МІНІСТЕРСТВО ОСВІТИ І НАУКИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»
Інститут комп'ютерних наук та інформаційних технологій

Кафедра систем штучного інтелекту

Лабораторна робота №4
з курсу “Дискретна математика”

Моделювання основних логічних операцій

Виконав:
ст. гр. КН-110
Помірко Олег

Викладач:
Мельникова Н.І.

Львів – 2018

Лабораторна робота № 4.

Тема: Основні операції над графами. Знаходження остова мінімальної ваги за алгоритмом Пріма-Краскала

Мета роботи: набуття практичних вмінь та навичок з використання алгоритмів Пріма і Краскала.

ТЕОРЕТИЧНІ ВІДОМОСТІ ТА ПРИКЛАДИ РОЗВ'ЯЗАННЯ ЗАДАЧ

Теорія графів дає простий, доступний і потужний інструмент побудови моделей прикладних задач, є ефективним засобом формалізації сучасних інженерних і наукових задач у різних областях знань.

Графом G називається пара множин (V, E), де V – множина вершин, перенумерованих числами 1, 2, ..., $n = \upsilon$; $V = \{\upsilon\}$, E – множина упорядкованих або неупорядкованих пар $e = (v', v'')$, $v' \in V$, $v'' \in V$, називаних дугами або ребрами, $E = \{e\}$. При цьому не має примусового значення, як вершини розташовані в просторі або площині і які конфігурації мають ребра.

Неорієнтованим графом G називається граф у якого ребра не мають напрямку. Такі ребра описуються неупорядкованою парою (v', v''). *Орієнтований граф (орграф)* – це граф ребра якого мають напрямок та можуть бути описані упорядкованою парою (v', v''). Упорядковане ребро називають *дугою*. Граф є *змішаним*, якщо наряду з орієнтованими ребрами (дугами) є також і неорієнтовані. При розв'язку задач змішаний граф зводиться до орграфа.

Кратними (паралельними) називаються ребра, які зв'язують одні і ті ж вершини. Якщо ребро виходить та й входить у дну і ту саму вершину, то таке ребро називається *петлею*.

Мультиграф – граф, який має кратні ребра. *Псевдограф* – граф, який має петлі. *Простий граф* – граф, який не має кратних ребер та петель.

Будь яке ребро e *інцедентно* двом вершинам (v',v''), які воно з'єднує. У свою чергу вершини (v',v'') інцендентні до ребра e . Дві вершини (v',v'') називають *суміжними*, якщо вони належать до одного й того самого ребра e , і *несуміжні* у протилежному випадку. Два *ребра називають суміжними*, якщо вони мають спільну вершину. Відношення суміжності як для вершин, так і для ребер є симетричним відношенням. *Степенем вершини* графа G називається число інцидентних їй ребер.

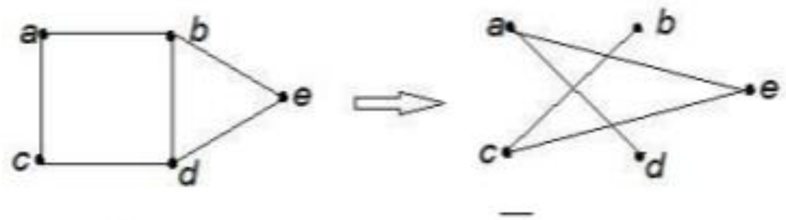
Граф, який не має ребер називається *пустим графом*, *нуль-графом*. Вершина графа, яка не інцедентна до жодного ребра, називається *ізольованою*. Вершина графа, яка інцедентна тільки до одного ребра, називається *звисаючою*.

Частина $G'=(V',E')$ графа $G=(V,E)$ називається *підграфом* графа G , якщо $V' \subseteq V$ і E' складається з тих і тільки тих ребер e = (v',v''), у яких обидві кінцеві вершини $v',v'' \in V'$. Частина $G'=(V',E')$ називається *суграфом* або *остовим підграфом* графа G , якщо виконано умови: $V' = V$, $E' \subseteq E$.

<u>Операції над графами</u>

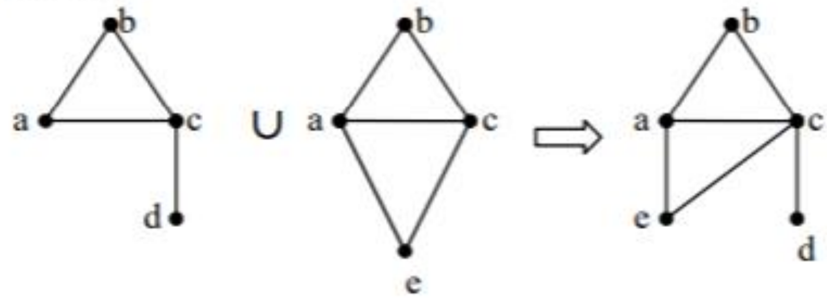
1. *Вилученням ребра e* ($e \in E$) з графа $G=(V,E)$ – є така операція внаслідок якої отримаємо новий граф G_1 для якого G_1 = $(V,E\setminus\{e\})$.

2. *Доповненням графа* $G=(V,E)$ називається граф $G=(V,E')$, якщо він має одну і ту саму кількість вершин та дві його вершини суміжні тоді і тільки тоді коли вони не суміжні в G (тобто ребро $(v_i,v_j)\in E'$ тоді коли $(v_i,v_j)\notin E$). Наприклад:



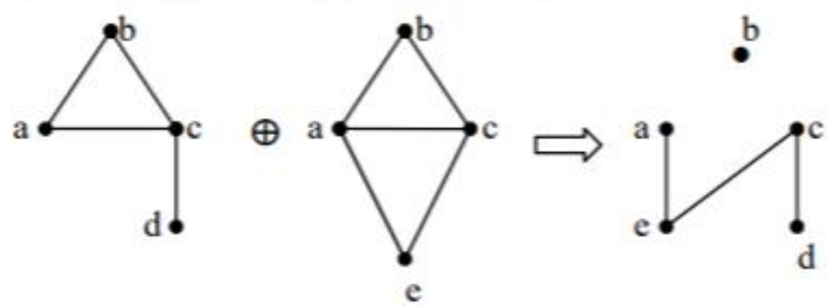
G $\qquad\qquad$ $\overline{G}$

3. *Об'єднанням графів* $G_1 = (V_1, E_1)$ та $G_2 = (V_2, E_2)$ називається граф $G = (V, E) = G_1 \bigcup G_2$ у якому $V = V_1 \cup V_2$ та $E = E_1 \cup E_2$. Наприклад:



G_1 $\qquad\qquad$ G_2 $\qquad\qquad$ $G_1 \cup G_2$

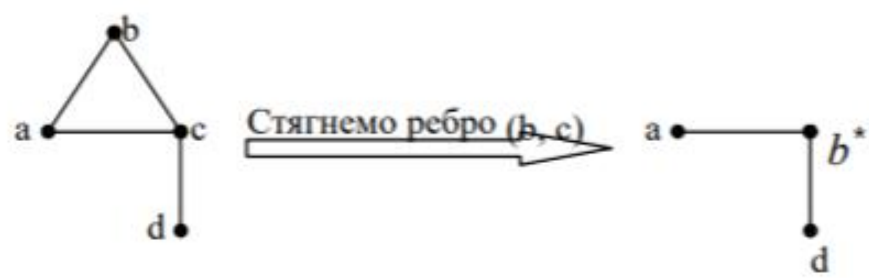
4. *Кільцевою сумою графів* $G_1 = (V_1, E_1)$ та $G_2 = (V_2, E_2)$ називається граф $G = (V, E) = G_1 \oplus G_2$ у якому $V = V_1 \cup V_2$ та $E = E_1 \Delta E_2 = (E_1 \cup E_2) \setminus (E_1 \cap E_2)$. Наприклад:



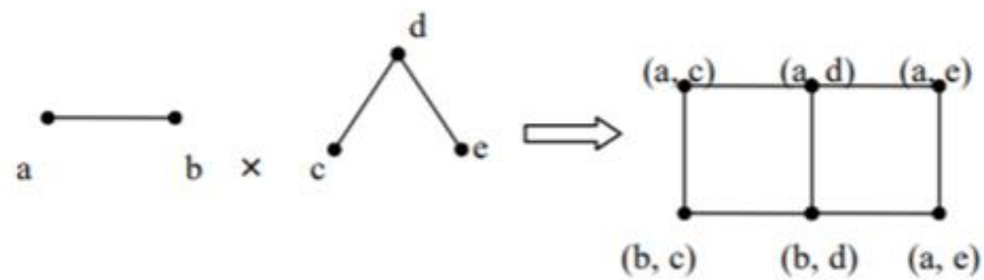
G_1 $\qquad\qquad$ G_2 $\qquad\qquad$ $G_1 \oplus G_2$

5. *Розщеплення (роздвоєння) вершини графа*. Нехай υ – вершина графа $G=(V, E)$. Множину усіх суміжних з нею вершин довільним чином розділимо на дві множини $N_1(\upsilon)$ та $N_2(\upsilon)$, таких що $N_1(\upsilon)\bigcup N_2(\upsilon)=V$. Видаливши вершину υ разом з інцидентними їй ребрами, додамо дві нові вершини υ_1 та υ_2, які з'єднані ребром (υ_1, υ_2). Вершину υ_1 з'єднаємо ребром з кожною вершиною множини $N_1(\upsilon)$, а вершину υ_2 – з кожною вершиною множини $N_2(\upsilon)$. Таким чином з графа G отримаємо новий граф G_v^*. Виконана операція називається *розщепленням вершини* υ. Наприклад:

6. *Стягування ребра (дуги)*. Ця операція означає видалення ребра та ототожнення його суміжних вершин. Граф G_1 стягується до графа G_2, якщо граф G_2 може бути отриманим з G_1 в результаті деякої послідовності стягування ребер (дуг). Наприклад:



7. Добутком графів $G_1 = (V_1, E_1)$ та $G_2 = (V_2, E_2)$ називається граф $G = G_1 \times G_2$ у якого $V = V_1 \times V_2$ а множина ребер визначається наступним чином: вершини (u_1, υ_1) та (u_2, υ_2) суміжні у G тоді і тільки тоді коли $u_1 = u_2$ і υ_1 та υ_2 суміжні у G_2, або $\upsilon_1 = \upsilon_2$ і u_1, u_2 суміжні у G_1. Наприклад:



G_1 $\qquad$ G_2 $\qquad$ $G_1 \times G_2$

Таблицею (матрицею) суміжності $R = [r_{ij}]$ графа $G = (V, E)$ називається квадратна матриця порядку n (n – число вершин графа), елементи якої $r_{i,j}$ (i=1, 2, …n; j=1,2, …n) визначаються наступним чином:

$$r_{ij} = \begin{cases} 1, \text{ якщо існує дуга з } \upsilon_i \text{ в } \upsilon_j; \\ 0, \text{ в іншому випадку.} \end{cases}$$

Матриця суміжності повністю визначає структуру графа.

Ексцентриситет вершини графа – відстань до максимально віддаленої від неї вершини. Для графа, для якого не визначена вага його ребер, відстань визначається у вигляді числа ребер.

Радіус графа – мінімальний ексцентриситет вершин.

Діаметр графа – максимальний ексцентриситет вершин.

Діаметром зв'язного графа називається максимально можлива довжина між двома його вершинами.

Нехай дано неорієнтований граф $G=(V,E)$. *Маршрутом* довжини $j-1$ з вершини υ_1 у υ_j називається послідовність $M=\{(\upsilon_1,\upsilon_2),(\upsilon_2,\upsilon_3),...,(\upsilon_i,\upsilon_{i+1}),...,(\upsilon_{j-1},\upsilon_j)\}$, яка складається з ребер $j=(v_s,v_{s+1})\in E$, при цьому кожні два сусідніх ребра мають спільну кінцеву вершину. Маршрут називається *ланцюгом*, якщо всі його ребра різні. Відкритий ланцюг називається *шляхом*, якщо всі його вершини різні. Замкнений ланцюг називається *циклом*, якщо різні всі його вершини, за винятком кінцевих. Шлях і цикл називаються *гамільтоновими*, якщо вони проходять через усі вершини графа.

Алгоритми знаходження найкоротшого кістякового дерева

Алгоритм Прима для даного n-вершинного графа $G=(V, E)$ будує по кроках $s=1, 2, \ldots, l\le n-1$ зростаюче дерево $D_s=(V_s, E_s)$, $V_s\subseteq V$, $E_s\subseteq E$. $S=1$. Фіксуємо довільну вершину v_0, серед усіх ребер, інцидентних вершині v_0 знаходимо найкоротше ребро $e_1=(v_0, v_1)$. Покладемо, що $D_1=(V_1, E_1)$, $V_1=\{v_0, v_1\}$, $E_1=\{e_1\}$ і переходимо до кроку $s=2$.

Нехай здійснено $s<n-1$ кроків, у результаті чого в графі G виділено зростаюче дерево $D_s=(V_s, E_s)$. Тоді на кроці $(s+1)$ серед усіх ребер $e=(v',v'')$, таких що $v'\in V_s$, $v''\in(V\setminus V_s)$, знаходимо найкоротше ребро $e_{s+1}=(v_r, v_{s+1})$ і приєднуємо його до дерева D_s, у результаті чого одержуємо дерево $D_{s+1}=(V_{s+1}, E_{s+1})$, $V_{s+1}= V_s\cup\{v_{s+1}\}$, $E_{s+1}= E_s\cup\{e_{s+1}\}$. Алгоритм закінчує свою роботу в двох випадках: 1) результативно на кроці $s=n-1$ у випадку, якщо граф G зв'язний; 2) безрезультатно, якщо G – незв'язний граф.

Алгоритм Краскала. Перший етап – підготовчий, для даного графа G упорядковуються ребра $e\in E$ у послідовність $e_1, e_2, \ldots, e_m$, $m=|E|$, у порядку неспадання ваг цих ребер: $w(e_1)\le w(e_2)\le\ldots\le w(e_s)\le\ldots\le w(e_m)$.

Другий етап виконується по кроках s=1, 2, …, $m_0 \le m$ у такий спосіб. На кроках s=1, 2 ребра e_1, e_2 з послідовності офарблюються. На кожному наступному кроці s розглядається ребро e_s з послідовності, і воно офарблюється тоді і тільки тоді, коли не утворює циклу з ребрами, пофарбованими на попередніх кроках. У противному випадку ребро e_s умовно викреслюється з графа $G=(V, E)$. Алгоритм закінчує роботу на кроці $s=m_0$, коли пофарбованим виявиться (n–1) по рахунку ребро e_s, $n=|V|$, тому що по необхідності n–1 пофарбованих ребер утворюють кістякове дерево n-вершинного графа.

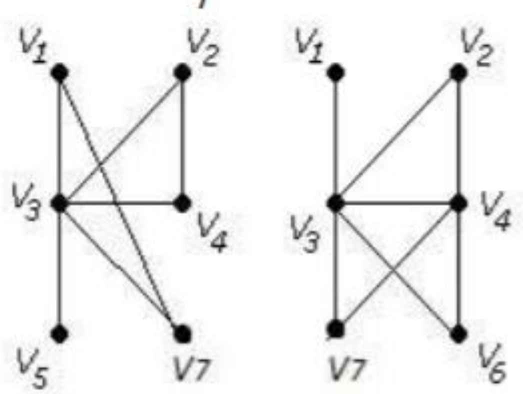
Варіант №7

Завдання № 1. Розв'язати на графах наступні задачі:

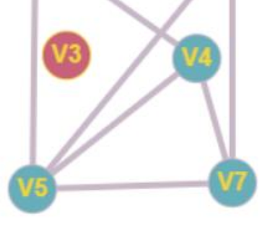
1. Виконати наступні операції над графами:

1) знайти доповнення до першого графу,
2) об'єднання графів,
3) кільцеву суму G1 та G2 (G1+G2),
4) розщепити вершину у другому графі,
5) виділити підграф A, що складається з 3-х вершин в G1 і знайти стягнення A в G1 (G1\ A), 6) добуток графів.

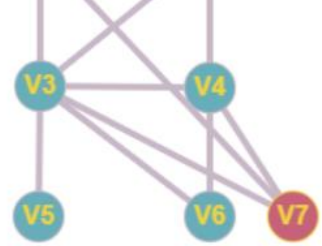
7



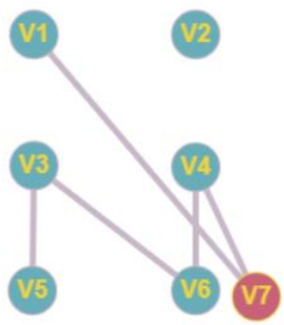
1) $\overline{G}_1$



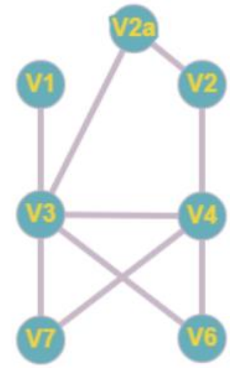
2) $G_2 \cup G_1$



3) $G_2 \oplus G_1$

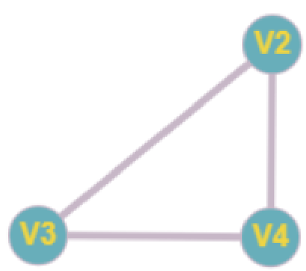


4) розщепити вершину у другому графі

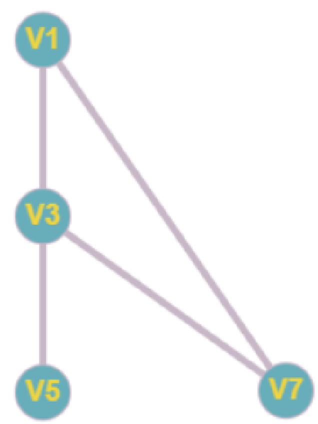


5) виділити підграф А, що складається з 3-х вершин в G1 і знайти стягнення А в G1 (G1\ A)

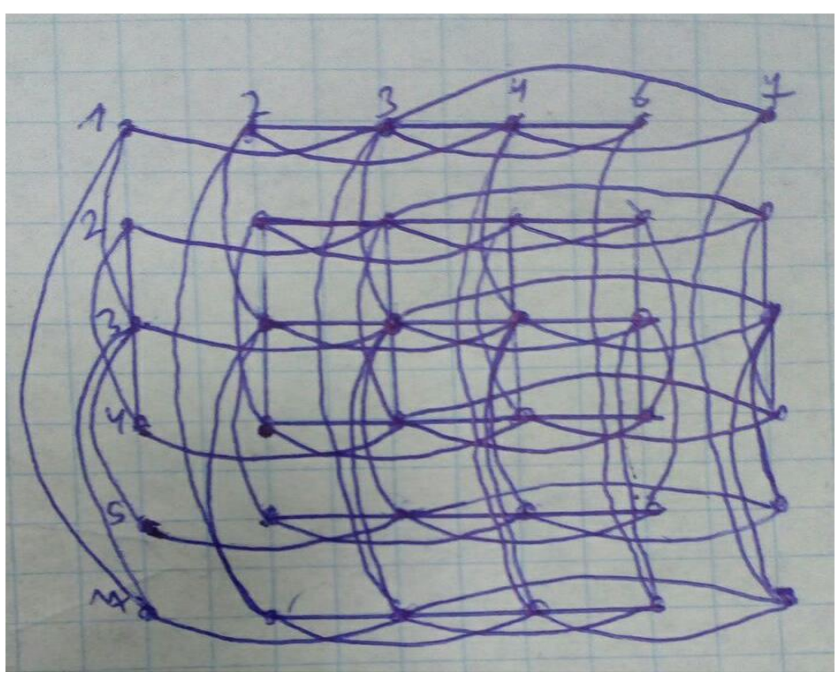
A:



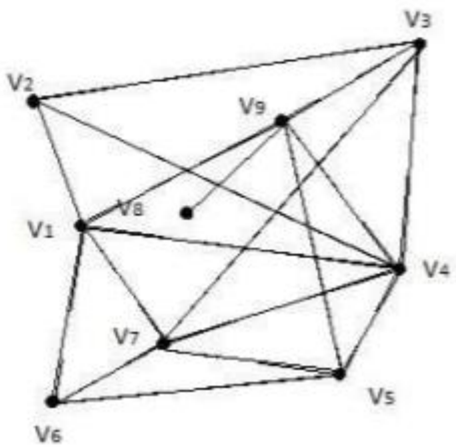
(G1\ A):



6) добуток графів.



Завдання №2. Знайти таблицю суміжності та діаметр графа

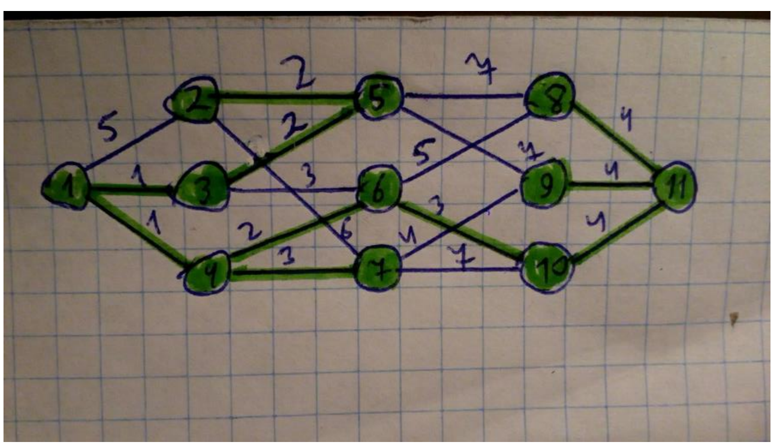


	V1	V2	V3	V4	V5	V6	V7	V8	V9
V1	0	1	0	1	0	1	1	0	1
V2	1	0	1	1	0	0	0	0	0
V3	0	1	0	1	0	0	1	0	1
V4	1	1	1	0	1	0	1	0	1
V5	0	0	0	1	0	1	1	0	1
V6	1	0	0	0	1	0	1	0	0
V7	1	0	1	1	1	1	0	0	0
V8	0	0	0	0	0	0	0	0	1
V9	1	0	1	1	1	0	0	1	0

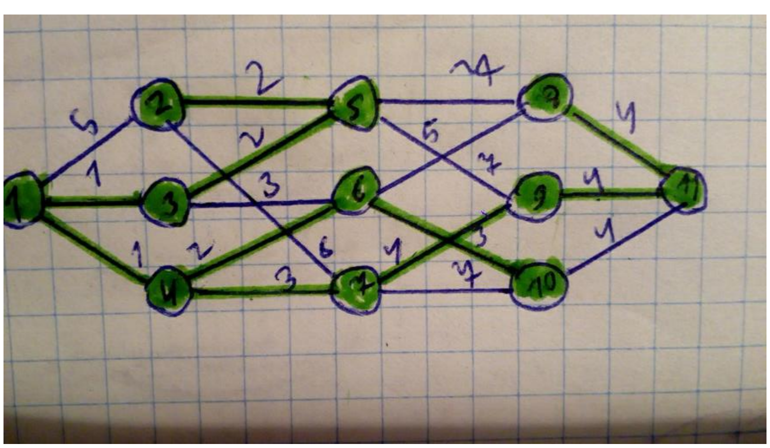
Max={1;2;3}
d=3 (V8 V9; V9 V1; V1 V2)

Завдання №3. Знайти двома методами (Краскала і Прима) мінімальне остове дерево графа.

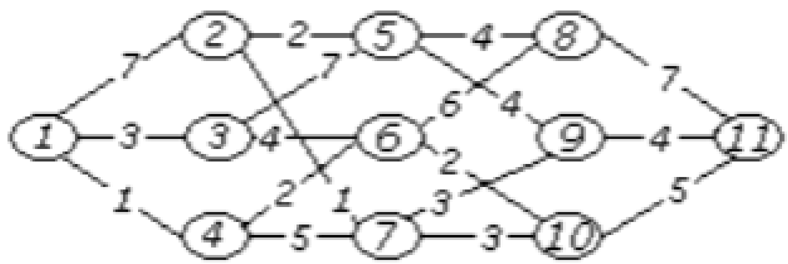
Краскала:



Прима:



Варіант № 7 За алгоритмом Прима знайти мінімальне остове дерево графа. Етапи розв'язання задачі виводити на екран. Протестувати розроблену програму на наступному графі:




```c
#include <stdlib.h>
#include <stdio.h>
//створення структури для подальшого зважування ребер
struct A
 {
 int beginv;
 int finishv;
 int w;
 };
void sorting(struct A *array,int first,int last)
 {
 int i = first, j = last,m = array[(first + last) / 2].w;
 struct A swap;
 do
  {
  while (array[i].w < m)
  i++;
  while (array[j].w > m)
  j--;
  if(i <= j)
   {
   if (i < j)
    {
    swap=array[i];
    array[i]=array[j];
    array[j]=swap;
    }
   i++;
   j--;
  }
 }
 while (i <= j);
 if (i < last)
 sorting(array, i, last);
 if (first < j)
 sorting(array,first,j);
 }
 //функція для кістяка графу
 int func1(int array[],int lenght,int vertex)
 {
 for(int i=0;i<lenght;i++)
  {
  if(array[i]==vertex)
   {
   return 1;
   }
  }
return 0;
}
int main()
//введення кількості точок та ребер
{
 int vertexnumber;
 printf("Input number of vertexes\n");
 scanf("%d",&vertexnumber);
 int edgenumber;
 printf("Input number of edges\n");
 scanf("%d",&edgenumber);
//побудова графа
 struct A graph[edgenumber];
 for(int i=0; i<edgenumber; i++)
  {
  printf("Input your start vertex\n");
  scanf("%d",&graph[i].beginv);
  printf("Input your finish vertex\n");
  scanf("%d",&graph[i].finishv);
  printf("Input your weight\n");
  scanf("%d",&graph[i].w);
  printf("printf\n");
  }
 for(int i=0;i<edgenumber; i++)
  {
  printf("%d-(%d)->%d\n",graph[i].beginv,graph[i].w, graph[i].finishv);
  }
  //сортування ребер
 sorting(graph,0,edgenumber-1);
 printf("Sorted is coming\n" );
 for(int i=0;i<edgenumber;i++)
  {
  printf("%d-(%d)->%d\n",graph[i].beginv,graph[i].w, graph[i].finishv);
  }
 int fvertex[vertexnumber];
 int counter=0;
 fvertex[counter++]=1;
 int fedges=0;
 struct A bonesgraph[edgenumber];
 for(int i=0;i<vertexnumber;i++)
 {
  //перевірка вершин на включення в кістякове дерево
 for(int j=0;j<edgenumber;j++)
   {
   int a=func1(fvertex, counter+1, graph[j].beginv);
   int b=func1(fvertex, counter+1, graph[j].finishv);
   if(( a && !b)||(!a && b))
    {
    bonesgraph[fedges++]=graph[j];
    if(a)
     {
      fvertex[counter++]=graph[j].finishv;
     }
     else
     {
     fvertex[counter++]=graph[j].beginv ;
     }
    graph[j].beginv=0;
    graph[j].finishv=0;
    break;
    }
   }
 }
 printf("Your bones graph \n");
 int value=0;
 printf("Minimum spanning tree\n");
 for(int i=0;i<fedges;i++)
 {
  printf("%d-(%d)->%d\n",bonesgraph[i].beginv,bonesgraph[i].w,
  bonesgraph[i].finishv);
  value = value + bonesgraph[i].w;
 }
 //виведення ваги остового дерева
 printf("Minimum weight=%d\n",value);
 printf("\n");
 return 0;
}
// 11   18
/*
1 2 7
2 5 2
5 8 4
8 11 7
11 10 5
10 7 3
7 4 5
4 1 1
1 3 3
3 5 7
2 7 1
3 6 4
4 6 2
7 9 3
6 10 2
6 8 6
5 9 4
9 11 4
*/
```

Висновок:на лабараторній роботі я навчився будувати кістякові дерева методом Прима та Краскала,а також програмно реалізовувати вище згадані методи