ПОШУК НАЙКОРОТШИХ ШЛЯХІВ НА ГРАФАХ

Алгоритми

Лабораторна робота №7

Варіант 7

Виконав: Конча Вадим

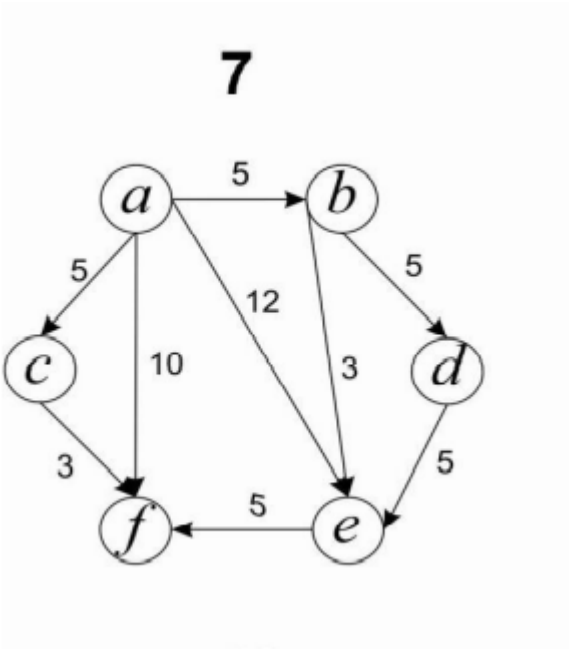
Завдання:

Вимоги до виконання роботи

Непарні номери варіантів (алгоритм Дейкстри).

Складіть програму для реалізації алгоритму Дейкстри пошуку найкоротших шляхів до всіх вершин графа від вказаної вершини a. Спосіб представлення графа у пам'яті комп'ютера оберіть за бажанням. У програмі забезпечте:

- можливість вибору початкової вершини a (ввід з клавіатури);
- вивід на екран найкоротших шляхів у графі та їх довжин (ваг).



Код:

```
#include <iostream>
#include <vector>
#include <string.h>
#include <functional>

using namespace std;

int inf = 9999;

class point { //описывает точку
	public:
		string name;
	bool status = true; //выкресленность
	vector <point> sosedi; //к кому есть пути
	vector <int> dist_to_sosedi; //расстояния к этим точкам
	int distance_to_it = inf;
```

```
	//соседи это те, к кому ты можешь пойти в гости. Они к тебе нет.
	//Таково определение ориентерованных графов
	void printSosedi()
	{
		cout << "\n There are sosedi\n";
		for (int i = 0; i < sosedi.size(); i++)
		{
			cout << "\n" << sosedi[i].name << endl;
			cout << "\n" << dist_to_sosedi[i] << endl;
		}
	}
};

point a, b, c, d, e, f;
void printRESULTS(vector <point>& points)
{
	for (int i = 0; i < points.size(); i++)
	{
		cout << points[i].name << " is = " << points[i].distance_to_it << endl;
	}
}
bool main_status(vector <point>& points)
{
	bool mainstatus = false;
	for (int i = 0; i < points.size(); i++)
	{
		if (points[i].status == true)
		{
			mainstatus = true;
		}
	}
	return mainstatus;
}
int iden(vector <point> &points, point &a)
	{//поиск идентефикатора элемента "a" в массиве
		int iden;
		bool meow = false;
		for (int k = 0; k < points.size(); k++)
		{
			if (a.name == points[k].name)
			{
				iden = k;
				meow = true;
				break;
			}
		}
		if (meow == false) iden = 999;
		return iden;
	}
int idenSTRING(vector <point>& points, string& a)
{//поиск идентефикатора элемента "a" в массиве
	int iden;
```

```
	bool meow = false;
	for (int k = 0; k < points.size(); k++)
	{
		if (a == points[k].name)
		{
			iden = k;
			meow = true;
			break;
		}
	}
	if (meow == false) iden = 999;
	return iden;
}
int choice(vector <point> &points)
{
	int ide;
	string startpoint = "a";
	cout << "Enter the startpoint: ";
	cin >> startpoint;
	if (idenSTRING(points, startpoint) == 999)
	{
		cout << "\nThis is not the avaliable point\n";
		exit(0);
	}
	for (int i = 0; i < points.size(); i++)
	{
		if (points[i].name == startpoint)
		{
			ide = i;
		}
	}
	points[ide].distance_to_it = 0;
	return ide;
}
void updateDistans(vector <int> distans, vector <point> points)
{
	for (int i = 0; i < distans.size(); i++)
		distans[i] = points[i].distance_to_it;
}
int min(vector <point> &points) {

	int min; //надо найти элемент который ближе всего к нам И НЕ ОБРАБОТАН
	for (int i = 0; i < points.size(); i++)
	{
		if (points[i].status == true)
		{
			min = i;
		}
	}
	for (int i = 0; i < points.size(); i++)
	{
		if (points[i].status == true)
```

```
                        if (points[i].distance_to_it < points[min].distance_to_it)
                        {
                                min = i;
                        }
        }
        return min;
}
bool isPointInMassive(vector <point> points, point a)
{//поиск идентефикатора элемента "a" в массиве
        int iden;
        bool meow = false;
        for (int k = 0; k < points.size(); k++)
        {
                if (a.name == points[k].name)
                {
                        iden = k;
                        meow = true;
                        break;
                }
        }
        return meow;
}

//не даёт пользоваться точкой в дальнейшем
void delete_point(vector <point>& points, point &a)
{

                //удаление всех данных об этом элементе из всег остальных точек(его забывают
и не обращают на него внимание в дальнейшем)
                for (int i = 0; i < points.size(); i++)
                {

                        int iden_a = iden(points[i].sosedi, a);
                        if (iden_a != 999)
                        {
                                points[i].sosedi.erase(points[i].sosedi.begin() + iden_a);
                                points[i].dist_to_sosedi.erase(points[i].dist_to_sosedi.begin() +
iden_a);
                        }
                }

}
void autocalc(vector <point>& points ,int &way1, int &way2, int ide)
{
        for (int k = 0; k < points.size(); k++)
        {
                if (k == ide) continue;
                if (isPointInMassive(points[ide].sosedi, points[k])) {
                        way1 = 0, way2 = 0;
                        way1 = points[ide].distance_to_it +
points[ide].dist_to_sosedi[iden(points[ide].sosedi, points[k])];
                        //найдём индекс точки "g" которая может лежать между a и б
```

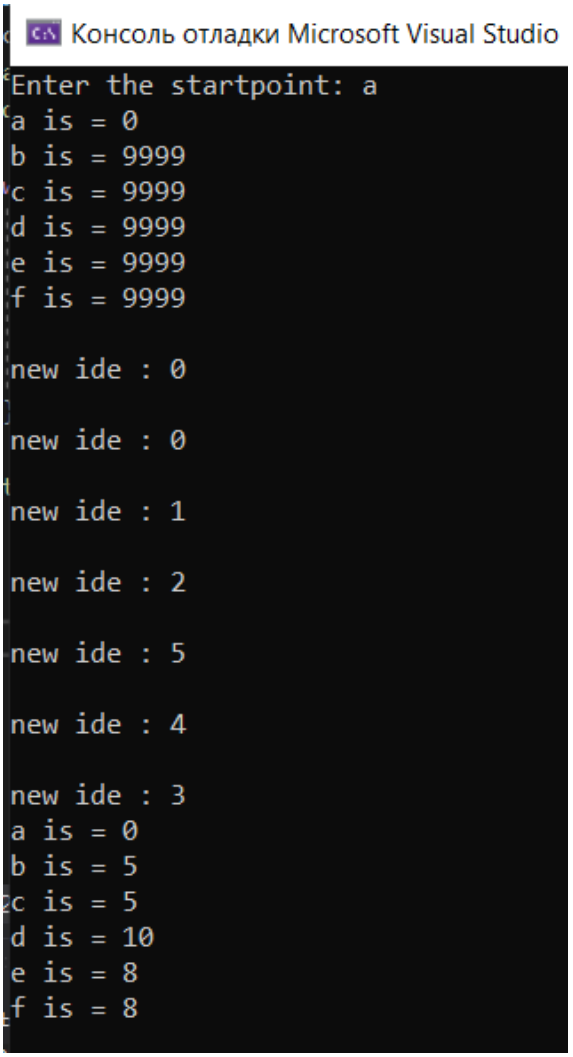
```
vector <string> PointsWithB;
PointsWithB.clear();
for (int i = 0; i < points[ide].sosedi.size(); i++) //ищем среди всех соседей
{
	vector <point> sosedi_x2;

	for (int ii = 0; ii < points[ide].sosedi.size(); ii++)
		for (int kk = 0; kk < points.size(); kk++)
		{
			if (points[ide].sosedi[ii].name == points[kk].name)
				sosedi_x2.push_back(points[kk]);
		}
	{

		if (isPointInMassive(sosedi_x2[i].sosedi, points[k])) //ищем
тех, которые соприкасаются с b
		{
			PointsWithB.push_back(points[ide].sosedi[i].name);
//и добавляем их в массив
	/*		if (k == 5)
			{
				cout << "\n 2 is name of sosed =" <<
sosedi_x2[i].name << "\n";
				sosedi_x2[i].printSosedi();
				for (int ket = 0; ket < PointsWithB.size();
ket++)
					cout << "\n " <<
points[idenSTRING(points, PointsWithB[ket])].name << "\n";
			}*/
		}
	}
}
if (!PointsWithB.empty())
{
	int min = 0; //индекс элемента с наименшим расстоянием к точке
b в массиве PointsWithB
	for (int q = 0; q < PointsWithB.size(); q++) //среди всех элементов
PointsWithB
	{
		if (points[idenSTRING(points,
PointsWithB[q])].dist_to_sosedi[iden(points[idenSTRING(points, PointsWithB[q])].sosedi, points[k])] <
points[idenSTRING(points, PointsWithB[q])].dist_to_sosedi[iden(points[idenSTRING(points,
PointsWithB[q])].sosedi, points[k])]) //если найдётся тот чё меньше(отличие только в индексе)
		{
			min = q; //то это новый минимум-index
		}
	}
	int g = idenSTRING(points, PointsWithB[min]); //ищем индекс этого
элемента в изначальных поинтах и это наш g
```

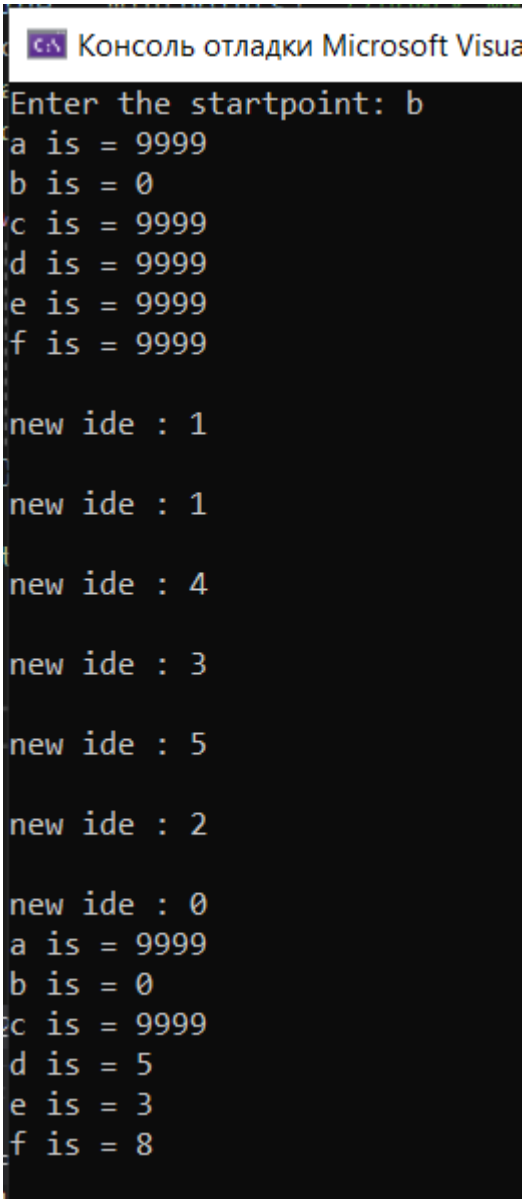
```
                                        way2 = points[ide].distance_to_it +
points[ide].dist_to_sosedi[iden(points[ide].sosedi, points[g])] +
points[g].dist_to_sosedi[iden(points[g].sosedi, points[k])];

                                        //сравним результаты и будем искать меньшее(короткий путь)
                                        if (way1 > way2) points[k].distance_to_it = way2;
                                        else points[k].distance_to_it = way1;
                                }
                                else points[k].distance_to_it = way1;
                                //проверка
                                /*if (k == 5)
                                {
                                        cout << "\nit is 5\n";
                                        if (PointsWithB.empty()) cout << "\nEMPTY\n";
                                        for (int i = 0; i < PointsWithB.size(); i++)
                                        {
                                                cout << endl << " dasd";
                                        }
                                }*/
                        }
                }
}
void pipec_koroche_fake_variantam_exe(int ide, vector <point>& points)
{//удаляет все элементы к которым нет доступа
        vector <string> ne_fake; //тут все соседи и соседи соседей
        int sizeofarr1, sizeofarr2, sizeofarr3; //чтобы код не прыгал в рекурсию
        sizeofarr1 = points[ide].sosedi.size();
        for (int ii = 0; ii < sizeofarr1; ii++) //поиск среди всех соседей элемента
        {
                                        ne_fake.push_back(points[ide].sosedi[ii].name);
                                        sizeofarr2 = ne_fake.size();
                                        for (int k = 0; k < sizeofarr2; k++) //для всех элементов "не фейка"
                                        {
                                                ne_fake.push_back(points[idenSTRING(points,
ne_fake[k])].name); //закинуть всех соседей этих элементов
                                                sizeofarr3 = ne_fake.size();
                                                for (int kk = 0; kk < sizeofarr3; kk++) //для всех элементов
"не фейка"
                                                {
                                                        ne_fake.push_back(points[idenSTRING(points,
ne_fake[k])].name); //закинуть всех соседей этих элементов
                                                }
                                        }
        }
        for (int l = 0; l < points.size(); l++) //меняет статус всех элементов на фолз
        {
                points[l].status = false;
        }
        for (int l = 0; l < ne_fake.size(); l++)
        {
                if (idenSTRING(points, ne_fake[l]) != 999)
                {
```

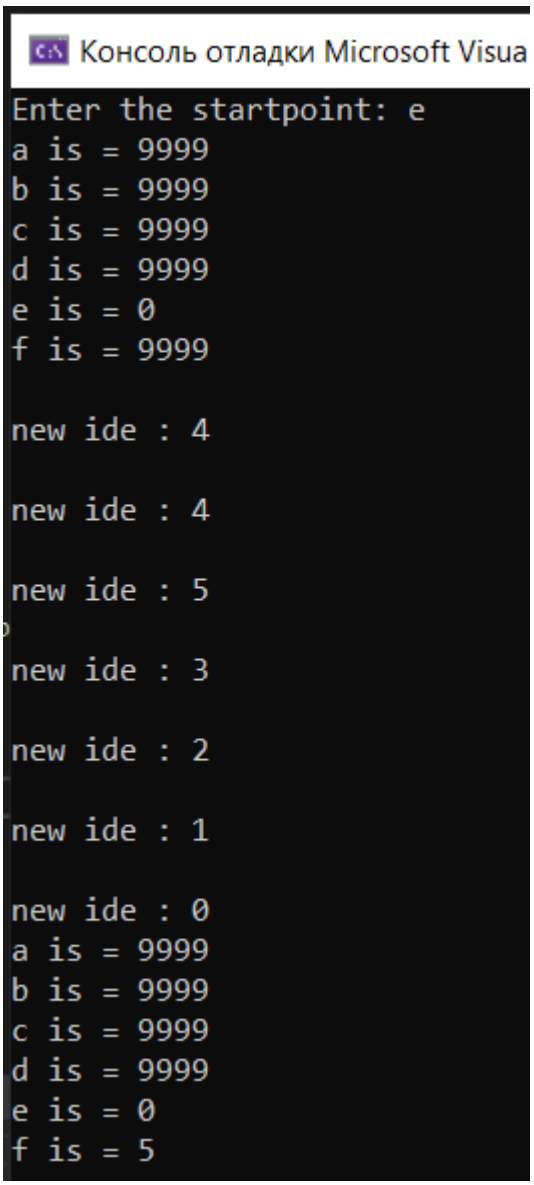
```
                        points[idenSTRING(points, ne_fake[l])].status = true;
                }
        }
        for (int l = 0; l < points.size(); l++) //для всех элементов поинтс
        {
                if (points[l].status == false) //если он фолз
                {
                        delete_point(points, points[l]);
                        points.erase(points.begin() + l);
                }
        }
}
int main()
{
        a.name = "a"; b.name = "b"; c.name = "c"; d.name = "d"; e.name = "e"; f.name = "f";
        a.sosedi = { b,c,f,e };
        a.dist_to_sosedi = { 5, 5, 10, 12 };
        b.sosedi = { e,d };
        b.dist_to_sosedi = { 3,5 };
        c.sosedi = { f };
        c.dist_to_sosedi = { 3 };
        d.sosedi = { e };
        d.dist_to_sosedi = { 5 };
        //y f соседей нету(
        e.sosedi = { f };
        e.dist_to_sosedi = { 5 };

        vector <point> points{ a, b, c, d, e, f }; //массив из всех точек

        /*
        массив самых коротких путей к точкам(привязано по сути индексами к массиву точек)
        здесь за каждым значением прикреплена своя буква по порядку.
        то какое число тут случится по итогу и будет определять кратчайшее расстояние к этой
точке
        */

        int ide = choice(points); // выбор начальной точки
        //pipec_koroche_fake_variantam_exe(ide, points);
        ide = min(points); //поиск минимальной точки

        printRESULTS(points);

        int way1, way2;
        /*
        //посчитаем путь от а до ф  И от а до с и до ф
        way1 = points[ide].distance_to_it + points[ide].dist_to_sosedi[iden(points[ide].sosedi,
points[5])];
                //путь к а + путь к ф от а
        way2 = points[ide].distance_to_it + points[ide].dist_to_sosedi[iden(points[ide].sosedi,
points[2])] + points[2].dist_to_sosedi[iden(points[2].sosedi, points[5])];

        //сравним результаты и будем искать меньшее(короткий путь)
```

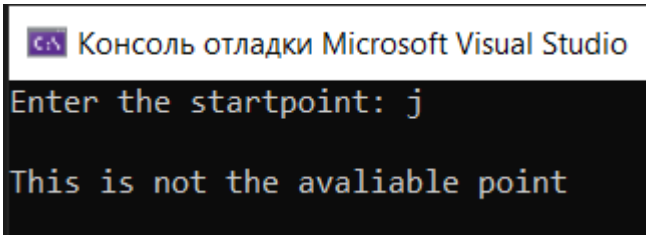
```
	if (way1 > way2) points[5].distance_to_it = way2;
	else points[5].distance_to_it = way1;
	//cout << "iden eblan? " << iden(points[0].sosedi, points[3]) << endl;
	printRESULTS(points);

	*/



////////////////////////////////////AUTO
		//мы на входе уже имеем начальный член и его индекс ide

	//

		ide = min(points); //поиск минимальной точки
		cout << "\nnew ide : " << ide <<"\n";
		autocalc(points, way1, way2, ide);
		delete_point(points, points[ide]);

		while (main_status(points) == true) {
			ide = min(points); //поиск минимальной точки
			cout << "\nnew ide : " << ide << "\n";
			autocalc(points, way1, way2, ide);
			points[ide].status = false;
			delete_point(points, points[ide]);
		}

	printRESULTS(points);
}
```

РЕЗУЛЬТАТИ:


Консоль отладки Microsoft Visual Studio
Enter the startpoint: a
a is = 0
b is = 9999
c is = 9999
d is = 9999
e is = 9999
f is = 9999
new ide : 0
new ide : 0
new ide : 1
new ide : 2
new ide : 5
new ide : 4
new ide : 3
a is = 0
b is = 5
c is = 5
d is = 10
e is = 8
f is = 8


Консоль отладки Microsoft Visua
Enter the startpoint: b
a is = 9999
b is = 0
c is = 9999
d is = 9999
e is = 9999
f is = 9999
new ide : 1
new ide : 1
new ide : 4
new ide : 3
new ide : 5
new ide : 2
new ide : 0
a is = 9999
b is = 0
c is = 9999
d is = 5
e is = 3
f is = 8





Джерела:

[Алгоритм Дейкстри — Вікіпедія (wikipedia.org)]
[Граф (prog-cpp.ru)]