

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

Пояснительная записка

К курсовому проектированию
по курсу «Логика и основы алгоритмизации
в инженерных задачах»
на тему «Реализация алгоритма Дейкстры»

Выполнил:

Студент группы 21ВВ4

Гребенев А.С.

Принял:

Акифьев И.В.

Пенза 2022

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

Пояснительная записка

К курсовому проектированию
по курсу «Логика и основы алгоритмизации
в инженерных задачах»
на тему «Реализация алгоритма Дейкстры»

Выполнил:

Студент группы 21ВВ4

Гребнев А.С.

Принял:

Акифьев И.В.

А.С. Гребнев
30.12.2021.

Пенза 2022

Объем работы по курсу

1. Расчетная часть

Ручной расчет работы алгоритма

2. Графическая часть

Время алгоритма в формате блок-схем

3. Экспериментальная часть

Тестирование программы

Результаты работы программы на тестовые данные

Срок выполнения проекта по разделам

1 Изучение теоретической части курса

2 Разработка алгоритмов программы

3 Разработка программы

4 Тестирование и завершение разработки программы

5 Оформление письменной работы

6

7

8

Дата выдачи задания "10" сентября

Дата защиты проекта " " "

Руководитель

Акиров И.В.

Задание получил

"10"

сентября

2022 г.

Студент

Гребнев Александр Сергеевич

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ

«__» __ 20__

ЗАДАНИЕ

на курсовое проектирование по курсу

Имена и основы алгоритмизации в информационных задачах

Студенту Гребенцову Александру Сергеевичу

Группа 21ВВ41

Тема проекта Реализация алгоритма Дейкстры

Исходные данные : (технические требования) на проектирование

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

Паспортный документ должен содержать:

1. Постановка задачи

2. Определение задачи

3. Описание алгоритма постановки задачи

4. Пример ручного решения задачи и вычисления на небольшом участке работы алгоритма

5. Описание самой программы

6. Текст

7. Список литературы

8. Листинг программы

9. Результаты работы программы

Оглавление

Реферат	8
Введение	9
1. Постановка задачи	10
2. Теоретическая часть задания.....	11
3. Описание алгоритма программы.....	12
4. Описание программы.....	13
5. Тестирование	15
6. Ручной расчёт задачи	17
Заключение	19
Список литературы	20
Приложение А	21

Реферат

Отчёт стр. 20 , рисунков 14, таблиц 1.

ГРАФ, ТЕОРИЯ ГРАФОВ, ДОСТИЖИМОСТЬ, АЛГОРИТМ
ДЕЙКСТРЫ.

Цель исследования – разработка программы, в которой реализован алгоритм Дейкстры.

Введение

Целью данной курсовой работы является создание программы, которая будет искать кратчайший путь до вершины.

Алгоритм Дейкстры — алгоритм на графах, изобретённый нидерландским учёным Эдсгером Дейкстрой в 1959 году. Находит кратчайшие пути от одной из вершин графа до всех остальных. Алгоритм работает только для графов без рёбер отрицательного веса. Алгоритм широко применяется в программировании, например, его используют протоколы маршрутизации OSPF и IS-IS.

В качестве среды разработки мною была выбрана среда Microsoft Visual Studio 2019, язык программирования – C.

1. Постановка задачи

Требуется разработать программу, которая представит граф в виде матрицы смежности, найдет кратчайший путь до вершин.

Программа должна иметь текстовое или графическое меню.

При генерации данных должны быть предусмотрены граничные условия. Программа должна работать так, чтобы пользователь вводил количество вершин для генерации матрицы.

После обработки этих данных пользователь вручную заполнял матрицу. Необходимо сделать возможность случайного заполнения матрицы.

Необходимо предусмотреть различные исходы поиска, чтобы программа не выдавала ошибок и работала правильно.

Результат выполнения программы должен сохраняться в файл.

2. Теоретическая часть задания

Для того, чтобы реализовать алгоритм Дейкстры нужно знать, что граф не должен иметь ребра отрицательного веса.

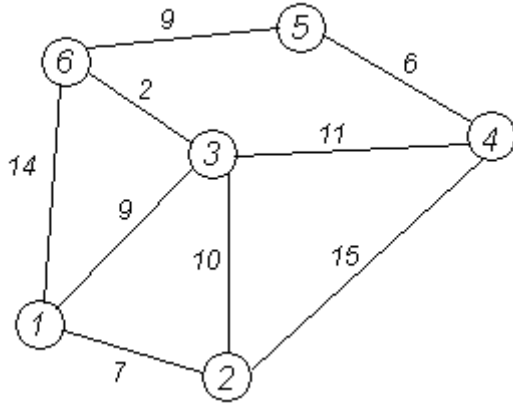


Рисунок 1 – Пример правильного построения графа

3. Описание алгоритма программы

Я реализовал алгоритм данным образом:

```
do {
    minindex = 10000;
    min = 10000;
    for (int i = 0; i < SIZE; i++)
    { // Если вершину ещё не обошли и вес меньше min
        if ((v[i] == 1) && (d[i] < min))
        { // Переприсваиваем значения
            min = d[i];
            minindex = i;
        }
    }
    // Добавляем найденный минимальный вес
    // к текущему весу вершины
    // и сравниваем с текущим минимальным весом вершины
    if (minindex != 10000)
    {
        for (int i = 0; i < SIZE; i++)
        {
            if (a[minindex][i] > 0)
            {
                s = min + a[minindex][i];
                if (s < d[i])
                {
                    d[i] = s;
                }
            }
        }
        v[minindex] = 0;
    }
} while (minindex < 10000);
```

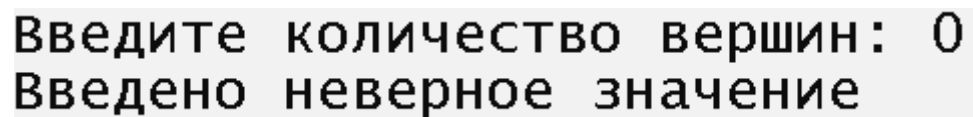
4. Описание программы

Каждой вершине из V сопоставим метку — минимальное известное расстояние от этой вершины до a .

Алгоритм работает пошагово — на каждом шаге он «посещает» одну вершину и пытается уменьшать метки.

Работа алгоритма завершается, когда все вершины посещены.

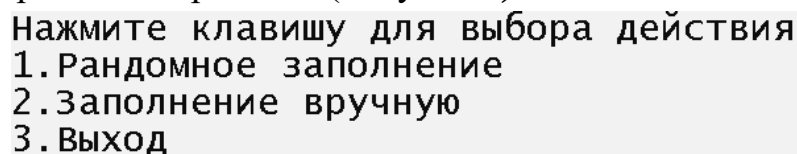
Программа начинается с вывода в консоль текстового меню. Пользователю предлагается задать количество вершин в графе. Если пользователь ввёл меньше 1 вершины, программа выведет в консоль сообщение «Введено неверное значение» и запросит повторный ввод (Рисунок 2).



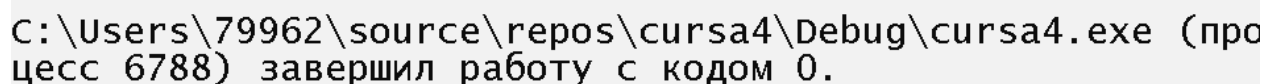
```
Введите количество вершин: 0
Введено неверное значение
```

Рисунок 2 – пример ошибки ввода вершин

Если пользователь ввел правильно число вершин, то программа выведет текстовое меню. В меню пользователю предлагают выбрать вариант заполнения матрицы смежности: заполнить матрицу случайным образом с помощью функции `rand()`, нажав клавишу 1, либо заполнить матрицу вручную с клавиатуры, нажав клавишу 2. Если пользователь нажмет клавишу 3 программа завершится. (Рисунок 3).



```
Нажмите клавишу для выбора действия
1. Рандомное заполнение
2. Заполнение вручную
3. Выход
```



```
C:\Users\79962\source\repos\cursa4\Debug\cursa4.exe (процесс 6788) завершил работу с кодом 0.
```

Рисунок 3 –выбор выхода из программы

Выбор пользователем режима заполнения матрицы был реализован с помощью функции `_getch()`.

При заполнении матрицы смежности пользователем учитывается, что в графе не может быть отрицательный вес ребра, то есть можно будет ввести только числа больше или равные 0.

После заполнения матрицы программа выводит её в консоль и в файл, который пользователь именует сам (Рисунок 4,5)

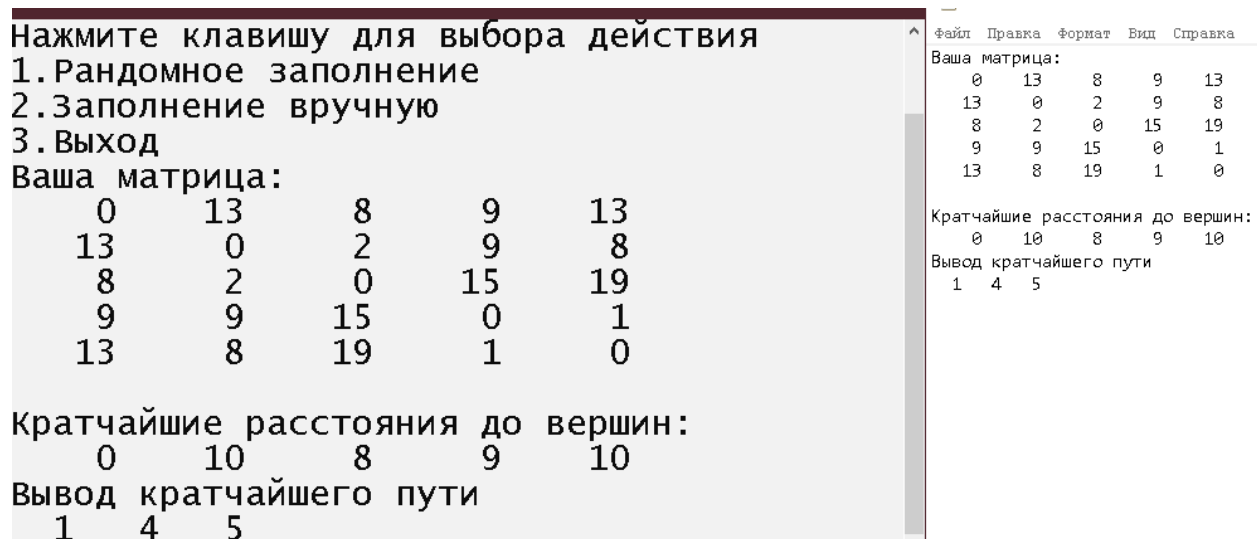


Рисунок 4 – вывод случайно матрицы в консоль и файл

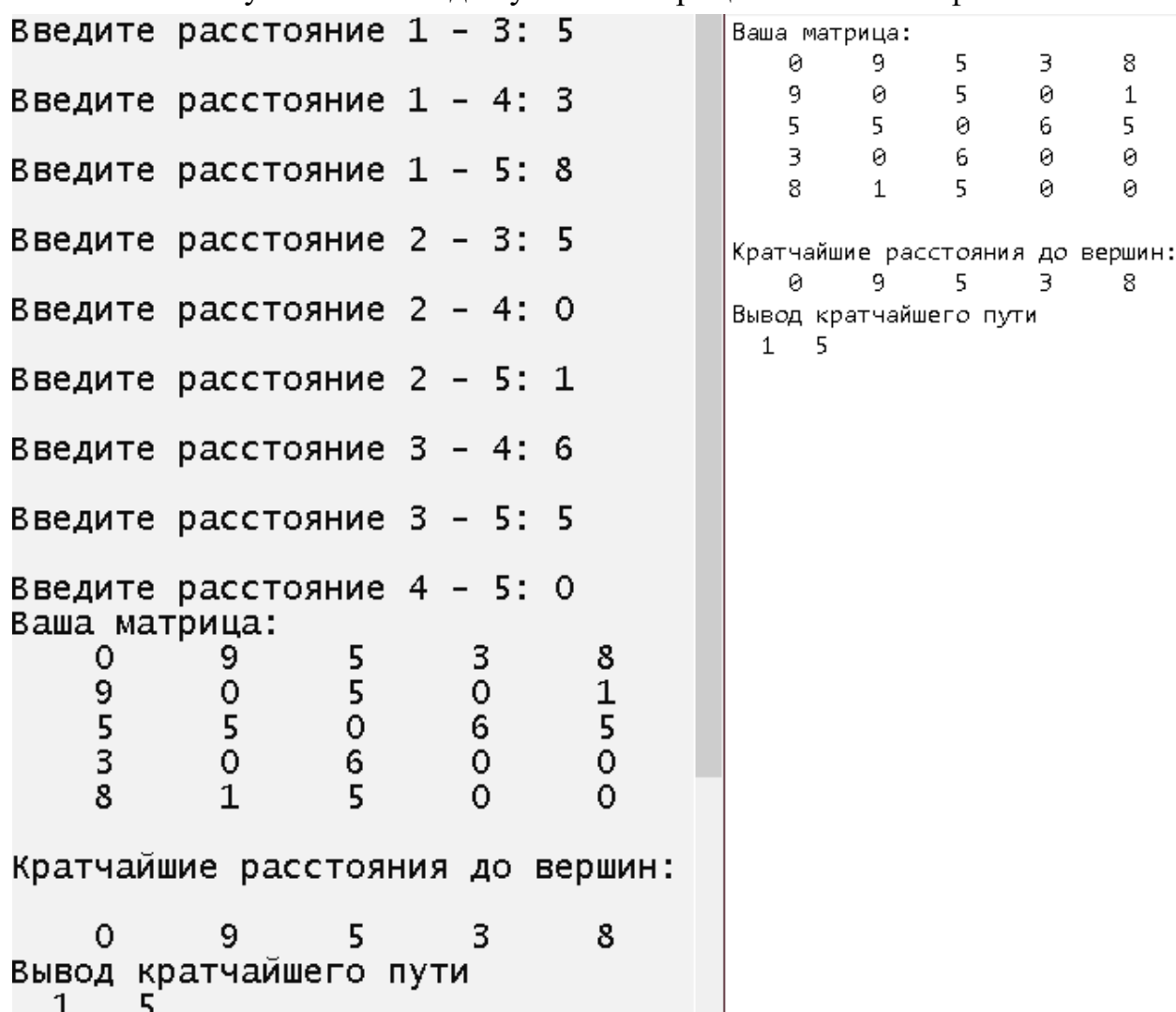


Рисунок 5 – ручной ввод матрицы, вывод в консоль и файл

После создания матрицы программа найдет и выведет кратчайшие пути до каждой вершины, после выведет кратчайший путь до конечной вершины (через какие вершины нужно идти).

5. Тестирование

Выполнение программы с 5 вершинами в графе при случайном заполнении матрицы (Рисунок 6)

```
Нажмите клавишу для выбора действия
1.Рандомное заполнение
2.Заполнение вручну
3.Выход
Ваша матрица:
  0    13    8    9    13
 13    0    2    9    8
  8    2    0   15   19
  9    9   15    0    1
 13    8   19    1    0

Кратчайшие расстояния до вершин:
  0    10    8    9   10
Вывод кратчайшего пути
  1    4    5
```

Рисунок 6 – тестирование случайного заполнения

Выполнение программы с 5 вершинами в графе при ручном заполнении матрицы (Рисунок 7)

```
Введите расстояние 1 - 3: 5
Введите расстояние 1 - 4: 3
Введите расстояние 1 - 5: 8
Введите расстояние 2 - 3: 5
Введите расстояние 2 - 4: 0
Введите расстояние 2 - 5: 1
Введите расстояние 3 - 4: 6
Введите расстояние 3 - 5: 5
Введите расстояние 4 - 5: 0
Ваша матрица:
  0    9    5    3    8
  9    0    5    0    1
  5    5    0    6    5
  3    0    6    0    0
  8    1    5    0    0

Кратчайшие расстояния до вершин:
  0    9    5    3    8
Вывод кратчайшего пути
  1    5
```

Рисунок 7 – тестирование ручного заполнения

Выполнение программы с некорректным вводом вершин
(Рисунок 8)

```
Введите количество вершин: 0
Введено неверное значение
Введите количество вершин: -1
```

Рисунок 8 – тестирование матрицы с неправильным вводом

Выполнение программы с некорректным вводом веса рёбер
(Рисунок 9)

```
Введите расстояние 1 - 2: -1
Введено неверное значение
```

Рисунок 9 – Тестирование с неправильным вводом

Описание теста	Ожидаемый результат	Полученный результат
Запуск программы	Вывод текстового меню с выбором режима генерации матрицы смежности графа	верно
Выбор генерации матрицы	Вывод сообщения о выборе количества вершин в графе	верно
Выбор количества вершин в графе	Вывод матрицы, если был выбран автоматический метод генерации либо заполнение и вывод матрицы, если был выбран ручной метод генерации	верно
Нахождение кратчайшего пути	Вывод кратчайшего пути	верно

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

6. Ручной расчёт задачи

Пример работы программы для графа с 3 вершинами (Рисунок 10)

```
Введите количество вершин: 3
Нажмите клавишу для выбора действия
1. Рандомное заполнение
2. Заполнение вручную
3. Выход
Ваша матрица:
    0    3    2
    3    0   14
    2   14    0

Кратчайшие расстояния до вершин:
    0    3    2
Вывод кратчайшего пути
    1    3
```

Рисунок 10 – выполнение для графа из 3 вершин

Воссоздадим граф по заданной матрице смежности (Рисунок 11)

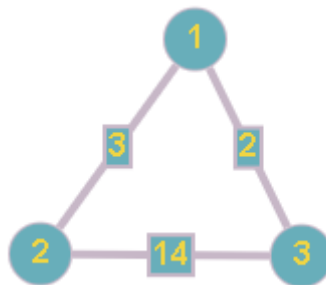


Рисунок 11 – граф из 3 вершин

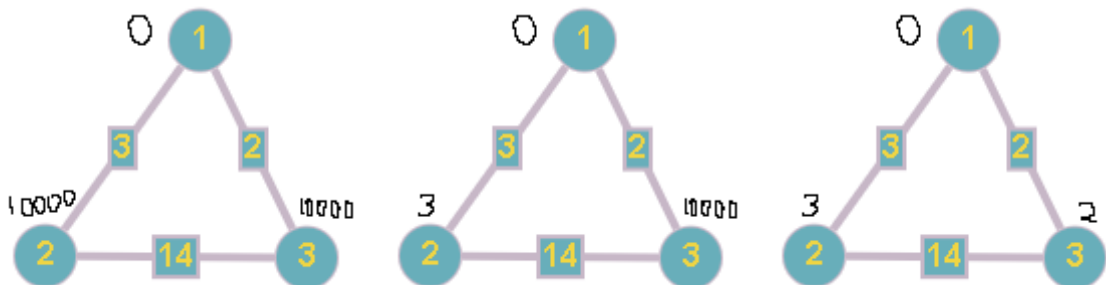


Рисунок 12,13,14 – работа алгоритма Дейкстры

Алгоритм присваивает всем вершинам практически недостижимое число. Используя обход в ширину, проверяет вес пути с тем, который присвоен вершине. Если вес меньше, то вершине присваивается новое число – вес пути.

Ручной расчёт совпал с программным, следовательно, программа работает верно.

Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритма Дейкстры в Microsoft Visual Studio 2019. При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания матриц смежностей, основанных на теории графов. Углублены знания языка программирования С.

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. Алгоритм Дейкстры
<https://w.wiki/5BW4>

Приложение А

Листинг программы

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <locale.h>
#include <time.h>
int main()
{
    FILE* fp;
    int s = -1, minindex, min, SIZE = 0, menu = 0;
    int **a; // матрица связей
    int *d; // минимальное расстояние
    int *v; // посещенные вершины
    int begin_index = 0;
    char name[20];
    char klav;
    system("chcp 1251");
    system("cls");
    setlocale(LC_ALL, "RUS");
    srand(time(NULL));
    printf("Введите название файла (до 20 символов): ");
    gets_s(name);
    fp = fopen(name, "a");
    // меню
    printf("Введите количество вершин: ");
    while (SIZE <= 0) {
        scanf_s("%d", &SIZE);
        if (SIZE <= 0) {
            printf("Введено неверное значение\n");
            printf("Введите количество вершин: ");
        }
    }
    a = (int**)malloc(SIZE * sizeof(int*));
    for (int i = 0; i < SIZE; i++) a[i] = (int*)malloc(SIZE * sizeof(int));
    while (menu == 0) {
        printf("\nНажмите клавишу для выбора действия\n1.Рандомное\n2.Заполнение вручную\n3.Выход\n");
        klav = _getch();
        if (klav == 49) {
            for (int i = 0; i < SIZE; i++)
            {
                a[i][i] = 0;
                for (int j = i + 1; j < SIZE; j++) {
                    a[i][j] = rand() % 20;
                    a[j][i] = a[i][j];
                }
            }
            menu = 1;
        }
        if (klav == 50) {
            for (int i = 0; i < SIZE; i++) {
                a[i][i] = 0;
                for (int j = i + 1; j < SIZE; j++) {
                    printf("\nВведите расстояние %d - %d: ", i + 1, j + 1);
                    scanf_s("%d", &s);
                    if (s < 0) {
                        while (s < 0) {
```

```

        printf("Введено неверное значение");
        printf("\nВведите расстояние %d - %d: ", i + 1, j + 1);
        scanf_s("%d", &s);
    }
}
a[i][j] = s;
a[j][i] = s;
}
}

menu = 1;
}
if (klav == 51) exit(0);
}
// Вывод матрицы связей
printf("Ваша матрица:\n");
fprintf(fp, "Ваша матрица:\n");
for (int i = 0; i < SIZE; i++)
{
    for (int j = 0; j < SIZE; j++) {
        printf("%5d ", a[i][j]);
        fprintf(fp, "%5d ", a[i][j]);
    }
    printf("\n");
    fprintf(fp, "\n");
}
//Инициализация вершин и расстояний
v = (int*)malloc(SIZE * sizeof(int));
d = (int*)malloc(SIZE * sizeof(int));
for (int i = 0; i < SIZE; i++)
{
    d[i] = 10000;
    v[i] = 1;
}
d[begin_index] = 0;
// Шаг алгоритма
do {
    minindex = 10000;
    min = 10000;
    for (int i = 0; i < SIZE; i++)
    { // Если вершину ещё не обошли и вес меньше min
        if ((v[i] == 1) && (d[i] < min))
        { // Переприсваиваем значения
            min = d[i];
            minindex = i;
        }
    }
    // Добавляем найденный минимальный вес
    // к текущему весу вершины
    // и сравниваем с текущим минимальным весом вершины
    if (minindex != 10000)
    {
        for (int i = 0; i < SIZE; i++)
        {
            if (a[minindex][i] > 0)
            {
                s = min + a[minindex][i];
                if (s < d[i])
                {
                    d[i] = s;
                }
            }
        }
    }
    v[minindex] = 0;
}

```

```

    }
} while (minindex < 10000);
// Вывод кратчайших расстояний до вершин
printf("\nКратчайшие расстояния до вершин: \n");
fprintf(fp, "\nКратчайшие расстояния до вершин: \n");
for (int i = 0; i < SIZE; i++) {
    printf("%5d ", d[i]);
    fprintf(fp, "%5d ", d[i]);
}
// Восстановление пути
int *ver; // массив посещенных вершин
int end = SIZE - 1; // индекс конечной вершины
ver = (int*)malloc(SIZE * sizeof(int));
ver[0] = end + 1; // начальный элемент - конечная вершина
int k = 1; // индекс предыдущей вершины
int weight = d[end]; // вес конечной вершины

while (end != begin_index) // пока не дошли до начальной вершины
{
    for (int i = 0; i < SIZE; i++) // просматриваем все вершины
        if (a[i][end] != 0) // если связь есть
        {
            int s = weight - a[i][end]; // определяем вес пути из предыдущей вершины
            if (s == d[i]) // если вес совпал с рассчитанным
            {
                // значит из этой вершины и был переход
                weight = s; // сохраняем новый вес
                end = i; // сохраняем предыдущую вершину
                ver[k] = i + 1; // и записываем ее в массив
                k++;
            }
        }
}
// Вывод пути (начальная вершина оказалась в конце массива из k элементов)
printf("\nВывод кратчайшего пути\n");
fprintf(fp, "\nВывод кратчайшего пути\n");
for (int i = k - 1; i >= 0; i--) {
    printf("%3d ", ver[i]);
    fprintf(fp, "%3d ", ver[i]);
}
fclose(fp);
return 0;
free(a);
free(v);
free(d);
free(ver);
}

```