Київський національний університет ім. Т. Шевченка
Факультет комп’ютерних наук та кібернетики

Терещенко В.М.

Аналіз методів розв’язання задач геометричного пошуку

Навчальний посібник

з виконання лабораторних робіт з курсу «Обчислювальна геометрія комп’ютерна графіка»
для студентів факультету комп’ютерних наук та кібернетики

Київ 2023

ВСТУП

Досить часто при розробці ефективних рішень в області новітніх технологій так чи інакше виникають задачі для яких самим продуктивним підходом розв'язання є їх геометрична формалізація і подальше застосування методів обчислювальної геометрії і, зокрема, методів геометричного пошуку. Тому не дивно, що в курсах обчислювальної геометрії для практичних чи лабораторних занять значне місце посідають задачі геометричного пошуку. При цьому слід зазначити, що геометричний пошук має свої особливості: вхідними даними є складні геометричні об'єкти (точки, відрізки, полігони, многогранники, розбиття простору та інші складні геометричні об'єкти, а пошуковий запит має бути порядку O(logn) з оптимальним використанням пам'яті. На відміну від класичної математики нам необхідно знайти не лише розв'язок задачі в принципі, а найефективніший розв'язок. У цьому й полягає складність розв'язання задач геометричного пошуку. Проте, обнадійливим фактором є те, що для розв'язання задач геометричного пошуку напрацьовано багато теоретичного та практичного матеріалу, а також розроблені ефективні алгоритмічні інструменти: структури даних, геометричні структури та швидкі алгоритми.

Посібник присвячений огляду основних методів та підходів розробки ефективних алгоритмів для розв'язання задач геометричного пошуку при виконанні лабораторних робіт з курсу «Обчислювальна геометрія та комп'ютерна графіка».

Усі ми добре знайомі зі звичайним пошуком на множині цілих (дійсних) чисел чи будь-якому файлі даних, наприклад, пошук слова (зразка) в текстовому документі. Результатом в цьому випадку може бути місце у тексті, яке співпадає із шуканим словом-шаблоном. Тобто суттю такого пошуку є встановлення зв'язку між файлом із зразком. Геометричний пошук має свою специфіку так, як зразок і файл можуть мати складну структуру (геометричне розбиття простору у вигляді сітки чи графа, множина многокутників (многогранників)). І тут скоріше йде мова не про співпадіння зразка і файлу, а про їх взаємне розташування. Тому при розробці алгоритмів необхідно розглядати не стільки традиційні підходи, як спеціальні підходи, які враховують топологію розташування об'єктів.

Клас задач геометричного пошуку можна розділи на два основні підкласи:

- задачі локалізації точки на геометричному розбитті простору;
- задачі регіонального пошуку на множині n об'єктів в евклідовому E^d просторі.

Список задач геометричного пошуку представлено нижче, таб. 1.

№	код	Назва та постановка задачі	Оцінки складності
		Геометричний пошук	
		Локалізація точки	
1	ЛТПР	*Планарне розбиття.* Для заданого n-вершинного планарного розбиття на площині локалізувати точку P.	
2	ЛТОР	*Ортогональне розбиття.* Для заданого n-вершинного ортогонального планарного розбиття на площині локалізувати точку P.	
3	ЛТЕDP	*Прямолінійне розбиття E^d простору.* Для заданого n-вершинного розбиття евклідового Ed простору локалізувати точку P.	
4	ЛТЕОР	*Ортогональне розбиття E^d простору.* Для заданого n-вершинного планарного розбиття на площині локалізувати точку P.	
		Регіональний пошук	
5	ГРПП(АГ0)	*Регіональний пошук множини точок у прямокутному регіоні.* В E^d (d ≥ 2) задано n точок. Знайти усі точки, які потрапляють в середину прямокутного запитного регіону сторони, якого паралельні координатним осям.	O(klogn)
6	ГРПОМ(АГ1)	*Регіональний пошук множини точок в опуклому регіоні.* В E^d (d ≥ 2) задано n точок. Знайти усі точки, які потрапляють в середину опуклого k –вершинного регіону.	O(klogn)
7	ГРПРМ(АГ2)	*Регіональний пошук множини точок у простому регіоні.*	O(klogn)

		В E^d (d ≥ 2) задано n точок. Знайти усі точки, які потрапляють в середину простого k–вершинного многокутника (многогранника).	
8	ГРПК(АГ3)	*Регіональний пошук множини точок у круговому регіоні.* В E^d (d ≥ 2) задано n точок. Знайти усі точки, які потрапляють в середину кругового регіону радіусу R.	O(logn)
9	ГРППВ(ВГ0)	*Регіональний пошук множини відрізків, які перетинають прямокутний регіон.* В E^d (d ≥ 2) задано n прямолінійних відрізків. Знайти усі відрізки, які перетинають чи потрапляють в середину прямокутного запитного регіону сторони, якого паралельні осям координат	O(klogn)
10	ГРПОМВ (ВГ1)	*Регіональний пошук множини відрізків, які перетинають опуклий регіон.* В E^d (d ≥ 2) задано n прямолінійних відрізків. Знайти усі відрізки, які перетинають чи потрапляють в середину опуклого k –вершинного регіону.	O(klogn)
11	ГРППМВ (ВГ2)	*Регіональний пошук множини відрізків, які перетинають простий регіон.* В E^d (d ≥ 2) задано n прямолінійних відрізків. Знайти усі відрізки, які перетинають чи потрапляють в середину простого k –вершинного регіону.	O(klogn)
12	ГРПКВ(ВГ3)	*Регіональний пошук множини відрізків, які перетинають круговий регіон.* На площині задано n прямолінійних відрізків. Знайти усі відрізки, які перетинають чи потрапляють в середину круга радіусу R.	O(logn)

Таблиця 1. Список задач геометричного пошуку

Наведені у списку задачі мають практичне застосування і є одними з пріоритетних задач в більшості прикладних систем, пов'язаних із зберіганням, аналізом та обробкою даних, які являють собою геометричні об'єкти. До таких систем відносяться ГІС, системи автоматизованого проектування, навігаційні системи, тощо.

АЛГОРИТМІЧНІ ІНСТРУМЕНТИ

У роботі [1] описані основні алгоритмічні інструменти (стратегії, структури даних) для розв'язування оптимізаційних задач обчислювальної геометрії. Ці інструменти дозволяють досягти високої ефективності алгоритмів розв'язання не лише задач оптимізації, але й переважної більшості класів задач обчислювальної геометрії і, в тому числі, задач геометричного пошуку. До цих інструментів варто додати інтегральні геометричні структури даних такі, як Діаграма Вороного та Триангуляція Делоне, які дозволяють на комплексному рівні отримати високу ефективність великої кількості алгоритмів розв'язування різних задач обчислювальної геометрії.

1.1 СТРУКТУРИ ДАНИХ

Як уже зазначалося раніше вхідні дані в обчислювальній геометрії мають свою специфіку і зазвичай являють собою множини геометричних об'єктів складної структури: множина точок, множина ребер, множина многокутників, множина многогранників, множина об'єктів обертання, планарне розбиття, тощо. Для розробки ефективних алгоритмів геометричного пошуку ключове місце посідають структури даних для представлення множин вхідних геометричних даних. Найбільш поширеними представленнями множин геометричних об'єктів в обчислювальній геометрії і геометричному пошуці зокрема є: списки, стеки, черги, РСПЗ, дерева, хеш таблиці [2-5] та такі геометричні інтегральні структури як Діаграма Вороного та Триангуляція Делоне. Коротко охарактеризуємо основні структури даних, які ми будемо використовувати в задачах геометричного пошуку.

Означення 1.1. *Опис складних об'єктів засобами більш простих типів даних, які безпосередньо представляються у машині, називається структурами даних.*

Склад структур даних: структура пам'яті для зберігання даних, способи її формування, модифікація та доступ до даних, набір операцій над елементами даних.

Основні типи структур даних

Списки

Розрізняють: односторонні кільцеві зв’язані списки(однозв’язні), двосторонні кільцеві зв’язані списки (двозв’язні) (рис.1.1), динамічні списки (стеки і черги, рис.1.2), спеціальні списки (реберний список з подвійними зв’язками РСПЗ, рис.1.3), індексовані списки, тощо.

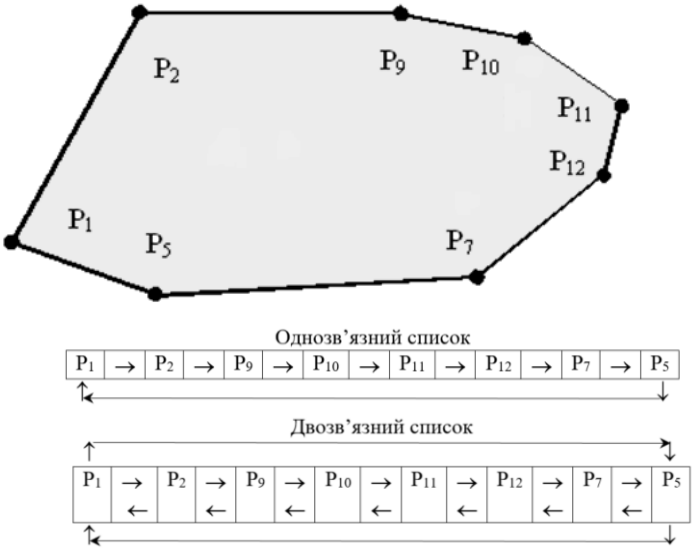


Рис. 1.1. Представлення полігону у вигляді одно та двозвязних кільцевих спсиків.

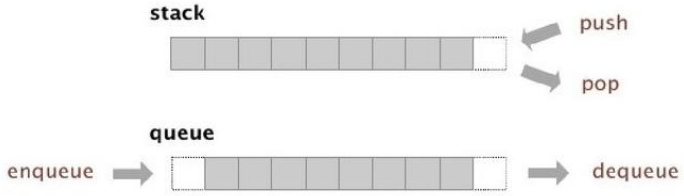
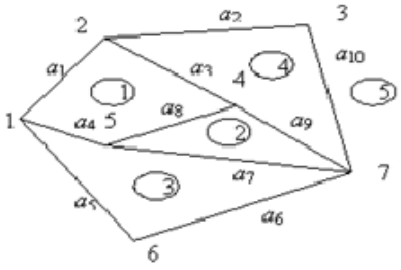


Рис.1. 2 Приклад стеку та черги.



	V_1	V_2	F_1	F_2	P_1	P_2
a_1	1	2	5	1	5	3
a_2	2	3	5	4	1	10
a_3	2	4	4	1	2	8
a_4	1	5	1	3	1	7
a_5	1	6	3	5	4	6
a_6	6	7	3	5	5	10
a_7	5	7	2	3	8	6
a_8	5	4	1	2	4	9
a_9	4	7	4	2	3	7
a_{10}	3	7	5	4	2	9

Рис. 1. 3. Предстaлення планарного графа за допомогою РСПЗ.

Бінарні дерева [2-6].

Дерево відрізків. Дерево відрізків визначено на цілочисельному інтервалі $[L, N]$ і будується рекурсивно як двійкове дерево, рис. 1. 4. На дереві визначено операції вставки та вилучення інтервалу.

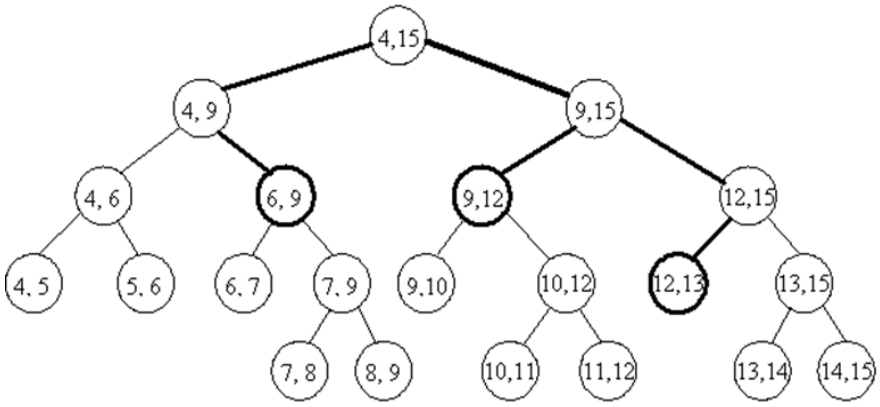


Рис. 1. 4. Дерево відрізків для інтервалу [4,15]; Вставка інтервалу [6, 13] в T(4,15). [6, 13]= [6, 9]Ú [9, 12]Ú [12, 13]. Вузли віднесення: [6, 9], [9, 12], [12, 13].

Дерево регіонів (рис. 1.5). ДР -це дворівнева структура даних, перший рівень якої – дерево відрізків по одній із координат (наприклад *x*), а другий – вузли дерева відрізків прошиті упорядкованими списками по іншій координаті (наприклад *y*). На побудованому дереві регіонів виконується операції вставки інтервалу, що дозволяє визначити вузли віднесення. Далі у вузлах віднесення застосовується дихотомія пошуку по іншій координаті (*y*).

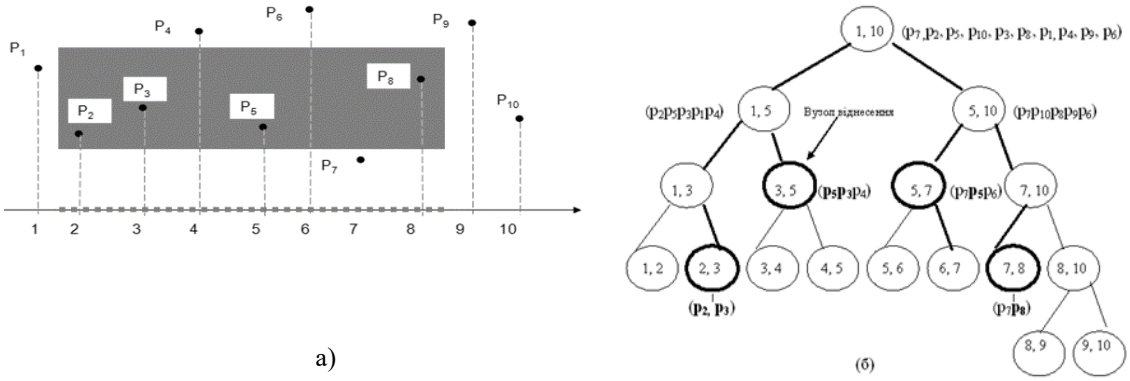


Рис. 1. 5.

Багатовимірне двійкове дерево (*k*-*d*-дерево) [5] (рис. 1. 6). *k*-*d*-дерево для множини точок – це бінарне дерево із коренем, вузлами, якого є точки заданої множини. Для його побудови застосовується рекурсивна процедура, яка на кожному кроці рекурсії почергово (по координаті *x* та *y*) визначає медіану впорядкованих списків U_x, U_y. На рис. 6 показано приклад побудови такого дерева.

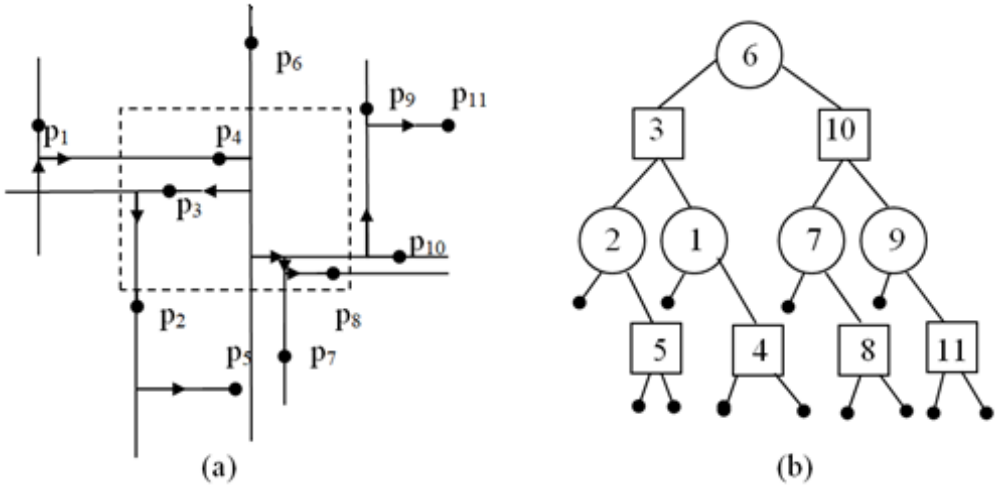


Рис.1. 6. Приклад побудови *k*-*d*-дерева

Дерево інтервалів [2, 3]. Це дерево, побудоване на множині паралельних відрізків, вузлами, якого є цілочисельні x координати медіан інтервалів утворених кінцями найлівішого (x_{min}) та найправішого (x_{max}) відрізків. На рис. 1. 7 подано приклад дерева інтервалів для відрізків паралельних осі *OX*.

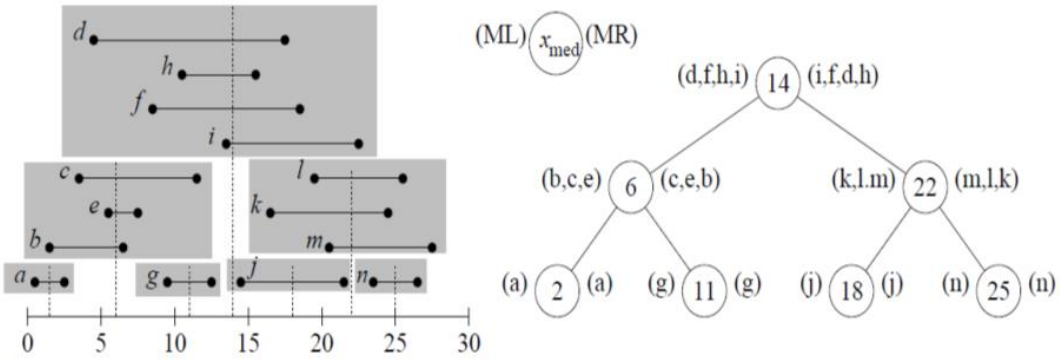


Рис. 1. 7.

Зчеплена черга [6]. Зчеплена черга - це бінарне дерево з вказівниками, вузлами якого є точки заданої множини. Многокутник можна представити у вигляді лінійного списку P ={P_5, P_1, P_2, P_9, P_{10}, P_{11}, P_{12}, P_7} і у той же час у вигляді зчепленої черги, рис.1. 8.

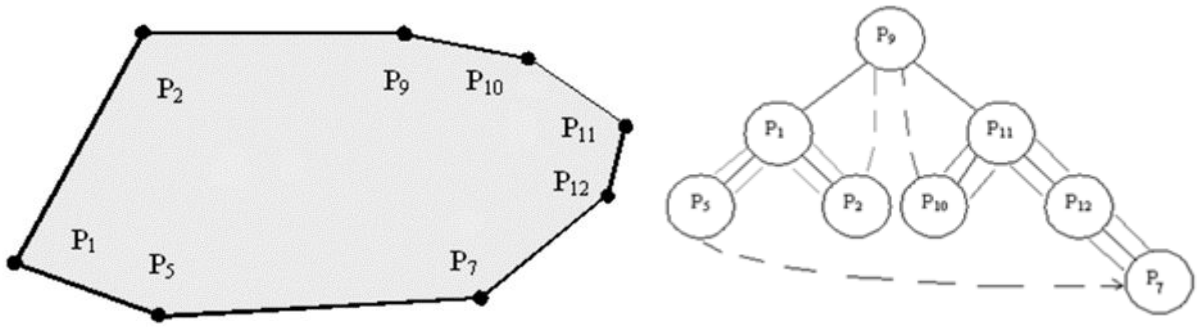


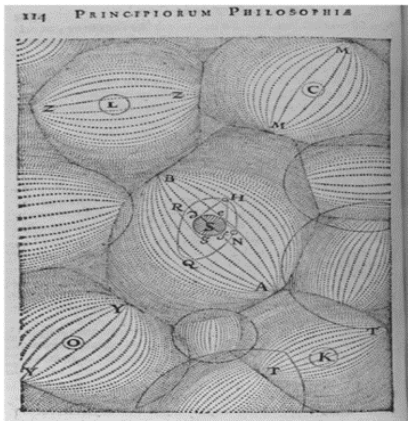
Рис.1. 8. Зчеплена черга.

1.2 ГЕОМЕТРИЧНІ СТРУКТУРИ ДАНИХ

В обчислювальній геометрії існують особливі алгоритмічні інструменти, які мають інтегральні властивості щодо їх застосування. Це такі геометричні структури, як Діаграма Вороного та Триангуляція Делоне. Вони мають ефективне застосування до широкого класу задач обчислювальної геометрії і, зокрема, до задач геометричного пошуку, а тому варто їх розглянути.

Діаграма Вороного

Історія опису (принаймні використання) цієї геометричної структури сягає ще античних часів, і зокрема, часів древнього Єгипту. Ще стародавні єгиптяни, мабуть самі того не усвідомлюючи , споруджували свою систему зрошення в долинах Нілу так, що мережа зрошувальних каналів являла собою геометрично не що інше, як Діаграму Вороного [7], при цьому ніяких особливих, принаймні відомих нам, математичних підходів вони не використовували. Про діаграму Вороного є згадка в роботах геніального Р. Декарта, видатного вченого математика, фізика, філософа[8]. В цьому напрямку працювали відомі математики Л. Дирхле [9], А. Тісен [10], описуючи області близькості для множини точок у просторі. Проте, найбільший внесок в дослідження цієї геометричної структури, безперечно, зробив Г.Ф. Вороний [11], а тому геометричне розбиття простору на області близькості, який містить задану множину об’єктів по праву носить ім’я нашого земляка – Діаграма Вороного.



Рене Декарт
(1596 - 1650)

Лежен Діріхле (1805-1859) Альфред Тіссен (1872-1956)

Георгій Феодосійович Вороний(1968-1908)

Задача (ОБЛАСТІ БЛИЗЬКОСТІ). На площині задана множина S, яка містить N точок. Необхідно для кожної точки p_i множини S визначити локус точок (x, y) на площині, для яких відстань до p_i менша, ніж до будь - якої іншої точки множини S.

Для двох точок на площині p_1 та p_2, області близькості розділяє серединний перпендикуляр до відрізка $p_i p_j$, рис.1.9. Областями близькості є півлощини $H(p_1, p_2)$ і та $H(p_2, p_1)$, відповідно. Тобто ГМТ, які ближчі до p_1 ніж до p_2 є півлощина $H(p_1, p_2)$, аналогічно $H(p_2, p_1)$.

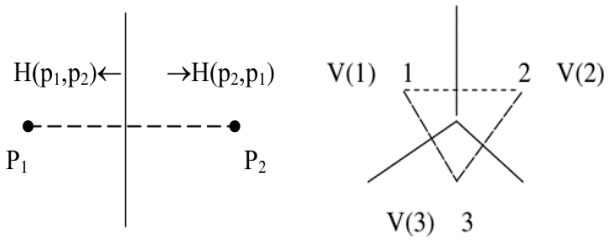


Рис.1.9. Області близькості для 2 і 3 точок.

Розглянемо області близькості для n точок. ГМТ ближчих до p_i ніж до будь-якої іншої точки, будемо позначати через V_i . Ця область є опуклим многокутником як результат перетину $n-1$ півплощин.

$$V_i = \cap H(p, p_i)$$

Означення 1.2. Область V_i називається *многокутником Вороного*, яка відповідає точці p_i. Отримані таким чином Nобластей утворюють розбиття площини, яке називається **діаграмою Вороного** (рис.1.10). Діаграма Вороного множини точок S позначається як $Vor(S)$.

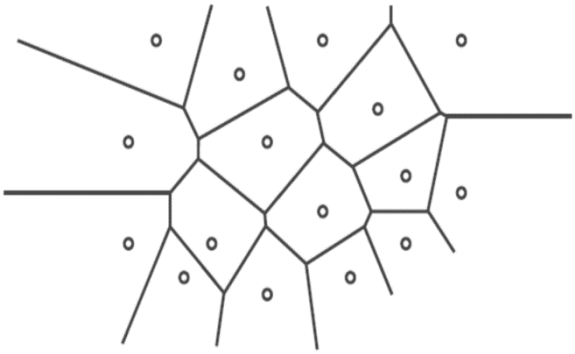


Рис.1.10. Діаграма Вороного

Властивості Діаграми Вороного

1. Кожна p_i з n вхідних точок множини S належить лише одному многокутнику Вороного. Тому, якщо точка $p(x, y) \in V_i$, то p_i є найближчим сусідом p, рис. 1.11 а, б.
2. Вершини Діаграми Вороного є центрами кіл, кожне з яких визначається точками близькості цих вершин (рис.1.11в).

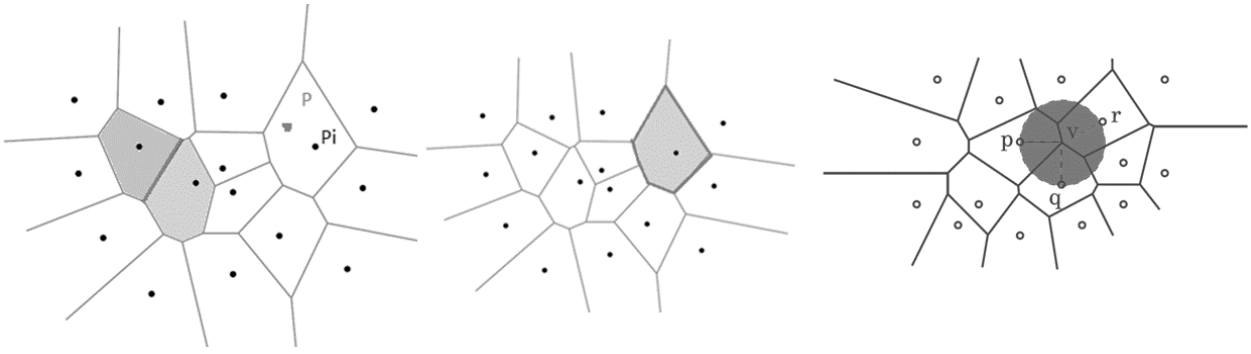


Рис.1.11. а) найближчий сусід, б) вершини ДВ є центрами кіл.

3. Кожний найближчий сусід точки p_i множини S визначає ребро у многокутнику Вороного V_i (рис.1.11 а).
4. Діаграма Вороного множини з n точок має не більш $2n-5$ вершин та $3n-6$ ребер.
5. Граф, двоїстий діаграмі Вороного, є Тріангуляцією Делоне множини S, рис. 1.12.

6. Многокутник V_i є необмеженим тоді і тільки тоді, коли точка p_i лежить на границі опуклої оболонки множини S, рис. 1.13.

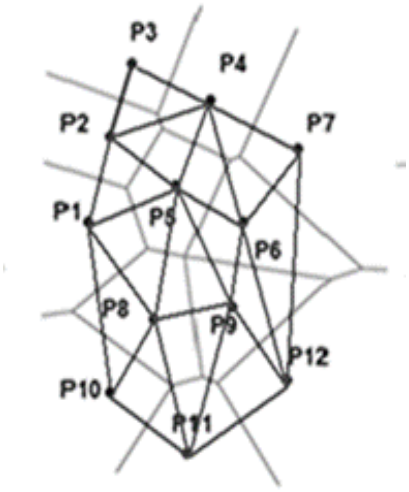


Рис.1.12.

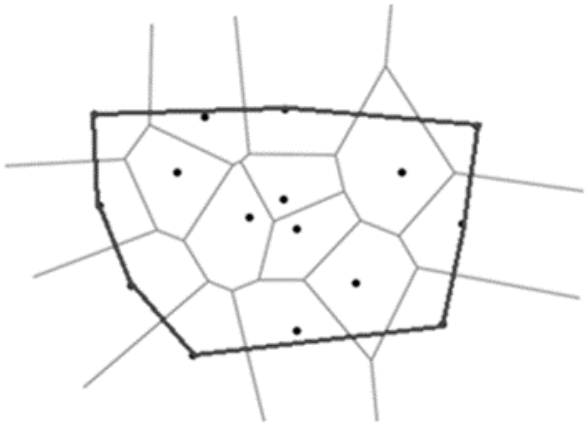


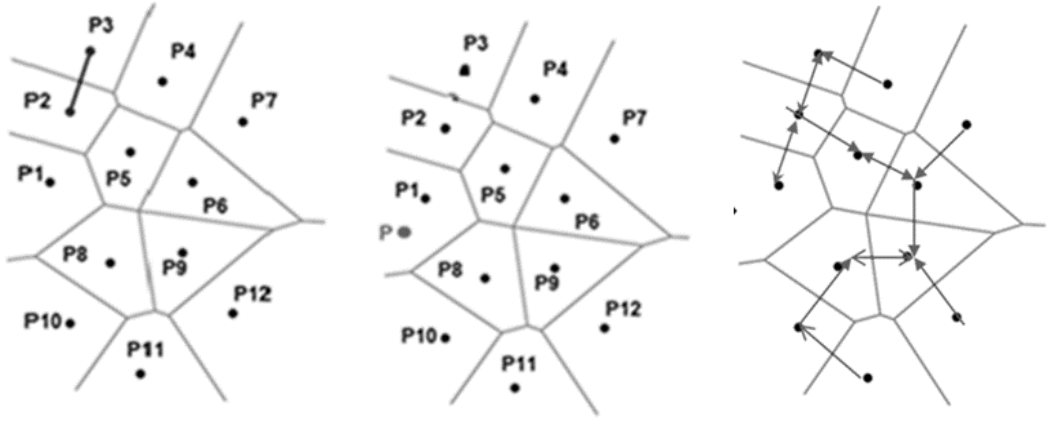
Рис.1.13.

Розв'язання задач за допомогою Діаграми Вороного

Використовуючи Діаграму Вороного, як геометричну структуру даних на етапі попередньої обробки можна розв'язати широкий клас задач обчислювальної геометрії. Має місце теорема та її наслідок.

Теорема 1.1. *Задачі: НАЙБЛИЖЧА ПАРА, ТРИАНГУЛЯЦІЯ ДЕЛОНЕ, ЕМКД, ПОШУК УСІХ НАЙБЛИЖЧИХ СУСІДІВ, ПОШУК НАЙБІЛЬШОГО ПОРОЖНЬОГО КОЛА, ОПУКЛА ОБОЛОНКА звідні за час* $O(n)$ *до ДІАГРАМА ВОРОНОГО і тому їх можна розв'язати за оптимальний час* $\theta(n\log n)$.

Наслідок теореми 1.1. *Якщо на заданій множині точок уже побудована Діаграма Вороного, то задачі НАЙБЛИЖЧА ПАРА, ТРИАНГУЛЯЦІЯ ДЕЛОНЕ, ЕМКД, ПОШУК УСІХ НАЙБЛИЖЧИХ СУСІДІВ, ОПУКЛА ОБОЛОНКА можна розв`язати за оптимальний час* $\theta(n)$ (рис. 1.12 - 1.14).



а) б) в)

Рис.1.14. Розвязання задач за допомогою ДВ: а) ПОШУК НАЙБЛИЖЧОЇ ПАРИ; б) ПОШУК НАЙБЛИЖЧОГГО СУСІДА; в) ПОШУК УСІХ НАЙБЛИЖЧИХ СУСІДІВ

Представлення Діаграми Вороного

Діаграму Вороного можна представити за допомогою РСПЗ або *k-d*-дерева, де вузли дерева визначають точки заданої множини та відповідні многокутники Вороного. На рис.1.16 подано Діаграму Вороного та *k-d*-дерево, яке її підтримує, а також пошук многокутника Вороного, який містить деяку точку *P*.

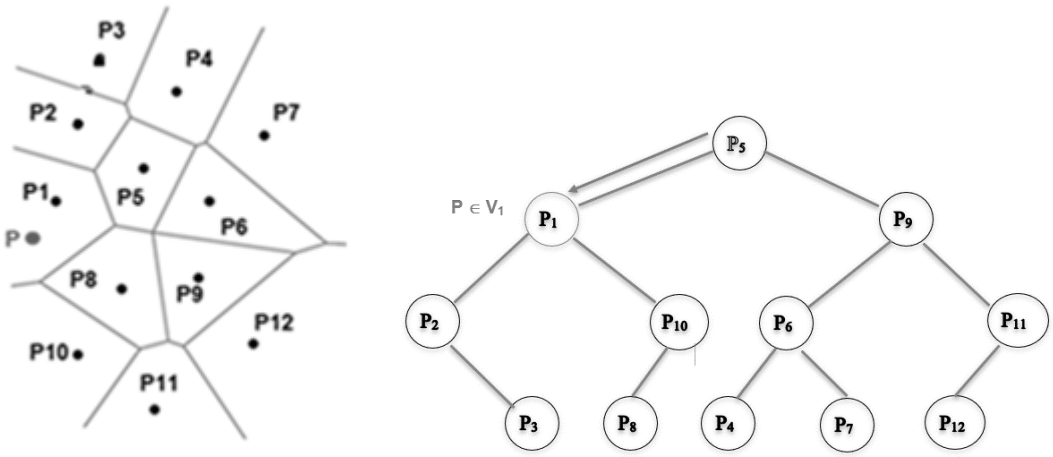


Рис.1.16. Діаграма Вороного представлена у вигляді *k-d-* дерева.

Побудова Діаграми Вороного

Для побудови Діаграми Вороного існують два основних методи: метод «розділяй та пануй»[12,14] та метод «плоского замітання» [15]. В основі методу «розділяй та пануй» лежать два основних кроки: рекурсивне розбиття заданої множини на рівнопотужні підмножини та крок рекурсивного злиття діаграм Вороного за допомогою розділяючого монотонного ланцюга (θ(*n*log*n*)), рис.1.17. Метод плоского замітання заснований ідеї замітання прямою плщини з фіксацією точок подій та статусу замітаючої прямої (θ(*n*log*n*)), рис.1.18. Існує також ще інкрементальний рендомізований алгоритм [16] .

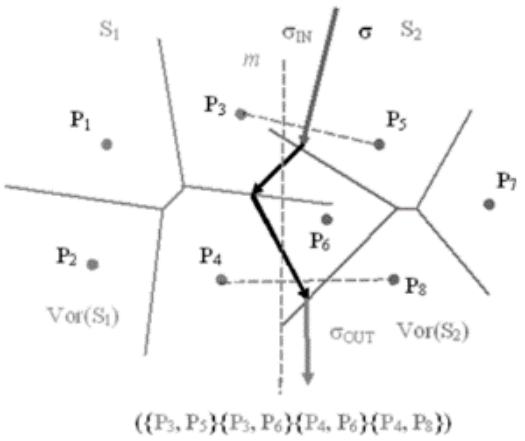


Рис.17. Метод «розділяй та пануй»

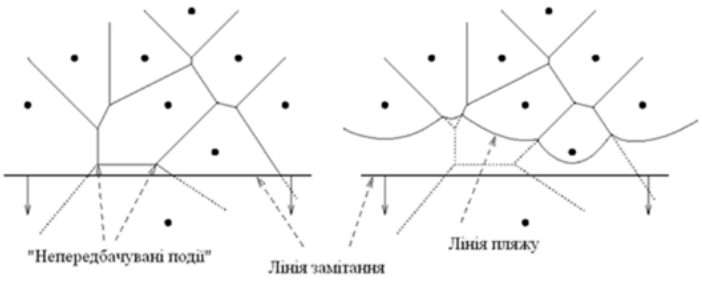
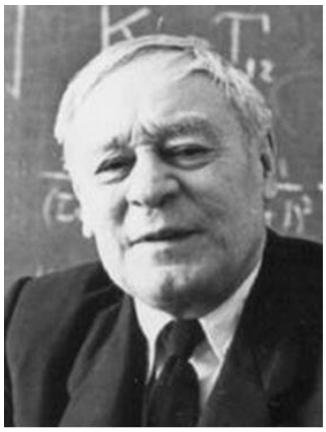


Рис.18. Метод плоского замітання

Триангуляція Делоне

Розглянемо, ще одну популярну геометричну структуру, яка поряд з Діаграмою Вороного теж має інтегральні властивості щодо застосування. Із властивості 5 Діаграми Вороного (ст. 14., рис.1.12.) триангуляція Делона є двоїстим графом до ДВ. Почнемо із означення. Автором триангуляції Делоне вважається Б.Н. Делоне, який у 1934 році вперше описав цю геометричну структуру [17].



Ніколай Борисович Делоне (1890 - 1980)

Означення 1.3. Нехай задана множина *S* із *n* точок площині. Нехай CH(*S*) границя опуклої оболонки *S*. Тоді триангуляція множини точок *S* обмеженої границею опуклої оболонки CH(*S*) називається **Триангуляцією Делоне (ТД)**, якщо вона задовольняє умові Делоне: кола описані навколо усіх її трикутників не містять жодної точки заданої множини *S* (рис.1.19.).

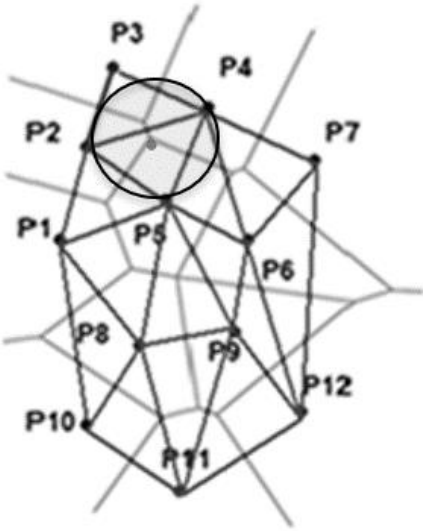


Рис.1.19. Триангуляція Делоне.

Властивості Триангуляції Делоне

Нехай *n* -кількість точок, а *d* – розмірність простору. Маємо наступні властивості.

1. Триангуляція Делоне є двоїстим графом Діаграми Вороного і навпаки, рис.1.20.
2. Коло описане навколо довільного трикутника ТД не містить всередині інших точок заданої множини *S*, рис.1.20.
3. Якщо коло, що проходить через дві вхідні точки не містить всередині жодних інших, тоді ребро, що з'єднує ці дві точки є ребром тріангуляції Делоне цих точок.

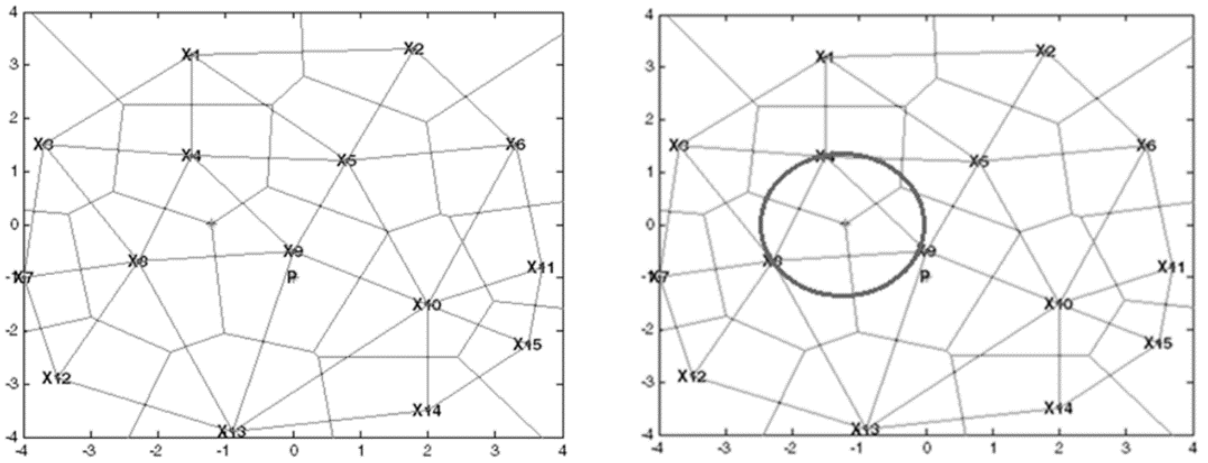
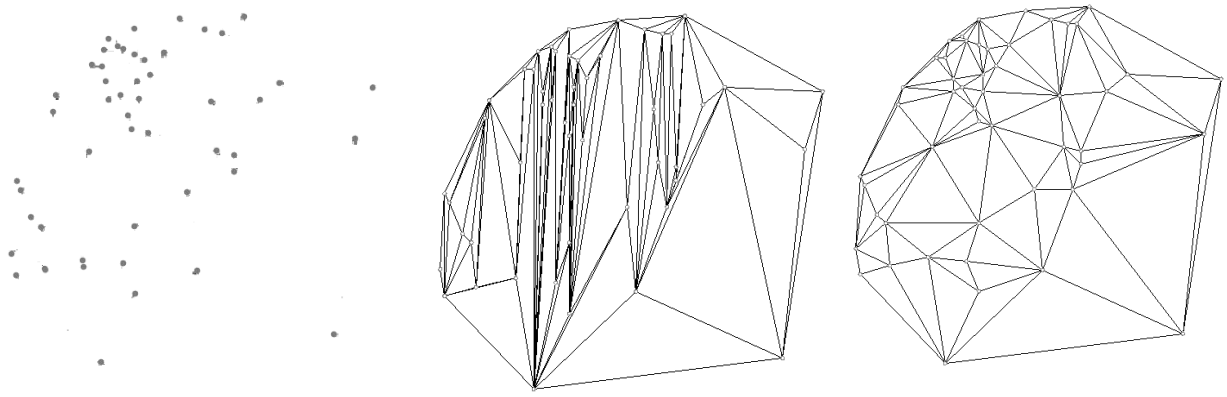


Рис.1.20.

4. Об’єднання всіх симплексів у тріангуляції Делоне є опукла оболонка заданої множини *S* із *n* точок.
5. На площині кожна вершина має в середньому 6 інцидентних трикутників.
6. На площині ТД максимізує найменший кут. Найменший кут у ТД буде не меншим ніж у будь-якій іншій тріангуляції. При цьому ТД не обов’язково мінімізує максимальний кут. Ця властивість дозволяє в задачах обробки зображення (розфарбування, рендерінг), інтерполяції та інтегрування за найоптимальнішу тріангуляцію обирати саме ТД, рис. 1.21.



а) б) в)

Рис.1.21. а) вхідна множина точок; б) неякісна тріангуляція (довгі і вузькі трикутники);
в) якісна тріангуляція (без малих кутів, майже правильні трикутники)- Тріангуляція Делоне.

7. Якщо на площині (d=2), b вершин входять до опуклої оболонки, то будь яка тріангуляція точок має що найбільше $2n-2-b$ трикутників, іще одну зовнішню грань.

8. Тріангуляція Делоне містить щонайбільше $O(n^{[d/2]})$ симплексів.

9. ТД множини точок в d-вимірному просторі є проекцією точок опуклої оболонки на d+1 -мірний параболоїд.

10. Найближчий сусід b довільної точки p лежить на ребрі bp тріангуляції (бо граф найближчих сусідів це підграф ТД).

11. **Теорема 1.** Тріангуляція Делоне має максимальну суму мінімальних кутів усіх своїх трикутників серед усіх можливих тріангуляцій.

12. **Теорема 2**. Тріангуляція Делоне має мінімальну суму радіусів кіл, описаних навколо трикутників, серед усіх можливих тріангуляцій.

Представлення Триангуляції Делоне

Для представлення Триангуляції Делоне існує багато структур даних, які детально представлені в [2] і пов'язанні з вибором основних типів об'єктів: *вузли* (точки, вершини), *ребра* (відрізки) і *трикутники*. Так, наприклад, Триангуляцію Делоне як і Діаграму Вороного можна представити за допомогою РСПЗ або графа із коренем, вузли якого трикутники триангуляції, рис. 1.22.

Побудова Триангуляції Делоне

З побудовою Триангуляції Делоне набагато простіше. Існує багато методів для побудови звичайної триангуляції [18], які можна застосувати до побудови Триангуляції Делоне за допомогою наступної теореми.

Теорема 1.2. *Триангуляцію Делоне можна отримати з будь-якої іншої триангуляції на тій же множині точок, послідовно перебудовуючи пари сусідніх трикутників ΔABC і ΔBCD, які не задовольняють умові Делоне у пари трикутників ΔABD і ΔACD, рис.* 1.23*. Така процедура називається фліпінгом.*

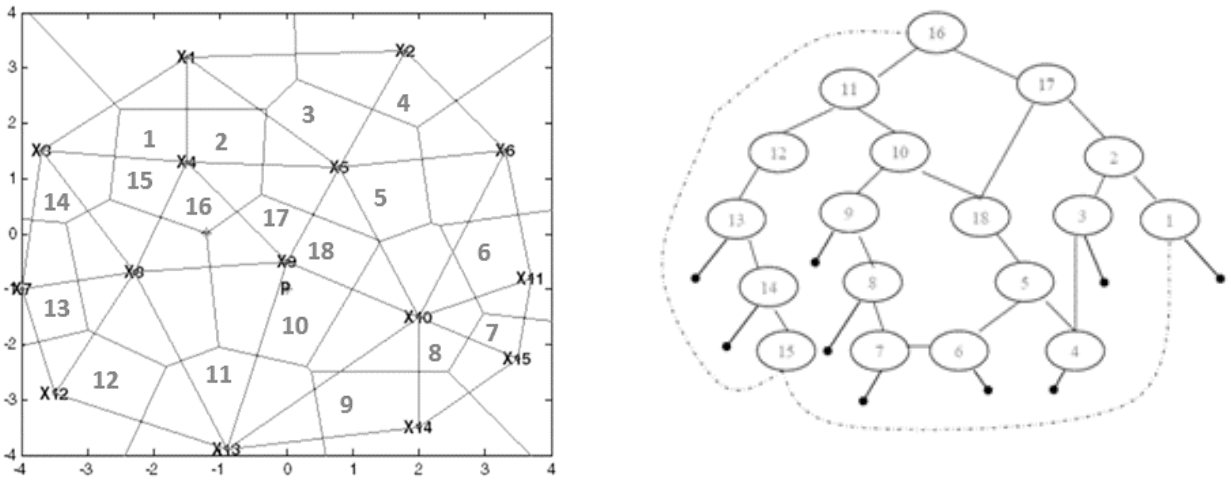


Рис.1.22. Представлення Триангуляції Делоне

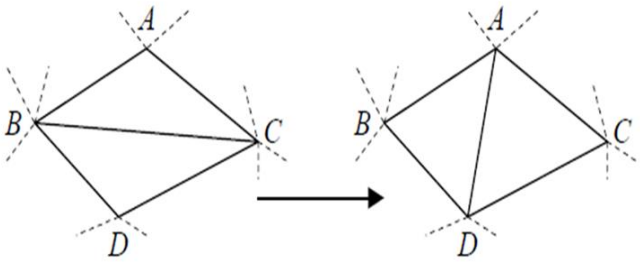


Рис.1.23. Фліпінг.

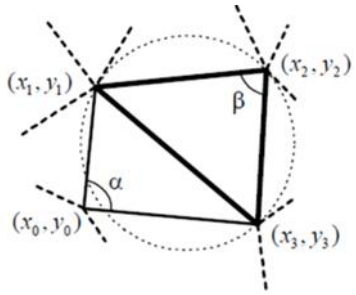


Рис. 1.24.

Існує декілька способів перевірки умови Делоне:

1. Перевірка за допомогою рівняння описаного кола.

$$[(a(x_i + y_i)^2 - bx_i + cy_i - d]\text{sign}a \geq 0 \qquad (9.1)$$

2. Перевірка із попередньо обчисленого описаного кола:

$$(x_i - x_c)^2 + (y_i - Y_c)^2 \geq r^2 \qquad (9.2)$$

3. Перевірка суми протилежних кутів, рис.1.24: $\alpha + \beta \leq \pi$

Один із простих методів триангуляції Делоне подано в роботі [19] на основі методу Грехема. Суть його полягає у наступному. Починаючи із заданої множини S із n точок будуємо обходом Грехема послідовно опуклі оболонки CH(S), CH(S_1), CH(S_2) …, CH(S_k) для множин S, S_1, …, S_k., рис.25.

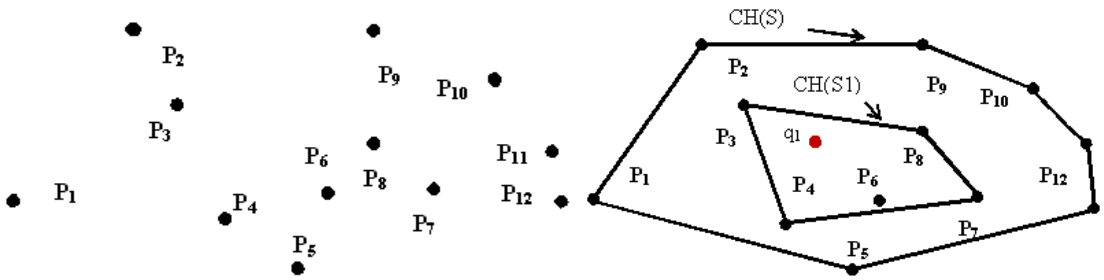


Рис. 1.25.

Таким чином, сусідніми границями опуклих оболонок утворяться шари, які ми можемо послідовно триангулювати, рис.1.26.

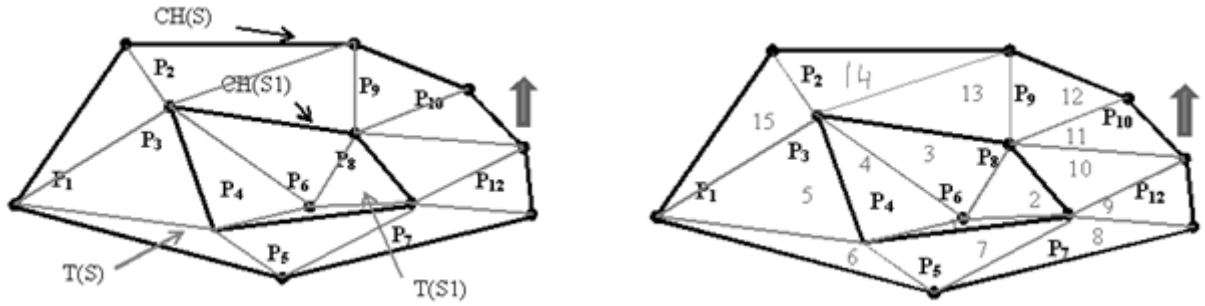


Рис. 1.26.

Позначивши грані триангуляції (цілими цифрами) можемо представити їх у вигляді зчепленої черги (рис. 27,а) чи дерева граней (рис. 1.27.б) . А далі, застосуваши до одержаної триангуляції (по ходу триангуляції) фліпінг, можемо побудувати триангуляцію Делоне.

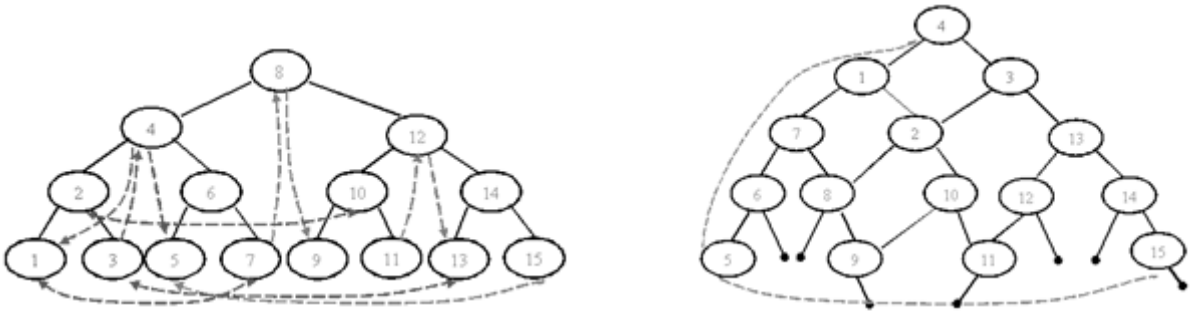


Рис.1.27. Представлення триангуляції: а) у вигляді зчепленої черги; б) у вигляді бінарного дерева граней.

2. ЗАДАЧІ ГЕОМЕТРИЧНОГО ПОШУКУ ТА МЕТОДИ ЇХ РОЗВЯЗАННЯ

Різноманіття задач геометричного може бути досить широким, проте виділяють два основних класи: задачі локалізації точки та задачі регіонального пошуку. Сформулювати їх можна таким чином:

Задача ГП1 (ЛОКАЛІЗАЦІЯ ТОЧКИ). Нехай задане n – вершинне розбиття $G(V, E)$ евклідового простору E^d (у випадку двовимірного простору – планарний граф) і точка P. Визначити область розбиття, яка містить точку P (Локалізувати точку P на заданому розбитті).

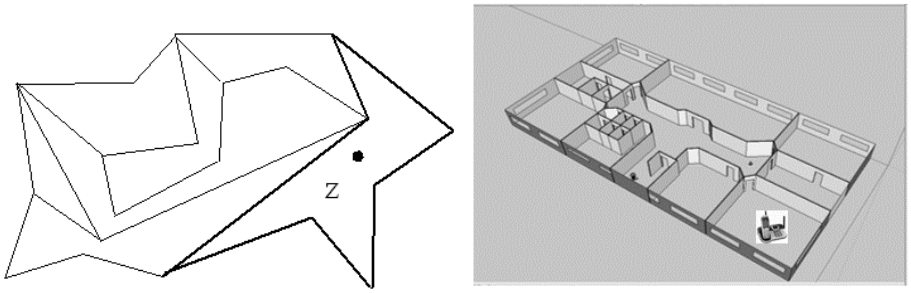


Рис.2.1. Постановка задачі локалізації точки

Таким чином у задачі ГП1 файл являє собою розбиття геометричного простору на області, а запитом є точка P, Рис.2.1. Локалізація полягає у визначені області, яка містить шукану точку.

Задача ГП2 (РЕГІОНАЛЬНИЙ ПОШУК). На заданій множині S із n точок задано запитний регіон R. Знайти підмножину точок множини S (або їх кількість), які містяться в регіоні R.

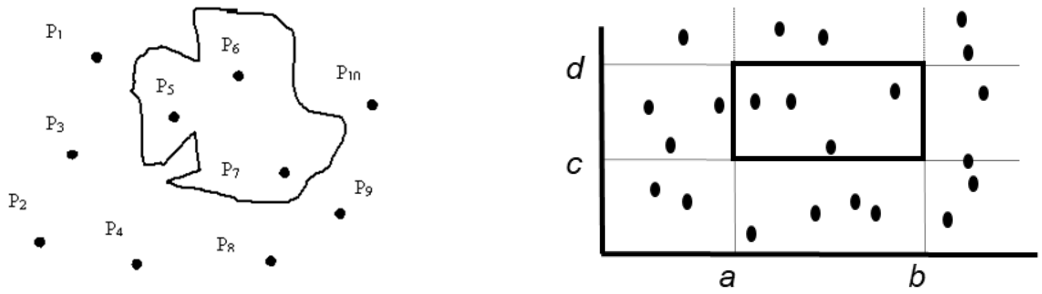


Рис.2.2. Регіональний запит.

Файл в задачі ГП2 являє собою набір точок простору, а запитом є деяка стандартна геометрична фігура, яка довільно переміщується у цьому просторі (типовий запит у 3-вимірному просторі це куля або брус). Регіональний пошук полягає у визначені (задачі звіту) або у підрахунку кількості (задача підрахунку) усіх точок в середині заданого регіону, рис.2.2.

Нижні оцінки складності задач геометричного пошуку

Для встановлення нижніх оцінок складності задач геометричного пошуку використовується метод звідності [6]. У цьому випадку задачею прототипом, яка задовольняє умовам звідності, буде задача бінарного пошуку на множині n дійсних чисел, нижня оцінка складності якої відома і рівна $\Omega(\log n)$ [6].

Теорема 2.1. *Задача бінарного пошуку розмірності* n *зводиться за час* $O(1)$ *до задачі локалізації точки та задачі регіонального пошуку, а тому нижня оцінка складності задачі локалізації точки та задачі регіонального пошуку рівна* $\Omega(\log n)$.

Доведення (для локалізації точки). Нехай задана множина цілих чисел $A = \{ x_1, x_2, \ldots, x_N \}$ і число P. Дійсно, БІНАРНИЙ ПОШУК $\propto o(1)$ ЛОКАЛІЗАЦІЯ ТОЧКИ так, як вхідні дані БП можна відразу зобразити у вигляді упорядкованого списку точок на осі OX: $[\{ x_1, x_2, \ldots, x_N \}, P] \rightarrow [\{ (x_1,0), (x_2,0), \ldots, (x_N,0)\}, (P_x, 0)]$. Результатом розв'язання задачі ГП1 буде інтервал із двох сусідніх точок $[(x_i,0), (x_{i+1},0)]$**,** між якими міститься точка $P(P_x, 0)$, а це визначає вірний результат задачі бінарного пошуку.

Аналогічно доводиться звідність для задачі регіонального пошуку.

2.1 ЛОКАЛІЗАЦІЯ ТОЧКИ

Задача локалізації точки (ЗЛТ) є однією з основних тем обчислювальної геометрії, а з розвитком сучасних ІКТ технологій і, зокрема, технологій штучного інтелекту ЗЛТ стала важливою в задачах комп'ютерного зору та задачах розпізнавання зображень.

Отже для розв'язання задачі локалізації точки нам необхідно запропонувати стратегію, яка б дозволяла розробляти алгоритми із складністю близькою до $O(\log n)$ чи $\theta(\log n)$. Для цього слід провести аналіз властивостей вхідних та вихідних даних.

ДВОВИМІРНИЙ ВИПАДОК У E^d (d=2)

Постановка задачі. Для заданого n-вершинного планарного розбиття на площині локалізувати точку P.

Аналіз вхідних даних та результатів розв'язання.

Розглянемо приклад поданий на рис.2.1.

Клас задач: Задачі геометричного пошуку (ГП), **підклас** – задача локалізації точки.

Вхід : n- вершинний планарний граф(задається у вигляді $G(V, E)$), точка P. **Вихід:** многокутник (у вигляді списку вершин чи ребер), який містить P. Маємо для найгіршого випадку: кількість многокутників – $O(n)$; кількість сторін многокутника – $O(n)$.

Якщо спробувати розв'язати задачу перебором, тобто для кожного із n многокутників перевіряти належність точки P k- кутнику, то отримаємо час пошуку $O(n^2)$, що є далеко від логарифмічної складності. Нам необхідно розробити стратегію, яка б давала час пошуку близький до логарифмічного. Відомо, що найпростіший алгоритм перевірки належності простому многокутнику дає час $O(n)$ [6]. Аналізуючи вхідні та вихідні дані, можна зробити висновок, щоб досягти бажаного результату нам необхідно перейти від перевірки належності точки многокутнику за $O(n)$ до перевірки за константний час $O(1)$, а $O(n)$ многокутників упорядкувати так, щоб отримати логарифмічний вибір.

Таким чином стратегія розробки алгоритмів локалізації точки на планарному розбитті геометричного пошуку базуватиметься на підрозбитті заданого планарного графа $G(V, E)$ на множину упорядкованих простих k-кутних фігур (смуг, трикутників, чотирикутників, k-ланцюгів, тощо), де перевірка належності точки рівна $O(1)$.

У роботах [6,20-22] описані основні ефективні методи та алгоритми, які базуються на вказаній стратегії. Зокрема варто виділити такі методи: метод смуг(МС) [23,24], метод монотонних ланцюгів (ММЛ) [25], метод деталізації триангуляції (МДТ) [26] та метод трапецій (МТ)[27], які дають оптимальні оцінки по часу і пам'яті, відповідно: МС ($O(\log n, O(n\log n)$), ММЛ ($O(\log^2 n, O(n)$), МДТ($O(\log n, O(n\log n)$), МТ($O(\log n, O(n\log n)$). Алгоритми, що реалізують ці методи є найбільш вживаними при розв'язанні задач локалізації точки [20], особливо це стосується ММЛ, який також успішно застосовується для розв'язання інших класів задач [28-31].

Аналіз останніх досліджень.

Перші оптимальні рішення були представлені Ліптоном і Тар'яном [32] та Кіркпатріком [26]. Метод Ліптона-Тар'яна заснований на теоремі сепаратора графу, і досі представляє лише теоретичний інтерес. Метод деталізації тріангуляції [26], який будує ієрархію тріангуляцій, можна реалізувати, але він має занадто великі константи. Окрім того, жоден з цих методів не поширюється на планарні розбиття з криволінійними ребрами.

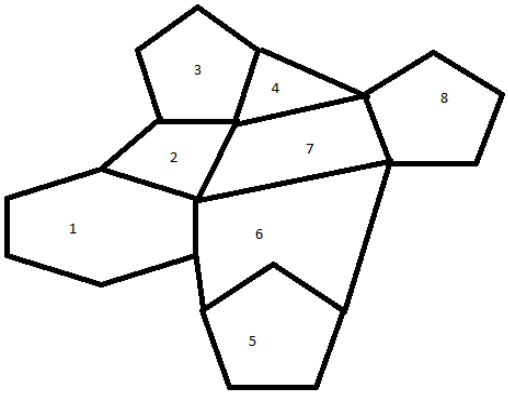
Історично, Добкін і Ліптон [23] були першими в досягненні $O(\log n)$ оцінки часу запиту, при цьому використовуючи $O(n^2)$ пам'яті). Їхній метод, що отримав назву «метод смуг» був згодом покращений Препаратою [27] так, що оцінка пам'яті стала $O(n\log n)$. Пізніше Білардо і Препарата [33], ще раз покращили оцінку складності пам'яті в середньому до $O(n\log n)$. Окрім цього, ці методи мають позитивну рису: вони можуть бути застосовані до розбиття з криволінійними ребрами. Істотно інший підхід був розглянутий Шеймосом [6], що привів до відомої роботи про проблему локалізації точки Лі і Препарати [34] заснованої на побудові розділяючих ланцюгів. Час попередньої обробки побудови цієї структури даних складає $O(n\log n)$ з використанням пам'яті $O(n)$ в гіршому випадку, та часом пошуку $O(\log^2 n)$.

Також варто згадати оригінальні ідеї, щодо розробки ефективних алгоритмів локалізації точки із залученням теорії комплексної змінної та апарату аналітичних функцій [79]. В статті [80] пропонується оригінальний метод локалізації точки на площині із несподіваним застосуванням теорем Кощі про лишки із теорії функцій комплексної змінної. В результаті відповідь на запит про приналежність точки зводиться до обчислення деякої величини.

2.1.1 Алгоритм покрокового злиття підобластей

Трудомісткість розв'язання цієї задачі суттєво залежить від природи простору і способу його розбиття. Особливістю запропонованого алгоритму є покрокове злиття оболонок і побудова структури даних, по якій відбувається пошук. Це дозволяє виконувати запит на локалізацію точки за час $O(\log^2 n)$ і з використанням пам'яті $O(n \log n)$.

Попередня обробка та структура даних: Будуємо структуру даних таким чином. Наш граф являє собою множину областей, які з'єднані між собою гранями. Області в свою чергу складаються з множини точок і границь, рис.2.3.



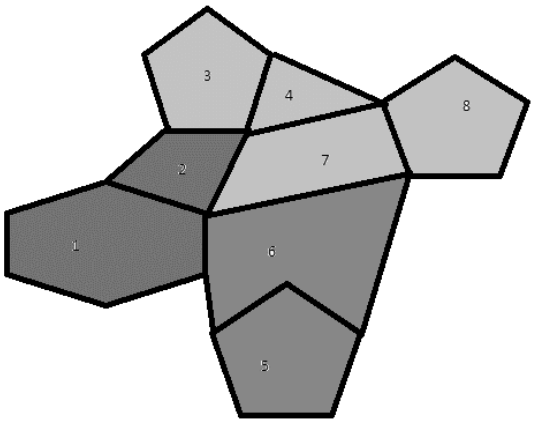
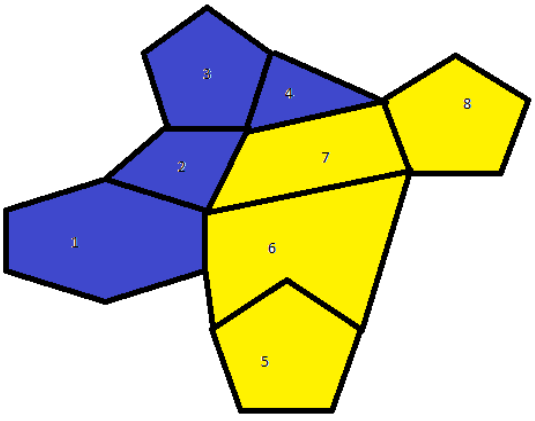


Рис.2.3. Рис.2.4. Області після 1 проходу

1. Робимо 1-ий прохід – Зливаємо 2 сусідні області, в одну об'єднану область. При цьому, спільні грані, які присутні одночасно в обох областях, що з'єднуються, видаляються, після цього переходимо до наступної пари областей. Так попарно зливаємо всі області, рис.2.4.

2. Якщо після такого проходу залишаться області, які не були злиті, зливаємо їх із уже одержаними областями, рис.2.5.



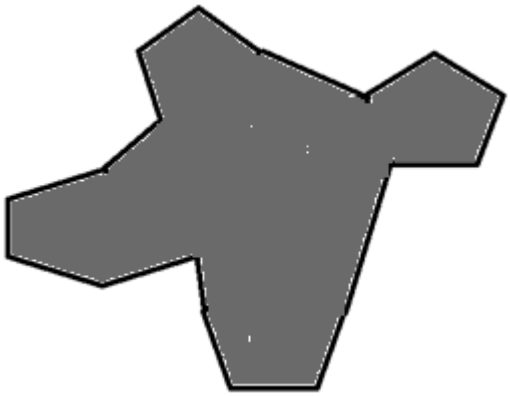


Рис. 2.5. Обл. після 2-го проходу

3.Злиття продовжуємо доки не отримаємо єдину область.

4.Під час злиття будуємо відповідну структуру даних, рис.2.6

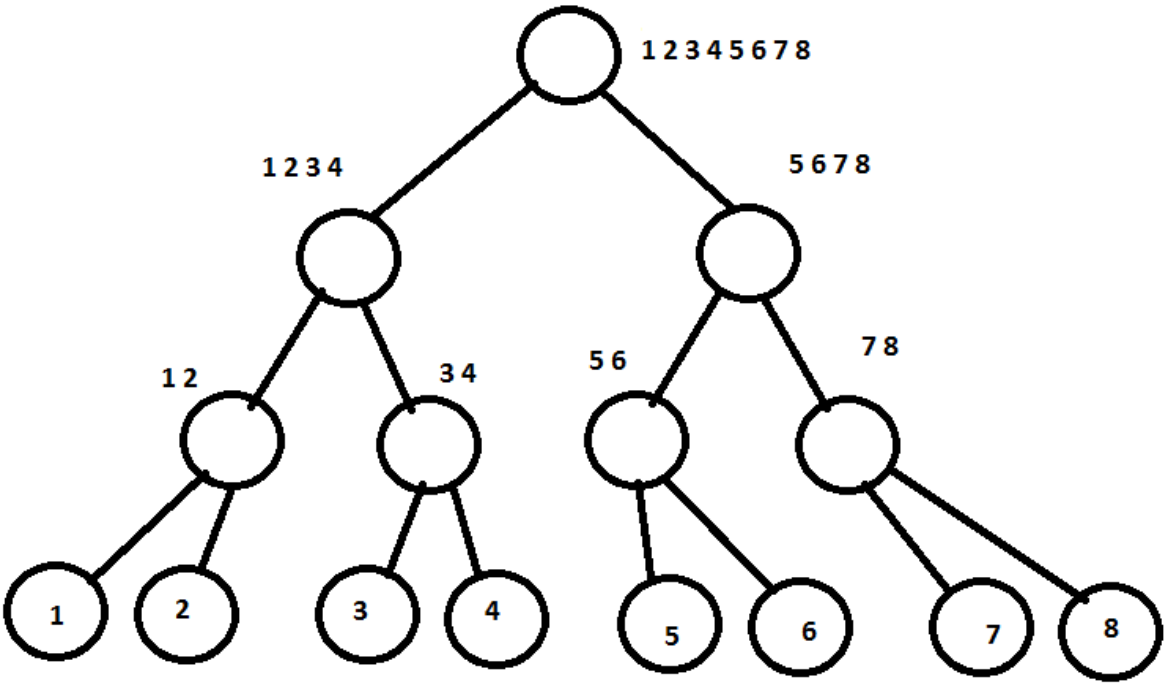


Рис 2.6. Структура даних.

Наша структура даних буде містити оболонки (множини точок із гранями) по якій буде проводитися пошук.

Пошук: По структурі даних ми рухаємося з вершини до листків.

1. Перевіряємо чи точка *P* належить оболонці [6 ст.42], яка представлена у нашій структурі даних на вершині (Приклад *S* (1 2 3 4 5 6 7 8)) чи ні.

2. Якщо точка не належить тоді алгоритм закінчується з висновком що точка поза межами нашого графа, якщо ні рухаємося далі.

3. Рухаємося по нашому дереву вниз і перевіряємо до яких з двох (або більше) оболонок належить точка *P* (Приклад S_1(1 2 3 4) чи S_2(5 6 7 8)). використовуючи «Належність точки до оболонки»

4. Так ми рухаємося поки не досягнемо листка нашої структури тобто перевірка належності точки буде проводитися лише для одної оболонки.

Обґрунтування складності алгоритму

Теорема 2.2 *Належність точки P внутрішній оболонці CH(S) можна встановити за час O*(*n*) *без передобробки.* (*Для* 2*d випадку належність точки P внутрішній області простого n-кутника P можна встановити за час O*(*n*) *без передобробки).*

Доведення. Оскільки ми використовуємо нашу структуру даних у вигляді дерева то пошук реалізується за час $O(\log n)$. Часу пішло на попередню обробку $O(\log n)$. Отже статичну локалізацію точки у прямолінійному 3*d* графі можна реалізувати за час $O(\log n)$ з використанням $O(n)$ пам'яті, якщо $O(n\log n)$ часу пішло на попередню обробку.

Висновки

Особливістю запропонованого алгоритму є покрокове злиття оболонок і побудова структури даних, по якій відбувається пошук. При проходженні по дереву відбувалася перевірка належності точки до оболонки. В основі цієї перевірки використалась ідея підрахунку кількості перетинів променя, який виходить із точки *P* з гранями оболонки. За допомогою попередньої обробки було створено структуру даних, яка дозволяє виконувати запит на локалізацію точки за час $O(\log^2 n)$ і з використанням пам’яті $O(n\log n)$

БАГАТОВИМІРНИЙ ВИПАДОК У E^d ($d \geq 3$)

Постановка задачі. Для заданого n-вершинного розбиття евклідового E^d простору локалізувати точку *P*.

Варто зазначити, що деякі із методів, розроблених для випадку d=2 мають своє розширення для вищих розмірностей. Проте для вищих вимірів досить складно вийти на оптимальні оцінки складності по часу і памяті із за можливої складності розбиття простору. Так відомі методи локалізації точки у загального випадку не досягають одночасно лінійності пам'яті та логарифмічного часу запиту. Лише в окремих випадках можна одержати оптимальний результат.

У роботі [23] автори запевняють, що логарифмічний час пошукового запиту може бути отриманим проекціюванням граней підрозбиття на комірки. Найгірший випадок вимагає $O(n^{2^d})$ пам'яті. Тамассія та Гудріх запропнували прямий метод [35], який підтримує необхідні операції щодо забезпечення структури даних локалізації точки для розгорнутої площини. Використання копії вузлів, для створення стійкої структури (вимагає $O(n\log n)$ об'єму пам'яті) дозволяє отримати час запиту $O(\log^2 n)$ з O(nlogn) часом попередньої обробки. В роботі [36] автори пропонують кілька підходів тетраедрізації Делоне та встановили практичну ефективність ієрархічного підходу для 3D.

У випадку більш простіших розбиттів, зокрема ортогональних, інколи вдається досягти оптимальних результатів пошуку. Так для ортогонального d-вимірного розбиття авторам [37] вдалося побудувати багатовимірне дерево відрізків і локалізувати точку для $O(n)$ прямокутників за $O(\log^{d-1} n)$ часу, $O(n)$ пам'яті та $O(n\log n)$ попередньої обробки. Їх можна зробити динамічними, використовуючи структуру Гійори та Каплана [38] на найнижчому рівні.

В роботі [39] для випадку d = 2, 3] автори використали так зване стратифіковане дерево та досконале хешування для одержання $O(\log\log U)^{d-1}$ часу запиту у фіксованому просторі та $O(\log n)$ в загальному випадку.

2.1.2 Алгоритм 2

Розглянемо ще один підхід до розв'язання нашої задачі, який дає час пошуку $O(\log(n))$ та попередньої обробки $O(n\log(n))$, де n – число підмножин підрозбиття. Хоча, швидкість алгоритму пошуку є основним критерієм ефективності алгоритму, проте існують також інші важливі міри ефективності алгоритму такі, як попередня обробка, обсяг пам'яті та зручність реалізації.

Велика частина алгоритмів узагальнюють пласку локалізацію або базуються на проекції підрозбиття на площину, тобто зниженні розмірності. Наприклад, узагальнення алгоритму Кіркпатріка на тривимірний випадок [40]. Автори використовують зв'язаний список вершин для зображення тетраедралізації тривимірного об'єкту. При цьому кожен відрізок має доступ до граней, яким належить, а грані в свою чергу до регіонів з "правої" та "лівої" сторони. Але тут постає головний недолік більшості алгоритмів просторової локалізації, а саме, залежність від способу розбиття. Описаний вище метод використовує тетраедралізацію, а тут представлено алгоритм лише для опуклого підрозбиття. Ще одним прикладом є локалізація у d-вимірному розбитті прямокутниками [37]. В роботі [37] використовується структура під назвою *дерево скосу* (англ. *Skewer Tree*), що зберігає d-вимірні прямокутники. Такий підхід дозволив отримати пошук за $O(\log^d(n))$ і побудову за $O(n\log(n))$. Саме таку асимптотику має більшість алгоритмів просторової локалізації.

Зустрічається використання перманентних (англ. *persistent*) структур даних. Це означає, що після проведення деякої операції минулий стан зберігається, тобто є доступ до всіх або декількох минулих версій. Наприклад, в результаті надання структурі перманентності було отримано пришвидшення алгоритму локалізації в опуклому підрозбитті [41] з $O(\log^2(n))$ до $O(\log(n))$. У представленому алгоритмі один із вимірів розглядається як час, а далі використовується метод рухомої площини: коли вона натрапляє на вершину, то вершина додається до структури, що створює нову копію. Таким чином, зберігається планарний граф для кожного з проміжків, що потім і використовується для пошуку. Перманентність хоч і зберігає минулі копії, але економить пам'ять, що і є перевагою даного алгоритму. Введемо деяку формалізацію задачі для багатовимірного випадку.

Означення 2.1. *Опуклим підрозбиттям простору E^d називається множина внутрішніх непересічних d-вимірних опуклих півпросторів.* [42]

Властивість 2.1. *Якщо многогранник опуклий, то він повністю лежатиме в одному з півпросторів відносно будь-якої площини, що проходить через його грань.*

Постановка задачі. Нехай задано опукле підрозбиття тривимірного простору E^3 з N регіонами, H гранями та n вершинами. Необхідно для заданої точки знайти регіон, якому вона належить.

Тут під терміном регіон мається на увазі многогранник підрозбиття.

Алгоритм

1. Попередня обробка. Побудова структури даних.

Розглянемо такі об’єкти:
регіон: зберігає інформацію про свої грані
грань: має список відрізків і регіонів, яким належить
відрізок: має кінцеві точки та грані, яким належить
точка: має координати та відрізки, яким належить

Ідея. Маючи конкретний регіон, можна за гранями віднайти його сусідів. Це і є ключовою особливістю представленої структури. Вона являє собою граф з регіонами у вершинах. Дві вершини будуть зв’язані між собою тоді і тільки тоді, коли їх регіони матимуть спільну грань. Алгоритм локалізації полягає в тому, щоб з деякої початкової (випадкової або ж конкретної кореневої) вершини рухатись за гранями у напрямку до точки. Представлений підхід має деякі недоліки:

- у випадку незв’язного графа залишається нерозв’язаною проблема вибору потрібної компоненти зв’язності;
- не завжди існує прямий шлях до шуканого регіону з початкового (наприклад, наявність “дірок”), що може сповільнити алгоритм, а також ускладнити реалізацію.

Тому далі буде представлена модифікація як структури, так і відповідного алгоритму.

2. Покращення алгоритму.

На цьому етапі буде запропоновано удосконалення алгоритму. Замість лише прямого сусіда будуть обиратися усі регіони, що знаходяться за напрямом до точки, поки не отримаємо один конкретний регіон. Нова структура буде представляти собою бінарне дерево розбиття простору з розділяючими площинами у вузлах та регіонами у листових вузлах.

Алгоритм побудови нової структури даних:

1) кореневий вузол містить усе підрозбиття;
2) обирається деякий регіон і його деяка грань - вона і буде задавати площину розбиття;
3) наявні регіони діляться на 2 групи за відносним розташуванням до площини: “знизу” та “зверху”;
4) дві групи рекурсивно діляться навпіл у той же спосіб;
5) алгоритм побудови СД закінчується, коли поточна група буде містити лише один регіон.

Тут прямо використовується властивість 1 опуклих многогранників, з якої слідує, що на кожному кроці одна з підмножин міститиме як мінімум на один регіон менше. Неоднозначність може виникнути, якщо є регіони, що перетинають роздільну площину. Цю проблему можна вирішити двома способами:

1. Розділяти аномальний регіон площиною на два нові;
2. Вносити аномальний регіон в обидві множини і, якщо одна з них не змінила своєї потужності, обрати іншу розділяючу грань наступного разу.

Вибір регіону на кожному кроці може бути як випадковим, так і більш раціональним, з урахуванням того, щоб поточна підмножина підрозбиття ділилась приблизно порівну.

3. Алгоритм локалізації точки.

Отримана деревовидна структура буде використовуватися для локалізації у такий спосіб:

1) пошук починається з кореневого вузла;
2) визначається відносне розташування шуканої точки до площини розбиття та обирається наступний вузол;
3) пошук продовжується аналогічним шляхом, поки не буде отримано конкретний регіон;
4) перевіряється належність точки знайденому регіону.

Алгоритм для останнього пункту можна знову ж вивести з властивості опуклого многогранника. Якщо точка належить регіону, то для кожної з граней вона лежить з тієї ж сторони, що й усі вершини. Введемо операції додавання та видалення регіонів: *Insert* та *Remove*.

Алгоритм *Insert*:

1) за однин обхід знаходиться потрібний листовий вузол;
2) створюється нова роздільна площина і відповідний їй вузол, що додається за мість знайденого у минулому кроці;
3) знайдений регіон і новий, доданий, стануть його нащадками.

Алгоритм *Remove*:

1) обходом знаходиться потрібний листовий вузол, що містить шуканий регіон;
2) листовий вузол видаляється;
3) замість батьківського видаленого вузла ставиться його нащадок.

Таким чином, при додаванні і видаленні зберігається структура і повнота дерева.

Обґрунтування складності алгоритму

Терема 2.3. *Задачу локалізації точки на n вершинному розбитті у просторі* E^d *можна виконати за час* $O(\log(n))$, *з часом попередньої обробки* $O(n\log(n))$.

Доведення. Розглянемо випадок, коли задане підрозбиття вдалося розділити без перетинів. Тоді множина регіонів кожного разу розбиватиметься на дві непересічні множини, а отже, отримане бінарне дерево не матиме повторень. Це дає $O(\log(n))$ часу локалізації. Якщо ж під час розбиття регіону отримаємо множини, які перетинаються, то час локалізації точки може досягати $O(n)$. Тому грань варто обирати таким чином, щоб мінімізувати число перетинаючих регіонів.

Аналогічно, складність операцій зі структурою:

побудова - $O(n\log(n))$, слідує з властивостей бінарного дерева або
$O(n^2)$ в найгіршому випадку
Insert - $O(\log(n))$ або $O(n)$ в найгіршому випадку
Remove - $O(\log(n))$ або $O(n)$ в найгіршому випадку

Практична частина

Алгоритми можна реалізувати мовою C++ з використанням бібліотеки CGAL. Було створено структуру даних (СД) у вигляді дерева розв'язку для пошуку, очищення, додавання і видалення регіонів та локалізації точки. СД можна створювати двома способами: за допомогою *.off* файлу або розбиттям прямокутного паралелепіпеда заданих розмірів одиничними кубами. У першому випадку, з використанням функціоналу вказаної вище бібліотеки, прочитана фігура розбивається на опуклі підмножини. Проте для відносно складних вхідних фігур це розбиття може мати перетини, що може сповільнити реалізацію, а саме дерево може вийти некоректним. Для додавання регіону програма буде читати список точок та будувати опуклу оболонку. У головній програмі також зберігається список регіонів, які існують в дереві на даний момент, кожен з яких має свій унікальний ідентифікатор, щоб за ним отримати потрібний регіон і передати дереву для видалення. У разі успішної локалізації знайдений регіон виводиться у форматі *.off* файлу і може бути переглянутий відповідним програмним засобом, наприклад, *Geomview*.

Висновки

Розглянуто новий підхід до розв'язання проблеми локалізації точки в опуклому тривимірному підрозбитті. Перевагами даної структури є швидка побудова, не поступається іншим відомим алгоритмам, відносна простота, зрозумілість та можливість отримати логарифмічну швидкість. Недоліком же є швидкість алгоритму локалізації при існування перетинів, що у найгіршому випадку може до- сягати лінійної. Додавання регіонів може погіршити ефективність структури, у випадку наявності перетину, тому представлена структура даних більш придатна для статичної побудови. Одним із варіантів покращення алгоритму є створення спеціального методу бінарного розбиття простору, що позбавить структуру від проблеми перетинів регіонів і площин розбиття.

2.1.3 Алгоритм на основі ідеї методу «деталізації триангуляції»

Як уже зазначалося сьогодні важко побудувати структуру даних, яка б використовувала лінійну пам'ять з логарифмічним запитом для простору розмірністю більше двох [21]. Якщо ж говорити про тривимірний простір, то загальний підхід до розв'язання поставленої задачі полягає в підтримці деревовидної структури даних, яка містить в собі певну інформацію про розбиття простору [35], або ж застосовувати методи, на кшталт замітаючої площини. [22]

Представлений алгоритм заснований на методі «деталізації тріангуляції» [26] є тривимірним його аналогом [40,44]. Запропонований метод дозволяє підвищити загальну якість і точність обчислень. Ми будуємо відповідну структура даних, що має кращі часові оцінки завдяки застосуванню технології OpenMP. Аналогом тріангуляції полігону у тривимірному випадку буде тетраедрізація політопа [45]. Аналогом трикутника буде тетраедр.

На першому етапі виконується тетраедрізація політопів розбиття для того, щоб звести задачу до випадку розбиття на тетраедри. У тривимірному випадку основна ідея залишається такою ж самою - це побудова ієрархії тетраедрів. Щоб визначити в якому з тетраедрів найвищого рівня знаходиться точка запиту, необхідно з'ясувати чи знаходиться вона між 4 точками. Для цього достатньо скористатися змішаним добутком векторів. Нехай маємо вектори ***a***, ***b,*** **c**, відкладені від одної точки, та вектор ***d***, проведений від тієї ж точки до точки запиту. Якщо змішані добутки (***a***·***b***·***d***), (***b***·***c***·***d***) та (***c***·***a***·***d***) мають однаковий знак, то точка запиту знаходиться в середині сектора побудованого на векторах ***a***, ***b***, ***c***. На чотирьох вершинах тетраедра можна побудувати чотири трійки векторів. Якщо точка запиту знаходиться в середині двох трійок, то вона знаходиться в середині тетраедра. Інший варіант, це визначення направленості граней. Для цього підходу можна використати точку, яка точно знаходиться в серединні тетраедра. Для цього можна взяти центроїд трьох точок. Його можна знайти вирахувавши середнє арифметичне координат вершин тетраедра. Якщо точка запиту знаходиться по одну сторону від граней тетраедра до точки всередині, то точка запиту теж знаходиться в середині тетраедра. Для побудови ієрархії тетраедрів ми беремо розбиття простору. Тетраедри розбиття будуть листками нашого дерева ієрархії. Потім ми вилучаємо множину незалежних вершин цього розбиття. Далі ми тетраедрелізуємо. Отримаємо наступний рівень розбиття. Цей алгоритм міг би бути еквівалентним двовимірному випадку, за виключенням операції тетраедрізації. Операція тетраедрізації у загальному випадку може бути неможливою без додавання вершин. А задача пошуку розбиття із найменшою кількістю доданих вершин може не мати поліноміальний час її розв'язання.

Побудова структури даних для розв'язку задачі локалізації

Нехай маємо деяке розбиття простору графом $G(V, E)$ у тривимірному просторі. Якщо це розбиття не є триангуляція, то підрозбиваємо його тріангуляцією Делоне. Таким чином, задача зводиться до визначення відповідного тетраедра, до якого належить точка з запиту. Розв'язок задачі почнемо з припущення: нехай маємо – n-вершинну тріангуляцію G.

Алгоритм (основні кроки)

1. Якщо, границя графа G не являє собою тетраедр, оточуємо G тетраедром, і частину області між тетраедром і границею графа G триангулюємо. Одержимо триангуляцію G'.

2. Далі застосовуємо процес деталізації триангуляції. Будується послідовність тріангуляцій S_1, S_2, …. S_h, де $S_1 = G'$, а S_i одержується з S_{i-1} за наступними правилами:

1) Видаляємо деяку множину незалежних і несуміжних вершин з Si-1 та інциденті їм ребра.

2) Тріангулюємо ті області, що утворилися після вилучення вершин. Нові тетраедри у творять триангуляцію S_i.

3) Проводимо дуги від вузлів нової триангуляції до вузлів попередньої, які мають не порожній перетин.

4) Процес деталізації триангуляції продовжуємо до тих пір доки зовнішній тетраедр не міститиме в середині вершин графа G.

5) Таким чином, кінцева S_h не буде мати внутрішніх вершин.

Означення 2.2. Незалежними вершинами будемо називати несумісні та неграничні вершини [26].

Тетраедри в тріангуляції будемо позначати T_i.Топологією структури буде направлений ациклічний граф [26]. Вигляд триангуляції в тривимірному просторі виглядає досить заплутано, тому на рисунку 2.7 наведено тільки структуру, яка побудована для деякого підрозбиття. На цьому закінчується фаза попередньої обробки даних.

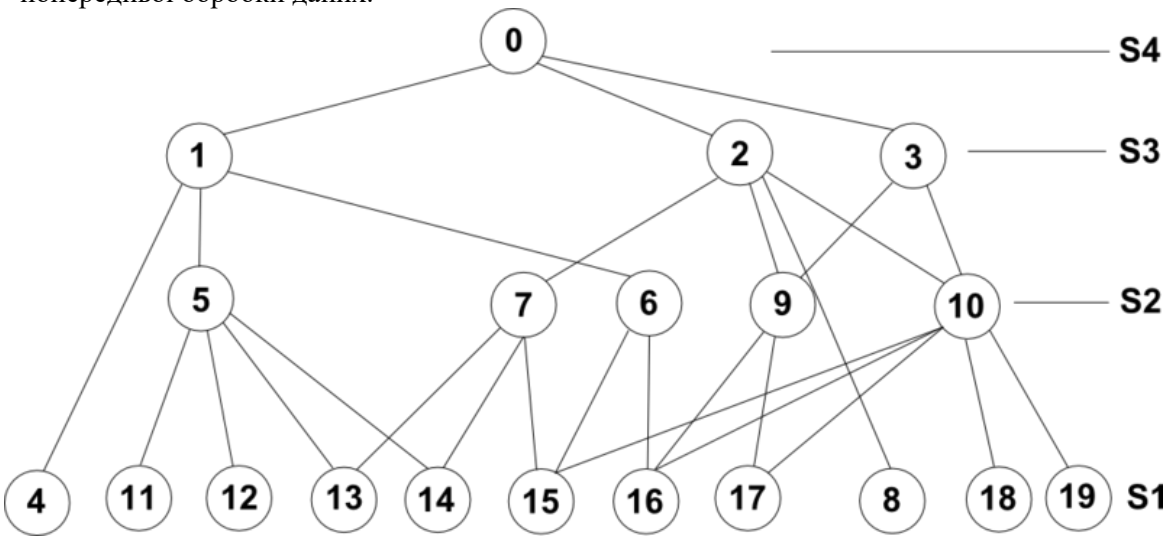


Рис 2.7. Схематичне зображення структури *D*

По завершенню попередньої фази, побудови структури даних *D*, результат якої наведено на рис.2.5, легко зрозуміти, як відбувається процес локалізації точки. Елементарною операцією є перевірка належності точки тетраедру. Початковий шаг полягає в локалізації потрібної точки відносно . Наступним кроком необхідно поступово опускаючись вниз по шляху в структурі *D* до зупинки в одному з тетраедрів, що належать , яка відбудеться обов'язково. Сама ж побудова шляху проходить наступним чином: при досягненні будь-якого вузла на шляху перевіряється належність точки до всіх нащадків даного вузла. Оскільки точка належить лише одному з нащадків, то до шляху додається ще одне ребро.

Найгіршу часову оцінку в даному підході має процес попередньої обробки, як буде зазначено далі. Для покращення цього показника та підвищення загальної ефективності застосування даної структури застосуємо технології паралельної обробки даних OpenMP.

Під час побудови структури даних, направленого ациклічного графу, сам процес побудови окремих рівнів можна розподілити поміж окремими процесами. Таким чином, загальний час побудови буде зменшено через те, що тріангуляції на різних рівнях будуть відбуватися не послідовно, а паралельно. Процес злиття результатів при цьому не дуже відрізняється від звичайної послідовної реалізації і відбувається в одному головному процесі.

Обґрунтування складності

Теорема 2.4. *Локалізацію точки за допомогою методу деталізації тріангуляції в тривимірному просторі можна зробити за час* $O(\log n)$ *витративши на попередню обробку час* $(O(n\log n)/p)$, *де* n *- кількість вершин в* G', *а* p *- кількість паралельних процесів.*

Доведення. Для побудови відповідної структури даних, необхідно передати на вхід список точок та їх початкову тріангуляцію. На кожному кроці попередньої обробки певна кількість точок вилучається та проводиться тріангуляція тих, що залишились. Тріангуляція в середньому для тривимірного простору вимагає не більш ніж $(O(n\log n)$ операцій. При цьому розподіливши процес тріангуляції між процесами маємо остаточно загальну оцінку для середнього випадку $(O(n\log n)/p)$. Безпосередньо, процес локалізації, пошук шляху в дереві від кореня до деякого листа, займає не більш ніж $O(\log n)$ операцій.

Висновки

У роботі запропоновано структуру даних, яка розв'язує задачу локалізації точки в тривимірному підрозбитті за час $O(\log n)$. В основі алгоритму лежить ідея Кірпатрика [26] для двовимірного випадку, яка знайшла успішну реалізацію і для тривимірного випадку з використанням паралельних обчислень(p

паралельних процесів). Такий підхід дозволяє зменшити загальний час побудови структури в p раз, де p - кількість паралельних процесів.

2.1.1 Адаптація методу смуг до локалізації точки в тривимірному просторі

Постановка задачі. *Задано розбиття геометричного простору n вершинним прямолінійним графом на області, а точка запиту K. Необхідно локалізувати точку на заданому розбитті.*

В даному випадку простором є тривимірний евклідовий простір, в якому задано n точок і множина граней $O(n)$, що побудовані на цих точках. Необхідно для заданої точки знайти грані що її обмежують.

Розглянемо наступну схему побудови алгоритму локалізації точки для масових запитів:

1. *Попередня обробка даних.* Для попередньої обробки важливо побудувати такі структури даних, що дозволять значно скоротити час запиту на локалізацію точки.
2. *Структура даних (СД)*. Необхідно побудувати такі структури даних, в яких будуть зберігатися результати попередньої обробки. Враховуючи те, що метою алгоритму є досягнення логарифмічного часу пошуку, нам необхідно намагатися будувати СД у вигляді дерева пошуку. При цьому, ми також повинні забезпечувати оптимізацію об'єму пам'яті що використовується.
3. *Пошук.* Основним критерієм є час пошуку – локалізації точки. Тобто необхідний алгоритм який використовуючи результат попередньої обробки за мінімальний час знаходить усі грані що локалізують точку.

Попередня обробка

Основна ідея алгоритму полягає в тому, що б розбити весь простір на обмежені підпростори в яких є структурована інформація про грані графа. Скористаємося ідеєю методу смуг із двовимірного випадку [23]

1. На першому кроці розбиваємо простір на підпростори паралельними площинами (рис.2.8).

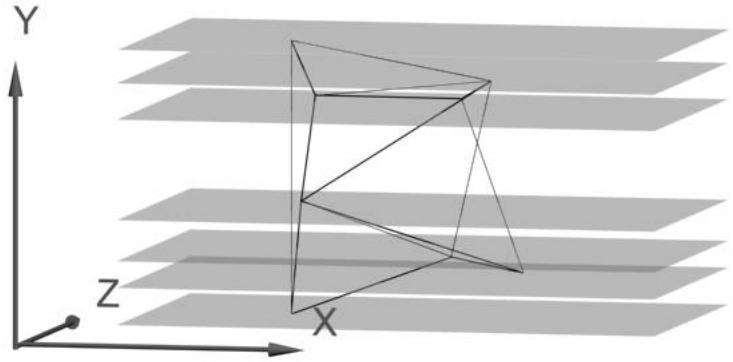


Рис 2.8. Площини, що розрізають граф на шари.

Таким чином ми маємо підрозбиття на N-1 шарів. Впорядкувавши вершини розбиття, ми можемо знайти шар який містить запитну точку за час $O(logn)$.

2. На другому кроці розбиваємо площини на смуги і визначаємо грані що їх перетинають. В результаті перетину граней одержимо області, які нагадують призми (рис.2.9). Таким чином можна провести площини паралельні кожній з осей і через кожну точку розбиття (див. рис. 2.10).

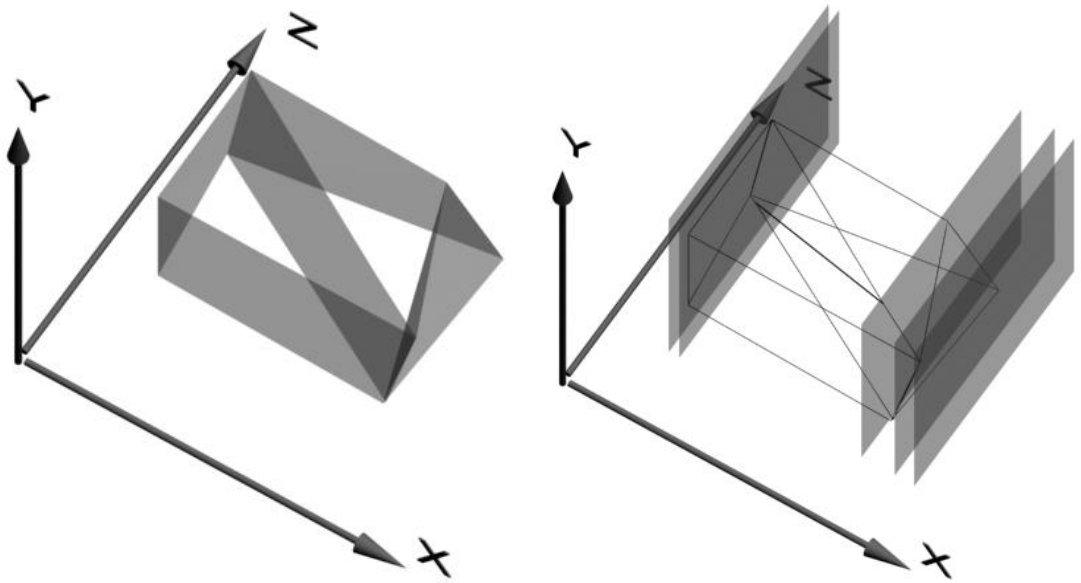


Рис.2.9. Вигляд одного з шарів. Рис.2.10. Побудова «стержнів» для шару.

В результаті ми розіб'ємо шари на «стержні» (див рис. 2.11). Впорядкувавши «стержні» можемо знайти «стержень», який містить запитну точку.

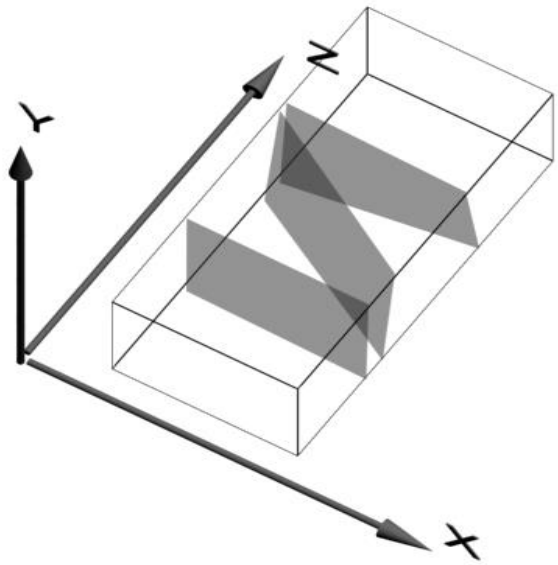


Рис.2.11. Остаточне розбиття кожного шару «стержнями».

3. На третьому кроці будуємо дерево граней в кожній смузі. Нижче приведений детальний алгоритм.

Алгоритм:

1. Сортуємо усі задані точки простору по *y* координаті. І через кожну точку простору будуємо площину паралельну *OXZ*. Будуємо дерево площин.
2. Шукаємо перетин побудованих в п.2 площин з усіма гранями, які її перетинають. В перетині буде або точка або пряма. Нехай пряма на площині задається двома точками.
3. В кожній побудованій площині сортуємо точки по *z* координаті і через кожну точку на ній проводимо лінії паралельні осі *OX*. Отримуємо смуги. Будуємо дерево смуг для кожної площини.
4. В кожній смузі визначаємо ребра які її перетинають. Для кожного ребра відома початкова грань до якої воно належить. Сортуємо ці грані по *x* координаті середньої точки *O* (рис.2.12) і будуємо дерево граней.

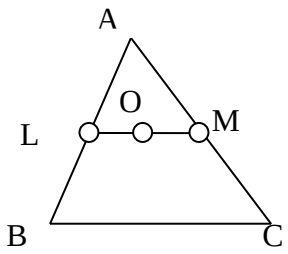


Рис. 2.12. Визначення середньої точки грані

$$L.x = \frac{(A.x + B.x)}{2} \qquad O.x = \frac{(L.x + M.x)}{2}$$
$$L.y = \frac{A.y + B.y}{2} \qquad O.y = \frac{L.y + M.y}{2} \qquad (2.1)$$
$$L.z = \frac{A.z + B.z}{2} \qquad O.z = \frac{L.z + M.z}{2}$$

В даному алгоритмі присутній крок перетину площин. Цей крок описаний нижче.

Структури даних

Данні про точки і грані зберігаються у двох двозв'язних списках. Всі інші надбудови є вказівниками на елементи цього списку.

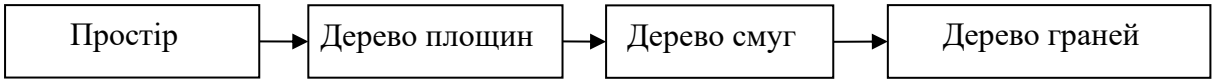


Рис. 2.13. Схема структур даних

На рис. 2.13 "стрілочку" слід читати як "містить", тобто весь простір містить площини, що зберігаються у бінарному дереві. Дерево зберігається як впорядкований масив. Кожна площина містить смуги, що також зберігаються у вигляді бінарного дерева, а для кожної смуги є бінарне дерево граней. Тобто три вкладених бінарних дерева. Дерево площин впорядковане за зростанням координати *y*, дерево смуг впорядковане за координатою *z*, а дерево граней за *x* координатою середньої точки грані.

Алгоритм пошуку

Оскільки в результаті передобробки ми одержали структуру даних, яка складається з трьох дерев, то алгоритм пошуку не викликає труднощів.

1. Дана точка *K*(x,y,z). По дереву площин шукаємо дві площини що обмежують дану точку по *y* координаті. Нехай ці площини H_1, H_2.
2. За допомогою дерева смуг шукаємо в площині H_1 смугу *P* в яку проектується точка *K* по координаті *z*.
3. За допомогою дерева граней в смузі *P* шукаємо грані що обмежують задану точку. Отримаємо множину граней *bound*(H_1).
4. Кроки 2 і 3 повторюються для площини H_2. Отримуємо множину граней *bound*(H_2).
5. Результатом буде множина *bound* = *bound*(H_1) ∩ *bound*(H_2).

Задача перетину площин

Задача. Нехай у просторі задано дві площини H_1, H_2 , кожна з яких задається трьома точками $\{A_1, A_2, A_3\} \in H_1$, $\{B_1, B_2, B_3\} \in H_2$. Необхідно знайти їх перетин.

Рівняння площини H_1 матиме вигляд:

$$Ax + By + Cz + D = 0 \quad (2.2)$$

Далі знайдемо *sign*[i], i=1..3 для кожної з трьох точок в площині H_2.

$$sign[i] = Ax_i + By_i + Cz_i + D$$
$$sum = \left| \sum_{i=1}^{3} sign[i] \right| \quad (2.3)$$

Розглянемо крайні випадки: якщо *sum*=3 то площини не перетинаються, якщо *sum*=(: H_2. Якщо *sum*=2 то $\exists\, k: sign[k] = 0$ і результатом перетину буде B_k. Якщо *sum*=1 то $\exists\, k, p: sign[k$ $n[p]! \neq 0$ і B_k=(x_1, y_1, z_1), B_p=(x_2, y_2, z_2) лежать по різні сторони від площини. Тобто пряма B_kB_p перети цину H_1.

$$\begin{cases} \frac{x-x_1}{x_2-x_1} = \frac{y-y_1}{y_2-y_1} = \frac{z-z_1}{z_2-z_1} \\ Ax + By + Cz + D = 0 \end{cases} \quad (2.4)$$

Розв'язок системи (2.4) і буде точкою перетину прямої і площини, а значить і однією з точок, що належать прямій перетину двох площин H_1, H_2. Друга точка прямої знаходиться аналогічно (обираємо інші p і k). Алгоритм знаходження перетину двох площин використано для визначення перетину граней з допоміжними площинами.

Обґрунтування складності та пам'ять

Теорема 2.5. *Задачу статичної локалізації точки у тривимірному прямолінійному графі можна розв'язати за час* $O(\log n)$ *з використанням попередньої обробки* $O(n^3 \log n)$.

Доведення. Для локалізації точки розглянутим алгоритмом необхідно послідовно зробити пошуки в п'ятьох бінарних деревах. Перевірка вузла дерева потребує $O(1)$ часу. Час пошуку по одному дереву складає $O(\log n)$, а тоді для п'яти дерев матимемо $O(5\log n)$. Тобто час пошуку дійсно $O(\log n)$. Час попередньої обробки згідно алгоритму складає $O(n^3 \log n)$. Так як для даного алгоритму треба зберігати 3 вкладені дерева то пам'ять, яка використовується, буде $O(N^3)$.

Практична частина

Для реалізації алгоритму розроблена програма мові C# для WPF (Windows Presentation Foundation). WPF – графічний фреймворк для рендерінгу користувацьких інтерфейсів під Windows. Користувачу програми надано можливість задавати довільно точку, під час виконання програми, а також редагувати вхідний файл з тривимірним графом. На рис. 2.14. Подано один із прикладів реалізації алгоритму.

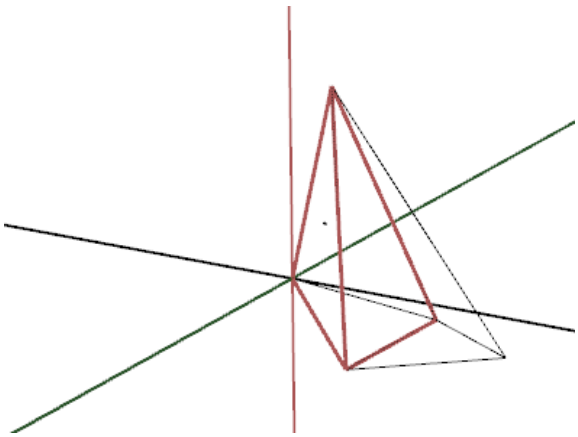


Рис.2.14. Приклад реалізації

Висновки

Запропоновано новий підхід і алгоритм до вирішення задачі локалізації точки в тривимірному просторі, обґрунтована складність і описані структури даних, що використовувалися для вирішення даної задачі. Описаний алгоритм є модифікацією методу смуг для двовимірного випадку. Досягнуто оптимальний час пошуку $O(\log n)$ з використанням $O(n^3)$ пам'яті.

ОРТОГОНАЛЬНЕ РОЗБИТТЯ E^d ПРОСТОРУ (ЛТЕОР)

Постановка задачі. Для заданого n-вершинного ортогонального розбиття евклідового E^d простору локалізувати точку P.

Ортогональне розбиття є одним із найбільш простіших розбиттів евклідового геометричного простору і тому виникає думка про можливість побудови швидкого алгоритму локалізації точки у просторі з часом пошукового запиту $\underline{O}(\log n)$. Почнемо з формулювання однієї проблеми обчислювальної геометрії, яка нещодавно вважалася відкритою [22].

ПРОБЛЕМА 34.7.2. В підрозбитті $d > 3$ - вимірної прямокутної призми на n призм розробити алгоритм локалізації точки з часом запиту $O(\log n)$, та $O(n)$ пам'яті. (Приховані константи під O можуть залежати від d. Для точкової моделі обчислень дана проблема вже досліджена для d=3)

2.1.2 Метод стратифікованих дерев для локалізації точки на ортогональному розбитті *d* вимірного простору

Розглянемо метод локалізації точки на ортогональному розбитті, який використовує такі структури даних як стратифіковані дерева. Стратифіковане дерево - це дерево інтервалів T побудоване у просторі $[0, \mathrm{U}-1]$ з деревом пошуку побудованим на рівнях дерева T. Головним у реалізації є хешування, для зменшення необхідної пам'яті. Спочатку з'ясуємо що насправді ми будемо розуміти під поняттям «стратифіковані дерева», а потім як вони застосовуються.

Означення 2.3. Дерево інтервалів T (рис. 2.15) на $[0, U-1]$ це повне бінарне дерево, що зберігає інтервали на відрізку $[0, U-1]$ (це визначення відрізняється від визначення Еделбрунера [9]). Кількість рівнів дерева інтервалів T $\log U + 1$. Кількість листків T U. Із листком j асоціюється діапазон $p(j) = \left[j - \frac{1}{2}, j + \frac{1}{2}\right]$; вершина τ дерева інтервалів T асоціюється з діапазоном $p(\tau)$ що є об'єднанням діапазонів листків піддерева з коренем τ.

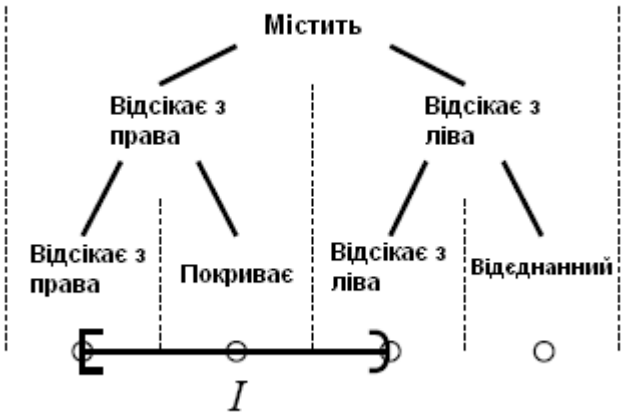


Рис. 2.15. Дерево інтервалів T на $[0,3]$ та інтервал $I = [0,2)$

З рис.2.15 можна побачити, що множина діапазонів на рівні l розбиває $[0, U-1]$ на 2^l підмножин однакового розміру. Інтервал $I = [i_{min}, i_{max})$ покриває вершину дерева $\tau \in T$, якщо I містить діапазон $p(\tau)$. Вершина τ містить інтервал I, якщо $p(\tau)$ містить I. Інтервал I відсікає вершину τ, якщо $p(\tau)$ містить в точності одну кінцеву точку I. Далі ми можемо розрізняти чи I відсікає з ліва, мається на увазі що I містить найменшу границю $p(\tau)$, чи I відсікає τ з права, мається на увазі що I містить найбільшу границю $p(\tau)$.

Нехай l_I буде найвищій рівень на якому інтервал I відсікає вершину дерева T; це можна підрахувати як найбільший значущій біт у якому i_{min} та i_{max} відрізняються. Як видно з рис. 2.15 інтервал I міститься у корені та у одній вершині на рівень нижче рівня $l_I - 1$. З l_I нижче по рівнях I відсікає дві вершини на рівень, одну з права, та одну з ліва, та покриває всі вершини між ними.

Далі треба заповнити дерево пошуку рівнів L, це збалансоване k кратне дерево пошуку побудоване на рівнях T.

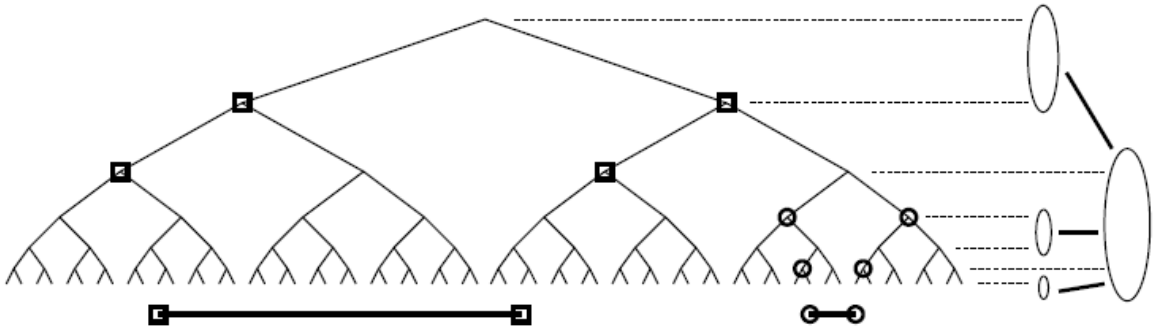


Рис.2.16. Стратифіковане дерево, яке зберігає два інтервали.

Рис.2.16 ілюструє дерево інтервалів з трикратним деревом пошуку рівнів. L заповнюється $k-1$ рівномірно розподіленими рівнями дерева T, та рекурсивно конструюється k піддерев для рівнів що знаходяться між обраними на попередньому кроці. Висота дерева L $h = O(\log\log U / \log k)$. Для кожної вершини v дерева T, визначається глибина v як найглибший рівень T у піддереві з коренем v.

Щоб зберігати інтервал I у дереві Емде Боаса [46, 47] потрібно записати границі I в кожну вершину T що відсікає I. Необхідно, використовуючи дерево пошуку рівнів, зберегти I тільки на вибраних рівнях. Зберегти інтервал I на рівні l_I дерева T та для кожної вершини v на шляху від l_I до кореня дерева. Рис. 2.16 ілюструє зберігання двох інтервалів. Кожен інтервал подається щонайбільше до $2h$ вершин T.

Позначимо вершини T на рівні l мітками-числами від 0 до $2^l - 1$ мітка вершини τ яка містить точку $q \in [0, U-1]$ - це бінарне число отримане з перших l біт числа q. Зберігаються тільки не порожні вершини дерева T, та вказівники на їх інтервали у таблиці використовуючи схему хешування Фредмана, Комлоса та Сземерді [48]. Ця схема зберігає m повних вершин використовуючи $O(m)$ пам'яті та має доступ до збережених вершин за час $O(1)$. Час передобробки - мінімум з $O(mU)$ та $O(m^3 \log U)$, тому маємо теорему:

Теорема 2.6. *Щоб зберегти n інтервалів які є розбиттям $[0, U-1]$ у стратифікованому дереві з деревом пошуку рівнів висоти h потрібно $O(nh + log\ U)$ пам'яті.*

Доведення: Дерево пошуку рівнів займає $O(\log U)$ пам'яті, знехтуємо збереженням для відповідних листків. Ці рівні, які зберігають вершини дерева T, містять $2hn$ інтервалів, а отже, максимальне число вершин та кількість пам'яті буде $O(nh)$ для всіх листків.

Одновимірний випадок

Спочатку розглянемо як задача локалізації точки вирішується в одновимірному випадку, використовуючи модифікацію стратифікованих дерев, щоб потім розширити цей алгоритм на d-вимірний випадок. Має місце теорема:

Теорема 2.7. *Використовуючи стратифіковані дерева на $[0, U-1]$ з деревом пошуку рівнів висоти $1 \le h \le log\ log\ U$, можна виконати одновимірну локалізацію точки у розбитті $I_1, I_2, \dots, I_n$ використавши $O(nh)$ пам'яті та часом запиту $O(h(log\ U)^{\frac{1}{h}})$. Використавши скорочення, можна досягти $O(n)$ пам'яті та $O(log\ log\ U)$ часу запиту.*

Доведення. Розглянемо вершину τ стратифікованого дерева та щонайбільше два інтервали які вона отримала. Якщо τ отримала інтервал I_j який відсікає τ з ліва (права), зберігається більша (менша) границя I_j. Якщо τ не отримала жодного інтервалу, тобто порожня, тоді якійсь інтервал I_j покриває $p(\tau)$.

Припустимо, тільки на один абзац, що ми дали кожен інтервал I_j до кожної вершини, яку вона відсікає – це в точності те що зроблено у заповнені дерева Емве Боаса [46, 47]. Тоді ми можемо визначити чи інтервал, який містить точку запиту q , був збережений вище, чи нижче рівня l дерева інтервалів T, виконавши наступну процедуру:

- береться мітка точки q, яка є номером отриманим з перших l біт числа q, та хешується за константний час, отримується вершина $\tau \in T$;

- якщо τ порожне, тоді q знаходиться всередині інтервала I_j , який покриває τ та усіх нащадків вершини τ. – тому немає смислу шукати глибше у дереві інтервалів;
- інакше перевіряються інтервали, які відтинають τ ліворуч та праворуч, на належність точки q;
- якщо один з них містить точку q, зупиняємося та видаємо результат;
- інакше q знаходиться всередині інтервалу I_j якій міститься у τ, отже, міститься у вершинах рівня вище, тому непотрібно шукати у рівнях вище.

Так, як кожен інтервал не передається до кожної вершини яку він відсікає, потрібно пам'ятати інтервали які ми бачили у дереві пошуку рівнів L. Для відповіді на запит точки q ми починаємо з кореня дерева L та встановлюється інтервал пошуку $I = [i_{min} = 0, i_{max} = U - 1]$.

З вершиною v асоціюється $k - 1$ листок дерева T $l_1, l_2, \dots, l_{k-1}$. Коли розглядається вершина v, використовується хеш таблиця для кожного рівня l_j щоб визначити чи вершина τ_j, чия мітка перші l_j бітів q, порожня чи зберігає однин чи два інтервали. Якщо якійсь з двох інтервалів містить q, тоді зупиняємось та видаємо цей інтервал. Інакше, ми маємо використати I щоб визначити у якому з нащадків вершини v продовжувати пошук.

Нехай i_{max} це мінімум з i_{max} та всіх границь збережених інтервалів більших за q; нехай i_{min} це максимум з i_{min} та всіх границь збережених інтервалів менших за q. Тоді інтервал пошуку $I = [i_{max}, i_{min}]$ не перетинає жоден з збережених інтервалів, але містить інтервал який містить q. Ми продовжуємо пошук у нащадку v , піддерево якого містить рівень l_I. Ми знаходимо цього нащадка вирахувавши числа вершини для i_{min} та i_{max} у рівнях $l_1, l_2, \dots, l_{k-1}$. Якщо i_{min} та i_{max} вперше зустрічаються у одній і тій-ж вершині рівня l_j, тоді ми шукаємо рекурсивно у нащадку нижче l_j - це перший нащадок v' з глибиною $< l_j$. Якщо i_{min} та i_{max} потрапляють у різні вершини на кожному рівні, тоді ми продовжуємо пошук у найвищому потомку вершини v. Лема 2.1 доводить коректність цієї процедури.

Лема 2.1. *Нехай v це вершина дерева пошуку L. Нехай I це найбільший інтервал якій містить точку запиту q який не перетинається із усіма інтервалами знайденими на рівнях по дорозі переходів по дереву пошуку рівнів до вершини v включно. Тоді інтервал I_j який містить q знаходиться у піддереві найвищого потомка v нижче всіх рівнів маючих вершину що містить I.*

Доведення: Зважаючи на те, що рівні кореня та листків додаються до відповідної вершини v; тоді ми можемо знайти два рівні l та l', з одним нащадком між ними, таким, що I міститься у вершині τ на рівні l та не міститься на рівні l'.

Ми знаємо що будь-якій інтервал який зберігається у стратифікованому дереві який перетинає I міститься у I – включаючи інтервал I_j який містить q. Т. я. I міститься у τ, тому не потрібно шукати вище рівня l.

Тепер розглянемо вершину $\tau' \in T$ яка містить q на рівні l'. Інтервал I_j чи відсікає чи покриває τ' - ми доведемо що він покриває τ'. Припустимо що це не так, що I_j відсікає τ. Тоді I містить інтервал I' який зберігається у стратифікованому дереві і відсікає τ. Але тоді вершина найвищого рівня яку I' відсікає має бути між l та l', тоді б I' зберігалася у вершині τ та була б знайдена як та що містить q або для інтервалу I. Тому I_j покриває τ та також всіх послідовників τ - тому не потрібно шукати нижче ніж l'. Це доводить лему.■

Доведення Теореми 2.7 майже закінчено. Для кожного рівня з h дерева пошуку рівнів розглядається $k - 1$ рівень дерева інтервалів за константний час. Час запиту $O(hk)$ та пам'ять $O(nh)$, де $h = \log\log U / \log k$. Варіюючи параметр h маємо компроміс між об'ємом пам'яті та часом запиту: Обираючи k константою дає $O(\log\log U)$ час запиту та $O(n \log\log U)$ пам'яті. Обираючи h константою маємо $O((\log U)^{1/h})$ час запиту та $O(n)$ пам'яті.

Компроміс за рахунок h непотрібний для одновимірного випадку локалізації точки. Може бути використана техніка *зменшення* ван Емде Боаса[49] для того щоб зменшити пам'ять до лінійної та збільшити час запиту лише на константу. Обирається кожна $\log\log U$ границя інтервалу для формування $n/\log\log U$ над-інтервалів та збереження цих над-інтервалів у стратифікованому дереві використовуючи $O(n)$ пам'яті. Даний запит знайде над-інтервал у стратифікованому дереві, потім використовується лінійний пошук, щоб знайти дійсний інтервал. Час запиту залишається $O(\log\log U)$. Це завершує доведення Теореми 2.7.

Так як об'єднання прямокутників не є прямокутник, зменшення у розмінностях простору більше ніж одномірному є складнішим. У кінці наступної частини буде використана теорему планарного розбиття щоб показати що відсікання досі можливе у двомірному просторі. Для тривимірного випадку нам потрібен компроміс щоб отримати лінійну пам'ять.

Двовимірний випадок

У цьому розділі буде показано як виконати локалізацію точки у прямокутному розбитті площини використовуючи шари стратифікованих дерев.

Теорема 2.8. *Використовуючи стратифіковані дерева на* $[0, U-1]^2$, *можна виконати двовимірну локалізацію точки у прямокутному розбитті* $R_1, R_2, \dots, R_n$ *за* $O(h(\log U)^{\frac{1}{h}} \log\log U)$ *часом запиту та використавши* $O(nh)$ *пам'яті, для будь якого цілого* $1 \le h \le \log\log U$. *Використовуючи скорочення можна досягти* $O(n)$ *пам'яті.*

У двовимірному випадку локалізації точки, потрібно заповнити стратифіковане дерево інтервалами по осі x майже так як у попередній частині. Ми даємо прямокутник на найвищий рівень вершини, яку x інтервал відсікає та до всіх вершин, які він відсікає на шляху до кореня дерева пошуку рівнів.

Розглянемо вершину τ у дереві інтервалів T: вона має діапазон $p(\tau) = [x_{min}, x_{max}]$ та отримує множину прямокутників R яка відсікає її (Див. рис. 2.18).

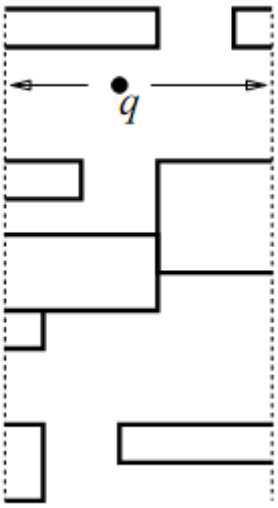


Рис. 2.18: Прямокутники які передаються вершині τ

Перетин лінії $x = x_{min}$ або $x = x_{max}$ з множиною R - це множина інтервалів (не розбиття, тому-що існують пропуски залишені прямокутниками), які покривають τ або вони зберігаються ще десь у стратифікованому дереві. Якщо заповнити ці пропуски, можна використати одновимірну локалізацію точки, щоб знайти проекцію точки запиту q на прямі $x = x_{min}$ та $x = x_{max}$. Заповнення пропусків щонайбільше може подвоїти кількість інтервалів, ми можемо знайти обидві проекції за час $O(\log\log U)$ використавши пам'ять, обсяг якої буде пропорційним кількості прямокутників отриманих вершиною τ. Це означає що загально потрібно $O(nh)$ пам'яті.

Якщо проекція точки q лежить в y-інтервалі прямокутника $r \in R$, тоді можна перевірити (за константний час) чи точка q також лежить в x-інтервалі прямокутника r. Щоб визначити прямокутник, якій містить точку запиту q ми починаємо з кореня дерева пошуку рівнів, та встановлюємо сегмент пошуку I на розбиття горизонтальних ліній через q з координатами x та проміжку $[0, U-1]$. У вершині v дерева пошуку ми використовуємо хешування щоб визначити вершину, яка містить q для кожного $k-1$ рівня асоційованого з v, та одновимірну локалізацію точки щоб перевірити прямокутники, які містять точку q. Якщо такий прямокутник знайшовся, ми зменшуємо сегмент I до сегменту, який лежить між найближчим з прямокутників, які перетинають I ліворуч та праворуч точки q. Ми продовжуємо пошук у нащадку v, який знаходиться точно нижче найнижчого рівня з вершиною що містить I. Знову Лема 2.1 доводить коректність цієї процедури.

Для кожного рівня з h рівнів дерева пошуку запит перевіряє $k-1$ рівень дерева за час $O(\log\log U)$. Тому час запиту буде $O(hk\log\log U)$, де $h = \log\log U / \log k$. Якщо не брати до уваги скорочення, це доводить Теорему 2.8. Обираючи k за константу, матимемо $O((\log\log U)^2)$ час запиту та $O(n\log\log U)$ використаної пам'яті. Обираючи h константою маємо $O((\log U)^{\frac{1}{h}}\log\log U)$ час запиту із лінійною пам'яттю.

На закінчення цієї частини буде показано двовимірний аналог техніки скорочення ван Емде Боаса для зменшення пам'яті до лінійної, при цьому збільшивши час запиту лише на константу. Фактично потрібно довести що можна об'єднати прямокутники $R_1, R_2, \dots, R_n$ у групи розміром $O((\log\log U)^2)$ та покрити ці групи новим розбиттям з $m = O(n/\log\log U)$ прямокутників $R'_1, R'_2, \dots, R'_m$ так, щоб кожен новий

прямокутник R'_i перетинав тільки одну групу. Ми можемо зберігати нові прямокутникі $R'_1, R'_2, \dots, R'_m$ у структурі для локалізації точки яка використовує $O(n)$ пам'яті та знайти прямокутник R'_i який містить точку запиту за час $O((\log\log U)^2)$. Прямокутник R'_i вказує на групу з $O((\log\log U)^2)$ прямокутників, які можна просто перебрати. Таким чином частина скорочення Теореми 2.8 буде доведена після доведення Леми 2.2.

Лема 2.2. *Нехай прямокутники $R_1, R_2, \dots, R_n$ з прямокутного розбиття. За лінійний час можна розділити ці n прямокутників на множини розміром $O((\log\log U)^2)$ та знайти нове розбиття $R'_1, R'_2, \dots, R'_m$ щонайбільше з $m = O(n/\log\log U)$ прямокутників таких, що кожен новий прямокутник R'_i перетинає лише одну множину.*

Доведення: Згідно теореми Ліптона та Тарьяна про планарне розділення [50] випливає, що будь який планарний граф з v вершин має підмножину з $2\sqrt{2v}$ вершин вилучивши які граф розбивається на компоненти щонайбільше з $2v/3$ вершин кожна.

Планарним графом розбиття $R_1, R_2, \dots, R_n$ буде граф, вершини якого прямокутники, а ребра у відношенні сусідства прямокутників. Цей граф очевидно буде планарним. Якщо зв'язаних компонент цього графа v буде більше за величину вершин (прямокутників)(v > (log(logU))^2), застосуємо цю теорему для видалення підмножини вершин (з більш ніж $2v/3$ вершинами). Використавши алгоритм Гудріча [51] ми можемо підрахувати усі ці планарні розбиття за лінійний час.

Після завершення алгоритму позначимо ті вершини, які відносяться до двох вершин та були видалені як множина C. Видалення прямокутників C з розбиття залишає з'єднані групи розміром не більше ніж $O(\log\log U)^2$. Далі, кожну групу ми об'єднуємо та розбиваємо на прямокутники розрахувавши вертикальну карту (робляться вертикальні розрізи через розділяючи вершини границі об'єднання). Це можна зробити простим замітанням якщо граничні точки збережені. Нехай D це множина всіх прямокутників заповнених цим способом. Рис.2.19 показує маленьке прямокутне розбиття і його розділення на прямокутники C та D.

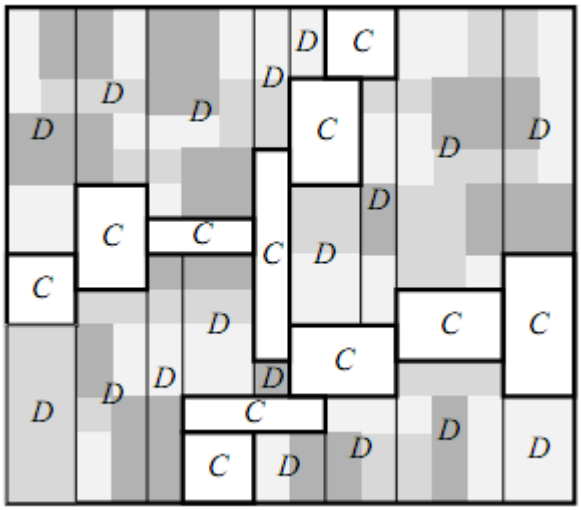


Рис.2.19: Прямокутники C та D

Нашим новим розбиттям буде $C \cup D$. Так, як кожен прямокутник з D лежить всередині об'єднання щонайбільше $(\log\log U)^2$ прямокутників, очевидно що новий прямокутник перетинатиме щонайбільше $(\log\log U)^2$ старих прямокутників. Необхідно показати, що кількість прямокутників $|C \cup D|$ буде $O(n/\log\log U)$. Достатньо показати це для $|C|$ так як кількість прямокутників у D пропорційна до складності границі об'єднання з'єднаних компонент, яке в свою чергу пропорційне до $|C|$.

Згідно з теоремою Ліптона та Таряна [50] ми формуємо дерево на множині всіх компонент побудованих алгоритмом: компоненти які розділяються є родичами результуючої компоненти. Ми нумеруємо кожну компоненту у дереві по найбільшій відстані до листків. Листки, компоненти яких не розділені, нумеруються нулем.

Компоненти з номером 1 розділяється один раз, тому вони мають розмір щонайменше $(\log\log U)^2$. Використовуючи індукцію, можна заначити, що кожна компонента з номером i має розмір щонайменше $(3/2)^{i-1}(\log log\, U)^2$. З того що вершина хоча б в одній компоненті з даним номером, тоді існує така $m_i \leq n\left(\frac{2}{3}\right)^{i-1}/(\log\log U)^2$ компонента з номером i.

Розглянемо внесок C з m_i компонент з номером i. Якщо n_j це розмір компоненти j з номером i, тоді внесок у C буде $\sum_{1\leq j m_i} 2\sqrt{2n_j}$. Так, як нерівність Курчі-Ісрцарза $\sum_{1\leq j m_i} n_j \leq n$, показує, що максимальний внесок

буде коли компоненти мають однаковий розмір; внесок щонайбільш $m_i 2\sqrt{2n/m_i} = 2\sqrt{2nm_i}$. Сумуємо по всіх додатних числах компонент, маємо максимальний загальний внесок:

$$\sum_{i\geq 1} 2\sqrt{2nm_i} \leq \sum_{i\geq 1} 2\sqrt{2}\left(\frac{2}{3}\right)^{(i-1)/2} \frac{n}{\log\log U} = O\left(\frac{n}{\log\log U}\right). \quad (2.5)$$

Це доводить Лему 2.2 та Теорему 2.8.

Примітка: На цілочисельних решітках ця структура даних може покращити багато алгоритмів, які використовують локалізацію точки, як частину алгоритму. Наприклад, потрібно визначити k горизонтальних сегментів які перетинають вертикальний сегмент запиту, це може бути зроблено за час $O((\log\log n)^2 + k)$.

Тривимірний випадок

Тепер алгоритм тривимірної локалізації точки - це логічне продовження попередніх статей: використовується стратифіковані дерева з двовимірною локалізацією точки, як вторинна структура для кожного ребра.

Теорема 2.9. *Задачу локалізації точки на тривимірному ортогональному розбитті* $R_1, R_2, \dots, R_n$ *можна розв'язати за час* $O(h(\log U)^{\frac{1}{h}}(\log\log U)^2)$ *за допомогою стратифікованих дерев на* $[0, U-1]$ *з використавши* $O(nh)$ *пам'яті для будь якого цілого* $1 \leq h \leq \log\log U$.

Доведення: Знову заповнюється стратифіковане дерево на інтервалах по осі x та передається кожен паралелепіпед у вершину найвищого рівня, яку цей x інтервал відсікає, та до вершин інших рівнів, які також відсікає цей інтервал нижче на шляху до кореня дерева пошуку рівнів. Нехай R це множина паралелепіпедів які відсікають вершину $\tau \in T$ праворуч (множина яка відсікає τ ліворуч будується так само). Усі паралелепіпеди з R перетинають площину, що проходить через праву границю інтервалу $p(\tau)$, формуючи прямокутники. Ці прямокутники розширюються до прямокутного двовимірного розбиття формуючи вертикальну карту сусідства – робимо вертикальні зрізи з кутів прямокутників з R. Цей процес заповнення прогалин на площині збільшує кількість прямокутників на константну величину. Потім ми можемо використати двовимірну локалізацію точки щоб знайти проекцію точки запиту q на площину границі $p(\tau)$ за час $O((\log\log U)^2)$ з лінійною пам'яттю. Це доводить, що загальна необхідна пам'ять $O(nh)$.

Якщо проекція q лежить у прямокутнику, тоді можна перевірити за константний час чи лежить q у тривимірному паралелепіпеді, який утворив прямокутник. Тому, щоб виконати запит, ми починаємо з кореня дерева пошуку інтервалів та встановлюємо сегмент I на розбиття лінії через q та паралельно осі x, який має координати у $[0, U-1]$. У вершині дерева пошуку рівнів v використовується хешування щоб виявити вершину, яка містить q на кожному з $k-1$ рівнів, асоційованому з v та використовується двовимірна локалізація точки, щоб перевірити прямокутники, що містять q. Якщо жоден не містить, ми ділимо сегмент I найближчими паралелепіпедами, що перетинають I ліворуч та праворуч q. Ми продовжуємо пошук у нащадку v нижчому за найнижчий асоційований рівень, що містить вершину з інтервалом I. Лема 2.3 доводить коректність цієї процедури. Аналіз часу виконання такий самий як у попередніх розділах за виключенням заміни двовимірної локалізації точки скороченням вторинної структури. Це доводить Теорему 2.9. ■

Зміна параметра h дозволяє варіювати співвідношення пам'ять/час: Обираючи h як константу, одержимо $O((\log U)^{\frac{1}{h}}(\log\log U)^2)$ часу запиту та лінійну пам'ять. А якщо, $h = \lfloor \log\log U \rfloor$, матимемо $O((\log\log U)^3)$ час запиту та $O(n\log\log U)$ пам'яті.

Цей метод не може бути розширений на вищі розмірності з такою ж ефективністю, тому що вже неможливо буде заповнювати прогалини використовуючи лінійну пам'ять. Так, матимемо множини з m паралелепіпедів які містяться лише у розбитті розміром $\Omega(m^{3/2})$.

D-вимірний випадок

У попередніх розділах була розроблена структура даних для локалізації точки у фіксованих просторах. У цій частині буде знайдена просте застосування цієї структури даних для не фіксованих решіток. Це включає логарифмічний час, та лінійну пам'ять для структури даних локалізації точки у тривимірному прямокутному розбитті з будь якими координатами, локалізація однієї точки у декількох розбиттях.

Локалізація точки у реальному тривимірному розбитті

Скористаємося простою нормалізацією.

Теорема 2.10. *На даному тривимірному прямокутному розбитті розміру n з будь якими дійсними координатами можна локалізувати точку запиту за час $O(\log n)$ з лінійною пам'яттю та $O(n \log n)$ часом передобробки.*

Доведення: Кожен паралелепіпед це добуток інтервалів за трьома координатними напрямками. Для кожної з цих напрямків із збереженої черги границь інтервалів замінюємо кожен дійсний інтервал $[a, b)$ інтервалом з цілими границями $\left[ranc(a), ranc(b)\right)$. Це вимагатиме $O(n \log n)$ часу передобробки.

Тепер прямокутне розбиття може розглядатися як розбиття цілочисельної решітки максимальна координата якої n. Потім можна використати Теорему 2.10 та відповідну структуру даних для визначення прямокутника, який локалізує запитну точку за час $O(\log n)$, використовуючи лінійну пам'ять. (Якщо дерево пошуку рівнів на всіх вимірах взято висоти $h = 3$, наприклад, тоді ці границі досягаються без скорочення та час для побудови цієї структури буде $O(n)$.)

Для того щоб виконати запит з дійсною точкою q, нормалізуємо точку q до точки решітки: замінюємо кожну координату точки q її рангом, який визначається бінарним пошуком у списку границь для цієї координати. Потім ми визначаємо паралелепіпед, який містить цю точку. Обидва кроки: нормалізація та локалізація точки у решітці, виконуються за час $O(\log n)$. ■

Приклад з локалізації точки у декількох розбиттях з загальною кількістю прямокутників n. Усі границі прямокутників з відповідними інтервалами на осях можуть бути об'єднанні у один список рангів та всі нормалізації можуть бути виконані у цьому списку.

Теорема 2.11. *Локалізація точки на k прямокутних розбиттях розмірності n для d -вимірного простору $(d \geq 3)$ бути може бути виконана за час $O(\log n + k(\log \log n)^d)$.*

Вилучення етапу заповнення порожнин

На практиці частіше зустрічаються не повні розбиття простору. Не повне розбиття - це такий набір прямокутників $R_1, R_2, \dots, R_n$ з простору $\mathcal{R}$ що не перетинаються, а їх об'єднання не дорівнює $\mathcal{R}$, але повністю міститься в $\mathcal{R}$. Для такого випадку можна виконати операцію заповнення прогалин, це збільшить кількість прямокутників на лінійну величину, тобто складність алгоритму не збільшиться. Іншим варіантом є модифікація алгоритму.

На етапі початкового алгоритму, починаючи з двовимірного випадку, підчас заповнення прогалини можна ввести стан алгоритму як такий, що не знайшов відповіді. Тобто якщо у двовимірному випадку перевірка вершини τ не знаходить інтервал на який проектується точка запиту q, то ця точка проектується на інтервал який мав бути доданий. Алгоритм буде здійснювати пошук по дереву пошуку рівнів. А оскільки висота дерева обмежена, алгоритм обов'язково зупиниться. Таким чином якщо не заповнювати простір до повного розбиття загальний алгоритм також зупиниться. Якщо алгоритм зупиниться не видавши прямокутник, який містить точку запиту, тоді точка q не знаходиться в жодному з прямокутників, тобто вона знаходиться у прямокутнику, який мав би бути доданий у першому випадку.

Програмна реалізація

Для розглянутого підходу було розроблену програмну реалізацію, яка локалізує точку на d вимірному прямокутному розбитті проте з деякими обмеженнями. Розмірність простору має бути в діапазоні від 2 до 8. Кількість прямокутників розбиття не більше 1000.

Приклад роботи програми продемонстровано на рис. 2.20.

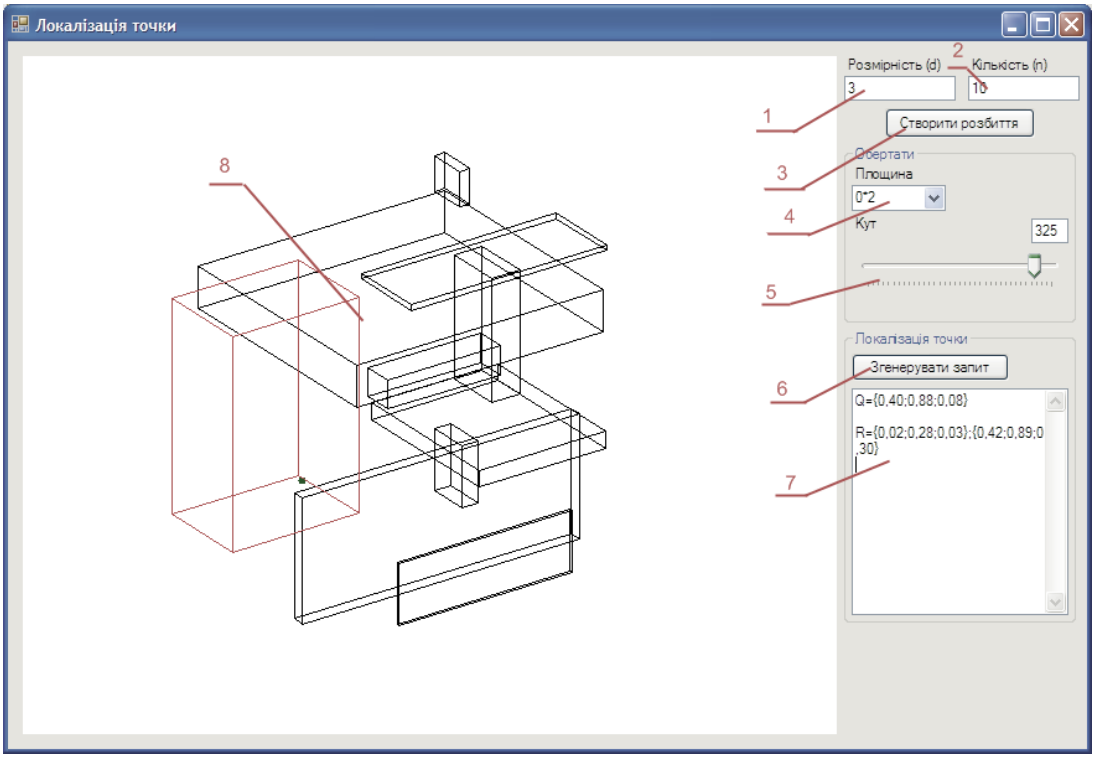


Рис.2.20. Приклад роботи алгоритму.

1. Поле для вводу розмірності простору на якому створюється розбиття.

2.Кількість d вимірних прямокутників які є розбиттям.

3.Кнопка яка генерує випадкове розбиття з вище заданими параметрами.

4.Площина у якій виконується обертання прямокутників розбиття для просторового уявлення.

5.Повзунок для зміни кута на якій повертаються прямокутники розбиття у обраній площині.

6.Кнопка генерації випадкової точки запиту та виконання запиту.

7. Поле для чисельної перевірки коректності роботи алгоритму. У ньому відображаються координат точки запиту, та прямокутник, що її містить, якщо такий є.

8. Графічне відображення розбиття, точки запиту та результату запиту.

Для відображення d – вимірних прямокутників було розглянуто деякі їх властивості. d – вимірний прямокутник є декатровим добутком d відрізків $[a_i; b_i)\ i = \overline{1..d}$ на відповідних координатних осях. Тому прямокутник можна задати двома векторами $\vec{a}$ та $\vec{b}$. Цей прямокутник має вершини з координатами $\vec{c}$ де $c_i = a_i$ або b_i. Тому вершин буде 2^d. Ребра будуть паралельні осям координат, та через кожну точку буде проходити ребро паралельно всім осям координат. Тому ребер буде $d2^d/2 = d2^{d-1}$.

Таким чином накладаються обмеження на розмірність простору 8 та кількість прямокутників 1000. Кількість ребер у 8 вимірного прямокутника 1024. Загалом буде 1024000 ребер.

Для просторового уявлення реалізовано функцію обертання. d - вимірний простір проектується на двовимірну площину ортогонально. Перед цим виконується обертання базисних відрізків у площинах. Перетворення поворот зберігає відстані та кути. Площини утворюються двома одиничними векторами паралельними осям координат. Тому площин буде $d(d-1)$.

Висновки

У представленому підрозділі розглянуті особливості локалізації точки на d-вимірному розбитті. На даний момент не існує алгоритму у загальному випадку, що має логарифмічний час та використовує лінійну пам’ять. Були запропоновані розширення існуючих алгоритмів. Алгоритм методу смуг у тривимірній аналогії дав логарифмічний час та $O(n^3)$ пам’яті. Аналог алгоритму деталізації тріангуляції алгоритм Кірпатріка має проблему аналога тріангуляції – тетраєдрізації. Тетраєдрізація політопа вимагає додавання вершин, та мінімізація кількості доданих вершин задача не поліноміальної складності. Було запропоновано

метод зведення трьохвимірної задачі до планарного випадку. Алгоритм зведення має логарифмічну складність та потребує $O(n^2)$ пам'яті. Зведену задачу можна розв'язати модифікацією існуючих алгоритмів.

Був розглянути частинний випадок коли розбиття є прямокутним, коли всі області є декартовим добутком відрізків на координатних осях. Для розв'язання цієї задачі було розглянуто та використано теорію стратифікованих дерев. Був розроблений алгоритм який робить локалізацію точки у одномірному просторі $[0, U-1]$ за час $O(\log\log U)$ та лінійною пам'яттю. Запропоновано розширення до двовимірного простору $[0, U-1]^2$ зі складністю $O((\log\log U)^2)$ та лінійною пам'яттю. Було показано зведення задачі з фіксованого простору з цілими координатами до не фіксованого простору з дійсними координатами. Цей алгоритм розширений до d вимірного простору з дійсними координатами. Алгоритм має лінійну пам'ять та час запиту $O(\log n + k(\log\log n)^d)$. Були розроблення спрощення алгоритму для простоти реалізації, які не бажано робити на етапі доведення коректності роботи алгоритму.

2.2 РЕГІОНАЛЬНИЙ ПОШУК

Загальна постановка задачі РЕГІОНАЛЬНИЙ ПОШУК. На заданій множині S із n об'єктів (точок) в евклідовому просторі E^d (d = 2 - площині) задано запитний регіон (у вигляді деякої геометричної фігури) R. Знайти підмножину об'єктів (точок) S' множини S (або їх кількість k), які містяться в регіоні R. Розглядаємо випадки $k << n$.

Запитом є область R (регіон) у E^d, а результатом пошуку є звіт про множину точок файлу, яка міститься у цій області (запит у режимі звіту), або - підрахунок числа точок в області запиту (запит у режимі підрахунку). Приклади на рис. 2.21.

а) б)

Рис 2.21. Приклади задач регіонального пошуку на практиці: а) задача комп'ютерного зору, б) топографічна навігація.

Аналіз та вибір ефективної стратегії розв'язання задач регіонального пошуку.

Оскільки нижня оцінка складності задач цього класу складає $\Omega(\log n)$, де n кількість вхідних даних, то необхідно обрати ту стратегію, яка б досягала складності алгоритмів пошуку для n даних $O(\log n)$ чи $\theta(\log n)$. Одним із підходів пошуку бажаної стратегії зазвичай розглядають більш просту постановку задачі (наприклад одновимірний випадок), а потім знайшовши ефективну ідею, адаптують її на вищі розмірності (якщо це вдається, звичайно).

Розглянемо одновимірний випадок задачі регіонального пошуку. У цьому випадку вхідна множина S із n точок розташована на числовій осі, а запитом є інтервал $[x', x'']$, або x - регіон, рис. 2.22. Список точок на числовій осі $P = \{P_1, P_2, \ldots, P_n\}$ можна подати у вигляді бінарного дерева пошуку, рис.2.23.

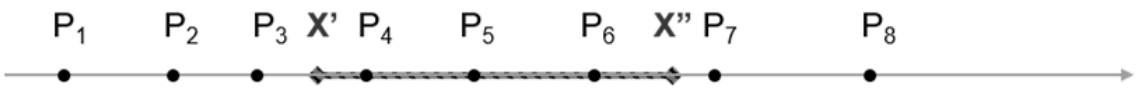


Рис.2.22. Вхідні дані задачі.

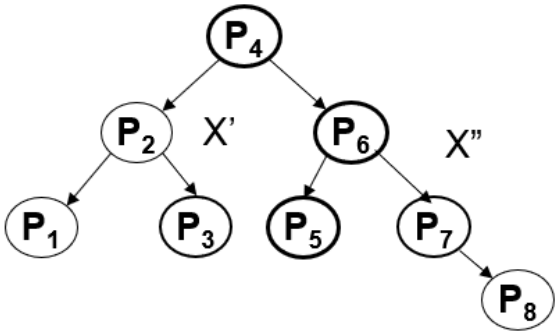


Рис. 2.23. Дерево пошуку.

За логарифмічний час ми локалізуємо *x'* (лівий кінець *x* -регіону), а потім, рухаючись по дереву визначаємо точки, які потрапили в запитний регіон доки не дійдемо до *x"* (правий кінець *x* -регіону). Структура даних, яка забезпечує вказану дію, є *прошите двійкове дерево*, т.т. таке збалансоване двійкове дерево, листки якого додатково зв'язані у списку, який виражає порядок абсцис; дерево і список обробляються на фазах відповідного пошуку й вибірки. Маємо оптимальну оцінку для часу запиту $\theta(\log n+k)$ і для пам'яті $\theta(n)$. Таким чином одновимірний випадок дає нам бажану оцінку для регіонального пошуку $\theta(\log n)$. Далі розроблено на основі цієї ідеї методи, адаптовані до багатовимірного випадку і дають близькі до нижньої оцінки часові складності пошукових запитів для прямокутного запитного регіону. Найбільш відомими із цих методів є метод *k-d-* дерева та метод дерева регіонів, які детально описані в роботах [52, 53].

РЕГІОНАЛЬНИЙ ПОШУК НА ПЛОЩИНІ

2.2.1 РЕГІОНАЛЬНИЙ ПОШУК У ПРЯМОКУТНИКУ (ГРПП(АГ0))

Постановка задачі. На площині задано *n* точок. Знайти усі точки, які потрапляють в середину прямокутного запитного регіону сторони, якого паралельні осям координат.

Основними методами розв'язання цього класу задач є, зазначені вище, метод k-d-дерева та метод дерева регіонів [52,53].

Коротко опишемо ці методи, для $k = 2$.

2.2.1.1 Метод 2-d дерева [52]

Ідея алгоритму. Нехай задана множина *S* із *N* точок на площині. Для побудови структури даних, яка називається "2-D дерево" використовується схема "розподіляй та володарюй" суть якої полягає в рекурсивному розбиті площини на прямокутники *R*(v), означені нижче. Результатом розбиття є бінарне дерево пошуку, вузлами якого є точки із заданої множини *S* на площині, рис.2.24.

Означення 2.4. Назвемо *узагальненим прямокутником* таку область на площині, яка визначена декартовим добутком $[x_1, x_2]\times[y_1,y_2]$ *x*- інтервалу - $[x_1, x_2]$ та *y* – інтервалу - $[y_1,y_2]$, включаючи граничні випадки, коли в довільній комбінації допускається: $x_1 = -\infty$, $x_2 = \infty$, $y_1 = -\infty$, $y_2 = \infty$.

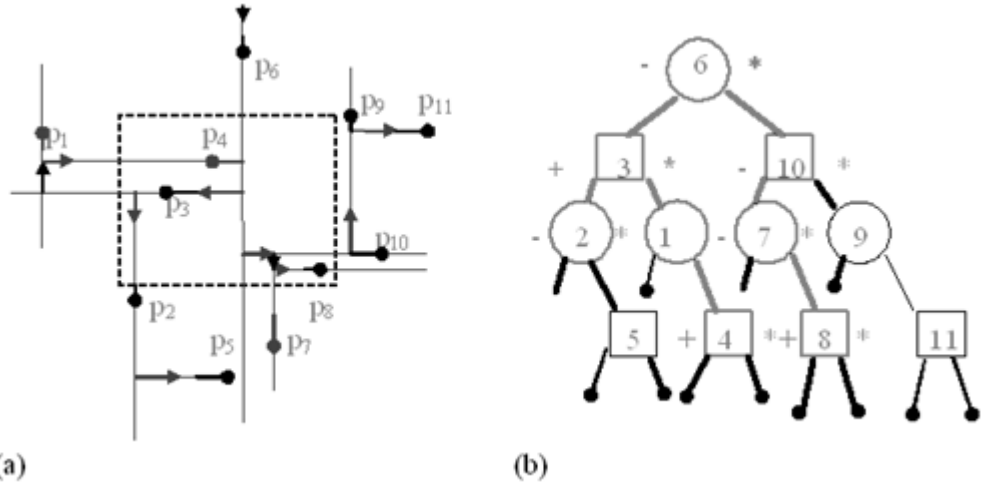


Рис.2.24. Ілюстрація методу пошуку за допомогою двовимірного двійкового дерева: а) рекурсивне розбиття площини на прямокутники; б) дерево пошуку.

Алгоритм побудови структури даних.

1. Спроектувати усі точки множини **S** на вісь *OX* та вісь *OY*.

2. Сформувати два упорядковані списки по $\boldsymbol{x}$ (U_x) та по $\boldsymbol{y}$ (U_y) координатах.
$U_x = \{P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9, P_{10}, P_{11}\}$, $U_y = \{P_5, P_7, P_2, P_8, P_{10}, P_3, P_4, P_1, P_{11}, P_9, P_6\}$.

3. Знайти медіану $P(v)$ списку U_x, і через неї провести **вертикаль** $l(v)$. Точка $P(v)$ буде коренем шуканого дерева і розіб'є список U_x на дві рівнопотужні підмножини $U_{x1} = \{P_1, P_2, P_3, P_4, P_5\}$, $U_{x2} = \{P_7, P_8, P_9, P_{10}, P_{11}\}$, для яких існують упорядковані аналоги по $\boldsymbol{y}$ координаті із списку U_y : $U_{y1}=\{P_5,P_2,P_3, P_4, P_1\}$, $U_{y2} = \{P_7, P_8, P_{10}, P_{11}, P_9\}$.

4. Списки U_{y1}, U_{y2} та U_{x1}, U_{x2} передаються на наступний рівень рекурсії.

5. На цьому рівні рекурсії визначаються медіани ($P_1(v)$, $P_2(v)$) у списках U_{y1}, U_{y2}, відповідно, і через них проводяться **горизонталі** $l_1(v)$ і $l_2(v)$. Точки $P_1(v)$ і $P_2(v)$ будуть наступними вузлами шуканого дерева і розіб'ють списки U_{y1}, U_{y2} на рівнопотужні підмножини, які передаються на наступний рівень рекурсії.

6. Процес розбиття вертикалями та горизонталями продовжується доти, поки не будуть вичерпані усі точки множини *S*, і тим самим буде побудоване дерево пошуку. Тобто, процес подрібнення завершиться, коли з'явиться прямокутник, який не містить всередині жодної точки, відповідний йому вузол являється листком дерева *T*.

7. В результаті отримаємо структуру даних, яка називається *2-d-деревом* (Рис.2.24 б).

Алгоритм пошуку

1. Кожен вузол *v* завантажується такими параметрами: $P(v)$, $R(v)$, $S(v)$, $l(v)$, D.
Де $P(v)$- поточна точка, $R(v)$- прямокутна область, $S(v)$- множина точок яка належить R(v), $l(v)$- розрізаюча пряма, яка проходить через точку $P(v)$, D – запитний регіон.

2. Від взаємного розташування $R(v)$, $S(v)$, $l(v)$, D для кожного вузла $v \in T$, залежить регіональний пошук.

3. $\forall v \in T$, починаючи з кореня дерева (вузол 6) перевіряємо взаємне розташування запитного регіону D і прямої $l(v)$(вертикаль через точку P_6), яка розбиває $R(v) = R_1(v) \cup R_2(v)$, $R_1(v) \cap D$, $R_2(v) \cap D$.

3. В залежності від розташування можуть бути наступні випадки:

1) $D \cap R_1(v) \neq \varnothing$ і $R_2(v) \cap D = \varnothing \Rightarrow$ лівий пошук (у $D \cap R_1(v)$);
2) $D \cap R_2(v) \neq \varnothing$ і $R_1(v) \cap D = \varnothing \Rightarrow$ правий пошук (в $D \cap R_2(v)$);
3) $D \cap R_1(v) \neq \varnothing$ і $R_2(v) \cap D \neq \varnothing \Rightarrow$ лівий (в $D \cap R_1(v)$) і правий (у $D \cap R_2(v)$); перевірка $P(v) \notin D$?.

У нашому випадку маємо варіант 3), помічаємо вузол «*» і робимо перевірку належності точки P_6 *y*-регіону D. $P_6 \notin D$, ставимо «-».

4. У вузлі 3 маємо варіант 3), помічаємо вузол «*» і перевірка належності *x* -регіону D дає позитивний результат, $P_3 \in D$. У вузлі 10 маємо варіант 3), помічаємо вузол «*» і робимо перевірку належності точки P_{10} *x*-регіону D. $P_6 \notin D$, ставимо «-».

5. Процес пошуку здійснюється рекурсивно від кореня до листків дерева.

Теорема 2.12. *За допомогою метода* 2-D-*дерева регіональний пошук на площині, яка містить* n *точок можна провести за час у гіршому випадку* $O(\sqrt{n})$ *і у середньому* $O(\log n)$ *із використанням* $O(n)$ *пам'яті, якщо витратити* $O(n \log n)$ *часу на попередню обробку*.

2.2.1.2 Метод дерева регіонів [78]

Ідея алгоритму. Будується дворівнева структура даних (дерево регіонів, рис. 1.5), перший рівень якої – дерево відрізків по одній із координат (наприклад *x),* а другий – вузли дерева відрізків прошиті упорядкованими списками по іншій координаті (наприклад *y).*На побудованому дереві регіонів виконується операції вставки інтервалу (проекція запитного регіону на вісь *OX),* що дозволяє визначити вузли віднесення. Далі у вузлах віднесення застосовується дихотомія пошуку по іншій координаті (*y*).

Побудова структури даних – дерево регіонів (a range-tree).

1. Спроектуємо точки множини *S* на вісь *OX*.

2. *Нормалізація.* Кожній проекції точки множини S, починаючи із самої крайньої лівої у порядку зростання, присвоїмо цілі числа від 1 до n. Одержимо множину інтервалів на осі OX з цілочисельними кінцями які належать інтервалу $[1, n]$, (рис. 1.5 а) .

3. Для інтервалу $[1, n]$ будуємо дерево відрізків, рис.1.5 б.

4. Прошиваємо вузли побудованого дерева відрізків упорядкованими по y координаті списками точок множини S, проекції яких по x координаті належать відповідним інтервалам вузлів. Одержимо таким чином дерево регіонів.

Алгоритм пошуку

1. Спроектуємо запитний регіон R на вісь OX. Отримаємо інтервал $[k, l]$, який відповідає проекції запитного регіону на вісь OX.

2. Визначимо вузли віднесення в дереві регіонів виконавши операцію вставки інтервалу $[k, l]$ в дерево відрізків побудоване на кроці 3.

3. У знайдених вузлах віднесення застосуємо дихотомію пошуку точок які належать запитному регіону по y координаті.

Теорема 2.13. *Регіональний пошук на площині, що містить n точок можна провести методом "Дерева регіонів" за час* $O(\log^2 n)$, *якщо витрати на пам'ять та попередню обробку складатимуть* $O(n\log n)$.

Існує модифікований алгоритм (Уіаларда-Люкера) [54, 55] цього методу, який отримав назву "метод розшарованого дерева регіонів", в результаті чого була покращена оцінка пошуку до $O(\log n)$, при збереженні тих же самих оцінок для пам'яті та попередньої обробки.

2.2.2 РЕГІОНАЛЬНИЙ ПОШУК У КРУЗІ (ГРПК(АГЗ))

Постановка задачі. На площині задано n точок. Знайти усі точки, які потрапляють в середину круга радіусу R.

Для цього випадку розглянемо декілька підходів. Якщо радіус круга фіксований, проблему можна вирішити за допомогою методу локусів. У 1985 році був запропонований алгоритм [56] із оцінкою $(n,\log n)$. У випадку довільного радіусу і розташування круга відомий метод вирішення цієї проблеми має складність $(n(\log n \log\log n)^2, \log n)$ [57].

2.2.2.1 Алгоритм регіонального пошуку у крузі на основі ідеї методу дерева регіонів

В основі запропонованого методу лежить ідея з методу дерева регіонів для прямокутної області [58, 59]. Це дозволило одержати алгоритм з часом пошуку $O(\log_2 n)$, за рахунок попередньої обробки $O(n\log_2 n)$ та використання $O(n\log_2 n)$ пам'яті. Розглянемо основні кроки алгоритму.

1.Обираємо полярну систему координат (ρ, φ) із центром в точці O (0, 0), рис.2.25.

2. Будуємо два впорядкованих списки з n точок заданої множини S: $U_\varphi = \{P(\varphi_i), i = 1, \ldots, n\}$ – упорядкований за полярним кутом φ відносно початку полярної системи координат (обхід проти годинникової стрілки, $\varphi \geq 0$), $U_\rho = \{P(\rho_j), j = 1, \ldots, n\}$- упорядкований за довжиною полярного радіуса ρ.

3. Проводимо нормалізацію точок списку U_φ. Починаючи з першої точки, поставимо послідовно кожній точці натуральне число, одержавши, таким чином, на основі списку точок U_φ список натуральних чисел $U = \{1, 2, \ldots, n\}$.

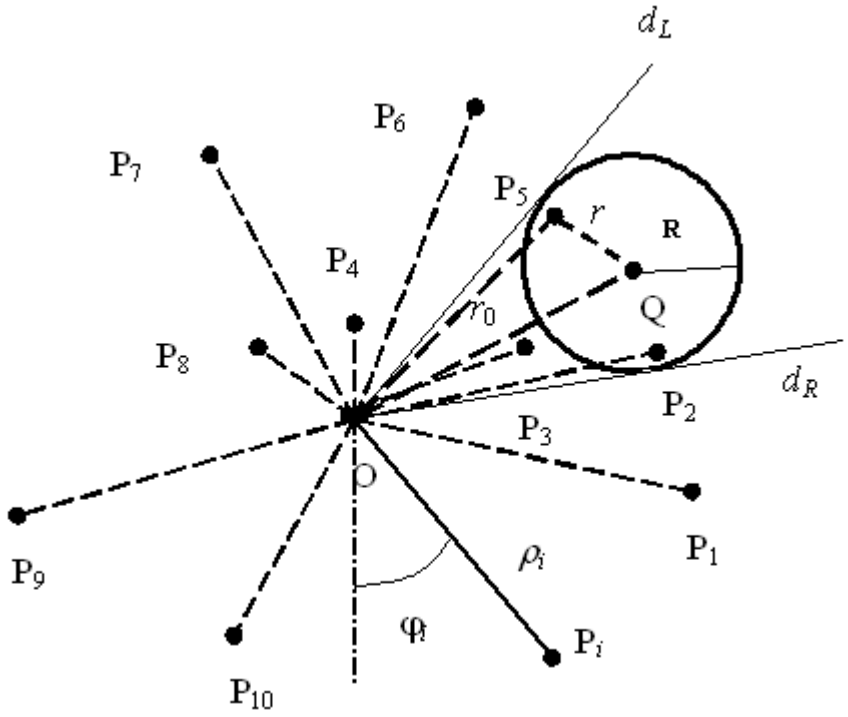


Рис.2.25.

4. Будуємо структуру даних. Це дерево клинів для сектора [1, *n*], вершини якого прошиті відповідними вузлам упорядкованими по ρ списками точок із $U\rho$, рис. 2.26.

5. Проводимо до заданого круга (КРУГ(центр-*q*, радіус – *R*)) дві дотичні з точки O d_L і d_R. Ці дотичні утворюють кути α_L і α_R відповідно. Локалізувавши ці прямі, одержимо клин [*l*, *m*], позначений у відповідності із 4.

6. Далі виконуємо фрагментацію клину [*l*, *m*] на дереві клинів для сектора [1, *n*], визначивши вузли віднесення.

7. У вузлах віднесення, прошитих упорядкованими списками точок за полярним радіусом, знаходимо бінарним пошуком точки, які належать заданому кругу, за формулою $|r| \le R$. Де $|r| = \sqrt{r_0^2 + \rho_i^2 - 2r_0\rho_i \cos(r_0, \rho_i)}$, r_0, ρ_i – довжини полярних радіусів центра заданого кола радіуса *R* та токи P_i відносно точки *O*(0,0), відповідно (i = 1,…, *n*).

Теорема 2.14. *Задачу регіонального пошуку для круга на множині S із n точок на площині можна розв'язати методом клинів за час O*(log*n*) *з витратами O*(*n*log*n*) *на пам'ять та попередню обробку*.

Доведення. ▸ Побудувавши за *O*(*n*log*n*) кроків структуру даних – прошите дерево клинів можна за час *O*($\log_2 n$) визначити вузли віднесення, а потім за час *O*($\log_2 n$) знайти множину точок, яка потрапляє в запитний регіон, витративши при цьому *O*(*n*log*n*) пам'яті. ◂

Цей алгоритм має очевидне узагальнення на випадок *d*-вимірного Евклідового простору. А тому має місце наступна теорема.

Теорема 2.14'. *Задачу регіонального пошуку для кулі в d-вимірному Евклідовому просторі множині S із n точок можна розв'язати методом клинів за час O*($\log^{d-1} n$) *з витратами* O($n\log^{d-1} n$), *які підуть на пам'ять та попередню обробку. Алгоритм методу клинів можна віднести до множини алгоритмів* A_1^k *розв'язання задач класу звідності* $K_{Z:\log N}^{OG}$.

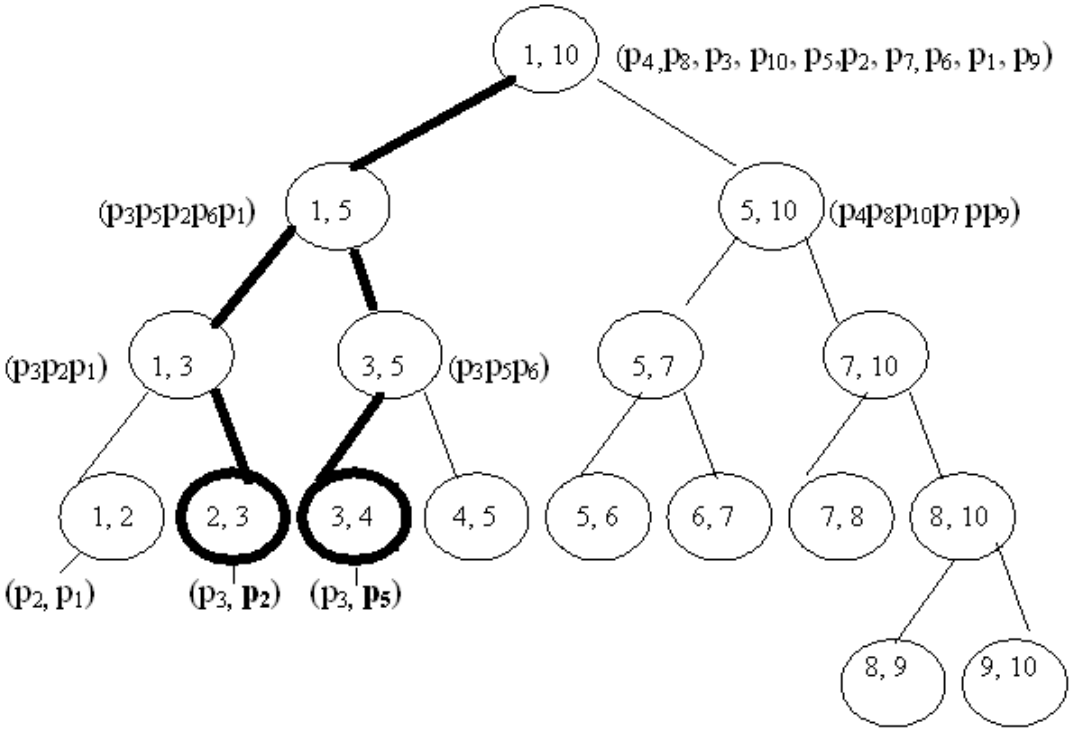


Рис. 2.26. Дерево клинів.

Дійсно, якщо множину точок *S* із *n* точок в *d*- вимірному Евклідовому просторі представити в полярній системі координат, то замість дерева клинів матимемо дерево конусів, а усі кроки алгоритму залишаються не змінними.

2.2.2.2 Метод модифікованого *k-d* дерева для регіонів з криволінійними границями [60].

У цьому методі, для розв'язання задачі у випадку запитного регіону з криволінійною границею, використовується ідея схожа до методу *k-D*-дерева [61]. Не порушуючи загальності, вважатимемо, що множина точок *S* розташована в першому квадранті декартової системи координат. Оскільки запити – масові, доречно виконати етап попередньої обробки.

Алгоритм

Попередня обробка.

1.Через кожну точку множини *S* проведемо промінь з початку координат і чверть коло з радіусом $0p_i$ і центром в початку координат. В результаті утвориться сітка, у вузлах якої розташовані точки з множини *S* , рис.2.27.

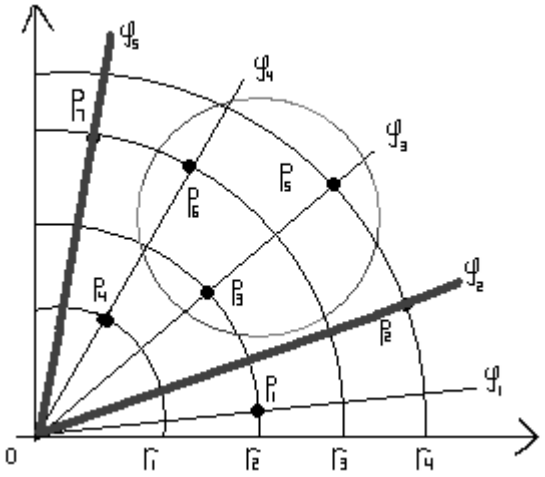


Рис. 2.27.

2. Для множини точок S формуємо лінійно упорядкований за кутом (φ_i) список U^*, кожен елемент якого містить інформацію про кут і радіус.

3. На основі U^* рекурсивно будуємо бінарне дерево, корінь якого містить $\left[\frac{n}{2}\right]+1$ - шу точку, яка розбиває список на два рівнопотужні підсписки U_1^* і U_2^*. В корені лівого піддерева знаходиться точка, яка розбиває лівий підсписок навпіл, а в корені правого піддерева – точка, яка розбиває правий підсписок навпіл. При цьому в кожній вершині дерева зберігається інформація про значення кута, φ_i, і значення радіуса, r_j.

На цьому попередня обробка завершується. Переходимо до етапу пошуку.

Організація пошуку. Задаєтьсая коло з центром $O(x^*,y^*)$ і радіусом R.

1. За логарифмічний час локалізується коло в сітці.

2. Організація пошуку по дереву:

2.1.Починаючи з кореня дерева, за кутом і радіусом перевіряється, чи потрапляє середня в списку точка в сектор , обмежений заданим колом.

2.2. Якщо потрапляє, то перевіряється, чи задовольняє вона обмеження по радіусу, яке для даної точки неважко обчислити, скориставшись геометричними формулами.

2.3. Якщо обидва обмеження вона задовольняє, то вона включається до списку-відповіді.

2.4. Якщо вона потрапляє у сектор кола, але не задовольняє обмеження по радіусу, то вона пропускається і здійснюється перехід на рівень вузлів-синів по дереву пошуку.

2.5. Якщо ж точка не потрапляє у сектор кола, то перевірка не робиться. і до списку-відповіді ця точка не включається, а пошук до синів цього вузла припиняється..

3. Обійшовши таким способом дерево, знайдемо усі точки з множини S, що потрапляють в круг заданого радіусу R.

Приклад. Нехай маємо множину точок на площині $P=\{p_1,p_2,p_3,p_4,p_5,p_6\}$.

Попередня обробка.

1. Введемо декартову систему координат так, щоб усі точки множини P опинилися в першому квадранті.

2. Через кожну точку множини P проводимо промінь з початком в початку координат і чверть коло з радіусом $0p_i$ і центром в початку координат (рис.2.27).

3. Складемо список точок, впорядкований по куту:

$$[(\varphi_1,r_2),(\varphi_2,r_4),(\varphi_3,r_2),(\varphi_3,r_4),(\varphi_4,r_1),(\varphi_4,r_3),(\varphi_5,r_3)].$$

4 . Будується бінарне дерево (рис.2.28)

Попередня обробка на цьому закінчена. Структура даних побудована.

Пошук.

5. Задаємо коло з центром $O(x^*,y^*)$ і радіусом R.

6. Круг локалізується в сітці за логарифмічний час, тобто знаходиться $\min\limits_i \varphi_i=\varphi_2$, $\max\limits_i \varphi_i=\varphi_5$.

7. Тепер переходимо до пошуку по дереву, рис.2.28:

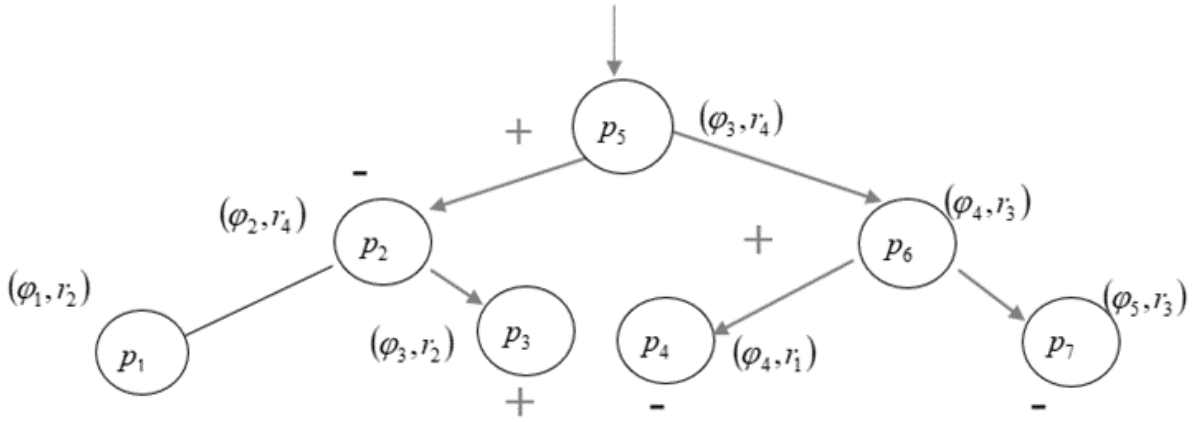


Рис.2.28. Дерево пошуку

7.1. Знаходимось в корені дерева p_5: $\varphi_3 \in (\varphi_3, \varphi_5)$. Елементарними геометричними формулами знаходимо r^5_{min}, r^5_{max}. $r_4 \in \lfloor r^5_{min}, r^5_{max} \rfloor$. Точка p_5 пройшла обидві перевірки. $V = \{p_5\}$.

7.2. Спускаємось нижче. Знаходимось у вершині дерева p_2. Оскільки $\varphi_2 \notin (\varphi_2, \varphi_5)$, то вліво спускатися нема сенсу. Знаходимось у вершині p_3: $\varphi_3 \in (\varphi_2, \varphi_5)$. Знаходимо r^3_{min}, r^3_{max}. $r_2 \in \lfloor r^3_{min}, r^3_{max} \rfloor$. $V = \{p_3, p_5\}$.

7.3. Вершина p_6: $\varphi_4 \in (\varphi_2, \varphi_5)$. Знаходимо r^6_{min}, r^6_{max}. $r_3 \in \lfloor r^6_{min}, r^6_{max} \rfloor$. $V = \{p_3, p_5, p_6\}$. Спускаємось до синів.

7.4. Вершина p_4: $\varphi_4 \in (\varphi_2, \varphi_5)$. Знаходимо r^4_{min}, r^4_{max}. $r_1 < r^4_{min}$.

7.5. Вершина p_7: $\varphi_5 \notin (\varphi_2, \varphi_5)$. Відповідь: $V = \{p_5, p_3, p_6\}$.

8. Завершення.

Теорема 2.15. *Задачу регіонального пошуку для регіону з криволінійними границями на множині S із* n *точок на площині можна розв'язати методом модифікованого* k-d *дерева для регіонів з криволінійними границями за час* $O(\log n)$ *з витратами* $O(n)$ *на пам'ять та* $O(n \log n)$ *на попередню обробку*.

Таким чином, запропоновано ще один підхід до розв'язання задачі регіонального пошуку, який на попередню обробку витрачає $O(n \log n)$ часу, займає $O(n)$ пам'яті, а пошук виконується за час $O(\log n)$ для регіонів з криволінійними границями.

2.2.2.3 Метод матриць [62]

Основна проблема полягає у розробці загального підходу, який би дозволяв будувати алгоритми для регіонів довільної форми. Розглянемо підхід, який дозволяє розробити алгоритм з логарифмічним часом пошуку для запитаних регіонів з криволінійними границями. Розглянемо метод, який назвемо *методом матриць* на прикладі задачі регіонального пошуку для круга. В загальному випадку регіоном пошуку може бути будь-яка геометрична фігура: для площини це область обмежена кривими другого порядку, а для d-вимірного простору E^d ($d > 2$), область обмежена поверхнями вищих порядків.

Алгоритм (покроково)

1. Не порушуючи загальності, розглянемо круг, заданий у першому квадранті декартової системи координат. Позначимо координати його центру $O(x^*, y^*)$, а радіус R. Оскільки задачею є пошук точок, що належать цьому кругу, то можна звузити простір пошуку з $R^{2+} = \{(x, y) | x \geq 0, y \geq 0\}$ до $M = \{(x, y) | x^* - R \leq x \leq x^* + R, y^* - R \leq y \leq y^* + R\}$. Це можна зробити в загальному випадку за час $O(\log_2 n)$, оскільки для кожного з чотирьох обмежень пошук множини точок, які більші або менші за x чи y координатою проводиться за час $O(\log_2 n)$.

2. Без порушення загальності розглянемо одну з чотирьох частин одержаного простору, в якій частина кола є монотонною функцією. Таке розбиття простору здійснюється точкою $O(x^*, y^*)$, яка є центром кола.

3. Нехай на початку була задана множина точок S. Після перших двох кроків алгоритму отримано множину точок $S' = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8\}$, яка належить простору $M = \{(x, y) | x^* - R \leq x \leq x^* + R, y^* - R \leq y \leq y^* + R\}$. Впорядкувавши цю множину точок спочатку по координаті x, а потім по координаті y отримаємо список $\{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8\}$. Операція сортування виконується за час $\theta(n \log_2 n)$, але вона може бути винесена в етап попередньої обробки, як і наступна операція. Кожна точка розбиває площину на чотири частини.

Унаслідок такого розбиття утвориться „сітка”. У „вузлах” такої „сітки” розташовані точки з множини *S’* та точки, що мають координатами комбінації *x* та *y* точок з множини *S’*, рис.2.29. Назвемо їх нульовими.

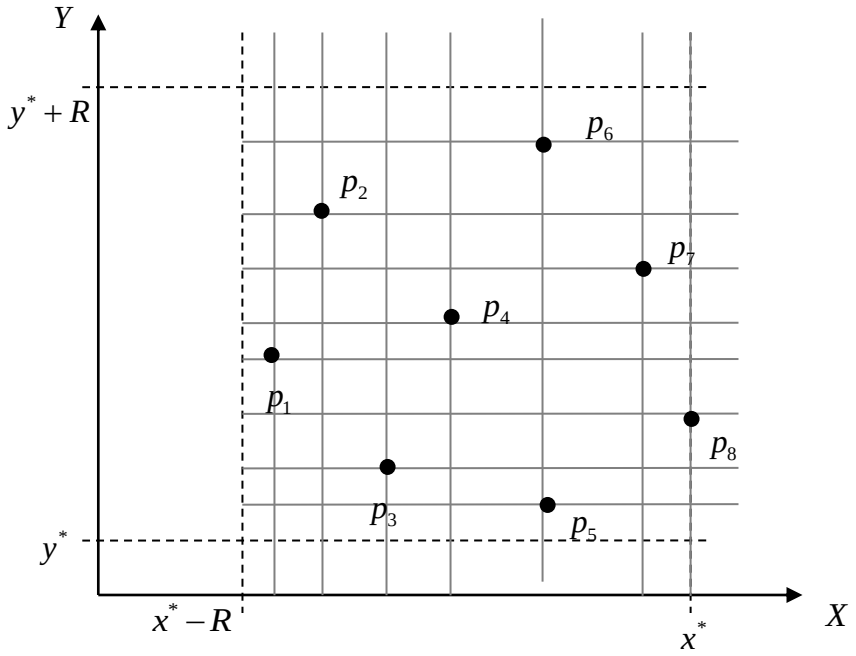


Рис. 2.29.

4. Складемо матрицю розмірності $n \times m$ (в нашому випадку 8×7), в комірках якої будуть зберігатися координати „вузлів”(2.6):

$$K = \begin{pmatrix} (x_1, y_8) & (x_2, y_8) & . & . & . & . & (x_7, y_8) \\ (x_1, y_7) & (x_2, y_7) & . & . & . & . & (x_7, y_7) \\ . & . & & & & & . \\ . & . & & & & & . \\ . & . & & & & & . \\ . & . & & & & & . \\ . & . & & & & & . \\ (x_1, y_1) & (x_2, y_1) & . & . & . & & (x_7, y_1) \end{pmatrix} \qquad (2.6)$$

Її можна подати у вигляді (2.7)

$$\hat{K} = \begin{pmatrix} 0 & 0 & 0 & 0 & p_6 & 0 & 0 \\ 0 & p_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & p_7 & 0 \\ 0 & 0 & 0 & p_4 & 0 & 0 & 0 \\ p_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & p_8 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & p_5 & 0 & 0 \end{pmatrix}, \qquad (2.7)$$

де 0 – позначення комірки, у якій зберігаються координати нульової точки, а p_i – позначення комірки, у якій зберігаються координати точки з множини *S’*. Таким чином, матриця $\hat{K}$ (2.7) містить в кожному рядку і кожному стовпчику хоча б одне значення p_i.

5. За побудованою матрицею $\hat{K}$ формуємо дерево, з кожного вузла якого виходить чотири гілки за наступним правилом: i та j – відповідно індекси по рядках та по стовпчиках матриці $\hat{K}$: p_1 має координати в матриці $\hat{K}$ $(5,1)$, $p_2 - (2,2)$, $p_3 - (7,3)$, $p_4 - (4,4)$, $p_5 - (8,5)$, $p_6 - (1,5)$, $p_7 - (3,6)$, $p_8 - (6,7)$. Для

кожного „вузла” на етапі попередньої обробки визначається множина елементів матриці $\hat{K}$, які мають індекси i та j, такі що належать вузлу і відповідному піддереву, яке виходить із нього. Якщо на певному етапі виявляється, що „вузол” не містить точок із множини *S'*, то далі піддерево для нього не генерується, оскільки зрозуміло, що в жодному листку такого дерева елементів з *S'* не буде. Процес утворення піддерев для вузлів відбувається доти, доки на певному рівні рекурсії (дерево генерується рекурсивно) не виявиться, що елемент із *S'*, $p_{ij} \in \hat{K}$ належать „листку” (i, j). Матриця $\hat{K}$ та відповідне дерево використовують $O(n^2)$ пам'яті.

6. Наступним етапом є пошук на створеному дереві. Для цього послідовно від „кореня” до „листків” „вузли” перевіряються на виконання певних умов:

1) якщо елемент „вузла” з максимальним індексом i та максимальним індексом j, що має відповідне значення в матриці K, $k_{ij} = (x_j, y_i)$ не задовольняє нерівність $(x_j - x^*)^2 + (y_i - y^*)^2 \le R^2$, то і все відповідне піддерево (його елементи) не належить кругу з центром в точці $O(x^*, y^*)$, радіуса R, або він є „нульовим” (порожнім);

2) якщо елемент вузла з мінімальним індексом *i* та мінімальним індексом *j*, що має відповідне значення в матриці K, k_{ij} =(x_j, y_i) задовольняє нерівність $(x_j - x^*)^2 + (y_i - y^*)^2 \le R^2$, то і все відповідне піддерево (тобто його елементи, виписані поряд з кожним вузлом) належать кругу;

3) інакше, перевірка рекурсивно переходить на нижчий рівень і так до дна рекурсії.

7. Застосуємо алгоритм для побудованого дерева.

1) Для „вузла” $\{i \in [1,8], j \in [1,7]\}$ – третій випадок.

$\{i \in [1,4], j \in [1,3]\}$ – перший випадок, бо для k_{43} не виконується $(x_3 - x^*)^2 + (y_4 - y^*) \le R^2$,тому точка p_2 не належить кругу. Викреслюємо її з усіх „вузлів”, що лежать вище поточного.

$\{i \in [1,4], j \in [4,7]\}$ – третій випадок:

$\{i \in [1,2], j \in [4,5]\}$ – перший випадок, вилучається точка p_6;

$\{i \in [1,2], j \in [6,7]\}$ – перший випадок, бо піддерево порожнє;

$\{i \in [3,4], j \in [4,5]\}$ – третій випадок;

$\{i \in [3], j \in [4]\}$, $\{i \in [3], j \in [5]\}$, $\{i \in [4], j \in [5]\}$ – перший випадок, бо порожні;

$\{i \in [4], j \in [4]\}$ – другий випадок, точка p_4 лишається, вона належить кругу;

$\{i \in [3,4], j \in [6,7]\}$ – другий випадок, для k_{36} нерівність $(x_6 - x^*)^2 + (y_3 - y^*) \le R^2$ виконується, тому p_7 лишається;

$\{i \in [5,8], j \in [1,3]\}$ – третій випадок;

$\{i \in [5,6], j \in [1]\}$ – перший випадок, бо точка k_{61} не задовольняє нерівності, p_1 вилучається;

$\{i \in [5,6], j \in [2,3]\}$, $\{i \in [7,8], j \in [1]\}$ – перший випадок, бо вузли порожні

$\{i \in [7,8], j \in [2,3]\}$ – другий випадок, k_{72} задовольняє нерівності, p_3 належить кругу;

5) $\{i \in [5,8], j \in [4,7]\}$ – другий випадок, бо для k_{54} нерівність $(x_5 - x^*)^2 + (y_4 - y^*) \le R^2$ виконується, тому точки p_5 та p_8 лишаються.

Таким чином множина точок з *S'* в корені дерева, що лишилася є розв'язком поставленої задачі. Відповідь: $\{p_3, p_4, p_5, p_7, p_8\}$.

8. Алгоритм аналогічно поширюється і на решту 3/4 круга. Змінюється лише вигляд елементів для перевірки в умовах.

Теорема 2.16. *Задачу регіонального пошуку для регіону з криволінійними границями на множині S із n точок у Евклідовому d- вимірному просторі можна розв'язати методом матриць за час* $O(\log_4 n)$ *з витратами* $O(n^d)$ *на пам'ять та* $O(n\log n)$ *на попередню обробку.*

Оскільки дерево і матриці будуються на етапі попередньої обробки, а під час роботи алгоритму для кожного конкретного регіону здійснюється пошук по тетрагілковому дереві, то складність роботи такого алгоритму $O(\log_4 n)$, перевірка „вузлів" здійснюється за константний час.

Запропонований алгоритм має узагальнення на випадок областей обмежених будь-якими кривими (параболи, гіперболи, еліпси і т.п.), а також на випадок просторів більших розмірностей R^d, де j > 2, тоді відповідно змінюються інтервали монотонності та функції перевірки елементів у вузлах дерева та вид самого дерева. Так, наприклад, якщо розглядати регіональний пошук всередині простору обмеженого кулею в просторі R^3 кожен вузол дерева матиме максимум 8 гілок, оскільки кожна точка тривимірного простору ділить його на 8 підобластей (в R^2 на 4).

Таким чином, під час попередньої обробки будується сітка з n^2 точок, де вузлами є точки координати яких є комбінаціями координат вхідної множини. По заданій матриці точок будується квадродерево (тетрагілкове дерево) у вершинах якого містяться множини точок, які належать поточній півплощині. У цьому алгоритмі для кожної частини круга, де частина кільця являється монотонною функцією, відбувається процедура пошуку по квадродереву вершин, які лежать всередині частини круга.

2.2.3 РЕГІОНАЛЬНИЙ ПОШУК У ОПУКЛОМУ МНОГОКУТНИКУ (ГРПОМ(АГ1))

Постановка задачі. На площині задано n точок. Знайти усі точки, які потрапляють в середину опуклого k – кутника.

На відміну від регіонального пошуку в прямокутному запитному регіоні, мало відомо про структури даних регіонального пошуку в опуклому многокутнику, які б дозволяли отримати логарифмічний час запиту, використовуючи лінійну пам'ять. Фактично, нижні оцінки складності не дають оптимізму на побудову такої структури даних, оскільки час запиту структури даних лінійного розміру, згідно з напівгруповою моделлю, складає $\Omega(n^{1-1/d})$ $(\sqrt{n}\ , d = 2))$. Тому можна виділити наступні запитання на які варто дати відповіді:

1) Який буде мати розмір структури даних, яка б дозволила отримати логарифмічний час пошуку в опуклому многокутнику?

2) Який найкращий час пошуку в опуклому многокутнику можна досягти на структурі даних лінійного розміру?

СТРУКТУРИ ДАНИХ З ЛОГАРИФМІЧНИМ ЧАСОМ ЗАПИТУ.

Для спрощення розглянемо задачу регіонального пошуку в режимі підрахунку для півпростору, який містить S із n точок у E^d . Представимо S структурою даних так, щоб кількість точок S, які лежать над гіперплощиною запиту, можна було швидко підрахувати. Використовуючи двоїстість задач локалізації точки (ЗЛТ) та регіонального пошуку (ЗРП) наведену вище задачу можна звести до задачі локалізації точки: для заданої множини H із n гіперплощин у R^d, визначити кількість гіперплощин H, що лежать над точкою запиту q. Оскільки одна й та ж підмножина гіперплощин лежить над усіма точками однієї комірки $A(H)$, кількість гіперплощин H, що лежать над q, можна визначити, знайшовши комірку $A(H)$, яка містить q. Наступна теорема Чазелле [63] розв'язує поставлену задачу і дає структуру даних з часом запиту $O(\log n)$ для підрахунку у півпросторі.

Теорема 2.17. (ТЕОРЕМА 40.3.1 [63]) *Нехай H — набір із n гіперплощин і* $r \le n$ *— параметр. Встановимо* $s = \lceil \log_2 r \rceil$. *Існує s розрізів* $P_1, ..., P_s$, *таких, що* $P_i \in (1/2^i)$*-розрізом розміру* $O(2^{id})$, *кожен симплекс* P_i *міститься в симплексі* P_{i-1}, *а кожен симплекс* P_{i-1} *містить постійну кількість симплексів* P_i. *Крім того,* $P_1, \ldots, P_s$ *можна обчислити за час* $O(nr^{d-1})$.

Наведемо доведення теореми, подане в [63] , оскільки воно важливе для розуміння алгоритму розв'язання задачі.

Доведення.

1) Оберемо $r = \lceil n/\log_2 n \rceil$.
2) Побудуємо розрізи $P_1, ..., P_s$, для $s = \lceil \log_2 r \rceil$;
3) Для кожного симплексу $\triangle \in P_i$, i < s, зберігаємо вказівники на симплекси P_{i+1}, які містяться в $\triangle$;

4) Для кожного симплексу $\triangle \in P_s$ зберігаємо $H_\triangle \subseteq H$, набір гіперплощин, які перетинають $\triangle$, і $k_\triangle$, кількість гіперплощин H, які лежать над $\triangle$. Оскільки $P_s = O(r^d)$, загальний розмір структури даних дорівнює $O(nr^{d-1}) = O(n^d/\log^{d-1} n)$.

Для точки запиту $q \in R^d$, обходячи вказівники, знаходимо симплекс $\triangle \in P_s$, який містить q, підраховуємо кількість гіперплощин $H_\triangle$, які лежать над q, і повертаємо $k_\triangle$ плюс цю кількість. Загальний час запиту буде $O(\log n)$. Простір можна зменшити до $O(n^d/\log^d n)$, зберігаючи час запиту рівним $O(\log n)$ [64].

Наведений вище підхід можна поширити на задачу регіонального пошуку в режимі підрахунку для симплексів: зберігати розв'язок кожного комбінаторно відмінного симплексу (два симплекси є комбінаторно різними, якщо вони не містять ту саму підмножину S). Оскільки існує $\Theta(n^{d(d+1)})$ комбінаторно різних симплексів, такий підхід вимагатиме зберігання $\Omega(n^{d(d+1)})$. Chazelle та ін. [65] запропонували структуру даних розміром $O(n^{d+\epsilon})$ для будь-якого $\epsilon > 0$, використовуючи багаторівневу структуру даних, з часом пошукового запиту $O(\log n)$. Границю простору можна звести до $O(n^d)$, збільшивши час запиту до $O(\log^{d+1} n)$ [64].

СТРУКТУРИ ДАНИХ ЛІНІЙНОГО РОЗМІРУ.

Більшість структур даних лінійного розміру для пошуку в опуклому многокутнику базуються на деревах розбиття, введених Уіллардом [68] для набору точок на площині. Грубо кажучи, дерево розбиття — це схема ієрархічної декомпозиції, яка рекурсивно розбиває точки на канонічні підмножини та охоплює кожну канонічну підмножину простою опуклою областю (наприклад, симплексом), так що будь-яка гіперплощина перетинає лише частину областей, пов'язаних із нащадками вузлів канонічної підмножини. Час запиту залежить від максимальної кількості регіонів, пов'язаних із дочірніми елементами вузла, який може перетинати гіперплощина. Дерево розбиття, запропоноване Уіллардом, розбиває кожну канонічну підмножину на чотири нащадки, кожен з яких міститься в клині так, що будь-яка лінія перетинає щонайбільше три з них. У результаті час, витрачений на звіт про всі k точок, що лежать всередині трикутника, складає $O(n^\alpha + k)$, де $\alpha = \log_4 3 \approx 0{,}793$. Великий прорив у регіональному пошуці для опуклого многокутника був зроблений Хауслером і Вельцлем [69]. Вони сформулювали регінальний пошук в абстрактній постановці і, використовуючи елегантні ймовірнісні методи, запропонували рандомізований алгоритм для побудови лінійного дерева розбиття з часом пошуку $O(n^\alpha)$, де $\alpha = 1 - (1/(d(d-1)+1)) + \epsilon$ для будь-якого $\epsilon > 0$. Найвідоміша структура даних лінійного розміру для пошуку в опуклому многокутнику, яка майже відповідає нижнім оцінкам, згаданим нижче, була вперше запропонована Матусеком [64] і згодом спрощена Chan [70]. Ці структури даних дають час пошуку в опуклому многокутнику в режимі підрахунку $O(n^{1-1/d})$ (відповідно $O(n^{1-1/d+k})$) і базуються на теоремі Матусека [66].

Теорема 2.18 (40.3.2. Matouˇsek [66])*Нехай маємо множину S із n точок* E^d *, і параметр r* : $1 < r \le n/2$. *Тоді існує сімейство пар* $\Pi = \{(S_1, \Delta_1), \ldots, (S_m, \Delta_m)\}$ *таких, що* $S_i \subseteq S$ *лежать в середині симплексів* Δ_i, $n/r \le |S_i| \le 2n/r$, *які не перетинаються* $S_i \cap S_j = \emptyset \ \forall \ i \neq j$, *кожна гіперплощина перетинає не більше* $cr^{1-1/d}$ *симплексів* Π; *c - константа. Якщо r константа, то* Π *можна побудувати за час* $O(n)$.

Використовуючи теорему Матусека [66], дерево розбиття T можна побудувати наступним чином.

1) Кожен внутрішній вузол v у T асоціюється з підмножиною $S_v \subseteq S$ і симплексом Δ_v, що містить S_v;

2) Корінь T асоціюється з S і R^d.

3) Виберіть r як достатньо велику константу.

4) Якщо $|S| \le 4r$, T складається з одного вузла, і в ньому зберігаються всі точки S.

5) Інакше будуємо сімейство пар $\Pi = \{(S_1, \Delta_1), \ldots, (S_m, \Delta_m)\}$ за допомогою теореми [66].

6) Рекурсивно будуємо дерево розбиття T_i для кожного S_i і приєднуємо T_i як i-те піддерево u. Корінь T_i також зберігає Δ_i .

Загальний розмір структури даних буде лінійним, і його можна побудувати за час $O(n\log n)$. Оскільки будь-яка гіперплощина перетинає щонайбільше $cr^{1-1/d}$ симплексів Π, час запиту в режимі звіту в симплексі становить $O(n^{1-1/d+\log rc} + k)$; член $\log_r c$ в експоненті можна звести до будь-якої як завгодно малої додатної константи ϵ, вибравши r достатньо великим. Хоча час запиту можна покращити до $O(n^{1-1/d}\text{polylog}(n) + k)$, вибравши r як n^ϵ, сильніша версія теореми Матусека [66], яка будує симпліціальне розбиття, ієрархічно аналогічне теоремі 2.17, замість того, щоб будувати його на кожному рівні незалежно, приводить до структури даних лінійного розміру з часом запиту $O(n^{1-1/d} + k)$. Рандомізований алгоритм Чана [70] створює ієрархічне симпліціальне розбиття, в якому (відносні) внутрішні симплекси на кожному рівні попарно не перетинаються, і вони разом індукують ієрархічне розбиття R^d. Оптимальні час і пам'ять для регіонального пошуку в опуклому

многокутнику можуть бути досягнуті шляхом поєднання лінійних розмірів і логарифмічних структур даних під час запиту. Якщо точки в S лежать на b-вимірній алгебраїчній поверхні сталого ступеня, на запит в режимі підрахунку в опуклому многокутнику можна відповісти за час $O(n^{1-\gamma+\epsilon})$, використовуючи лінійний простір, де $\gamma = 1/\lfloor (d + b)/2 \rfloor$ [[71].

ЗАПИТ ДЛЯ ПІВПРОСТОРУ

Пошуковий запит для такого запитного регіону як півпростір можна отримати швидшим, ніж для симплексного регіону, використовуючи дрібне розрізання. Для простоти припустимо, що запитний півпростір лежить нижче його обмежувальної гіперплощини. Тоді розглянемо більш просту задачу: чи містить півпростір якусь вхідну точку. За допомогою перетворення подвійності задач ЗЛТ та ЗРП запит для півпростору в R^d можна сформулювати так: чи лежить запитна точка $q \in R^d$ нижче всіх гіперплощин у заданому наборі H гіперплощин у R^d. Цей запит еквівалентний запиту про те, чи лежить q всередині опуклого многогранника $P(H)$, визначеного перетином півпросторів, що лежать під гіперплощинами H. Для $d \leq 3$ запит про розташування точки в $P(H)$ виконується за оптимальний час $O(\log n)$. При цьому використовується $O(n)$ памяті та O(nlog n) часу піде на попередню обробку, оскільки $P(H)$ має лінійний розмір [72]. Для $d \geq 4$ запит про приналежність точки P(H) стає більш складним, і відповідь на запит можна отримати за допомогою структури даних на основі дрібного розрізу. Наступна теорема Матусека [67] може бути використана для побудови структури даних для перевірки належності точки $P(H)$:

Теорема 2.19 (ТЕОРЕМА 40.3.3 Матусек [67]) *Нехай H — набір із n гіперплощин і* $r \leq n$ *— параметр. Неглибоке (1/r)-зрізання рівня* 0 *відносно H розміру* $O(r^{\lfloor d/2 \rfloor})$ *може бути обчислене за* час $O(nr^c)$, *де c є константою, що залежить від d.*

Інші версії задачі регіонального пошуку можна класифікувати на основі типу області пошуку. Один такий клас отримується шляхом вибору домену пошуку як k-кутник (пошук в многокутнику. Зокрема Уіллард розробив алгориитм [68], який пізніше був удосконалений [73].

Таким чином роботи Хауслера та Велзла, Кларксона, і інших дослідників не лише перегорнули нову сторінку в геометричному пошуку, а й надали значний імпульс для розвитку обчислювальної геометрії в цілому. Завдяки цим роботам та наступним публікаціям було досягнуто значного прогресу в розробці ефективних алгоритмів та структур даних і доведенні нетривіальних нижніх границь складності. З теоретичної точки зору, на сьогодні проблема геометричного пошуку практично повністю розв'язана[74]. Найбільш важливими елементами сучасних схем розв'язання задач регіонального пошуку є ε-мережі (ε-nets), 1/r-розрізання (1/r-cuttings), дерева розбиття (partition trees) та багаторівневі структури даних(multi-level data structures) [74,63,75].

2.2.3.1 Метод двоїстості

У цьому алгоритмі реалізується ідея заснована на теоремі Чазеле 2.17 [63] та теоремі Матусека 2.18 [66] й описана у попередніх розділах 3.3. Алгоритм розв'язання складається з наступних етапів:

Алгоритм (покроково)

1. Перехід до двоїстого геометричного простору.

На площині **геометрична двоїстість**(geometric duality) встановлює взаємну відповідність між точками та прямими. Ми використовуватимемо таку відповідність: точці $p = (a, b)$ відповідає двоїста пряма $D(x) = l = \{(x,y) \mid y = 2a - b\}$, а прямій l відповідає двоїста точка $D(l) = p = (a, b)$. Таким чином можна побудувати двоїсте відображення для будь-яких точок та прямих (окрім вертикальних – їм відповідають ***невизначені точки*** з нескінченними координатами)[74, 66].

Отже, перейдемо до двоїстого геометричного простору скориставшись властивістю геометричної двоїстості: множину точок P перетворимо у множину прямих $H = D(P)$, а L - множину прямих, на яких лежать відрізки-сторони опуклого многокутника R – у множину точок $D(L)$. Випадок невизначених точок ми обробляємо окремо на етапі передобробки – перевизначаємо множину P, відкинувши зайві точки за допомогою локалізації проекції таких прямих на вісь OX .

Найважливішою для нас властивістю двоїстості є те, що якщо точка знаходиться вище прямої в первинному геометричному просторі, то в двоїстому це відношення зберігається (звичайно, в інвертованому вигляді).

2. Розбиття площини на грані та побудова дерева пошуку

Нехай H – скінчена множина прямих на площині. Ці прямі розбивають площину на опуклі множини – грані (інколи вживається термін клітини) розмірності 2. Тобто множина H прямих площини визначає

декомпозицію цієї площини на множину граней потужності $O(N^2)$ [66]. Така декомпозиція є розбиттям площини, назвемо його A(H). Нам потрібно побудувати структуру даних, яка б забезпечувала ефективну локалізацію точки площини в розбитті A(H)

Теорема 2.20 [76] .*Локалізація точки серед N гіперплощин може бути виконана за час O(log N), з часом передобробки O(N^2) .*

Отже, можливо виконати локалізацію за час $O(\log N)$, хоча стандартні алгоритми локалізації (наприклад, деталізації тріангуляції) виконуватимуться за час $O(\log N^2)$. В конструктивному доведенні вищезазначеної теореми використовувалася ієрархічна структура даних, що відповідала послідовним (1/r)-розрізанням площини[74-76].

Для нашого пошуку ми скористаємося іншою конструкцією. Це буде дерево, що будуватиметься наступним чином:

1) нехай $H = \{ h_1,h_2,..,h_N \}$, кореню дерева V відповідає вся площина R^2;

2) розбиваємо площину прямою h_1 на 2 грані – розбиття $A(\{h_1\})$, зберігаємо їх у вузлахV_1 і V_2;

3) розбиваємо розбиття $A(\{h_1\})$ прямою h_2 , отримуємо розбиття $A(\{h_1,h_2\})$, що складається з 4 граней, зберігаємо в вузлах;

4) будуємо розбиття $A(\{h_1,h_2,h_3,h_4\})$, потім $A(\{h_1,h_2,..,h_8\})$, $A(\{h_1,h_2,..,h_{16}\})$, ..., $A(H)$.

Якщо під час ітерації грань, відповідна вузлу не ділиться, вузол видаляється з поточного рівня та переноситься на наступний. Таким чином, утворюється дерево локалізації висотою $\log N$ (рис.2.30).

Вузли дерева містять інформацію про грань, якій вони відповідають (координати вершин), а також – інформацію про прямі, які відсікли відповідну грань і лежать нижче неї.

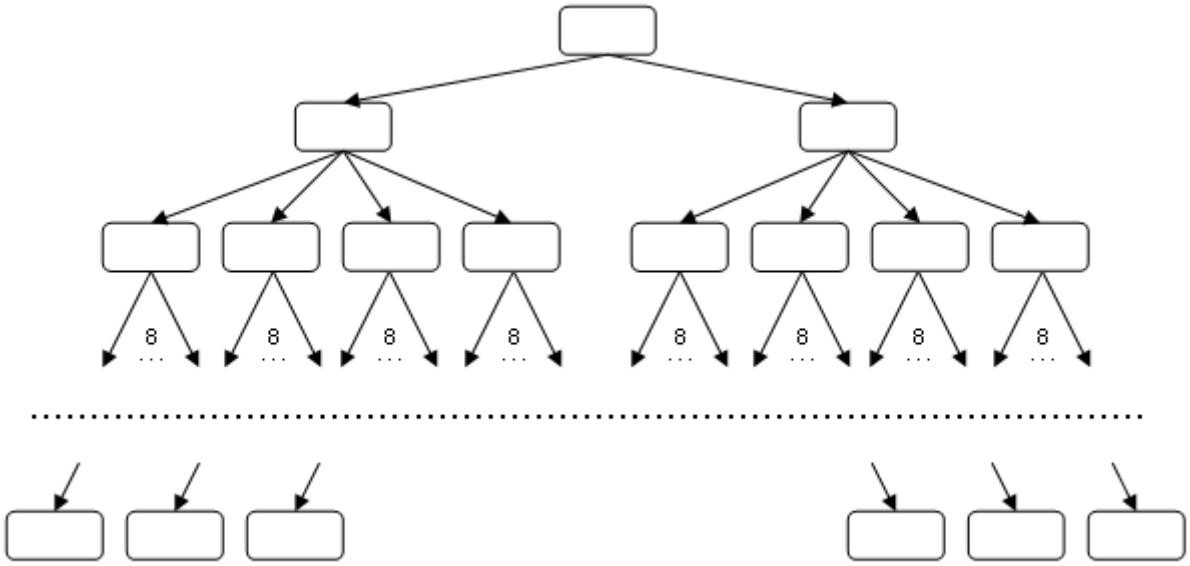


Рис.2.30.

3. Локалізація точок в планарному розбитті

На цьому етапі нам потрібно локалізувати точки множини D(L) у розбитті A(H).Для цього скористаємось побудованим деревом пошуку. При проході по дереву пошуку ми накопичуємо інформацію з його вузлів про відсікаючи прямі, і, таким чином, дійшовши до листка дерева ми знаємо, які прямі лежать нижче відповідної йому грані, а отже, і нижче точки множини D(L), локалізованої в цій грані розбиття A(H). Позначимо множину прямих, що лежать нижче j-ої точки D(L) , K_j , $j = \overline{1,|\ D(L)\ |= k}$.

4. Побудова звіту за результатами послідовних локалізацій.

1. Запитний опуклий многокутник (з виключеними вертикалями) можна розбити на 2 ланцюги, монотонні осі Ox – верхній та нижній, назвемо їх відповідно верхньою та нижньою опуклими оболонками.

2.Для верхньої опуклої оболонки запитного регіону будуємо множину всіх прямих A(H), що лежать над відповідними точками D(L).

3. Для нижньої опуклої оболонки запитного регіону будуємо множину всіх прямих A(H), що лежать під відповідними точками D(L). Перетин цих множин дасть множину прямих, які відповідають двоїстим точкам

що в нашому первинному геометричному просторі містяться всередині запитного опуклого многокутника [74].

Алгоритм завершено.

Оцінки складності алгоритму

Теорема 2. 21. *Задачу регіонального пошуку для опуклого k-кутника на множині S із n точок на площині методом двоїстості можна розв'язати за час* $O(k \cdot \log n)$, *для* $k << n$ *з використанням* $O(n^2)$ *пам'яті та часу на попередню обробку.*

Доведення. Локалізація точки по дереву пошуку висотою *log N* очевидно вимагає $O(\log n)$ часу. Оскільки виконується k локалізацій, то загальна оцінка складності у найгіршому випадку буде $O(k \cdot \log n)$. Окремим етапом є розгляд вертикальних ребер, але, знову ж таки, оцінка складності цієї дії - $O(\log n)$.

Висновки.

Ми розглянули оригінальний алгоритм регіонального пошуку з опуклим многокутник запитним регіоном. Метод має теоретичну оцінку часу виконання, що наближається (за умови виключення з розгляду деяких факторів) до нижньої границі складності для цього типу задач.

Подальший розвиток запропонованого методу можливий в наступних напрямках:

- пошук інших ізоморфних чи гомеоморфних відображень з одного геометричного простору в інший для удосконалення звідності задачі;

- удосконалення практичної реалізації

- використання інших, більш „елегантних" та потужних структур даних (наприклад (1/r)-розрізань тощо)

- теоретичні удосконалення задля виключення фактору множника k з оцінки складності

- більш формальний опис методу, структури даних та доведення оцінки складності.

Щодо недоліків даної роботи та методу в цілому можна зазначити використання пошуку в півпросторі для розв'язання задачі пошуку в опуклому многокутнику – можливо, і навіть напевне, можна переробити алгоритм таким чином, щоб позбутися послідовних уточнень розв'язку.

Хотілося б зазначити, що робота використовує досить „свіжі" наробки сучасної геометричної думки. Головним плюсом застосованого методу можна вважати використання ідеї геометричної двоїстості, тобто переходу до іншого простору, яка в узагальненому вигляді може застосовуватися до широкого класу задач. Метод також ілюструє методику звідності задач, що є одною з основних парадигм розв'язання задач обчислювальної геометрії.

2.2.3.2 Наївний алгоритм регіонального пошуку в опуклому многокутнику(O(nlogk))

Розглянемо простий алгоритм регіонального пошуку в опуклому многокутнику з часом пошуку O(nlogk). Цей алгоритм не оптимальний, проте для невеликої множини точок він може бути корисним.

Алгоритм

Нехай маємо множину S із n точок на площині і запитний регіон – опуклий k -кутник ($k<<n$).

1.Вершини опуклого многокутника впорядкуємо за полярним кутом відносно довільної внутрішньої точки q, за яку можна взяти, наприклад, центроїд трьох вершин многокутника M.

2. Проведемо k променів з точки q через вершини многокутника. Ці промені розіб'ють площину на n клинів, кожен з яких розбивається стороною многокутника на дві частини.

3. За допомогою двійкового пошуку знаходимо клин, в якому лежить запитна точка P (промені розташовані в порядку зростання їх полярних кутів проти годинникової стрілки), рис. 2.31.

4. Точка P лежить між променями p_i та p_{i+1} тоді і тільки тоді, коли кут (Pqp_{i+1}) додатний, а кут (Pqp_i) від'ємний. Коли точки p_i та p_{i+1} знайдено, то точка P буде внутрішньою тоді і тільки тоді, коли кут $(p_i p_{i+1} P)$ від'ємний.

Час відповіді на запит про належність кожної з n точок опуклому k - кутнику дорівнює $O(\log k)$. Для всієї множини S із n точок час пошуку буде $O(n\log k)$ при витраті $O(n)$ пам'яті та попередньої обробки.

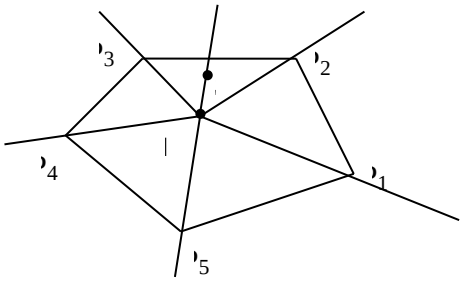


Рис.2.31.

На рис.2.32 показано приклад роботи програмної реалізації алгоритму.

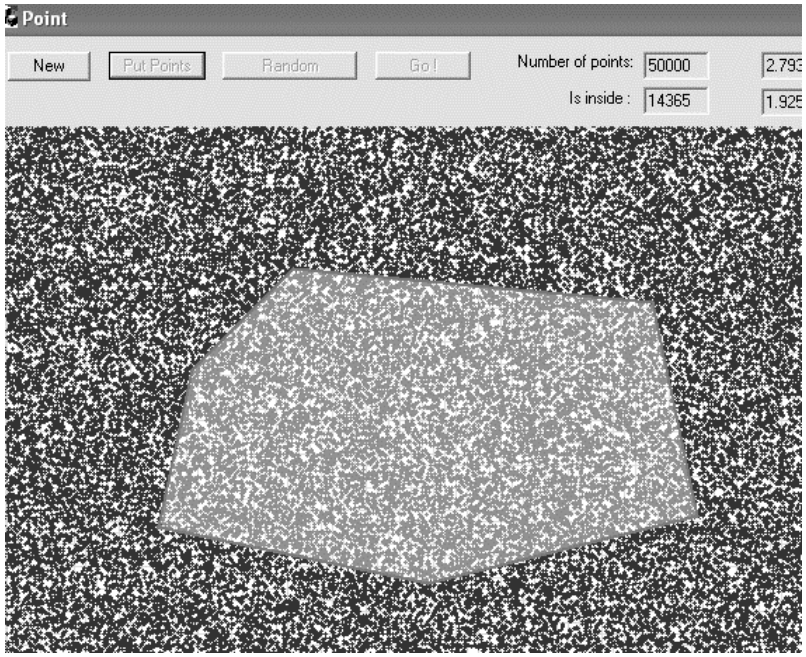


Рис.2.32. Приклад роботи програми. Червоним виділені точки, що знаходяться у многокутнику. Чорним- точки, що знаходяться поза многокутником.

2.2.4 РЕГІОНАЛЬНИЙ ПОШУК У ПРОСТОМУ МНОГОКУТНИКУ (ГРППМ(АГ2))

Постановка задачі. На площині задано S із n точок. Знайти усі точки, які потрапляють в середину простого k –кутника.

2.2.4.1 Алгоритм з використанням Діаграми Вороного

В цьому алгоритмі реалізована ідея регіонального пошуку для запитного регіону у вигляді простого многокутника, за допомогою діаграми Вороного. Алгоритм демонструє високу продуктивність в середньому випадку, що підтверджено емпірично [77]. Розглянемо алгоритм регіонального пошуку у простому многокутнику (рис.2.33) заснований на діаграмі Вороного.

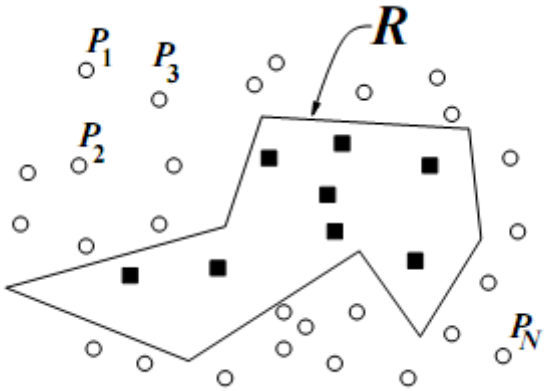


Рис.2.33. Задача регіонального пошуку у простому многокутнику.

Алгоритм

1. Спочатку ми будуємо діаграму Вороного для множини точок *S* в якості попередньої обробки для основного алгоритму.

2.Також створюємо масив *A* довжини *n*, кожен елемент якого ініціалізуємо значенням ***False***. Це означає, що кожна вхідна точка ще не є перевіреною.

3. Якщо точка P_i перевірена, то встановлюємо $A[i]$ рівним ***True.***

4. Для кожного запиту, також здійснюється попередній крок, у якому створюється черга *Q*, що містить усі вхідні точки, що вже є перевіреними, але їхні сусіди ще не є повністю перевіреними.

5. Спочатку *Q* – порожня. Потім, виконуються 4 кроки для кожного запиту. Крок 4 є підготовчим для наступних запитів.

- ***Крок 1*** (Локалізація точки на діаграмі Вороного). Обираємо довільну граничну точку запитного регіону, наприклад Q_0, і знаходимо найближчу до Q_0 точку P_s з множини *S* із *n* вхідних точок.
- ***Крок 2*** (див. рис. 2.34). Починаючи з точки Q_0, здійснюємо обхід по точкам, що належать клітинці діаграми, яку перетинає ребро запитного регіону.

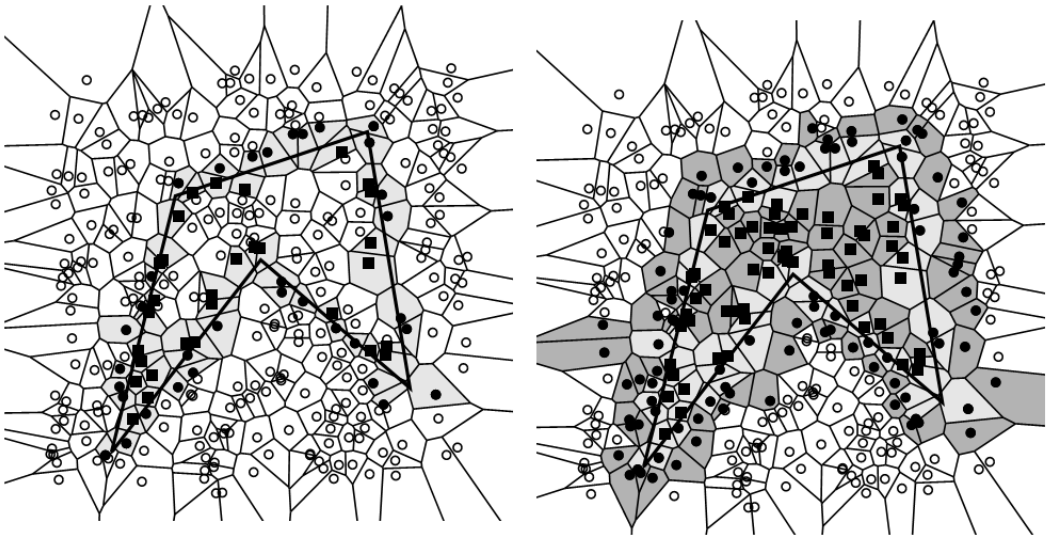


Рис. 2.34. Жирна лінія відображає границю запитного регіону R, жирні кружки і квадратики відповдають точкам, що відповідають обходу на кроці 2. Світіло-сірі многокутники відображають клітинки діаграми Вороного відповідних точок.

For кожної з вхідних точок P_i (i = 1, 2,…, n), що зустрічаються у нашому обході **begin**
If A[i] is False, **begin**
Insert P_i into Q
If P_i лежить всередині R, вивести P_i.
Set A[i] True.
End
End

- ***Крок 3*** (Основна частина, див. Рис. 3.16).

While Q не порожня, **begin**
Видаляємо точку P_i з Q
For кожної точки $P_{i'}$ що є сусідньою для P_i, **begin**
If A[i'] is False, **begin**
If $P_{i'}$ лежить всередині R вивести $P_{i'}$ і вставити $P_{i'}$ в Q
Set A[i'] True.
End
End
End

Рис 2.35. Третій крок.

- ***Крок 4*** **.** Встановити всі занчення елементів масиву A рівними False.

Оцінки складності алгоритму

Теорема 2.22. *Розв'язати задачу регіонального пошуку для простого многокутника методом на основі діаграми Вороного можна за час* $O(n)$, *з використанням* $O(n)$ *пам'яті та* $O(n\log n)$ *попередньої обробки*.

Доведення.
Попередня обробка. На етапі попередньої обробки здійснюється побудова діаграми Вороного, що дає оцінки $(O(n\log n),O(n))$: $O(n\log n)$ час, який витрачається на побудову діаграми Вороного [6]:
Алгоритм.
Крок 1- $(O(n),O(n))$;
Крок 2 - $(O(n),O(n))$: у найгіршому випадку обійдемо всі точки(запитний регіон перетинає кожну клітинку діаграми Вороного);
Крок 3- $(O(n),O(n))$: найгірший випадок, коли всі точки лежить всередині запитного регіону.
Отже в найгіршому випадку оцінка складності часу даного алгоритму є лінійною, але він є достатньо ефективним по усередненим результатам [77]. На рис. 2.36 подано приклад реалізації алгоритму.

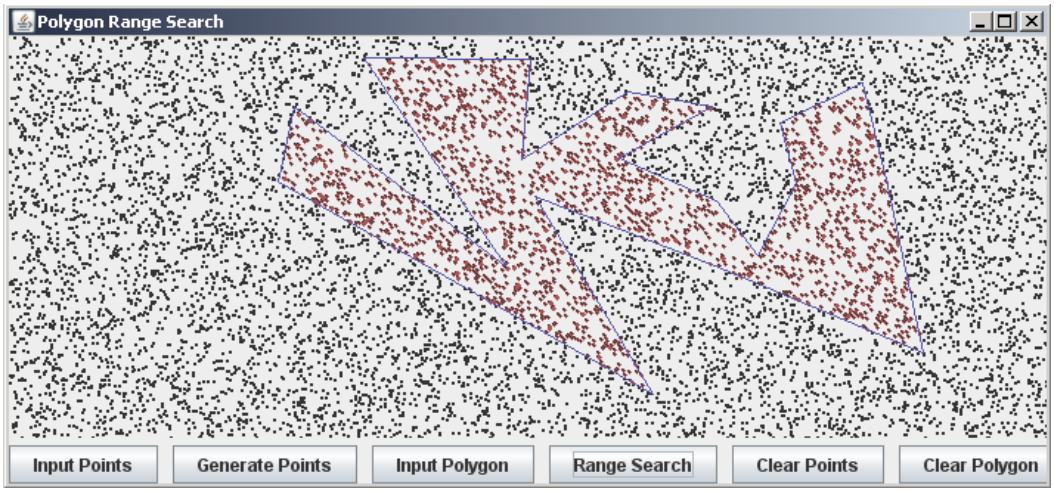


Рис 2.36. Робота програми. Регіональний пошук на великій кількості точок.

Висновки

Реалізовано ефективний алгоритм розв’язання задач регіонального пошуку на площині для простого многокутника. В цьому методі, на етапі попередньої обробки здійснюється побудова діаграми Вороного, вона використовується для зменшення кількості точок для обходу в основному кроці алгоритму. Отримано високі показники продуктивності для загальних випадків, що значним чином залежать від площі запитного регіону, і незначним – від кількості вхідних точок. Більше того, досліджено, що цей алгоритм показує кращі результату у більшості випадків [77]. Лише коли кількість результуючих точок велика і кількість кутів запитного регіону також велика, метод, що базується на діаграмі Вороного програє, алгоритму k-d дерева після тріангуляції. Тому з практичної точки зору, алгоритм, що було запропоновано може замінити багато інших алгоритмів для розмірності простору 2.

Література

1. В.М. Терещенко. Аналіз методів розв'язання оптимізаційних задач обчислювальної геометрії : навч. посіб. / В. М. Терещенко. – К. : ВПЦ "Київський університет". – 2022. – 112 с
2. 9. Dave Mount. Data Structures.\ Lecture Notes. 2001. P. 123. http://www.cs.umd.edu/users/mount/420/Lects/420lects.pdf
3. 10. M.T. Goodrich, R. Tamassia, and D. Mount, Data Structures and Algorithms in C++, Second Edition, John Wiley and Sons, Inc., 2011.
4. 11. M.T. Goodrich, R. Tamassia, and M. Goldwasser, Data Structures and Algorithms in Python, John Wiley and Sons, Inc., 2013.
5. 12. M.T. Goodrich, R. Tamassia, and M. Goldwasser, Data Structures and Algorithms in Java, Sixth Edition, John Wiley and Sons, Inc., 2014.
6. 5. F. Preparata and M.I. Shamos. Computational Geometry: An introduction. Springer-Verlag, Berlin, 1985, -475 p.
7. Atsuyuki Okabe, Barry Boots, Kokichi Sugihara, Sung Nok Chin. Spatial Tessellations. Second Edition.New York. -2000.
8. Renati Descartes. Principia philosophiae. Amstelodami, Typographia Blaviana M DC Lxxxv. -1644.
9. Gustav Lejeune Dirichlet (1850). Über die Reduktion der positiven quadratischen Formen mit drei unbestimmten ganzen Zahlen. Journal für die Reine und Angewandte Mathematik, 40:209-227.
10. A. H. Thiessen, “Precipitation Averages for Large Areas,” Monthly Weather Review, Vol. 39, No. 7, 1911, pp. 1082-1084.
11. Voronoi G. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Deuxième Mémoire. Recherches sur les parallélоèdres primitifs. Journal für die reine und angewandte Mathematik. 1908. 134(3): 198–246; 1908. 134(4): 247–287; 1909. 136(2): 67–178.
12. M. I. Shamos and D. Hoey, Closest-point problems, Sixteenth Annual IEEE Symposium on Foundations of Computer Science, pp. 151-162 (Oct. 1975).
13. D. T. Lee, Proximity and reachability in the plane, Tech. Rep. No. R-831, Coordinated Sci. Lab., Univ. of Illinois at Urbana, IL, 1978.
14. D. T. Lee, Two dimensional Voronoi diagram in the Lp-metric, J. ACM 27,604-618 (1980a).
15. 46. S. J. Fortune, A sweep line algorithm for Voronoi diagrams, Algorithmica 2 (1987), 153-174.
16. GUIBAS, L. J., KNUTH, D. E., AND SHARIR, M. 1992. Randomized incremental construction of Delaunay and Voronoi diagrams. Algorithmica volume 7, pages 381–413-. -1992.
17. B. N. Delone, “On Emptiness of the Sphere,” Izv. Akad. Nauk SSSR OMEN, No. 4, 793–800 (1934).
18. Henry, S., Christian, J.A.: Absolute triangulation algorithms for space exploration. J. Guid. Control Dyn. 46(1), 21–46 (2023). https://doi.org/10.2514/1.G006989
19. Vasyl Tereshchenko, Yaroslav Tereshchenko. Point Triangulation using Graham’s Scan . Journal of Data Processing Vol., 4, N 3, September, 2014, pp. 100-105
20. Терещенко В.М. Оптимізація структур даних в методах локалізації точки / Терещенко В.М., Скляр П.В.// Вісник київського університету. Серія: фіз.-мат. науки.-2011.-Вип. 2.-С. 159-162.
21. 33. Mark de Berg, Otfried Cheong, Marc van Kreveld, Mark Overmars. Computational Geometry:Algorithms and Applications.Berlin Heidelberg: Springer-Verlag,- 2nd, revised edition.- 2008. – 386 p.
22. Goodman J.E., O'Rourke J. , C. D. Toth. Handbook of discrete and computational geometry. CRC, Press, - 3ed. – 2018. - 1927 p.
23. D.P. Dobkin and R.J. Lipton. Multidimensional searching problems. SIAM J. Comput., 5:181–186, 1976.
24. N. Sarnak and R.E. Tarjan. Planar point location using persistent search trees. Commun. ACM, 29:669–679, 1986.
25. H. Edelsbrunner, L.J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. SIAM J. Comput., 15:317–340, 1986.
26. D.G. Kirkpatrick. Optimal search in planar subdivisions. SIAM J. Comput., 12:28– 35, 1983.
27. F.P. Preparata. A new approach to planar point location. SIAM J. Comput., 10:473– 482, 1981.

28. Терещенко В.М. Оптимальний алгоритм локалізації точки для тривимірного простору/ Терещенко В.М. // Вісник київського університету. Серія: фіз.-мат. науки.-2013.-Вип. 1.-С. 235-238 (ФВ).
29. В.М. Терещенко. Модифікація методу ланцюгів для задачі локалізації точки / Терещенко В.М. // Вісник київського університету. Серія: фіз.-мат. науки.-2013.-Вип. 2.-С. 233-236 (ФВ).
30. Vasyl Tereshchenko ; Tania Kolianova ; Yaroslav Tereshchenko . An approach to triangulate area bounded simple polygons //IEEE First Ukraine Conference on Electrical and Computer Engineering (UKRCON).- Kyiv, May 2017.- P. 655-659 (SCOPUS).
31. Vasyl Tereshchenko, Yaroslav Tereshchenko. Triangulating a region between arbitrary polygons / International Journal of Computing.- 2017.- Vol. 16, Issue 3.- P 160-165.
32. R.J. Lipton and R.E. Tarjan. Applications of a planar separator theorem. SIAM J. Comput., 9:615–627, 1980.
33. Bilardi and F. P. Preparata, Probabilistic analysis of a new geometric searching technique, unpublished manuscript, 1981.
34. D. T. Lee and F. P. Preparata, Location of a point in a planar subdivision and its applications, SIAM Journal on Computing 6(3),594-606 (Sept. 1977).
35. M.T. Goodrich and R. Tamassia. Dynamic trees and dynamic point location. In Proc. 23rd ACM Sympos. Theory Comput., pages 523–533, 1991.
36. O. Devillers, S. Pion, and M. Teillaud. Walking in a triangulation. Int. J. Found. Comp. Sci., 13.:181–199, 2002.
37. H. Edelsbrunner, G. Haring, and D. Hilbert. Rectangular point location in d dimensions with applications. Comput. J., 29:76–82, 1986.
38. Y. Giora and H. Kaplan. Optimal dynamic vertical ray shooting in rectilinear planar subdivisions. ACM Trans. Algorithms, 5:28:1–28:51, 2009.
39. M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. Meyer auf der Heide, H. Rohnert, and R.E. Tarjan. Dynamic perfect hashing: upper and lower bounds. SIAM J. Comput., 23:738–761, 1994.
40. Abdullah Zawawi Talib та Ahmad Izani Md Ismail. “Three-Dimensional Extensi- on of Kirkpatrick’s Planar Point Location Method”. B: *Proceedings Of The 2nd International Conference On Information Technology, Jordan*. 2005.
41. Bernard Chazelle та Micha Sharir. “An Algorithm for Generalized Point Location and Its Applications”. B: *J. Symbolic Computation* 10 (1990), с. 281—309.
42. Csaba D. Toth. “Convex Subdivisions with Low Stabbing Numbers”. B: *Periodica Mathematica Hungarica* 57 (2008), с. 217—225.
43. Andrea Danyluk. *CS 136 - Lectur 27*. http://www.cs.williams.edu/~andrea/cs136/Lectures/Lec27.html. 2000.
44. John Lacono. A 3-D visualization of kirkpatrick's planar point location algorithm.// Conference: Proceedings of the 19th ACM Symposium on Computational Geometry, June 8-10, 2003, San Diego, CA, USA, pp 377.
45. Bernard Chazelle. Convex Partitions of Polyhedra: A Lower Bound and Worst-Case Optimal Algorithm//SIAM Journal on ComputingVol. 13, Iss. 3 (1984)10.1137/0213031
46. P. van Emde Boas. Preserving order in a forest in less than logarithmic time. Proceedings of the 16th Annual Symposium on Foundations of Computer Science, October 1975 Pages 75–84. https://doi.org/10.1109/SFCS.1975.26.
47. P. van Emde Boas, R. Kaas, and E. Zijlstra. Design and implementation of an efficient priority queue. *Mathematical systems theory* **volume 10**, pages99–127 (1976)
48. M. L. Fredman, J. Komlos, and E. Szemeredi. Storing a sparse table with O(1) worst case access time. J. ACM 31(3): 538-544 (1984).
49. P. van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. Information Processing Letters. Volume 6, Issue 3, June 1977, Pages 80-82.
50. R. J. lipton and R. E. Tarjan. Applications of planar separator theorem.SIAM J. Compt., Vol., N3, 1980, P. 615-627.
51. M. T. Goodrish. Planar separators and parallel polygon triangulation. Journal of Computer and System Sciences. Volume 51, Issue 3, December 1995, Pages 374-389.
52. J. L. Bentley. Multidimensional binary search trees used for associative searching/ Comunication of ACM , v. 18, N 9, Student Award , Stanford University, 1975, P. 509-517. (http://cgi.di.uoa.gr/~ad/MDE515/p509-bentley.pdf);
53. F. P. Preparata M. I. Shamos. Computational Geometry: An Introduction. Springer-Verlag, New York, 1985.
54. Willard, D. E. (1978), Predicate-Oriented Database Search Algorithms, Ph.D. thesis, Harvard University, Cambridge, MA, Aiken Computational Laboratory, PhD Tesis, Pg. 20-78, 1978.

55. 2. G. S. Leuker. A Data Structure for Orthogonal Range Queries. Proceedigs of the 19th Annual IEEE Symposium of Fundations of Computer Science,28-34 pg, 1978.
56. B. M. Chazelle and H. Edelsbrunner, Optimal solutions for a class of point retrieval problems, J. Symbol. Comput. 1,47-56 (1985).
57. B. M. Chazelle, R. Cole, F. P. Preparata, and C. K. Yap, New upper bounds for neighbor searching, lriformation and Control, 68(1-3), 105-124 (1986).
58. J.L. Bentley. Multidimensional divide-and-conquer. *Commun. ACM,* 23:214-229, 1980.
59. Brown R. Building a balanced k-d tree in $O(kn \log n)$ time // Journal of Computer Graphics Techniques. 4 (1): 50–68.
60. Терещенко В.М., Михащук М. Модифікація методу регіонального пошуку для регіонів криволінійними границями // International Conference “ Problems of decision making under uncertainties ” , PDMU-2006, Alushta, 180-183 с.
61. M.T. Goodrich and R. Tamassia. Dynamic trees and dynamic point location. In Proc. 23rd Annu. ACM Sympos. Theory Comput., pages 523–533, 1991.
62. Терещенко В.М., Касьянов А.А. Один підхід до розв’язання задач регіонального пошуку // Вісник київського університету, КУ, Київ, вип. № 3, 2006, ст. 265-269.
63. B. Chazelle. Cutting hyperplanes for divide-and-conquer. Discrete Comput. Geom., 9:145–158, 1993.
64. J. Matouˇsek. Range searching with efficient hierarchical cuttings. Discrete Comput. Geom., 10:157–182, 1993.
65. B. Chazelle, M. Sharir, and E. Welzl. Quasi-optimal upper bounds for simplex range searching and new zone theorems. Algorithmica, 8:407–429, 1992.
66. J. Matouˇsek. Efficient partition trees. Discrete Comput. Geom., 8:315–334, 1992.
67. J. Matouˇsek. Reporting points in halfspaces. Comput. Geom., 2:169–186, 1992.
68. D.E. Willard. Polygon retrieval. SIAM J. Comput., 11:149–165, 1982.
69. D. Haussler and E. Welzl. Epsilon-nets and simplex range queries. Discrete Comput. Geom., 2:127–151, 1987.
70. T.M. Chan. Optimal partition trees. Discrete Comput. Geom., 47:661–690, 2012.
71. P.K. Agarwal and J. Matouˇsek. On range searching with semialgebraic sets. Discrete Comput. Geom., 11:393–418, 1994.
72. M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. Computational Geometry: Algorithms and Applications. Springer, Berlin, 3rd edition, 2008
73. H. Edelsbrunner and E. Welzl, Halfplanar range search in linear space and O(n** 0.695) query time, Info. Proc. Lett., 23, 289-293 (1986).
74. P.K. Agarwal and J. Erickson. Geometric range searching and its relatives. In B. Chazelle, J. E. Goodman, and R. Pollack, editors, Advances in Discrete and Computational Geometry, vol. 223 of Contemporary Mathematics, pages 1–56, AMS, Providence, 1999.
75. J. Matousek. 1994. Geometric range searching. Computing Surveys 26, 4 (1994), 421--461. DOI:http://dx.doi.org/10.1145/197405.197408.
76. Chazelle, B. The Discrepancy Method, book in preparation, 1998.
77. T. Kanda, K. Sugihara. Two-dimensional range search based on the Voronoi diagram. Computational Science and Its Applications — ICCSA 2003, Conference paper. - pp 776–786.
78. J. L. Bentley and M. I. Shamos, A problem in multivariate statistics: Algorithms, data structure, and applications, Proceedings of the i5th Annual Allerton Conference on Communication, Control. and Computing, pp. 193-201 (1977).
79. О.Ю. Грищенко, С.І. Ляшко. Теорія функцій комплексної змінної.ВПЦ «Київський університет». – 2009.-495 с.
80. Тітяпкин А.С., Зайченко М.Д., Тітяпкин С.С. Використання інтегральної формули Коші в задачі локалізації точки в довільному полігоні\ XXI міжнародна науково-практична конференція МАТЕМАТИЧНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ІНТЕЛЕКТУАЛЬНИХ СИСТЕМ (МПЗІС-2023), 22-24 листопада, 2023, Дніпро. – 281-282.

ДОДАТКИ