**ЛАБОРАТОРНА РОБОТА №15**

## Алгоритми програмування задач з графами. Побудова остовного дерева (каркаса) графа мінімальної вартості.

**Мета:**

* Використовуючи алгоритми Прима, Крускала навчитись будувати мінімальне кістякове дерево.

**Загальна інформація**

Завдання складається з двох частин, обидві обов'язкові для виконання. Необхідно реалізувати два алгоритму побудови каркаса графа мінімальної вартості: алгоритм Прима і алгоритм Краскала.

**Хід виконання**

Розглянемо деякий зв'язний неорієнтований зважений граф.

Каркасом, або остовним деревом для цього графа називається зв'язний підграф цього графа, що містить всі вершини графа і не має циклів. Кількість ребер в каркасі зв'язкового графа завжди на одиницю менше кількості вершин графа.

Ясно, що у графа може бути кілька каркасів. Наприклад, різні каркаси можна побудувати, запускаючи [пошук в глибину](https://translate.google.com/translate?hl=ru&prev=_t&sl=ru&tl=uk&u=http://hci.fenster.name/304y/lab6/%23search#search) від різних вершин графа. Назвемо вартістю каркаса суму ваг ребер, що входять в нього.

**Постановка задачі**

*1.Розробити та налагодити проекти алгоритмів Пріма, Краскала на графах в середовищі програмування.*

2.Перевірте роботу програм за власно побудованими графами.(мін. 5-9 вузлів)

3.*Кожен малює свій граф (бажано змоделювати предметну область, потім зробити математичну модель у вигляді графа).* *Зробіть перевірку розвязку, розв’язавши задачу математично. Вставити модель та розвязок у звіт.*

# 4.Реалізувати подані алгоритми за технологією структурного програмування, тобто розбивши реалізацію на певні блоки- підпрограми (функції.)

5.Оформіть електронний звіт про виконання проєктів ЛР15.

**Задача 1 Алгоритм Прима**

**3)Лістинг програми;**

#include <iostream>

#include <cstdio>

#include <fstream>

#include <Windows.h>

const int INF = 1e9;

using namespace std;

void Algorithm\_Prima(int\*\* array\_weights, int kil\_vertix);

void Show\_resul\_of\_Prim(int\* parent, int\*\* array\_weights, int kil\_vertix);

int Find\_min\_index\_for\_give\_u(int\* key, bool\* viseted\_vert, int kil\_vertix);

void Select\_menu();

int main()

{

setlocale(LC\_ALL, "ukr");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int P = 1, Pokazcuk;

while (P != 0)

{

Select\_menu();

cin >> Pokazcuk;

switch (Pokazcuk)

{

case 1:

{

int kil\_vertix;

cout << "Введіть кількість вершин: ";

cin >> kil\_vertix;

int\*\* array\_weights = new int\* [kil\_vertix];

for (int i = 0; i < kil\_vertix; i++)

{

array\_weights[i] = new int[kil\_vertix];

for (int j = 0; j < kil\_vertix; j++)

{

cin >> array\_weights[i][j];

}

}

Algorithm\_Prima(array\_weights, kil\_vertix);

break;

}

case 2:

{

}

case 3:

{

P = 0;

break;

}

default:

cout << "Ввели неправильне значення" << endl;

break;

}

}

}

void Algorithm\_Prima(int\*\* array\_weights, int kil\_vertix)

{

int\* key = new int[kil\_vertix];

int\* parent = new int[kil\_vertix];

bool\* viseted\_vert = new bool[kil\_vertix];

for (int i = 0; i < kil\_vertix; i++)

{

key[i] = INF;

parent[i] = -1;

viseted\_vert[i] = false;

}

key[0] = 0; //умовно задали початкову вершину | масив мінімальних довжин відносно усіх інших пройжених вершин

for (int counter = 0; counter < kil\_vertix - 1; counter++)

{

int u = Find\_min\_index\_for\_give\_u(key, viseted\_vert, kil\_vertix); // поветрає номер рядка який містить мінімальне ребро відносно попередніх вершин

viseted\_vert[u] = true;

for (int v = 0; v < kil\_vertix; v++)

{

if (array\_weights[u][v] && !viseted\_vert[v] && array\_weights[u][v] < key[v])

{

parent[v] = u; // порядок пройдених вершин | запис фактичної шапки матриці

key[v] = array\_weights[u][v];

}

}

//можна по виводити key\parent\viseted

}

Show\_resul\_of\_Prim(parent, array\_weights, kil\_vertix);

}

int Find\_min\_index\_for\_give\_u(int\* key, bool\* viseted\_vert, int kil\_vertix) // шукає мінімальне значення по рядку і повертає індекс вершини

{

int min\_idx = -1, min\_val = INF;

for (int i = 0; i < kil\_vertix; i++)

{

if (!viseted\_vert[i] && key[i] < min\_val)

{

min\_idx = i;

min\_val = key[i];

}

}

return min\_idx;

}

void Show\_resul\_of\_Prim(int\* parent, int\*\* array\_weights, int kil\_vertix)

{

cout << "Кістякове дерево: " << endl;

int sum\_min\_edges\_Prim = 0;

for (int i = 0; i < kil\_vertix; i++)

{

if (array\_weights[i][parent[i]] > -1)

{

cout << parent[i] + 1 << " - " << i + 1 << "\t" << array\_weights[i][parent[i]] << endl;

sum\_min\_edges\_Prim += array\_weights[i][parent[i]];

}

}

cout << "Вага кістякового дерева: " << sum\_min\_edges\_Prim << endl;

cout << "Посортований вивід: " << endl;

for (int j = 1; j <= kil\_vertix; j++)

{

for (int i = 0; i < kil\_vertix; i++)

{

if (array\_weights[i][parent[i]] > -1 && j == parent[i] + 1)

{

cout << parent[i] + 1 << " - " << i + 1 << "\t" << array\_weights[i][parent[i]] << endl;

}

}

}

}

void Select\_menu()

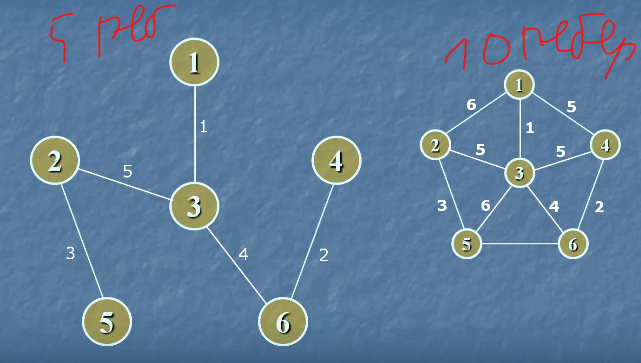
{

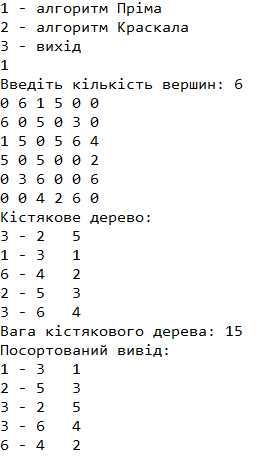
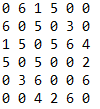
cout << "1 - алгоритм Пріма" << endl;

cout << "2 - алгоритм Краскала" << endl;

cout << "3 - вихід" << endl;

}

**5)Результати**



**Задача 2 Алгоритм Крускала**

**3)Лістинг програми;**

#include <iostream>

#include <cstdio>

#include <fstream>

#include <Windows.h>

#include <algorithm>

const int MAXN = 100; // максимальна кількість вершин

using namespace std;

struct Edge

{

int u, v, weight;

};

Edge edges[MAXN \* MAXN]; // масив ребер

int parent[MAXN]; // масив батьківських вершин

bool cmp(Edge a, Edge b);

int find(int x);

void merge(int a, int b);

void kruskalMST(int\*\* array\_weight\_Krus, int n);

int main()

{

setlocale(LC\_ALL, "ukr");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int kil\_vertix\_Krus;

cout << "Введіть кількість вершин: ";

cin >> kil\_vertix\_Krus;

int\*\* array\_weight\_Krus = new int\* [kil\_vertix\_Krus];

for (int i = 0; i < kil\_vertix\_Krus; i++)

{

array\_weight\_Krus[i] = new int[kil\_vertix\_Krus];

for (int j = 0; j < kil\_vertix\_Krus; j++)

{

cin >> array\_weight\_Krus[i][j];

}

}

kruskalMST(array\_weight\_Krus, kil\_vertix\_Krus);

}

void kruskalMST(int\*\* array\_weight\_Krus, int kil\_vertix\_Krus)

{

int edgeCount = 0;

for (int i = 0; i < kil\_vertix\_Krus; i++)

{

for (int j = i; j < kil\_vertix\_Krus; j++)

{

if (array\_weight\_Krus[i][j] > 0)

{

edges[edgeCount].u = i;

edges[edgeCount].v = j;

edges[edgeCount].weight = array\_weight\_Krus[i][j];

edgeCount++;

}

}

}

cout << "Вивід списка перед сортуванням:" << endl;

for (int i = 0; i < kil\_vertix\_Krus; i++)

{

cout << edges[i].u + 1 << " - " << edges[i].v + 1 << "\t" << edges[i].weight << endl;

}

sort(edges, edges + edgeCount, cmp);

cout << "Після" << endl;

for (int i = 0; i <= kil\_vertix\_Krus; i++)

{

cout << edges[i].u + 1 << " - " << edges[i].v + 1 << "\t" << edges[i].weight << endl;

}

cout << endl;

for (int i = 0; i < kil\_vertix\_Krus; i++)

{

parent[i] = i;

}

int sum\_min\_edges\_Krus = 0;

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

{

if (find(edges[i].u) != find(edges[i].v))

{

merge(edges[i].u, edges[i].v);

cout << edges[i].u + 1 << " - " << edges[i].v + 1 << "\t" << edges[i].weight << endl;

sum\_min\_edges\_Krus += edges[i].weight;

}

}

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

{

cout << parent[i] << " ";

}

cout << endl;

cout << "Weight of MST: " << sum\_min\_edges\_Krus << endl;

}

bool cmp(Edge a, Edge b)

{

return a.weight < b.weight;

}

int find(int x)

{

if (x == parent[x])

return x;

return find(parent[x]);

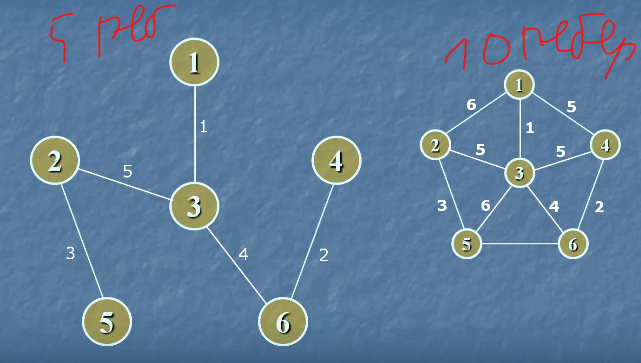
}

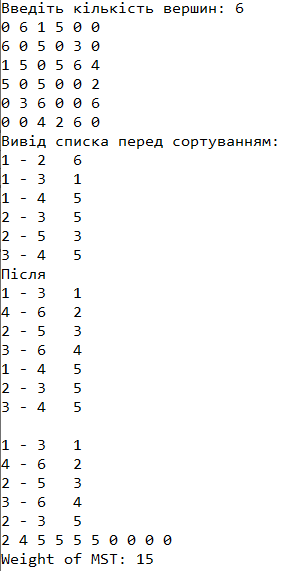
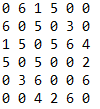
void merge(int a, int b)

{

parent[find(a)] = find(b);

}

**5)Результати**



**Висновок:** Навчився використовувати алгоритм Прима і Крускала, також навчився будувати мінімальне кістякове дерево.