УКРАЇНСЬКИЙ КАТОЛИЦЬКИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНИХ НАУК
Комп’ютерні науки

ПОШУК НАЙКОРОТШОГО ШЛЯХУ МІЖ ДВОМА ТОЧКАМИ ПОВЕРХНІ

Автори:
Пелех Богдан
Казимир Арсеній
Маркіян Круглій
Барановська Тетяна
Шаманський Кирило
(команда 9)

16 грудня 2021

ВСТУП

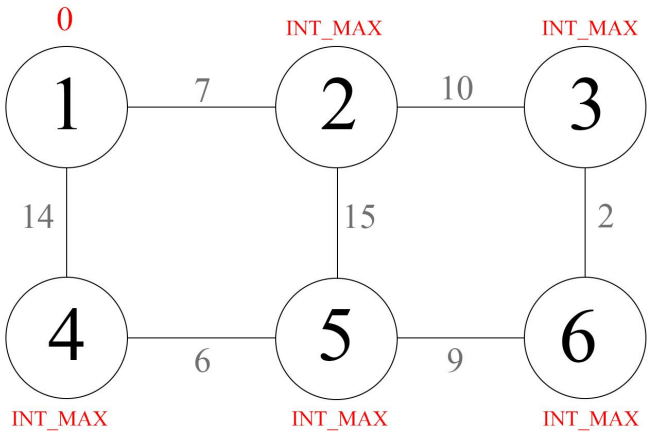
На сьогодні GPS та інші види навігації здобули широку славу, оскільки вирішують значні проблеми, допомагають прокласти найкоротший маршрут між двома пунктами. Даний факт надихнув нас створити власний програмний продукт, який знаходить найкоротший шлях між двома точками поверхні. Для цього ми використали алгоритм Дейкстри з додатковою функцією priority_queue, яка створює чергу з пріоритетом для збереження локальних точок. Для обходу графів він використовує пошук вшир, який вивчався нами на заняттях з дискретної математики окремо від даного алгоритму. Зберігати вершини будемо у матриці де кожен елемент є висотою вершини, які в свою чергу можуть нагадати знайомі відстані між перехрестями.

Алгоритм Дейкстри призначений для вирішення задачі пошуку найкоротшого шляху на графі. Для заданого зваженого графа з невід'ємною вагою алгоритм знаходить найкоротші відстані від виділеної вершини-джерела до всіх інших вершин графа. Алгоритм Дейкстри виконується за час O(m+nlnn) і є асимптотично найшвидшим із відомих послідовних алгоритмів для даного класу задач.

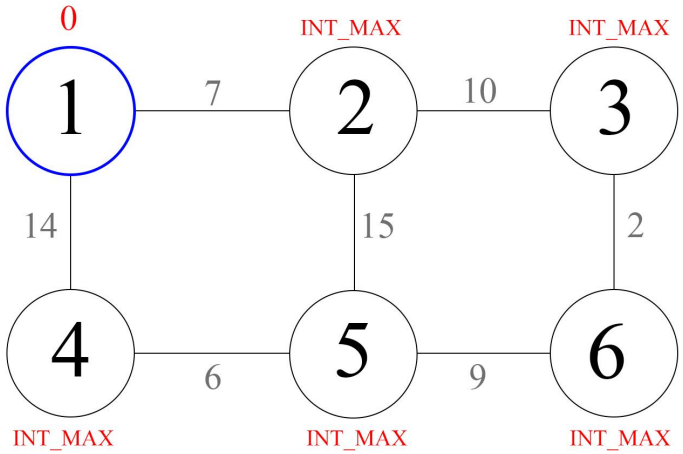
Тобто створивши програмний продукт ми будемо в змозі швидко визначати найкоротший маршрут між двома точками.

Покроковий опис запропонованого алгоритму

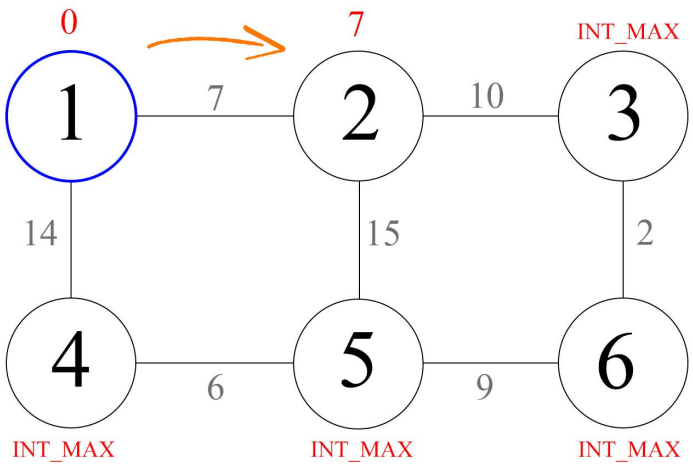
Крок 1: відстань до всіх вершин графу **:=** INT_MAX (дуже велике число). Відстань до **a** = 0 (точка з якої ми починаємо пошук; точка з індексом 1 на рисунку). Жодної вершини графу ще не опрацьовано. Тобто на даному етапі ми вказуємо стартову вершину та присвоюємо їй відстань 0. А також заносимо її у нашу чергу.



Крок 2: заходимо в поточну чергу з пріоритетами і беремо вершину з найменшим шляхом до неї. В нашому випадку це єдина вершина 1.

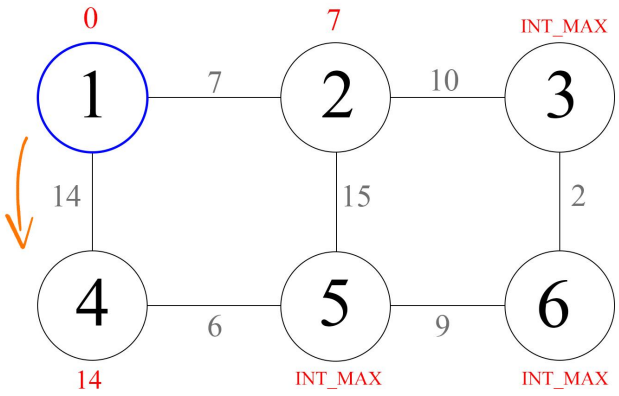


Крок 3: перший по порядку сусід 1-ї вершини — 2-а вершина. Шлях до неї через 1-у вершину дорівнює найкоротшій відстані до 1-ї вершини + довжина ребра між 1-ю та 2-ю вершиною, тобто 0 + 7 = 7. Це менше поточного найкоротшого шляху до 2-ї вершини(INT_MAX), тому найкоротший шлях до 2-ї вершини дорівнює 7. Позначаємо його як 7 і заносимо вершину у нашу чергу.

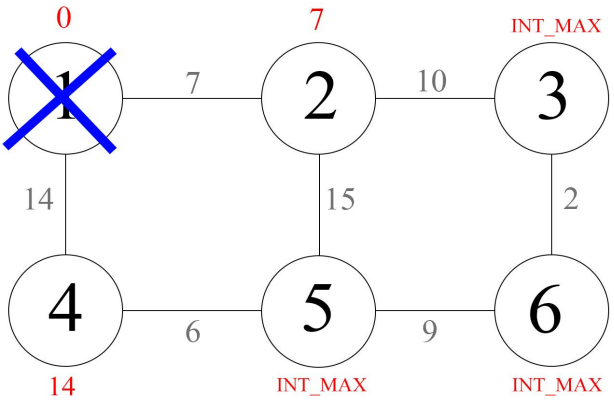


ПРИМІТКА: Довжину ребер попередньо визначаємо як корінь квадратний із суми квадрату різниць висот двох сусідніх вершини графу та квадрату горизонтальної відстані між вершинами–step

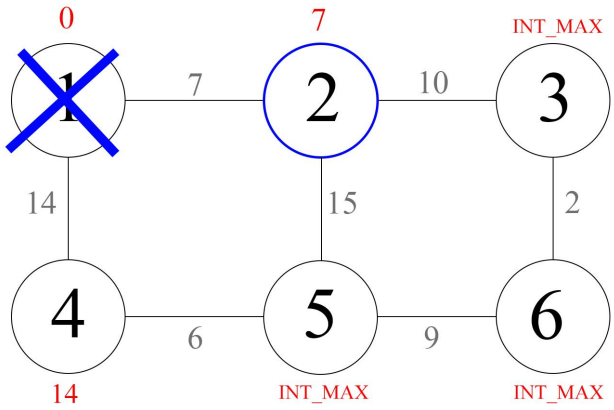
Крок 4: аналогічну операцію виконуємо з іншим сусідом 1-ї вершини — 4-ю вершиною, поточний найкоротший шлях до якої також становить (INT_MAX).



Крок 5: всі сусіди вершини 1 перевірені. Поточна мінімальна відстань до вершини 1 вважається остаточною. Надалі ми більше не будемо додавати цю вершину у чергу знову.



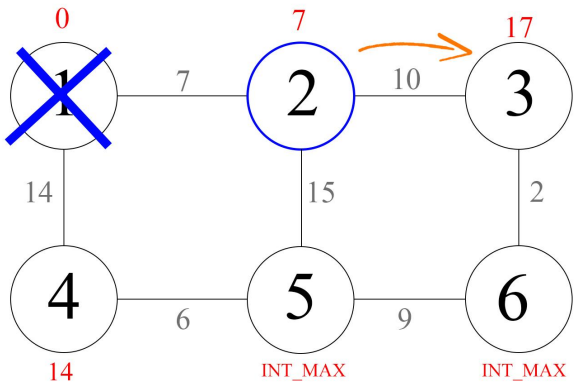
Крок 6: практично відбувається повернення до кроку 2. Знаходимо у черзі «найближчу» за відстанню необроблену вершину. Це вершина 2 з поточною найкоротшою відстанню до неї = 7.



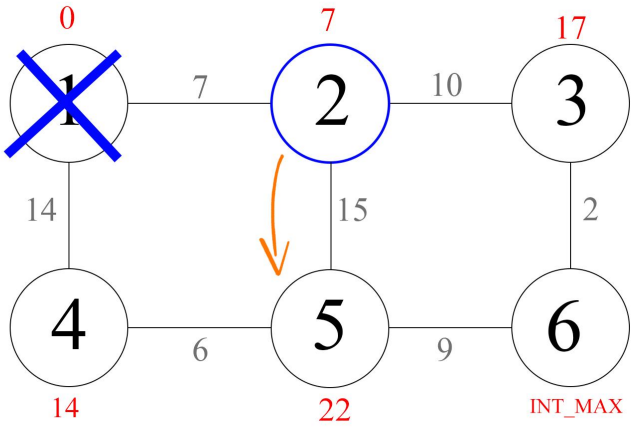
І знову намагаємося зменшити відстань до всіх сусідів 2-ї вершини, намагаючись пройти в них через 2-у. Сусідами 2-ї вершини є 1, 3, 5.

Крок 7: перший (по порядку) сусід вершини 2 — 1-ша вершина. Але шлях до неї з 2 вершини не буде меншим ніж той що вже лежить у 1. Тому з 1-ю вершиною нічого не робимо.

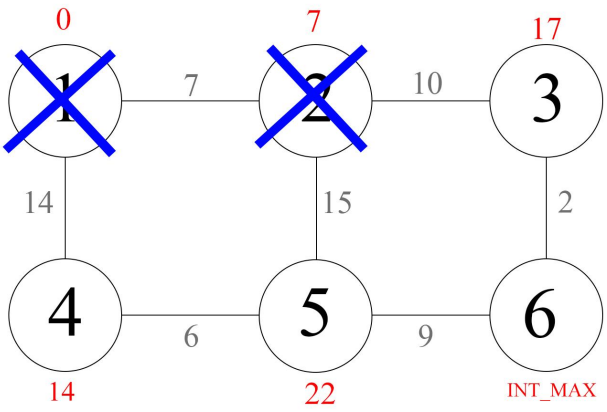
Крок 8: інший сусід вершини 2 — вершина 3. Якщо йти в неї через 2-у, то шлях буде: найкоротша відстань до 2-ї + відстань між 2-ю і 3-ю вершинами = 7 + 10 = 17. Оскільки 17 < INT_MAX, встановлюємо відстань до вершини 3 рівним 17 і заносимо вершину 3 у чергу.



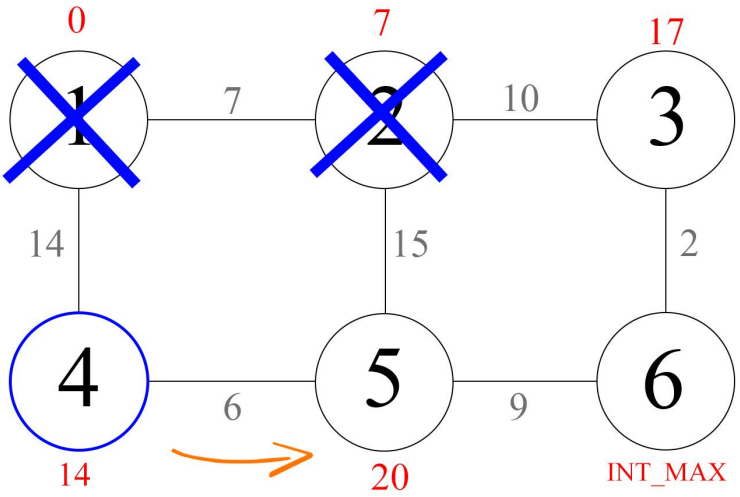
Крок 9: ще один сусід вершини 2 — вершина 5. Якщо йти в неї через 2-у, то шлях буде = 7 + 15 = 22. Встановлюємо відстань до вершини № 5 рівним 22 і заносимо вершину 5 у чергу.



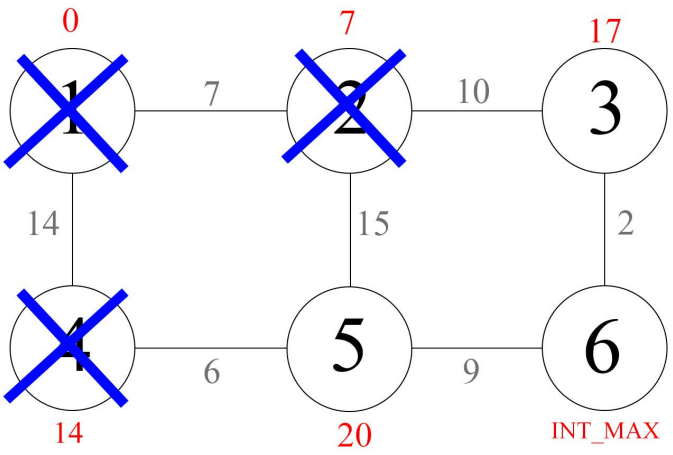
Крок 10: всі сусіди вершини 2 переглянуті, тому викреслюємо її з черги.



Кроки 11 — 15: по вже «відпрацьованій» схемі повторюємо кроки 2 — 6. Тепер «найближчою» у черзі виявляється вершина 4. Наступна суміжна до неї неопрацьована вершина – 5. Якщо йти до неї через 4-у вершину, то шлях буде 14 + 6 = 20. А оскільки 20 менше за попередньо збережену відстань 22, то відстань до вершини 5 замінюємо на 20 і заносимо її у чергу.

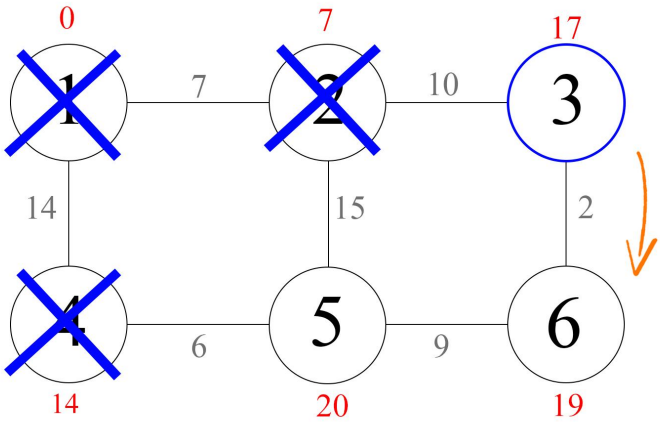


Після опрацювання 4-ї вершини:

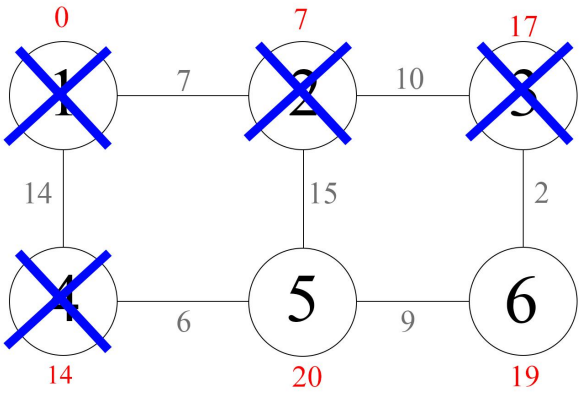


Наступною найближчою вершиною є вершина з номером 3.

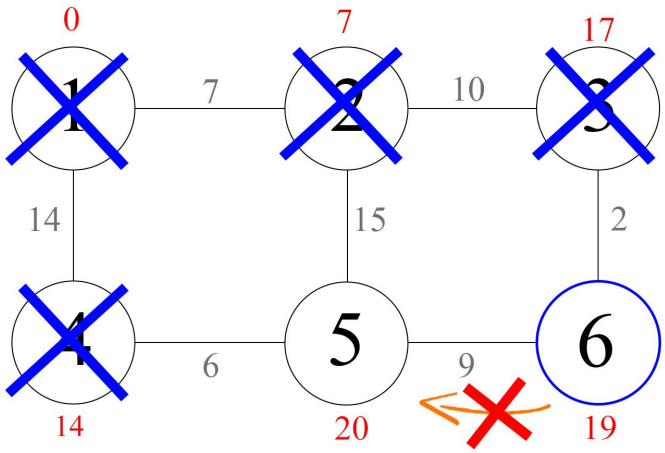
Повторюємо попередній алгоритм:



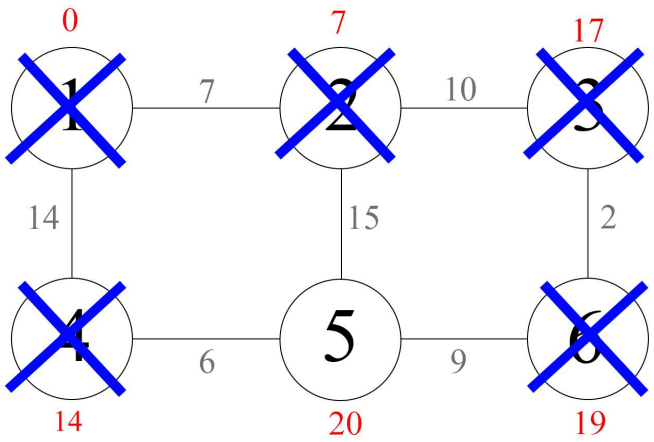
Отриманий результат:



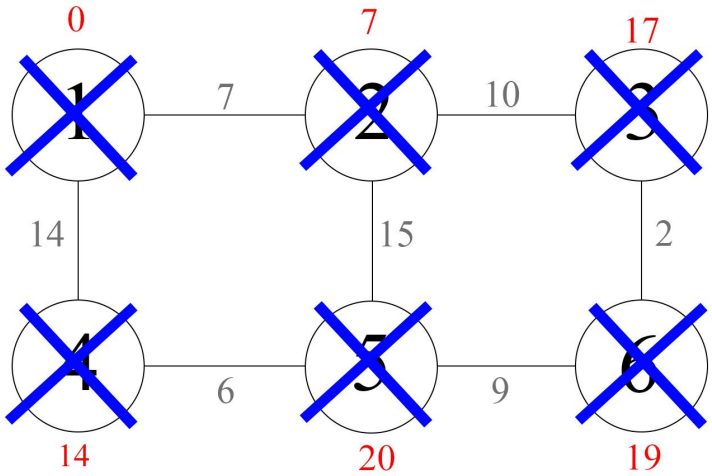
За аналогією, розглядаємо вершину 6: наступний сусід є вершина 5. Якщо йти до неї через 6-у вершину, то шлях буде 19 + 9 = 28. Але 28 більше попередньо збереженого значення (20). Тому поточну відстань до вершини 5 не міняємо.



Результат:



Повторюємо алгоритм із останньою вершиною 5: у черзі більше не залишилося жодних “вільних” вершин, отже:



Завершення виконання алгоритму: алгоритм закінчує роботу, коли в черзі не залишилося жодної вершини. Результат його роботи видно на останньому малюнку: **найкоротший шлях від 1-ї вершини до 2-ї становить 7, до 3-ї — 17, до 4-ї — 14, до 5-ї — 20, до 6-ї — 19 умовних одиниць.**

ЕЛЕМЕНТИ КОДУ

Программа написана на мові програмування “C++”. Даний вибір зроблений з метою прискорення виконання алгоритму зменшення часу роботи програми.

Зчитування даних з файлу, занесення вершин у матрицю:

```
freopen("example1.csv", "r", stdin);
int n;
double step;
cin >> n >> step;
int FROM_i, FROM_j, TO_i, TO_j;
cin >> FROM_i >> FROM_j;
cin >> TO_i >> TO_j;
vector<vector<double> > arr(n, vector<double>(n));
for(int i = 0;i < n; ++i)
{
    for(int j = 0;j < n; ++j)
    {
        scanf("%lf",&arr[i][j]);
    }
}
```

Створення вектору векторів для зберігання найкоротших відстаней до вершин а також вектору векторів для відновлення шляху.

```
vector<vector<double> > dist(n, vector<double>(n, INT_MAX));
vector<vector<pair<int, int> > > p(n, vector<pair<int, int> >(n, make_pair(0,0)));
```

Створення черги з пріоритетами та занесення в неї стартової вершини

```
priority_queue<pair<double, pair<int, int> > > q;
q.push(make_pair(0,make_pair(FROM_i, FROM_j)));
```

```
pair<double, pair<int, int> > front = q.top();
int from_i = front.second.first;
int from_j = front.second.second;
double len = -front.first;
q.pop();
```

```
if (len > dist[from_i][from_j] or len > dist[TO_i][TO_j])  {
    continue;
}
```

```
if((from_i - 1) >= 0)
{
    int to_i = from_i-1;
    int to_j = from_j;
    double cost = pow(pow(abs(arr[from_i - 1][from_j] - arr[from_i][from_j]),2) + pow(step,2),1.0/2);
    if(dist[to_i][to_j] > len + cost) {
        dist[to_i][to_j] = len + cost;
        q.push(make_pair(-dist[to_i][to_j],make_pair(from_i-1,from_j)));
        p[to_i][to_j] = make_pair(from_i, from_j);
    }
}
if(from_i + 1 < n)
{
    int to_i = from_i+1;
    int to_j = from_j;
    double cost = pow(pow(abs(arr[from_i + 1][from_j] - arr[from_i][from_j]),2) + pow(step,2),1.0/2);
    if(dist[to_i][to_j] > len + cost) {
        dist[to_i][to_j] = len + cost;
        q.push(make_pair(-dist[to_i][to_j],make_pair(from_i+1,from_j)));
        p[to_i][to_j] = make_pair(from_i, from_j);
    }
}
if(from_j - 1 >= 0)
{
    int to_i = from_i;
    int to_j = from_j-1;
    double cost = pow(pow(abs(arr[from_i][from_j-1] - arr[from_i][from_j]),2) + pow(step,2),1.0/2);
    if(dist[to_i][to_j] > len + cost) {
        dist[to_i][to_j] = len + cost;
        q.push(make_pair(-dist[to_i][to_j],make_pair(from_i,from_j-1)));
        p[to_i][to_j] = make_pair(from_i, from_j);
    }
}
if(from_j + 1 < n)
{
    int to_i = from_i;
    int to_j = from_j+1;
    double cost = pow(pow(abs(arr[from_i][from_j+1] - arr[from_i][from_j]),2) + pow(step,2),1.0/2);
    if(dist[to_i][to_j] > len + cost) {
        dist[to_i][to_j] = len + cost;
        q.push(make_pair(-dist[to_i][to_j],make_pair(from_i,from_j+1)));
        p[to_i][to_j] = make_pair(from_i, from_j);
    }
```

ВИСНОВОК

Під час роботи над даним проєктом, наша група значно розширила свої знання з дискретної математики, а також змогла знайти їм практичне застосування. Процес роботи приніс задоволення всім учасникам, не зважаючи на певні труднощі пов’язані зі спробою організувати все так, щоб кожен член команди міг працювати ефективно і виконав свою частину роботи. Було реалізовано кілька основних аспектів та досягнуті поставлені цілі:

- Вивчено принцип роботи алгоритму Дейкстра;
- Реалізовано програму, яка на основі алгоритму Дейкстра знаходить найкоротший шлях між двома точками поверхні;
- Покращено алгоритм завдяки використанню priority queue.
- Оптимізовано дану програму так, щоб вона швидко проводила обчислення для великих даних.