Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №5**

По дисциплине «Математические основы интеллектуальных систем»

Тема: «Нахождение минимального остового дерева связанного неориентированного графа»

**Выполнил:**

Студент 2 курса

Группы ИИ-21

Кабак Д. Н.

**Проверил:**

Козинский А. А.

Брест 2022

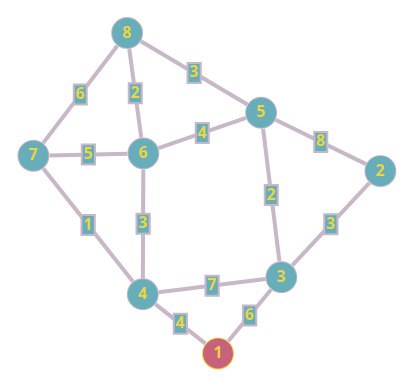
**Цель:** научиться создавать алгоритмы Прима и Крускаля.

**Ход работы:**

**Вариант 3**

{1,4},{1,3},{2,5},{2,3},{3,4},{3,5},{4,7},{4,6},{5,6},{5,8},{7,8},{7,6},{6,8}[4,6.8,3,7,2,

12,3,4,3,6,5,2]

#include <set>

#include <iostream>

#include <cstring>

#include <vector>

#include <algorithm>

#include "../graphs.h"

#define INF 1000000000

using namespace std;

void PRIMA(int \*\*graph, int tops){

int selected[tops];

memset(selected, false, sizeof(selected));

int no\_edge = 0;

int x, y;

while (no\_edge < tops) {

int min = INF;

x = 0;

y = 0;

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

if (selected[i]) {

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

if (!selected[j] && graph[i][j]) {

if (min > graph[i][j]) {

min = graph[i][j];

x = i;

y = j;

}

}

}

}

}

cout << x + 1 << " - " << y + 1 << " : " << graph[x][y];

cout << endl;

selected[y] = true;

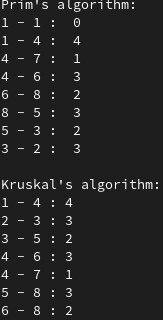
no\_edge++;

}

}

void KRUSKAL(std::vector<std::vector<int>> &adjacencyMatrix){

int max\_node = adjacencyMatrix.size();

 std::vector<std::vector<int>> distance(max\_node, std::vector<int>(max\_node, INT\_MAX));

std::vector<std::vector<int>> tree(max\_node, std::vector<int>(max\_node, 0));

std::vector<std::vector<int>> edges;

for (int i = 0; i < max\_node; i++){

for (int j = 0; j < max\_node; j++){

if (adjacencyMatrix[i][j] != 0){

edges.push\_back({ adjacencyMatrix[i][j], i, j });

}

}

}

std::sort(edges.begin(), edges.end());

std::vector<int> tree\_id(max\_node);

for (int i = 0; i < max\_node; i++){

tree\_id[i] = i;

}

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

int cost = edges[i][0];

int a = edges[i][1];

int b = edges[i][2];

if (tree\_id[a] != tree\_id[b]){

tree[a][b] = cost;

tree[b][a] = cost;

int old\_id = tree\_id[b];

int new\_id = tree\_id[a];

for (int j = 0; j < max\_node; j++){

if (tree\_id[j] == old\_id){

tree\_id[j] = new\_id;

}

}

}

}

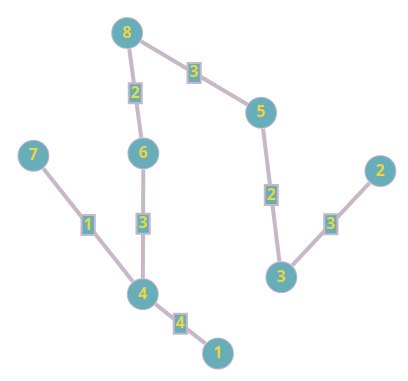
for(int i = 0; i < max\_node; i++)

for(int j = i; j < max\_node; j++)

if(tree[i][j] != 0)

cout << i + 1 << " - " << j + 1 << " : " << tree[i][j] << endl;

}



int main(){

char path[] = "cons";

const int TOPS = getTopsCount(path), EDGES = getEdgesCount(path);

int \*\*adjacencyMatrix = adjacencyMatrixFromConnectionsWithHeights(path);

vector<vector<int>> graph(TOPS, vector<int>(TOPS, 0));

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

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

graph[i][j] = adjacencyMatrix[i][j];

}

}

cout << "Prim's algorithm:" << endl;

PRIMA(adjacencyMatrix, TOPS);

cout << endl << "Kruskal's algorithm:" << endl;

KRUSKAL(graph);

}

**Вывод:** в ходе лабораторной работы я научился создавать алгоритмы Крускала и Прима.