**Лабораторная работа № 7**

**Тема**: Алгоритмы работы с графами. Алгоритм Прима-Крускала и Дейкстры

**Цель**: Научиться реализовывать алгоритм Прима-Крускала для построения каркасного дерева графу и алгоритм Дейкстры для поиска кратчайшего пути

Задания к лабораторной работе:

Реализовать программно абстрактный тип данных неориентированный граф. Создать его экземпляр, который заполнить данными, указанными в Вашем варианте. Разработать программу, которая должна:

1. Построить для данного графу минимальное каркасное дерево, используя алгоритм Прима-Крускала, определить суммарный вес построенного дерева, вывести полученные данные на экран.

2. Определить и вывести на экран кратчайший путь от вершины 1 к вершине 6 с помощью алгоритма Дейкстры.

вариант 1

1

2

5

3

4

6

5

1

3

1

2

5

4

10

7

1

2

5

3

4

6

5

1

1

2

4

1. Построить для данного графу минимальное каркасное дерево, используя алгоритм Прима-Краскала, определить суммарный вес построенного дерева, вывести полученные данные на экран.*/\**

/*\** *Алгоритм* Прима - *Краскала*

*\*\* Программирование = Лабараторная работа №7*

*\*\* Аннаев Арслан КИ-156*

*\** *\*/*

#include <iostream>

#include <cstdio>

#include <vector>

#include <algorithm>

*using* *namespace* std;

*struct* edge *//* *структура* *хранимая* *начальное* *и* *конечное* *положение* *и* *вес* *ребра*

{

int x, y, w;

edge(){}

edge(int x, int y, int w): x(x), y(y), w(w) {}

};

bool cmp(*const* edge& a, *const* edge& b) *//* *если* *вес* *ребра* *больше*

{

*return* a.w < b.w;

}

vector <int> leader; *//* *посещеный* *вектор*

int getLeader(int x) *//* *проверка* *посещали* *ли* *мы* *эту* *вершину*

{

*if* (x == leader[x])

*return* x;

*return* leader[x] = getLeader(leader[x]);

}

bool unite(int x, int y) *//* *обьединение* *двух* *вершин*

{

x = getLeader(x);

y = getLeader(y);

*if* (x == y)

*return* *false*;

*if* (rand() % 2 == 0)

swap(x, y);

leader[x] = y;

*return* *true*;

}

int main()

{

int n, m;

cout << "Enter Vertex count \_"; *//* *приглашение* *ввода* *кол.* *вершин*

cin >> n; *//* *ввод* *значения*

cout << "\nEnter count of -- \_";*//* *приглашение* *ввода* *кол.* *ребер*

cin >> m; *//* *ввод* *значения*

*//* *массив* *структур* *хранящий* *начальную* *вершину* *и* *конечную* *с* *весом* *ребра*

vector <edge> e(m);

*//* *Определение* *графа*

*for* (int i = 0; i < m; i++)

{

cout << " Enter vertex \"x\" (" << i+1 << ") = "; *//* *приглашение*

cin>> e[i].x ; *//* *ввод* *значения*

cout << " Enter vertex \"y\" (" << i+1 << ") = "; *//* *приглашение*

cin >> e[i].y ; *//* *ввод* *значения*

cout << " Enter weigth \"w\" (" << i+1 << ") = "; ; *//* *приглашение*

cin>> e[i].w; *//* *ввод* *значения*

e[i].x--; *//* *декремент* *x*

e[i].y--; *//* *декремент* *y*

}

sort(e.begin(), e.end(), cmp); *//* *сортировка* *по* *возрастанию*

leader.resize(n); *//* *изменение* *размера*

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

leader[i] = i;

vector <edge> ans;

*for* (int i = 0; i < m; i++)

{

int x = e[i].x, y = e[i].y;

*if* (unite(x, y))

ans.push\_back(e[i]);

}

int sum = 0;

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

sum+=ans[i].w;

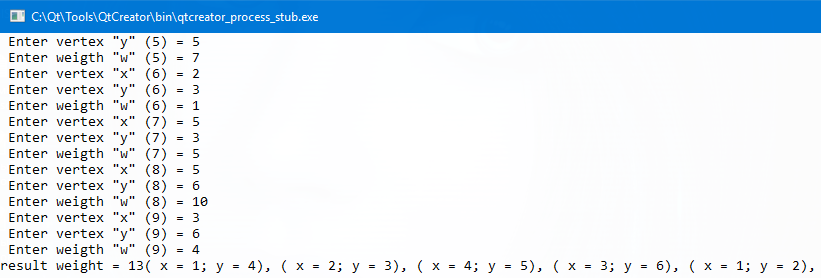
cout << "result weight = "<< sum;

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

cout << "( x = " << ans[i].x + 1 << "; y = " << ans[i].y + 1 << "), ";

*return* 0;

}



2. Определить и вывести на экран кратчайший путь от вершины 1 к вершине 6 с помощью алгоритма Дейкстры.

Шаг 1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 0 | 5 | \* | 1 | 3 | \* |

Шаг 2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 0 | 5 | 8 | 1 | 3 | 13 |

Шаг 3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 0 | 5 | 6 | 1 | 3 | 13 |

Шаг 4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 0 | 5 | 6 | 1 | 3 | 10 |

#include <iostream>

#include <vector>

#include <string>

#include <algorithm>

*using* *namespace* std;

*static* int INF = INT\_MAX / 2;

int n; *//количество* *вершин* *в* *орграфе*

int m; *//количество* *дуг* *в* *орграфе*

vector<int> \*adj; *//список* *смежности*

vector<int> \*weight; *//вес* *ребра* *в* *орграфе*

vector<bool> used; *//массив* *для* *хранения* *информации* *о* *пройденных* *и* *не* *пройденных* *вершинах*

int \*dist; *//массив* *для* *хранения* *расстояния* *от* *стартовой* *вершины*

int \*pred; *//массив* *предков,* *необходимых* *для* *восстановления* *кратчайшего* *пути* *из* *стартовой* *вершины*

int start; *//стартовая* *вершина,* *от* *которой* *ищется* *расстояние* *до* *всех* *других*

*//процедура* *запуска* *алгоритма* *Дейкстры* *из* *стартовой* *вершины*

void dejkstra(int s) {

dist[s] = 0; *//кратчайшее* *расстояние* *до* *стартовой* *вершины* *равно* *0*

*for* (int iter = 0; iter < n; ++iter) {

int v = -1;

int distV = INF;

*//выбираем* *вершину,* *кратчайшее* *расстояние* *до* *которого* *еще* *не* *найдено*

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

*if* (used[i]) {

*continue*;

}

*if* (distV < dist[i]) {

*continue*;

}

v = i;

distV = dist[i];

}

*//рассматриваем* *все* *дуги,* *исходящие* *из* *найденной* *вершины*

*for* (int i = 0; i < adj[v].size(); ++i) {

int u = adj[v][i];

int weightU = weight[v][i];

*//релаксация* *вершины*

*if* (dist[v] + weightU < dist[u]) {

dist[u] = dist[v] + weightU;

pred[u] = v;

}

}

*//помечаем* *вершину* *v* *просмотренной,* *до* *нее* *найдено* *кратчайшее* *расстояние*

used[v] = *true*;

}

}

*//процедура* *считывания* *входных* *данных* *с* *консоли*

void readData(){

cout << "Enter Vertex count \_"; *//* *приглашение* *ввода* *кол.* *вершин*

cin >> n; *//* *ввод* *значения*

cout << "\nEnter count of edge \_";*//* *приглашение* *ввода* *кол.* *ребер*

cin >> m; *//* *ввод* *значения*

cin >> start; *//считываем* *количество* *вершин,* *количество* *дуг* *графа* *и* *стартовую* *вершину*

start--;

*//инициализируем* *списка* *смежности* *графа* *размерности* *n*

adj = *new* vector <int>[n];

*//инициализация* *списка,* *в* *котором* *хранятся* *веса* *ребер*

weight = *new* vector <int> [n];

*//считываем* *граф,* *заданный* *списком* *ребер*

*for* (int i = 0; i < m; ++i) {

int u, v, w;

cout << " Enter vertex \"x\" (" << i+1 << ") = "; *//* *приглашение*

cin >> u ; *//* *ввод* *значения*

cout << " Enter vertex \"y\" (" << i+1 << ") = "; *//* *приглашение*

cin >> v ; *//* *ввод* *значения*

cout << " Enter weigth \"w\" (" << i+1 << ") = "; ; *//* *приглашение*

cin >> w; *//* *ввод* *значения*

u--;

v--;

adj[u].push\_back(v);

weight[u].push\_back(w);

}

used.resize(n, *false*);

pred = *new* int[n]; *//массив* *предков,* *необходимых* *для* *восстановления* *кратчайшего* *пути* *из* *стартовой* *вершины*

dist = *new* int[n]; *//массив* *для* *хранения* *расстояния* *от* *стартовой* *вершины*

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

pred[i] = -1;

dist[i] = INF;

}

}

*//процедура* *восстановления* *кратчайшего* *пути* *по* *массиву* *предком*

void printWay(int v) {

*if* (v == -1) {

*return*;

}

printWay(pred[v]);

cout << "-" << v + 1;

}

*//процедура* *вывода* *данных* *в* *консоль*

void printData() {

*for* (int v = 0; v < n; ++v) {

*if* (dist[v] != INF) {

cout << dist[v] << " - "; *//*

} *else* {

cout << "-1 ";

}

}

cout << "\n";

*for* (int v = 0; v < n; ++v) {

cout << v + 1;

*if* (dist[v] != INF) {

printWay(v);

}

cout << "\n";

}

}

void run() {

readData();

dejkstra(start);

printData();

}

int main()

{

run();

*return* 0;

}

