МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра інтелектуальних технологій

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
БАКАЛАВРА

на тему:

«Визначення віршового розміру українського тексту»

Галузь знань **12 «Інформаційні технології»**
Спеціальність **122 «Комп'ютерні науки»**
Освітня програма **«Комп'ютерні науки»**
Освітній рівень: бакалавр

Виконав: студент 4 курсу, групи КН- 41
Стус Олег Андрійович
Керівник: Тмєнова Наталія Пилипівна
канд. фіз.-мат. наук, доцент

Випускна кваліфікаційна робота бакалавра допущена до захисту
рішенням кафедри інтелектуальних технологій
Протокол № 11 від 06.06.2022 р.
зав. кафедри_____ доц. Іларіонов О.Є.

Київ - 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

Кафедра інтелектуальних систем

ЗАТВЕРДЖУЮ

Завідувач кафедри

інтелектуальних технологій

Іларіонов О.Є.

_____

«___»_____2022 р.

ЗАВДАННЯ

НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ

1. Тема проекту - «Визначення віршового розміру українського тексту», затверджена протоколом засідання кафедри від 23 грудня 2021 року протокол №4.
2. Термін здачі студентом закінченого проекту 29 травня 2022 року.
3. Вхідні дані до роботи над проектом - база словоформ з наголосами та відомості про віршові розміри в українській мові.
4. Зміст розрахунково-пояснювальної записки:
    1) Вступ.
    2) Аналітичний огляд та постановка задачі визначення віршового розміру.
    3) Проектні рішення для задачі визначення віршового розміру.
    4) Реалізація та тестовий приклад роботи інтелектуальної системи визначення віршового розміру.
    5) Висновки.
5. Перелік презентаційного матеріалу :
    1) Титульний слайд.
    2) Визначення віршового розміру.
    3) Актуальність задачі визначення віршового розміру.
    4) Мета роботи та поставлені завдання.
    5) Проектні рішення: діаграми та їх пояснення.
    6) Програмна реалізація: приклад програмної роботи, інтерфейс програми.

7) Висновки.

6. Консультанти з випускної кваліфікаційної роботи із зазначенням розділів випускної кваліфікаційної роботи, що їх стосуються.

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1			
2			
3			

7. Дата видачі завдання 15 лютого 2022 року

Керівник _____/ Тмєнова Н.П.

Завдання прийняв до виконання _____/(ПІБ)

КАЛЕНДАРНИЙ ПЛАН

Пор №	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів випускної кваліфікаційної роботи	Примітка
1	Визначення теми кваліфікаційної роботи	23.12.2021	
2	Дослідженння віршового розміру	03.01.2022 - 05.02.2022	
3	Створення першого розділу	03.01.2022 - 16.02.2022	
4	Дослідження та визначення методу віршового розміру	05.03.2022 – 19.04.2022	
5	Складання алгоритму роботи	05.03.2022 – 19.04.2022	
6	Визначення мови програмування	05.03.2022 – 19.04.2022	
7	Створення другого розділу	05.03.2022 - 24.04.2022	
8	Написання програмного коду	24.04.2022 – 10.05.2022	
9	Тестування програми	04.05.2022 – 10.05.2022	
10	Створення третього розділу	24.04.2022 - 10.05.2022	
11	Завершення вищої кваліфікаційної роботи	10.05.2022 – 29.05.2022	

Студент-дипломник _____ / Стус О.А. /

Керівник випускної кваліфікаційної роботи _____/ Тмєнова Н.П. /

РЕФЕРАТ

Випускна кваліфікаційна робота складається зі вступу, 3 розділів, висновків, списку використаних джерел, 15 джерел та додатків. Загальний обсяг роботи 61 сторінки. Робота має 4 таблиці та 14 рисунків.

Мета роботи - визначення віршового розміру, що може допомогти лінгвістам при встановлені часової епохи прози, визначення ритмічності, допомогти програмісту при написанні перекладача, голосового керування чи штучного інтелекту, що сам може створювати вірші.

У ході виконання дипломної роботи були виконані наступні задачі:

Пошук інформації стосовно віршового розміру, визначення різновидів віршового розміру, пояснення відміностей та надання прикладів кожного з виду розміру.

Дослідження двох методів, а саме: продуційна модель представлення знань та нейронна модель віршового розміру. Наведення формул цих трьох методів, наведення прикладу їх роботи.

Представлення та пояснення роботи алгоритму, наведення діаграм для розуміння картини проекту, визначення основних моментів роботи програми, надання текстового файлу(словника), пояснення його роботи у програмі.

Ключові слова – силабо-тонічна система визначення віршового розміру, віршовий розмір, продуційна модель.

THE ABSTRACT

The final qualifying work consists of an introduction, 3 chapters, conclusions, a list of sources used, 15 sources and appendices. The total volume of work is 61 pages. The work has 4 tables and 14 figures.

The purpose of the work is to determine the size of the poem, which can help linguists in establishing the time epoch of prose, determine the rhythm, help the programmer in writing a translator, voice control or artificial intelligence, which can create poems.

During the thesis the following tasks were performed:

Search for information on verse size, identify types of verse size, explain differences, and provide examples of each type of verse.

Research of two methods, namely: production model of knowledge representation and neural model of verse size. Giving the formulas of these three methods, giving an example of their work.

Presentation and explanation of the algorithm, providing diagrams to understand the picture of the project, determining the main points of the program, providing a text file (dictionary), explaining its work in the program.

Key words - syllabic-tonic system of verse size determination, verse size, production model.

ЗМІСТ

ВСТУП

Віршовий розмір – силабо-тонічний термін для позначення особливостей ритміки, покладеної в основі твору. У силабо-тонічному віршовому розмірі варіюється, головним чином, довжина рядка: ямбічний метр може виступати у вигляді одностопного, двостопного, тристопного і т. д. ямбічного розміру.

Характеристикою розміру є також наявність чи відсутність цензури та характер каталектики(*розділ умовної назви системи версифікаційних правил певної поезії, де відображаються ритмічні закінчення віршового рядка*).

Мета дипломної роботи – визначення віршового розміру, що може допомогти лінгвістам при встановлені часової епохи прози, визначення ритмічності, допомогти програмісту при написанні перекладача, голосового керування чи штучного інтелекту, що сам може створювати вірші.

Досягнення цієї мети пов’язано з вирішенням наступних задач:

1. Визначити, що таке віршовий розмір.
2. Встановити, які види віршового розміру є в літературі.
3. Визначити винятки віршових розмірів.
4. Написання програмного коду для досягнення мети.
5. Протестувати для перевірки коду.

Для виконання вказаних задач були реалізовано:

1. Робота зі строфами віршів.
2. Розділення рядка.
3. Методи, що дозволяють проводити токенізації на рівні речень та слів.
4. Визначення віршового розміру за допомогою бібліотеки Nestor.

РОЗДІЛ 1 ТЕОРЕТИЧНА ЧАСТИНА ТА ВИЗНАЧЕННЯ ВІРШОВОГО РОЗМІРУ

Віршовий розмір – термін для позначення особливостей ритміки, покладеної у основі певного віршового твору. Для визначення віршового розміру програма буде працювати зі строфами та рядками.

Строфа - поєднання рядків у вірші, що мають певний віршовий розмір, ритм, і мають інтонаційно-синтаксичну будову. У римованій поезії також має певну схему римування. У силабо-тонічній системі, крім двовіршів, розрізняються тривірші, чотиривірші, п'ятивірші, шестивірші, восьмивірші, дев'ятивірші, десятивірші. Програма буде працювати з чотиривіршами.

1.1 Актуальність визначення віршового розміру

Поети та літературознавці потребують знання про віршовий розмір певного віршового твору. Віршовий розмір у сукупності з іншими елементами для аналізу (структура вірша, його жанр, тощо) допомагає проаналізувати певну часову епоху або здобутки окремого поета для виявлення його особистого стилю. За допомогою віршового розміру можна зрозуміти ритм вірша. Визначення віршового розміру також необхідне у поетичному мистецтві для контролю ритмічності вірша.

З кожним роком кількість технологій зростає, вже починається заміна людей автоматичними роботами та програмами. Наприклад, перекладач, в якому використовується визначення віршового розміру для правильності написання перекладу з однієї мови на іншу, визначення наголосів та інше.

Ще однією сферою, де використовується визначення віршового розміру, є голосове введення. За допомогою диктування ви можете перетворювати слова в текст у будь-який час.

Голосове керування — спосіб взаємодії з пристроєм за допомогою голосу. На відміну від розпізнавання мови, голосове керування призначене для введення команд, що керують.

Розпізнавання окремих команд дещо простіше, ніж розпізнавання злитого тексту, і вимагає значних обчислювальних потужностей. Завдяки цьому сьогодні існує багатий вибір програмного забезпечення та обладнання (спеціалізованих цифрових сигнальних процесорів), що мають невелику вартість та високу якість розпізнавання команд.

Ритм вірша - потрібен для посилення сенсового навантаження, що є у слові. Також надає мові більшої сили і краси та допомагає усвідомити скритий сенс вірша, що не відразу розпізнається нашим розумом.

1.2 Дослідження досягнень у визначенні віршового розміру

Провівши аналіз україномовних, російськомовних та англомовних ресурсів, дійшов до висновку, що дуже мало є інформації з приводу встановлення віршового розміру у віршах.

Онлайн-ресурсів з приводу встановлення віршового розміру на всім мовах не було знайдено, але я знайшов одну програму, що допомагає встановити віршовий розмір як в україномовних, так і в російськомовних віршах. Програма називається RitmInMe, була створена Зігулею Сергієм. RitmInMe – програма високого рівня, що має в собі такий функціонал:

- перевірка ритмічності поезій;
- визначення віршового розміру;
- вбудовані римівники:

російська на 137 600 слів

(2 950 800 словоформ),

українська: 135 800 слів

(1948700 словоформ);

- розмітка римування;
- перевірка правопису;
- словники (російська, українська);
- ведення словників користувача;
- кількість складів у рядку;
- сервіс для роботи із віршами з інтернету.

Переваги та недоліки програми RitmInMe

Переваги програми RitmInMe:

1. Великий функціонал програми, перелік який я надав вище в тексті
2. Приємний та простий для розуміння зовнішній вигляд програми та її інтерфейс

Недоліки програми RitmInMe:

1. Програма не завжди правильно ставить наголоси слів, що дається взнаки далі у визначені віршового розміру.

2. Програма не є автоматичною, потребує користувача для зміни наголосів.

1.3 Види віршового розміру

Метричне віршування, що спирається на чергуванні довгих та коротких складів, притаманне античній поезії (квантитативне віршування), натомість у силабо-тонічній системі за основу береться чергування наголошених та ненаголошених складів (квалітативне віршування).

Віршовий розмір буває двоскладовим та трискладовим:

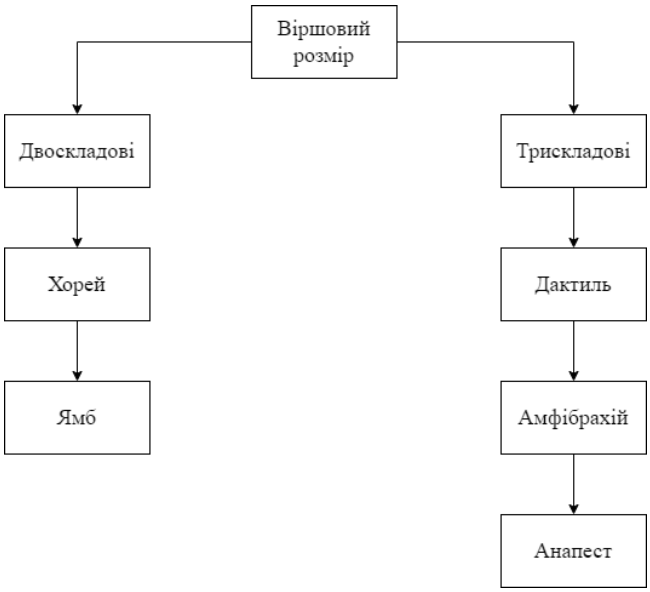


Рисунок 1.1 – Діаграма визначення видів віршового розміру

Я буду розглядати у двоскладовому віршовому розмірі лише основні види віршового розміру, оскільки пірихій та спондей є допоміжними і не мають великого впливу на розмір.

Для представлення віршового розміру вірша явикористовуватиму позначки U та _, де U – ненаголошений склад, _ - наголошений.

Двоскладовий віршовий розмір

Ямб — у силабо-тонічному віршуванні, поширеному в українській поезії, це двоскладова стопа із наголосом на другому складі.

У віршах, написаних ямбом, наголоси повинні падати на парні склади у кожному рядку (2-й, 4-й, 6-й тощо).

Розглянемо приклад:

Гурт Океан Ельзи – пісня «Сосни»

Ти вийшла заміж за весну,
ти вийшла заміж за весну.

U _/ U _/ U _/ U _/
U _ /U _ /U _/U _/

Хорей - двоскладова стопа з першим довгим і другим коротким складами.

У випадку з ямбом, у хореї наголос падає навпаки, а саме на непарні склади (1-й, 3-й, 5-й ...).

Приклад:

Гурт Плач Єремії – пісня «Вона»

Завтра прийде до кімнати твоїх друзів небагато,
Вип'єте – холодного вина.

_ U/ _ U/ _ U/ _ U/ _ U/ _ U/ _ U/ _ U/

_ U/ U U/_ U/ U U/ _

Трискладовий віршовий розмір

Дактиль - трискладова стопа з наголосом на першому складі (_UU).

У рядках дактилю наголоси мають падати на 1-й, 4-й, 7-й, 10-й склади тощо.

Приклад:

Гурт Тартак – пісня «Наше літо»

Сонце пече, річка тече
Влітку на вулиці так гаряче

_ U U/ _ _ U/ U _
_ U U/ _ U U/ _ U U/ _

Амфібрахій - трискладова стопа з середнім наголошеним складом (U_U).

Приклад:

Гурт Океан Ельзи – пісня «Все буде добре»

І все буде добре
Для кожного з нас.

U _ _/ U _ U
U _ U/ U _

Анапест - стопа, що складається з двох ненаголошених і одного наголошеного на кінці складу(UU_).

В анапесті наголос йде на наголошений склад в кінці, тобто на 3-й, 6-й, 9-й, 12-й.

Приклад:

Гурт Друга ріка – пісня «Відчиняй»

Відчиняй я стою на порозі
Я вже тут, я вже дома, привіт

U U _/ U U _/ U U _/ U
U U _/ U U _/ U U _

1.4 Вербальні методи віршового розміру

Найлегший спосіб визначити класичний метр - це скандівка. Треба відчути ритм, підставляючи наголоси там, де їх немає, але вони потрібні, і не роблячи їх там, де вони ніби є, але ритм велить не вдаряти. Це робиться інтуїтивно.

Також є ще один спосіб, більш спокійний, що не потребує викриків. Необхідно розставити наголоси або позначити межі стоп. Як правило, останні схематично позначаються знаком /, ненаголошені склади можна відзначити знаком U, а наголошені – нижнім підкресленням.

Для перевірки правильності поділу віршового рядка треба зіставити схематичне зображення твору. Інакше кажучи, ритмічний малюнок повинен співпадати.

Кількість наявних стоп у віршовому рядку визначає розмір тексту. Так, якщо в поетичному творі чотири стопи, до складу яких входить три склади, і наголос припадає на перший з них, то поезія написана чотиристопним дактилем.

Щоб правильно визначити розмір тексту, необхідно скласти схематичне зображення всіх рядків строфи, і подивитися - яким чином написано більшість рядків.

Правила постановки наголосу та розподілу рядка на стопи допомагають правильно визначити віршовий розмір поетичного твору. Окрему увагу важливо приділити звучанню віршового рядка визначення основного ритмічного рисунка.

Таблиця для визначення віршового розміру

	З якою періодичністю ненаголошені інтервали		
Розмір	Дуже часто або часто	Рідко або майже ніколи	Часто або рідко
Ямб	1, 3	0, 2, 4	5
Хорей	1, 3	0, 2, 4	5
Дактиль	2	0, 1, 3, 4	4
Амфібрахій	2	0, 1, 3, 4	4
Анапест	2	0, 1, 3, 4	4

1.5 Постановка задачі

Тема вищої кваліфікаційної роботи є «Визначення віршового розміру в українській мові»

Об’єктом дослідження даної кваліфікаційної роботи є система розпізнавання віршових розмірів силабо-тонічної системи віршування.

Предметом дослідження є продукційна модель представлення знань для автоматичного розпізнавання віршових розмірів.

Метою даної кваліфікаційної роботи є застосування засобів продукційних правил для автоматичного розпізнавання розмірів віршів, написаних українською мовою.

Для досягнення мети необхідно вирішити наступні завдання:

1. Проаналізувати україномовні вірші, написані у силабо-тонічній системі віршування .
2. Створити базу словоформ та наголосів.
3. Написати програму, що буде визначати віршовий розмір.
4. Провести тестування програми та при необхідності виправити помилки в коді.
5. Зробити виснововки та підвести підсумок.

У результаті роботи програми, що буде визначати віршовий розмір, має бути отримана відповідь, яким віршовим розміром написаний віршовий твір. Вхідні джерела – база словоформ та віршові твори.

1.6 Висновки до першого розділу

Отже, в першій частині дипломної роботи було розглянуто теоретично частину, а саме детально розібрано, що таке віршовий розмір, які види віршових розмірів бувають, та які є методи визначення віршових розмірів. Також було виявлено, що є програма, яка знаходить віршовий розмір. Назва цієї програми – RitmInMe. Дана програма виконана на високому рівні, проте в собі також має певні недоліки, наприклад програма не є автоматичною.

РОЗДІЛ 2 ПРОЕКТНІ РІШЕННЯ ВИЗНАЧЕННЯ ВІРШОВОГО РОЗМІРУ

Визначити модель для знаходження віршового розміру є дуже важливою частиною роботи, оскільки кожен з методів має свої плюси та мінуси. Для себе я визначив дві моделі – продуційна та нейронна. Також важливо розібратися якою мовою програмування буде написана програма, які бібліотеки будуть використані при встановлені віршового розміру.

2.1 Продуційна модель представлення знань

Продуційна модель — модель представлення знань, яка дозволяє представити знання у вигляді речень виду «Якщо, то», або простіше для програмістів «if, else»

Умовна частина продукції — Умова 1, Умова 2 і так далі, ще інакше називається антецедент. Частина проудкції, що працює, називається консеквентом. Умовою є деяке речення, за яким здійснюється пошук в базі знань, а консеквентом — дії, що виконуються при успішних результатах пошуку. Умови, які описують нинішній стан бази, розміщуються в робочій пам'яті. Вивід в такій базі може бути прямим (прямий вивід – це вивід від даних), або зворотним (зворотний вивід – вивід від цілі до даних). Даними є вихідні факти, що надходять в робочу пам'ять, на основі яких запускається вивід, що здійснює повтор «розпізнавання-дія» перебираючи правила з продукційної бази знань.

База знань продуційної моделі — це об'єднання бази фактів і бази правил. Кожне продукційне правило в базі знань втілює частину експертних знань, що можуть працювати самостійно, одержаних від експерта при набутті знань

вручну або використовуючи методи автоматичного видобування знань. При спільному застосуванні правил та наданні виведенню правила синергетично виробляють нові знання, видаючи кращий результат, ніж результати застосування окремих правил. В дійсності правила бази знань є взаємозалежними. Наприклад, додавання нового правила може конфліктувати з існуючими правилами і може вимагати перегляду правил загалом. Правила можна розглядати, в певному сенсі, як симуляцію поведінки експерта в деякій проблемній області.

Продукційна модель приваблює користувачів відносною простотою, наочністю, легкістю до корегувань, простотою схеми ло виводу. Існує велика кількість програмних засобів, що реалізують продукційну модель. Це так звані оболонки, або «пусті» бази знань. Прикладами таких баз знань є RuleBook та ін.

Ім'я продукції — унікальний ідентифікатор, що надається словом. Ім'я продукції дозволяє лише одним образом визначати продукцію у системі. Найбільш часто продукція задається за допомогою ідентифікаційного номера.

Умова застосування ядра продукції — логічний вираз, за допомогою якого активізується ядро продукції. Наприклад: якщо (певна умова) — істина, то ядро активізується, у протилежному випадку — активізації не відбудеться. У багатьох випадках умова застосування ядра відсутня у продукції, або об'єднується з ядром продукції.

Ядро продукції — центральний компонент продукції. Як правило, ядро продукції має вигляд речення-правила «Якщо *умова* то *результат*». Знак логічної секвенції (стрілка вправо) має зміст логічного випливання із істинного. Якщо *умова* не істинна, то про *результат* не можна зробити ніяких висновків. У базах знань умова ядра виступає також як логічний вираз, за яким здійснюється пошук у базі знань. Завершення ядра - процедура, що виконується при успішному завершенні пошуку.

Післяумова продукції - це опис процедур, які необхідно виконати у разі виконанні роботи ядра продукції. В нечітких продукційних системах представлення знань кожне з правил продукцій може додатково мати параметричну кількісну оцінку ступеня істинності правила.

2.2 Нейронна модель

Штучні нейронні мережі — це обчислювальні системи, що були створені на основі біологічних нейронних мереж мозку живих організмів. Google Neural Machine Translation (GNMT) – це система нейронного машинного перекладу (NMT), розроблена Google. Вона використовує штучну нейронну мережу для підвищення швидкості та точності Google Translate.

GNMT покращує якість перекладу, застосовуючи метод машинного перекладу на основі прикладів (EBMT), у якому система «вчиться на мільйонах прикладів». GNMT архітектура системного навчання була протестована більш ніж ста мовами, що підтримуються Google Translate. Завдяки великій наскрізній структурі система з часом вчиться створювати більш живі та натуральні переклади. GNMT намагається перекладати пропозиції повністю, а не частинами. Мережа GNMT може виконувати міжмовний машинний переклад, кодуючи семантику пропозиції, а не запам'ятовуючи пофразові переклади.

Кілька років тому Google почали використовувати рекурентні нейронні мережі (RNN) для безпосереднього вивчення відображення між вхідною послідовністю та вихідною послідовністю.

Рекурентні нейронні мережі (Recurrent Neural Networks) – це мережі, що містять зворотні зв’язки і дозволяють зберігати інформацію.

У той час як машинний переклад на основі фраз (GMNT) розбиває вхідну пропозицію на слова та фрази, які мають бути переведені значною мірою

незалежно один від одного, нейронний машинний переклад (NMT) розглядає всю вхідну пропозицію як одиницю перекладу. Перевага цього підходу полягає в тому, що потрібний менше вибір інженерного дизайну, ніж попередні системи перекладу на основі фраз. Коли він вперше з'явився, NMT показав еквівалентну точність з системами перекладу на основі фраз на наборах загальнодоступних еталонних даних скромного розміру.

2.3 Вибір мови програмування

Після визначення теми вищої кваліфікаційної роботи треба було визначити якою мовою програмування повинна бути написана програма. Вибір впав на дві мови програмування – Python та C#. Розберемо, яка мова програмування здається кращою.

Почнімо з Python. Інтерпретована об’єктно-орієнтовна мова програмування високого рівня зі строгою динамічною типизацією.

Переваги мови програмування Python

Гнучкість — це, основна перевага мови, так як завдяки цьому мова отримала популярність серед багатьох розробників.

Можливість розширення. Існують бібліотеки і фреймворки під будь-який тип завдань. Також величезним плюсом є те, що ми можемо використовувати C код з Python.

Синтаксис — код простий як для початківця, так і для вже досвідченого програміста. Код максимально виглядає чисто та гармонічно, без зайвих дужок, крапок з комою тощо.

Інтерпретованість. Інтерпретатор Python існує для всіх популярних платформ і за замовчуванням входить в більшість дистрибутивів Linux.

PEP — єдиний стандарт для написання коду, що робить код підтримуваним і читабельним навіть при переході від одного програміста до іншого.

Open Source — Python є відкритим, що дозволяє людям, яким цікаво прийняти участь у розвитку мови допомогти в розробці та покращені коду. Якщо дивитися деталі релізу однієї з версій мови, то можна побачити, що багато нового вункціоналу було створено стороніми програмістами.

Спільнота — приємна громада Python, яке готова прийти на допомогу починаючому або вже вмілому розробнику. Оскільки програмісти Python знаходяться по всій планеті, то завжди відкриті чати для спілкування з колегами по роботі, де можна знайти рішення проблеми.

Тепер переглянемо недоліки Python

Продуктивність - мова програмування Python є доволі повільною Це обумовлено тим, що Python є інтерпретованою мовою. Але навіть у порівнянні з іншими інтерпретованими мовами помітно, що Python програє в продуктивності. Але це легко можна нівелювати за допомогою C реалізацій того чи іншого проблемного ділянки коду. В умовах сьогоднішніх потужностей — це несильно помітно.

Динамічна типізація — вид типізації, при якому змінна зв'язується з типом у момент надання значення. Через динамічну типізацію Python споживає багато ресурсів, але це компенсується внутрішнім кешуванням.

Global Interpriter Lock. На даний момент це є основною проблемою продуктивності в Python, а також цим обумовлена погана реалізація

багатопоточності. GIL - це своєрідне блокування, що дозволяє лише одному потоку управляти інтерпретатором Python. Це означає, що у будь-який час буде виконуватися лише один конкретний потік. Робота GIL може бути несуттєвою для розробників, що створюють однопотокові програми. Але в багатопотокових програмах відсутність GIL може негативно позначатися на продуктивності процесоро-залежних програм.

Отже, вцілому Python є гарною мовою для написання коду визначення віршового розміру, оскілько у коді зможе розібратися будь хто, хто знає англійську мову. Python має багато бібліотек та фреймворків, тому знайти потрібну бібілотеку простіше. Також спільнота цієї мови завжди відкрита до допомоги, що не може не радувати.

Тепер переглянемо іншу мову програмування - C#.

Переваги C#

Корнева бібліотека - бібліотека є великою, та є доволі простою для використання у вирішенні простих завдань.

Мова досить сучасна, всі найпопулярніші особливості на ринку в ньому або є, або плануються. Синтаксис легше ніж Java, важчий ніж у сучасних мовах на кшталт Python. Код мови програмування C# при компіляції перетворюється на проміжну мову, яку виконує JIT-компілятор(*технологія збільшення продуктивності програмних систем, що використовують байт-код, шляхом компіляції байт-коду в машинний код або інший формат безпосередньо під час роботи програми*) на клієнтській машині — у цьому є свої переваги, але такий принцип роботи не залишає шансів писати настільки ж швидкий код, як на плюсах чи голанзі. C# є компілюємою мовою програмування, що означає, що

вона швидше інтерпретованих мов програмування. Але C# все ще досить швидкий у своїй категорії. Якщо просто сісти і писати незамудрений код, дуже довго не впиратимешся в якісь несподівані проблеми у продуктивності самої мови.

Мова не є надто швидкою — але є величезний простір для оптимізації. Структури – щоб зберігати дані на стеку, API збирача сміття, щоб оптимізувати його роботу в конкретних кейсах, є unsafe. Слово unsafe умові програмування C# означає небезпечний контекст, необхідний виконання будь-яких операцій із застосуванням покажчиків. Є багато інструментів для паралелізму.

У C# потужна підтримка ООП - тут будь-який код знаходиться у класах, підтримка класичного, а тепер і множинного успадкування через реалізацію інтерфейсів за замовчуванням. Є велика кількість модифікаторів доступу (private, private protected, protected, internal, protected internal, public), можливість розділяти відповідальність модулів на рівні збирання. Типізація строга статична - все що можна перевіряється на тип на етапі компіляції, все що не можна - метаінформація про типи їде в частину коду, що існує в виконуваному файлі, і може бути перевірена там.

Недоліки C#

Набагато менше мова програмуваннння C# підтримує функціональну парадигму. Функціональна парадигма — певна парадигма, яка розглядає програму як обчислення математичних функцій та уникає станів та даних, які можуть змінюватися. Така парадигма програмування наголошує на застосуванні функцій. Але й тут є чіткий рух уперед. Нещодавно завезли трохи кастрований патерн-матчинг, є тапли і майже є рекорди. Давно є лямбда виразу.

Найпоширеніший у дотнеті спосіб роботи з колекціями – LINQ вирази – виконаний повністю у функціональному стилі.

Але основну проблему мови це лише посилює. Вони завозять нові функціональні особливості, але безліч необхідних синтаксичних надбудов не дозволить їх використовувати без заморочок. Тут що далі, то гірше. Страшний вантаж зворотної сумісності легко розрулюється творцями мови технічно, але завжди виливається в ще одне обтяження синтаксису. Мені трохи соромно, коли я бачу код мінімально робочої програми на C# - мова буквально відмовляється звільняти мене від написання очевидного.

Тож, і Python, і C# є об'єктно-орієнтованими мовами загального призначення. Python буде чудовим варіантом, якщо ваш проект пов'язаний з дослідженням даних, оскільки він має велику стандартну бібліотеку. Вибір C# буде корисним для розробки адаптивних веб-сайтів, веб-сервісів та настільних програм.

Але чому я вибрав саме C# ? Мені потрібна була бібліотека, що працює саме з наголосами, і у мові програмування Python я її не знайшов, тому вибір C# також здався практичним.

2.4 База словоформ з наголосами

Можливо, одне з найважчих завдань було знайти базу словоформ з наголосами, оскільки дуже мало людей використовують саме українські словники, та і більш популярною до недавнього часу була російська мова.

Тож, вирушивши у пошуки в простору інтернету, перечитавши неймовірну кількість статей з цього приводу, було знайдено 3 варіанти словників української мови. Перший словник був звичайний, без наголосів, тому був швидко відкинутий.

Залишилося ще два варіанти. Один із них був словником ВЕСУМ. Дуже великий електронний словник, що славиться своїм різномаїттям слів, сленгів і тому подібне, мав при собі також список всіх словоформ та лем, які мені були потрібні. Була проблема в тому, що не зрозуміло, чи словоформи йшли з наголосами, оскільки щоб отримати текстовий файл бази словоформ треба було запустити Java код, і лише після цього з'являються ті самі бажані текстові файли. Тому я встановив Java та інтерпретатор IntelliJ IDEA для виконання коду. Результат мене шокував, оскільки не було ні однієї бази з наголосами.

Але я знайшов ще один словник, який повністю мав у собі всі словоформи з наголосами, що дуже сильно мене втішило, оскільки довелося б писати його вручну, що зайняло б дуже багато часу.

Отже, ця база знаходиться у текстовому файлі під назвою hagen. Знайдена вона була на Github, програміст використовував словник Залізняка та Хагена, звідси сама і назва. Оскільки словник надо великий, тому він буде прикріплений окремим файлом.(ссилка на базу словоформ)

2.5 Діаграма декомпозиції

Основний процес визначення віршового розміру представлений у вигляді діаграми декомпозиції на рисунку:

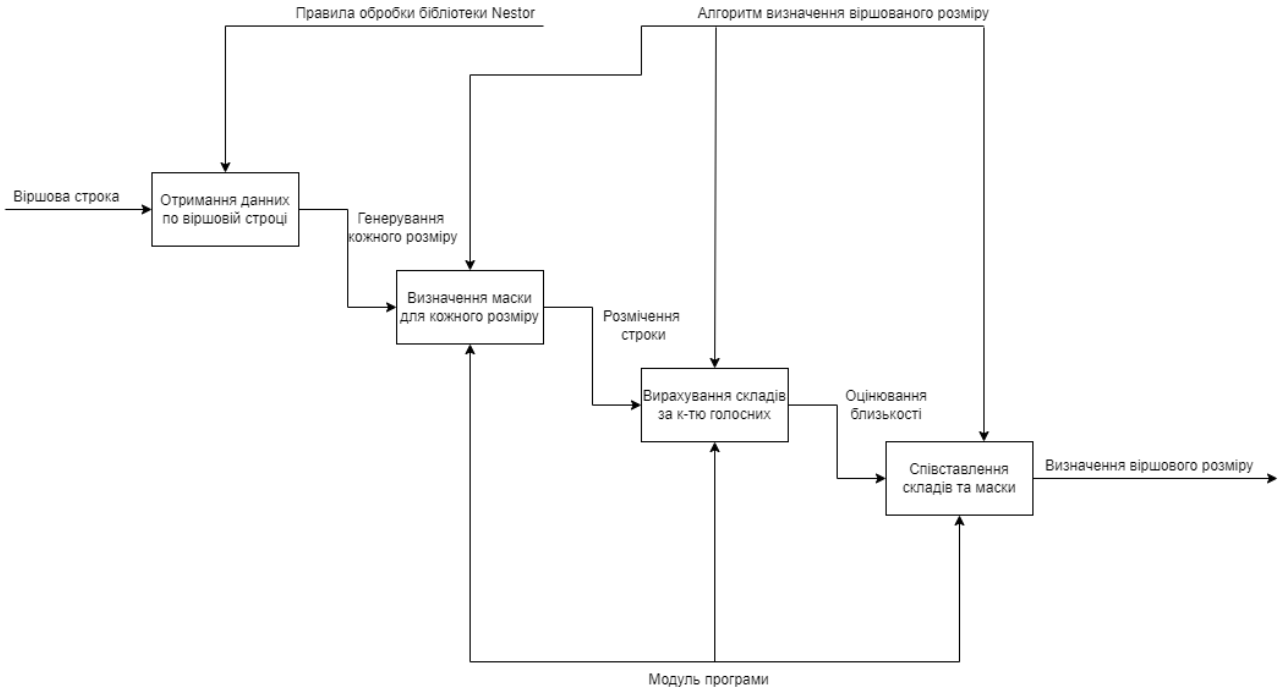


Рисунок 2.1 - Діаграма декомпозиції визначення віршового розміру

На діаграмі показано, що на вхід йде віршова рядок, з якої ми отримуємо дані, керуючись правилами обробки бібліотеки Nestor. Після цього алгоритм виконує генерування маски (шаблону) кожного розміру (ямб, хорей, дактиль, анапест, амфібрахій). Також ми розмічаємо нашу вхідну строку за певним алгоритмом та вираховуємо склади за кількістю голосних у слові. Потім, після проведення всіх операцій, наведених вище, іде оцінка близькості, тобто співставлення наших складів та маски, де за певними відхиленнями вираховується дистанція строки до віршового розміру.

2.6 Алгоритм програмного коду

Алгоритм визначення віршового розміру представлено в програмному коді таким чином:

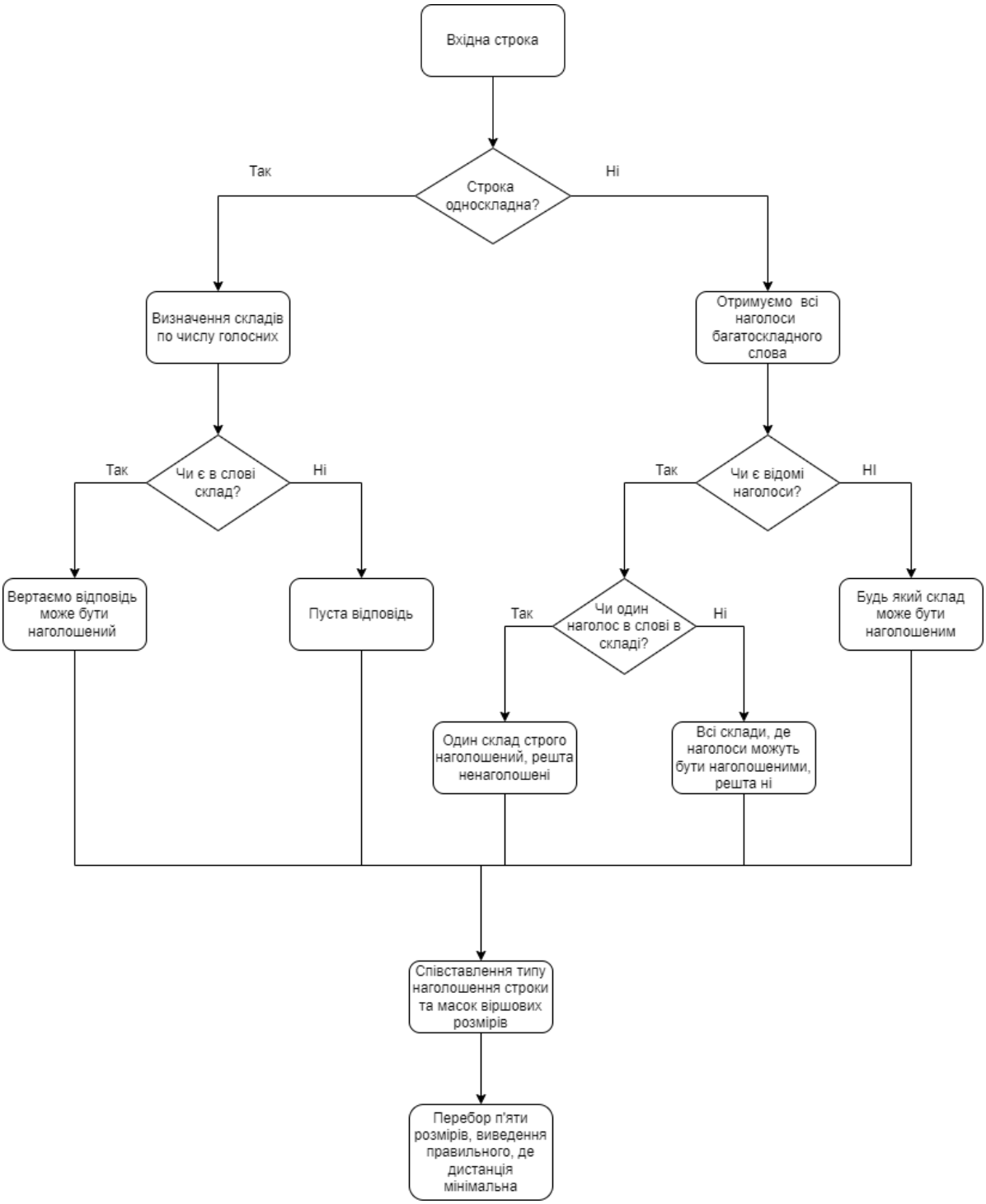


Рисунок 2.2 - Алгоритм визначення віршового розміру

На вхід подається текстовий рядок. Програма вважає, що це є вірш, рядок вірша. Наступною дією йде оцінення близькості отриманого програмою віршового рядка до п'яти віршових розмірів, що розглядаються.

Спочатку йде розмітка рядка. Для кожного слова повертається масив StressType, індекси у якому відповідають номерам складів у слові (номер складу починається з нуля). А значення можуть бути такими: «склад точно наголошений», «склад точно ненаголошений» і «склад може бути наголошеним».

Склади визначаються за кількістю голосних. Для випадку, коли у складі кількість голосних дорівнює нулю, записується порожня відповідь. Якщо в слові один склад, то одразу для складу повертається значення може бути наголошеним.

До цього було переглянуто логіку простих слів. Тепер ми розглянемо алгоритм роботи зі складними словами складних слів. Якщо слово з отриманого віршового рядка є у базі словоформ, отримуємо всі наголоси цього слова і складаємо в HashSet (невпорядкована динамічна структура унікальних елементів, яка має в собі лише значення) номери наголошених складів.

Якщо відомих наголосів немає, то будь-який склад може бути наголошеним. Якщо наголос один, то склад є строго наголошеним, а всі інші склади строго ненаголошені. Якщо ж наголосів декілька, то всі вони можуть бути наголошеними, а інші склади, відповідно, строго ненаголошені.

Для кожного розміру заздалегідь запишемо маску StressType, яка міститиме лише значення CanBeStressed (може бути наголошеним) та StrictlyUnstressed (строго ненаголошений), оскільки, якщо слово довге, під наголошеною часткою віршового розміру простим чином може бути ненаголошений склад.

Далі ми зліва направо зіставимо для кожного складу StressType вихідного рядка і StressType маски. Введемо деяке число dist, що показує штраф в оцінці

близькості (що менше dist, тим ближче). Спочатку значення dist дорівнює нулю. Далі при порівнянні складів можливі такі варіанти:

	В масці склад CanBeStressed (може бути наголошеним)	**В масці склад StrictlyUnstressed (строго ненаголошений)**
В строці склад StrictlyStressed (строго наголошений)	Наголошений склад у наголошеній частці, повний збіг. Не штрафуємо.	Наголошений склад не може потрапляти під ненаголошену частку у розмірі. +5 до штрафу
В строці склад CanBeStressed (може бути наголошеним)	Склад у рядку може бути будь-яким незалежно від маски. Не штрафуємо.	Склад у рядку може бути будь-яким незалежно від маски. Не штрафуємо.
В строці склад StrictlyUnstressed (строго ненаголошений)	У наголошеній частині розміру знаходиться ненаголошений склад рядка. Можливо, але не надто добре. +2 до штрафу	Ненаголошений склад у ненаголошеній частці, повний збіг. Не штрафуємо.

Залишилось лише перебрати п'ять розмірів і вивести той, для якого отримана дистанція мінімальна.

2.7 Діаграма основного процесу

Діаграма основного процесу визначення віршового розміру зображена на рисунку 2.3:

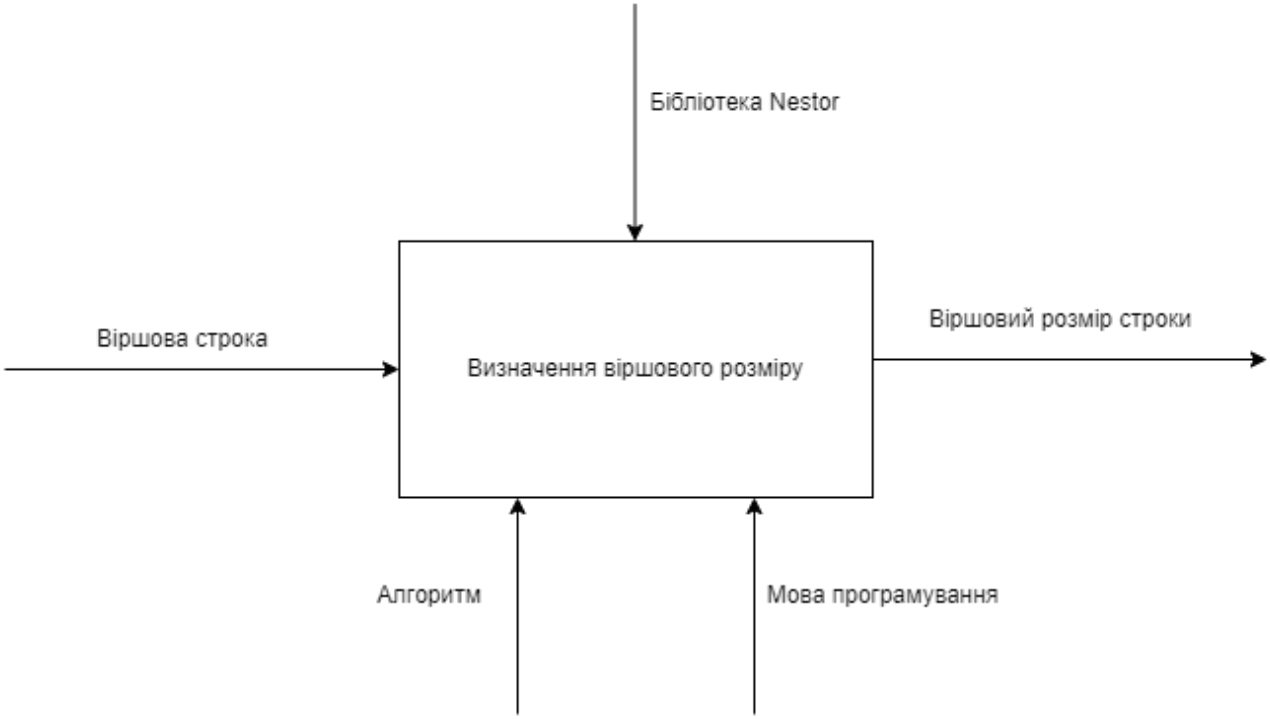


Рисунок 2.3 – Діагарама основного процесу

Вхід: віршова строка.

Механізми: алгоритм визначення віршового розміру, мова програмування.

Управління кодом: методи та правила бібліотеки Nestor.

Вихід: віршовий розмір вірша.

2.8 Висновки до другого розділу

У другій частині дипломного проекту було відображено алгоритм роботи програми, описана діагарма декомпозиції програмного коду, визначено основну роботу коду. Також з'ясовано використання прогрманого коду у нашому суспільстві та наведено приклади цього використання.

Було наведено три діаграми:

1. Діаграма декомпозиції
2. Алгоритм програми
3. Діаграма основного процесу

Діаграма декомпозиції призначена для деталізації роботи. На відміну від моделей, що відображають структуру організації, робота на діаграмі верхнього рівня в *IDEF0* - це не елемент управління роботами нижнього рівня.

Роботи нижнього рівня - це те ж саме, що і роботи верхнього рівня, але в більш детальному викладі. Як наслідок цього кордону роботи верхнього рівня - це те ж саме, що і кордони діаграми декомпозиції.

Також були розглянуті моделі визначення віршового розміру:

1. Продуційна модель представлення знань.
2. Нейронна модель віршового розміру.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ, ТЕСТОВИЙ ПРИКЛАД

3.1 Пояснення роботи програмного коду

Коментарі в коді наведені, але розберемо програму по частинам. Спочатку визначаємо змінні штрафу (WordToMaskMismatchPenalty та MaskToWordMismatchPenalty). Потім у програмі наведений метод, що визначає віршовий розмір по строці. Метод перебирає п’ять розмірів та виводить той розмір, де дистанція мінімальна.

Наступний метод підрахунку стоп по строці. Як параметр туди входить строка, що містить один рядок вірша. На виході ми маємо масив усіх стоп з відстаннями.

Третій метод – відстань до стопи. Йде обчислення між строковим наголосом та наголосом стопи. Параметрами методу виступають поточна стопа та поточні наголоси строки. Результатом буде відстань, яка чим менше, тим буде кращою.

Тут детальніше розбір:

```
if (lineStress != StressType.CanBeStressed)
        {
```

Якщо голосний рядка може бути наголошений чи ні, це нульова відстань до маскування, інакше:

```
switch (lineStress)
            {
                        case StressType.StrictlyStressed when maskStress ==
StressType.StrictlyUnstressed:
                dist += WordToMaskMismatchPenalty;
                break;
```

```
                case StressType.StrictlyUnstressed when maskStress ==
StressType.CanBeStressed:
                    dist += MaskToWordMismatchPenalty;
                    break;
```

Використовуючи конструкцію switch/case (конструкція, що оцінює вираження і порівнює його з набором значень, тобто порівняння сточним наголосом та наголосои стопи) в першому case маємо маску строго ненаголошену, коли рядкового голосного немає. Ми отримуємо великий штраф, тому додаємо значення WordToMaskMismatchPenalty, що дорівнює 5.

В другому case ми маємо випадок, коли маска може бути наголошеною, коли рядковий голосний у ненаголошеній, тобто отримуємо легкий штраф та додаємо 2.

Залишаються лише два випадки, при яких немає штрафу:

1.Коли рядковий голосний суворо наголошений, а маска може бути наголошена

2.Коли голосний рядковий і маска суворо ненаголошені

Наступний метод GetPoeticStresses(), з якого ми отримуємо всі можливі наголоси в словах з урахуванням можливої поетичної стопи. Параметр – вхідне слово, результат представлений у вигляді масивів наголосів, де кожен індекс – це число голосних у слові, яку починається з 0.

Розберемо метод докладніше.

```
int vCount = word.Count(NestorMorph.IsVowel);
```

В цій строці програмного коду ми перевіряємо за допомогою методу бібліотеки IsVowel() чи є буква наголошеною, повертаємо у вигляді числа. У випадку, якщо немає голосних, тобто змінна дорівнює 0, ми повертаємо порожній масив.

Якщо ж ми маємо одну голосну букву, повертаємо одиночний нечіткий наголос. Чому нечіткий, бо в поезії слово з одним складом може бути як наголошеним, так і ненаголошеним.

Третій варіант, коли в нас немає відомих наголосів, тоді повертаємо кожен голосний може бути наголошеним.Робимо перевірку голосної букви, що тільки певні голосні є наголосами, будь-які інші ні. Повертаємо остаточний наголос.

Також у нас є Foot клас, де ми визначаємо маски наших віршових розмірів:

```
public StressType[] Mask => Type switch {
        FootType.Unknown => new [] { StressType.StrictlyUnstressed },
        FootType.Iambic => new []{ StressType.StrictlyUnstressed,
StressType.CanBeStressed },
        FootType.Chorea => new []{ StressType.CanBeStressed,
StressType.StrictlyUnstressed },
        FootType.Dactyl => new []{ StressType.CanBeStressed,
StressType.StrictlyUnstressed, StressType.StrictlyUnstressed},
```

```
            FootType.Amphibrachium => new []{ StressType.StrictlyUnstressed,
StressType.CanBeStressed, StressType.StrictlyUnstressed },
            FootType.Anapest => new []{ StressType.StrictlyUnstressed,
StressType.StrictlyUnstressed, StressType.CanBeStressed },
            _ => throw new ArgumentOutOfRangeException()
        };
```

Записуємо типи стоп та типи наголосів:

```
public enum FootType
    {
        Unknown,
        Iambic,
        Chorea,
        Dactyl,
        Amphibrachium,
        Anapest
    }

    public enum StressType
    {
        StrictlyUnstressed,
        CanBeStressed,
        StrictlyStressed,
    }
```

А також повертаємо довжину маски віршових розмірів:

```
public StressType[] GetMaskOfLength(int l)
```

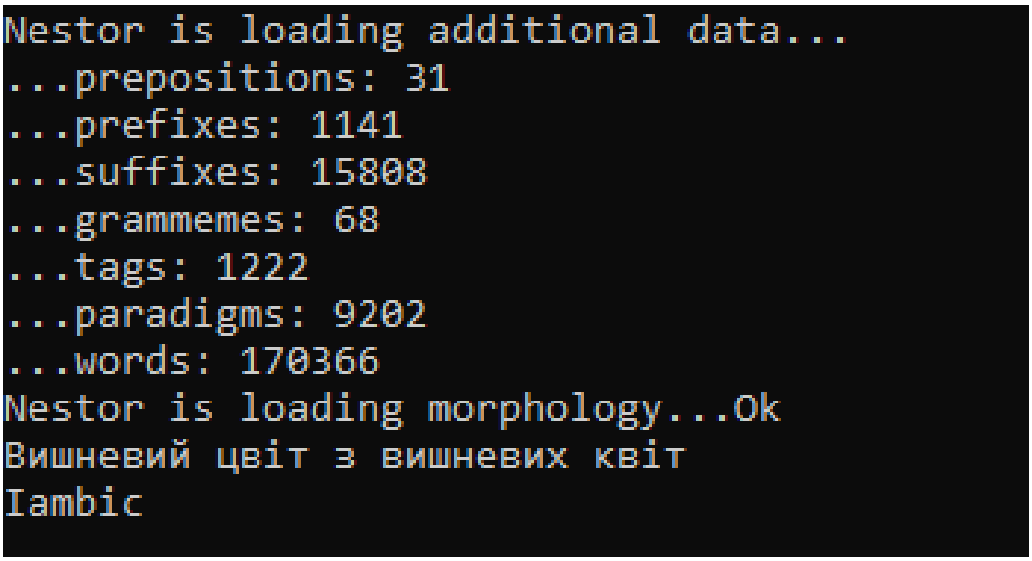
```
    {
        return Enumerable.Range(0, l).Select(x => Mask[x %
Mask.Length]).ToArray();
    }
```

Залишився лише один блок програмного коду, виведення результатів, що оформлено в окремій папці Nestor.Poetry.Tests

В цій папці знаходиться файл FootAnalyzerTests, що тестує всі віршові розміри строк, які були туди вписані, та файл, який виводить результат на консольній панелі.

3.2 Перевірка результату з оригіналом

Першим прикладом буде віршова строка «Вишневий цвіт з вишневих квіт» - є ямбом. Програмне рішення:



Визначення віршового розміру:

Вишневий цвіт з вишневих квіт

U _/ U _/ U _/ U _/

```
Село неначе погоріло
Неначе люди подуріли
Німі на панщину ідуть
І дуточок своїх ведуть.
Iambic
```

Село неначе погоріло

Неначе люди подуріли

Німі на панщину ідуть

І діточок своїх ведуть

U _/ U _/ UU /U _/ U/

U _/ U _/ UU /U _/ U/

U _/ U _/ UU /U _/

UU / U _/ U _/ U _/

Наступна віршова строка відповідає хорею – «В сотах мозку золотим прозорим»

```
Nestor is loading additional data...
...prepositions: 31
...prefixes: 1141
...suffixes: 15808
...grammemes: 68
...tags: 1222
...paradigms: 9202
...words: 170366
Nestor is loading morphology...Ok
В сотах мозку золотом прозорим
Chorea
```

Визначення віршового розміру:

В сотах мозку золотим прозорим

_ U / _ U/ U U / _ U / _ U/

```
Тихо. Зорі потопають,
В океані хмар і ночі,
Понад хвилі грім гуркоче,
По каютах скрізь дрімають.
Chorea
```

Тихо. Зорі потопають,

В океані хмар і ночі,

Понад хвилі грім гуркоче,

По каютах скрізь дрімають

_ U / _ U / U U / _ U /

U _ / _ U / _ U /

_ U / _ U / _ U / _ U /

_ U / _ U /_ U /

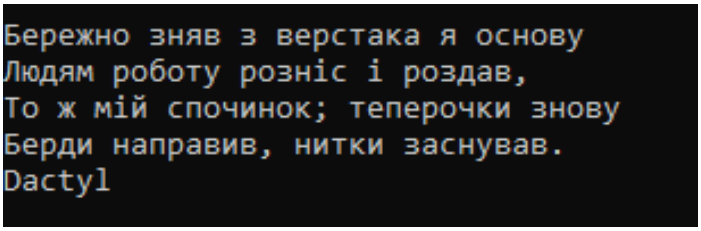
Наступний дактиль – «Бережно зняв з верстака я основу»

```
Nestor is loading additional data...
...prepositions: 31
...prefixes: 1141
...suffixes: 15808
...grammemes: 68
...tags: 1222
...paradigms: 9202
...words: 170366
Nestor is loading morphology...Ok
Бережно зняв з верстака я основу
Dactyl
```

Визначення віршового розміру:

Бережно зняв з верстака я основу

_ U U / _ U U/ _ U U/



Бережно зняв з верстака я основу

Людям роботу розніс і роздав,

То ж мій спочинок; теперочки знову

Берди направив, нитки заснував.

_ U U / _ U U/ _ U U/

_ U U/ _ U U / _ U / U _ /

_ U / _ U U / _ U U / _ U

_ U U / _ U U / _ U U/

Після нього – амфібрахій «О метана бурею, Нене злиденна»

```
Nestor is loading additional data...
...prepositions: 31
...prefixes: 1141
...suffixes: 15808
...grammemes: 68
...tags: 1222
...paradigms: 9202
...words: 170366
Nestor is loading morphology...Ok
О метана бурею, Нене злиденна
Amphibrachium
```

Визначення віршового розміру:

О метана бурею, Нене злиденна

U _ U / _ U / _ U / U _ U /

```
І все буде добре,
Для кожного з нас.
І все буде добре,
Настане наш час.
Amphibrachium
```

І все буде добре,

Для кожного з нас.

І все буде добре,

Настане наш час.

U _ _/ U _ U

U _ U/ U _

U _ _/ U _ U

U _ U/ U _

І останній «Давня ратуша в землю вростає, немов мухомор» - анапест

```
Nestor is loading additional data...
...prepositions: 31
...prefixes: 1141
...suffixes: 15808
...grammemes: 68
...tags: 1222
...paradigms: 9202
...words: 170366
Nestor is loading morphology...Ok
Давня ратуша в землю вростає, немов мухомор
Anapest
```

Визначення віршового розміру:

Давня ратуша в землю вростає, немов мухомор

U U _ / U U _ / U U _ / U _/ U U _/

```
На горі ніби снігом біліє давнезний собор,
Що скликає всі вірні серця вечорами і зранку.
Давня ратуша в землю вростає, немов мухомор,
Сивий майстер в льошку замовляє вина філіжанку
Anapest
```

На горі ніби снігом біліє давнезний собор,

Що скликає всі вірні серця вечорами і зранку.

Давня ратуша в землю вростає, немов мухомор,

Давня ратуша в землю вростає, немов мухомор,

U U _ / _ U / _ U / U _ / U U _ /

U U _ / U _ U / U _ / U U _ / _ U /

U U _ / U U _ / U U _ / U _/ U U _/

U _ / U U _ / U _ / U U _/ U _/ U U _ /

3.3 Висновки до третього розділу

У третьому розділі вищої кваліфікаційної роботи було розглянуто роботу програмного коду, наведено коментарі до кожного методу та програми вцілому, протестовано програму при визначені всіх п'яти віршових розмірів як для рядка, так і для вірша.

Було визначено та пояснено роботу програмного коду, тобто методів, змінних, класів і тощо.

Наведено приклад роботи програмного коду, були отримані результати, після яких розібрано віршові строки.

Також приведений до огляду інтерфейс програми, відображено на виході вірш чи рядок, який іде на вхід програми, та визначений віршовий розмір вірша, що подавався на вхід.

ВИСНОВКИ

У дипломній роботі було визначено і обґрунтовано теоретичні та практичні засади визначення віршового розміру для віршів української мовою.

Актуальність теми дипломної роботи пов'язана із появою нових технологій, які здатні облегшити життя та допомогти при спілкуванні з іншими людьми. Тенденції розвитку перекладачів та голосовогу вводу, з однією сторони, негативно впливає на людей, оскільки облегшують роботу мозку, проте з іншої сторони навпаки наділяють більшою інформацією пізнання та спілкування серед навколишнього середовища.

У першій частині дипломної роботи було визначено теоретичні аспекти процесу визначення віршового розміру. Було встановлено, шо є п'ять основних віршових розмірів – ям, хорей, дактиль, анапест, амфібрахій. Встановлено певні схожості та відмінності між ними, визначення кожного з них, наведені приклади україномовних текстів для кожного розміру. Схожості між собою мають ямб та хорей – це двоскладові віршові розміри. Однак дактиль, амфібрахій та анапест є трискладовими розмірами. Також встановлена актуальність визначення задачі віршового розміру в українській мові.

У другому розділі розглядався алгоритм роботи визначення віршового розміру, опис процесів за допомогою діаграм, а саме – діагарма декомпозиції, діаграма основного процесу та алгоритм програми. Алгоритм розглядає не тільки односкладні але й багатоскладні слова, визначає по різному у них кількість стоп.

У третьому розділі розглядалася сама програма, де розписано кожну рядок коду, визначено кількість методів є у програмному коді, знайдено

відповідності між прикладами з інтернету та роботою коду над віршовими строфами та рядками. Робота коду показує, що все працює добре, визначення розмірів є правильним.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Зігуля Сергій, додаток «Ритм во мне» [Електроний ресурс]/ Зігуля Сергій// Режим доступу: http://www.ritminme.ru/
2. Ілля Гузев, Бібліотеки Natasha і Spacy [Електроний ресурс]/ @einhorn// Режим доступу: https://habr.com/ru/post/575100/ , https://github.com/natasha/natasha-spacy
3. Ямб чотиристопний // Шевченківська енциклопедія: — Т. 6: Т—Я : у 6 т. / Гол. ред. М. Г. Жулинський. — Київ: Ін-т літератури ім. Т. Г. Шевченка, 2015. — С. 1094-1095.
4. Літературознавчий словник-довідник / Р. Т. Гром'як, Ю. І. Ковалів та ін. — К.: ВЦ «Академія»
5. ЗНО 2022. Українська мова та література. Частина 1 – Грамота, Авраменко О.М.
6. ЗНО 2022. Українська мова та література. Частина 2 – Грамота, Авраменко О.М.
7. Іванюк Борис, Ямби // Лексикон загального та порівняльного літературознавства//[Електроний ресурс]/ Іванюк Борис// Режим доступу: https://shron1.chtyvo.org.ua/Volkov_Anatolii/Leksykon_zahalnoho_ta_porivnialnoho_literaturoznavstva.pdf?PHPSESSID=kvkakcfkmfa8crp75dktjg1nj4
8. Галліямб // Літературознавча енциклопедія: у 2 т. / авт.-уклад. Ю. І. Ковалів. — Київ: ВЦ «Академія», 2007. — Т. 1: А — Л. — С. 211-212.
9. Хорей // Літературознавча енциклопедія: у 2 т. / авт.-уклад. Ю. І. Ковалів. — Київ: ВЦ «Академія», 2007. — Т.: М — Я. — С. 559-561.
10. Українська радянська енциклопедія: у 12 т. / гол. ред. М. П. Бажан; редкол.: О. К. Антонов та ін. — 2-ге вид. — К.: Головна редакція УРЕ, 1974–1985.

11. *Літературознавчий словник-довідник* за редакцією Р. Т. Гром'яка, Ю. І. Ковалів, В. І. Теремка. — К.: ВЦ «Академія», 2007
12. Дактиль // Літературознавча енциклопедія: у 2 т. / авт.-уклад. Ю. І. Ковалів. — Київ: ВЦ «Академія», 2007. — Т. 1: А — Л
13. Світлана Салогуб, «Віршовий розмір ямб чи хорей, як визначити» [Електроний ресурс]/СвітланаСалогуб// Режим доступу:http://promovu.info/literatura/virshovij-rozmir-yamb-chi-horej-yak-viznachiti/
14. Брахіколон // Літературознавчий словник-довідник /[Електроний ресурс]/ Режим доступу: https://shron1.chtyvo.org.ua/Hromiak_Roman/Literaturoznavchyi_slovnyk-dovidnyk.pdf?#page=92
15. Метр // Енциклопедичний словник класичних мов //[Електроний ресурс]// Режим доступу: https://shron1.chtyvo.org.ua/Zvonska_Lesia/Entsyklopedychnyi_slovnyk_klasychnykh_mov.pdf#page=322

ДОДАТОК

Визначення масок віршових розмірів

```
using System;
using System.Linq;

namespace Nestor.Poetry
{
   public record Foot(FootType Type)
   {
      public string Name => Type switch {
         FootType.Unknown => "",
         FootType.Iambic => "ямб",
         FootType.Chorea => "хорей",
         FootType.Dactyl => "дактиль",
         FootType.Amphibrachium => "амфібрахий",
         FootType.Anapest => "анапест",
         _ => throw new ArgumentOutOfRangeException()
      };

      public StressType[] Mask => Type switch {
         FootType.Unknown => new [] { StressType.StrictlyUnstressed },
         FootType.Iambic => new []{ StressType.StrictlyUnstressed,
StressType.CanBeStressed },
         FootType.Chorea => new []{ StressType.CanBeStressed,
StressType.StrictlyUnstressed },
         FootType.Dactyl => new []{ StressType.CanBeStressed,
StressType.StrictlyUnstressed, StressType.StrictlyUnstressed},
         FootType.Amphibrachium => new []{ StressType.StrictlyUnstressed,
StressType.CanBeStressed, StressType.StrictlyUnstressed },
         FootType.Anapest => new []{ StressType.StrictlyUnstressed,
StressType.StrictlyUnstressed, StressType.CanBeStressed },
         _ => throw new ArgumentOutOfRangeException()
      };

      public int StepsCount
      {
         get
         {
```

```
            switch (Type)
            {
                case FootType.Iambic:
                case FootType.Chorea:
                    return 2;
                case FootType.Dactyl:
                case FootType.Amphibrachium:
                case FootType.Anapest:
                    return 3;
                case FootType.Unknown:
                default:
                    throw new ArgumentOutOfRangeException();
            }
        }
    }

    public StressType[] GetMaskOfLength(int l)
    {
        return Enumerable.Range(0, l).Select(x => Mask[x %
Mask.Length]).ToArray();
    }
}

public enum FootType
{
    Unknown,
    Iambic,
    Chorea,
    Dactyl,
    Amphibrachium,
    Anapest
}

public enum StressType
{
    StrictlyUnstressed,
    CanBeStressed,
    StrictlyStressed,
}
                                                    }
```

Робота алгоритму програми

```
using System;
using System.Collections.Generic;
using System.Linq;
using Nestor.Models;

namespace Nestor.Poetry
{
   public class FootAnalyser
   {
      private readonly NestorMorph _nestor;
      private const int WordToMaskMismatchPenalty = 5;
      private const int MaskToWordMismatchPenalty = 2;

      public FootAnalyser(NestorMorph nestor = null)
      {
         _nestor = nestor ?? new NestorMorph();
      }

      /// <summary>
      /// Знайдіть найкращу відповідну стопу для всього вірша
      /// </summary>
      /// <param name="poem">Рядок містить рядки вірша українською мовою,
розділені символом нового рядка</param>
      /// <returns>Найкраще підібрана стопа</returns>
      public Foot FindBestFootByPoem(string poem)
      {
         IEnumerable<string> lines = poem.Split(new[] { "\r\n", "\n" },
StringSplitOptions.None)
            .Where(l => !string.IsNullOrEmpty(l.Trim()));

         var distances = new Dictionary<Foot, int>();
         foreach (string line in lines)
         {
            (Foot foot, int distance)[] result = ScoreAllFootsByLine(line);
            foreach ((Foot foot, int distance) in result)
            {
               if (!distances.ContainsKey(foot))
               {
                  distances.Add(foot, distance);
```

```
                }
                else
                {
                    distances[foot] += distance;
                }
            }
        }

        // сортувати результат і повернути його
        KeyValuePair<Foot, int> best = distances.OrderBy(d => d.Value).First();
        return best.Key;
    }

    /// <summary>
    /// Знайдіть найкращу відповідну стопу для поточної лінії
    /// </summary>
    /// <param name="line">Строка містить один рядок вірша українською мовою</param>
    /// <param name="distance">Знайдено відстань до стопи, чим менше, тим краще</param>
    /// <returns>Найкраща стопа знайдена</returns>
    public Foot FindBestFootByLine(string line, out int distance)
    {
        (Foot foot, int distance)[] allFoots = ScoreAllFootsByLine(line);
        distance = allFoots.First().distance;
        return allFoots.First().foot;
    }

    /// <summary>
    ///Оцініть усі базові фути за відстанню до поточної лінії, краще менше відстані
    /// </summary>
    /// <param name="line">Строка містить один рядок вірша</param>
    /// <returns>Масив усіх стоп із відстанями</returns>
    public (Foot foot, int distance)[] ScoreAllFootsByLine(string line)
    {
        string[] tokens = _nestor.Tokenize(line, MorphOption.RemoveHyphen);
        StressType[] lineStresses = tokens.SelectMany(GetPoeticStresses).ToArray();

        // підрахувати всі стопи foot
        var result = new List<(Foot foot, int distance)>();
        foreach (FootType footType in Enum.GetValues<FootType>())
```

```
            {
                if (footType == FootType.Unknown) continue;
                var foot = new Foot(footType);
                result.Add((foot, DistanceToFoot(foot, lineStresses)));
            }

            return result.OrderBy(r => r.distance).ToArray();
        }

        /// <summary>
        /// Обчисліть відстань між строковим наголосом та наголосом стопи
        /// </summary>
        /// <param name="foot">Поточна стопа</param>
        /// <param name="lineStresses">Поточні  наголоси строки</param>
        /// <returns>Відстань,чим менше, тим краще</returns>
        public int DistanceToFoot(Foot foot, IList<StressType> lineStresses)
        {
            StressType[] mask = foot.GetMaskOfLength(lineStresses.Count);
            var dist = 0;
            for (var i = 0; i < mask.Length; i++)
            {
                StressType lineStress = lineStresses[i];
                StressType maskStress = mask[i];

                //якщо голосний рядка може бути наголошений чи ні, це нульова відстань до маскування
                // інакше:
                if (lineStress != StressType.CanBeStressed)
                {
                    switch (lineStress)
                    {
                        // маска строго ненаголошена, коли рядковий голосний не є, це великий штраф
                        case StressType.StrictlyStressed when maskStress == StressType.StrictlyUnstressed:
                            dist += WordToMaskMismatchPenalty;
                            break;
                        //маска може бути наголошена, коли рядковий голосний у ненаголошеній, легкій штрафі
                        case StressType.StrictlyUnstressed when maskStress == StressType.CanBeStressed:
                            dist += MaskToWordMismatchPenalty;
```

```
                    break;
                }

                // коли рядковий голосний суворо наголошений, а маска може бути наголошена, штрафу немає
                // коли голосний рядковий і маска суворо ненаголошені, штрафу також немає
            }
        }

        return dist;
    }

    /// <summary>
    /// Отримайте всі можливі наголоси в словах з урахуванням можливої поетичної стопи
    /// </summary>
    /// <param name="word">Вхідне слово</param>
    /// <returns>Масив наголосів, кожен індекс - це число голосних у слові, яке починається з нуля</returns>
    public StressType[] GetPoeticStresses(string word)
    {
        int vCount = word.Count(NestorMorph.IsVowel);

        // якщо немає голосних, поверніть порожній масив
        if (vCount == 0)
        {
            return Array.Empty<StressType>();
        }

        // з одним голосним повертають одиночний нечіткий наголос
        // у поезії слово з одним складом може бути як наголошеним, так і ненаголошеним
        if (vCount == 1)
        {
            return new[] { StressType.CanBeStressed };
        }

        var knownStressedVowelNumbers = new HashSet<int>();

        Word[] wordInfos = _nestor.WordInfo(word);
        foreach (Word wordInfo in wordInfos)
```

```
            {
                WordForm[] exactForms = wordInfo.ExactForms(word);
                foreach (WordForm form in exactForms)
                {
                    if (form.Stress > 0)
                    {
                        knownStressedVowelNumbers.Add(form.Stress);
                    }
                }
            }

            // немає відомих наголосів, кожен голосний може бути наголошеним
            if (knownStressedVowelNumbers.Count == 0)
            {
                return Enumerable.Repeat(StressType.CanBeStressed, vCount).ToArray();
            }

            //
            StressType[] finalStresses =
Enumerable.Repeat(StressType.StrictlyUnstressed, vCount).ToArray();
            if (knownStressedVowelNumbers.Count == 1)
            {
                // один сильний наголос
                finalStresses[knownStressedVowelNumbers.First() - 1] =
StressType.StrictlyStressed;
            }
            else
            {
                // різні наголоси
                foreach (int knownStressNumber in knownStressedVowelNumbers)
                {
                    finalStresses[knownStressNumber - 1] = StressType.CanBeStressed;
                }
            }

            return finalStresses;
        }

    }
                                                }
```

Тестування та отримання результату програми

```
using System;
using System.Linq;
using NUnit.Framework;

namespace Nestor.Poetry.Tests
{
   [TestFixture]
   public class FootAnalyzerTests
   {


      [Test]
      public void IambicTests()
      {
         Console.OutputEncoding = System.Text.Encoding.Default;
         var analyzer = new FootAnalyser();
         const string line = "Вишневий цвіт з вишневих квіт";//Садок вишневий
коло хати, Село неначе погоріло
         Foot foot = analyzer.FindBestFootByLine(line, out _);
         Console.WriteLine(line);
         Console.WriteLine(foot.Type.ToString() + "\n");


       const string poem = "Село неначе погоріло\n" +
                           "Неначе люди подуріли\n" +
                           "Німі на панщину ідуть\n" +
                           "І діточок своїх ведуть.";
         foot = analyzer.FindBestFootByPoem(poem);

         Console.WriteLine(poem);
         Console.WriteLine(foot.Type.ToString() + "\n");

      }


    [Test]
    public void ChoreaTests()
    {
        Console.OutputEncoding = System.Text.Encoding.Default;
```

```
        var analyzer = new FootAnalyser();
      const string line = "В сотах мозку золотом прозорим";
      Foot foot = analyzer.FindBestFootByLine(line, out _);
        Console.WriteLine(line);
        Console.WriteLine(foot.Type.ToString() + "\n");

         const string poem = "Тихо. Зорі потопають,\n" +
                    "В океані хмар і ночі,\n" +
                    "Понад хвилі грім гуркоче,\n" +
                    "По каютах скрізь дрімають.";
        foot = analyzer.FindBestFootByPoem(poem);
        Console.WriteLine(poem);
        Console.WriteLine(foot.Type.ToString() + "\n");
    }

    [Test]
public void DactylTests()
    {
        Console.OutputEncoding = System.Text.Encoding.Default;
        var analyzer = new FootAnalyser();
        const string line = "Бережно зняв з верстака я основу";
   Foot foot = analyzer.FindBestFootByLine(line, out _);
        Console.WriteLine(line);
        Console.WriteLine(foot.Type.ToString() + "\n");

   const string poem = "Бережно зняв з верстака я основу\n" +
                "Людям роботу розніс і роздав,\n" +
                "То ж мій спочинок; теперочки знову\n" +
                "Берди направив, нитки заснував.";
   foot = analyzer.FindBestFootByPoem(poem);
        Console.WriteLine(poem);
        Console.WriteLine(foot.Type.ToString() + "\n");
    }

    [Test]
public void AmphibrachiumTests()
{
        Console.OutputEncoding = System.Text.Encoding.Default;
        var analyzer = new FootAnalyser();
        const string line = "О метана бурею, Нене злиденна";
   Foot foot = analyzer.FindBestFootByLine(line, out _);
        Console.WriteLine(line);
```

```
        Console.WriteLine(foot.Type.ToString() + "\n");

        const string poem = "І все буде добре,\n" +
                    "Для кожного з нас.\n" +
                    "І все буде добре,\n" +
                    "Настане наш час.";
    foot = analyzer.FindBestFootByPoem(poem);
        Console.WriteLine(poem);
        Console.WriteLine(foot.Type.ToString() + "\n");
     }

     [Test]
public void AnapestTests()
{
        Console.OutputEncoding = System.Text.Encoding.Default;
        var analyzer = new FootAnalyser();
        const string line = "Давня ратуша в землю вростає, немов мухомор";

    Foot foot = analyzer.FindBestFootByLine(line, out _);
        Console.WriteLine(line);
        Console.WriteLine(foot.Type.ToString() + "\n");

        const string poem = "На горі ніби снігом біліє давнезний собор,\n" +
                    "Що скликає всі вірні серця вечорами і зранку.\n" +
                    "Давня ратуша в землю вростає, немов мухомор,\n" +
                    "Сивий майстер в льошку замовляє вина філіжанку";
    foot = analyzer.FindBestFootByPoem(poem);
        Console.WriteLine(poem);
        Console.WriteLine(foot.Type.ToString());;
     }
     /*
[Test]
public void ExactFormsTest()
{
        var analyzer = new FootAnalyser();
        const string line = "У лукоморья дуб зеленый";
    Foot foot = analyzer.FindBestFootByLine(line, out _);
        Console.WriteLine(foot.Type.ToString());
     }*/

  }
                                        }
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Nestor.Poetry.Tests
{
   internal class Class1
   {
      static void Main(string[] args)
      {
         var analyzer = new FootAnalyzerTests();
         analyzer.IambicTests();
         analyzer.ChoreaTests();
         analyzer.DactylTests();
         analyzer.AmphibrachiumTests();
         analyzer.AnapestTests();

      }
   }
}
```