1. Текст индивидуального задания.

|  |  |
| --- | --- |
| 17 | Проверка наличия цикла отрицательной стоимости в ориентированном графе с нагруженными ребрами |

1. Математическая формулировка задачи в терминах теории множеств.

**Циклом** в ориентированном графе G = (V, E) {\displaystyle G=(V,E)} называется последовательность попарно смежных вершин, у которой начало и конец совпадают.

1. Выбор и обоснование способа представления данных.

Поскольку предложенный в методическом пособии алгоритм нахождения цикла подразумевает работу с матрицей смежности, целесообразно будет задание графа именно в таком виде.

В пользу этого говорит и «экономичность» расхода памяти массивом битов (пусть даже и двумерным). В отличие от предыдущего примера с деревом, количество ребер, инцидентных вершине, может варьироваться от 0 до n-1 (полный граф), в силу чего перебор их всех приобретает гораздо большую оправданность.

Наконец, в отличие от дерева, визуальное представление графа средствами псевдографики может оказаться довольно затруднительной задачей, в то время как вывод на экран матрицы смежности может быть хорошей альтернативой демонстрации графа в понятном пользователю виде (опять же, не нужно выделять дополнительную память под «экран»).

Исходя из этого, отпадает необходимость в создании отдельных объектов-узлов, т.к. вся необходимая информация уже содержится в классе, представляющих «цельный» граф. Последняя содержит:

* Максимально возможное число вершин – в данном случае их количество ограничивается числом возможных тегов, т.е. строчных латинских букв.
* Реальная мощность графа (нужна для ручного ввода, когда пользователь может не использовать всего объема памяти, выделенного под граф)
* Матрица смежности
* Массив тегов (на случай необходимости перестановки строк/столбцов)
* Вспомогательные массивы для алгоритма поиска максимальной клики.

1. Описание алгоритма и оценка его временной сложности.
2. Создание и заполнение массива

|  |  |
| --- | --- |
| А. Пользовательский ввод (O(n2)) | Б. Автоматическая генерация (O(n2)) |
| - вызов конструктора, принимающего в качестве аргумента нужное число вершин.  - очистка матрицы от «мусора»,  - заполнение массива тегов,  - ввод пользователем пар вершин, которые нужно соединить ребрами + контроль корректности ввода. Ввод прекращается, если пользователь вводит 0.  - заполнение вспомогательного массива. | - вызов конструктора,  - очистка матрицы от «мусора»,  - симметричное заполнение матрицы смежности единицами в произвольном порядке,  - заполнение массива тегов,  - заполнение вспомогательного массива. |

1. Обработка данных – O(n4)

Цикл по всем вершинам 0≤i≤size:

Добавляем текущую вершину в массив с результатом,

Цикл по всем вершинам, начиная от (i+1)-ой: (i+1≤st≤size):

Если вершины i и st смежны, то:

Присваиваю счетчику k значение 1;

Цикл по всем вершинам, начиная от st (st≤j≤size):

Присваиваю счетчику num значение 1;

Пока выполняются условия [вершины K[num-1] и [j] смежны] и [num ≤k], увеличиваю num на единицу,

Если [num-1=k], то увеличиваю k на единицу и записываю j в k-1 ячейку массива с результатами.

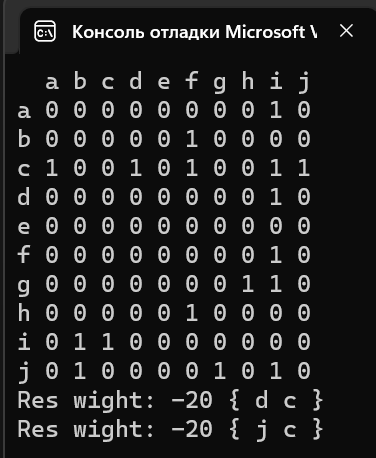
Если k больше последнего сохраненного числа вершин в клике, то обновляю максимум и сохраняю найденный результат (список вершин).

Если k равно последнему сохраненному максимуму, то вывожу результат.

III. Вывод данных (матрица) - O(n2)

1. Набор тестов и результаты проверки алгоритма на ЭВМ.

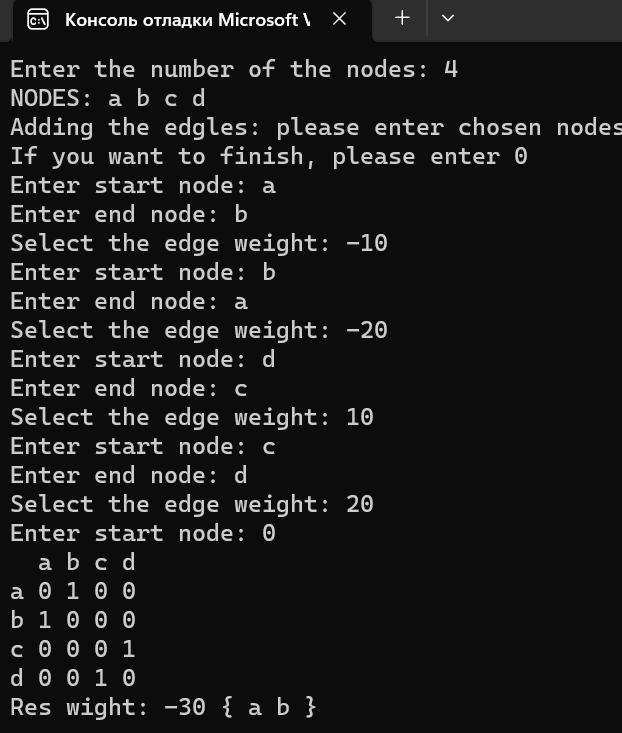
А. Произвольно сгенерированный граф из 10 вершин:



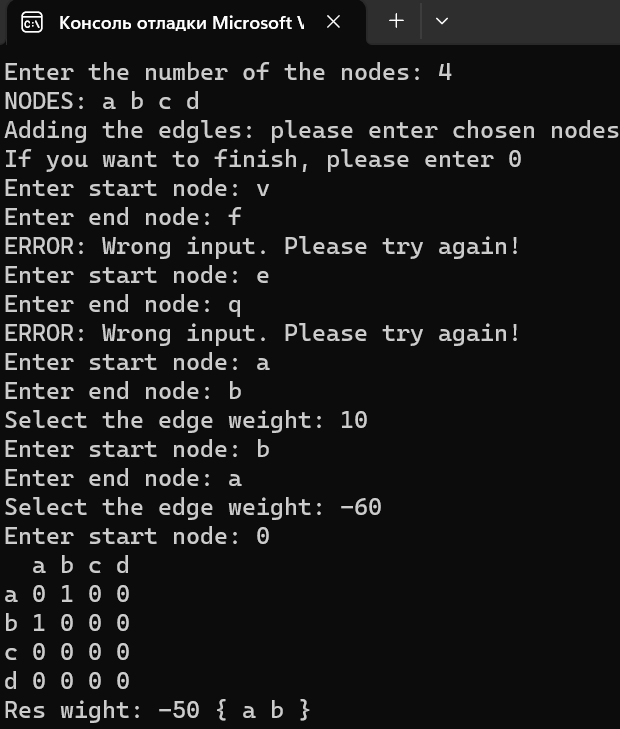
Б. Произвольно сгенерированный граф из 26 вершин:



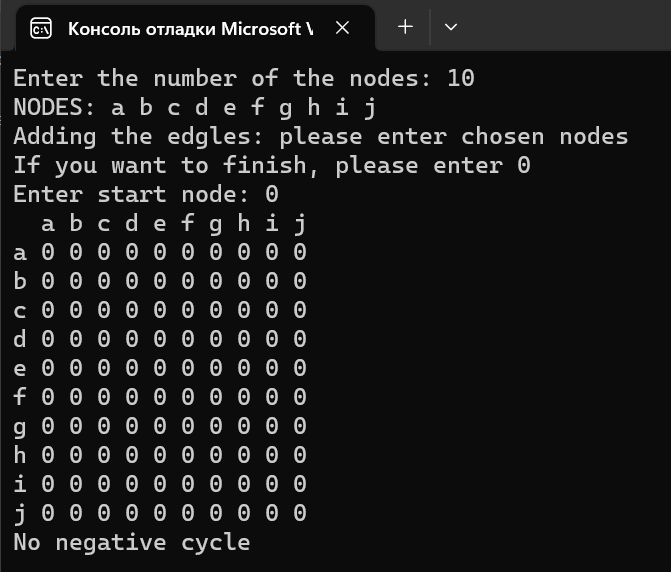
В. Ручной ввод – два цикла из четырех вершин, мощность графа – 4 вершин:



Г. Ручной ввод – некорректные данные



Д. Ручной ввод - пустой граф



1. Выводы.

Проведено испытание программы, реализующей в ориентированном графе с нагруженными ребрами поиска отрицательной стоимости цикла

Несмотря на явное сродство графа и его частного случая – дерева, на практике для их реализации оказалось удобно использовать совсем разные способы хранения информации.

В качестве «имен» вершин графа были выбраны строчные буквы латинского алфавита – в отличие от обычных чисел (которые не накладывают ограничений на количество вершин), такой выбор позволяет избежать путаницы с их расположением в матрице – подразумевалось, что могла возникнуть необходимость в перестановке/удалении некоторых из них.

Программа может как сгенерировать множество заданной мощности автоматически, так и работать с пользовательским вводом.

Для поиска цикла с отрицательной стоимостью использовался эмпирический алгоритм, поскольку диапазон значений возможной мощности множества позволяет получать с его помощью оптимальное решение со значительно меньшей временной сложностью.

1. Список использованных источников.

1) Поздняков С. Н., Рыбин С. В. Дискретная математика: учебник для вузов. М.: Академия, 2008. – 448 с.

2) Хагерти Р. Дискретная математика для программистов. Изд. 2‑е, испр. — М.: Техносфера, 2012. — 400 с.;

3) Новиков Ф. А. Дискретная математика для программистов. — СПб.: Питер, 2000. — 304 с., ил.

4) Страуструп Б. Язык программирования С++. 2‑е доп. изд. — М.: Бином-пресс, 2001. – 1098 с.;

1. Приложение: исходный текст программы для ЭВМ (на машинном носителе), файлы с тестами.

#include <iostream>

#include <ctime>

#include <random>

using namespace std;

static const int MAXSIZE = 26;

class Graph {

int size;

bool M[MAXSIZE][MAXSIZE];

int weight[MAXSIZE][MAXSIZE];

int testM[MAXSIZE][MAXSIZE];

char order[MAXSIZE];

int \*res\_wight;

char vertex\_name[MAXSIZE];

char \*\*name\_res\_wight;

int count\_wight = 0;

int count\_vertex\_name = 0;

int \*count\_arr\_vertex\_name;

int count\_name\_res\_wight = 0;

bool check\_two = true;

void print\_search();

void clearM();

void clear\_test\_M();

void fill();

void search(int x, int& l);

void creat\_dynamic\_arr();

public:

Graph(int i);

Graph();

~Graph();

void print();

void addEdgles();

void search\_cycle(int start);

};

void Graph::search\_cycle(int start = 0)

{

for (int i = 0; i < size; ++i) {

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

if (M[i][k]) {

int l = 0;

if (testM[i][k] == 0) {

testM[i][k] += 1;

search(k, l);

}

}

}

}

print\_search();

}

Graph::Graph() : size(MAXSIZE) {

clearM();

clear\_test\_M();

creat\_dynamic\_arr();

fill();

print();

}

void Graph::search(int i, int& l)

{

for (size\_t k = 0; k < size; ++k) {

if (M[i][k]) {

if (testM[i][k] == 0) {

testM[i][k] += 1;

search(k, l);

}

if (testM[i][k] == 1) {

testM[i][k] += 1;

l += weight[i][k];

if (count\_vertex\_name == 0) {

vertex\_name[count\_vertex\_name++] = order[i];

}

else {

if (vertex\_name[count\_vertex\_name - 1] != order[i])

vertex\_name[count\_vertex\_name++] = order[i];

}

check\_two = true;

search(k, l);

}

if (testM[i][k] == 2) {

if (check\_two && count\_vertex\_name >= 2) {

check\_two = false;

res\_wight[count\_wight++] = l;

l = 0;

for (int b = 0; b < count\_vertex\_name; ++b) {

name\_res\_wight[count\_name\_res\_wight][b] = vertex\_name[b];

}

count\_arr\_vertex\_name[count\_name\_res\_wight] = count\_vertex\_name;

++count\_name\_res\_wight;

count\_vertex\_name = 0;

}

}

}

}

}

void Graph::creat\_dynamic\_arr()

{

res\_wight = new int[100];

count\_arr\_vertex\_name = new int[100];

name\_res\_wight = new char \*[100];

for (int i = 0; i < 100; ++i) {

name\_res\_wight[i] = new char[100];

}

}

Graph::Graph(int i) : size(i) {

clearM();

clear\_test\_M();

creat\_dynamic\_arr();

cout << "NODES: ";

for (int i = 0; i < size; i++) {

order[i] = 'a' + i;

cout << order[i] << ' ';

}

cout << endl;

addEdgles();

print();

}

Graph::~Graph() {

delete[] res\_wight;

delete[] count\_arr\_vertex\_name;

for (int i = 0; i < 100; ++i) {

delete[] name\_res\_wight[i];

}

delete[] name\_res\_wight;

}

void Graph::addEdgles() {

cout << "Adding the edgles: please enter chosen nodes" << endl;

cout << "If you want to finish, please enter 0" << endl;

char second = '1', first = '1';

while (second != '0' || first != '0') {

cout << "Enter start node: ";

cin >> first;

if (first == '0') break;

cout << "Enter end node: ";

cin >> second;

if (first >= 'a' && first < 'a' + size && second >= 'a' && second < 'a' + size && first != second) {

int iFirst = 0, iSecond = 0;

for (int i = 0; i < size; i++) {

if (order[i] == first) iFirst = i;

if (order[i] == second) iSecond = i;

}

M[iFirst][iSecond] = true;

int choice\_weight = 0;

std::cout << "Select the edge weight: ";

std::cin >> choice\_weight;

weight[iFirst][iSecond] = choice\_weight;

}

else cout << "ERROR: Wrong input. Please try again!" << endl;

}

}

void Graph::print() {

cout << " ";

for (int i = 0; i < size; i++) cout << order[i] << ' ';

cout << endl;

for (int i = 0; i < size; i++) {

cout << order[i] << ' ';

for (int j = 0; j < size; j++) cout << M[i][j] << ' ';

cout << endl;

}

}

void Graph::print\_search()

{

bool flag\_if\_negative\_cycle = false;

for (int i = 0; i < count\_wight; ++i) {

if (res\_wight[i] < 0) {

flag\_if\_negative\_cycle = true;

std::cout << "Res wight: " << res\_wight[i] << " { ";

for (int z = 0; z < count\_arr\_vertex\_name[i]; ++z) {

std::cout << name\_res\_wight[i][z] << ' ';

}

std::cout << "}" << std::endl;

}

}

if (!flag\_if\_negative\_cycle) {

std::cout << "No negative cycle" << std::endl;

}

}

void Graph::clearM() {

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) M[i][j] = false;

}

}

void Graph::clear\_test\_M()

{

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) testM[i][j] = 0;

}

}

void Graph::fill() {

for (int i = 0; i < MAXSIZE; i++) {

order[i] = 'a' + i;

for (int j = 0; j < MAXSIZE; j++) {

if (rand() % 7 > 5 && i != j) {

M[i][j] = true;

int var\_weight = 0;

if(rand() % 7 > 3)

var\_weight = -10 + rand() % 1;

else

var\_weight = 10 + rand() % 1;

weight[i][j] = var\_weight;

}

}

}

}

int main()

{

srand(time(0));

//cin

int x;

cout << "Enter the number of the nodes: ";

cin >> x;

Graph g(x);

//rand

//Graph g;

g.search\_cycle();

}