|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Министерство науки и высшего образования Российской Федерации | | | | |
| Федеральное государственное бюджетное образовательное  учреждение высшего образования | | | | |
| «Новосибирский государственный технический университет» | | | | |
|  | path817.png | | |  |
| Кафедра теоретической и прикладной информатики | | | | |
| **Курсовая работа** | | | | |
| **по дисциплине «Структуры данных и алгоритмы»** | | | | |
|  | | | | |
| эмблема_светлая.png | | Факультет: | ПМИ | |
| Группа: | ПМИ-81 | |
| Студенты: | Воронкина Дарья, | |
|  |  | |
| Преподаватель: | Еланцева Ирина Леонидовна | |
|  | | | | |
| Новосибирск | | | | |
| 2019 | | | | |

1. Условие задачи

Найти длину самого длинного простого пути от города А в город В, в заданной системе односторонних дорог.

1. Анализ задачи

2.1 **Исходные данные задачи**: n – количество городов, n ∈ Z. towns[40] – массив городов из заданной системы дорог. Distance[n][n] – матрица весов. Структура Way, элементами которой являются: символьное поле begin[15], символьное поле end[15], целочисленное значение distance.

Система дорог = {Way->begin, Way->end, distance}. Way->begin, Way->end ∈ город, distance ∈ Z.

город = {gj| gj ∈ {подмножество литер-букв}, j = 1, ... , n}

2.2 **Результат:** bestpath – самый длинный путь из города-начала, в город-конец. bestpath ∈ Z.

2.3 **Решение:** Математическая модель системы односторонних дорог – это ориентированный взвешенный мультиграф G=(V, E), с весовой функцией w: E-> Z, причем w(u,v) ≥ 0, для всех (u,v) ∈ Е. При этом вершины соответствуют городам, а дуги – односторонним дорогам. Метка вершины – название города. Дуга (u,v) ведет от вершины u, соответствующей городу-началу дороги, в вершину v, соответствующую городу, являющемуся концом дороги. Вес дуги – это длина дороги. Длина пути, соединяющая вершину u и v – это сумма весов дуг графа, т.е сумма длин дорог этого пути. При этом длина пути от вершины u до вершины v максимальна. Таким образом, чтобы решить задачу необходимо найти все пути из вершины u в вершину v и их длины. Для поиска всех путей из вершины u в вершину v, необходимо обходить граф в глубину, каждый раз сохраняя длину пути из вершины u в v и сравнивать с предыдущим сохраненным значением. Обход продолжать пока не будут, обнаружены все пути. Если пути от u до v не найдено, то пути между этими вершинами не существует. В системе дорог не может быть петель.

**Формальная постановка задачи**: В взвешенном ориентированном мультиграфе найти максимально длинный простой путь из одной вершины графа, в другую.

Простой путь – это такой маршрут в котором ни одна из вершин не повторяется.

Для поиска такого пути возможно просматривать граф в глубину начиная с вершины u – вершины начала пути, каждый раз увеличивая расстояние от одной вершины к другой, за счет складывания значений дуг графа, до тех пор, пока не будет достигнута вершина v – конечная вершина. При достижении конечной вершины, возвращаться по графу обратно к такой вершине, у которой еще есть смежные ребра, при этом каждый раз вычитать из длины пути ту дугу, по которой возвращаемся обратно.

Если система дорог содержит только один город, то невозможно найти такой путь, длина которого будет максимальна. Если в системе дорог только два города, то задача сводится к выводу расстояния между этими городами.

Начальная спецификация алгоритма решения:

Для решения понадобится матрица весов графа, описанная в программе, как Distanceij

Плюсы представления:

1. Быстрый поиск максимального пути по его длинам (весам).

Минусы представления:

1. Расходуется большой объем оперативной памяти.

Основные подзадачи:

1. Поиск максимальных по длине путей между заданными городами.
2. Ввод системы дорог.
3. Структуры данных, используемых для представления исходных данных и результатов задачи

3.1 **Входные данные:**

Внешнее представление данных:Информация о системе дорог хранится в файле “input.txt”. И имеет вид:

<Система-дорог>::=<Количество городов><Список-дорог>

<Список-дорог>::= <дорога>пробел<дорога><Список-дорог>

<дорога>::=<откуда>пробел<куда>пробел<длина-дороги>

<откуда>::=<название>

<куда>::=<название>

<длина-дороги>::= целое число

<количество-городов>::= целое число

<название>::= строка

Если в системе дорог есть изолированные города, то они во внешнем представлении представлены также как и остальные города системы дорог.

Начальный и конечные города читаются с консоли.

**Внутреннее представление системы дорог:** Для алгоритма обхода графа в глубину, представлением выбрана матрица весов. Если есть кратные пути, то хранится путь с наибольшим весом. Чтобы не усложнять матрицу весов, вместо названия городов вершина помечается тем же индексом, что и индекс города в массиве towns. Так как число городов ограничено, то матрица весов представлена размерами n, n.

3.2 **Выходные данные:**

Внешнее представление:

Вывод внутреннего представления графа: «Система дорог представлена в виде матрицы весов, следующим образом: Distance[i][j]», где Distance [i][j]-матрица весов, i = 1, ... , n; j = 1, ... , n.

bestpath – длина максимального пути из начальной вершины в конечную.

Вывод : «Максимальный путь из начального города в конечный bestpath»

В случае если город не найден в системе дорог, вывод: «»

В случае если пути между начальным и конечным городом не найдено, вывод: «»

Внутреннее представление:

int bestpath.

1. Алгоритм решения задачи

4.1 Глобальные переменные: int i = 0 ,path=0 ,bestpath = 0, k=0; FILE \*file;

string towns [40].

4.2 Структура пути:

struct Way {

символьное поле begin[15];

символьное поле end[15];

целочисленное поле distance=0;

};

4.3 Добавление города в массив городов и добавление расстояния пути в матрицу весов:

mark – целочисленное значение, помогающее определить место добавление города в массив, и добавление в матрицу весов

begin – город начала

end – город конца

dis – расстояние между городами

void AddList(string begin, string end, int dis, int \*\*Distance) {

mark = 0

целочисленные значения k = 0, m = 0

for (j от 0 до i)

{

если (begin = towns[j]) {

mark = 1 – метка, что город начала есть в массиве towns

m = j

}

}

for (j от 0 до i)

{

если (end = towns[j]) {

если (mark = 1) {

mark = 3- метка, что город начала и конца есть в списке

k = j

}

иначе {

mark = 2 – метка , что город конца есть в массиве towns

k = j

}

}

}

Если (mark = 1) {

towns[i] = end

Distance[m][i] = dis

i++;

}

Если (mark = 2) {

towns[i] = begin;

Distance[i][k] = dis;

i++;

}

Если (mark = 3) {

Если (Distance[m][k] < dis) {

Distance[m][k] = dis;

}

}

Если ( mark == 0) {

towns[i] = begin;

i++;

towns[i] = end;

Distance[i - 1][i] = dis;

i++;

}

}

4.4 Ввод системы городов:

Distance – массив для хранения матрицы весов

переменная символьного типа symbol;

Указатель на структуру Way \*Inf;

file – указатель на файл с системой дорог;

void ReadTowns(int \*\*Distance) {

Inf = new Way

Чтение из file строк Inf->begin, Inf->end

Чтение из file целочисленного значения Inf->distance

вызов подпрограммы AddList(Inf->begin, Inf->end, Inf->distance, Distance);

symbol = символ из файла

если (symbol = '\n') ReadTowns(Distance)

если (symbol = конец файла) вернуть

}

4.5 Поиск индекса города в массиве towns

f – город чей индекс нужно найти

index - переменная куда сохраняется индекс

int FindIndexTown(string f, int index, int n) {

for (k от 0 до n)

{

Если (f = towns[k]) {

index = k

вернуть k

}

}

вернуть -1 (город не найден)

}

4.6 Инициализация матрицы весов

int \*\*Incialization(int \*\*Distance, int n) {

for (j от 0 до n)

Создание динамического массива Distance[j]

for (j от 0 до n)

{

for (k от 0 до n)

{

Distance[j][k] = 0;

}

}

вернуть массив Distance;

}

4.7 Поиск самого длинного пути от одной вершины в другую, алгоритм обхода графа в глубину с возвратом

indexBegin – индекс начального города

indexEnd - индекс конечного города

Visited – массив посещения вершин

path – длина пути из одной вершины в другую

bestpath – максимально длинный путь из одной вершины в другую

void DFS(int indexBegin, int indexEnd, int \*\*Distance, int \*Visited, int n) {

Если (indexBegin = indexEnd) {

k++;

bestpath = максимальное из(bestpath, path);

return;

}

Visited[indexBegin] = 1;

for (j от 0 до n)

{

Если (Distance[indexBegin][j]) {

Если (Visited[j] == 0) {

path += Distance[indexBegin][j];

DFS(j, indexEnd, Distance, Visited, n);

path -= Distance[indexBegin][j];

}

}

}

Visited[indexBegin] = 0;

}

4.8 Главная функция

main() {

открыть файл “input.txt”

читать из файла целочисленное значение в n

Visited создание одномерного динамического массива

Distance создание двумерного динамического массива

Distance = Incialization(Distance, n);

объявление строк для начального и конечно города fbegin[15], fend[15];

инициализация индекса начала и конца bindex = -1, eindex = -1;

ReadTowns(Distance);

Печатать ("Система дорог представлена в виде матрицы весов, следующим образом:");

for (j от 0 до n)

{

for (k от 0 до n){

Печатать целочисленное значение Distance[j][k]

}

}

Печатать ("Введите город из которого нужно найти максимальный путь: ");

Читать с консоли строку fbegin

Печатать ("Начальный город:", fbegin);

bindex = FindIndexTown(fbegin, bindex, n);

Если (bindex = -1) {

Печатать ("Такого города в данной системе не существует!");

}

Иначе {

Печатать ("Введите город в который нужно найти максимальный путь: ");

Читать с консоли строку fend

Печатать ("Конечный город:", fend);

eindex = FindIndexTown(fend, eindex, n);

Если (eindex == -1) {

Печатать("Такого города в данной системе не существует!");

}

иначе{

for (i от 0 до n)

{

Visited[i] = 0;

}

DFS(bindex, eindex, Distance, Visited, n);

Если (k == 0) {

Печатать ("Пути между этими городами не существует!");

}

иначе

{

Печатать("Максимальный путь из начального города в конечный:", bestpath);

}

}

}

}

1. Структура программы

Текст программы разбит на два модуля:

Модуль 1 содержит основные функции реализации решения задачи

Модуль 2 содержит основную программу

* 1. Состав модуля 1 – Header.h

Функция AddList

Прототип: void AddList(string begin, string end, int dis, int \*\*Distance)

Назначение: Добавление городов в массив городов, создание матрицы весов.

Функция ReadTowns

Прототип: void ReadTowns(int \*\*Distance))

Назначение: Ввод системы дорог

Функция FindIndexTown

Прототип: int FindIndexTown(string f, int index, int n)

Назначение: Поиск индекса города

Функция Incialization

Прототип: int \*\*Incialization(int \*\*Distance, int n)

Назначение: Инициализация матрицы весов

Функция DFS

Прототип: void DFS(int indexBegin, int indexEnd, int \*\*Distance, int \*Visited, int n)

Назначение: Обход графа в глубину и поиск максимально длинного пути из одной вершины в другую

5.2 Состав модуля 2 – “source.cpp”

Функция main

Прототип: void main()

Назначение : Определение входных и выходных данных задачи, поиск наибольшего пути, вывод результата.

* 1. Структура программы по управлению

Incialization

main

AddList

DFS

FindIndexTown

ReadTowns

1. Текст программы

Модуль 1

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <conio.h>

#include <string>

#include <algorithm>

#include <locale.h>

using namespace std;

FILE \*file;

string towns[40];

int i = 0;

int path = 0, bestpath = 0, k = 0;

struct Way {

char begin[15] = "0";

char end[15] = "0";

int distance = 0;

};

void AddList(string begin, string end, int dis, int \*\*Distance) {

int mark = 0;

int k = 0, m = 0;

for (int j = 0; j < i; j++)

{

if (begin == towns[j]) {

mark = 1;

m = j;

}

}

for (int j = 0; j < i; j++)

{

if (end == towns[j]) {

if (mark == 1) {

mark = 3;

k = j;

}

else {

mark = 2;

k = j;

}

}

}

if (mark == 1) {

towns[i] = end;

Distance[m][i] = dis;

i++;

}

if (mark == 2) {

towns[i] = begin;

Distance[i][k] = dis;

i++;

}

if (mark == 3) {

if (Distance[m][k] < dis) {

Distance[m][k] = dis;

}

}

if (mark == 0) {

towns[i] = begin;

i++;

towns[i] = end;

Distance[i - 1][i] = dis;

i++;

}

}

void ReadTowns(int \*\*Distance) {

char symbol;

Way \*Inf;

Inf = new Way;

fscanf(file, "%s %s", Inf->begin, Inf->end);

fscanf\_s(file, "%i", &Inf->distance);

AddList(Inf->begin, Inf->end, Inf->distance, Distance);

symbol = fgetc(file);

if (symbol == '\n') ReadTowns(Distance);

if (symbol == EOF) return;

}

int FindIndexTown(string f, int index, int n) {

for (int k = 0; k < n; k++)

{

if (f == towns[k]) {

index = k;

return k;

}

}

return -1;

}

void DFS(int indexBegin, int indexEnd, int \*\*Distance, int \*Visited, int n) {

if (indexBegin == indexEnd) {

k++;

bestpath = max(bestpath, path);

return;

}

Visited[indexBegin] = 1;

for (int j = 0; j < n; j++)

{

if (Distance[indexBegin][j]) {

if (Visited[j] == 0) {

path += Distance[indexBegin][j];

DFS(j, indexEnd, Distance, Visited, n);

path -= Distance[indexBegin][j];

}

}

}

Visited[indexBegin] = 0;

}

int \*\*Incialization(int \*\*Distance, int n) {

for (int j = 0; j < n; j++)

Distance[j] = (int\*)malloc(n \* sizeof(int));

for (int j = 0; j < n; j++)

{

for (int k = 0; k < n; k++)

{

Distance[j][k] = 0;

}

}

return Distance;

}

Модуль 2

#include "Header.h"

void main() {

fopen\_s(&file, "input.txt", "r");

setlocale(LC\_ALL, "Russian");

int n=0;

fscanf\_s(file, "%i", &n);

int \*Visited = (int\*)malloc(n \* sizeof(int));

int \*\*Distance = (int\*\*)malloc(n \* sizeof(int\*));

Distance = Incialization(Distance, n);

char fbegin[15], fend[15];

int bindex = -1, eindex = -1;

ReadTowns(Distance);

printf\_s("Система дорог представлена в виде матрицы весов, следующим образом:\n");

for (int j = 0; j < n; j++)

{

for (int k = 0; k < n; k++) {

printf\_s("%i ", Distance[j][k]);

}

printf\_s("\n");

}

printf\_s("Введите город из которого нужно найти максимальный путь: ");

scanf("%s", fbegin);

printf\_s("Начальный город: %s", fbegin);

bindex = FindIndexTown(fbegin, bindex, n);

if (bindex == -1) {

printf\_s("\nТакого города в данной системе не существует!");

}

else {

printf\_s("\nВведите город в который нужно найти максимальный путь: ");

scanf("%s", fend);

printf\_s("Конечный город: %s", fend);

eindex = FindIndexTown(fend, eindex, n);

if (eindex == -1) {

printf\_s("\nТакого города в данной системе не существует!");

}

else{

for (int i = 0; i < n; i++)

{

Visited[i] = 0;

}

DFS(bindex, eindex, Distance, Visited, n);

if (k == 0) {

printf\_s("\nПути между этими городами не существует!");

}

else

{

printf\_s("\nМаксимальный путь из начального города в конечный:%i", bestpath);

}

}

}

\_getch();

}

7.Тесты

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| №  теста | №  рисунка | Дано | Результат | Примечание |
| 1 | 1 | 2  Ny Vsh 40 | Начальный город: Ny  Конечный город: Vsh  Максимальный путь из начального города в конечный:40 | Два города, между ними одна дорога |
| 2 | 1 | 2  Ny Vsh 40 | Начальный город: Ny  Конечный город: Msk  Такого города в данной системе не существует ! | Два города, второго города в системе дорог нету |
| 3 | 2 | 6  Nsk Spb 30  Nsk Spb 50  Spb Nsk 60  Spb Vld 40  Nsk Vld 10  Msk Spb 60  Vld Spb 45  Vld Nsk 98  Ny Vsh 40 | Начальный город: Ny  Конечный город: Msk  Пути между этими городами не существует! | Ny – изолированная вершина. Пути между начальной и конечной вершинами не сущетсвует |
| 4 | 2 | 6  Nsk Spb 30  Nsk Spb 50  Spb Nsk 60  Spb Vld 40  Nsk Vld 10  Msk Spb 60  Vld Spb 45  Vld Nsk 98  Ny Vsh 40 | Начальный город: Msk  Конечный город: Nsk  Максимальный путь из начального города в конечный:198 | Нахождение максимального пути, при условии что при обходе в глубину , данный путь будет обноружен вторым |
| 5 | 3 | 22  Novosibirsk Vladivostok 400  Novosibirsk Kazan 320  Kazan Moscow 200  Moscow Novosibirsk 500  Vladivostok Sochi 700  Sochi Astana 400  Astana NewYork 1389  Sochi Krasnoyarsk 600  Krasnoyarsk Peterburg 890  Omsk Astana 297  Kazan London 856  LA London 1000  York London 780  York Omsk 900  NewYork Vancuver 400  Vancuver California 679  Petropavlovsk Novorosiysk 754  Novorosiysk California 1234  Novorosiysk Adler 432  Adler Vologda 232  Vologda Ekaterinburg 543  Ekaterinburg Almaty 745  Almaty Samara 321  Ekaterinburg Sochi 598  Kazan Omsk 987  Samara Peterburg 509  California Novorosiysk 1232  Peterburg Peterburg 1432 | Начальный город: Adler  Конечный город: Peterburg  Максимальный путь из начального города в конечный:2863 | Множество городов, наличие петель, которые игнорируются так как путь должен быть простым. |
| 6 | 3 | 22  Novosibirsk Vladivostok 400  Novosibirsk Kazan 320  Kazan Moscow 200  Moscow Novosibirsk 500  Vladivostok Sochi 700  Sochi Astana 400  Astana NewYork 1389  Sochi Krasnoyarsk 600  Krasnoyarsk Peterburg 890  Omsk Astana 297  Kazan London 856  LA London 1000  York London 780  York Omsk 900  NewYork Vancuver 400  Vancuver California 679  Petropavlovsk Novorosiysk 754  Novorosiysk California 1234  Novorosiysk Adler 432  Adler Vologda 232  Vologda Ekaterinburg 543  Ekaterinburg Almaty 745  Almaty Samara 321  Ekaterinburg Sochi 598  Kazan Omsk 987  Samara Peterburg 509  California Novorosiysk 1232  Peterburg Peterburg 1432 | Начальный город: Peterburg  Конечный город: Peterburg  Максимальный путь из начального города в конечный:0 | Из одного и того же города , то есть проверка петли. Результат показывает что петля программой игнорируется. |

8. Результаты работы программы

Программа работает правильно на всех тестовых наборах данных.

9. Литература

1. «Структуры данных и алгоритмы». В.П.Хиценко, Т.А.Шапошникова

2. «Практикум на ЭВМ», В.П.Хиценко, Т.А.Шапошникова

**Рисунок 1**

40

**Рисунок 2**

50

60

60

40

45

10

98

40

**Рисунок 3**