ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ассистент |  |  |  | М. А. Мурашова |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ |
| «АЛГОРИТМЫ НА ГРАФАХ» |
| по курсу: СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

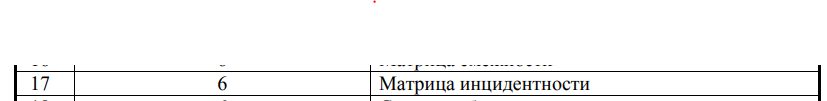
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4134к |  |  |  | Костяков Н.А. |
|  |  |  | подпись, дата |  | инициалы, фамилия |

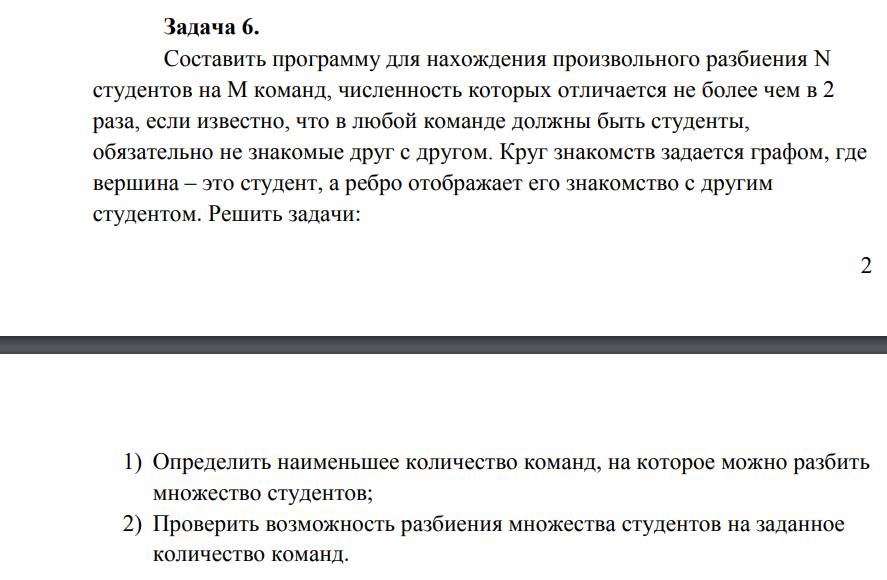
Санкт-Петербург 2022

**1.1 Цель работы** Целью работы является изучение графов и получение практических навыков их использования.

**1.2 Задание на лабораторную работу** Разработать на языке программирования высокого уровня программу, которая должна выполнять функцию, в соответствии с вариантом задания. Варианты задания приведены в таблице 7 (формулировки задач приведены после таблицы).

**Вариант 17**





**Листинг**

#include <iostream>

#include <vector>

class matrix

{

public:

matrix();

~matrix();

void add\_student();

void add\_link(int f, int s);

void show();

std::vector<int> find\_link(int student);

std::vector<int> no\_link(int student);

void make\_groups(int num);

private:

std::vector <std::vector <int>> groups ;

int\*\* table =0;

int students = 0;

int links = 0;

};

matrix::matrix()

{

}

matrix::~matrix()

{

}

void matrix::add\_student()

{

students++;

int \*\*temp\_table = new int\*[students];

for (int i = 0; i < students-1; i++)

{

temp\_table[i] = table[i];

}

for (int q = 0; q < links; q++)

{

temp\_table[students - 1][q] = 0;

}

table = temp\_table;

}

void matrix::add\_link(int f, int s)

{

int\*\* temp\_table = new int\*[students];

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

{

temp\_table[i] = new int[links+1];

for (int q = 0; q < links; q++) {

temp\_table[i][q] = table[i][q];

}

temp\_table[i][links] = 0;

}

links++;

temp\_table[f][links-1] = 1;

temp\_table[s][links - 1] = 1;

table = temp\_table;

}

void matrix::show()

{

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

{

for (int q = 0; q < links; q++)

{

std::cout << table[i][q] << "\t";

}

std::cout << "\n";

}

}

std::vector<int> matrix::find\_link(int student)

{

std::vector <int> match;

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

{

if (table[student][i] == 1)

{

for (int q = 0; q < students; q++)

{

if (table[q][i]==1 and q!=student)

{

match.push\_back(q);

}

}

}

}

return match;

}

std::vector<int> matrix::no\_link(int student)

{

std::vector <int> match;

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

{

if (table[student][i] == 0)

{

for (int q = 0; q < students; q++)

{

if (table[q][i]==0 and q!=student)

{

match.push\_back(q);

}

}

}

}

return match;

}

bool already\_in(std::vector <int> el ,std::vector <int> arr) {

if (arr.empty())

{

return 0;

}

for (auto i: arr)

{

for (auto e: el)

{

if (i == e)

{

return 1;

}

}

}

return 0;

}

bool balanced(std::vector<int>arr)

{

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

for (int q = i; q < arr.size(); q++)

{

if (arr[i]>arr[q]\*2)

{

return 0;

}

}

}

return 1;

}

int can\_be\_moved\_to(std::vector <int> links, std::vector <std::vector <int>> groups, int from)

{

int group\_num = -1;

int move\_to = -1;

for (auto group : groups)

{

bool flag = 1;

group\_num++;

for (auto stud : group)

{

for (auto link : links)

{

if (link == stud)

{

flag = 0;

break;

}

}

}

if (flag and group\_num!=from)

{

move\_to = group\_num;

return move\_to;

}

}

return move\_to;

}

int vec\_find(std::vector <int> arr, int el)

{

for (int i = 0; i < arr.size(); i++)

{

if (arr[i]==el)

{

return i;

}

}

return -1;

}

void matrix::make\_groups(int num)

{

std::vector <int>group\_size;

group\_size.push\_back(0);

groups.push\_back(std::vector<int>());

int groups\_count;

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

{

std::vector <int> linked = find\_link(i);

int working\_group = 0;

groups\_count = groups.size();

while (true)

{

if (!already\_in(linked, groups[working\_group]))

{

groups[working\_group].push\_back(i);

group\_size[working\_group] = groups[working\_group].size();

std::cout << "Added student num " << i << " ,Size of group " << working\_group << " is " << group\_size[working\_group] << '\n';

break;

}

else

{

working\_group++;

if (groups.size()<working\_group+1)

{

groups.push\_back(std::vector<int>());

group\_size.push\_back(0);

}

}

}

}

while (1)

{

if (balanced(group\_size))

{

break;

}

for (int i = 0; i < group\_size.size(); i++)

{

bool flag = 0;

for (auto q : groups[i])

{

std::vector <int> linked = find\_link(q);

int move\_to = can\_be\_moved\_to(linked, groups, i);

if (move\_to!=-1)

{

if (vec\_find(groups[i], q)!=-1)

{

auto iter = groups[i].begin();

groups[i].erase(iter+vec\_find(groups[i], q));

groups[move\_to].push\_back(q);

group\_size[move\_to] ++;

group\_size[i] --;

}

}

}

if (balanced(group\_size))

{

break;

}

}

}

if (num<=students and num>=groups.size())

{

std::cout << "\n\nAvailable to make " << num << " Groups\n";

}

else

{

std::cout << "Imposible to make " << num << " Groups\n";

}

std::cout << "\n\nMinimal group count is "<<groups.size()<<"\n";

for (auto group : groups)

{

std::cout << "Group: ";

for (auto student : group) {

std::cout << student << "\t";

}

std::cout << "\n";

}

}

int main()

{

matrix mat;

std::cout << "Do you want to set your own Students and links? (y/n)\n";

char v;

std::cin >> v;

switch (v)

{

case 'y':

std::cout << "How many students you want: ";

int stud;

std::cin >> stud;

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

{

mat.add\_student();

}

while (1)

{

std::cout << "Set links in format 0 2 (student student), -1 to stop\n";

int f, s;

std::cin >> f;

std::cin >> s;

if (f != -1)

{

mat.add\_link(f, s);

}

else

{

break;

}

}

break;

default:

mat.add\_student();

mat.add\_student();

mat.add\_student();

mat.add\_student();

mat.add\_student();

mat.add\_student();

mat.add\_student();

mat.add\_student();

mat.add\_student();

mat.add\_student();

mat.add\_student();

mat.add\_link(2, 1);

mat.add\_link(2, 3);

mat.add\_link(1, 5);

mat.add\_link(6, 5);

mat.add\_link(4, 1);

mat.add\_link(6, 3);

mat.add\_link(5, 4);

mat.add\_link(6, 3);

mat.add\_link(4, 6);

mat.add\_link(0, 6);

break;

}

std::cout <<"Enter a number to check the ability to make such groups: ";

int num;

std::cin >> num;

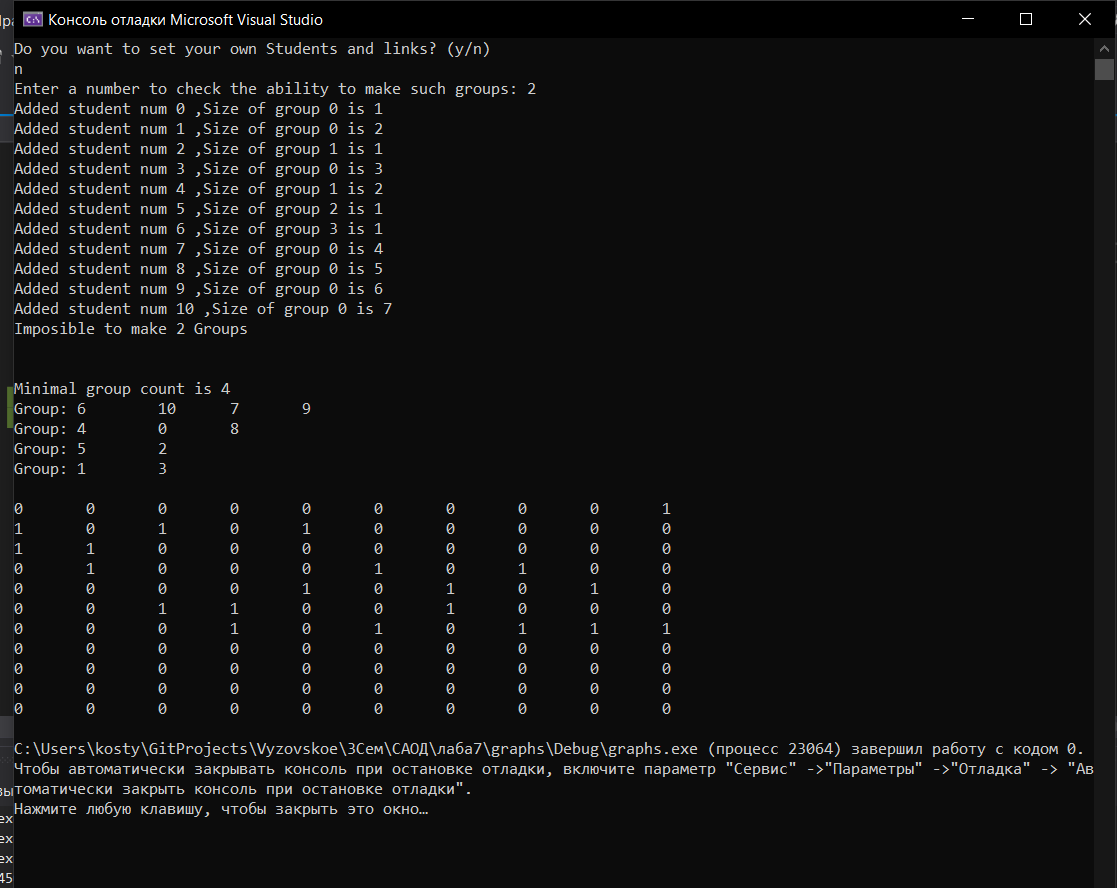
mat.make\_groups(num);

std::cout << "\n";

mat.show();

}

**Результат работы**



**Вывод**

Я освоил методы работы с графами и реализовал свою программу для разбиения студентов на группы