МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Київський національний університет будівництва і архітектури

Кафедра прикладної математики

**Лабораторна робота №4.**

**Реалізація алгоритмів Прима та Крускала.**

Виконала ст.групи

ІСТ-21 факультету АІТ

Пасічник М.С

**Мета роботи:**

Дослідити задачу знаходження остовного підграфа мінімальної ваги, використовуючи алгоритм Прима та алгоритм Крускала .

**Завдання**

1. Навести нерієнтований граф G(V,E), V=8, E=15.

2. Для заданого графа побудувати остовне дерево мінімальної вартості:

а) за допомогою алгоритму Прима;

б) за допомогою алгоритму Крускала.

3. Написати програми, що реалізують ці алгоритми.

4. Написати процедуру обчислення вартості побудованого остовного дерева.

Лістинг програми (алгоритм Крускала )

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

typedef pair<int, int> iPair;

struct Graph

{

int V, E;

vector< pair<int, iPair> > edges;

Graph(int V, int E)

{

this->V = V;

this->E = E;

}

void addEdge(int u, int v, int w)

{

edges.push\_back({ w, {u, v} });

}

int kruskalMST();

};

struct DisjointSets

{

int\* parent, \* rnk;

int n;

DisjointSets(int n)

{

this->n = n;

parent = new int[n + 1];

rnk = new int[n + 1];

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

{

rnk[i] = 0;

parent[i] = i;

}

}

int find(int u)

{

if (u != parent[u])

parent[u] = find(parent[u]);

return parent[u];

}

void merge(int x, int y)

{

x = find(x), y = find(y);

if (rnk[x] > rnk[y])

parent[y] = x;

else

parent[x] = y;

if (rnk[x] == rnk[y])

rnk[y]++;

}

};

int Graph::kruskalMST()

{

int mst\_wt = 0;

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

DisjointSets ds(V);

vector< pair<int, iPair> >::iterator it;

for (it = edges.begin(); it != edges.end(); it++)

{

int u = it->second.first;

int v = it->second.second;

int set\_u = ds.find(u);

int set\_v = ds.find(v);

if (set\_u != set\_v)

{

cout << u << " - " << v << endl;

mst\_wt += it->first;

ds.merge(set\_u, set\_v);

}

}

return mst\_wt;

}

int main()

{

int V = 9, E = 14;

Graph g(V, E);

g.addEdge(0, 1, 4);

g.addEdge(0, 7, 8);

g.addEdge(1, 2, 8);

g.addEdge(1, 7, 11);

g.addEdge(2, 3, 7);

g.addEdge(2, 8, 2);

g.addEdge(2, 5, 4);

g.addEdge(3, 4, 9);

g.addEdge(3, 5, 14);

g.addEdge(4, 5, 10);

g.addEdge(5, 6, 2);

g.addEdge(6, 7, 1);

g.addEdge(6, 8, 6);

g.addEdge(7, 8, 7);

g.addEdge(7, 1, 3);

g.addEdge(7, 2, 7);

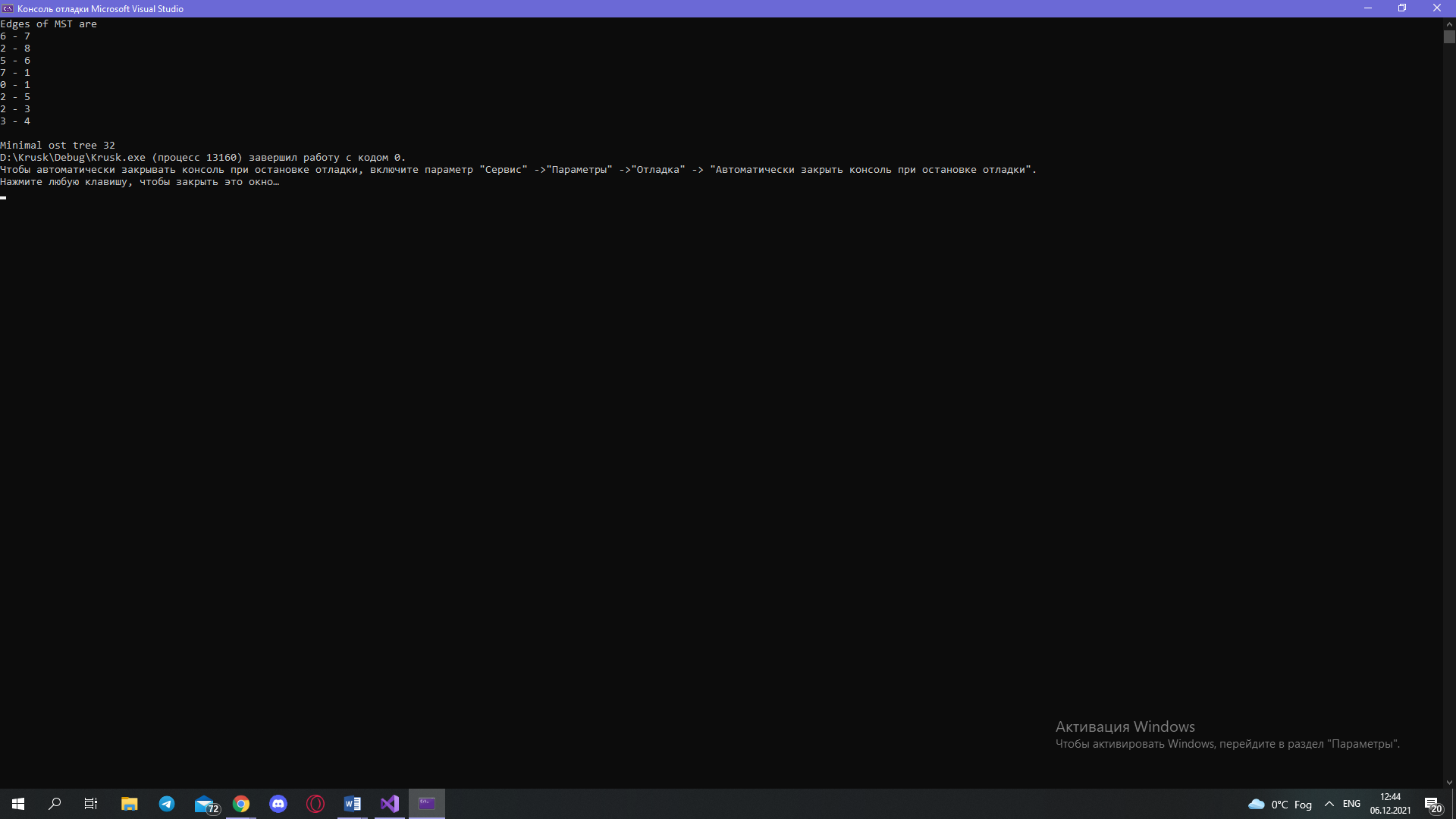
cout << "Edges of MST are \n";

int mst\_wt = g.kruskalMST();

cout << "\nWeight of MST is " << mst\_wt;

return 0;

}



Операції -

g.addEdge(0, 1, 4);

g.addEdge(0, 7, 8);

g.addEdge(1, 2, 8);

g.addEdge(1, 7, 11);

g.addEdge(2, 3, 7);

g.addEdge(2, 8, 2);

g.addEdge(2, 5, 4);

g.addEdge(3, 4, 9);

g.addEdge(3, 5, 14);

g.addEdge(4, 5, 10);

g.addEdge(5, 6, 2);

g.addEdge(6, 7, 1);

g.addEdge(6, 8, 6);

g.addEdge(7, 8, 7);

g.addEdge(7, 1, 3);

g.addEdge(7, 2, 7);

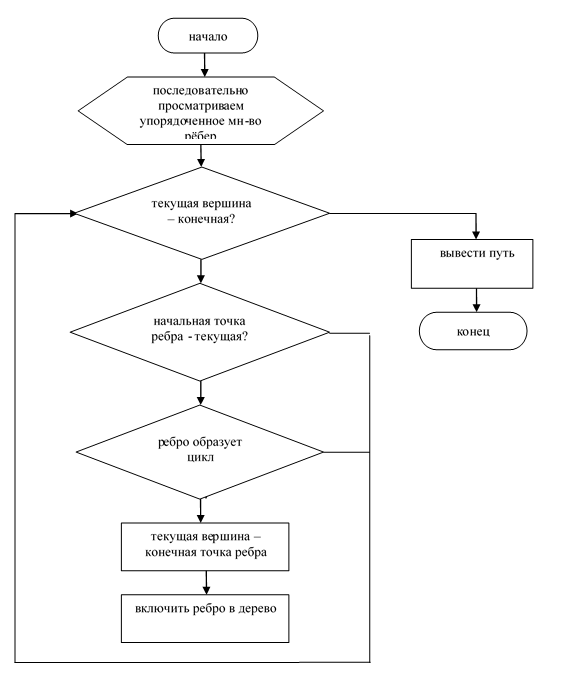
Сумма мінімального остовного дерева рівня 11

**Алгоритм**

Відсортуємо всі ребра вихідного графа за зростанням і сформуємо з них чергу так, щоб у "голові" черги знаходилося ребро з найменшою вагою, а в "хвості" - з найбільшою.

Спочатку поточна множина ребер порожня .

. Доки це можливо, проводиться наступна операція: з усіх ребер, додавання яких до вже наявної множини не викликає появи в ньому циклу, вибирається ребро мінімальної ваги і додається до вже наявної множини. Коли таких ребер більше немає, алгоритм завершено. Підграф даного графа, що містить всі його вершини і знайдене безліч ребер, є його остовним деревом мінімальної ваги.



**Час роботи алгоритма :**

Час роботи алгоритму Крускала для графа G = (V, Е) залежить від реалізації структури даних.

До початку роботи алгоритму необхідно відсортувати ребра за вагою, це вимагає O(E×log(E)) часу.

Складність цього алгоритму збігається зі складністю використовуваного алгоритму сортування, оскільки число операцій у циклі while лінійно за кількістю ребер. Тому складність алгоритму Крускала пошуку МОД дорівнює O (E\* log Е)

**Лістинг (алгоритм Прима )**

#include <iostream>

#include <cstring>

using namespace std;

#define INF 9999999

#define V 5

int G[V][V] = {

{0,4,8,7,2},

{4,0,1,5,9},

{8,1,0,4,5},

{7,5,4,0,7},

{2,9,5,7,0}

};

int main() {

int no\_edge;

int selected[V];

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

no\_edge = 0;

selected[0] = true;

int x;

int y;

cout << "Edge" << " : " << "Weight";

cout << endl;

while (no\_edge < V - 1) {

int min = INF;

x = 0;

y = 0;

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

if (selected[i]) {

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

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

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

min = G[i][j];

x = i;

y = j;

}

}

}

}

}

cout << x << " - " << y << " : " << G[x][y];

cout << endl;

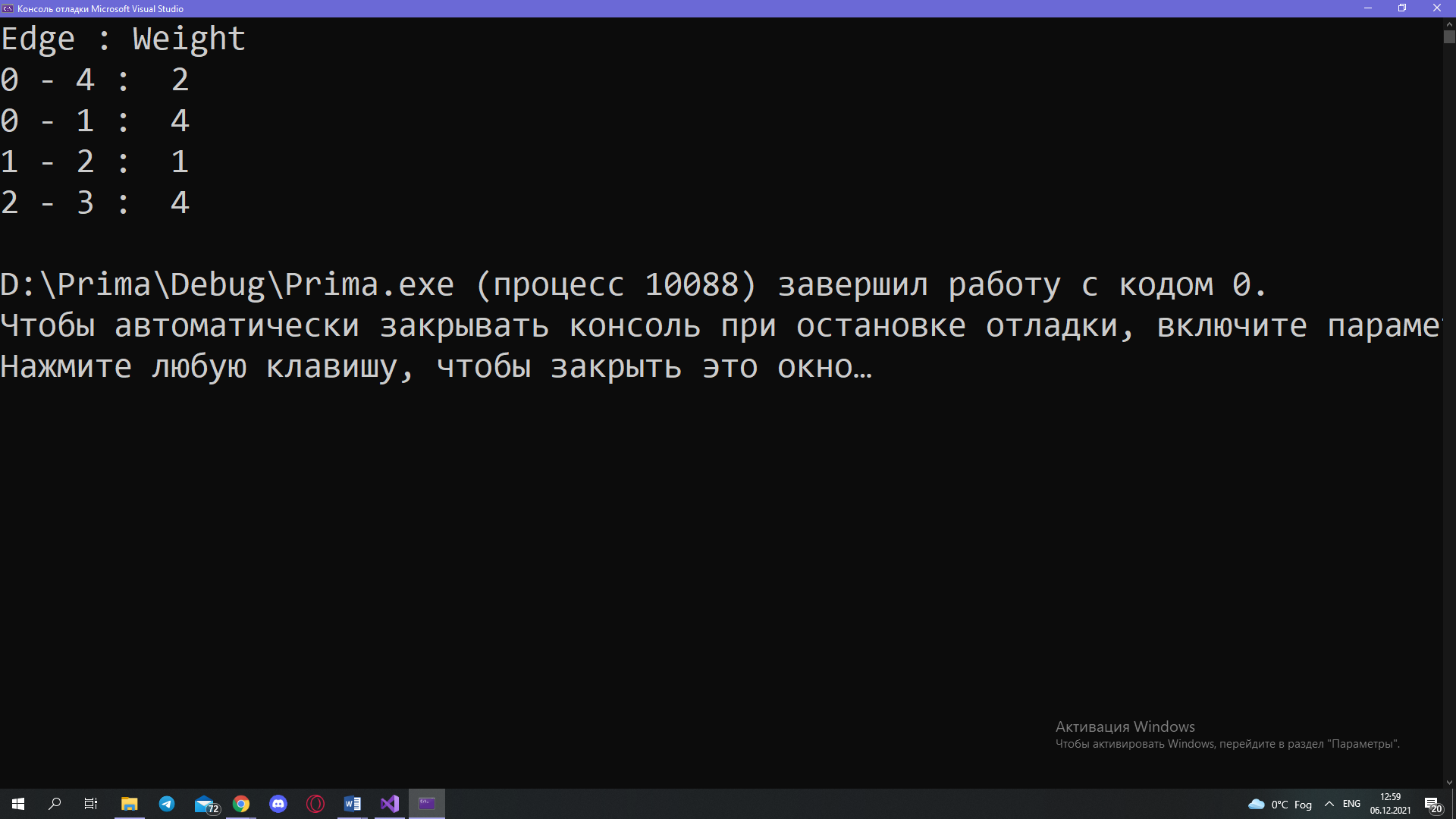
selected[y] = true;

no\_edge++;

}

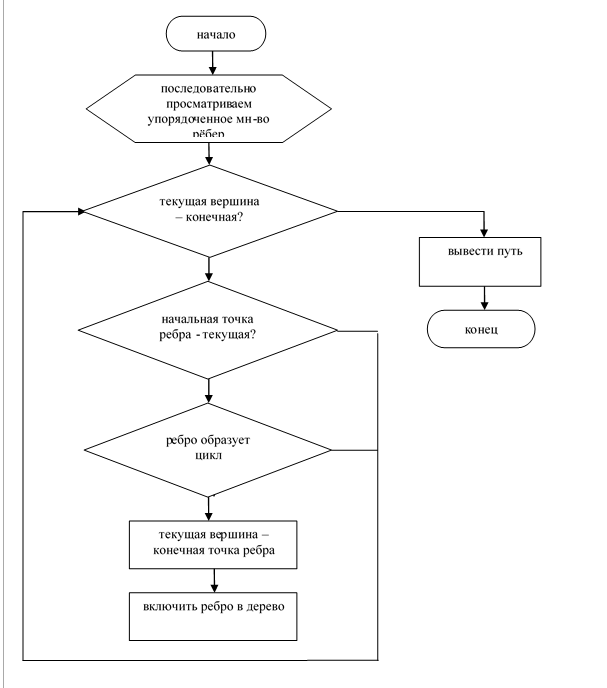
return 0;

}



**Алгоритм**

 Шуканий мінімальний остов будується поступово, додаванням в нього ребер по одному. Спочатку остов покладається складається з єдиною вершини (її можна вибрати довільно). Потім вибирається ребро мінімальної ваги, що виходить з цієї вершини, і додається в мінімальний остов. Після цього остов містить вже дві вершини, і тепер шукається і додається ребро мінімальної ваги, що має один кінець в одній з двох обраних вершин, а інший - навпаки, у всіх інших, крім цих двох. І так далі, тобто щоразу шукається мінімальне по вазі ребро, один кінець якого - вже взята в остов вершина, а інший кінець - ще не взяти, і це ребро додається в остов (якщо таких ребер кілька, можна взяти будь-яке). Цей процес повторюється до тих пір, поки остов чи не стане містити всі вершини (або, що те ж саме, n-1 ребро).



**Час роботи алгоритму**

Час роботи алгоритму Прима залежить від обраної реалізації черги з пріоритетами Q.

Тіло циклу while виконується |V| раз, а оскільки кожна операція займає час О (lg V), загальний час всіх викликів процедури становить О (V \* lg V). Таким чином, загальний час роботи алгоритму Прима становить O (V\*lg V+ Е\*lg V) = O (Е \* lg V), що збігається з часом роботи алгоритму Крускала.

**Контрольні запитання**.

1. Дайте визначення:

Граф – це сукупність об’єктів та зв’язків між ними.

Орієнтований граф – граф що містить дуги…

Степінь вершини – це число ребер, які інцидентні даній вершині.

Простий граф – граф у якому немає петель та кратних ребер.

Псевдо граф – допускається наявність кратних ребер( ті що мають одні і ті ж кінцеві вершини).

Дводольний граф – граф, множину вершин якого можна розбити на дві підмножини так, що кожне ребро графу має одну вершину з першої підмножини і одну з другої.

Ізоморфний граф –називається бієкція між множинами вершин графів f \ colon \ V_G \ rightarrow V_H така, що будь-які дві вершини *u* і *v* графа *G* суміжні, якщо і тільки якщо вершини *f (u)* і *f (v)* суміжні в графі *H* .

Мультиграф – це орієнтований граф, в якому допустимі кратні дуги, тобто є дуги, які мають однакові початкові і кінцеві вершини.

Підграф G'(V', E') графа G(V, E) називають остовним (каркасним), якщо V' = V, тобто множини вершин графа G і підграфа G' збігаються.

Зважений граф – називається граф (орграф) зі зваженими вершинами і / або ребрами (дугами).

1. Способи задання графів

* Матриця інцидентності
* Список
* Матриця суміжності

1. Постановка задачі пошуку остовного підграфу мінімальної ваги.

* Помістити кожну вершину у власний набір даних
* Обрати ребра в порядку зростання ваги.
* Для кожного ребра визначити чи є вершини, які визначають ребро, у різних наборах даних
* Якщо так, вставте цей край у набір мінімального кістяка і одночасно видаліть об'єднання, що містить кожну вершину в наборі, і, якщо ні, перейдіть до наступного ребра.
* Повтор доти, доки всі краї не будуть пройдені

1. Чи будуть коректно працювати алгоритми Крускала та Прима з графом, у якого є ребра з від’ємною вагою?

Так , дані алгоритми можуть коректно працювати з ребрами від'ємної ваги.

1. Дайте визначення:

Фрагмент – підмнодина вершин звязаних ребрами.

Ізольований фрагмент – називається фрагмент, який на даному етапі побудови не зв’язаний з іншими вершинами або фрагментами.

Ізольована вершина – вершина із петлями, але без інцидентних їй дуг чи ребер, вершина ступеня 0.

1. В чому полягає ідея алгоритмів Крускала та Прима?

Замість того, щоб починати з ребра, алгоритм Прима починається з вершини і продовжує додавати ребра з найменшою вагою, яких немає в дереві, доки всі вершини не будуть покриті.

Алгоритм Крускала спочатку поміщає кожну вершину у своє дерево,

а потім поступово об'єднує ці дерева, поєднуючи на кожній ітерації

два деякі дерева деякий руба. Перед початком виконання алгоритму,

всі ребра сортуються за вагою (у порядку невтрати). Потім починається

процес об'єднання: перебираються всі ребра від першого до останнього

(у порядку сортування), і якщо у поточного ребра його кінці належать

різним піддеревам, то ці піддерева об'єднуються, а ребро додається

до відповіді. Після закінчення перебору всіх ребер всі вершини виявляться

що належать одному піддереву, і відповідь знайдено.