КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

Кафедра теорії та технології програмування

Кваліфікаційна робота

на здобуття ступеня бакалавра

за спеціальністю 122 Комп'ютерні науки

на тему:

РОЗРОБКА ЗАСТОСУНКУ ДЛЯ ОЦІНЮВАННЯ ЗНАНЬ СТУДЕНТІВ З ПРЕДМЕТУ «ОБЧИСЛЮВАЛЬНА ГЕОМЕТРІЯ ТА КОМП'ЮТЕРНА ГРАФІКА»

Виконав студент 4-го курсу
Германюк Всеволод Станіславович

(підпис)

Науковий керівник:
професор, доктор фіз.-мат. наук
Терещенко Василь Миколайович

(підпис)

Консультант:
Шкільняк Степан Степанович

(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 теорії та технології програмування
«01» червня 2022 р.,

протокол No 10

Завідувач кафедри

М. С. Нікітченко

(підпис)

КИЇВ – 2022

РЕФЕРАТ

Обсяг роботи 34 сторінок, 10 ілюстрацій, 1 таблиця, 13 використаних джерел, 3 додатки.

ОЦІНЮВАННЯ АЛГОРИТМІЧНИХ ЗАДАЧ, ПРОГРАМНИЙ ПРОДУКТ, ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ НАВЧАННЯ, REST API, КЛІЄНТСЬКА ПРОГРАМА

Дана робота носить проєктувальний характер.

Об’єктом розробки є система для оцінки виконання алгоритмічних завдань, з врахуванням можливих специфічних правил оцінювання, оформлення відповіді, тощо.

Метою роботи є створення системи, яка автоматизує перевірку викладачами виконання алгоритмічних задач.

Інструментами розробки є: мова програмування Python3; для валідації та серіалізації об’єктів використана бібліотека pydantic, для Rest API – django3.2, django-rest-framework, для web-клієнту - React. Середовище розробки – безкоштовне, вільно поширюване середовище розробки Visual Studio Code.

Результат роботи: Виконано огляд предметної області, проблематики задачі; Створено прототип системи, здатний перевірити правильність виконання задачі побудови опуклої оболонки множини точок методом Грехема.

Програмний продукт може застосовуватися у освітніх закладах для автоматизації перевірки розв’язання алгоритмічних задач. Перевірка завжди об’єктивна та економить робочий час викладача.

Сфера застосування - освітня. У перспективі функціонал системи можна розширити валідацією відповіді на задачу не алгоритмічного характеру, спрощенням для викладача створення та опису задач нових типів.

ЗМІСТ

ВСТУП

Оцінка сучасного стану об’єкта дослідження або розробки. Студенти сучасних вищих навчальних закладів у рамках освоєння освітніх програм вивчають дисципліни, частиною яких є розв’язання алгоритмічних задач.

Зважаючи на одноманітність таких задач, оцінювати їх виконання – рутинна робота для викладача, для якого не існує інструменту, який дозволить формалізувати вимоги до студентів, оцінювання таким чином, щоб можна було організувати автоматичну перевірку.

Особливу актуальність така система має у часи дистанційного навчання, коли викладач у будь-якому разі має перевіряти роботи віддалено та через комп’ютер, а студенти мають фотографувати роботи або оформлювати їх у цифровому вигляді.

На даний момент не існує рішень, які дозволяють оцінити виконання задачі з урахуванням потенційно складної системи оцінювання та інваріантів відповіді, наприклад, при вказанні множини не враховувати порядок її елементів.

Актуальність роботи та підстави для її виконання. Викладачі витрачають багато часу на дію, яку можна автоматизувати. Також існування подібної системи спростить написання роботи студентами, дозволить одразу дізнатися свій результат та коментарі щодо допущених помилок. В умовах пандемії дуже актуальним є питання цифровізації освіти, адже сучасне дистанційне навчання не обходиться без комп’ютерної техніки та доступу до всесвітньої мережі.

Мета й завдання роботи. Метою роботи є створення системи перевірки виконання алгоритмічних задач для спрощення їх перевірки.

Для досягнення мети поставлено завдання - Створити систему, яка дозволяє описати будь-які алгоритмічні задачі, формалізувати вхідні та

вихідні дані для них, створити алгоритм перевірки, який враховує усі деталі та інваріантності відповіді і згідно з ним оцінити виконання задачі. У цієї системи має бути інтерфейс для використання різними клієнтами, також необхідно створити web-клієнт для демонстрації роботи системи. Для цього необхідно спроєктувати гнучку схему взаємодії компонентів.

Таким чином, завдання можна розділити на три відносно незалежних блоки:

- Бібліотека для формалізації даних відповіді на задачу й умови, їх валідації, перевірки з використанням бібліотеки, у якій реалізовано дані алгоритми;
- Rest API, який слугує інтерфейсом для клієнтів, у ньому створено моделі користувачів і сутності, які пов’язані з оцінюванням задач та задачами;
- Web-клієнт, який використовує наявний функціонал Rest API.

Об'єкт, методи й засоби розроблення. Об’єктом розроблення є процес перевірки виконання алгоритмічної задачі з урахуванням потенційно комплексної системи оцінювання, започаткованої індивідуально викладачем, структури даних відповіді та умови, які використовує конкретний викладач.

Під час розробки продукту багато уваги приділялося повноті та мінімальності дій зі сторони викладача для опису бажаного вигляду задачі для студента, системи оцінювання. Для цього створено моделі, які можна перевикористовувати та змінювати для досягнення індивідуальних цілей.

Засобами розроблення є мова програмування Python3, для валідації, формалізації та серіалізації використано бібліотеку pydantic, для створення програмного веб інтерфейсу – django3.2, для програми веб-клієнта – React.

Можливі сфери застосування. Перш за все розглядається застосування системи у освітній сфері, але також систему можна

використовувати для перевірки коректності альтернативних алгоритмів розв’язання аналогічних алгоритмічних задач на прикладах задач, описаних у системі через програмний веб-інтерфейс, описаний у якості Rest API.

Взаємозв’язок з іншими роботами. У роботі використано алгоритмічну бібліотеку CGLib, розроблену в ході командної курсової роботи студентів Фісуненка Артема, Германюка Всеволода, Рудя Максима та Черкашина Михайла.

Створена бібліотека та backend створено за сприяння Фісуненка Артема, метод Грехема описаний Фісуненко Артемом і наведений в якості прикладу застосування бібліотеки користувачем.

РОЗДІЛ 1 ПРОЄКТУВАННЯ СИСТЕМИ ОЦІНЮВАННЯ

У цьому розділі описано процес проєктування основних сутностей системи - які ставилися вимоги, власне розв’язання задачі проєктування та коментар щодо виконання вимог.

1.1 Розгляд сутності “Задача”

Основні вимоги до моделі алгоритмічної задачі:

- Мінімальні вимоги до потенційно використаних бібліотек з реалізацією розв’язку алгоритмічних задач.
- Можливість опису всіх складових алгоритмічної задачі для легкого розширення набору задач системи.

Тип даних “Задача” задамо множиною можливих значень та операцій.

Дані, що описують задачу:

- Задача: (Тип задачі, Умова)
- Тип задачі: (Назва, Метод розв’язку, Структура відповіді)
- Метод розв’язку: (Опис даних на вході, Набір інструкцій з отримання відповіді, Опис даних на виході)
- Структура відповіді: Ієрархічний список етапів задачі, які у свою чергу поділені на підетапи - [[Підетап1, Підетап2, ...], [Підетап3, Підетап4, ...], ...]
- Умова - структура даних, яка відповідає опису даних входу для методу розв’язку даного типу задачі

Операції над типом “Задача”:

- Створення(Тип, Умова) - повертає інстанцію типу “Задача” з заданими даними;
- Отримати_Відповіді(Задача) - повертає структуру даних, що відповідає структурі відповіді заданого типу задачі.

Операції типу даних “Тип задачі”:

- Операція бієктивного приведення відповідей методу розв’язку до структури відповіді;
- Операція бієктивного приведення умов з наданих задачі до умов, структура яких описана у методі розв’язку даного типу задачі.

Таким чином, для того щоб формально описати алгоритмічну задачу у системі треба буде:

- Описати метод розв’язання даного класу задач, який приймає умови
- Створити тип задачі, у якому описано дві його операції і назва, привласнити створений метод розв’язання
- Створити умову, структура якої відповідає структурі умови методу приведення умов даного типу задачі
- Створити задачу за створеними типом задачі та умовою.

У подальшому для створення задач даного типу необхідним буде лише створення нових умов.

Наведена структура моделі алгоритмічної задачі виконує наведені вимоги.

1.2 Розгляд сутності “Умови задачі”

Вимоги до моделі умови:

- Алгоритм створення розробником нових типів умов має бути універсальним для великого різноманіття можливих умов, модель має валідувати отримані дані, має бути реалізована серіалізація та десеріалізація об’єктів класу.

Зважаючи на різноманіття потенційних умов алгоритмічних задач залежно від їх типу для їх формалізації використовується бібліотека Pydantic, яка дозволяє валідувати передану у задачу структуру умови згідно до описаної структури даних, яка може бути ієрархічною і використовувати композиції усіх доступних типів даних, що існують у бібліотеках мови Python3, або додатково описана розробниками. Також бібліотека дозволяє

серіалізувати об’єкти умов у формат JSON і робити зворотню операцію. Для кожного типу задачі структура умови описується індивідуально.

1.3 Розгляд сутності “Відповідь задачі”

Аналогічний до розгляду сутності “Умови задачі”, за винятком того, що відповідь у загальному випадку має розбиватися на етапи, які у свою чергу розбиваються на підетапи. Це необхідно для повної відповідності форматів у яких відповідають користувач та алгоритм, щоб розділити оцінювання задачі цілком на оцінювання окремих етапів. Тому структура відповіді має бути списком зі списками довільних об’єктів, структура кожного має бути описана індивідуально.

1.4 Розгляд сутності “Оцінювач”

Вимоги до моделі оцінювання:

- Для кожного етапу, на які розбита відповідь на задачу має бути можливість написати свій метод порівняння з урахуванням можливої інваріантності, тощо;
- Для кожного етапу задачі оцінювач рахує бали окремо, потім просумовує бали за кожний етап;
- Оцінювач має описувати знайдені помилки й мати можливість створення нових їх типів та описань;
- Має підтримуватися будь-яка кількість етапів та підетапів за умови коректності їх ієрархічного списку;
- Має бути можливість описання системних помилок, які мають оцінюватися не як звичайні - для системних помилок існує кількість однотипних помилок, для якої подальші помилки цього типу не знімають бали.

Тип даних “Оцінювач” задамо множиною значень та операціями.

Множина значень:

- Оцінювач: (Тип задачі, Дані оцінювання, Методи оцінювання)

- Дані оцінювання: [Дані оцінювання етапу]
- Дані оцінювання етапу: (Мінімальна оцінка, Максимальна оцінка)
- Методи оцінювання: [Метод оцінювання]

Операції:

- Оцінка(Завідомо правильна відповідь на задачу, Відповідь для перевірки) - Повертає кортеж(Оцінка, {Етап: Оцінка за етап}

Операції класу “Метод оцінювання”:

- Оцінити(Правильна відповідь, Надана відповідь, Бали за помилку) - Повернути список помилок у задачі, для кожної помилки вказати кількість знятих балів

Множина значень класу “Помилка”:

- Помилка: (Зняті бали, Опис, Системна);
- Опис - будь які дані, необхідні для пояснення природи помилки;
- Системна - помітка, що помилка системна. За такі помилки можна зняти лише обмежену кількість балів, скільки б їх не було.

Надана модель оцінювання виконує вимоги й є розширюваною, дозволяє побудувати різноманітні структури оцінювання для кожної алгоритмічної задачі й описати помилки користувача довільним способом, наприклад з указанням місця та кількістю знятих балів.

1.5 Розгляд допоміжних сутностей системи

Для забезпечення більшої гнучкості системи створено додаткові сутності, які не є настільки цікавими з точки зору їх структури. Вони допомагають перетворити дані згідно з вимогами описаними у основних структурах

РОЗДІЛ 2 ОПИС ЗАДАЧ НА МЕТОД ГРЕХЕМА У СПРОЄКТОВАНІЙ СИСТЕМІ

2.1 Клас задач методу Грехема

Клас задачі реалізовано згідно з описаною у розділі 1 моделлю даних та операцій.

Поля класу:

- Клас для перетворення відповіді алгоритму до стандартного вигляду
- Опис методу (може використовуватися для графічного відображення умов задачі)
- Етапи задачі
- Алгоритм розв’язку задачі
- Суперклас – клас моделі даних задачі у загальному, використовується для наслідування методів згідно з парадигмою об’єктно-орієнтованого програмування

Поля об’єкту класу:

- Умова

Дійсно, задачі на один метод відрізняються лише умовами. Поля класу описують задачу у загальному розумінні, а інстанції класу відрізняються умовами і вся інша інформація є загальною для них

У свою чергу, класи етапів складаються з наступних полів:

- Опис етапу (може використовуватися для графічного відображення умов задачі)
- Список підетапів задачі

Класи підетапів мають наступні поля:

- Опис підетапу (може використовуватися для графічного відображення умов задачі)

Наведена структура даних описує структуру задач на метод Грехема як класу задач та як окремої задачі, тобто інстанції класу.

Цей опис займає 60 рядків коду та оформлений згідно зі стандартом PEP8. Схему класів наведено у додатку А

2.2 Опис структури даних умови та відповіді

У методі Грехема умовою є список точок, тобто список кортежів їх координат. Створено описи класу точки:

```
class Point(BaseModel):
    x: float
    y: float
```

Та списку точок:

```
class PointListCondition(Condition):
    point_list: list[Point]
```

Ці описи слугують для формалізації вимог на вході, дозволяють валідувати вхідні дані та перетворювати у різні формати.

Аналогічно до опису умови створено опис відповіді по етапах, ці описи використовуються у тих самих цілях, але більш об’ємні, тому не представлені у цьому розділі, схема наведено у додатку 2.

Загалом описи займають 50 рядків коду за умови оформлення за стандартом PEP8.

2.3 Функції-перетворювачі

Необхідні для перетворення даних умови у вигляд, що підходить до реалізованого алгоритму розв’язку задачі, та навпаки, для перетворення відповіді у допустимий вигляд, описаний для даної задачі.

Опис даних функцій займає 50 рядків коду за умови оформлення за стандартом PEP8.

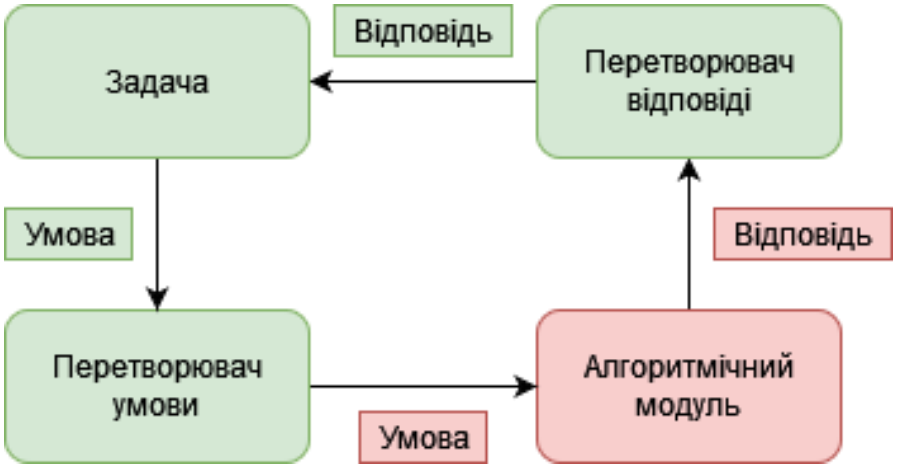
2.4 Схема отримання відповіді на задачу

Для створення нової інстанції задачі необхідно передати умову.

Далі, для отримання правильної відповіді на задачу використовуються наступні сутності (Рисунок 2.1):

1. Задача передає свою умову у Перетворювач умови, він повертає умову у форматі, прийнятному для алгоритмічного модуля
2. Алгоритмічний модуль отримує умову, обчислює правильну відповідь і передає її у Перетворювач відповіді, який «перекладає» умову у вигляд, описаний для даного класу задач.
3. Задача отримує відповідь у необхідному форматі для подальшого використання.

Даний підхід дозволяє користувачу бібліотеки абстрагуватися від алгоритму розв'язання задачі, не переписувати його для відповідності даних задачі та алгоритму.



Примітка 1: Зеленим позначено сутності, які необхідно описати користувачу бібліотеки, і механізм використання яких лежить у межах бібліотеки.

Примітка 2: Червоним позначено сутності, на структуру яких не впливає користувач бібліотеки в ході користування нею.

Рисунок 2.1 – схема отримання еталонної відповіді на задачу

2.5 Висновки щодо доречності використання бібліотеки.

Базові моделі бібліотеки описують логіку отримання відповіді загалом, а це спрощує процес формалізації алгоритмічної задачі.

Описана формалізація даних методу Грехема займає 160 рядків коду, з яких близько половини є порожніми для дотримання стандарту PEP8.

Формалізація даних алгоритмічної задачі буде використовуватися для:

- Можливості створити необхідні форми для студента, дані яких валідуються за типами і значеннями;
- Отримання відповіді на задачу у вигляді, який описано і для студента;
- Можливості порівняти відповіді алгоритмічного модуля і студента, тощо, алгоритмічно чи вручну;
- Можливості використання будь-якого алгоритмічного модуля у якості еталонного з мінімальною підготовкою.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ СИСТЕМИ ОЦІНЮВАННЯ БІБЛІОТЕКОЮ НА ПРИКЛАДІ МЕТОДУ ГРЕХЕМА

3.1 Схема оцінювання методу Грехема

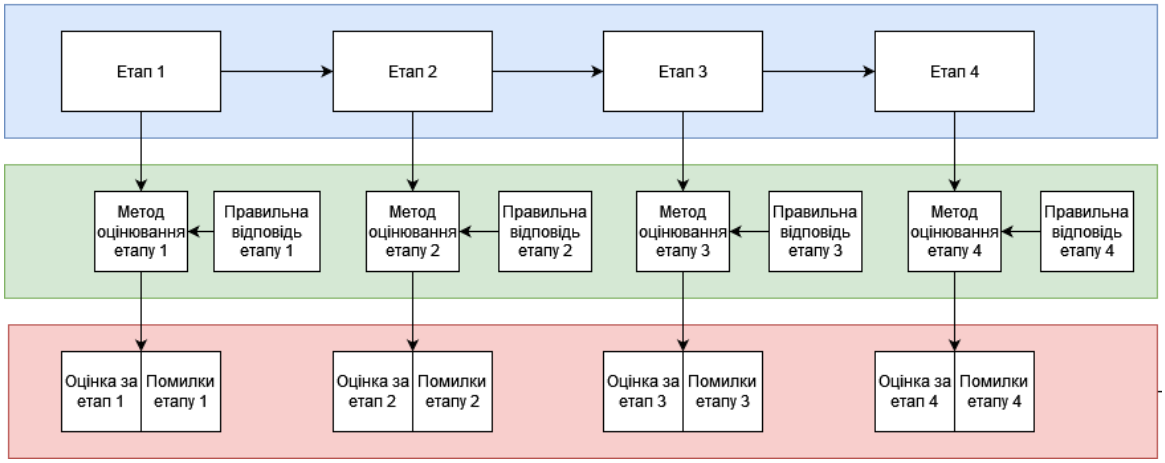
Схема оцінювання методу Грехема наведена **у додатку В**.

Для повної відповідності схемі оцінювання треба побудувати наступні структури даних:

- Можливі помилки кожного етапу;
- Створити методи оцінювання кожного етапу задачі;
- Створити дані оцінювання задачі;
- Створити оцінювач для даного типу задачі, використавши створені дані та створивши його методи оцінки.

Цей набір даних здатний описати наведену структуру методу оцінювання.

3.2 Загальна схема оцінювання



Примітка 1. Синім позначено надану на оцінку відповідь.

Примітка 2. Зеленим позначено операції та сутності на рівні опису даних оцінювання задачі.

Примітка 3. Червоним позначено дані, які повертаються алгоритмом оцінювання й використовуються для вирахування остаточної оцінки.

Рисунок 3.1 – Схема оцінювання відповіді на алгоритмічну задачу

При обчисленні оцінки за відповідь студента (Рисунок 3.1):

1. Відповідь розбивається на етапи задачі
2. Для кожного етапу викликається метод його оцінювання, який порівнює надану відповідь з правильною і повертає оцінку за етап і список помилок
3. Отримані пари (Оцінка, Помилки) утворюють загальну оцінку за наступною схемою: від оцінки за етап віднімаються відповідні штрафи за помилки, властиві для кожної окремої помилки.
4. Повертається фінальна оцінка

Дана схема оцінювання реалізована у бібліотеці й використовується за замовчуванням. За потреби може бути реалізована принципово інша схема.

Схема призначена для оцінювання розв'язку алгоритмічних задач і не підходить для оцінювання задач іншого характеру, адже використовує порівняння з «еталонним» розв'язком задачі. За умови деякої свободи дій для отримання вірного результату цією схемою неможливо валідувати правильність виведення, тощо.

Для валідації відповіді варто використовувати іншу схему оцінювання, яку за умови готової її реалізації можна використати, вказавши відповідні методи відповідних класів у описі задачі.

3.3 Реалізація оцінювання для методу Грехема

Декларативна частина опису виглядає так:

```
internal_point = StageMarkData(max_mark=0.25)
ordered = StageMarkData(max_mark=0.25)
origin = StageMarkData(max_mark=0.25)
steps_table = StageMarkData(max_mark=1.25)


class GrahamGrader(Grader):
    markdata = MarkData(stages=[internal_point, ordered, origin, steps_table])
        stage_grading_methods = [default025, iterable025, default025,
steps_table_grading]
```

На цьому фрагменті ініціалізуються етапи задачі, полем в них є максимальна оцінка за етап.

Потім описується клас GrahamGrader, у якому описуємо послідовність його етапів та відповідні їм методи оцінювання.

При оцінюванні методи співставляються етапам за відповідністю індексів. Якщо довжина списку етапів не співпадає з довжиною списку методів оцінювання, то співставлення відбувається з коротшим зі списків.

У прикладі наведено приклади оцінювання. default025 – метод оцінювання, який означено за допомогою часткового застосування стандартного методу бібліотеки із фіксованим параметром штрафу за помилку – 0.25

```
default025 = partial(default_grading, sub=0.25).
```

Цей метод оцінки є найбільш тривіальним, він повертає одну помилку з відповідним штрафом за умови нерівності «еталонної» та наданої відповідей, інакше – повертає порожній список помилок.

`iterable025` – Аналогічно, за винятком того, що застосовується така ж схема до кожного елементу з переданої колекції, цей метод відповідає етапу, дані якого є списком точок.

`steps_table_grading` – Приклад специфічного методу оцінки, описаного незалежно від наявних у бібліотеці.

Даний приклад ілюструє, що з використанням бібліотеки можна суттєво полегшити задачу опису алгоритму оцінювання за умови використання стандартних методів оцінювання та їх композицій. Також ілюструється можливість реалізації нетривіального методу оцінювання етапу, з можливим урахуванням складної структури даних етапу, тощо.

3.4 Висновки щодо доречності використання бібліотеки для опису схеми оцінювання

Загалом опис схеми оцінювання займає 65 рядків за умови оформлення згідно до стандарту PEP8. Описано чотири методи оцінювання, три з них використовують реалізований бібліотекою функціонал, що спрощує створення схеми оцінювання у рамках бібліотеки.

Логіка реалізації схеми оцінювання відділена від опису задачі, що дозволяє створювати різні схеми оцінювання для одного класу задач і дозволяє гнучко використовувати наявний у бібліотеці функціонал.

РОЗДІЛ 4 WEB BACKEND БІБЛІОТЕКИ

4.1 Огляд необхідного функціоналу.

Для повноцінної роботи системи необхідно, щоб у ній можна було створити задачі для перевірки. Таке право має тільки викладач. Для студента має бути можливість перегляду задач у системі, відкрити умову конкретної задачі та відправити відповідь до неї. Отож, у базі даних необхідно зберігати умови задач, користувачів, а для користувача необхідно зберігати його роль – викладач, чи студент. Також, краще створити моделі для композиції задач та студентів - модульні контрольні роботи та групи студентів відповідно.

При відправці студентом задачі необхідно вичитати відповідну задачу з бази даних, знайти на неї відповідь, перетворити її у єдиний формат з відповіддю студента, потім оцінити згідно зі схемою оцінювання даної задачі та повернути оцінку за задачу.

Сутність задачі у базі даних було вирішено зберігати як шлях до файлу, у якому записана задача. Це пов’язано з тим, що наразі задача не підтримує серіалізацію виключно даних, що пов’язані з нею.

Тобто, для створення задачі у системі необхідно створити python-об’єкт задачі, серіалізувати його за допомогою pickle, стандартного модуля мови Python і завантажити даний файл на сервер. До того ж бажано дати задачі ім’я та опис.

Для роботи зі сторони викладача вирішено використовувати Django-admin, модуль фреймворку Django для контент-менеджменту. Для клієнта створено кінцеві точки веб-інтерфейсу, які дозволяють використовувати вище описаний функціонал.

4.2 Можливе застосування Rest API бібліотеки

Перш за все Rest API призначений для демонстрації роботи бібліотеки. З його допомогою за використання будь-якого HTTP-клієнта доступний основний функціонал бібліотеки через мережу Internet, тощо.

На основі даного Rest API можна побудувати повноцінну систему для оцінки алгоритмічних задач. Один з можливих варіантів реалізації – викладач може створювати модульні контрольні роботи, попередньо додавши їх задачі у систему, після цього назначає групі студентів відповідні МКР. Студенти бачать завдання і можуть здати розв’язки тільки під час МКР. Після відправки відповідей студенти і викладачі одразу бачать оцінки з описом допущених помилок, тощо.

У рамках даної роботи Rest API застосовується web-клієнтом бібліотеки, функціонал якого описано у розділі 5.

4.3 Робота з задачами через Rest API бібліотеки

Основний функціонал програмного інтерфейсу пов’язаний із задачами – створення, отримання умов та списку задач.

Отримання списку задач (Рисунок 4.1):

```
GET /tasks/
```

```
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "id": 1,
        "name": "test",
        "description": "test",
        "pickle_dump": "http://192.168.1.174:8000/tasks/kek_65W1TTB.pickle"
    },
    {
        "id": 2,
        "name": "task2",
        "description": "task2",
        "pickle_dump": "http://192.168.1.174:8000/tasks/kek_z5me4GS.pickle"
    }
]
```

Рисунок 4.1

Отримання умов задачі (Рисунок 4.2):

Рисунок 4.2

4.4 Процес створення задачі у системі

Для створення задачі необхідно ініціалізувати задачу відповідного типу мовою Python3, після чого за допомогою стандартного модуля pickle записати об’єкт у файл. Після цього створити у системі за посиланням /admin задачу, вказавши назву, опис та завантаживши відповідний файл. Після цього задача з’явиться у загальному списку і можна буде отримати її умови.

РОЗДІЛ 5 WEB КЛІЄНТ БІБЛІОТЕКИ

5.1 Загальні положення

Для реалізації клієнта було обрано фреймворк мови Javascript, React. Задача клієнта – демонстрація роботи бібліотеки через Rest API

При реалізації застосовано функціонал фреймворка, що дозволяє писати компоненти інтерфейсу, які можна використовувати повторно, означено основні компоненти для роботи з задачами обчислювальної геометрії, реалізовано форму для методу Грехема, що повністю відповідає структурі даних, описаних за допомогою бібліотеки.

5.2 Огляд функціоналу

Було реалізовано дві основні сторінки – зі списком задач та сторінку з формою для відправки відповіді на задачу на метод Грехема.

Шляхи:

- /graham

На сторінці відображається список задач у системі (Рисунок 5.1). При кліку на задачу зі списку відбувається перехід на сторінку відповідної задачі

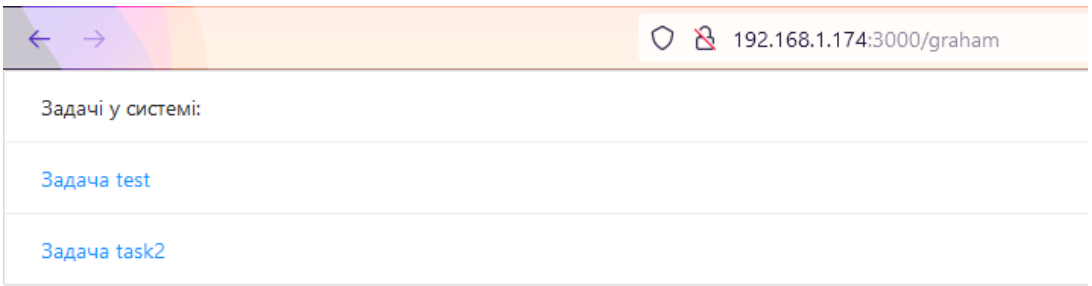


Рисунок 5.1 – список задач у тестовій системі

- /graham/:id

На сторінці браузера відображається форма для відправки відповіді на задачу(Рисунок 5.3). На ній також виводиться умова (Рисунок 5.2) і відображається поточна оцінка

Побудуйте оболонку методом Грахама. Точки: (6, 4), (4, 2), (4, 0), (1, 0), (3, 2), (2, 4)

Рисунок 5.2 – Умови обраної задачі

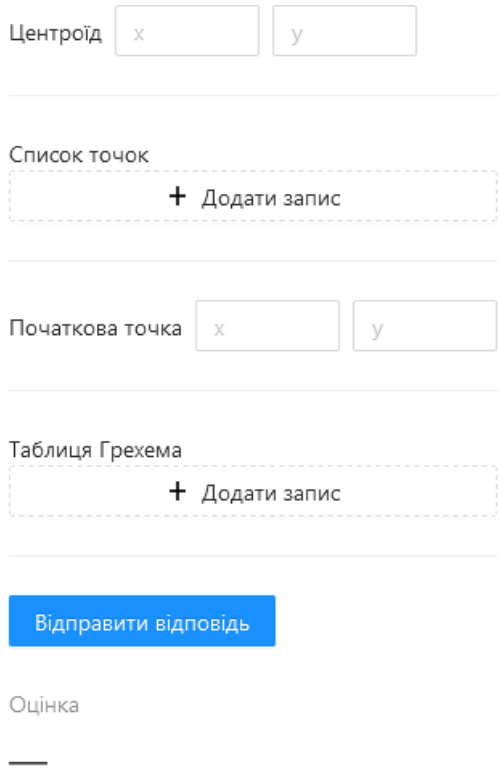


Рисунок 5.3 – Форма для відповіді до обраної задачі

5.3 Основні компоненти

Для вводу числа використовується компонент Number

Точка на площині задається двома координатами:

```
const Point = (props) =>
<Space>
    {props.label||""}
    <Number placeholder="x" name={[...[...props.name], "x"]}/>
    <Number placeholder="y" name={[...[...props.name], "y"]}/>
</Space>
```

Для вводу довільної кількості точок реалізовано наступний компонент:

```
const PointList = () =>
```

```
<>
    Список точок
    <Form.List name="points">
    {(fields, { add, remove }) => (
    <>
        {fields.map(({ key, name }) => (
        <Space key={key} style={{ display: 'flex', marginBottom: 8 }}>
            <Point name={[name]}/>
            <MinusIcon onClick={() => remove(name)} />
        </Space>
        ))}
        <AddButton onClick={() => add()}/>
    </>
    )}
    </Form.List>
</>
```

Він використовує компонент для вводу даних точки, дозволяє додавати нові точки та видаляти існуючі. Використовується для введення списку точок у форму.

Для введення таблиці Грехема використовується схожий компонент, його ідея аналогічна. Різниця лише у компонентах, які додаються чи видаляються, у цьому випадку він складається з трійки точок, окремої точки та двох компонентів Checkbox.

Компонент GrahamForm складається з описаних вище компонентів. При натисканні на кнопку форми відбувається перетворення інформації, зібраної формою у вигляд, який відповідає описаному в моделі даних задачі і дані відправляються на сервер, відповідь оцінюється і оцінка зображується у відповідному компоненті (Рисунок 5.4):

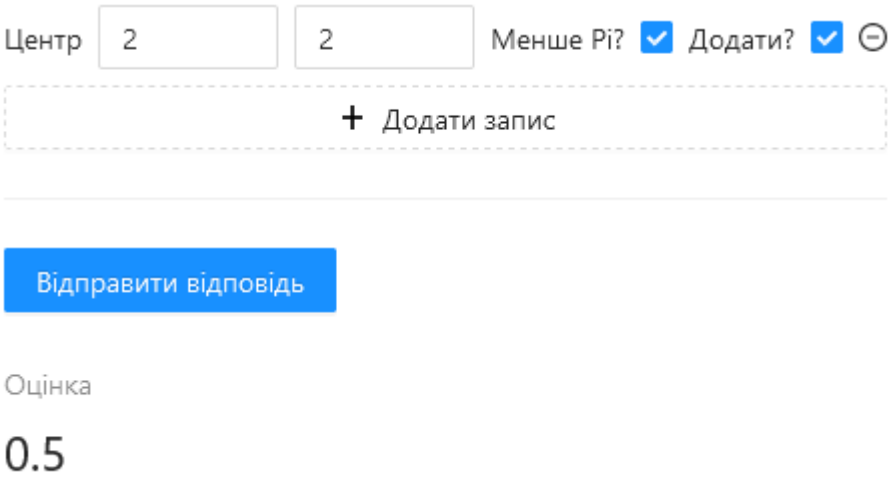


Рисунок 5.4 – Компонент, який зображує оцінку

ВИСНОВКИ

В ході роботи розроблено та реалізовано наступні компоненти системи оцінювання алгоритмічних задач:

1. Бібліотека для формалізації алгоритмічних задач
2. Backend server для використання формалізованих у бібліотеці задач у мережі Інтернет
3. Web-клієнт для демонстрації роботи системи

За допомогою бібліотеки повністю формалізовано задачу побудови опуклої оболонки методом Грехема з предмету «Обчислювальна геометрія та комп'ютерна графіка», додано відповідний функціонал у Web-клієнт.

Оцінка одержаних результатів. Використання бібліотеки для формалізації алгоритмічних задач спрощує задачу створення системи автоматизованого оцінювання алгоритмічних задач. Використовуючи бібліотеку на прикладі методу Грехема побудовано повністю автоматизовану систему перевірки. Функціонал бібліотеки є корисним і легко розширюваним для потреб користувачів, будь який ключовий аспект з реалізації за замовчуванням за потреби можна змінити. Потенційно, за сприяння бібліотеки можна повністю позбавити викладачів від потреби перевірки правильності виконання алгоритмічних задач і залишити їм більше часу для наукової діяльності, тощо.

Відповідність результатів сучасному рівню знань та технологій. Під час розробки всіх компонентів системи використані сучасні програмні продукти, які відповідають високим стандартам індустрії і є популярними серед розробників. Результати роботи є унікальними, адже у Україні ще не існує системи з настільки широким функціоналом і можливістю описати всі тонкощі як самої суті алгоритмічної задачі так і системи її оцінювання.

Соціально-економічна значущість роботи. На мою думку, подібна система може стати першим кроком у повністю автоматизоване оцінювання

робіт студентів, школярів, тощо. Перевірка робіт – напевне один з найнеприємніших аспектів роботи педагога, адже його завдання – навчити людей розв’язувати задачі, а якщо неприємний процес можна автоматизувати, то це підвищить мотивацію працівників навчальних закладів, збереже їх робочий час на щось більш цінне.

Доцільність продовження розробок за даною тематикою, перспективи розвитку. Продовження розробок стосовно вже наявних ідей вважаю доцільним через те, що алгоритмічні задачі – значна частина від усіх, але ще більш значною є частина задач не алгоритмічного характеру. Для перевірки їх розв’язання необхідно застосувати принципово інший підхід, як було описано у розділі 3.2. Тобто, можна дуже помітно збільшити спектр використання системи.

Також у планах:

- Робота над зменшенням обсягу дій, необхідних для розгортання подібних систем – створення адаптивного клієнта, який здатний створювати форми залежно від типів даних, які формалізовано у задачі, спрощення механізму створення задачі у системі;
- Створення графічного редактора для спрощення формалізації алгоритмічної задачі;
- Створення редактора у клієнті для заміни листочку під час написання роботи студентом;
- Розробка модуля для генерації документації щодо системи оцінювання задачі для повного її розуміння студентами

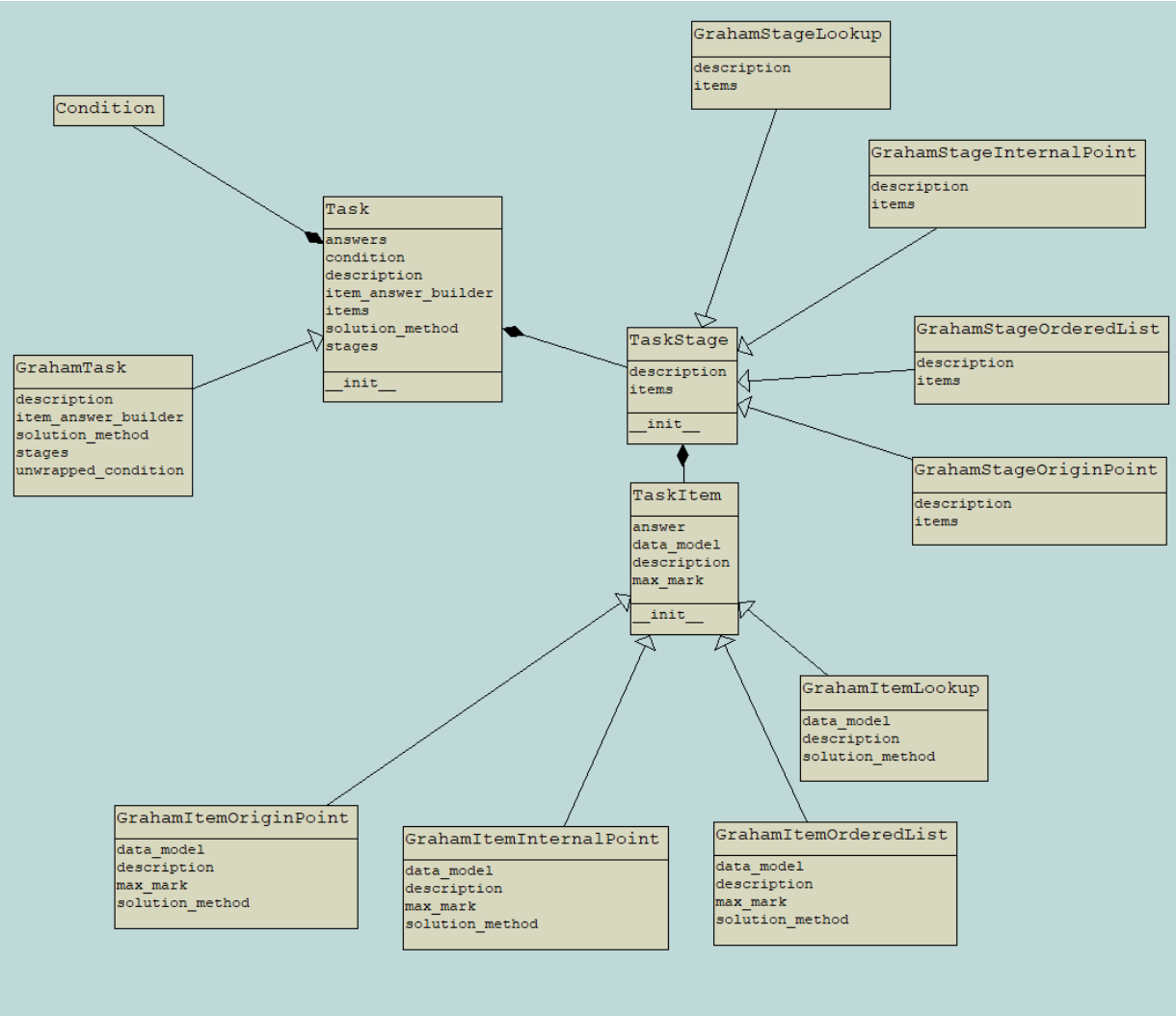
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Vasyl Tereshchenko, Yaroslav Tereshchenko. Point Triangulation using Graham's Scan . Journal of Data Processing Vol., 4, N 3, September, 2014, pp. 100-105.
2. Downey A. Think Python: How to Think Like a Computer Scientist. 2nd ed. Needham, Massachusetts : Green Tea Press, 2015. 244 p.
3. Dr. Charles R. Severance. Python for Everybody: Exploring Data Using Python 3. 2nd ed. 2009. 249 p.
4. Kuhlman D. A Python Book: Beginning Python, Advanced Python, and Python Exercises.
5. PEP 484 – Type Hints [Електронний ресурс]:[Веб-сайт] – Режим доступу до ресурсу: https://peps.python.org/pep-0484/
6. PEP 8 – Style Guide for Python Code [Електронний ресурс]:[Веб-сайт] – Режим доступу до ресурсу: https://peps.python.org/pep-0008/
7. Pydantic documentation – [Електронний ресурс]:[Веб-сайт] – Режим доступу до ресурсу: https://pydantic-docs.helpmanual.io/
8. React Documentation – [Електронний ресурс]:[Веб-сайт] – Режим доступу до ресурсу: https://uk.reactjs.org/docs/
9. Django Documentation – [Електронний ресурс]:[Веб-сайт] – Режим доступу до ресурсу: https://docs.djangoproject.com/en/3.2/
10. Django REST framework documentation – [Електронний ресурс]:[Веб-сайт] – Режим доступу до ресурсу: https://www.django-rest-framework.org/
11. Djoser documentation – [Електронний ресурс]:[Веб-сайт] – Режим доступу до ресурсу: https://djoser.readthedocs.io/en/latest/

12. JWT – [Електронний ресурс]:[Веб-сайт] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/JSON_Web_Token
13. Ant Design, документація до компонентів – [Електронний ресурс]:[Веб-сайт] – Режим доступу до ресурсу: https://ant.design/components/overview/

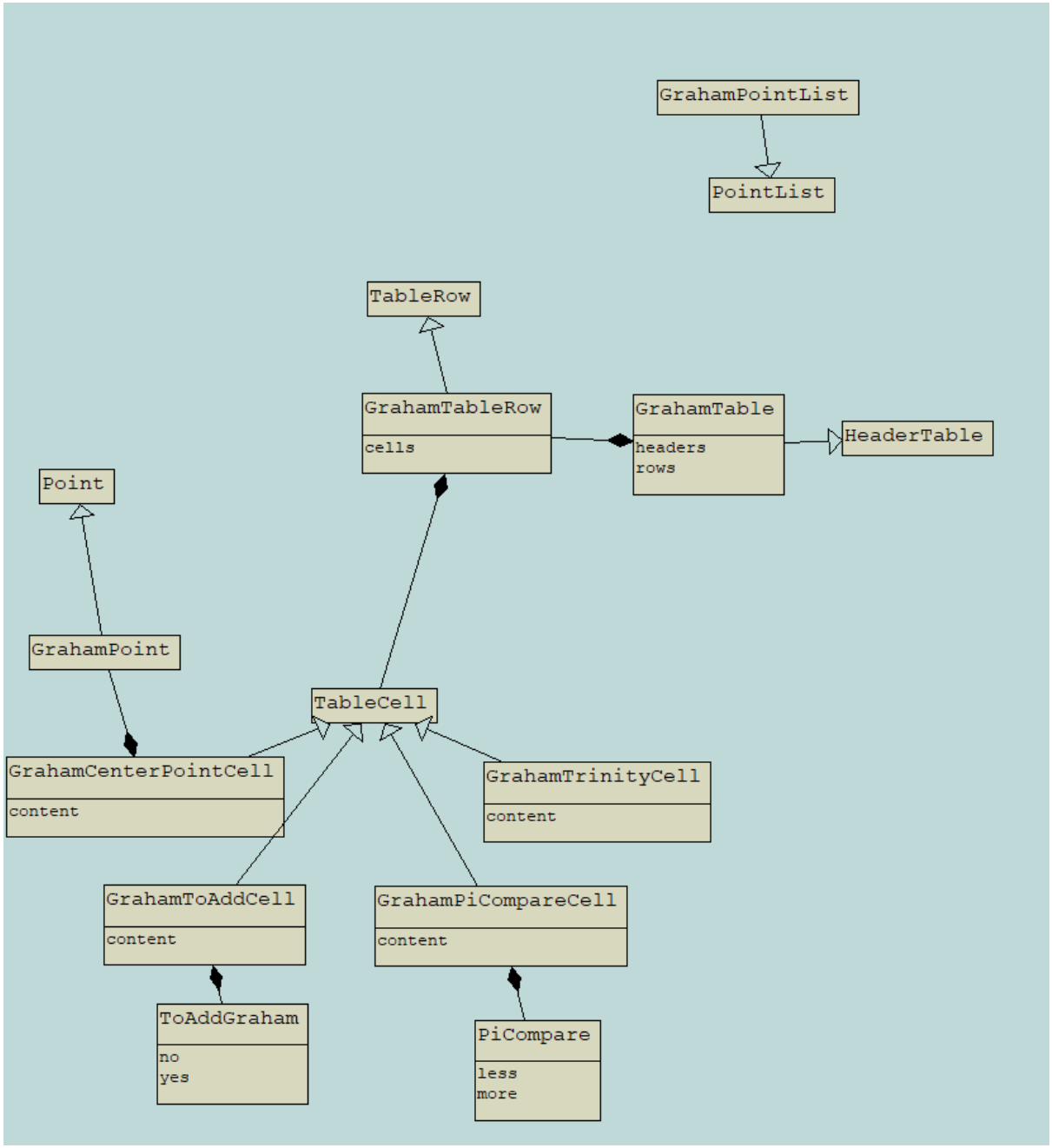
ДОДАТОК А

Схема класів системи та класів, що описують оцінювання методу Грехема



ДОДАТОК Б

Схема класів, що описують умову та відповідь на задачу Грехема



ДОДАТОК В

Описана схема оцінювання задач методу Грехема

Етап	Опис етапу	Бали	За що знімати бали	Скільки балів знімати
1	Задана множина S із N точок на площині. Знайти внутрішню точку q.	0.25	Не вірно знайдена, або відсутній крок	0.25
2	Використовуючи q як початок координат, побудувати упорядкований за полярним кутом список точок множини S починаючи із точки «початок» проти годинникової стрілки.	0.25	Не вірно упорядковано, або відсутній крок	0.25
3	Знайти точку «Початок».	0.25	Не вірно знайдена, або відсутній крок	0.25
4	Організувати обхід.	1,25	Не вірно виконується, або відсутній крок	1.25
4.1	Починаючи з точки «початок» розглядаємо трійки сусідніх точок.	0.15	Не вірно виконується, або відсутній крок	0.15
4.2	Перевіряємо внутрішній кут по відношенню до q, який утворює трійка.	0.15	Не вірно знайдено, або відсутній крок	0.15

4.3	Якщо кут менше ПІ – просуваємось вперед по списку на точку (яка перевірялась на крайність).	0.25	Не вірно виконується, або відсутній крок	0.25
4.4	Якщо кут більше або дорівнює ПІ – вилучаємо середню точку трійки і повертаємось на точку назад по списку	0.6	Не вірно виконується, або відсутній крок	0.6 – системна помилка 0.3 - поодинока
4.5	При досягнені точки «початок» третьою точкою трійки перевіряємо кут, якщо він менше ПІ зупиняємося, інакше повертаємось на точку назад по списку	0.1	Не вірно виконується (циклиться через точку «початок»), або не завершено крок	0.1