МІНІСТЕРСТВО ОСВІТИ І НАУКИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Інститут комп'ютерних наук та інформаційних технологій

Кафедра систем штучного інтелекту



Лабораторна робота №5

з курсу “Дискретна математика ”

Виконав:  
 Збриський Костянтин

ст. гр. КН-110

Викладач:

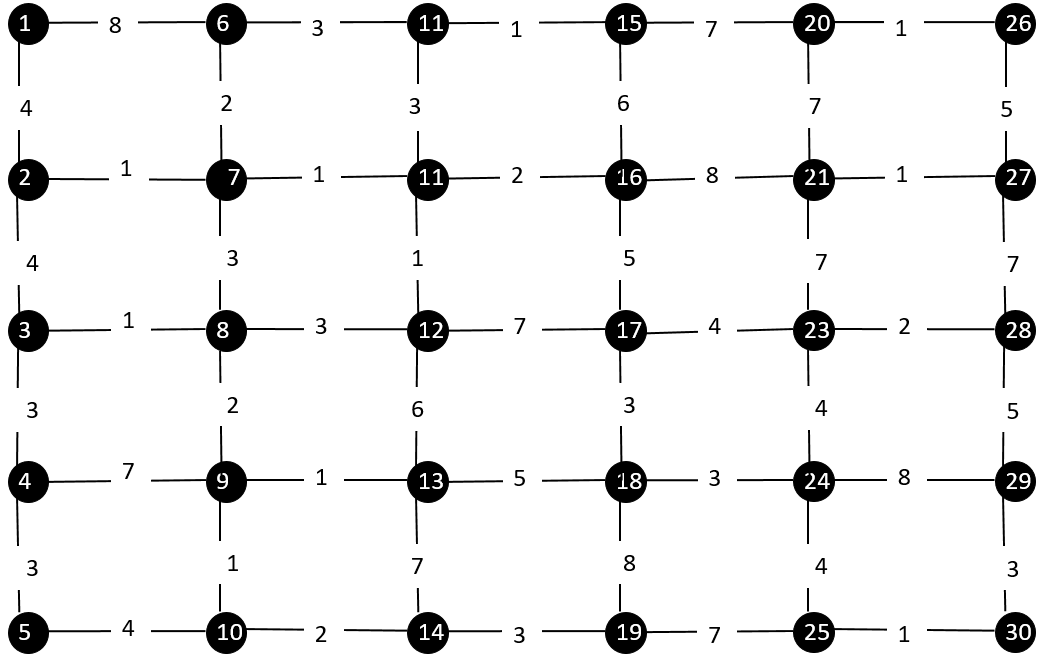
Мельникова Н.І.

Львів – 2018

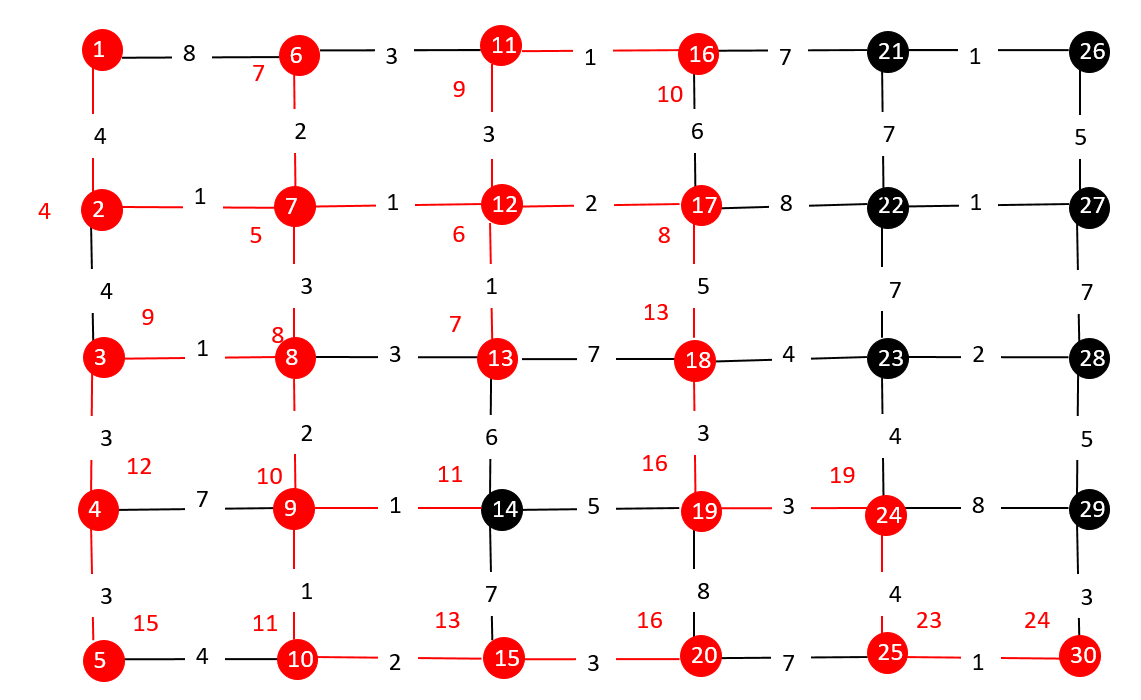
Варіант 13

Мета роботи: набуття практичних вмінь та навичок з використання алгоритму Дейкстри

1.



Результатом роботи буде такий граф



Мінімальний шлях від (V1 > V30) V1 > V2 > V7 > V12 > V17 > V18 > V19 > V24 > V25 > V30

Висновок: Я набув вміння роботи і програмної реалізації алгоритма Дейкстри.

Код програми

#include <iostream>

typedef struct vertex vertex;

typedef struct edge edge;

typedef struct min min;

typedef struct min {

int min;

int vertex1, vertex2;

vertex \* to;

}min;

typedef struct vertex {

int vertex;

struct vertex \* next;

min min;

int in\_tree = 0;

int way = 1000;

int way\_vertex1;

int way\_vertex2;

struct vertex \* way\_to;

int annon = 0;

edge \* head = nullptr;

}vertex;

typedef struct edge {

int vertex1, vertex2, price;

struct edge \* next;

vertex \* to;

}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;

}

// mb bug here

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;

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

}

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;

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

}

}

tracer = tracer->next;

}

}

vertex \* search(vertex \* head, int vert\_val) {

vertex \* tracer = head;

while (tracer != NULL) {

if (tracer->vertex == vert\_val) {

return tracer;

}

tracer = tracer->next;

}

return NULL;

}

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

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

new\_edge->vertex1 = vertex\_node->vertex;

int vert = 1;

if (vertex\_node->vertex != vertex1) vert = vertex1;

else if (vertex\_node->vertex != vertex2) vert = vertex2;

new\_edge->vertex2 = vert;

new\_edge->price = price;

// 111

new\_edge->to = search(head, new\_edge->vertex2);

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(head, 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 find\_min\_way(vertex \* head, vertex \* & min\_node) {

int min\_com = 1000;

vertex \* min\_way\_to = head;

vertex \* tracer = head;

while (tracer != NULL) {

if (tracer->in\_tree) {

if (min\_com > tracer->way && (tracer->way > 0)) {

min\_com = tracer->way;

min\_way\_to = tracer;

}

}

tracer = tracer->next;

}

min\_node = min\_way\_to;

}

int find\_min\_way(vertex \* head) {

int min\_way = 1000;

vertex \* tracer = head;

while(tracer != NULL) {

if ( (tracer->way < 1000) && (tracer->way > 0)) {

if (tracer->way < min\_way) {

min\_way = tracer->way;

}

}

tracer = tracer->next;

}

return min\_way;

}

void update\_neib(vertex \* node) {

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

int min\_way\_neib = 1000;

while (tracer\_edge != NULL) {

if (tracer\_edge->to->way > (tracer\_edge->price + node->way)) {

tracer\_edge->to->way = tracer\_edge->price + node->way;

if (tracer\_edge->to->way < min\_way\_neib) {

node->way\_vertex1 = tracer\_edge->vertex1;

node->way\_vertex2 = tracer\_edge->vertex2;

node->way\_to = tracer\_edge->to;

min\_way\_neib = tracer\_edge->to->way;

}

}

tracer\_edge = tracer\_edge->next;

}

}

void update\_own(vertex \* node) {

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

find\_min\_l(node);

node->way = node->min.min;

node->way\_vertex1 = node->min.vertex1;

node->way\_vertex2 = node->min.vertex2;

node->way\_to = node->min.to;

}

void clean\_visited(vertex \* node, vertex \* & head) {

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

while (tracer\_edge != NULL) {

if (tracer\_edge->to->in\_tree) {

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

}

tracer\_edge = tracer\_edge->next;

}

}

void dijkstra\_algo(vertex \* & head, int goal) {

vertex \* last = head;

last->way = 0;

last->in\_tree = 1;

min \* edge\_to\_go = new(min);

while (last->vertex != goal) {

update\_neib(last);

find\_min\_l(last);

find\_min\_tree(head, edge\_to\_go);

last = edge\_to\_go->to;

delete\_edge(head, edge\_to\_go->vertex1, edge\_to\_go->vertex2);

if (!last->annon) {

std::cout << "moved to " << last->vertex << " from " <<edge\_to\_go->vertex1 << std::endl;

}

last->annon = 1;

last->in\_tree = 1;

}

}

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);

add\_vertex(head, 12);

add\_vertex(head, 13);

add\_vertex(head, 14);

add\_vertex(head, 15);

add\_vertex(head, 16);

add\_vertex(head, 17);

add\_vertex(head, 18);

add\_vertex(head, 19);

add\_vertex(head, 20);

add\_vertex(head, 21);

add\_vertex(head, 22);

add\_vertex(head, 23);

add\_vertex(head, 24);

add\_vertex(head, 25);

add\_vertex(head, 26);

add\_vertex(head, 27);

add\_vertex(head, 28);

add\_vertex(head, 29);

add\_vertex(head, 30);

add\_edge(head, 1, 2, 6);

add\_edge(head, 1, 6, 4);

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

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

add\_edge(head, 3, 4, 5);

add\_edge(head, 3, 8, 1);

add\_edge(head, 4, 5, 3);

add\_edge(head, 4, 9, 7);

add\_edge(head, 5, 10, 4);

add\_edge(head, 6, 11, 1);

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

add\_edge(head, 7, 12, 1);

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

add\_edge(head, 8, 13, 3);

add\_edge(head, 8, 9, 2);

add\_edge(head, 9, 14, 3);

add\_edge(head, 9, 10, 2);

add\_edge(head, 10, 15, 7);

add\_edge(head, 11, 16, 1);

add\_edge(head, 11, 12, 3);

add\_edge(head, 12, 17, 4);

add\_edge(head, 12, 13, 1);

add\_edge(head, 13, 18, 7);

add\_edge(head, 13, 14, 7);

add\_edge(head, 14, 19, 1);

add\_edge(head, 14, 15, 3);

add\_edge(head, 15, 20, 3);

add\_edge(head, 16, 21, 3);

add\_edge(head, 16, 17, 1);

add\_edge(head, 17, 22, 2);

add\_edge(head, 17, 18, 4);

add\_edge(head, 18, 23, 5);

add\_edge(head, 18, 19, 8);

add\_edge(head, 19, 24, 1);

add\_edge(head, 19, 20, 8);

add\_edge(head, 20, 25, 3);

add\_edge(head, 21, 26, 5);

add\_edge(head, 21, 22, 3);

add\_edge(head, 22, 27, 4);

add\_edge(head, 22, 23, 1);

add\_edge(head, 23, 28, 2);

add\_edge(head, 23, 24, 4);

add\_edge(head, 24, 29, 8);

add\_edge(head, 24, 25, 1);

add\_edge(head, 25, 30, 6);

add\_edge(head, 26, 27, 7);

add\_edge(head, 27, 28, 7);

add\_edge(head, 28, 29, 5);

add\_edge(head, 29, 30, 7);

printe(head);

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

dijkstra\_algo(head, 5);

system("pause");

}