Міністерство освіти і науки України

Національному університеті "Львівська Політехніка"

Кафедра систем штучного інтелекту

Лабораторна робота № 4

з дисципліни

Виконав:

студент групи КН-114

Сиротюк Владислав

Викладач:

Мельникова Н.І.

Львів - 2019р.

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.

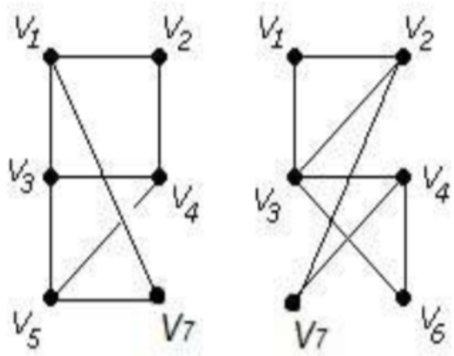
ІНДИВІДУАЛЬНІ ЗАВДАННЯ

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

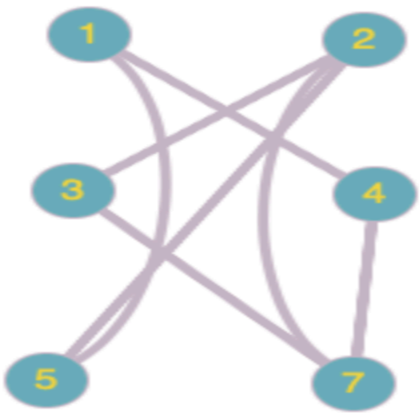
1. Виконати наступні операції над графами:

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.

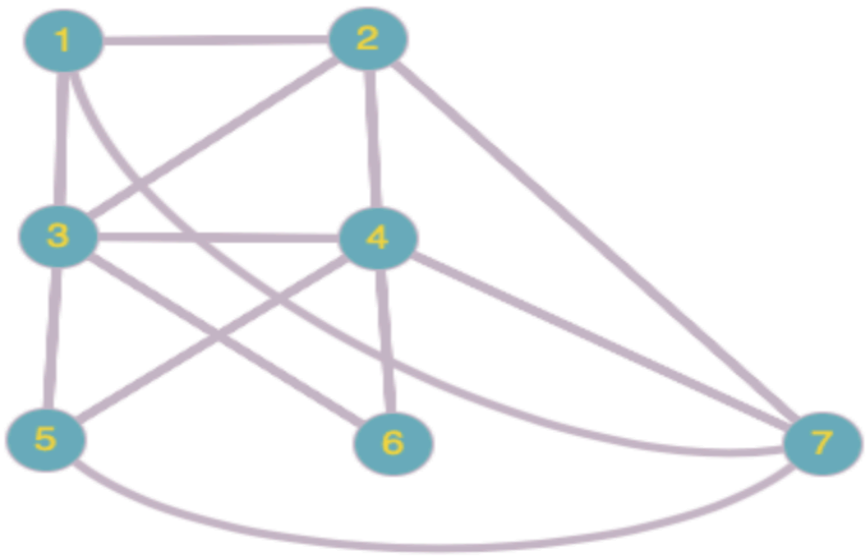
10



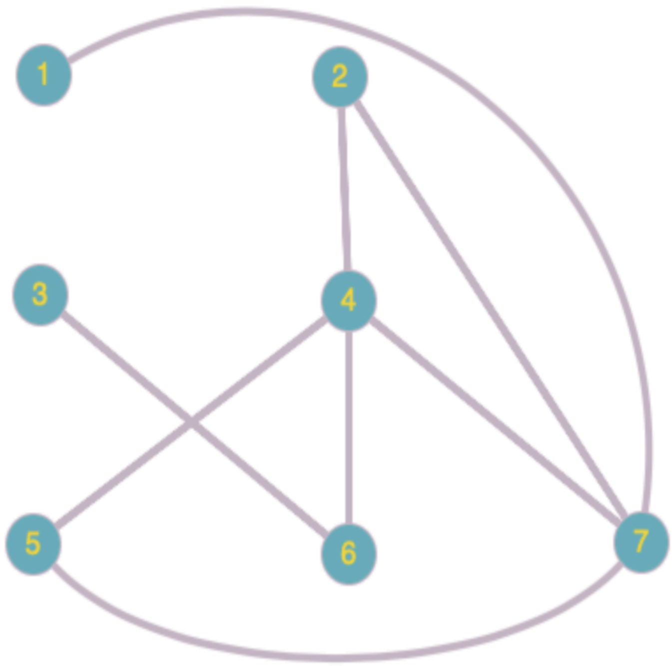
1)



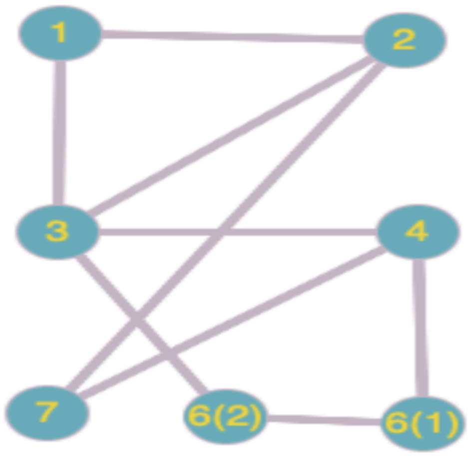
2)



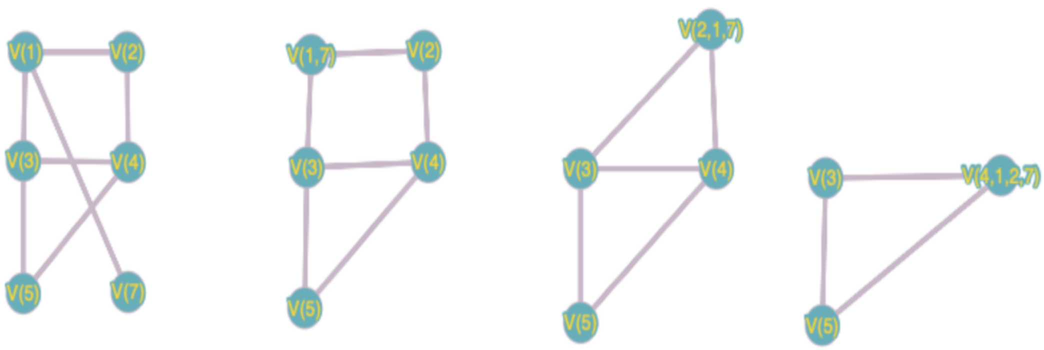
3)



4)



5)



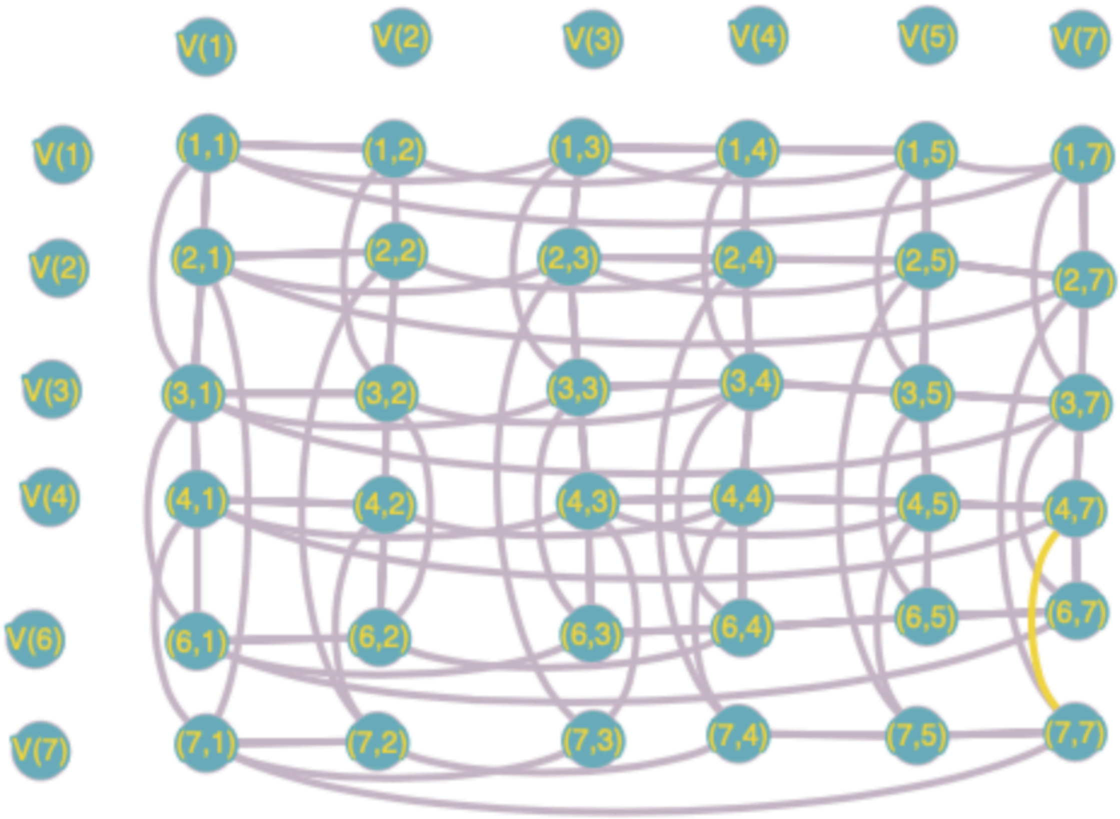
1)Стягуємо V(7)

2)Стягуємо V(1,7)

3)Стягуємо V(2,1,7)

4)Вийшов підграф А,що складається з 3-ох вершин.

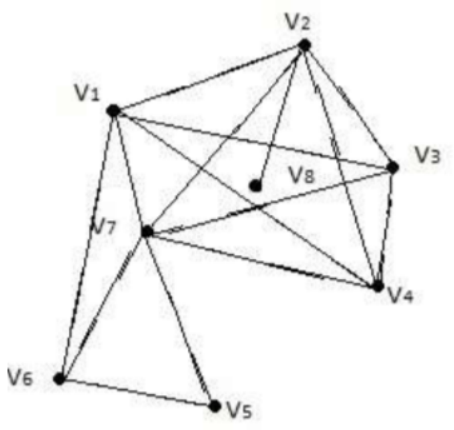
6)



6)

2. Знайти таблицю суміжності та діаметр графа.

10



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

Діаметр графа = 3

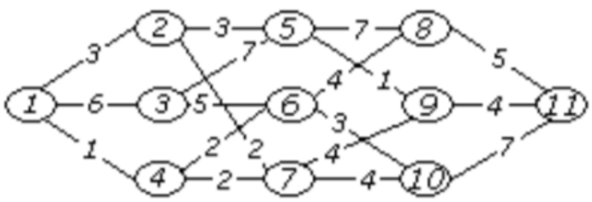
Відстань між V8 і V5 є найдовшою.

3)

3. Знайти двома методами (Краскала і Прима) мінімальне остове дерево графа.

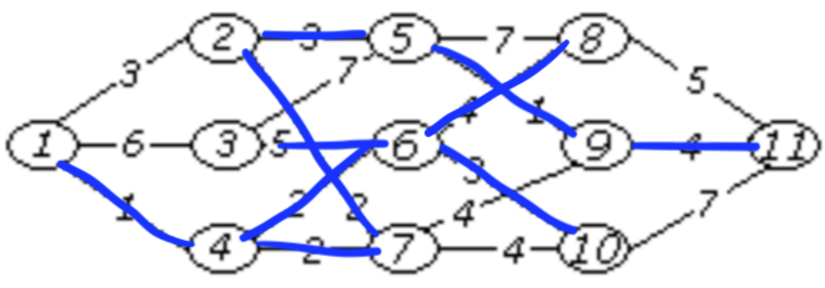
1

10



Алгоритм Краскала:

10



1)(1;4)-1;

(5;9)-1

2)(4;7)-2

(7;2)-2

(4;6)-2

3)(2;5)-3

(6;10)-3

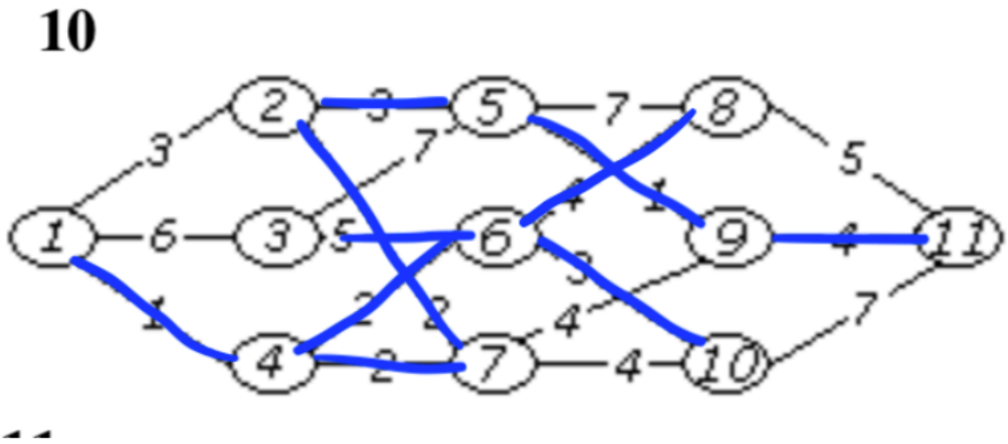
4)(9;11)-4

(6;8)-4

5)(3;6)-5

Weight = 27

Алгоритм Прима:



1)(1;4)-1;

2)(4;7)-2

3)(4;6) - 2

3)(7;2)-2

4)(2;5)-3

5)(5;9) - 1

6)(9;11)-4

(6;8)-4

5)(6;3)-5

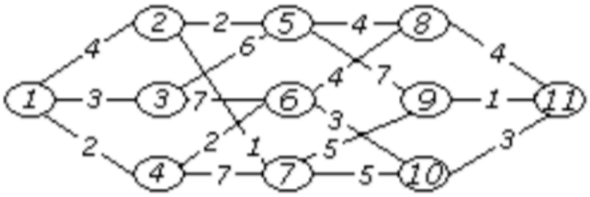
6)(6;10) - 3

Weight = 27

Завдання №2. Написати програму, яка реалізує алгоритм знаходження остового дерева мінімальної ваги згідно свого варіанту.

Варіант № 10

За алгоритмом Краскала знайти мінімальне остове дерево графа. Етапи розв'язання задачі виводити на екран. Протестувати розроблену програму на наступному графі:



```
#include <iostream>

using namespace std;

class Edge
{
public:
  int src, dest, weight;
};


class Graph
{
public:

  int V, E;

  Edge* edge;
};

Graph* createGraph(int V, int E)
{
  Graph* graph = new Graph;
  graph->V = V;
  graph->E = E;

  graph->edge = new Edge[E];

  return graph;
}

class subset
{
public:
  int parent;
  int rank;
};

int find(subset subsets[], int i)
```

```
{

  if (subsets[i].parent != i)
    subsets[i].parent = find(subsets, subsets[i].parent);

  return subsets[i].parent;
}

void Union(subset subsets[], int x, int y)
{
  int xroot = find(subsets, x);
  int yroot = find(subsets, y);


  if (subsets[xroot].rank < subsets[yroot].rank)
    subsets[xroot].parent = yroot;
  else if (subsets[xroot].rank > subsets[yroot].rank)
    subsets[yroot].parent = xroot;


  else
  {
    subsets[yroot].parent = xroot;
    subsets[xroot].rank++;
  }
}


int myComp(const void* a, const void* b)
{
  Edge* a1 = (Edge*)a;
  Edge* b1 = (Edge*)b;
  return a1->weight > b1->weight;
}

void KruskalMST(Graph* graph)
{
  int V = graph->V;
  Edge *result= new Edge[V];
  int e = 0;
  int i = 0;


  qsort(graph->edge, graph->E, sizeof(graph->edge[0]), myComp);


  subset* subsets = new subset[(V * sizeof(subset))];


  for (int v = 0; v < V; ++v)
  {
    subsets[v].parent = v;
    subsets[v].rank = 0;
  }
```

```
  while (e < V - 1 && i < graph->E)
  {

    Edge next_edge = graph->edge[i++];

    int x = find(subsets, next_edge.src);
    int y = find(subsets, next_edge.dest);



    if (x != y)
    {
      result[e++] = next_edge;
      Union(subsets, x, y);
    }

  }


  cout << "Following are the edges in the constructed MST\n";
  for (i = 0; i < e; ++i)
    cout << result[i].src << " - " << result[i].dest << " = " <<
result[i].weight << endl;
  return;
}

int main()
{

  int V, E;
  cout << "kil`kist vershyn: ";
  cin >> V;
  cout << "kil`kist reber: ";
  cin >> E;

  Graph* graph = createGraph(V, E);
  for (int index = 0; index < E;index++) {
    cout << "rebro[" << index << "]vershyna 1 =";
    cin>>graph->edge[index].src ;
    cout << "rebro[" << index << "]vershyna 2 =";
    cin>>graph->edge[index].dest;
    cout << "rebro[" << index << "]vaga =";
    cin>>graph->edge[index].weight;

  }


  KruskalMST(graph);

  return 0;
}
```

Висновок: на цій лабораторній роботі я набув практичних вмінь та навичок з використання алгоритмів Прима і Краскала.