Науково-практичний звіт на тему

ОДИН МЕТОД РОЗВ'ЯЗАННЯ ЗАДАЧІ НАЙКОРОТШОГО ШЛЯХУ НА ПЛОЩИНІ

В.С. Аввакумов , студент 3 курсу, групи комп-мат 2

Анотація. У роботі запропоновано метод побудови найкоротшого шляху на площині для матеріальної точки через перешкоди, що задаються багатокутниками. В даному випадку це виявилось можливим завдяки побудові граф а видимості та використання алгоритму дейкстри.

Abstract. This paper considers the problem of computing the shortest path between two points in the plane in the presence of polygonal obstacles. We focus on the visibility graph approach combined with Dijkstra's algorithm. The method constructs a visibility graph over all obstacle vertices and query points, and then finds the shortest path in this graph with respect to Euclidean edge weights. We describe the geometric foundations of the approach, discuss its algorithmic complexity, and present a practical implementation that achieves O($n^2 \log n$) running time for n vertices of the obstacles.

1 Вступ

Постановка проблеми. Одним із ключових напрямів сучасної обчислювальної геометрії є побудова ефективних алгоритмів для визначення найкоротших шляхів у середовищах із геометричними перешкодами. Подібні задачі виникають у робототехніці, автоматизованому проєктуванні, моделюванні фізичних процесів, системах навігації та геоінформаційних технологіях. На площині такі перешкоди зазвичай задаються множиною полігональних об'єктів, сумарна кількість вершин яких дорівнює n. Завдання полягає у визначенні найкоротшого шляху між точками S і T, який проходить крізь допустиму область, не перетинаючи внутрішніх частин перешкод.

Аналіз останніх досліджень. Проблема побудови евклідового найкоротшого шляху серед полігональних перешкод активно досліджується з 1970-х років і має кілька фундаментальних підходів. Першим історично й одним із найпоширеніших є метод графа видимості, вперше систематизований Лозано-Пересом і Вестом [1], а пізніше розвинений О'Рурком та співавторами [2, 3]. Метод передбачає побудову графа, вершинами якого є точки S, T та всі вершини перешкод, а ребра з'єднують усі взаємно видимі пари вершин. Такий підхід забезпечує правильність і відносну простоту реалізації, проте найгірша кількість ребер графа сягає O(n^2), що визначає і часову складність повної побудови.

Значним розвитком цієї ідеї стала поява алгоритмів, що формують граф видимості у ранжованому за кутом порядку з використанням методів sweeping, зокрема робота Гоша та Маунта [4], де показано, що складність можна зменшити до O($n^2 \log n$). Подальше вдосконалення аналізу видимості було спрямоване на розробку структур даних та відсікання непотрібних перевірок, однак асимптотична межа залишилася сталою через квадратичну природу можливої кількості видимих пар точок.

Другий напрям — алгоритми типу «безперервної Дейкстри», або wavefront-методи, започатковані Мітчеллом [5] і розвинені Гершбергером та Сурі [6]. У цих роботах розглянуто геометричне розповсюдження хвилі від точки S у вільному просторі, що дозволяє досягнути теоретично оптимальної складності $O(n \log n)$. Однак реалізація таких алгоритмів є значно складнішою, потребує опрацювання великої кількості локальних подій і використання спеціалізованих структур даних, що обмежує їх практичне поширення під час написання продуктивного прикладного коду.

Третій напрям включає апроксимаційні методи. Починаючи з роботи Кларксона [7], сформувалися алгоритми, які будують шлях, близький до оптимального, у час, менший за квадратичний. Подальші дослідження у сфері геометричних спанерів [8, 9] та відтинальних структур дозволили отримати $(1+\varepsilon)$-апроксимацію за майже лінійний час. Однак ці методи втрачають точність і в ряді ситуацій не гарантують коректної поведінки при складній конфігурації перешкод.

Окремо слід виділити роботи, присвячені випадку опуклих перешкод. Ронерт [10] запропонував алгоритм, який використовує лише дотичні до опуклих полігонів, що дозволяє суттєво зменшити граф можливих переходів. Подібні підходи застосовні лише у специфічних умовах, а отже не є універсальними для загального випадку.

З практичної точки зору найширшого застосування набув саме підхід графа видимості зі складністю $O(n^2 \log n)$. Це пояснюється тим, що:

1. метод легко реалізується в імперативних мовах програмування;
2. складність алгоритму зумовлена геометричними обмеженнями задачі, а не складністю структур даних;
3. для реальних наборів даних квадратний випадок зустрічається рідко;
4. побудований граф може бути перевикористаний для багатьох запитів S–T;
5. метод поєднується з класичними алгоритмами пошуку найкоротших шляхів, зокрема Дейкстри або A^*.

У порівнянні з методами wavefront-типу, що мають найкращі теоретичні межі, підхід графа видимості потребує значно меншого обсягу коду, не вимагає складних подієвих структур і не стикається з труднощами числової стабільності, характерними для геометричного поширення хвилі. Ці особливості підтверджуються численними роботами, включно з монографією О'Рурка [3] та оглядом Гоша [11], де підкреслюється практична придатність видимісних структур у задачах планування руху.

Слід також зазначити, що навіть у дослідженнях, які спрямовані на розробку прискорених або оптимальних алгоритмів, такі структури часто використовуються як базовий інструмент. Зокрема, у роботах Гібаса, Гартфілда та співавторів [12–14] видимість між вершинами залишається ключовим елементом у побудові дорожніх графів та локальних структур навігації.

У загальному теоретичному контексті задача побудови найкоротшого шляху без обмеження на форму перешкод вважається доволі складною. Хоча існують алгоритми з оптимальною асимптотикою, вони мають високу комбінаційну складність і непрактичні для широкого інженерного застосування. Таким чином, метод побудови графа видимості зі складністю $O(n^2 \log n)$ залишається збалансованим компромісом між теоретичними результатами та практичними вимогами до реалізації.

Мета статті. У даній роботі систематизуються основні алгоритмічні підходи до задачі побудови найкоротшого шляху серед полігональних перешкод, аналізується їх обчислювальна складність та умови ефективності, а також обґрунтовується застосування алгоритмів на основі графа видимості як найбільш практично придатного підходу для інженерних та програмних застосувань.

2 Основна частина.

Сформулюємо геометричну постановку задачі роздільності.

Постановка задачі найкоротшого шляху. Нехай на площині задано дві точки S та T, а також список полігонів, вони ж багатокутники, які в свою чергу задані списком точок, що їх формують (в моїй реалізації прямокутник - список почергово поєднаних точок в порядку наявності в списку). Необхідно визначити точки, які формують найкоротший маршрут () з S в T.

Лема 1 (про найкоротший шлях). *Будь-який найкоротший шлях між двома вершинами за наявності полігональних перешкод є ламаною, вершини якої — це вершини полігонів.*

Доведення: *Нехай найкоротший шлях не є ламаною. У такому разі на шляху існує така точка , яка не належить жодному прямому відрізку. Це означає, що існує ε -околиця точки p, у яку не потрапляє жодна перешкода (випадок, коли точка лежить на ребрі, розглядається аналогічно). Тоді підшлях, що знаходиться всередині ε-околиці, за нерівністю трикутника можна скоротити хордою, яка сполучає точки перетину межі ε-околиці зі шляхом. Раз частину шляху можна зменшити, то можна зменшити й увесь шлях, а це означає, що вихідне припущення некоректне.*

2.1. Побудова розв'язку задачі знаходження найкоротшого шляху.

Означення 1 Вершина p_j вважається *видимою* з вершини p_i, якщо не знайдеться жодного такого відрізка (сегмента) s_k, що перетне сегмент (відрізок) $[p_j, p_i]$, в тривіальному випадку коли $s_k = [p_j, p_i]$ точка p_j - видима.

2.1.1 постановка задачі побудування графа видимості

Нехай є множина сегментів - ребер перешкод $\mathbb{S}$, а також множин точок - вершин цих ребер $\mathbb{P}$, для кожної вершини $p_i \in \mathbb{P}$ потрібно знайти всі такі вершини $p_j \in \mathbb{P}, i \neq j$, що *видимі (див. Означення 1)*, щойно ми пропробимо такий пошук для $\forall i = 1, ..., |P|$ ми отрмаємо граф видимості заданий списком суміжності $p_i \to$ {список видимих для неї вершин}

2.1.1.1 наівний алгоритм

Якщо ми будуватимемо граф, то отримаємо доволі погану асимптотичну оцінку, а саме $O(n^3)$, адже ми матимемо наступний алгоритм.

для кожної точки $p_i \in \mathbb{P}$:

для кожної точки $p_j | j \neq i \in \mathbb{P}$:

для кожного ребра $s_k \in \mathbb{S}$:

якщо s_k пересікає $[p_i, p_j]$ то переходимо до $[p_{j+1}]$, інакше продовжуємо

Додаємо p_j до списку суміжності p_i

Подібна реалізаціє має право на життя коли необхідно зробити алгоритм що швидко пишеться та легко модифікається, але на великій кількості вхідних даних від буде працювати повільно

2.1.1.2 Алгоритм лі

Якщо ми покращимо знаходження видимих точок до $O(n \log(n))$, то побудова графа видимості стане $O(n^2 \log(n))$, що значно краще аніж $O(n^3)$. Ідея Лі, полягає в використанні замітаючого променя, де подією є вершина (точка) сегмента, а статус - впорядкований список сегментів Тобто ми дивимось чи пересікає $[p_i, p_j]$ найближчий до нього s_k, цієї перевірки на видимість досить, адже якщо кілька відрізків пересікають $[p_i, p_j]$, то пересікає і найближчий.

2.1.2 Постановка задачі знаходження найкоторшого шляху в графі

Пошук в графі, сьогоднішнім state of the art є алгоритм деікстри, а саме його версія удосконалена еврестичною відстанню - A^*, я використовуватиму простий алгоритм деікстри з вагою ребра p_i, p_j дорівнюватиме довжині відрізка $[p_i, p_j]$, обидва ці алгоритми працюють за $O(n^2 \log(n))$, але A^* константно кращий (в кращому випадку).

2.2 Побудова розв'язку задачі видимості методом заметаючого променя

Нагадаю що ми розглядаємо множину точок $\mathbb{P}$ до якої входять S (початок шляху) та T (кінець шляху), а також точки множини сегментів $\mathbb{S}$, тобто ребер полігонів перешкод.

Попердня обробка 1 Для кожної $p_i \in \mathbb{P}$ проводимо промінь вертикально вгору, також створюємо червоно-чорне дерево подій aka `ordered set` в большості мов програування, де компоратором є відстань від p_i до сегмента та заповнюємо це дерево сегментами, які перетинає наш вертикальний промінь.

Попердня обробка 2 Для кожної $p_i \in \mathbb{P}$ Створюємо список вісіх $p_j |\ j \neq i \in \mathbb{P}$, та сортуємо його за радіальним кутом відносно p_i (за годинниковою стрілкою)

Статус променя Статус - збалансоване дерево за відстаню до перетину Неодхідні нам операції:
- вставка $\log(n)$
- видалення $\log(n)$
- вилучення найменшого $\log(n)$

Після обробки кожної p_j ми:
- видаляємо всі ребра (максимум 2) зі статусу що закінчюються () на p_j,
- додаємо всі ребра (максимум 2) зі статусу що починаються на p_j

Означення 2 ребро $[p_n, p_m]$ *закінчується* на $p_n \leftrightarrow p_n$ знаходиться нижче за рядіальним кутом аніж p_m, відповідно $[p_n, p_m]$ починається на p_m

Перевірка видимості Якщо відрізок $[p_i, p_j]$ не має перетину з найбижчим до нього сегментом з стутусу, то точка p_j вважається видимою

Пободова графа для кожної точки p_i шукаємо всі видимі p_j та з'єднуємо їх ребрами

Пошук в графі Простий алгоритм деікстри (BFS з пріоритетною чергою, де пріоритет - довжина ребра)

Алгоритм.

Наведу псевдокод та ілюстрації

```
// Побудова графа видимості та пошук найкоротшого шляху

// Вхідні дані:

// PP — множина вершин (включно з S та T)

// SS — множина сегментів (ребер полігональних перешкод)

// Вихідні дані:

// 1. Побудова графа видимості G

нехай G — порожній граф

для кожної p_i ∈ ℙ:
```

// Попередня обробка 1: ініціалізація статусу

нехай статус — порожнє червоно-чорне дерево

провести промінь вертикально вгору з точки p_i

для кожного сегмента $s_k \in \mathbb{S}$:

якщо промінь перетинає s_k,

вставити s_k у статус з ключем «відстань від p_i до точки перетину»

// Попередня обробка 2: формування списку кандидатів

нехай C — порожній список вершин-кандидатів

для кожної вершини $p_j \in \mathbb{P}$, $j \neq i$:

якщо $p_j.x \geq p_i.x$ (права півплощина відносно p_i),

додати p_j до C

// Сортування за радіальним кутом

відсортувати C за зростанням радіального кута $\angle(p_i, p_j)$ за годинниковою стрілкою

// Прохід замітаючим променем

для кожної вершини p_j в C у відсортованому порядку:

// Оновлення статусу

видалити зі статус усі сегменти s_k, що *закінчуються* в p_j

додати до статус усі сегменти s_k, що *починаються* в p_j

// Перевірка видимості вершини p_j

нехай $s_{\min}$ — найближчий до p_i сегмент у статус (якщо він існує)

якщо відрізок $[p_i, p_j]$ *не перетинає* $s_{\min}$ або $s_{\min}$ *не існує*:

//p_j видима з p_i

додати в G ребро (p_i, p_j) довжини $| p_i p_j |$

додати в G ребро (p_j, p_i) довжини $| p_i p_j |$

//2. Пошук найкоротшого шляху в графі G (алгоритм Дейкстри)

для кожної вершини $p \in \mathbb{P}$:

$\text{dist}(p) := +\infty$

$\text{parent}(p) := \emptyset$

$\text{dist}(S) := 0$ нехай Q — пріоритетна черга (мін-heap) за значенням dist

вставити S у Q з ключем $\text{dist}(S)$

поки Q не порожня:

витягнути з Q вершину u з мінімальним $\text{dist}(u)$

якщо $u = T$, перервати цикл

для кожного сусіда v вершини u в G:

якщо $\text{dist}(u) + w(u, v) < \text{dist}(v)$:

$\text{dist}(v) := \text{dist}(u) + w(u, v)$

$\text{parent}(v) := u$

оновити ключ вершини v в Q

нехай шлях P — порожній список нехай cur := T

поки cur $\neq \emptyset$:

додати cur на початок списку P

cur := $\text{parent}(\text{cur})$

// На виході:

// P — послідовність вершин найкоротшого шляху від S до T

// dist(T) — довжина найкоротшого маршруту

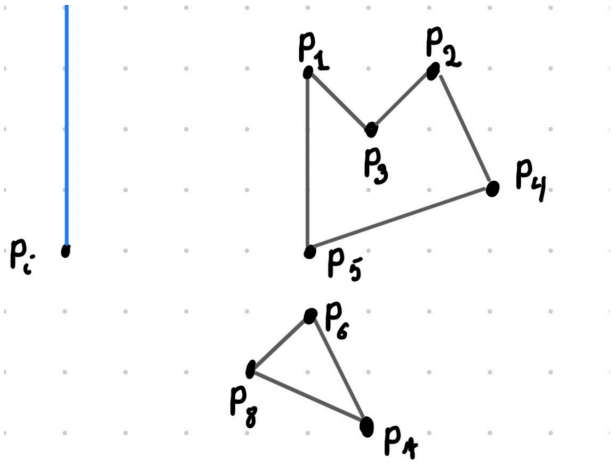
Покрокова демонстрація на приватному випадку

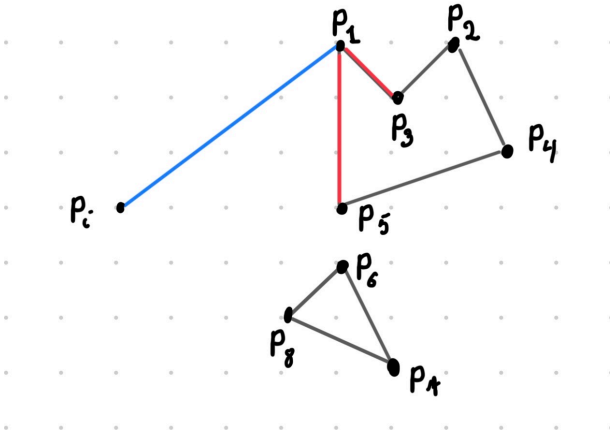
C = $\{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9\}$ (вже відсортовані)

статус = {}

$\mathbb{S} = \{[p_1, p_3], [p_2, p_3], [p_2, p_4], [p_1, p_5], [p_4, p_5], [p_6, p_8], [p_6, p_7], [p_8, p_7]\}$

видимі = {}

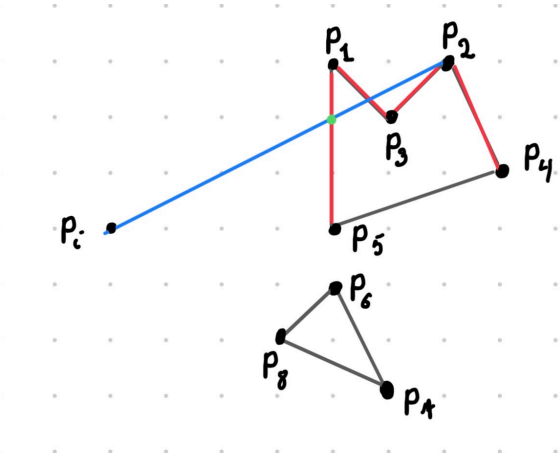




$s_{\min} = \emptyset$

статус = {} + $[p_1, p_5]$,$[p_1, p_3]$

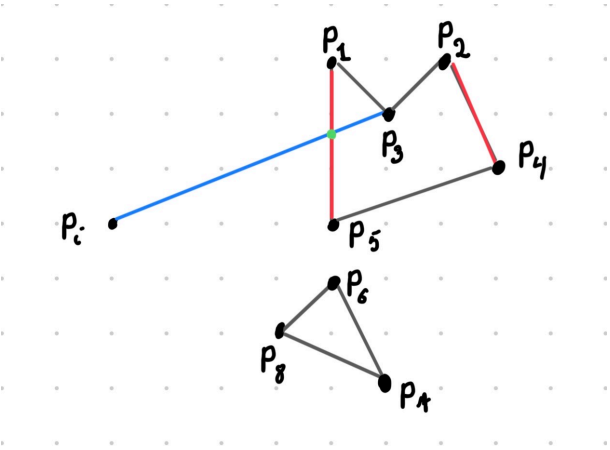
видимі = {} + p_1



$s_{\min} = [p_1, p_5], [p_i, p_2]$ перетинає $\Rightarrow p_2$ не видно

статус = { $[p_1, p_5]$,$[p_1, p_3]$} + $[p_2, p_3]$, $[p_2, p_4]$

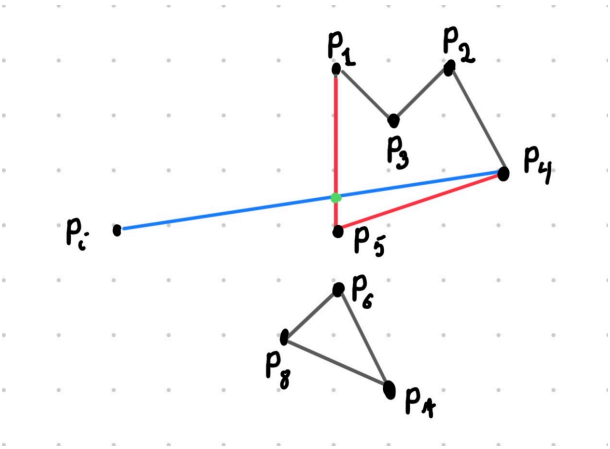
видимі = {p_1}



$s_{\min} = [p_1, p_5], [p_i, p_3]$ перетинає $\Rightarrow p_3$ не видно

статус = { $[p_1, p_5]$, $[p_1, p_3]$, $[p_2, p_3]$, $[p_2, p_4]$} - $[p_1, p_3]$, $[p_2, p_3]$

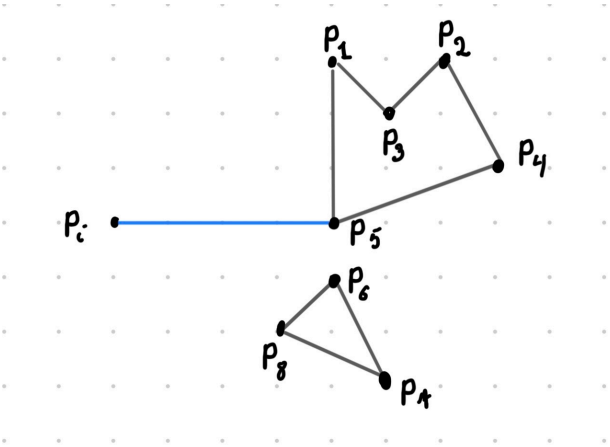
видимі = {p_1}



$s_{\min} = [p_1, p_5], [p_i, p_4]$ перетинає $\Rightarrow p_4$ не видно

статус = { $[p_1, p_5], [p_2, p_4]$} - $[p_2, p_4]$ + $[p_4, p_5]$

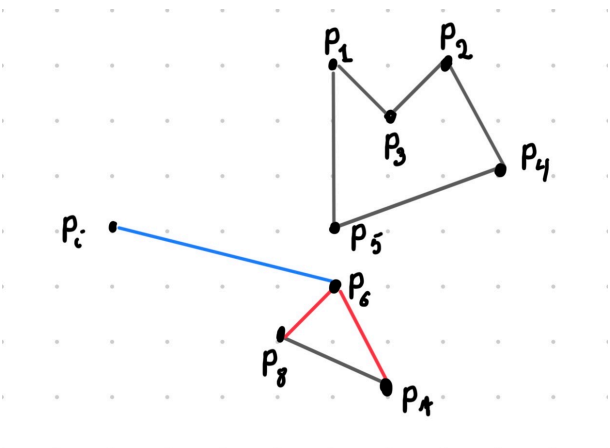
видимі = $\{p_1\}$



$s_{\min} = [p_1, p_5], [p_i, p_5]$ НЕ перетинає $\Rightarrow p_5$ видно

статус = { $[p_1, p_5], [p_4, p_5]$} - $[p_1, p_5], [p_4, p_5]$

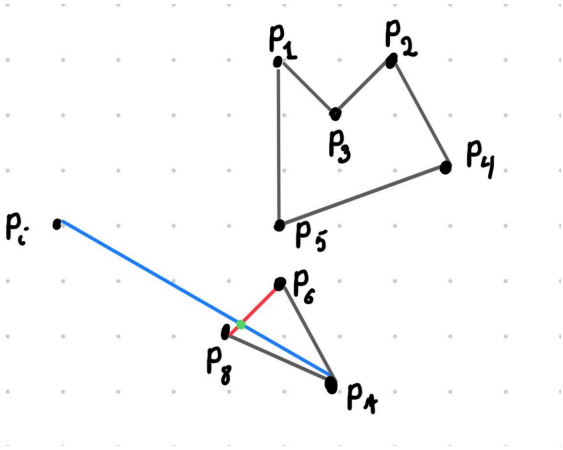
видимі = $\{p_1\}$ + p_5



$s_{\min} = \emptyset \Rightarrow p_6$ видно

статус = {} + $[p_6, p_8], [p_6, p_7]$

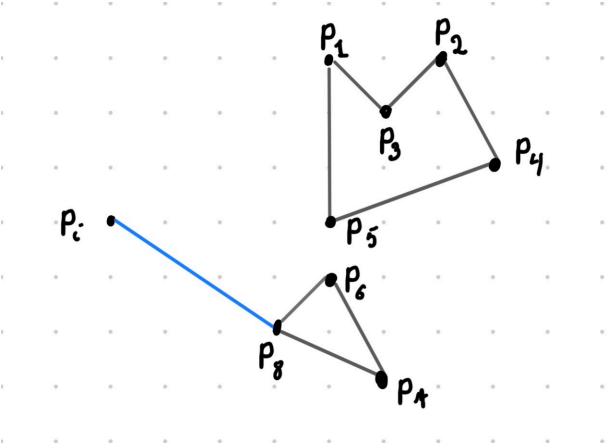
видимі = $\{p_1, p_5\}$ + p_6



$s_{\min} = [p_6, p_8], [p_i, p_7]$ перетинає $\Rightarrow p_7$ не видно

статус = { $[p_6, p_8], [p_6, p_7]$} - $[p_6, p_8]$

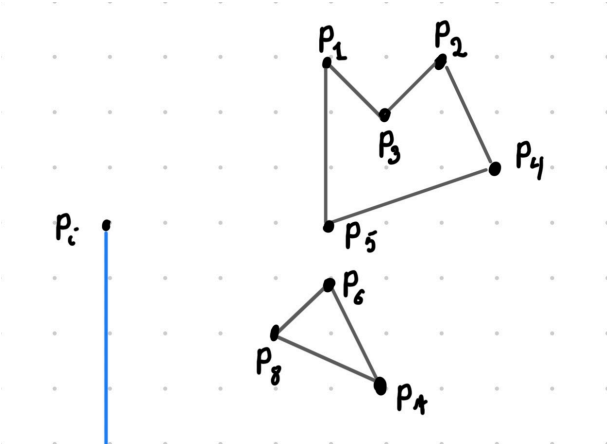
видимі = $\{p_1, p_5, p_6\}$



$s_{\min} = [p_6, p_8], [p_i, p_8]$ НЕ перетинає $(p_8 \in [p_6, p_8]) \Rightarrow p_8$ не видно

статус = { $[p_6, p_7]$} - $[p_6, p_7]$

видимі = $\{p_1, p_5, p_6\}$ + p_8



The end

видимі = $\{p_1, p_5, p_6, p_8\}$

3 Обґрунтування складності

Теорема 1. Часова складність розв'язання задачі найкоротшого шляху між точками S та T серед полігональних перешкод методом побудови графа видимості (заметаючий промінь за Лі) та подальшим пошуком найкоротшого шляху алгоритмом Дейкстри становить $O(n^2 \log n)$, де n = $|\mathbb{P}|$ — кількість усіх вершин (включно з S і T). Передбачається, що базові геометричні операції (орієнтація, перевірка перетину двох відрізків) виконуються за $O(1)$.

Доведення. Нехай $\mathbb{S}$ — множина ребер перешкод, $\mathbb{P}$ — множина всіх вершин. Для полігональних перешкод кількість ребер та вершин одного порядку, тому $|\mathbb{S}| = O(n)$.

1) Побудова графа видимості. Для кожної вершини $p_i \in \mathbb{P}$ виконуємо процедуру getVisibleVertices(p_i):

1.1) Побудова початкового статусу. Проводимо вертикальний промінь з p_i та переглядаємо всі ребра $s_k \in \mathbb{S}$, щоб визначити, які з них перетинають цей промінь. Кожна така перевірка займає $O(1)$, отже перегляд усіх ребер дає $O(|\mathbb{S}|) = O(n)$. Кожен сегмент, що перетинає промінь, вставляється у статус (червоно-чорне дерево) за O(log n). У гіршому випадку вставляємо O(n) сегментів, тому маємо O(n log n).

1.2) Формування списку кандидатів та сортування. Створюємо список C усіх $p_j \in \mathbb{P} (j \neq i)$, що лежать у правій півплощині відносно p_i. Формування списку — один перегляд $\mathbb{P}$, тобто $O(n)$. Далі сортуємо C за радіальним кутом навколо p_i, що займає $O(n \log n)$.

1.3) Прохід заметаючим променем (обробка подій). Розглядаємо $p_j \in C$ у відсортованому порядку. Після обробки кожної події p_j змінюється статус: • видаляються ребра, що "закінчуються" у p_j (не більше 2), • додаються ребра, що "починаються" у p_j (не більше 2). Кожна операція вставки/видалення у червоно-чорному дереві займає $O(\log n)$. Оскільки для кожної події виконується $O(1)$ оновлень, а подій O(n), сумарно на оновлення статусу йде $O(n \log n)$.

1.4) Перевірка видимості. Для кожної події p_j достатньо перевірити перетин відрізка $[p_i, p_j]$ з найближчим сегментом статусу. Отримання "найближчого" елемента зі статусу виконується за $O(\log n)$, а перевірка перетину — O(1). Отже в гіршому випадку на всі перевірки маємо $O(n \log n)$.

З пунктів 1.1–1.4 випливає, що для фіксованої вершини p_i процедура пошуку видимих вершин працює за $O(n \log n)$. Оскільки таких вершин $p_i \in n$, побудова всього графа видимості займає: $n \cdot O(n \log n) = O(n^2 \log n)$.

2) Пошук найкоротшого шляху в графі. Після побудови графа видимості застосовуємо алгоритм Дейкстри з пріоритетною чергою. У гіршому випадку граф видимості може бути щільним: кількість ребер $m = O(n^2)$. Тоді час роботи Дейкстри становить $O(m \log n) = O(n^2 \log n)$.

3) Підсумок. Загальний час роботи алгоритму дорівнює сумі: $O(n^2 \log n)$ (побудова графа) + $O(n^2 \log n)$ (Дейкстра) = $O(n^2 \log n)$.

Отже, твердження теореми доведено.

4 Практична частина

Особливістю даної реалізації є можливість роботи з довільною множиною полігональних перешкод та автоматичне відшукання найкоротшого маршруту між двома заданими

точками S та T. Програма будує граф видимості за полігономиальними перешкодами, а потім знаходить найкоротший шлях у цьому графі, що дозволяє працювати як з простими сценами (декілька прямокутників), так і з більш складними наборами багатокутників.

Алгоритмічна частина програмної реалізації написана мовою Python (файл `Graph.py`). Обчислювальна геометрія (орієнтація трійки точок, перевірка перетину відрізків, нормалізація контурів) реалізована у вигляді окремих функцій над примітивами `Point` та `Segment`. Побудова графа видимості виконується у функціях `visibility_graph_segments` та `build_visibility_graph`, а пошук найкоротшого шляху реалізовано алгоритмом Дейкстри у функції `dijkstra_path` із використанням модуля `heapq` стандартної бібліотеки як пріоритетної черги.

Для підтримання статусу заметаючого променя (відсортованого за відстанню набору активних відрізків) застосовується структура даних `SortedList` з пакета `sortedcontainers`, що реалізує збалансоване дерево пошуку з логарифмічним часом вставки, видалення та пошуку. Це дозволяє досягти сумарної асимптотики O(n² log n) при побудові графа видимості, відповідно до теоретичного аналізу.

Для графічного відображення сцени та знайденого маршруту використовується бібліотека `matplotlib`. Окрема функція `plot_scene` візуалізує полігональні перешкоди, початкову та кінцеву точки, а також послідовність вершин найкоротшого шляху. Демонстраційний режим (`demo_with_random_scene`) дозволяє автоматично згенерувати випадкову сцену з прямокутними перешкодами за допомогою допоміжного модуля `random_scene.py`, запустити алгоритм і наочно перевірити коректність роботи реалізації.

5 Висновки

У роботі було розв'язано задачу знаходження найкоротшого шляху між двома точками на площині за наявності полігональних перешкод шляхом побудови графа видимості та подальшого пошуку маршруту в ньому. Основою підходу став алгоритм побудови графа видимості методом заметаючого променя, що дає змогу ефективно визначати пари вершин, які мають взаємну видимість. На відміну від наївного підходу з асимптотикою $O(n^2)$, використання впорядкованої структури даних для статусу променя дозволило зменшити складність побудови графа до $O(n^2 \log n)$.

Реалізація алгоритму виконує повний аналіз сцени з довільними полігональними перешкодами та обробляє їх вершини у радіальному порядку навколо кожної точки. Статус променя підтримується у вигляді збалансованого дерева (`SortedList`), що забезпечує логарифмічний час вставки та видалення і дозволяє швидко визначати найближчий активний сегмент. Це робить перевірку видимості кожної вершини локальною та ефективною. Додатковий механізм контролю сусідності вершин усередині одного багатокутника гарантує, що алгоритм не утворює невалідних діагональних ребер.

Після побудови графа видимості застосовується алгоритм Дейкстри для визначення найкоротшого маршруту між початковою та кінцевою точками. Оскільки ваги ребер відповідають евклідовій відстані між вершинами, знайдений шлях є оптимальним. Часова складність цього етапу — $O(E \log V)$, що в типовому випадку також не перевищує $O(n^2 \log n)$.

Реалізація підтримує графічне відображення сцени та знайденого маршруту, що дає змогу наочно оцінити коректність роботи алгоритму. Генерація випадкових сцен із багатокутними перешкодами показала стабільну роботу методу та підтвердила теоретичні оцінки продуктивності.

Запропонований підхід демонструє універсальність та можливість застосування в задачах робототехніки, навігації автономних агентів, комп'ютерної графіки та моделювання. Висока точність геометричних обчислень і масштабованість алгоритму роблять його придатним для використання в практичних системах, що потребують побудови найкоротших маршрутів у складних середовищах. З урахуванням модульності реалізації можливе подальше розширення, наприклад, застосування евристичних методів A^*, оптимізація структур даних або адаптація алгоритму для динамічних сцен.

Список літератури

[1] T. Lozano-Pérez, M. Wesley, "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles," Communications of the ACM, 1979.

[2] J. O'Rourke, "Visibility," in Handbook of Discrete and Computational Geometry, CRC Press, 1997.

[3] J. O'Rourke, "Computational Geometry in C," Cambridge University Press, 1998.

[4] S. Ghosh, D. Mount, "An Output-Sensitive Algorithm for Computing Visibility Graphs," SIAM J. Comput., 1991.

[5] J. S. B. Mitchell, "Shortest Paths Among Obstacles in the Plane," in Proceedings of FOCS, 1987.

[6] J. Hershberger, S. Suri, "An Optimal Algorithm for Euclidean Shortest Paths in the Plane," SIAM Journal on Computing, 1999.

[7] K. Clarkson, "Approximation Algorithms for Shortest Path Motion Planning," STOC, 1987.

[8] G. Das, D. Joseph, "The Complexity of Minimum Spanners in Euclidean Graphs," Discrete & Computational Geometry, 1990.

[9] S. Arya, D. M. Mount, "Approximate Nearest Neighbor Queries in Fixed Dimensions," SODA, 1993.

[10] W. Rohnert, "Shortest Path Problems in the Plane with Convex Polygonal Obstacles," IPL, 1986.

[11] S. Ghosh, "Visibility Algorithms in the Plane," Cambridge University Press, 2007.

[12] L. Guibas, J. Hershberger, "Moving Obstacles and the Shortest Path Problem," Algorithmica, 1989.

[13] D. Hartfield, L. King, "Shortest Paths in the Presence of Obstacles," Discrete Applied Math., 1992.

[14] J. Canny, "The Complexity of Robot Motion Planning," MIT Press, 1988.

[15] P. K. Agarwal, M. Sharir, "Applications of Davenport–Schinzel Sequences in Computational Geometry," Handbook, 2001.

[16] H. Edelsbrunner, "Algorithms in Combinatorial Geometry," Springer-Verlag, 1987.

Додатки