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 информатики

Группа: ПМИ-21

Студент: Комарова Анастасия Валерьевна

Преподаватель: Еланцева Ирина Леонидовна

НОВОСИБИРСК

2023

**1.Условие задачи**

Найти такую вершину заданного ориентированного графа, которая принадлежит каждому пути между двумя выделенными (различными) вершинами и отлична от каждой из них.

**2.Анализ задачи**

**2.1Исходные данные задачи**

В первой строке файла “graf.txt” вводится количество вершин (целое число). Затем, начиная со второй строки и до конца файла, в файле хранится список ребер вида «вершина\_начала вершина\_конца». Вершины графа нумеруются с цифры 1.

В первой строке файла “need.txt” записываются две выделенные вершины вида «вершина\_начала вершина\_конца», между которыми будет искаться вершина, принадлежащая всем путям.

*Пример входных данных:*

**“graf.txt” “need.txt”**

6 1 6

1 2

2 5

3 4

4 5

5 6

2 3

**2.2.Результат**

Если введенные данные корректны, то в результате выполнения задачи в файл ‘result.txt’ будет записана информация следующего вида:

Информация о вершинах [если такие вершины присутствуют], принадлежащих всем путям между двумя выделенными вершинами или информация о том, что нет путей между двумя выделенными вершинами, или сообщение о том, что нет вершин, принадлежащих всем путям между двумя заданными.

В ином случае в файл ‘result.txt’ будет записана информация об ошибке при открытии файла или сообщение о том, что один из файлов со входными данными пуст, или сообщение о том, что в файле 'need.txt' введены одинаковые вершины, или сообщение о том, что в файле 'need.txt' введены вершины, не принадлежащие графу.

**2.3. Решение**

Математическая модель - ориентированный, невзвешенный, помеченный, граф.

*Определения:*

Графом называется система объектов произвольной природы (вершин) и связок (ребер), соединяющих некоторые пары этих объектов.

Граф G = «M; R» называется ориентированным (орграфом), если найдется дуга (a, b) R такая, что (b, a) не R.

Невзвешенный граф — это взвешенный граф со всеми весами ребер, равными 1. Если не указано иное, все графы по умолчанию считаются невзвешенными.

Помеченный граф — это граф, вершинам которого приписываются определенные метки(названия).

*Анализ:*

Для нахождения решения необходимо введение корректных описанных ранее входных данных. Для поиска всех путей между двумя выделенными вершинами осуществляем рекурсивный обход графа в глубину, затем ищем вершины, принадлежащие всем найденным путям.

*Формальная постановка задачи:*

В ориентированном, невзвешенном, помеченном, циклическом графе найти вершины, принадлежащие всем путям между двумя выделенными вершинами, или вывести сообщение о невозможности этого.

Данную задачу можно разделить на подзадачи:

1.Проверка на корректность входных данных.

2.Построение списка смежности для заданного графа.

3.Обход графа в глубину, поиск всех путей между двумя выделенными вершинами.

4.Поиск вершин, принадлежащих всем заданным путям.

5.Вывод результата программы

*Алгоритм решения поставленных подзадач:*

*1.Проверка на корректность входных данных*

Проверяем, открыты ли файлы “graf.txt” и ‘need.txt’ для чтения. Проверяем, что данные файлы не пустые, а вершины в ‘need.txt’ различны.

2. *Построение списка смежности для заданного графа.*

Считываем количество вершин в графе из первой строчки файла ‘graf.txt’, создаем двумерный массив (с элементами типа bool) размера количество\_вершин \* количество\_вершин, изначально заполняем его значением false (0).

Из файла ‘graf.txt’ построчно считываем ребра между n и m вершинами, присваиваем элементу n\*m в двумерном массиве значение true(1)

*3. Обход графа в глубину, поиск всех путей между двумя выделенными вершинами.*

Для нахождения всех путей между двумя выделенными вершинами в матрице смотрим, с какими вершинами связана исходная вершина, запускаем данный обход для смежных вершин, добавляя вершину, от которой идет обход в путь (путь хранится в массиве размера количество\_вершин с элементами типа int), помечаем вершину посещённой. Когда найден путь между двумя выделенными вершинами записываем вершины, входящие в путь, увеличиваем число найденных путей на один.

*4*. *Поиск вершин, принадлежащих всем заданным путям.*

Среди записанных вершин, принадлежащих путям между двумя выделенными вершинами, ищем те, которые принадлежали путям столько раз, сколько существует путей.

*5.Вывод результата*

Записываем результат нашей программы в файл “result.txt”

1. **Структуры данных, используемых для представления исходных данных и результатов задачи**

*Внешнее представление данных:*

Представление входных данных:

Целое число, количество вершин

Строки из двух чисел, номеров вершин, между которыми есть ребро.

Два числа, номера вершин, между которыми будут искаться все пути.

Представление выходных данных:

Если решение найдено, то выводится строка(и) с информацией о вершине(ах), принадлежащей(их) всем путям между двумя выделенными вершинами.

Если решение не найдено, то выводится сообщение об отсутствии вершин, принадлежащих всем путям между двумя «выделенными» вершинами, или же сообщение об отсутствии путей между вершинами.

Если входные данные представлены некорректно, выводится сообщение об этом.

*Внутреннее представление данных:*

Представление входных данных:

Переменная целого типа, которая будет задавать размерность динамическому двумерному массиву, представляющую собой матрицу смежности

Целые числа, отвечающие за значение true на соответственных им местах в матрице смежности.

Переменные целого типа, первая – от чего будет запущен обход графа, вторая – до какой вершины этот обход осуществляется.

Представление выходных данных:

Строка с результатом работы программы.

**4.Укрупнённый алгоритм решения программы**

**4.1. Укрупненный алгоритм решения задачи**

**{**

Открыли файлы со входными данными для чтения

Открыли файл для записи (если его не существует – создали)

Если файлы успешно открыты и не пустые:

Считали количество вершин графа

Считали номера “выделенных вершин”

Если ‘выделенные вершины’ неодинаковые {

Если файл с графом не пустой {

Если ‘выделенные вершины’ принадлежат данному графу {

Создали двумерный массив (матрицу смежности) размером количество\_вершин \* количество\_вершин посредством вызова подпрограммы **create\_adj**

Пока не считали весь файл с графом{

Записываем в матрицу смежности существующие ребра(посредством установки значения true)

}

Если есть ребро между двумя заданными вершинами сразу выводим сообщение о том, что нет вершины, принадлежащей всем путям

Иначе{

Создаем динамический массив, отвечающий за количество посещение вершины при поиске всех путей, размером : количество\_вершин

Заполняем этот массив изначально

нулями

Создаем динамический массив, отвечающий за информацию о том, посещена ли вершина при поиске всех путей, размером : количество\_вершин

Создаем динамический массив, отвечающий за хранение текущего найденного пути, размером : количество\_вершин

Запускаем обход графа посредством вызова подпрограммы **detour**

**}**

Если количество найденных путей между вершинами 0, выводим сообщение об этом

Иначе{

Если количество посещений вершины равно количеству всех найденных путей и это не ‘выделенная вершина’, то выводим, что данная вершина принадлежит всем путям между двумя выделенными вершинами

Иначе выводим сообщение, что такой вершины нет

}

}

Иначе выводим сообщение, что файл с графом пуст

}

Иначе выводим сообщение, что данная(ые) вершина(ы) не принадлежит(ат) заданному графу

}

Иначе выводим сообщение, что файл 'need.txt' пуст или введена неподходящая условию задачи вершина 0

Иначе выводим сообщение, что возникла ошибка при открытии файла

**}**

**4.2.Укрупненный алгоритм создания матрицы смежности**

**{**

Выделяем динамическую память для массива указателей на булевы значения, каждый указатель в массиве указывает на массив булевых значений

Создаем динамический двумерный массив размером количество\_вершин \* количество\_вершин. Элементы данного массива имеют тип bool

Присваиваем всем элементам созданного массива значение false (0)

Возвращаем созданный массив

}

**4.3 Укрупненный алгоритм рекурсивного обхода графа в глубину**

**{**

Помечаем вершину посещённой

Добавляем вершину в текущий путь

Если вершина и есть та, к которой ищется путь{

Увеличиваем на единицу количество посещений у всех вершин, принадлежащих данному пути

Увеличиваем количество путей на единицу  
 }

Иначе{

Если существует ребро от данной вершины до какой-либо другой, то запускаем функцию обхода графа для вершины, к которой есть ребро

}

Уменьшаем длину текущего пути

Помечаем вершину не посещённой

}

**5.Структура программы**

Функция detour:

- Назначение:

Поиск всех путей между двумя “выделенными” вершинами, подсчёт количества посещений вершин, принадлежащих путям между двумя “выделенными” вершинами

* Прототип функции:

void detour(int v, int target, bool\*\* g, int\* path, int path\_len, int n, int\* kol, bool \*mark)

- Параметры:

int v – текущая вершина пути (при первом запуске – первая “выделенная” вершина

int target – вершина, до которой ищется путь (вторая “выделенная” вершина)

bool\*\* g – указатель на матрицу смежности графа

int\* path – указатель на текущий путь

int path\_len – длина текущего пути

int n – количество вершин графа

int\* kol – указатель на одномерный массив, отвечающий за количество посещений вершин, принадлежащих путям между двумя “выделенными” вершинами

bool \*mark - указатель на одномерный массив, отвечающий за контроль посещения вершин

Функция **create\_adj**:

* Назначение:

Создание двумерного массива(матрицы смежности) нужного размера с элементами типа bool, изначально имеющими значение false (0)

* Прототип функции:

bool\*\* create\_adj(int kol\_versh)

- Параметры:

int kol\_versh – количество вершин графа (размер матрицы смежности)

**6.Текст программы на языке С++**

#include <iostream>

#include <fstream>

using namespace std;

int path\_count = 0; // количество найденных путей

void detour(int v, int target, bool\*\* g, int\* path, int path\_len, int n, int\* kol, bool \*mark) {

mark[v] = true; //вершина - посещенная

path[path\_len++] = v; //добавили вершину в текущий путь

if (v == target) { //если вершина - та, до которой ищем путь

for (int i = 0; i < path\_len; i++) { //увеличиваем на 1 количество посещений вершин, принадлежащих найденному пути

kol[path[i]]++;

}

path\_count++; //увеличиваем количество найденных путей

}

else {

for (int u = 0; u < n; u++) { //если есть ребро между данной вершимной и какой-либо еще запускаем функцию от смежной данной вершине

if (g[v][u] && !mark[u]) {

detour(u, target, g, path, path\_len, n, kol,mark);

}

}

}

path\_len--; //уменьшаем длину текущего пути

mark[v] = false; //обозначаем вершину непосещённой

}

bool\*\* create\_adj(int kol\_versh) { //создание матрицы смежности

bool\*\* g = new bool\* [kol\_versh];

for (int i = 0; i < kol\_versh; i++) {

g[i] = new bool[kol\_versh];

}

for (int i = 0; i < kol\_versh; i++) { //заполняем созданную матрицу значениями false(0)

for (int j = 0; j < kol\_versh; j++) {

g[i][j] = 0;

}

};

return g; // возвращаем матрицу смежности

}

int main() {

setlocale(LC\_ALL, "Ru");

int sym1, sym2, kol\_versh;

int start, target; // объявляем начальную и конечную вершины пути

bool flag=true;

ifstream graf;

ofstream out("result.txt");

ifstream need\_versh;

need\_versh.open("need.txt");//открываем файл на чтение

graf.open("graf.txt");//открываем файл на чтение

if (graf && need\_versh) { //если файлы на чтение открыты

graf >> kol\_versh; //считываем количество вершин из файла graf.txt

need\_versh >> start; //считываем первую выделенную вершину из файла need.txt

need\_versh >> target; // считываем вторую выделенную вершину из файла need.txt

if (start > 0 && target > 0) { //если файл need.txt не пустой и указанные в нём вершины не 0

if (start != target) { //если "выделенные" вершины неодинаковы

if (kol\_versh >= 0) { //если файл не пустой

if (start <= kol\_versh && target <= kol\_versh) { //если "выделенные" вершины принадлежат данному графу

bool\*\* adj; //объявляем матрицу смежности

adj = create\_adj(kol\_versh); //создаем матрицу смежности и заполняем её значениями false по умолчанию

while (!graf.eof()) { //пока не считали все данные из файла заполняем матрицу смежности

graf >> sym1;

graf >> sym2;

adj[sym1 - 1][sym2 - 1] = 1;

}

if (adj[start - 1][target - 1]) { //если есть ребро между начальной и конечной вершиной - сразу выводим, что вершин нет

out << "Нет вершины, принадлежащей всем путям из " << start << " в " << target;

}

else {

int\* kol = new int[kol\_versh]; //сколько раз посещена вершина

for (int j = 0; j < kol\_versh; j++) {

kol[j] = 0;

}

bool\* mark = new bool[kol\_versh]; //массив посещений вершины

for (int j = 0; j < kol\_versh; j++) {

mark[j] = false;

}

int\* path = new int[kol\_versh]; // массив для хранения текущего пути

detour(start - 1, target - 1, adj, path, 0, kol\_versh, kol, mark); //запускаем функцию поска всех путей и подсчёта вхождений вершин в пути

if (path\_count == 0) { //если не найдено путей между "выделенными" вершинами

out << "Нет путей из " << start << " в " << target << endl;

flag = false;

}

else {

for (int j = 0; j < kol\_versh; j++) {

if (kol[j] == path\_count && (j != start - 1) && (j != target - 1)) { //если если вершина принадлежит всем путям и не является "выделенной"

out << "Вершина, входящая во все пути: " << j + 1 << endl;

flag = false;

}

}

}

if (flag) { //если нет вершин, принадлежащих всем путям

out << "Нет вершины, принадлежащей всем путям из " << start << " в " << target;

}

}

}

else {

out << "Данная(ые) вершина(ы) не принадлежит(ат) заданному графу";

}

}

else {

out << "Файл 'graf.txt' пуст";

}

}

else {

out << "В файле 'need.txt' введены одинаковые вершины";

}

}

else {

out << "Файл 'need.txt' пуст или введена неподходящая условию задачи вершина 0 ";

}

}

else {

out << "Ошибка при открытии файла";

}

return 0;

}

**7. Тесты**

**Тест №1**

|  |  |  |  |
| --- | --- | --- | --- |
| **graf.txt** | **need.txt** | **result.txt** | **Рисунок графа** |
|  | 1 3 | Файл 'graf.txt' пуст |  |

**Комментарий: проверка на пустоту файла с графом**

**Тест №2**

|  |  |  |  |
| --- | --- | --- | --- |
| **graf.txt** | **need.txt** | **result.txt** | **Рисунок графа** |
| 3  1 2  1 3 |  | Файл 'need.txt' пуст или введена неподходящая условию задачи вершина 0 |  |

**Комментарий: проверка на пустоту файла с «выделенными» вершинами**

**Тест №3**

|  |  |  |  |
| --- | --- | --- | --- |
| **graf.txt** | **need.txt** | **result.txt** | **Рисунок графа** |
|  |  | Файл 'need.txt' пуст или введена неподходящая условию задачи вершина 0 |  |

**Комментарий: в первую очередь проверяется полнота файла с выделенными вершинами. Если need.txt пуст, то нет смысла работать с графом и строить его матрицу смежности**

**Тест №4**

|  |  |  |  |
| --- | --- | --- | --- |
| **graf.txt** | **need.txt** | **result.txt** | **Рисунок графа** |
| 4  1 2  1 3  3 2  2 4 | **2 6** | Данная(ые) выделенная(ые) вершина(ы) не принадлежит(ат) заданному графу |  |

**Комментарий: проверка на поиск путей между вершинами, не принадлежащими данному графу**

**Тест №5**

|  |  |  |  |
| --- | --- | --- | --- |
| **graf.txt** | **need.txt** | **result.txt** | **Рисунок графа** |
| 4  1 2  1 3  3 2  2 4 | **2 3** | Нет путей из 2 в 3 |  |

**Комментарий: отсутствие путей между выделенными вершинами**

**Тест №6**

|  |  |  |  |
| --- | --- | --- | --- |
| **graf.txt** | **need.txt** | **result.txt** | **Рисунок графа** |
| 6  1 2  1 3  2 4  2 5  2 6  3 2  5 6  4 6 | **1 6** | Вершина, входящая во все пути: 2 |  |

**Комментарий: существование вершины, принадлежащей всем путям между двумя выделенными вершинами**

**Тест №7**

|  |  |  |  |
| --- | --- | --- | --- |
| **graf.txt** | **need.txt** | **result.txt** | **Рисунок графа** |
| 6  1 2  1 3  2 4  2 5  2 6  3 2  5 6  4 6  3 5 | **1 6** | Нет вершины, принадлежащей всем путям из 1 в 6 |  |

**Комментарий: отсутствие вершины, принадлежащей всем путям между двумя выделенными вершинами**

**Тест №8**

|  |  |  |  |
| --- | --- | --- | --- |
| **graf.txt** | **need.txt** | **result.txt** | **Рисунок графа** |
| 6 | **2 3** | Нет путей из 2 в 3 |  |

**Комментарий: все вершины графа – обособленные.**

**Тест №9**

|  |  |  |  |
| --- | --- | --- | --- |
| **graf.txt** | **need.txt** | **result.txt** | **Рисунок графа** |
| 4  1 2  1 3  3 2  2 4 | **2 2** | В файле 'need.txt' введены одинаковые вершины |  |

**Комментарий: проверка на то, что «выделенные» вершины различны**

**8.Список литературы**

1)С. В. Судоплатов, Е. В. Овчинникова Дискретная математика Новосибирск: НГТУ.- 2011. - 281 с.

2) Хиценко В. П. Структуры данных и алгоритмы: методические указания к курсовой работе для 1 курса ФПМИ (направление 010500-Прикладная математика и информатика, специальность 020303-Математическое обеспечение и администрирование информационных систем) дневного отделения. - Новосибирск : НГТУ, 2008. - 55 с.

3) Тракимус Ю. В. Структуры данных и алгоритмы : учебное пособие / Ю. В. Тракимус, В. П. Хиценко. – Новосибирск : Изд-во НГТУ, 2022. – 127 с.

4) https://skysmart.ru/articles/mathematic/osnovnye-ponyatiya-teorii-grafov