваш выбор: 4
  0    0   -1    0   -1
  0    0    3    2    0
 -1    3    0    0    0
  0    2    0    0    2
 -1    0    0    2    0

Меню:
1. Установить размер графа
2. Ввести рёбра вручную
3. Сгенерировать рёбра рандомно
4. Отобразить граф
5. Найти кратчайшие расстояния (алгоритм Форда-Беллмана)
6. Выход
Введите ваш выбор:
```

Рисунок 9 – Проверка вывода матрицы смежности

Программа корректно вывела матрицу смежности для вручную введенных рёбер и ожидает следующего ввода.

Тест №4

Выбрали в меню пункт 3 и ввели максимальный вес ребра (Рисунок 10).

```
меню:
1. Установить размер графа
2. Ввести рёбра вручную
3. Сгенерировать рёбра рандомно
4. Отобразить граф
5. Найти кратчайшие расстояния (алгоритм Форда-Беллмана)
6. Выход
Введите ваш выбор: 3
Введите максимальный вес для рёбер: 4
Граф успешно сгенерирован.
```

Рисунок 10 – Проверка случайной генерации рёбер

Выбрали в меню пункт 4 и ввели матрицу на экран (Рисунок 11).

```

Меню:
1. Установить размер графа
2. Ввести рёбра вручную
3. Сгенерировать рёбра рандомно
4. Отобразить граф
5. Найти кратчайшие расстояния (алгоритм Форда-Беллмана)
6. Выход
Введите ваш выбор: 4
  0   0   0   1   0
  0   0   0  -2   0
  0   0   0   0   0
  1  -2   0   0  -1
  0   0   0  -1   0

Меню:
1. Установить размер графа
2. Ввести рёбра вручную
3. Сгенерировать рёбра рандомно
4. Отобразить граф
5. Найти кратчайшие расстояния (алгоритм Форда-Беллмана)
6. Выход
Введите ваш выбор:

```

Рисунок 11 – Проверка вывода матрицы смежности

Программа корректно вывела матрицу смежности для случайно сгенерированных рёбер и ожидает следующего ввода.

Тест №5

Вывод матрицы на экран. Ввод числа 4.

```

Меню:
1. Установить размер графа
2. Ввести рёбра вручную
3. Сгенерировать рёбра рандомно
4. Отобразить граф
5. Найти кратчайшие расстояния (алгоритм Форда-Беллмана)
6. Выход
Введите ваш выбор: 4
  0   0   0   1   0
  0   0   0  -2   0
  0   0   0   0   0
  1  -2   0   0  -1
  0   0   0  -1   0

Меню:
1. Установить размер графа
2. Ввести рёбра вручную
3. Сгенерировать рёбра рандомно
4. Отобразить граф
5. Найти кратчайшие расстояния (алгоритм Форда-Беллмана)
6. Выход
Введите ваш выбор:

```

Рисунок 12 – Проверка вывода матрицы смежности

Программа корректно вывела матрицу смежности и ожидает следующего ввода (Рисунок 12).

Тест №6

Создадим матрицу любым способом и проверим результаты работы алгоритма (Рисунок 13).

```

0  0  0  -2  0  0
0  0  0  0  -1  0
0  0  0  0  0  0
-2  0  0  0  -3  -1
0  -1  0  -3  0  2
0  0  0  -1  2  0

меню:
1. Установить размер графа
2. Ввести рёбра вручную
3. Сгенерировать рёбра случайно
4. Отобразить граф
5. Найти кратчайшие расстояния (алгоритм Форда-Беллмана)
6. Выход
Введите ваш выбор: 5
Введите стартовую вершину: 3
Кратчайшие расстояния от вершины 3:
до вершины 0: недоступно (отрицательный цикл)
до вершины 1: недоступно (отрицательный цикл)
до вершины 2: 0
до вершины 3: недоступно (отрицательный цикл)
до вершины 4: недоступно (отрицательный цикл)
до вершины 5: недоступно (отрицательный цикл)
```

Рисунок 13 – Проверка работы алгоритма

Таблица 2 – Результаты поведения программы при тестировании

| № | Описание | Полученный результат |
|---|--------------------------|----------------------|
| 1 | Работа меню | верно |
| 2 | Выбор функции | верно |
| 3 | Ручное создание рёбер | верно |
| 4 | Случайное создание рёбер | верно |
| 5 | Вывод матрицы в консоли | верно |
| 6 | Работа алгоритма | верно |

Заключение

В рамках создания данного проекта была разработана программа, реализующая алгоритм Форда-Беллмана с использованием среды Microsoft Visual Studio 2022.

В процессе выполнения курсовой работы приобретены навыки программирования, изучены методы работы с матрицами смежности графов и освоен алгоритм поиска кратчайшего пути. Кроме того, были углублены знания языка программирования C.

Основным недостатком программы является малая эффективность по сравнению с алгоритмом Дейкстры для графов с положительными весами.

Список литературы

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.

Приложение А

Листинг программы

```
#include <iostream>
#include <locale.h>
#include <iomanip>
#include <climits>

using namespace std;

int** createG(int size) {
    int** G = new int* [size];
    for (int i = 0; i < size; i++) {
        G[i] = new int[size];
        for (int j = 0; j < size; j++) {
            G[i][j] = 0;
        }
    }
    return G;
}

void printG(int** G, int size) {
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++)
            cout << setw(5) << G[i][j];
        cout << endl;
    }
}

// Функция для рандомного ввода рёбер
void randomInp(int** G, int size) {

    int maxWeight;
    cout << "Введите максимальный вес для рёбер: ";
    cin >> maxWeight;

    srand(time(NULL));

    for (int i = 0; i < size; i++) {
        for (int j = i; j < size; j++) {
            int weight = rand() % (maxWeight + 1) - maxWeight / 2;
            G[i][j] = (rand() % 2 == 0) ? weight : 0; // 50% вероятность ребра
            if (i == j)
                G[i][j] = 0;
            G[j][i] = G[i][j];
        }
    }
}

// Функция для ручного ввода рёбер
void manualInp(int** G, int size) {
    int u, v, weight;
    int edges;
    cout << "Введите количество рёбер: ";
    while (!(cin >> edges) || edges < 0 || edges > size * (size - 1) / 2) {
        cout << "Такого количества ребер быть не может, попробуйте снова: ";
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
    }

    for (int i = 0; i < edges; i++) {
        cout << "Введите начало, конец и вес ребра (u v weight): ";
        cin >> u >> v >> weight;
```

```

    if (u >= size && v >= size) {
        cout << "Некорректные узлы, попробуйте снова." << endl;
        i--;
        continue;
    }
    G[u][v] = weight;
    G[v][u] = weight;
}
}

// Алгоритм Форда-Беллмана
void fordBellman(int** G, int size, int start) {
    int* distance = new int[size]; // Массив для хранения расстояний
    bool* inNegativeCycle = new bool[size]; // Массив для пометки вершин, связанных с отрицательными циклами

    for (int i = 0; i < size; i++) {
        distance[i] = INT_MAX; // Инициализация бесконечностью
        inNegativeCycle[i] = false; // Изначально ни одна вершина не помечена
    }
    distance[start] = 0; // Расстояние до начальной вершины = 0

    // Основной цикл (расслабляем рёбра)
    for (int k = 0; k < size - 1; k++) {
        for (int u = 0; u < size; u++) {
            for (int v = 0; v < size; v++) {
                if (G[u][v] != 0 && distance[u] != INT_MAX && distance[u] + G[u][v] < distance[v])
                    distance[v] = distance[u] + G[u][v];
            }
        }
    }

    // Проверка на наличие отрицательных циклов
    for (int u = 0; u < size; u++) {
        for (int v = 0; v < size; v++) {
            if (G[u][v] != 0 && distance[u] != INT_MAX && distance[u] + G[u][v] < distance[v]) {
                // Если расстояние можно улучшить, помечаем вершину как связанную с отрицательным циклом
                inNegativeCycle[v] = true;
            }
        }
    }

    // Распространение метки отрицательного цикла на все связанные вершины
    for (int k = 0; k < size; k++) { // Повторяем size раз, чтобы учесть все возможные связи
        for (int u = 0; u < size; u++) {
            for (int v = 0; v < size; v++) {
                if (G[u][v] != 0 && inNegativeCycle[u]) {
                    inNegativeCycle[v] = true; // Если вершина u в отрицательном цикле, то и v тоже
                }
            }
        }
    }

    // Вывод кратчайших расстояний
    cout << "Кратчайшие расстояния от вершины " << start << ":" << endl;
    for (int i = 0; i < size; i++) {
        if (inNegativeCycle[i]) {
            cout << "До вершины " << i << ": недоступно (отрицательный цикл)" << endl;
        }
        else if (distance[i] == INT_MAX) {
            cout << "До вершины " << i << ": " << 0 << endl;
        }
        else {
            cout << "До вершины " << i << ": " << distance[i] << endl;
        }
    }
}

```

```

// Освобождение памяти
delete[] distance;
delete[] inNegativeCycle;
}

void deleteG(int** G, int size) {
    for (int i = 0; i < size; i++) {
        delete[] G[i];
    }
    delete[] G;
}

// Главное меню
void menu() {
    int** G = nullptr; // Динамическая матрица графа
    int size = 0;       // Количество узлов

    while (true) {
        cout << "\nМеню:" << endl;
        cout << "1. Установить размер графа" << endl;
        cout << "2. Ввести рёбра вручную" << endl;
        cout << "3. Сгенерировать рёбра случайно" << endl;
        cout << "4. Отобразить граф" << endl;
        cout << "5. Найти кратчайшие расстояния (алгоритм Форда-Беллмана)" << endl;
        cout << "6. Выход" << endl;

        int choice = 0;
        cout << "Введите ваш выбор: ";
        cin >> choice;

        if (choice == 1) {
            // Установить количество узлов
            if (G != nullptr) {
                deleteG(G, size);
            }
            cout << "Введите размер графа: ";
            while (!(cin >> size) || size <= 0) {
                cout << "Введено неверное значение, попробуйте снова: ";
                cin.clear();
                cin.ignore(numeric_limits<streamsize>::max(), '\n');
            }
            G = createG(size);
            cout << "Размер графа - " << size << "." << endl;
        }
        else if (choice == 2) {
            // Ручной ввод рёбер
            if (size == 0 || G == nullptr) {
                cout << "Сначала введите размер графа." << endl;
            }
            else {
                manualInp(G, size);
            }
        }
        else if (choice == 3) {
            // Случайная генерация рёбер
            if (size == 0 || G == nullptr) {
                cout << "Сначала введите размер графа." << endl;
            }
            else {
                randomInp(G, size);
                cout << "Граф успешно сгенерирован." << endl;
            }
        }
        else if (choice == 4) {
            // Отобразить граф

```

```

    if (size == 0 || G == nullptr) {
        cout << "Граф не задан." << endl;
    }
    else {
        printG(G, size);
    }
}
else if (choice == 5) {
    // Запустить алгоритм Форда-Беллмана
    if (size == 0 || G == nullptr) {
        cout << "Граф не задан." << endl;
    }
    else {
        int startVertex;
        cout << "Введите стартовую вершину: ";
        cin >> startVertex;
        if (startVertex < 0 || startVertex >= size) {
            cout << "Некорректное значение." << endl;
        }
        else {
            fordBellman(G, size, startVertex);
        }
    }
}
else if (choice == 6) {
    // Выход
    cout << "Выход из программы." << endl;
    break;
}
else {
    cout << "Некорректный выбор. Попробуйте снова." << endl;
}
}

// Освобождаем память
if (G != nullptr) {
    deleteG(G, size);
}
}

int main(void) {

    setlocale(LC_ALL, "");
    srand(time(NULL));

    menu();

    return 0;
}

```