Министерство науки и высшего образования
Российской Федерации

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра теоретической и прикладной информатики

Курсовая работа по дисциплине
«Информационные технологии и основы программирования»

Факультет:	Прикладной математики и информатики
Группа:	ПМИ-12
Студент:	Панасенко Сергей Дмитриевич
Преподаватель:	Еланцева Ирина Леонидовна

Новосибирск
2022

1. Условие задачи

Задана система односторонних дорог. Найти путь, соединяющий города A и B и не проходящий через заданное множество городов.

2. Анализ задачи

2.1 Исходные данные задачи

В первой строке и второй строке входного файла input.txt находятся начальная и конечная вершины графа соответственно (целые числа).

Начиная с третьей строки, построчно вводятся E целых чисел, где E – количество ребёр в графе.

В конце вводятся 0 <= k <= N закрытых городов (целые числа), также построчно. Чтобы окончить ввод, требуется ввести 0.

Пример входных данных:

1

5

0 7 9 0 0 14

7 0 10 15 0 0

9 10 0 11 0 2

0 15 11 0 6 0

0 0 0 6 0 9

14 0 2 0 9 0

3

4

0

2.2 Результат

Если введены корректные данные, удовлетворяющие условию задачи, то выходном файле output.txt будет выглядеть следующим образом:

Одна строка, состоящая из узлов графа, представляющих путь из начальной вершины до конечной.

2.3 Решение

Математическая модель – ориентированный, невзвешенный, простой, несвязный, помеченный граф.

Определение:

Ориентированный граф – граф, ребра которого имеют направления.

Невзвешенный граф – граф, в котором все вершины равнозначны.

Простой граф - граф, две вершины которого связаны не более чем одним ребром.

Несвязный граф – граф, допускающий вершину не связанную с другими вершинами ребрами.

Помеченный граф – граф, вершины которого отображают какое-либо название. В данной задаче, метками являются номера вершин.

Анализ:

Для решения задачи необходимы корректно введённые входные данные, иначе решение найдено не будет. Корректно введенными входными данными являются верные количество вершин графа, матрица смежности, содержащая в себе только 0 и 1, а также массив закрытых городов. Чтобы завершить ввод закрытых городов, требуется ввести 0. В случае некорректно введённых данных будет выведено соответствующее сообщение Для поиска пути из начальной вершины до конечной вершины будем использовать алгоритм Дейкстры. Если путь найти удастся, выведем его на экран. В противном случае выведем сообщение об отсутствии пути.

Формальная постановка задачи:

В ориентированном, невзвешенном, простом, несвязном помеченном графе построить путь из начальной вершины до конечной вершины, непроходящей через заданное множество вершин.

Следовательно, данную задачу можно разбить на 2 подзадачи.

1. Обход графа при помощи алгоритма Дейкстры.
2. Построение пути по полученным данным после непосредственного обхода графа.

Алгоритм решения поставленных подзадач:

Для того чтобы приступить к решению поставленных подзадач, мы должны внести данные из исходного файла в матрицу смежности и массив закрытых городов. В противном случае будет выведено сообщение об ошибке.

1. Обход графа

В качестве решения будем использовать алгоритм Дейкстры поиска кратчайшего пути в графе. Поскольку граф невзвешенный, для удобства примем, что каждое ребро по умолчанию весит 1. Если путь из начальной вершины до конечной существует, то нужно его вывести. В противном случае вывести сообщение о том, что путь не существует. Перед запуском алгоритма сделаем так, чтобы закрытые города в процессе работы игнорировались. Для этого обнулим те элементы матрицы смежности, которые соответствуют каждому закрытому городу из массива. В начале алгоритма до каждой вершины бесконечное расстояние. Далее в соответствии с алгоритмом, будем производить обход графа, сохраняя расстояния от начальной вершины до посещенных вершин в массив, если вершину посетить удалось. В противном случае, расстояние до вершины в массиве так и останется бесконечным.

2. Построение пути

   Затем нам нужно будет восстановить полученный путь по расстояниям до вершин, сохраненных в массиве. На этом этапе можно совершить следующую проверку. Если расстояние до конечной вершины осталось бесконечным, значит путь до неё не был найден и об этом требуется вывести соответствующее сообщение. Путь восстанавливается с конца. Будем сравнивать расстояния конечной вершины до смежных с ней с расстояниями, сохраненными в массиве, и по ним восстанавливать путь до начальной вершины, т.е. если расстояние от конечной вершины до смежной с ней совпадает с расстоянием до неё, сохраненным в массиве, значит эта вершина содержится в требуемом пути. Переходим в неё и повторяем предыдущий шаг, пока не доберёмся до начальной вершины.

3. Структуры данных, используемых для представления исходных данных и результатов задачи

Внешнее представление данных:

Представление входных данных:

Целое число, количество вершин.

Строки, соответствующие строкам матрицы смежности.

Строка, соответствующая массиву закрытых городов.

Представление выходных данных:

Если решение задачи не было найдено, выводится строка с сообщением об этом.

Если решение найдено, то выводится строка, содержащая путь от начальной вершины до конечной.

Внутреннее представление данных:

Представление входных данных:

Целочисленная переменная, которая будет динамически задавать размерность матрице.

Двумерный массив для матрицы смежности графа.

Одномерный массив для хранения вершин (закрытых городов).

Представление выходных данных:

Строка, содержащая искомый путь.

4. Укрупненный алгоритм решения задачи

4.1. Укрупненный алгоритм решения задачи

{

Открываем файл.

Считываем входные данные.

Заполняем матрицу смежностей графа.

Заполняем массив закрытых городов.

Обнуляем элементы матрицы смежности, соответствующие закрытым городам.

Обходим граф, собираем информацию о расстояниях для каждой вершины.

Строим путь по полученным данным.

}

4.2 Укрупненный алгоритм обхода графа

{

Записываем в массив расстояния до смежных вершин от текущей, если оно меньше, чем то, которое уже записано, перезаписываем текущую ячейку.

Помечаем текущую вершину, как посещенную.

Переходим в следующую вершину.

Повторяем предыдущие действия, пока все вершины графа не будут помечены, как пройденные.

}

4.3 Укрупнённый алгоритм построения пути

{

Рассматриваем смежные с конечной вершины.

Если расстояние до конечной вершины бесконечное, выводим сообщение об этом.

Иначе сравниваем расстояние до вершины с соотвествующим ему расстоянием, записанным в массив с расстояними до всех вершин.

Если они совпадают, то перейти в эту вершину и повторить предыдущие действия, пока текущая вершина не совпадёт с начальной вершиной.

}

5. **Структура программы**

Текст программы разбит на два модуля.

Модуль 1 – dijkstra.h – содержит функции, осуществляющие работу модуля 2.

Модуль 2 – source.cpp (main) – считывание с файла, реализация алгоритма программы, запись в файл.

5.1 Состав модуля dijkstra.h

Функция *dijkstra:*

– назначение:

Заполнение массива с расстояними до каждой из вершин

Модификация массива с посещенными вершинами

- прототип функции:

```
void Dijkstra(int a[N][N], int visited[N], int dist[N], int temp, int begin_index, int end_index);
```

-параметры:

a – матрица смежности размерности N

visited – одномерный массив, содержащий информацию о посещаемости вершин

dist – одномерный массив, содержащий информацию о расстояниях до каждой из вершин.

temp – выделенная целочисленная переменная, в которую будет записываться вес текущей вершины

begin_index – индекс начальной вершины

end_index – индекс конечной вершины

Функция *build_path:*

– назначение:

Построение пути от начальной вершины до конечной.

- прототип функции:

```
void Build_Path(int a[N][N], int dist[N], int begin_index, int end_index);
```

-параметры:

a – матрица смежности размерности N

dist – одномерный массив, содержащий информацию о расстояниях до каждой из вершин.

begin_index – индекс начальной вершины

end_index – индекс конечной вершины

5.2 Состав модуля main.cpp (main)

Главная функция main:

– назначение:

Определение входных и выходных данных. Чтение данных с файла. Реализацию алгоритма по решению поставленной задачи(проверка на обязательные условия, решение подзадач).Организация связи с пользователем. Запись результата в файл.

- прототип функции:

int main( )

6. Текст программы на языке Си (C++)

DIJKSTRA.H

```
#include <iostream>
#include <cstdlib>
#include <limits>
#define INF std::numeric_limits<int>::max()
#define N 6

using namespace std;

void Build_Path(int a[N][N], int dist[N], int begin_index, int end_index)
{
  if (dist[end_index] == INF)
  {
    std::cout << "Path is not found." << std::endl;
    return;
  }

  int ver[N];
  ver[0] = end_index + 1;
  int k = 1;
  int weight = dist[end_index];

  while (end_index != begin_index)
```

```
  {
    for (int i = 0; i < N; i++)
      if (a[i][end_index] != 0)
      {
        int temp = weight - a[i][end_index];
        if (temp == dist[i])
        {
          weight = temp;
          end_index = i;
          ver[k] = i + 1;
          k++;
        }
      }
  }

  std::cout << "Path" << std::endl;
  for (int i = k - 1; i >= 0; i--)
    std::cout << ver[i] << " " << std::endl;
}

void Dijkstra(int a[N][N], int visited[N], int dist[N], int temp, int begin_index, int
end_index)
{
  int min_index;
  int min;

  do {
    min_index = INF;
    min = INF;
    for (int i = 0; i < N; i++)
    {
      if ((visited[i] == 1) && (dist[i] < min))
      {
        min = dist[i];
        min_index = i;
```

```
    }
   }

   if (min_index != INF)
   {
    for (int i = 0; i < N; i++)
    {
     if (a[min_index][i] > 0)
     {
      temp = min + a[min_index][i];
      if (temp < dist[i])
       dist[i] = temp;
     }
    }
    visited[min_index] = 0;
   }
  } while (min_index < INF);

  Build_Path(a, dist, begin_index, end_index);
}
```

MAIN.CPP

```
#include "dijkstra.h"

using namespace std;

int main()
{
  int a[N][N];
  int dist[N];
  int visited[N];
  int closed_cities[N];
  int closed_city;
  int temp;
  int begin_index;
```

```
int end_index;

std::cout << "Input Start Point: ";
std::cin >> begin_index;
std::cout << "Input Finish Point: ";
std::cin >> end_index;
begin_index--;
end_index--;

for (int i = 0; i < N; i++)
{
  a[i][i] = 0;
  for (int j = i + 1; j < N; j++)
  {
    std::cout << "Type distance " << i + 1 << " - " << j + 1 << ": ";
    std::cin >> temp;
    a[i][j] = temp;
    a[j][i] = temp;
  }
}

for (int i = 0; i < N; i++)
{
  for (int j = 0; j < N; j++)
    std::cout << a[i][j] << " ";
  std::cout << std::endl;
}

for (int i = 0; i < N; i++)
{
  dist[i] = INF;
  visited[i] = 1;
  closed_cities[i] = 0;
}
dist[begin_index] = 0;
```

```
  //ввод закрытых городов
  std::cout << "Type Closed Cities (0 to stop)" << std::endl;
  do
  {
    std::cin >> closed_city;
    if (closed_city == 0)
      break;
    if (!(closed_city > 0 && closed_city < N + 1))
    {
      std::cout << "Error" << std::endl;
      exit(1);
    }
    closed_cities[closed_city - 1] = 1;
  } while(closed_city != 0);

  for (int i = 0; i < N; i++)
    if (closed_cities[i] == 1)
      for (int j = 0; j < N; j++)
      {
        a[i][j] = 0;
        a[j][i] = 0;
      }

  Dijkstra(a, visited, dist, temp, begin_index, end_index);

  return 0;
}
```

7.Тесты

ТЕСТ 1:

input.txt	**output.txt**	**консоль**
1		**Error**

5 0 1 1 0 0 1 1 0 1 1 0 0 1 1 0 1 0 1 0 1 1 0 1 0 0 0 0 1 0 1 1 0 1 0 1 0 12		

Комментарий: Вводим некорректный номер вершины во время ввода закрытых городов

ТЕСТ 2:

input.txt	output.txt	консоль
1 5 0 1 1 0 0 1 1 0 1 1 0 0 1 1 0 1 0 1 0 1 1 0 1 0 0 0 0 1 0 1 1 0 1 0 1 0 0	1 6 5	

Комментарий: Введены корректный граф и корректное множество закрытых городов. Граф связный и циклический. Выводится путь с наименьшим количеством ребёр между начальной и конечной вершиной.

ТЕСТ 3:

input.txt	output.txt	консоль
1 5 0 1 1 0 0 1 1 0 1 1 0 0 1 1 0 1 0 1 0 1 1 0 1 0 0 0 0 1 0 1 1 0 1 0 1 0 2 6 0	1 3 4 5	

Комментарий: Введены корректный граф и корректное множество закрытых городов. Вершины 2 и 6 являются закрытыми городами. Таким

образом остаётся единственный путь до конечной вершины 1 3 4 5, который и выводится в файл.

ТЕСТ 4:

input.txt	output.txt	консоль
1 5 0 1 1 0 0 1 1 0 1 1 0 0 1 1 0 1 0 1 0 1 1 0 1 0 0 0 0 1 0 1 1 0 1 0 1 0 5 0	Path is not found.	

Комментарий: Введены корректный граф и корректное множество закрытых городов. Вершина 5, которая была обозначена, как конечная, является закрытым городом. Построить путь до неё невозможно, о чём и выводится соответствующее сообщение.

ТЕСТ 5:

input.txt	output.txt	консоль
1 5 0 1 1 0 0 1 1 0 1 1 0 0 1 1 0 1 0 1 0 1 1 0 1 0 0 0 0 1 0 1 1 0 1 0 1 0 1 0	Path is not found.	

Комментарий: Введены корректный граф и корректное множество закрытых городов. Вершина 1, которая была обозначена, как начальная, является закрытым городом. И так как изначально она «отрезана» от всех остальных вершин, построить путь до конечной вершины невозможно, о чём и выводится соответствующее сообщение.

8. Список использованной литературы

Хиценко, В.П. Структуры данных и алгоритмы: методические указания к курсовой работе для 1 курса ФПМИ (направление 010500 - Прикладная математика и информатика, специальность 010503 - Математическое обеспечение и администрирование информационных систем) дневного отделения / В.П. Хиценко, Т.А. Шапошникова. – Новосибирск : Изд-во НГТУ, 2008. – 55 с.