МІНІСТЕРСТВО ОСВІТИ І НАУКИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Інститут комп'ютерних наук та інформаційних технологій

Кафедра систем штучного інтелекту



Лабораторна робота №3

з курсу “Дискретна математика ”

Виконав:  
 Збриський Костянтин

ст. гр. КН-110

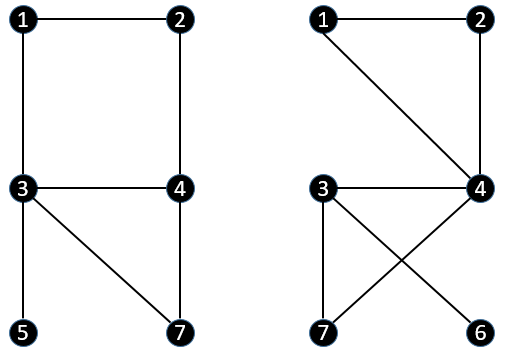
Викладач:

Мельникова Н.І.

Львів – 2018

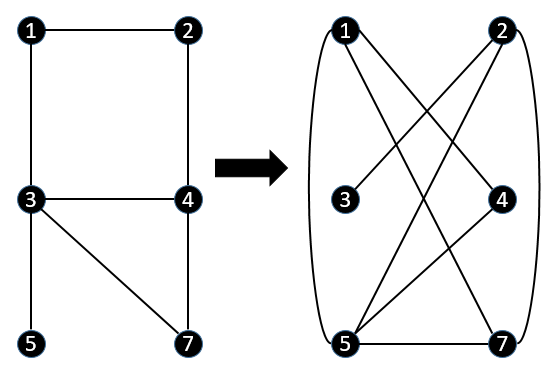
Тема: Основні операції над графами. Знаходження остова мінімальної ваги за алгоритмом Пріма-Краскала Мета роботи: набуття практичних вмінь та навичок з використання алгоритмів Пріма і Краскала.

1. Виконати наступні операції над графами

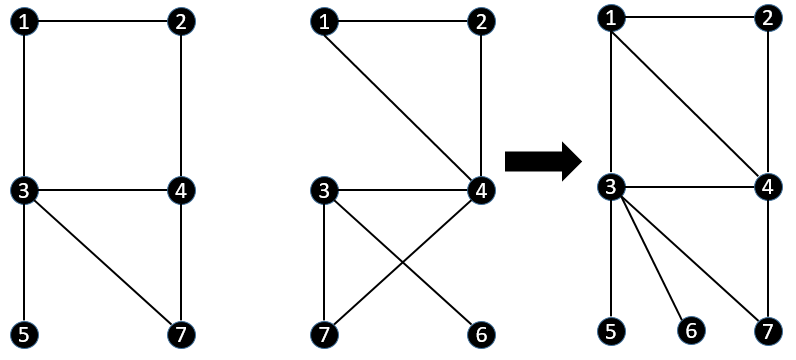


G1 G2

* 1. Знайти доповнення до першого графу

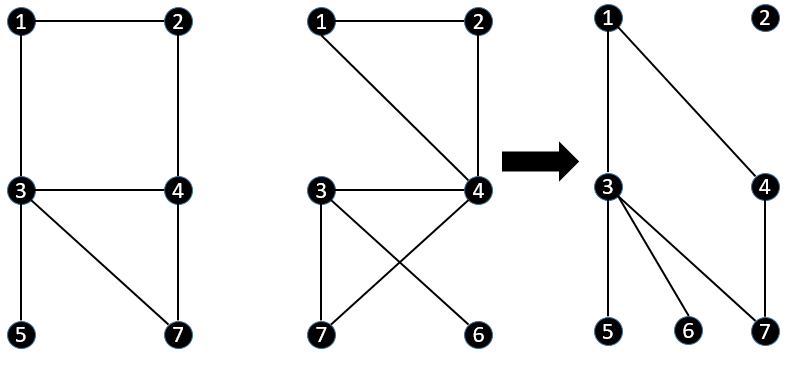


* 1. Знайти об’єднання графів G1 та G2



G1 G2

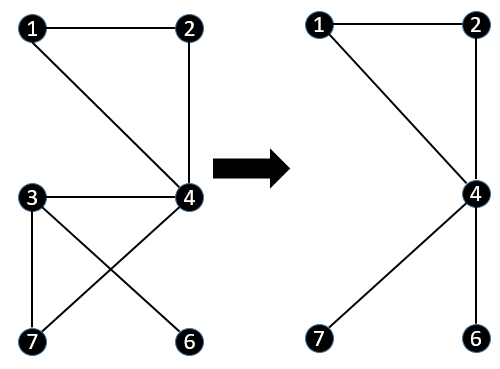
* 1. Знайти кільцеву суму G1 та G2



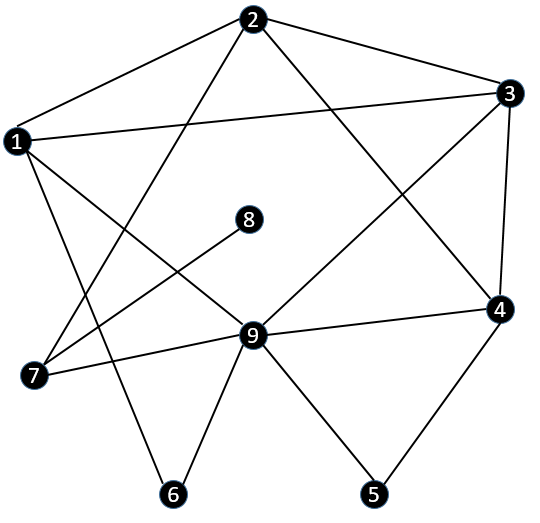
G1 G2

1.4 Розчепити вершину у G2

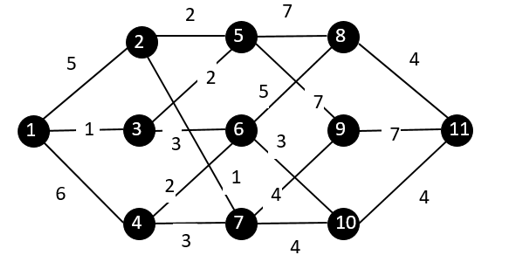
Знищуємо вершину 3, разом з вершиною зникають і всі інцидентні до неї ребра



2.0 Знайти таблицю суміжності , та діаметр графа



|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 1 | **1** | **1** | **1** | **0** | **0** | **1** | **0** | **0** | **1** |
| 2 | **1** | **0** | **1** | **1** | **0** | **0** | **1** | **0** | **0** |
| 3 | **1** | **1** | **0** | **1** | **0** | **0** | **0** | **0** | **1** |
| 4 | **0** | **1** | **1** | **0** | **1** | **0** | **0** | **0** | **1** |
| 5 | **0** | **0** | **0** | **1** | **0** | **0** | **0** | **0** | **1** |
| 6 | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** |
| 7 | **0** | **1** | **0** | **0** | **0** | **0** | **0** | **1** | **1** |
| 8 | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **0** | **0** |
| 9 | **1** | **0** | **1** | **1** | **1** | **1** | **1** | **0** | **0** |



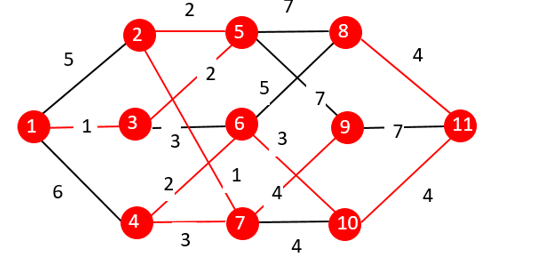
**(V1 V3) > (V3 V5) > (V5 V2) > (V2 V7) > (V7 V4) > (V4 V6) > (V6 V10) > (V10 V11) > (V11 V8) > (V7 V9)**

**для алгоритма Краскала:**

**(V1 V3) (V2 V7) (V3 V5) (V5 V2) (V4 V6) (V7 V4) (V6 V10) (V10 V11) (V11 V8) (V7 V9)**

**В результаті отримаємо таке остове дерево**

**Ціна - 22**



Висновок: Я навчився використовувати і програмно реалізовувати алгоритм Краска і Прими.

Код програми

#include <iostream>

typedef struct vertex vertex;

typedef struct edge edge;

typedef struct min min;

typedef struct min {

int min;

int vertex1, vertex2;

}min;

typedef struct vertex {

int vertex;

struct vertex \* next;

min min;

int in\_tree = 0;

edge \* head = nullptr;

}vertex;

typedef struct edge {

int vertex1, vertex2, price;

struct edge \* next;

}edge;

void add\_vertex(vertex \* & head, int vertex\_value) {

vertex \* new\_vertex = new(vertex);

new\_vertex->vertex = vertex\_value;

new\_vertex->min.min = 100;

if (!head) {

new\_vertex->next = NULL;

head = new\_vertex;

return;

}

vertex \* tracer = head;

while (tracer->next != NULL) {

tracer = tracer->next;

}

tracer->next = new\_vertex;

new\_vertex->next = NULL;

}

void find\_min\_l(vertex \* & element) {

edge \* tracer = element->head;

element->min.min = 100;

while (tracer != NULL) {

if (tracer->price < element->min.min) {

element->min.min = tracer->price;

element->min.vertex1 = tracer->vertex1;

element->min.vertex2 = tracer->vertex2;

}

tracer = tracer->next;

}

}

void delete\_edge(vertex \* & head, int vertex1, int vertex2) {

vertex \* tracer = head;

while (tracer != NULL) {

if ((tracer->vertex == vertex1) || (tracer->vertex == vertex2)) {

edge \* tracer\_edge = tracer->head;

edge \* prev = tracer\_edge;

if ((tracer\_edge->vertex1 == vertex1 || tracer\_edge->vertex1 == vertex2) && (tracer\_edge->vertex2 == vertex1 || tracer\_edge->vertex2 == vertex2)) {

edge \* tmp = tracer\_edge;

tracer->head = (tracer->head)->next;

delete tmp;

tracer\_edge = head->head;

//if ((tracer\_edge->vertex1 < -1) && (tracer\_edge->next == NULL)) {

// tracer\_edge = NULL;

//}

}

while (tracer\_edge != NULL) {

if ((tracer\_edge->vertex1 == vertex1 || tracer\_edge->vertex1 == vertex2) && (tracer\_edge->vertex2 == vertex1 || tracer\_edge->vertex2 == vertex2)) {

prev->next = tracer\_edge->next;

delete tracer\_edge;

tracer\_edge = prev->next;

continue;

}

prev = tracer\_edge;

tracer\_edge = tracer\_edge->next;

}

}

find\_min\_l(tracer);

tracer = tracer->next;

}

}

// you created new struct min to get data about vertixe that minimum effect, to use func in tree to finish algo becouse it was skiping som elements

void find\_min\_tree(vertex \* head, min \* & minimal) {

minimal->min = 100;

vertex \* tracer = head;

while (tracer != NULL) {

if (tracer->in\_tree) {

if (tracer->min.min < minimal->min) {

minimal->min = tracer->min.min;

minimal->vertex1 = tracer->min.vertex1;

minimal->vertex2 = tracer->min.vertex2;

}

}

tracer = tracer->next;

}

}

void add\_edge\_body(vertex \* & vertex\_node, int vertex1, int vertex2, int price) {

edge \* new\_edge = new(edge);

new\_edge->vertex1 = vertex1;

new\_edge->vertex2 = vertex2;

new\_edge->price = price;

if (!vertex\_node->head) {

new\_edge->next = NULL;

vertex\_node->head = new\_edge;

return;

}

edge \* tracer = vertex\_node->head;

while (tracer->next != NULL) {

tracer = tracer->next;

}

tracer->next = new\_edge;

new\_edge->next = NULL;

}

void add\_edge(vertex \* head, int vertex\_value1, int vertex\_value2, int price) {

vertex \* tracer = head;

while (tracer != NULL) {

if ((tracer->vertex == vertex\_value1) || (tracer->vertex == vertex\_value2)) {

add\_edge\_body(tracer, vertex\_value1, vertex\_value2, price);

find\_min\_l(tracer);

}

tracer = tracer->next;

}

}

void printe(vertex \* head) {

vertex \* tracer = head;

while (tracer != NULL) {

std::cout << "This is vetex " << tracer->vertex << " this is min " << tracer->min.min << std::endl;

edge \* tracer\_edge = tracer->head;

while (tracer\_edge != NULL) {

std::cout << "This is vertexes " << tracer\_edge->vertex1 << ' ' << tracer\_edge->vertex2 << ' ' << "this is price " << tracer\_edge->price << std::endl;

tracer\_edge = tracer\_edge->next;

}

tracer = tracer->next;

}

}

void add\_to\_tree(vertex \* & head, int vertex1, int vertex2) {

vertex \* tracer = head;

while (tracer != NULL) {

if (tracer->vertex == vertex1 || tracer->vertex == vertex2) {

tracer->in\_tree = 1;

}

tracer = tracer->next;

}

}

int if\_in\_tree(vertex \* head, int vertex\_value) {

vertex \* tracer = head;

while (tracer != NULL) {

if (tracer->in\_tree) {

if (tracer->vertex == vertex\_value) {

return 1;

}

}

tracer = tracer->next;

}

return 0;

}

void prima\_algo(vertex \* & head) {

int price\_forest = 0;

vertex \* tracer = head;

tracer->in\_tree = 1;

int i = 0;

while ((tracer != NULL) && (i != 11)) {

int add = 0;

min \* minimal = new(min);

find\_min\_tree(head, minimal);

if (i != 0) {

if ( (if\_in\_tree(head, minimal->vertex1)) && (if\_in\_tree(head, minimal->vertex2)) ) {

delete\_edge(head, minimal->vertex1, minimal->vertex2);

}

}

if (tracer->in\_tree) {

edge \* tracer\_edge = tracer->head;

while (tracer\_edge != NULL) {

if (tracer\_edge->price == minimal->min) {

std::cout << "vertex " << tracer\_edge->vertex1 << ' ' << tracer\_edge->vertex2 << ' ' << "price is " << tracer\_edge->price << std::endl;

add\_to\_tree(head, tracer\_edge->vertex1, tracer\_edge->vertex2);

price\_forest += tracer\_edge->price;

delete\_edge(head, tracer\_edge->vertex1, tracer\_edge->vertex2);

tracer = head;

add = 1;

i++;

break;

}

tracer\_edge = tracer\_edge->next;

}

}

if (add) continue;

tracer = tracer->next;

}

std::cout << "price of the forest is " << price\_forest << std::endl;

}

int main() {

vertex \* head = nullptr;

add\_vertex(head, 1);

add\_vertex(head, 2);

add\_vertex(head, 3);

add\_vertex(head, 4);

add\_vertex(head, 5);

add\_vertex(head, 6);

add\_vertex(head, 7);

add\_vertex(head, 8);

add\_vertex(head, 9);

add\_vertex(head, 10);

add\_vertex(head, 11);

// creating edges

add\_edge(head, 1, 4, 3);

add\_edge(head, 1, 3, 2);

add\_edge(head, 1, 2, 1);

add\_edge(head, 2, 5, 3);

add\_edge(head, 2, 7, 4);

add\_edge(head, 3, 5, 7);

add\_edge(head, 3, 6, 6);

add\_edge(head, 4, 6, 2);

add\_edge(head, 4, 7, 4);

add\_edge(head, 5, 8, 7);

add\_edge(head, 5, 9, 5);

add\_edge(head, 6, 8, 7);

add\_edge(head, 6, 10, 3);

add\_edge(head, 7, 9, 5);

add\_edge(head, 7, 10, 4);

add\_edge(head, 8, 11, 4);

add\_edge(head, 9, 11, 1);

add\_edge(head, 10, 11, 2);

printe(head);

std::cout << "--------------------------------" << std::endl;

printe(head);

prima\_algo(head);

system("pause");

}