КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра математичної інформатики

Кваліфікаційна робота
на здобуття ступеня бакалавра
за освітньо-професійною програмою "Інформатика"
спеціальності 122 Комп'ютерні науки на тему:

РЕАЛІЗАЦІЯ АЛГОРИТМУ «РОЗПОДІЛЯЙ ТА ВОЛОДАРЮЙ» ДЛЯ ПОБУДОВИ ДІАГРАМИ ВОРОНОГО

Виконав студент 4-го курсу
Євгеній ГОЛОВЕНЬ _____
(підпис)

Науковий керівник:
Завідувач кафедри Математичної Інформатики, професор
Василь ТЕРЕЩЕНКО _____
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри математичної інформатики

Протокол № _____ _____ 2023 р.

Завідувач кафедри
В. М. Терещенко _____
(підпис)

Київ – 2023

РЕФЕРАТ

Обсяг роботи: 43 сторінок, 8 ілюстрацій, 0 таблиць, 6 використаних джерел.

Ключові слова: ОБЧИСЛЮВАЛЬНА ГЕОМЕТРІЯ ТА КОМП'ЮТЕРНА ГРАФІКА, ДІАГРАМА ВОРОНОГО, АЛГОРИТМ РОЗПОДІЛЯЙ ТА ПАНУЙ.

Об'єктом роботи є реалізація алгоритму розподіляй та пануй для побудови діаграми Вороного з використанням мови програмування Python для бібліотеки CGLib.

Метою кваліфікаційної роботи є створення й публікація алгоритму розподіляй та пануй для побудови діаграми Вороного для використання в системі автоматичного оцінювання модульних контрольних робіт з курсу «Обчислювальна Геометрія та Комп'ютерна Графіка».

Інструментами створення є безкоштовний, вільно поширюваний редактор коду Visual Studio Code та мова розмітки тексту Typst, мови програмування Python. Використано бібліотеку tkinter та numpy.

Результат роботи: розроблено та опубліковано реалізацію алгоритму під ліцензією Apache License.

ЗМІСТ

ВСТУП 5

РОЗДІЛ 1. ОГЛЯД СИСТЕМИ З АВТОМАТИЧНОГО ОЦІНЮВАННЯ 7

1.1. Короткий огляд системи з автоматичного оцінювання 7

1.2. Алгоритми 7

1.3. Існуючі реалізації 8

РОЗДІЛ 2. ПОСТАНОВКА ЗАДАЧІ 10

2.1. Умова задачі 10

2.2. Критерії оцінювання 11

РОЗДІЛ 3. ДІАГРАМА ВОРОНОГО 13

3.1. Короткий огляд 13

3.2. Огляд існуючих алгоритмів 14

РОЗДІЛ 4. ОГЛЯД ІСНУЮЧИХ РЕАЛІЗАЦІЙ ПОБУДОВИ ДІАГРАМИ ВОРОНОГО 17

4.1. Реалізація в бібліотеках 17

4.2. Реалізація в публічних репозиторіях 18

4.3. Висновки з існуючих реалізацій 19

РОЗДІЛ 5. ТЕОРЕТИЧНИЙ ОПИС АЛГОРИТМУ 20

5.1. Опис алгоритму розподіляй та пануй 20

5.2. Опис побудови роздділяючого ланцюга 21

5.3. Опис відсікання зайвих променів 23

5.4. Опис структури даних 23

5.5. Найпростіші випадки 25

РОЗДІЛ 6. ОСНОВНІ СУТНОСТІ ДЛЯ РЕАЛІЗАЦІЇ АЛГОРИТМУ РОЗПОДІЛЯЙ ТА ПАНУЙ 27
6.1. Point 27
6.2. Site 27
6.3. Edge 28
6.4. VoronoiDiagram 29
6.5. Використання сутностей 30
РОЗДІЛ 7. РЕАЛІЗАЦІЯ АЛГОРИТМУ 32
7.1. Реалізація побудови множин U_x та U_y 32
7.2. Реалізація перевірки дерева розбиття 32
7.3. Реалізація побудови розділяючого ланцюга 34
РОЗДІЛ 8. Додатковий функціонал 36
РОЗДІЛ 9. МОЖЛИВІ МАЙБУТНІ ПОКРАЩЕННЯ 38
9.1. Оптимізація алгоритму 38
9.2. Узагальнення початкової задачі 39
9.3. Узагальнення алгоритму 40
ВИСНОВКИ 42
ПЕРЕЛІК ДЖЕРЕЛ 43

ВСТУП

Сучасний стан об’єкта дослідження: Діаграми Вороного є важливим інструментом в обчислювальній геометрії, який знаходить широке застосування в різних областях, включаючи комп’ютерну графіку. Проте, побудова цих діаграм в ефективний спосіб є складним завданням, що вимагає використання продуманих алгоритмів.

Актуальність роботи та підстави для її виконання: Використання алгоритму «Розподіляй і пануй» для побудови Діаграми Вороного є актуальним завданням, оскільки цей алгоритм дозволяє ефективно розробляти діаграми для великих наборів даних. Альтернативою цьому методу за обчислювальною складністю є алгоритм Форчуна.

Метою даної роботи є розробка алгоритму на Python, який буде використовуватися в системі автоматичного оцінювання модульних контрольних робіт з курсу «Обчислювальна Геометрія та Комп’ютерна Графіка».

Об’єктом дослідження є алгоритм «Розподіляй і пануй» для побудови Діаграми Вороного. Методи дослідження включають аналіз алгоритму, його реалізацію на Python та тестування для системи автоматичного оцінювання.

Результати цієї роботи можуть бути використані в освітніх процесах, зокрема, в курсах обчислювальної геометрії

та комп’ютерної графіки, а також в інших областях, де використовуються Діаграми Вороного.

Ця робота базується на попередніх дослідженнях в області алгоритмів побудови Діаграми Вороного та їх застосування в обчислювальній геометрії та комп’ютерній графіці зокрема divide-and-conquer approach of Shamos and Hoey.

РОЗДІЛ 1. ОГЛЯД СИСТЕМИ З АВТОМАТИЧНОГО ОЦІНЮВАННЯ

1.1. Короткий огляд системи з автоматичного оцінювання

Для успішного складання курсу з «Обчислювальної геометрії та комп’ютерної графіки» треба вміти застосовувати методи розв’язання задач. Формою оцінювання таких робіт є модульні контрольні роботи. Для спрощення перевірки таких завдань була запропонована система з автоматичного оцінювання модульних контрольних робіт.

Задачі, що виносяться на модульну контрольну роботу, передбачають алгоритмічне розв’язання, тому критерії оцінювання спрямовані на перевірку послідовного виконання алгоритму. У цьому контексті система, здатна самостійно розв’язувати такі задачі та формувати відповіді, може значно спростити та прискорити процес перевірки студентських робіт. Крім того, використання такої системи надає можливість створювати унікальні варіанти контрольних робіт у короткі терміни, оскільки будь-які зміни у вхідних даних призводять до відповідних змін у результатах.

1.2. Алгоритми

Одним з структурних компонентів є власне реалізація агоритмів, які могли б віддавати проміжні результати для перевірки правильності виконання.

Алгоритми, які мають бути реалізовані, для того, щоб система могла бути застосована в освітньому процесі:

- Задачі геометричного пошуку:
    - Метод смуг
    - Метод ланцюгів
    - Метод Д-Т
    - Метод kd-дерева
    - Метод дерева регіонів
- Побудова опуклої оболонки:
    - Метод Грехема
    - Метод Швидкобол
    - Метод Препарата
    - Метод підтримки динамічної опуклої оболонки
- Задачі близькості:
    - Метод «Розділяй та Пануй» пошуку найближчої пари
    - Метод «Розділяй та Пануй» побудови Діаграми Вороного

1.3. Існуючі реалізації

Описана система ще не використовується в освітньому процесі, проте є прототип, зроблений Фісуненко Артемом та Германюком Всеволодом[1]. Для того, щоб дана система мала змогу працювати потрібно реалізувати всі алгоритми, які було наведено. Один з яких є Метод «Розділяй та Пануй» побудови Діаграми Вороного

Дана робота прсвячена аналізу існуючих реалізацій побудови Діаграм Вороного та розробці алгоритму на основі стратегії «Розділяй

та володарюй» адаптованого в алгоритмічне середовище системи автоматичного оцінювання.

РОЗДІЛ 2. ПОСТАНОВКА ЗАДАЧІ

2.1. Умова задачі

На площині задано N точок, побудувати діаграму Вороного методом розподіляй та пануй.

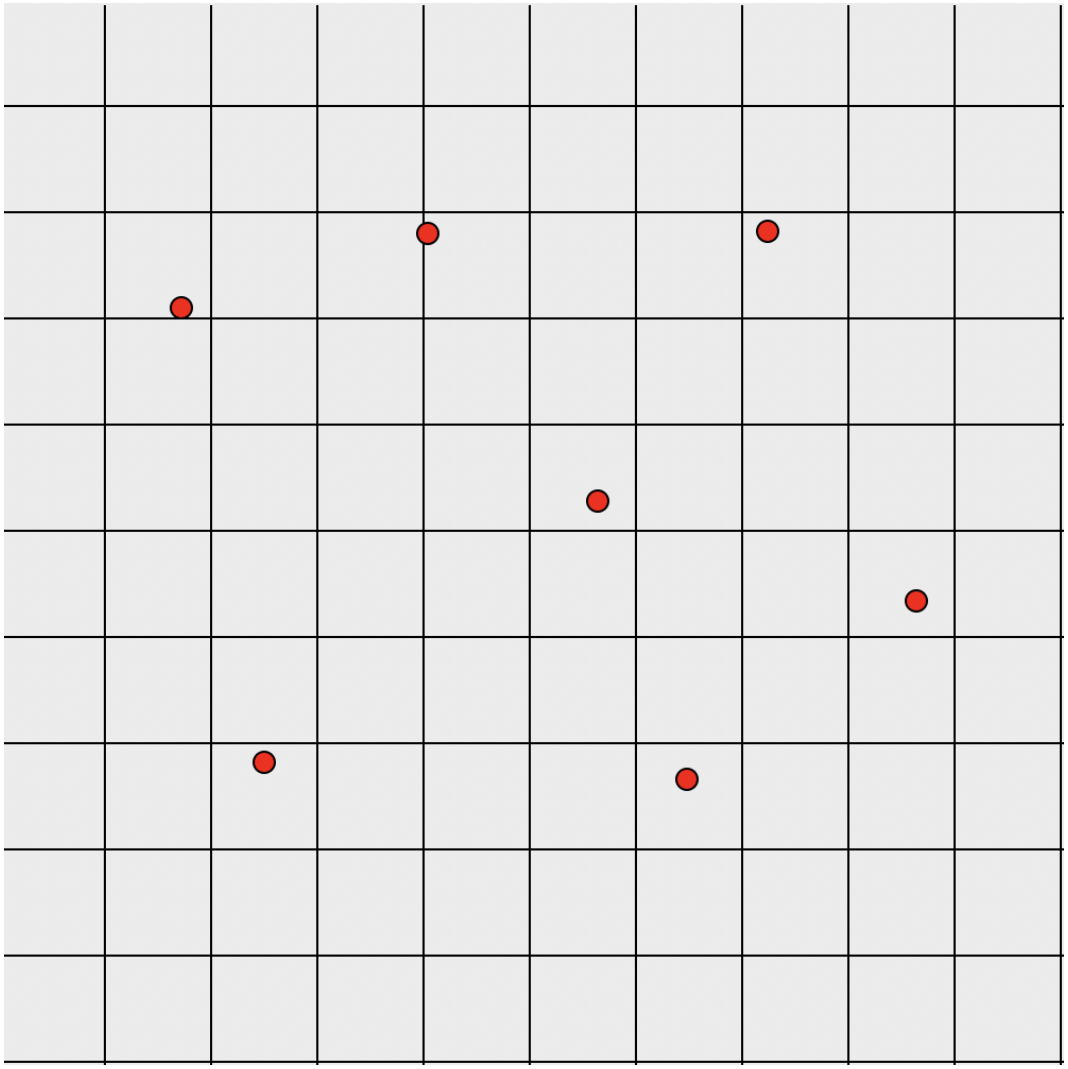

Рисунок 1 – 7 точок на площині

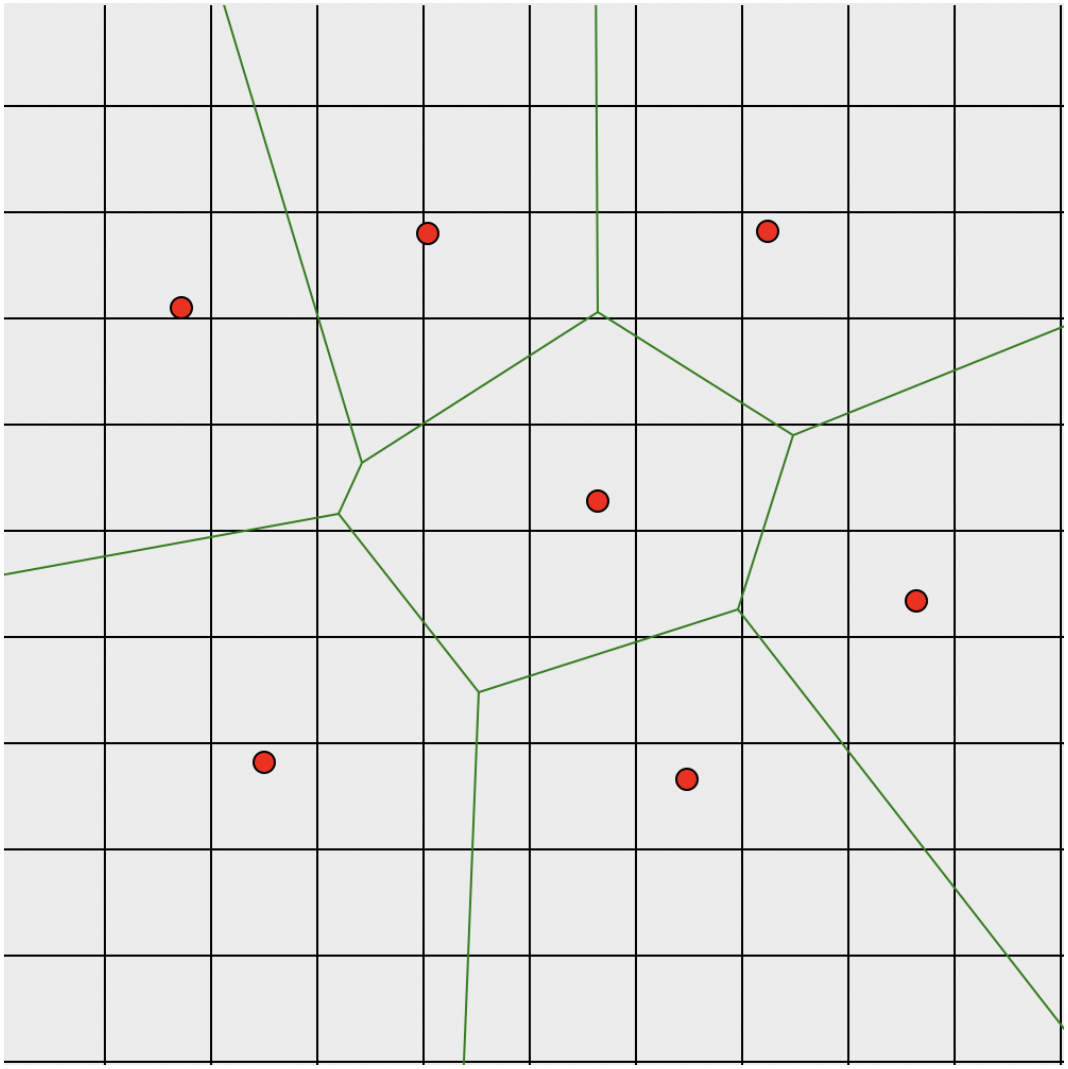

Рисунок 2 – Діаграма Вороного для 7 точок

2.2. Критерії оцінювання

Для тих, хто виконує завдання критерії наступні:

- Побудувати відсортовані списки за x та y координатах (U_x, U_y)
- Рекурсивний спуск. Рекурсивно розбити множину S (у вигляді U_x) на рівно потужні підмножини (побудувати дерево алгоритму). Прописати вузли дерева одержаними списками.
- Рекурсивний підйом. На кожному кроці рекурсії починаючи із листків дерева алгоритму побудувати (вказати на дереві

алгоритму) Діаграму Вороного як злиття діаграм для лівої і правої підмножин точок $vor(S_l)$ і $vor(S_r)$.

Окремо варто прописати критерії для рекурсивного підйому:

- Знайти верхнє і нижнє опорне ребро
- Будуємо вхідне розділяюче ребро σ_{in} та вихідне розділяюче ребро σ_{out}
- Будуємо розділяючий ланцюг σ одержуючи Діаграми Вороного.
- Подаємо діаграми Вороного у батьківський вузол для злиття. Рекурсивно одержуємо Діаграму Вороного для усієї множини. У вузлах вказуємо одержані Діаграми Вороного

РОЗДІЛ 3. ДІАГРАМА ВОРОНОГО

3.1. Короткий огляд

Діаграма Вороного названа на честь українського математика Георгія Вороного. Це математичний спосіб розділити простір на декілька областей (відомих як відсіки Вороного) на основі відстані до заданого набору точок (відомих як «сіті»). В кожному відсіку Вороного всі точки ближчі до однієї сіті, ніж до будь-якої іншої.

Діаграми Вороного мають багато практичних застосувань у широкому спектрі дисциплін:

- **Комп'ютерна графіка та геометрія**: Діаграми Вороного використовуються для генерації текстур, створення деталізованих моделей, оптимізації відображень та в геометрічній обробці
- **Робототехніка**: Діаграми Вороного можуть використовуватися для планування шляхів для роботів, щоб уникнути зіткнення з перешкодами
- **Географічні інформаційні системи (GIS)**: Діаграми Вороного можуть використовуватися для оцінки доступності ресурсів, аналізу демографічних даних та прогнозування екологічних моделей
- **Астрономія**: В астрономії діаграми Вороного використовуються для визначення густини зірок і вивчення структури Всесвіту

- **Біологія і хімія**: У цих дисциплінах діаграми Вороного можуть використовуватися для моделювання структури складних молекул, аналізу тканин та створення структурних моделей

Існують різні способи побудови діаграми Вороного, зокрема:

- **Алгоритм Брауера**: Це інкрементний алгоритм, який додає точки одну за одною до діаграми, розділяючи відсіки Вороного, коли це необхідно. Кожен новий вузол розбиває одну з областей на дві, відокремлюючи ближчі точки від недавно доданої точки
- **Алгоритм Форчуна[2]**: Цей алгоритм використовує техніку, відому як «метод свіп-лінії». Він включає створення віртуальної «лінії обходу», яка «вказує» через простір, і робить оновлення, коли ця лінія перетинає точку вхідних даних. Цей алгоритм є ефективним та широко використовується
- **Алгоритми розподіляй та пануй[3]**: Цей алгоритм розділяє вхідні дані на менші частини, обчислюють діаграми Вороного для кожної частини, а потім об’єднують їх разом
- **Алгоритми QuickHull[4]** та **Gift Wrapping**: Ці алгоритми, які часто використовуються для обчислення опуклої оболонки, також можуть бути адаптовані для побудови діаграм Вороного

3.2. Огляд існуючих алгоритмів

Алгоритм Брауера:

- Це простий і прямолінійний алгоритм, легко зрозуміти і реалізувати
- Підходить для невеликих наборів даних, де швидкість виконання не є критичною
- Він має квадратичну складність $O(n^2)$, тому не підходить для великих наборів даних
- Для кожного нового вузла потрібно перебудовувати більшу частину діаграми

Алгоритм Форчуна:

- Це один з найшвидших відомих алгоритмів для побудови діаграм Вороного, зі складністю $O(n \log n)$
- Він ефективний для великих наборів даних
- Він складний для розуміння та реалізації, особливо коли використовується для більш ніж двомірних діаграм
- Алгоритм вимагає використання балансованої структури даних для події та пляжної лінії, що може збільшити його складність реалізації

Алгоритм розподіляй та пануй:

- Алгоритм ефективний та швидкий, особливо для великих наборів даних, складність $O(n \log n)$

- легко паралелізуються, що робить його підходящими для використання на багатопроцесорних або розподілених системах
- Розбиття даних та злиття результатів може бути складним процесом, особливо для складних просторових даних
- Деякі реалізації можуть мати проблеми з чисельною стабільністю

Алгоритми QuickHull та Gift Wrapping:

- Вони прості для розуміння та реалізації
- Можуть бути використані для вирішення пов’язаних задач, таких як побудова опуклої оболонки
- Вони мають погану часову продуктивність у найгіршому випадку, яка може сягати до $O(n^2)$
- Вони не є ефективними для побудови діаграм Вороного, порівняно з іншими спеціалізованими алгоритмами

РОЗДІЛ 4. ОГЛЯД ІСНУЮЧИХ РЕАЛІЗАЦІЙ ПОБУДОВИ ДІАГРАМИ ВОРОНОГО

4.1. Реалізація в бібліотеках

Було розглянуто наступні бібліотеки, в яких реалізован алгоритм побудови діаграми Вороного: SciPy, scikit-image, OpenCV, PyVoro.

Короткий огляд реалізацій побудови діаграми Вороного:

- **SciPy**: SciPy використовує алгоритм Qhull для побудови діаграм Вороного. Qhull - це програмне забезпечення з відкритим вихідним кодом, яке використовує алгоритм `"quickhull"` для побудови діаграм Вороного, оболонок конвексу, триангуляцій Делоне та інших політопів.
- **scikit-image**: Scikit-image використовує алгоритм Fortune для побудови діаграм Вороного в 2D. Алгоритм Fortune - це ефективний алгоритм побудови діаграм Вороного, який працює за час $O(n \log n)$, де n - кількість точок.
- **OpenCV**: OpenCV використовує алгоритм Subdiv2D для побудови діаграм Вороного. Subdiv2D - це клас для побудови двовимірної триангуляції Делоне та відповідної діаграми Вороного.
- **PyVoro**: PyVoro є Python обгорткою над бібліотекою Voro++, яка використовує алгоритм заснований на методі обробки клітин для побудови діаграм Вороного в 2D і 3D.

Можна побачити, що немає існуючої реалізації алгоритму розподіляй та пануй, яка б могла стати основою для системи перевірки з заданими критеріями.

4.2. Реалізація в публічних репозиторіях

При аналізі публічних репозиторіїв GitHub мовою програмування Python було виявлено одну реалізацію алгоритму[5]. Опишемо цю реалізацію:

Надані точки сортуються за x координатою, потім передаються в фукцію voronoiLinesUtil, яка робить наступний алгоритм: наступним чином:

```
voronoiLinesUtil points
begin
  if length(points) == 1
  begin
    return vd(points)
  end
  if length(points) == 2
  begin
    return vd(points)
  end
  median := length(points)/2
  left := points[0..median]
  right := points[median..length(points)]
  left_vd := voronoiLinesUtil(left)
  right_vd := voronoiLinesUtil(right)
  VD := mergeVD(left_vd, right_vd)
  return VD
end
```

Можна побачити, що в даному алгоритмі не існує структури даних, якої потребують наші критерії, та алгоритм розбиває на найменші діаграми для однієї та двох точок.

4.3. Висновки з існуючих реалізацій

Аналізуючи існуючи реалізації зроблено наступні висновки:

- Діаграма Вороного будується за допомогою алгоритмів відмінних від розподіляй та пануй
- Відсутність структур даних, що задовольняла б критерії виконання задачі

РОЗДІЛ 5. ТЕОРЕТИЧНИЙ ОПИС АЛГОРИТМУ

5.1. Опис алгоритму розподіляй та пануй

Детальний опис алгоритму:

1. Спочатку на множині S із N точок побудуємо відсортовані списки точок по x координаті U_x і по y координаті U_{y}. Це робиться для того, щоб легко і швидко знайти точки за їх координатами
2. На основі списків U_x і U_y формуємо масив точок U, де i – індекс указує номер точки у впорядкованому списку U_x, а j – індекс указує номер точки у впорядкованому списку U_y
3. Розбиття множини точок на рівнопотужні підмножини. Для цього використовується рекурсивний спуск, де на кожному кроці задана множина S у вигляді списку U розбивається на дві підмножини вертикальною лінією, яка проходить через медіану множини точок
4. Злиття підмножин та побудова діаграми Вороного. Для кожної підмножини рекурсивно будуються діаграми Вороного, а потім злиті діаграми об’єднуються з допомогою розділяючого ланцюга
    - Припускаємо, що для підмножин точок S_1 і S_2 вже побудовані діаграми Вороного
    - Знаходимо верхній та нижній опорні відрізки для точок діаграм $Vor(S_1)$ та $Vor(S_2)$

- Побудуємо вхідний та вихідний промені розділяючого ланцюга, які є перпендикулярами до вхідного та вихідного опорних відрізків
- Рухаємось по вхідному променю до того моменту, коли перетнемо ребро однієї із діаграм $Vor(S_1)$ або $Vor(S_2)$
- Залежно від того, ребро якої діаграми перетнеться, із відповідної множини змінюється точка в наступній парі до якої із точки перетину будується серединний перпендикуляр
- Побудова ребер ланцюга продовжується до тих пір, доки не буде досягнуто вихідного променя
- Після побудови розділяючого ланцюга відсікаються зайві промені

5.2. Опис побудови розділяючого ланцюга

Побудова розділяючого ланцюга - важлива частина алгоритму побудови діаграми Вороного. Цей процес включає створення ряду ліній (або «ланцюга»), які розділяють дві підмножини точок на діаграмі Вороного.

Опис побудови розділяючого ланцюга:

1. Позначимо початкову точку лівої підмножини як p_L і початкову точку правої підмножини як p_R. Введимо

тимчасові змінні e (для поточного ребра ланцюга) і v (для поточної вершини)

2. Визначимо перше ребро границі діаграми Вороного для точок p_L і p_R, називаємо їх e_L і e_R відповідно. Ці ребра будуть початковими точками для нашого розділяючого ланцюга
3. Перебираємо ребра границі діаграми Вороного для обох підмножин, поки не знайдемо ребро, яке перетинається з поточним ребром e
4. Якщо точка перетину з ребром e_L ближча до v, ніж точка перетину з ребром e_R, то міняємо p_L на сусідню точку з підмножини S_1 і переходимо до наступного ребра на границі діаграми Вороного для цієї нової точки
5. Якщо ж точка перетину з ребром e_R ближча до v, то міняємо p_R на сусідню точку з підмножини S_2 і переходимо до наступного ребра на границі діаграми Вороного для цієї нової точки
6. У кожному випадку, після зміни точки p_L або p_R, ми змінюємо ребро e на серединний перпендикуляр до відрізка, що сполучає p_L і p_R.
7. Повторюємо цей процес, поки відрізок $[p_L, p_R]$ не стане рівним кінцевому опорному відрізку.

Цей алгоритм гарантує, що розділяючий ланцюг правильно поділить діаграму Вороного на дві підмножини, що відповідають двом підмножинам вихідного множини точок.

5.3. Опис відсікання зайвих променів

Відсікання зайвих променів - це останній етап побудови розділяючого ланцюга в алгоритмі побудови діаграми Вороного. Цей процес включає видалення ребер, які були побудовані під час рекурсивного розбиття, але які більше не є необхідними після об’єднання підмножин.

Після побудови розділяючого ланцюга, деякі з ребер, які були побудовані під час процесу, можуть виявитися зайвими. Це може статися, наприклад, якщо ребро повністю лежить в підмножині, якій воно не належить. В таких випадках, ці ребра не є частиною остаточної діаграми Вороного, і їх можна видалити.

Алгоритм відсікання зайвих променів такий:

1. Перебираємо всі ребра в розділяючому ланцюзі
2. Якщо ребро повністю лежить в межах іншої підмножини, видаляємо це ребро.

Цей процес допомагає оптимізувати остаточну діаграму Вороного, видаляючи ребра, які не вносять вкладу в розділення множини точок.

5.4. Опис структури даних

Для побудови такого алгоритму буде зручно використовувати бінарне дерево, як дерево розбиття.

Опис використання такого дерева в даному алгоритмі:

- Побудова дерева: Починається зі створення бінарного дерева, де кожний вузол представляє підмножину точок. Кореневий вузол представляє всю множину точок, а листя вузлів представляють найпростіші випадки. Якщо у вузла є двоє дітей, це означає, що його множина точок була розбита на дві підмножини
- Розбиття множини точок: На кожному кроці алгоритму розподіляй та пануй поточну множину точок розбивається на дві підмножини, вибираючи вертикальну лінію, яка проходить через медіану точок за x-координатою. Це розбиття відображається у бінарному дереві: поточний вузол генерує два дочірні вузли, які представляють дві підмножини
- Рекурсивний спуск і підйом: Після того, як сформовані листя вузлів, починається підйом назад вгору дерева, виконуючи етап «пануй». На цьому етапі об’єднуються діаграми Вороного для дочірніх вузлів, створюючи діаграму Вороного для батьківського вузла

Для даної задачі за найпростіші випадки побудови діаграми Вороного було обрано випадок двох та трьох точок на площині.

5.5. Найпростіші випадки

Як було зазначено за найпростіші випадки було обрано випадок двох та трьох точок на площині.

Для того щоб побудувати діаграму Вороного для двох точок достатньо знайти серединний перпендикуляр для цих точок, він і буде слугувати ребром в діаграмі Вороного.

Для того щоб побудувати діаграму Вороного для трьох точок ми знаходимо точку перетину бісекторів для цих трьох точок. Обираємо частини бісекторів, для яких виконується наступна умова: $\forall p : dist(p, C) \geq dist(p, A)$, де $dist$ – відстань між двома точками, p – точка на бісекторі, A – одна з двох точок, для яких проведен бісектор, C – третя точка.

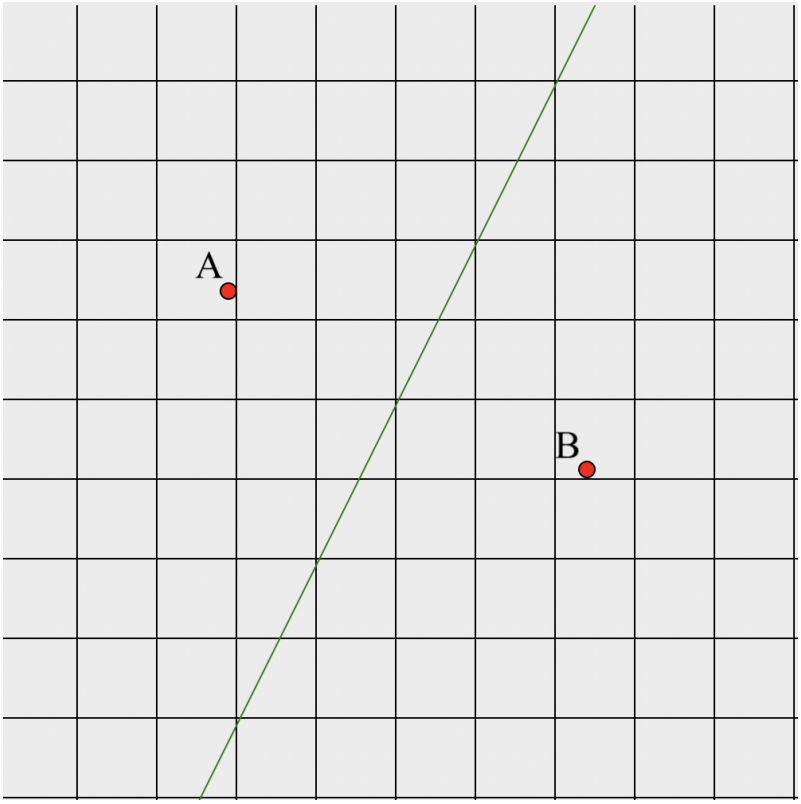

Рисунок 3 – Випадок двох точок

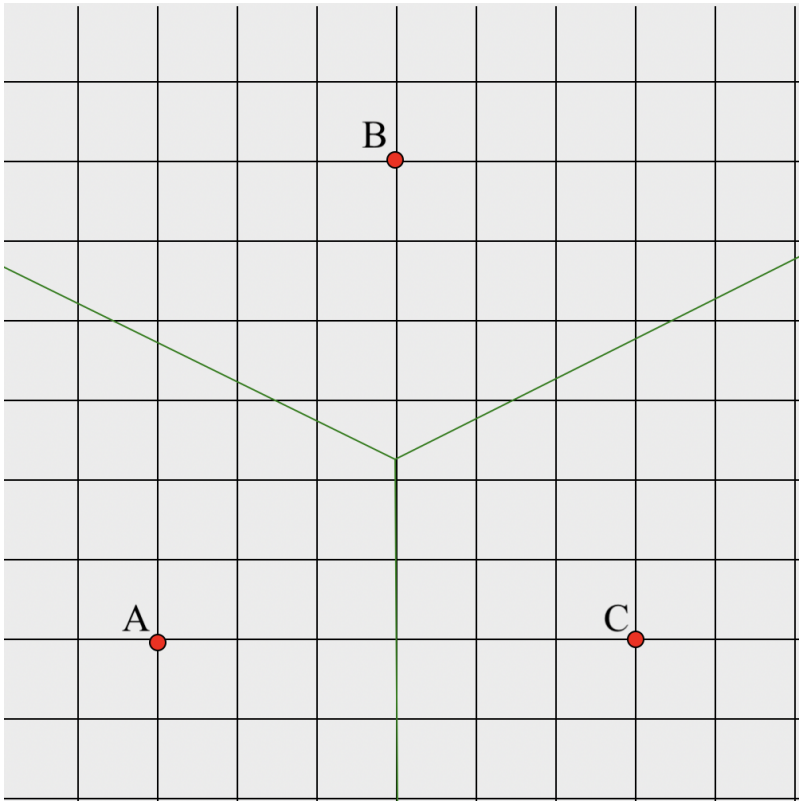

Рисунок 4 – Випадок трьох точок

РОЗДІЛ 6. ОСНОВНІ СУТНОСТІ ДЛЯ РЕАЛІЗАЦІЇ АЛГОРИТМУ РОЗПОДІЛЯЙ ТА ПАНУЙ

6.1. Point

Для реалізації треба працювати з точками на площині, тому було прийнято рішення скористатися бібліотекою CGLib[6] і взяти з неї як сутність Point, в нашому випадку вона виглядає так:

```
Point:
  x: float
  y: float
```

Де x, y – координати точки на площині.

6.2. Site

Site – це сутність, доданий до бібліотеки CGLib, під час розробки власної реалізації алгоритму розподіляй та пануй, яка виглядає наступним чином:

```
Site:
  name: str
  point: Point
  edges: list[Edge]
```

name – ім'я точки, яка задана нам (наприклад A, B, C).

point – об'єкт класу Point, з CGLib і відповідає за розташування, точки, для якої ми будемо обраховувати в подальшому ребра.

edges – об’єкт класу Edge, який буде розглянуто пізніше, який є списком ребер діаграми Вороного, для даної точки.

6.3. Edge

Edge – це сутність, додана до бібліотеки CGLib, під час розробки власної реалізації алгоритму розподіляй та пануй, яка виглядає наступним чином:

```
Edge:
  left_site: Site
  right_site: Site
  start_point: Point
  end_point: Point
  infinity: float
```

left_site, right_site – це об’єкти класу Site, між якими проведено ребро.

start_point – це об’єкт класу Point, який відповідає за стартову точку для ребра.

end_point – це об’єкт класу Point, який відповідає за кінцеву точку для ребра, для значень infinity 0.5 та 1 приймає вигляд $end\_point.x = start\_point.x + dx, \quad end\_point.y = start\_point.y + dy,$ де dx, dy – це координати нормалізованого вектора напрямку ребра, для значення infinity 0 end_point – це координата кінця ребра.

infinity – це спеціальне поле, яке створено для того щоб розуміти, де знаходиться кінець ребра, якщо ребро є прямою, то infinity = 1, якщо

ребро є променем, то infinity = 0.5, якщо ребро є відрізком, то infinity = 0.

6.4. VoronoiDiagram

VoronoiDiagram – це сутність, додана до бібліотеки CGLib, під час розробки власної реалізації алгоритму розподіляй та пануй, яка є бінарним деревом пошуку, для того, щоб відповідати критеріям для розв'язку задачі.

```
VoronoiDiagram:
  name: str
  Ux: list[Site]
  Uy: list[Site]
  left: VoronoiDiagram або None
  right: VoronoiDiagram або None
  edges: list[Edge]

  compute_diagram()
```

name – ім'я конкретного вузла дерева (для того, щоб його сформувати використовуються позначення точок з U_x).

Ux – це список з об'єктів класу Site, відсортований за зростанням x.

Uy – це список з об'єктів класу Site, відсортований за зростанням y.

left, right – це лівий та правий вузоли дерева, до виконання compute_diagram і якщо вузол є листком має значення $None$.

edges – це список ребер діаграми вороного для даного вузла.

compute_diagram – функція, яка обчислює діаграму вороного для даного вузла.

6.5. Використання сутностей

Наведемо приклад для 3х точок з використанням сутностей

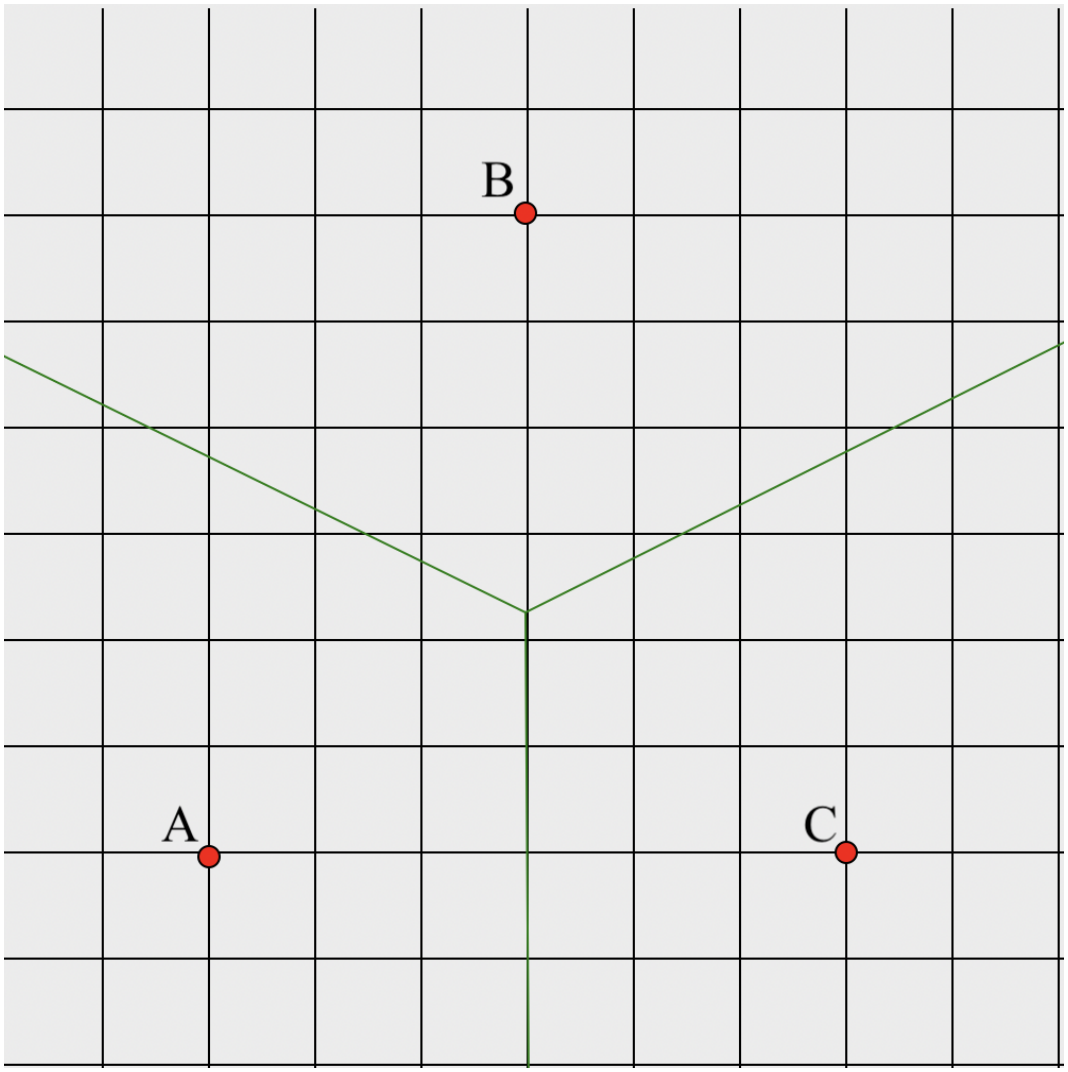

Рисунок 5 – Візуалізація для трьох точок

Для даного випадку будуть наступні сутності:

```
Sites:
  A(name: "A", point: Point(x: 2, y: 2), edges: [AB, AC])
  B(name: "B", point: Point(x: 5, y: 8), edges: [AB, BC])
  C(name: "C", point: Point(x: 8, y: 2), edges: [AC, BC])
```

```
Edges:
  AB(
    left_site: A,
    right_site: B,
    start_point: Point(x: 5, y: 4.25),
    end_point: Point(x: 4.106, y: 4.697),
    infinity: 0.5
  )
  AC(
    left_site: A,
    right_site: C,
    start_point: Point(x: 5, y: 4.25),
    end_point: Point(x: 5, y: 3.25),
    infinity: 0.5
  )
  BC(
    left_site: B,
    right_site: C,
    start_point: Point(x: 5, y: 4.25),
    end_point: Point(x: 5.894, y: 4.697),
    infinity: 0.5
  )

VoronoiDiagrams:
ABC(
  name: "A,B,C",
  Ux: [A, B, C],
  left: None,
  right: None,
  edges: [AB, AC, BC]
)
```

РОЗДІЛ 7. РЕАЛІЗАЦІЯ АЛГОРИТМУ

7.1. Реалізація побудови множин U_x та U_y

Щоб відповідати критеріям задачі треба побудувати відсортовані за x та y множини точок U_x та U_y. Для цього використаємо бібліотеку numpy, а саме функцію lexsort.

Отриманий результат буде відповіддю, щоб перевірити, користувач має ввести правильно рядок, який виглядає як перелік позначень точок через кому.

7.2. Реалізація перевірки дерева розбиття

Клас VoronoiDiagram реалізує функціонал дерева розбиття завдяки своїй структурі та функції comppute_diagram(), вигляд якої:

```
def compute_diagram(self):
  if len(Ux) == 2:
    self.edges += compute_two_points(Ux)
  elif len(Ux) == 3:
    self.edges += compute_three_points(Ux)
  else:
    median = len(Ux) // 2
    self.left = VoronoiDiagram(Ux[0:median])
    self.right = VoronoiDiagram(Ux[median:])
    self.left.compute_diagram()
    self.right.compute_diagram()
    self.merge()
    self.clean()
```

Для того, щоб наша реалізація відповідала критеріям поставленої задачі, нам потрібно, щоб ми могли перевірити правильність введеного користувачем дерева розбиття. VoronoiDiagram віддає побудоване дерево у вигляді рядка, з яким система може порівняти введені користувачем дані, та підтвердети чи спростувати правильність відповіді користувача.

Розглянемо приклад:

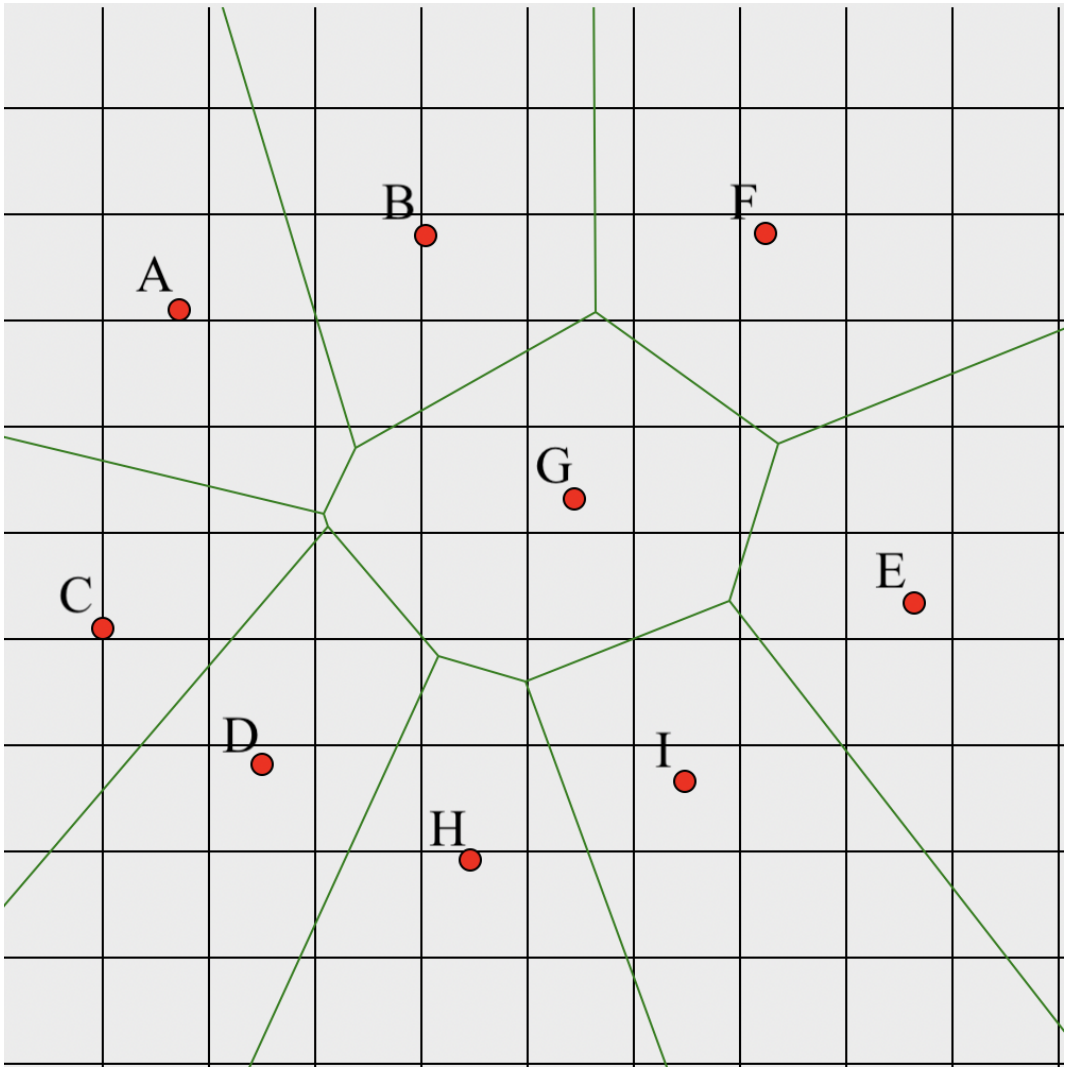

Рисунок 6 – Приклад для 9 точок

Вигляд рядка, який поверне програма:

```
C,A,D,B,H,G,I,F,E:
  C,A,D,B:
    C,A
    D,B
  H,G,I,F,E:
    H,G
    I,F,E
```

7.3. Реалізація побудови розділяючого ланцюга

Побудова розділяючого ланцюга реалізована в класі VoronoiDiagram та відбувається під час використання функції merge().

```
def merge(self):
  upper_tangent = find_upper_tangent(self.left, self.right)
  lower_tangent = find_lower_tangent(self.left, self.right)
  sigma_in = compute_sigma_in(upper_tangent)
  sigma_out = compute_sigma_out(lower_tangent)
  edge = sigma_in
  while edge != sigma_out:
    edge = compute_next_edge(edge, self.left, self.right)
```

Дана функція знаходить верхній та нижній опорний відрізок, у вигляді пар точок, які задаються однозначно: перша літера відрізку – позначення точки з лівого вузла дерева, друга літера – позначення точки з правого вузла дерева. Також знаходить σ_{in} та σ_{out}, а також будує ланцюг σ.

На прикладі останнього кроку для наступної діаграми вороного покажемо, як виглядають опорні відрізки та σ, які має ввести користувач.

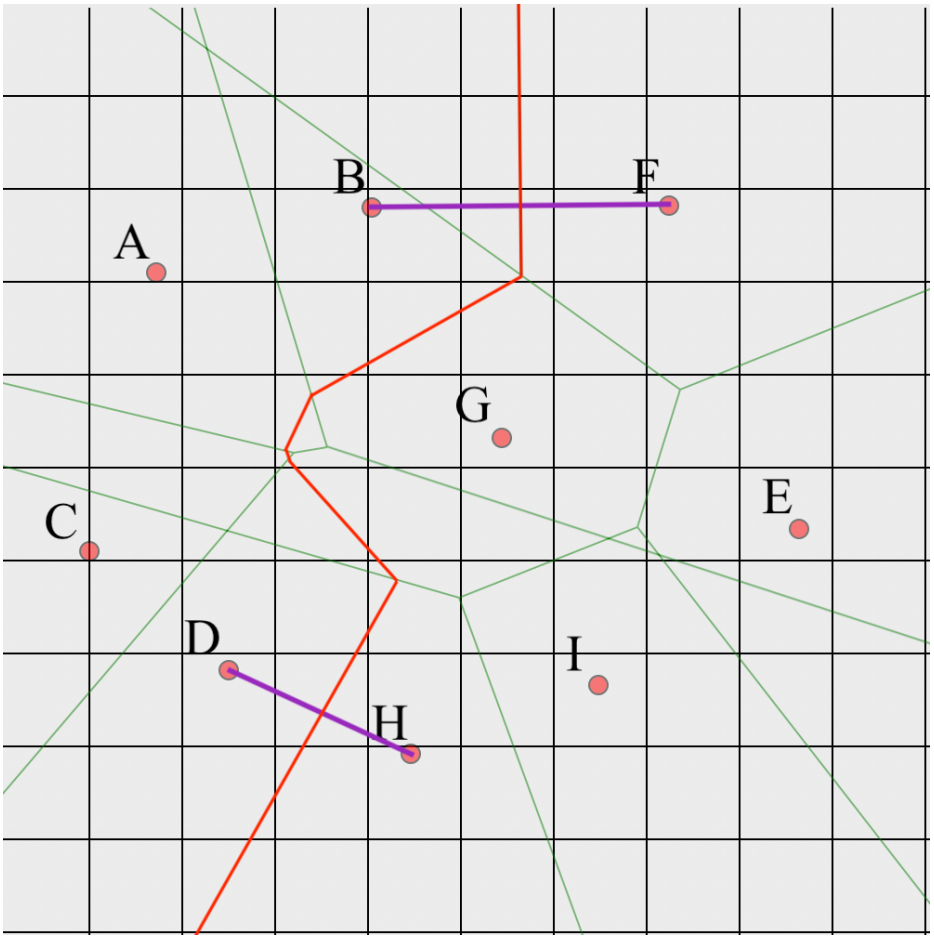

Рисунок 7 – Приклад розділяючого ланцюга для 9 точок

Програма поверне, як верхній опорний відрізок, BF, а, як нижній опорний відрізок, поверне DH.

Розділяючий ланцюг σ буде повернений у вигляді:

```
in
(B,F)
(B,G)
(A,G)
(C,G)
(D,G)
(D,H)
out
```

РОЗДІЛ 8. ДОДАТКОВИЙ ФУНКЦІОНАЛ

При розробці був розроблений невеликий застосунок-помічник, який має наступні можливості:

- Простий і зрозумілий візуальний інтерфейс
- Завантажувати точки для побудови діаграми Вороного
- Зберігати конкретні конфігурації точок, для подальшого вивчення їх
- Додавання, вилучення та переміщення точок на площині

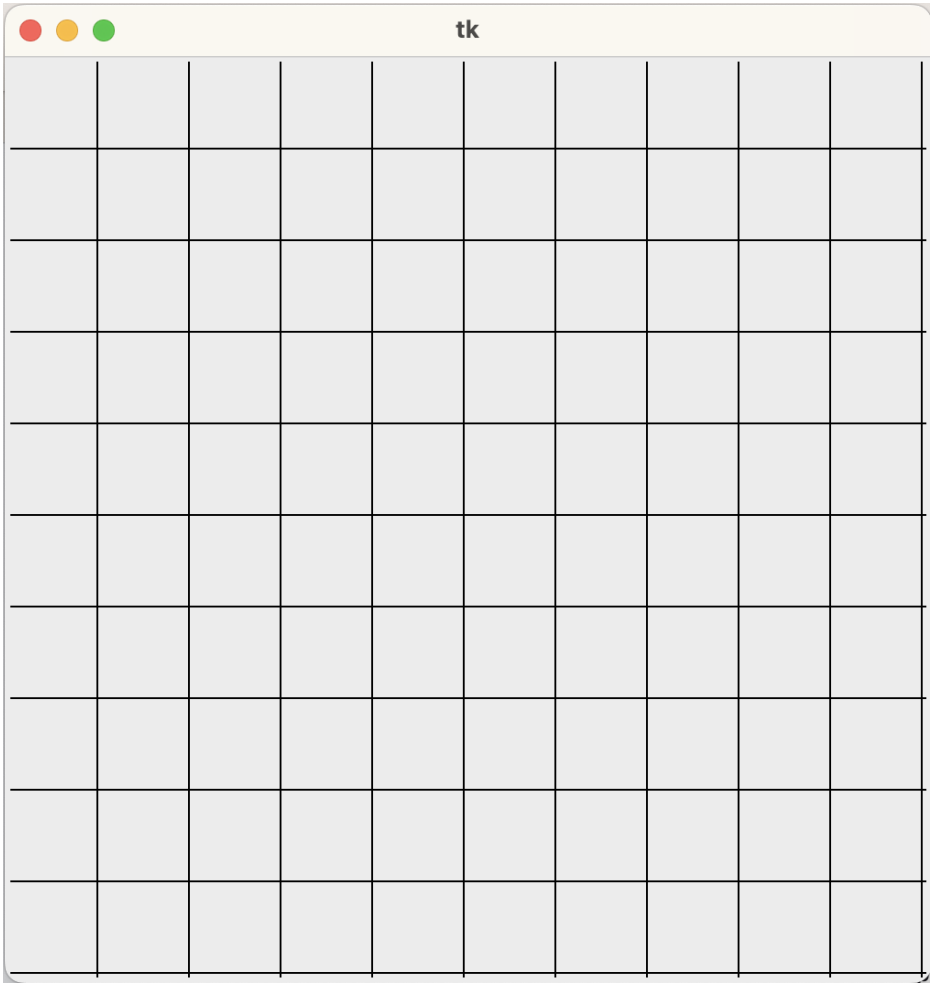

Рисунок 8 – зовнішній вигляд застосунка-помічника

Реалізований застосунок за допомогою бібліотеки tkinter для мови програмування Python. Зовнішній вигляд програми: площина з сіткою 10x10.

Додавання точки: лівий клік миші на площину.

Вилучення точки: подвійний лівий клік миші на точку, яку збираються видалити.

Збереження конфігурації точок: натиснути S на клавіатурі.

Обрахунок та візуалізація діаграми Вороного: натиснути L на клавіатурі.

РОЗДІЛ 9. МОЖЛИВІ МАЙБУТНІ ПОКРАЩЕННЯ

9.1. Оптимізація алгоритму

Стандартний алгоритм «розподіляй та пануй» є вже досить ефективним, але є можливість покращити його шляхом впровадження ряду оптимізацій:

- **Ефективніше використання пам’яті**: В алгоритмах, особливо в тих, що працюють з великими обсягами даних, ефективне використання пам’яті може значно покращити продуктивність. До стратегій оптимізації використання пам’яті може входити обмеження збереження непотрібних проміжних результатів, використання структур даних, які вимагають менше пам’яті, та розробка алгоритма, що мінімізує кількість копіювання даних. Математично, це може вплинути на просторову складність алгоритму, зменшуючи її.
- **Паралелізація**: Паралелізація використовується для одночасного виконання декількох частин алгоритму. Для «розподіляй та пануй» він може включати паралельне виконання рекурсивних викликів на різних частинах даних. Це особливо корисно, коли використовуються многопоточні або многоядерні обчислювальні системи. Математично, паралелізація може зменшити часову складність алгоритму в найкращому випадку до O(log n), припускаючи, що достатньо

ядер для паралельного виконання всіх рекурсивних викликів одночасно.

- **Використання додаткової інформації про дані**: Якщо є додаткова інформація про вхідну структуру, можна використовувати цю інформацію, щоб краще розподілити задачі. Наприклад, якщо відомо, що вхідні дані мають певну структуру або розподіл, то можна краще розбити дані на частини. Математично, це може допомогти зменшити складність алгоритму шляхом уникнення непотрібних розбиттів або операцій.

9.2. Узагальнення початкової задачі

В даній роботі ми розглянули просту задачу для N точок на площині, цю задачу можна узагальнити в наступні способи:

- **Вищі виміри**: Діаграми Вороного можна узагальнити на вищі виміри. У 3D, вони стають поліедрами Вороного, у яких кожна точка є центром поліедра, а всі точки в межах поліедра ближче до центральної точки, ніж до будь-якої іншої. Для вищих вимірів це стає геометрично складнішим і потребує більше обчислювальних ресурсів. Прикладом використання діаграм Вороного в багатовимірному просторі може бути класифікація даних в машинному навчанні.
- **Вагові функції**: Ви можете узагальнити діаграму Вороного, використовуючи вагові функції для точок. Це дозволяє

врахувати «важкість» або «важливість» кожної точки при побудові діаграми. Це може бути корисно, наприклад, у задачах географічного моделювання, де вам може знадобитися врахувати населення міст. Математично, це вимагає зміни ваги кожної точки відносно інших при виконанні побудови діаграми Вороного.

- **Точки, що рухаються**: Деякі застосування можуть вимагати відстежування динамічних діаграм Вороного, де точки змінюють своє положення в часі. Наприклад, це може бути корисно для відстежування руху різних об’єктів, як-то автомобілі в місті. Математично, це вимагає перебудови діаграми Вороного в моменти часу, коли точки змінюють своє положення.

9.3. Узагальнення алгоритму

Алгоритм «розподіляй та пануй» дуже добре підходить для даної задачі, але його можна узагальнити для інших випадків. Приклади узагальнення алгоритму:

- **Ітеративні покращення**: В багатьох випадках діаграму Вороного може бути необхідно постійно оновлювати за мірою додавання нових точок або видалення існуючих. Тут можна використовувати ітеративні алгоритми, які оновлюють діаграму Вороного замість повної перебудови. Щоб описати

цей процес, потрібно розробити методи для вставки нових точок у діаграму Вороного та видалення існуючих точок.

- **Інші стратегії поділу**: Хоча традиційний алгоритм «розподіляй та пануй» зазвичай ділить точки на дві приблизно рівні групи, інші стратегії поділу також можуть бути корисними. Наприклад, можна поділити точки на групи на основі їх геометричної конфігурації, щоб спростити побудову діаграми Вороного. Це може вимагати розробки нових алгоритмів для поділу точок або адаптації існуючих алгоритмів до нової стратегії поділу. Результати можуть сильно варіюватися в залежності від конкретної стратегії поділу та від природи вхідних даних.

ВИСНОВКИ

В даній роботі було висунуто задачу, для якої було розроблено доповнення до вже існуючої бібліотеки CGLib у вигляді алгоритму розділяй та пануй для побудови діаграми Вороного.

Під час виконання роботи було проаналізовано існуючі реалізації алгоритмів для побудови діаграми Вороного та наведені переліки їх сильних та слабких сторін.

Також було розроблено допоміжний застосунок для спрощення робочого процесу.

Додатково було надано декілька можливих напрямків для подальшого вдосконалення та узагальнення розроблених рішень, які можуть стати основою для майбутніх досліджень і розробок.

ПЕРЕЛІК ДЖЕРЕЛ

1. “Cg & cg.” [Електронний ресурс]. Режим доступу до ресурсу: https://github.com/OGKG
2. S. Fortune, “A sweepline algorithm for voronoi diagrams,” Yorktown Heights, New York, USA: Association for Computing Machinery, 1986, p. 313. [Електронний ресурс]. Режим доступу до ресурсу: https://doi.org/10.1145/10515.10549
3. M. I. Shamos and D. Hoey, “Closest-point problems,” *16th Annu. Symp. Foundations Comput. Sci. (Sfcs 1975)*, pp. 151–162, 1975, doi: 10.1109/SFCS.1975.8.
4. John Fisher, “Visualizing the connection among convex hull, voronoi diagram and delaunay triangulation.” [Електронний ресурс]. Режим доступу до ресурсу: https://pages.mtu.edu/~shene/PUBLICATIONS/2004/Hull2VD.pdf
5. “Voronoi-diagram.” [Електронний ресурс]. Режим доступу до ресурсу: https://github.com/Amit-Modi/Voronoi-Diagram/
6. “Cglib.” [Електронний ресурс]. Режим доступу до ресурсу: https://github.com/OGKG/CGLib