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: ст. гр. КН-113
Іванюшенко Нестор

Викладач: Мельникова Н.І.

[варіант 2]

Знаходження найкоротшого маршруту за алгоритмом Дейкстри. Плоскі планарні графи

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

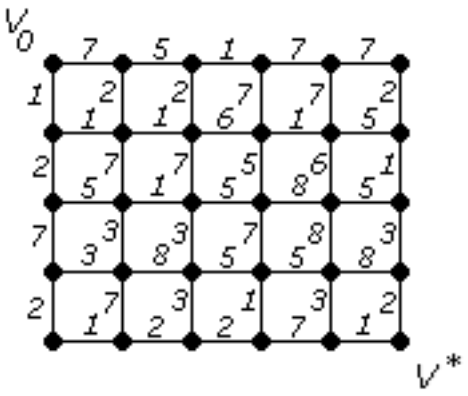
Теоретичні відомості

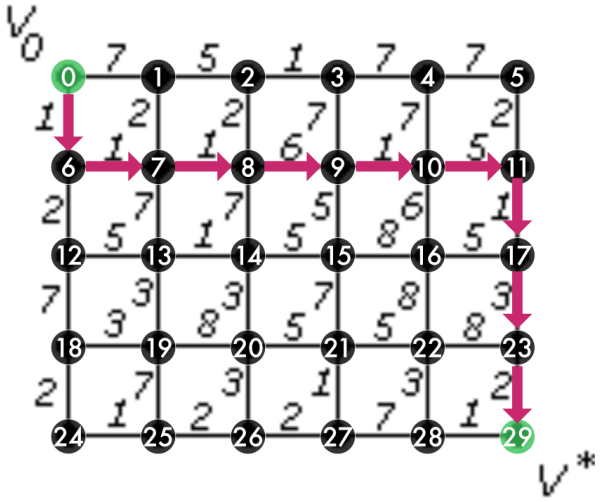
Задача знаходження найкоротшого шляху з одним джерелом полягає у знаходженні найкоротших(мається на увазі найоптимальніших за вагою) шляхів від деякої вершини(джерела) до всіх вершин графа G . Для розв'язку цієї задачі використовується «жадібний» алгоритм, який називається алгоритмом Дейкстри.

Плоским графом називається граф, вершини якого є точками площини, а ребра – безперервними лініями без самоперетинань, що з'єднують відповідні вершини так, що ніякі два ребра не мають спільних точок крім інцидентної їм обом вершини. Граф називається планарним, якщо він є ізоморфним плоскому графу.

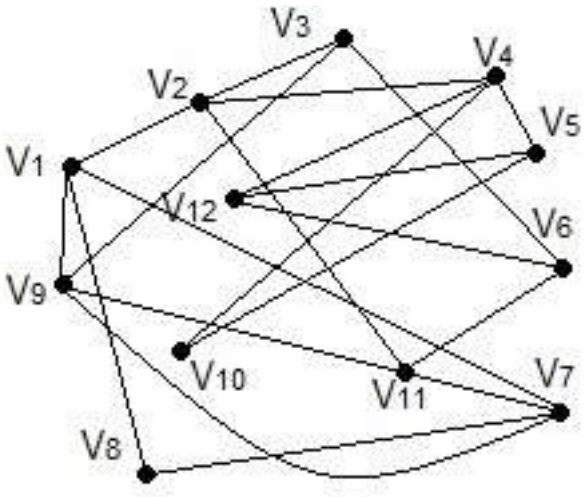
Завдання 1

1. За допомогою алгоритму Дейкстра знайти найкоротший шлях у графі поміж парою вершин V_0 і V^*

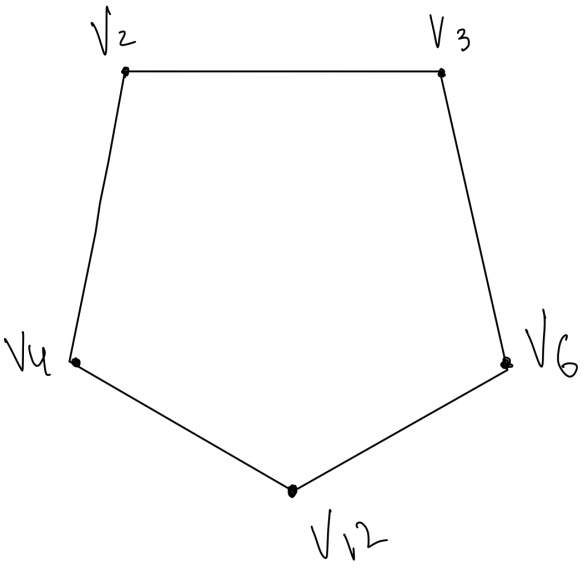




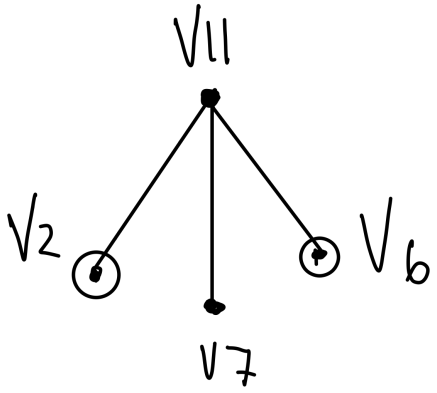
2. За допомогою γ - алгоритма зробити укладку графа у площині, або довести що вона неможлива.



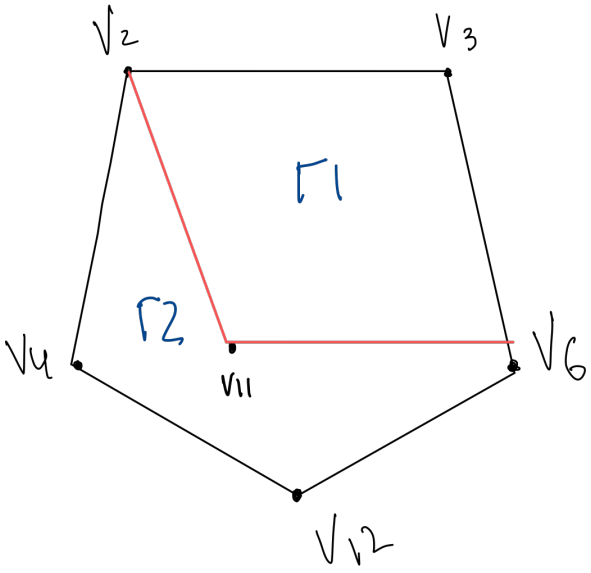
спочатку виділимо цикл:



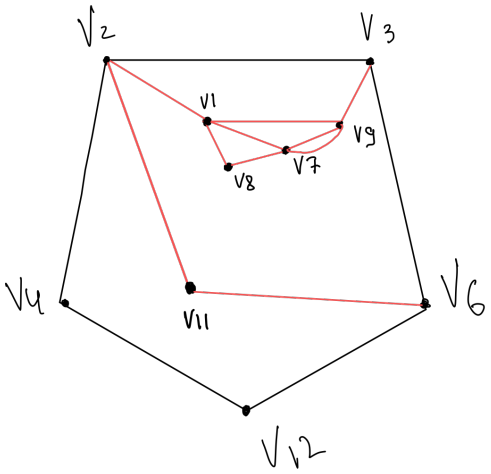
візьмемо сегмент який який не належить циклу

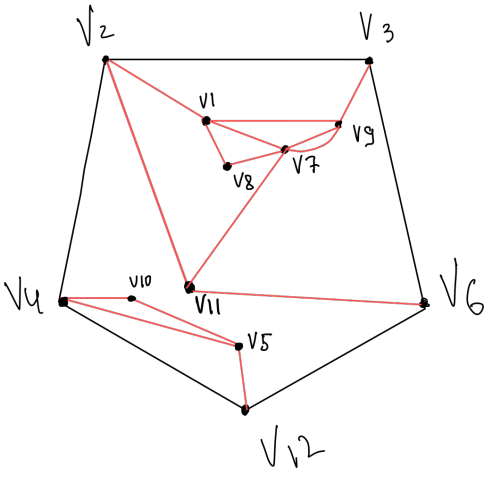


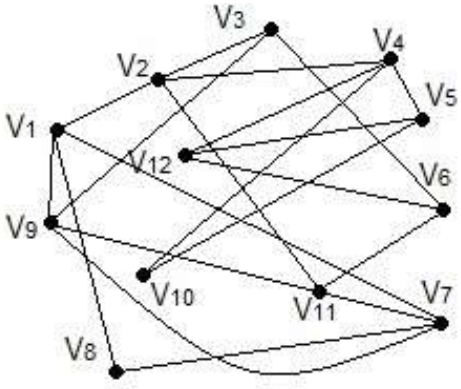
і впишемо його, розділяючи граф на 2 грані



повторимо доки будуть вільні сегменти:

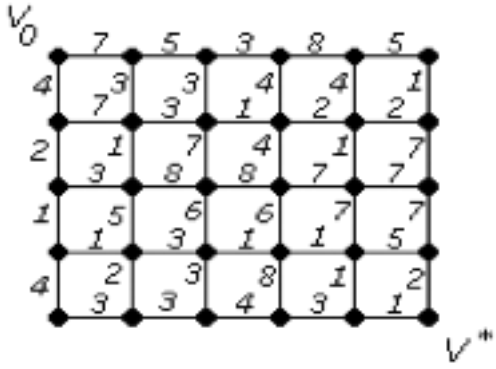






Завдання 2

Написати програму, яка реалізує алгоритм Дейкстри знаходження найкоротшого шляху між парою вершин у графі. Протестувати розроблену програму на графі згідно свого варіанту.



програма:
структура ребро:

```
struct Edge{
    int from;
    int to;
    int weight;
    Edge(int from_, int to_, int weight_){
        from = from_;
        to = to_;
        weight = weight_;
    }
    void print(){
        std::cout << "Edge from " << from << "; to " << to <<
        "; weight: " << weight << std::endl;
    }
};
```

структура сітки (так як граф - сітка я зробив структуру сітки щоб простіше задавати граф):

```
std::vector<Edge> edges;
struct Grid{
    int width;
    int height;
    Grid(int w, int h) : width(w), height(h){}
    void insert_horizontally(int row, int a,int b,int c, int d, int e){
        int weights[] = {a,b,c,d,e};
        for(int i = 0; i < width-1; i++){
            edges.push_back(Edge(i + width*row,
                    i+1 + width*row, weights[i]));
        }
    }
    void insert_vertically(int column, int a,int b,int c, int d){
        int weights[] = {a,b,c,d};
        for(int i = 0; i < height-1; i++){
            edges.push_back(Edge(column + i*width,
                    column + width*(i + 1), weights[i]));
        }
    }
};
```

функція яка знаходить інцидентні ребра до вказаної вершини:

```
std::vector<Edge> find_incident(int vertex, std::vector<Edge> edges){
    std::vector<Edge> incident;
    for(Edge e:edges){
        if(e.from == vertex //|| e.to == vertex
         ){
            incident.push_back(e);
        }
    }
    return incident;
}
```

задаємо граф:

```
int main() {
    Grid grid(6, 5);
    grid.insert_horizontally(0, 7, 5, 3, 8, 5);
    grid.insert_horizontally(1, 7, 3, 1, 2, 2);
    grid.insert_horizontally(2, 3, 8, 8, 7, 7);
    grid.insert_horizontally(3, 1, 3, 1, 1, 5);
    grid.insert_horizontally(4, 3, 3, 4, 3, 1);

    grid.insert_vertically(0, 4, 2, 1, 4);
    grid.insert_vertically(1, 3, 1, 5, 2);
    grid.insert_vertically(2, 3, 7, 6, 3);
    grid.insert_vertically(3, 4, 4, 6, 8);
    grid.insert_vertically(4, 4, 1, 7, 1);
    grid.insert_vertically(5, 1, 7, 7, 2);
```

масиви з вершинами, відстанями і попередніми вершинами:

```
    for(Edge e:edges){
        e.print();
    }
    std::cout << "\n--------------\n";
    int vert_num = 30;
    int vertices[vert_num];
    int dist[vert_num];
    int prev[vert_num];
    for(int i = 0; i < vert_num; i++){
        vertices[i] = i;
        dist[i] = 100000;
        prev[i] = -1;
    }
    prev[0] = 0;
    dist[0] = 0;
```

для всіх вершин, для всіх ребер які виходять з вершини порівнюєм відстань до інших вершин через інакше ребро

```
for(int v:vertices){
    std::vector<Edge> incident = find_incident(v, edges);
    std::cout << "Vertex: " << v << std::endl;
    for(Edge e:incident){
        e.print();
        int current = e.to;
        int previous = v;
        if(dist[current] > dist[previous] + e.weight){
            dist[current] = dist[previous] + e.weight;
            prev[current] = previous;
        }
    }
    std::cout << "\n";
}
```

обчислюєм шлях до певної вершини виходячи з масиву попередніх вершин

```
// distance calculation:
    for(int d:dist){
        std::cout << d << ",  ";
    }
    std::cout << std::endl;
    for(int p:prev){
        std::cout << p << ",  ";
    }
    std::cout << std::endl;
    // path_deconstruction
    int *path = new int[1];
    int DESTINATION = vert_num-1;
    path[0] = DESTINATION;
    size_t pathlen = 1;
    while(1){
        path = (int*)realloc(path, ++pathlen);

        path[pathlen-1] = prev[path[pathlen-2]];
        if(path[pathlen-1] == 0){
            break;
        }
    }

    std::cout << "\npath to destination vertex:\n";
    for(int i = 0; i<pathlen; i++){
        std::cout << path[pathlen-i-1] << ", ";
    }
```

обчислюємо вагу шляху

```
    // calculating path weight:
    int pathweight = 0;

    for(int i = 0; i < pathlen ; i++){
        for(Edge e: edges){
            if(e.from == path[pathlen - i] && e.to == path[pathlen - (i+1)]){
                pathweight += e.weight;
                break;
            }
        }
    }

    std::cout << "\n\npath weight: " << pathweight << std::endl;
    return 0;
}
```

вивід:
масив ваг, масив попередніх вершин, сам шлях і вага шляху до певної вершини

```
0,  7,  12,  15,  23,  28,  4,  10,  13,  14,  16,  18,  6,  9,  17,  18,  17,  24,  7,  8,  11,  12,  13,  18,  11,  10,  13,  17,  14,  15,
0,  0,  1,  2,  3,  4,  0,  1,  7,  8,  9,  10,  6,  12,  13,  9,  10,  16,  12,  18,  19,  20,  21,  22,  18,  19,  25,  26,  22,  28,

path to destination vertex:
0, 6, 12, 18, 19, 20, 21, 22, 28, 29,

path weight: 15
```

Висновок:

я набув практичних вмінь та навичок з використання алгоритму Дейкстри.