**МIНIСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Західноукраїнський національний університет**

**Факультет комп’ютерних інформаційних технологій**

Кафедра комп’ютерної інженерії

**Сергійчук Вікторія Вікторівна**

**«Алгоритм автоматичного доповнення програмного коду для інтегрованих середовищ розробки / Algorithm for automatic completion of program code for integrated development environments»**

спеціальність: 123 - Комп’ютерна інженерія

освітньо-професійна програма - Комп’ютерна інженерія

Кваліфікаційна робота

Виконав студент групи КІм-21

В.В. Сергійчук

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Науковий керівник:

к.т.н., доц. Ю.М. Батько

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кваліфікаційну роботу допущено

до захисту:

"\_\_\_" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_ р.

Завідувач кафедри

\_\_\_\_\_\_\_\_\_\_\_\_\_\_Л. О. Дубчак

**Тернопіль – 2022**

ЗМІСТ

[Вступ 7](#_Toc118655989)

[1 Апаратні cистеми створення та відлагодження програмних кодів на мовах програмування високого рівня 10](#_Toc118655990)

[1.1 Програмні файли, особливості створення, організації класифікація 10](#_Toc118655991)

[1.2 Програмний код, класифікація та основні характеристики 21](#_Toc118655992)

[1.3 Програмні засоби для написання програмних кодів 26](#_Toc118655993)

[1.4 Постановка задач дослідження 29](#_Toc118655994)

[1.5 Висновки до розділу 30](#_Toc118655995)

[2 Методи та алгоритми аналізу та обробки обробки текстів програмних кодів 31](#_Toc118655996)

[2.1 Методи та алгоритми аналізу текстової інформації 31](#_Toc118655997)

[2.2 Алгоритми предекативного введення текстів 43](#_Toc118655998)

[2.3 Алгоритми автоматичного доповнення слів в програмному коді 47](#_Toc118655999)

[2.4 Висновки до розділу 50](#_Toc118656000)

[3 Програмний додаток інтелектуалізованого введення програмних кодів 51](#_Toc118656001)

[3.1 Структура програмного додатку предекативного введення текстів 51](#_Toc118656002)

[3.2 Програмні модулі додатку інтелектуального вводу тексту 62](#_Toc118656003)

[3.3 Тестування та аналіз реалізованого програмного додатку 67](#_Toc118656004)

[3.4 Висновки до розділу 71](#_Toc118656005)

[Висновки 72](#_Toc118656006)

[Список використаної літератури 73](#_Toc118656007)

[Додаток А Лістинг програмного модуля аналізу текстів 77](#_Toc118656008)

[Додаток Б Світлокопії виданих публікацій 81](#_Toc118656009)

# Вступ

Актуальність роботи. На сьогодні помічники завершення коду на основі механізмів штучного інтелекту надають розробникам широкі пропозиції щодо фрагментів і цілеспрямовані завершення коду рядків безпосередньо в Visual Studio Code або IntelliJ Idea IDE. Подальші дослідження в даному напрямку будуть використані для додавання підтримки додаткових моделей ШІ та мов програмування.

Алгоритми предекативного введення текстів повторно використовують загальні шаблони кодування, щоб зменшити непотрібну працю розробників. На сьогодні, в індустрії програмного забезпечення вже існують прикладні варіанти використання штучного інтелекту для прискорення високошвидкісного накопичення даних, рефакторингу коду та автоматизації процесів, тому штучний інтелект швидкими темпами поширюється в IDE розробників.

Зважаючи на величезну кількість шаблонів і повторюваність у розробці програмного забезпечення, багато провідних розробників вважають, що ШІ – це майбутнє розробки. Однією з причин успіху помічників на основі штучного інтелекту є те, що життя розробника сьогодні складніше, тому що розробники повинні мати справу з безмежними інструментами замість того, щоб жити у всесвіті однієї конкретної мови.

Помічник штучного інтелекту ставить написання коду на рейки, вносячи пропозиції щодо найкращих методів написання коду та, можливо, перешкоджаючи розробнику писати код, який не дотримується найкращих практик. Додавання більш точних можливостей штучного інтелекту, які стосуються певних мов, дозволяє розробникам досліджувати новий світ можливостей.

Механіка штучного інтелекту дозволяє розробнику не виконувати повсякденніні завданя, допомагати йому в написанні коду та зосереджуватися на більш значущих завданнях, таких як аналіз, що робить роботу розробника більш значущою та цікавішою.

Таким чином для реалізації програмного додатку предекативного введення текстів програмних кодів необхідно спроектувати, промоделювати та реалізувати алгоритм інтелектуалізованого введення текстів програмних кодів на основі використання n-грам. При цьому необхідно враховувати особивості роботи програмних додатків, а саме робта в режимі реального часу, адаптивність під стиль написання програмних кодів конкретним розробником тощо. Тому задача створення алгоритму автоматичного доповлення програмного коду для інтегрованих середовищ розробки є актуальною.

Метою роботи є розробка алгоритму предекативного введення текстів програмних кодів на основі використання n-грам для інтегрованих середовищ розробки.

Для досягнення даної мети ставились наступні завдання:

* провести класифікацію програмних файлів та сфер їх використання;
* проаналізувати принципи роботи мов програмування різних класів;
* провести аналітичний огляд наявних програмних засобів для написання та відлагодження програмних кодів;
* проаналізувати існуючі алгоритми та підходи до аналізу та обробки текстових файлів;
* розробити алгоритм автоматичного доповнення програмного коду на основі семантичного аналізу тексту;
* реалізувати програмну систему написання програмного коду на мові Java з елементами автоматизованого доповнення коду семантичного аналізу тексту, провести її тестування та порівняти з подібними програмами.

Об’єкт дослідження – процес аналізу обробки текстових файлів в системах розробки.

Предмет дослідження – методи і алгоритми предикативного введення текстів.

Наукова новизна одержаних результатів визначається наступним чином:

* проведено комплексний аналіз та класифікацію алгоритмів обробки та аналізу для структурованої та природньої текстової інформації, що надало можливість підкреслити їх переваги та недоліки, а також розробити власний алгоритм предикативного введення тексту;
* розроблено предикативного введення тексту на основі використання n-грам, що дозволило зменшити обчислювальну складність процесу та персонізувати процес передбачення можливого закінчення введених частин слова.

Практична цінність одержаних результатів полягає в тому, що:

* розроблено та проведено теоретичне дослідження програмного додатку предикативного введення тексту на основі використання n-грам, що дозволило в подальшому програмно реалізувати та провести дослідження розроблених алгоритмів;
* реалізовано програмне забезпечення для предикативного введення тексту на основі використання класів для обробки стрічкових даних мови Java та з використанням алгоритмів предекативного введення текстів.

Публікації та апробація до випускної кваліфікаційної роботи. За результатами наукових досліджень, проведених у випускній кваліфікаційні роботі, підготовлено тези доповіді «Алгоритм предикативного вводу текстів на основі n‑грам» обсягом 1 сторінка на VІ Науково-практичній конференції молодих вчених і студентів «Інтелектуальні комп’ютерні системи та мережі» [1], а також «Аналіз алгоритмів предекативного вводу текстів» обсягом 1 сторінка на VІ Науково-практичній конференції молодих вчених і студентів «Інтелектуальні комп’ютерні системи та мережі» [2].

1 Апаратні cистеми створення та відлагодження програмних кодів на мовах програмування високого рівня

## Програмні файли, особливості створення, організації класифікація

Щодня ми стикаємося з різними типами комп’ютерного програмного забезпечення, яке допомагає нам виконувати наші завдання та підвищує нашу ефективність. Від MS Windows, яка вітає нас, коли вмикаємо систему, до веб-браузера, який використовується для серфінгу в Інтернеті до лічильника спалених калорій на нашому смартфоні – все це приклади програмного забезпечення. У цьому світі технологій навіть стикаємося з різними тенденціями розробки програмного забезпечення, які допомагають виконувати поставлені завдання та розвиватися, нас оточує все це програмне забезпечення, яке має намір полегшити наше життя.

Програмне забезпечення (також скорочено ПЗ) – це набір даних, програм, процедур, інструкцій і документації, які виконують різні попередньо визначені завдання в комп’ютерній системі. Вони дозволяють користувачам взаємодіяти з комп’ютером.

У галузі програмної інженерії та інформатики програмне забезпечення - це не що інше, як інформація, яка обробляється комп'ютерною системою та програмами. Програмне забезпечення включає бібліотеки, програми та відповідні невиконувані дані, такі як цифрові носії та онлайн-документація. Комп’ютерне обладнання та програмне забезпечення потребують одне одного, і жодне з них не може переконливо використовуватися окремо. Поєднання апаратного та програмного забезпечення забезпечує контроль і гнучкість сучасних обчислювальних систем. Без програмного забезпечення комп’ютери не принесуть користі. Наприклад, без допомоги програмного забезпечення веб-браузера не зможете переглядати Інтернет. Так само без операційної системи жодна програма не може працювати на комп’ютері.

Сьогодні доступна велика кількість високоякісних технологій і програмного забезпечення, які визначають спосіб нашого життя та задовольняють наші постійно змінювані та зростаючі потреби. Нескінченна кількість типів програмного забезпечення може бути надзвичайною для будь-кого, особливо якщо людина не розуміє досконально різні типи програмного забезпечення та їх користувачів.

Як правило, існує дві основні класифікації програмного забезпечення, а саме системне програмне забезпечення та прикладне програмне забезпечення, приклад класифікації наведено на рисунку 1.1.

Класи програмного забезпечення

Системне програмне забезпечення

Прикладне програмне забезпечення

* операційні системи;
* драйвери;
* мікропрограмне забезпечення;
* транслятори мов програмування;
* утиліти.
* текстові процесори;
* графічне програмне забезпечення
* мультимедійне програмне забезпечення
* навчальне та довідкове програмне забезпечення;
* веб-браузери

Рисунок 1.1 – Класифікація програмного забезпечення

Системне програмне забезпечення. Системне програмне забезпечення допомагає користувачеві та апаратному забезпеченню функціонувати та взаємодіяти одне з одним. По суті, це програмне забезпечення для керування поведінкою апаратного забезпечення комп’ютера, щоб забезпечити основні функції, необхідні користувачеві. Простими словами, можна сказати, що системне програмне забезпечення є посередником або середнім шаром між користувачем і обладнанням. Це комп’ютерне програмне забезпечення визначає платформу або середовище для роботи іншого програмного забезпечення. Ось чому системне програмне забезпечення є дуже важливим для керування всією комп’ютерною системою. Коли вперше вмикається комп’ютер, саме системне програмне забезпечення ініціалізується та завантажується в пам’ять системи. Системне програмне забезпечення працює у фоновому режимі й не використовується кінцевими користувачами. Саме тому системне програмне забезпечення також називають «програмним забезпеченням низького рівня».

Нижче наведено кілька поширених прикладів системного програмного забезпечення.

Операційна система: Це найклащий приклад системного програмного забезпечення. Це набір програмного забезпечення, яке обробляє ресурси та надає загальні послуги для інших програм, які над ними працюють. Хоча кожна операційна система відрізняється, більшість із них забезпечують графічний інтерфейс користувача, за допомогою якого користувач може керувати файлами та папками та виконувати інші завдання. Кожен пристрій, будь то настільний комп’ютер, ноутбук чи мобільний телефон, потребує операційної системи для забезпечення базових функцій. Оскільки ОС по суті визначає, як користувач взаємодіє з системою, тому багато користувачів вважають за краще використовувати одну конкретну ОС для свого пристрою. Існують різні типи операційних систем, наприклад операційна система реального часу, вбудована, розподілена, багатокористувацька, однокористувацька, Інтернет, мобільна та багато інших. Перед вибором операційної системи важливо врахувати апаратні характеристики.

* Android;
* MS Windows;
* Ubuntu;
* Unix.

Драйвери пристроїв: це тип програмного забезпечення, яке керує певним обладнанням, підключеним до системи. Апаратні пристрої, яким потрібен драйвер для підключення до системи, включають дисплеї, звукові карти, принтери, миші та жорсткі диски. Крім того, існує два типи драйверів пристроїв: драйвери пристроїв ядра та драйвери пристроїв користувача. Деякі приклади драйверів пристроїв:

* Драйвер BIOS;
* Драйвери дисплея;
* Драйвери материнської плати;
* Драйвери ПЗУ;
* Драйвери USB;
* Драйвери віртуальних пристроїв.

Мікропрограмне забезпечення: мікропрограмне забезпечення – це постійне програмне забезпечення, вбудоване в постійну пам’ять. Це набір інструкцій, які постійно зберігаються на апаратному пристрої. Він надає важливу інформацію про те, як пристрій взаємодіє з іншим обладнанням. Мікропрограму можна вважати «напівпостійною», оскільки вона залишається постійною, якщо її не оновити за допомогою програми оновлення мікропрограми. Деякі приклади прошивки:

* BIOS;
* Комп'ютерна периферія;
* Вбудовані системи;
* UEFI.

Транслятори мов програмування: це програми-посередники, на які програми покладаються для перекладу мовного коду високого рівня на простіший код машинного рівня. Окрім спрощення коду, перекладачі також роблять наступне:

* Призначити зберігання даних;
* Залучіть вихідний код, а також деталі програми;
* Пропонуйте діагностичні звіти;
* Виправлення системних помилок під час виконання;
* Прикладами трансляторів мов програмування є інтерпретатор, компілятор і асемблери.

Утиліти: Службове програмне забезпечення призначене для допомоги в аналізі, оптимізації, налаштуванні та обслуговуванні комп’ютерної системи. Він підтримує комп'ютерну інфраструктуру. Це програмне забезпечення зосереджується на тому, як функціонує ОС, а потім відповідно вирішує свою траєкторію, щоб згладити функціонування системи. Програмне забезпечення, як-от антивіруси, інструменти очищення та керування дисками, інструменти стиснення, дефрагментатори тощо, є допоміжними інструментами. Деякі приклади допоміжних інструментів:

* Антивірус McAfee;
* WinRAR;
* WinZip.

2. Прикладне програмне забезпечення. Програмне забезпечення, також відоме як програми кінцевого користувача або програми підвищення продуктивності – це програмне забезпечення, яке допомагає користувачеві виконувати такі завдання, як дослідження в Інтернеті, нотатки, встановлення будильника, створення графіки, ведення журналу облікового запису, виконання обчислень або навіть гра в ігри. Вони лежать над системним програмним забезпеченням. На відміну від системного програмного забезпечення, вони використовуються кінцевим користувачем і мають специфічні функції чи завдання та виконують роботу, для якої вони призначені. Наприклад, браузер – це програма, розроблена спеціально для перегляду веб-сторінок, або MS Powerpoint – це програма, яка використовується спеціально для створення презентацій. Прикладне програмне забезпечення або просто програми також можна віднести до неосновного програмного забезпечення, оскільки вимоги до них дуже суб’єктивні, а їх відсутність не впливає на функціонування системи. Усі програми, які ми бачимо на наших мобільних телефонах, також є прикладами програмного забезпечення. Є певне програмне забезпечення, створене виключно длярозробка програм, таких як Meteor і Flutter. Це також приклади програмного забезпечення. Існують різні типи прикладного програмного забезпечення:

Текстові процесори: ці програми для документації. Крім того, це також допомагає мені зберігати, форматувати та друкувати ці документи. Деякі приклади текстових процесорів:

* Сторінки Apple iWork;
* Документи Google;
* MS Word.

Програмне забезпечення бази даних: це програмне забезпечення використовується для створення та керування базою даних. Вона також відома як система керування базами даних або СУБД. Вони допомагають з організацією даних. Деякі приклади СУБД:

* dBase;
* MS Access;
* MySQL.

Мультимедійне програмне забезпечення: це програмне забезпечення, яке може відтворювати, створювати або записувати зображення, аудіо- чи відеофайли. Вони використовуються для редагування відео, анімації, графіки та редагування зображень. Деякі приклади мультимедійного програмного забезпечення:

* Adobe Photoshop;
* Picasa;
* Windows Movie Maker.

Навчальне та довідкове програмне забезпечення: ці типи програмного забезпечення спеціально розроблені для полегшення вивчення певної теми. Існують різні типи навчального програмного забезпечення, які підпадають під цю категорію. Їх також називають академічним програмним забезпеченням. Деякі приклади:

* KidPix;
* MindPlay;
* Tux Paint.

Графічне програмне забезпечення: як випливає з назви, графічне програмне забезпечення було розроблено для роботи з графікою, оскільки воно допомагає користувачеві редагувати або вносити зміни у візуальні дані чи зображення. Він складається з редакторів зображень та програмного забезпечення для ілюстрації. Деякі приклади:

* Adobe Photoshop;
* Corel Draw;
* GIMP;.

Веб-браузери: ці програми використовуються для перегляду веб-сторінок. Вони допомагають користувачеві знаходити та отримувати дані в Інтернеті. Деякі приклади веб-браузерів:

* Google chrome;
* Internet Explorer;
* Mozilla Firefox.

Окрім цього, усе програмне забезпечення, яке служить певній меті, підпадає під категорію прикладного програмного забезпечення.

Однак існує ще одна класифікація програмного забезпечення. Програмне забезпечення також можна класифікувати на основі їх доступності та можливості спільного використання. Ця класифікація наведена на рисунку 1.2:

Програмне забезпечення

Безкоштовне програмне забезпечення (Freeware software)

Умовно-безкоштовні програми (shareware)

Відкритий код (open-source)

Платне програмне забезпечення (software)

Рисунок 1.2 – Класифікація за критерієм доступу до ПЗ

1. Безкоштовне програмне забезпечення (Freeware software). Безкоштовне програмне забезпечення доступне безкоштовно. Будь-який користувач може завантажити його з Інтернету та використовувати без сплати жодної комісії. Однак безкоштовне програмне забезпечення не надає жодної свободи для модифікації програмного забезпечення або стягнення плати за його розповсюдження. Приклади:

* Adobe Reader;
* Skype;
* Yahoo Messenger.

2. Умовно-безкоштовні програми (shareware). Це програмне забезпечення, яке безкоштовно розповсюджується серед користувачів на пробній основі. Зазвичай він має обмеження за часом, і коли цей ліміт закінчується, користувача просять оплатити продовження послуг. Існують різні типи умовно-безкоштовних програм, як-от Adware, Donationware, Nagware, Freemium і Demoware (Crippleware і пробна версія). Деякі приклади умовно-безкоштовного програмного забезпечення:

* Adobe Acrobat;
* Відладчик PHP;
* Winzip.

3. Відкритий код (open-source). Ці види програмного забезпечення доступні користувачам із вихідним кодом, що означає, що користувач може вільно поширювати та змінювати програмне забезпечення та додавати додаткові функції до програмного забезпечення. Програмне забезпечення з відкритим кодом може бути безкоштовним або платним. Деякі приклади програмного забезпечення з відкритим кодом:

* Веб-сервер Apache;
* Moodle;
* Thunderbird.

4. Пладне програмне забезпечення (software). Вони також відомі як програмне забезпечення із закритим кодом. Цей типи програм зазвичай є платними та мають права інтелектуальної власності або патенти на вихідний код. Їх використання дуже обмежене, і зазвичай вихідний код зберігається та зберігається в секреті.

Коли мова йде про ефективне керування файлами, система функціональної класифікації є найефективнішим шляхом. Незалежно від того, чи потрібно створити систему функціональної класифікації з нуля чи просто налаштувати існуючу систему. Аналіз основних принципів класифікації файлів разом із практичними стратегіями розробки та впровадження правильної системи.

Для ефективного ведення роботи потрібен швидкий і легкий доступ до відповідної інформації та документальних підтверджень. Кожна організація потребує системи, яка розміщує записи по групах або категоріях і включає угоди про найменування та коди файлів для їх опису.

Система застосовується до записів у формі:

* Фізична система файлів;
* Електронні папки та підпапки;
* Електронні системи індексації та пошуку.

Професійні адміністратори записів багато говорять про «функціональну класифікацію документів» і не дарма. Кожного дня оформляємо документи для підтримки бізнес-функцій і щоденної діяльності. Пізніше, якщо відбудеться перевірка використовуються файли як докази для захисту цієї діяльності. Має сенс включити ті самі функції та дії в саму систему файлів.

Найпоширеніший спосіб включення функцій і дій у систему файлів – це почати з функції та рухатися вниз. Найвищий рівень ієрархії файлів повинен складатися з широких загальноорганізаційних функцій, таких як «Управління фінансами» або «Захист навколишнього середовища».

Кожна функція потім розбивається на більш конкретні дії, які визначають основні колекції або інші групи, які заповнюють вашу кімнату файлів або електронний репозиторій.

Суть системи класифікації полягає в тому, щоб розділити файли на категорії, що полегшує їх пошук. Якщо категорія містить таку кількість файлів, що важко визначити конкретний файл, потрібно додатково розділити ці файли на підкатегорії.

Подібно до того, як забагато підкатегорій може зробити систему реєстрації менш ефективною, непослідовне визначення цих категорій може зробити її повністю непридатною для використання.

Кожна діяльність і пов’язане з нею зібрання файлів підпорядковуватимуться ряду юридичних і операційних вимог, які визначають, коли файли закриваються та скільки часу вони повинні зберігатися після закриття.

Нижче наведено кілька основних кроків, яких слід виконати для створення та впровадження зручної для зберігання системи файлів:

* Для кожної бізнес-діяльності, включеної в ієрархію файлів, визначте всі правові та операційні вимоги, що застосовуються до записів цієї діяльності.
* Визначаючи підкатегорії (і можливі підкатегорії) для кожної діяльності, обов’язково включіть один рівень підрозділу, який дозволить групувати файли на основі року, у якому файл закрито.
* Розгляньте можливість створення окремої області зберігання для закритих або неактивних файлів.

Протягом багатьох років галузь управління записами розробила низку систем кодування, але загалом усі вони поділяються на один із трьох типів:

* Алфавітний – повністю складається з букв;
* Числовий – повністю складається з чисел;
* Буквено-цифровий – комбінація букв і цифр.

Коди дозволяють розмістити більше інформації на якомога меншому просторі та можуть бути дуже корисними під час створення мітки файлу чи назви електронних файлів.

Системи файлів поєднаннуються зі структурованою системою кодування, щоб реалізувати такі переваги:

* Допомагає користувачам швидше та легше знаходити файл.
* Полегшує виявлення помилок у файлі.
* Допомагає користувачам читати файли на відстані.
* Полегшує впорядкування файлів за цифровими або буквено-цифровими ідентифікаторами.

Подібно до багатьох завдань системного адміністратора, класифікація файлів починається, коли адміністратори визначають дані – де зберігаються дані, яку форму вони приймають і як їх класифікувати. Такий проект, як правило, виходить за межі компетенції ІТ-відділу та потребує введення щоденних користувачів. Щойно організації розуміють класифікацію даних, вони можуть створити дизайн для властивостей файлів і реалізацію за допомогою правил класифікації.

Багато організацій використовують диспетчер ресурсів файлового сервера (FSRM) для реалізації схем класифікації. Скласти графік застосування правил класифікації як форми поточного технічного обслуговування. Багатьом організаціям важко визначити конкретні типи даних і автоматично керувати ними. Класифікація даних допомагає ідентифікувати файли, які містять такі дані, і автоматизує керування на основі серії типів класифікації та відповідних правил.

Класифікація файлів FSRM дозволяє користувачам вказувати атрибути, крім автора, власника, останнього збереження тощо. Створюючи класифікації, адміністратори надають своїм організаціям більше гнучкості для автоматизації керування файлами та ідентифікації конкретних даних у збережених файлах і папках. Наприклад, організація може захотіти ідентифікувати та керувати ідентифікаційною інформацією, такою як номери соціального страхування та кредитних карток, або позначати файли значеннями, наприклад «Конфіденційно».

Додайте роль FSRM до Windows Server, щоб використовувати класифікацію файлів. Після встановлення виберіть утиліту в меню «Інструменти» в диспетчері сервера. У цьому підручнику описано, як класифікувати дані за допомогою FSRM.

## Програмний код, класифікація та основні характеристики

Оскільки комп’ютер, автоматизація та робототехніка все більше залучаються до нашого повсякденного життя, програмування стає дуже необхідним для контролю над усіма ними. Щоб контролювати все це в системах і машинах та отримати бажаний результат за допомогою кваліфікованих мов програмування необхідні засоби які будуть ними керувати. Однак область мови програмування стає настільки широкою, як тільки це підпадатиме під одну з двох категорій мов програмування (тобто низькорівневамова та мова високого рівня). На початку розвитку комп’ютерів мовний дизайн був сильно залежить від рішення використовувати компіляцію чи інтерпретацію як спосіб виконання.

Він залежав від таких інструментів, як компіляція та інтерпретація, щоб отримати наш письмовий код у форму, яку може виконати комп’ютер. Код може бути виконаний у власному вигляді операційною системою після її перетворення на машинний код (через компіляцію) або може бути інтерпретований рядок за рядком за допомогою іншої програми, яка замість цього обробляє виконання коду самої операційної системи (через інтерпретацію).

Мови програмування в основному класифікуються на дві основні категорії – мови низького рівня та мови високого рівня. Кожна мова програмування належить до однієї з ці категорії та підкатегорії.

Мови низького рівня. Мови низького рівня використовуються для написання програм, які відносяться до конкретної архітектури та апаратного забезпечення певного типу комп’ютерів. Вони ближчі до рідної мови комп’ютера (машинні коди, двійкове кодування), що ускладнює їх розуміння програмістами.

Програми, написані на мовах низького рівня, швидкі та ефективно використовують пам'ять. Однак, це вимага є від програміста високогорівня підготовки для того щоб писати, налагоджувати та підтримувати низькорівневі програми. Вони в основному використовуються для розробки операційних систем, драйверів пристроїв, баз даних і програм, які вимагають прямого доступу до апаратного забезпечення.

Мови низького рівня також поділяються на дві категорії – машинна мова та мова асемблера.

Машинна мова: машинна мова є найближчою мовою до апаратного забезпечення. Вона складається з набору інструкцій, які виконуються безпосередньо комп’ютером. Ці інструкції є послідовністю двійкових бітів. Кожна інструкція виконує дуже конкретне і невелике завдання. Інструкції, написані машинною мовою, залежать від машини та відрізняються від комп’ютера до комп’ютера.

Мова асемблера: мова асемблера є вдосконаленням машинної мови. Подібно до машинної мови, мова асемблера також безпосередньо взаємодіє з апаратним забезпеченням. Замість використання необробленої двійкової послідовності для представлення набору інструкцій мова асемблера використовує мнемоніку. У мові асемблера використовується спеціальна програма під назвою асемблер. Асемблер перетворює мнемоніку в певний машинний код.

Переваги мов низького рівня:

* Програми, розроблені з використанням мов низького рівня, є швидкими та ефективними для пам’яті.
* Програмісти можуть краще використовувати процесор і пам’ять за допомогою мови низького рівня.
* Немає потреби в будь-якому компіляторі або інтерпретаторі для перекладу вихідного коду в машинний код. Таким чином, скорочується час компіляції та інтерпретації.
* Забезпечують пряме маніпулювання терними регістрами та пам’яттю.
* Вона може безпосередньо спілкуватися з апаратними пристроями.

Недоліки мов низького рівня:

* Програми, розроблені з використанням мов низького рівня, залежать від машини і не є переносними.
* Важко розробляти, налагоджувати та підтримувати.
* Програми низького рівня більш схильні до помилок.
* Програміст повинен мати додаткові знання про комп'ютерну архітектуру конкретної машини для програмування на мові низького рівня.

Мови високого рівня. Мови високого рівня схожі на людську мову. Мови високого рівня є зручними для програмістів, прості у кодуванні, налагодженні та обслуговуванні. це забезпечує вищий рівень абстракція від машинної мови. Вони не взаємодіють безпосередньо з обладнанням. Швидше, вони більше зосереджені на складних арифметичних операціях, оптимальній ефективності програми і легкість у кодуванні. Програми на мові високого рівня використовують оператори написані англійською мовою (такі як Python, Java, C++ тощо). Вимагають програми високого рівня компілятори/інтерпретатори для перекладу вихідного коду на машинну мову. Розробник може скомпілювати вихідний код, написаний мовою високого рівня, на декілька машинних мов. Таким чином, вони є машинно незалежною мовами. Мови високого рівня згруповані в дві категорії на основі моделі виконання – скомпільовані або інтерпретовані мови.

Також можна класифікувати мову високого рівня за іншими категоріями на основі парадигма програмування.

Структуроване програмування (іноді відоме як модульне програмування) – це a парадигма програмування, спрямована на підвищення чіткості, якості та часу розробки комп’ютерної програми, широко використовуючи конструкції структурованого потоку керування вибір (if/then/else) і повторення (while і for), блокові структури та підпрограми. Отже, це робить код більш ефективним і легшим для розуміння та модифікації. Структуроване програмування часто використовує низхідну модель проектування, за якою розробники планують загальну структуру програми на окремі підрозділи. Зауважте, це можна зробити структурованим програмування будь-якою мовою програмування.

Процедурне програмування є парадигмою програмування, похідною від структурованого програмування, засноване на концепції виклику процедури. Процедури, також відомі як підпрограми, підпрограми або функції, просто містять серію обчислювальних кроків які необхідно здійснити. Будь-яка процедура може бути викликана в будь-який момент під час виконання програми, в тому числі іншими процедурами або самостійно.

Об’єктно-орієнтоване програмування — це парадигма програмування, заснована на концепції "об'єкти", які можуть містити дані у формі полів, часто відомих як атрибути; і код у формі процедур, часто відомих як методи. Особливістю об'єктів є те, що процедури об'єкта можуть отримувати доступ і часто змінювати поля даних об'єкта, з яким вони пов'язані. Таким чином, програмісти визначають не тільки тип даних структури даних, але також типи операцій (функцій), які можна застосувати до структури даних. Таким чином, структура даних стає об'єктом, який включає як дані, так і функції. На додачу, програмісти можуть створювати зв’язки між одним об’єктом та іншим.

Переваги мови високого рівня:

* Мови високого рівня зручні для програміста. Їх легко писати, налагоджувати та підтримувати.
* Це забезпечує вищий рівень абстракції від машинних мов.
* Це машинно незалежна мова.
* Менш схильні до помилок, їх легко знайти та виправити.
* Високорівневе програмування забезпечує кращу продуктивність програмування.

Недоліки мови високого рівня:

* Для перекладу вихідного коду в машинний код потрібен додатковий час перекладу.
* Програми високого рівня повільніші за програми низького рівня.
* Порівняно з низькорівневими програмами, вони, як правило, менш ефективні для пам’яті.
* Неможливо зв’язатися безпосередньо з обладнанням.

Загальна класифікація, приклади та особливості різних мов програмування наведено в таблиці 1.1

Таблиця 1.1 – Порівняння особливостей мов програмування

|  |  |  |  |
| --- | --- | --- | --- |
| Характеристика | Мови програмування | | |
| Клас мов прогармування | Мови низького рівня | Мови високого рівня |
| Приклади мов програмування | Асемблер, машинна мова | С++, Java, Python, Delphi |
| Швидкодія | Вони швидші за мову високого рівня | Вони порівняно повільніші |
| Робота з пам’яттю | Мови низького рівня ефективно використовують пам'ять | Мови високого рівня не є ефективними для пам'яті |
| Швидкість опанування | Мови низького рівня важко вивчити | Мови високого рівня легко вивчати |
| Додаткові знання архітектури апаратних засобів | Програмування на низькому рівні вимагає знання архітектури комп'ютера | Програмування на високому рівні не потребує додаткові знання комп'ютера |
| Залежність від типу апаратного забезпечення | Вони залежать від машини і не можуть переноситись на інші системи без додаткової модифікації. | Вони незалежні від машини та портативні |
| Відлагодження програмного коду | Налагодження та обслуговування складні | Налагодження та обслуговування є порівняно простішим |
| Робота з помилками | Вони більш схильні до помилок | Вони менш схильні до помилок |
| Сфери застосування | Використовуються для розробки системного програмного забезпечення (операційні системи) і вбудовані програми | Використовують для розробки різноманітних програми, такі як – прикладні програми, веб-сайти, мобільне програмне забезпечення тощо |

## Програмні засоби для написання програмних кодів

Інтегроване середовище розробки (IDE) – це програма, яка забезпечує середовище програмування для оптимізації розробки та налагодження програмного забезпечення. Замість виконання всіх кроків, необхідних для створення виконуваної програми як непов’язаних окремих завдань, він об’єднує всі необхідні інструменти в одну програму та робочу область. Кожен із інструментів орієнтований на навколишнє середовище, і вони працюють разом, щоб представити бездоганний набір для розробки для розробника.

Навіть простий пошук IDE дасть чимало варіантів. IDE доступні в спільнотах із відкритим кодом, постачальниках і компаніях, що займаються програмним забезпеченням. Вони варіюються від безкоштовних до цін залежно від кількості необхідних ліцензій. Немає стандарту для IDE, і кожна з них має власні можливості, а також сильні та слабкі сторони. Загалом IDE забезпечує простий у використанні інтерфейс, автоматизує етапи розробки та дозволяє розробникам запускати та налагоджувати програми з одного екрана. Він також може забезпечувати зв’язок від операційної системи розробки до цільової платформи додатків, як-от робоче середовище, смартфон або мікропроцесор.

IntelliJ IDEA - це рішення для розробки додатків, яке допомагає компаніям проводити аналіз коду, прогнозувати повторювані завдання, керувати процесами пошуку, контролювати версії тощо з єдиної платформи. Це дозволяє співробітникам виявляти повторювані фрагменти коду, керувати міжмовним рефакторингом і використовувати комбінації клавіш для виявлення та виправлення помилок.

IntelliJ IDEA дозволяє членам команди використовувати інтегроване середовище розробки (IDE) для автоматичного виділення змінних змін і керування операціями налагодження коду. Завдяки функції запуску тестів і покриття співробітники можуть виконувати модульне тестування на кількох платформах, таких як Cucumber, ScalaTest, Karma, Spock і JUnit. Крім того, розробники можуть отримати доступ до інструменту допомоги при кодуванні для виконання запитів, підключення до активних баз даних, експорту даних, керування схемами тощо на централізованій платформі.

IntelliJ IDEA дозволяє організаціям інтегрувати платформу з декількома зовнішніми інструментами збірки, включаючи Gant, NPM, Webpack, Gradle, Maven і Gulp. Він доступний за місячною та річною передплатою, а підтримка надається через відповіді на поширені запитання, телефоном, електронною поштою та іншими засобами онлайн.

IntelliJ IDEA пропонує 90-денну безкоштовну пробну версію для нових користувачів, після чого програмне забезпечення доступне в 2 цінових рівнях. Присутня безкоштована версія для навчання.

Visual Studio – це рішення для розробки додатків, яке допомагає компаніям створювати програми та ігри для Інтернету, мобільних пристроїв, Mac і Windows. Це дозволяє розробникам оптимізувати процеси, пов’язані з налагодженням, тестуванням і розгортанням на централізованій платформі.

Visual Studio постачається з вбудованими інструментами тестування та шаблонами проектів, які дозволяють компаніям писати, виконувати та налагоджувати модульні тести кількома мовами та тестовими фреймворками. Це дозволяє співробітникам орієнтуватися в кодах, виявляти помилки та застосовувати виправлення коду. Адміністратори можуть використовувати модуль прямого доступу для налаштування сеансів спільного редагування коду та налагодження з правами доступу та спеціальними налаштуваннями редактора, щоб забезпечити дотримання попередньо визначених інструкцій із кодування. Крім того, він надає кілька комбінацій клавіш для таких операцій, як пошук, навігація, редагування, рефакторинг і тестування.

Visual Studio полегшує інтеграцію з кількома сторонніми рішеннями, такими як ReSharper, CodeMaid і Github Extension. Ціни доступні за запитом, а підтримка надається через форум спільноти, поширені запитання, документацію, чат та інші засоби онлайн.

Eclipse – це безкоштовне інтегроване середовище розробки з відкритим кодом, яке дозволяє користувачам створювати програми на основі Java. Eclipse IDE складається з плагінів і може бути налаштований за допомогою сторонніх плагінів. Завдяки розширюваній структурі плагінів Eclipse може підтримувати інші мови програмування, такі як Fortran, Haskell, C, C++, JavaScript, Lua, PHP, Python, R, Ruby, Scala тощо.

Claris FileMaker. FileMaker – це рішення для розробки програм, яке дозволяє користувачам створювати та розгортати програми. Користувачі можуть створювати програми в мобільних, хмарних і локальних середовищах, і для цього не потрібні навички програмування.

FileMaker надає вбудовані шаблони для створення програм для керування контактами, вмістом, інвентаризацією та керування завданнями. Рішення можна розгорнути локально або в хмарі.

FileMaker дозволяє користувачам налаштовувати свої програми за допомогою фірмових брендів, логотипів і кольорів. Він надає інструменти звітності та діаграм для створення зведених звітів, діаграм, графіків і інформаційних панелей. Користувачі можуть експортувати дані з інформаційної панелі у форматі Excel і PDF.

Рішення допомагає користувачам створювати двосторонні з’єднання з джерелами даних Microsoft SQL Server, Oracle, MySQL, PostgreSQL і IBM DB2. Він забезпечує 256-бітне шифрування для захисту даних, що зберігаються на сервері. Підтримка надається через онлайн-форум і по телефону

Платформа Salesforce — це рішення корпоративної платформи як послуги (PaaS), яке дозволяє розробникам створювати, тестувати та виправляти дефекти в хмарних програмах перед розгортанням. Платформа надає інструменти та послуги для автоматизації бізнес-процесів, інтеграції із зовнішніми програмами та надання адаптивного макета для користувачів.

Користувачі можуть запускати та керувати програмами, написаними на відкритих мовах, таких як Ruby, Java і PHP, за допомогою Salesforce і Heroku. Heroku Enterprise забезпечує функціональність співпраці, підтримку корпоративного рівня та функції контролю для користувачів.

Salesforce Lightning Design System надає розробникам посібники та код для створення програм. Користувачі можуть створювати програми за допомогою багаторазових будівельних блоків, включаючи карти, календарі, кнопки та форми для введення чисел. Платформа дозволяє користувачам створювати корпоративні програми шляхом перетягування компонентів у Lightning App Builder. Користувачі також можуть змінювати зв’язки між об’єктами даних за допомогою конструктора схем.

Користувачі можуть підключати свої програми до зовнішніх джерел даних за допомогою Lighting Connect, а підтримка надається за допомогою відеоуроків, онлайн-форуму та по телефону.

Visual LANSA — це інтегроване середовище розробки (IDE), яке дозволяє користувачам розробляти та підтримувати хмарні та настільні програми. Основні функції включають майстри створення програм, підтримку SQL, моделювання даних та інтегровані компілятори. Варіанти розгортання включають Windows, IBM i та Linux, а також хмарне розгортання.

Користувачі можуть створювати міжплатформне програмне забезпечення за допомогою RDML (мова швидкої розробки та підтримки) LANSA та незалежного сховища метаданих. Використовуючи портативне сховище метаданих LANSA, користувачі можуть створити програму з доступом до JDBC, IBM i та ODBC-сумісних баз даних.

## 1.4 Постановка задач дослідження

Під час проведених досліджень в даному розділі було наведено результати аналізу задач створення та застосування програмних файлів різних класів, приклади та сфери їх використання. Досліджено принципи створення та відлагодження програмних кодів, а також проведено їх класифікацію. Досліджено програмні засоби для написання програмних кодів з елементами автоматизації процесу кодування, виділено їх основні функції.

Для досягнення поставленої мети необхідно розв’язати наступні задачі.

* провести класифікацію програмних файлів та сфер їх використання;
* проаналізувати принципи роботи мов програмування різних класів;
* провести аналітичний огляд наявних програмних засобів для написання та відлагодження програмних кодів;
* проаналізувати існуючі алгоритми та підходи до аналізу та обробки текстових файлів;
* розробити алгоритм автоматичного доповнення програмного коду на основі семантичного аналізу тексту;
* реалізувати програмну систему написання програмного коду на мові Java з елементами автоматизованого доповнення коду семантичного аналізу тексту, провести її тестування та порівняти з подібними програмами.

## 1.5 Висновки до розділу

Проведено аналіз та класифікацію існуючих груп програмних файлів, що надало можливість окреслити основні завдання та особливості внутрішньої архітектури виконуваних програмних файлів в різних операційних системах.

Проаналізовано аналіз та класивікацію характеристик поширених мов програмування, що дозволило виділити основні структурні елементи та особливості написання програмних кодів на мовах високого рівня.

Проведено аналіз наявних програмних засобів для написання та вілагодження програмних кодів який дозволив виділити основні структурні модулі та встановити інтерфейси обміну даними між окремими модулями.

# 2 Методи та алгоритми аналізу та обробки обробки текстів програмних кодів

## 2.1 Методи та алгоритми аналізу текстової інформації

Текст може бути надзвичайно багатим джерелом інформації, але вилучення з нього розуміння може бути складним і трудомістким через його неструктуровану природу.

Але завдяки прогресу в обробці природної мови та машинному навчанні, які обидва підпадають під широку математичну основу штучного інтелекту, сортувати текстові дані стає легше.

Він працює шляхом автоматичного аналізу та структурування тексту, швидко й економічно ефективно, тож підприємства можуть автоматизувати процеси й отримувати інформацію, що веде до кращого прийняття рішень.

Вилучення ознак тексту та попередня обробка для алгоритмів класифікації є дуже важливими. У обробці природної мови (NLP) більша частина тексту та документів містить багато слів, які є зайвими для класифікації тексту, наприклад, стоп-слова, орфографічні помилки, сленг тощо. У багатьох алгоритмах, таких як статистичні та імовірнісні методи навчання, шум і непотрібні функції можуть негативно вплинути на загальну продуктивність. Тому усунення цих ознак є надзвичайно важливим.

Обробка природної мови (NLP) є підсферою штучного інтелекту (AI). Це допомагає машинам обробляти та розуміти людську мову, щоб вони могли автоматично виконувати повторювані завдання. Приклади включають машинний переклад, узагальнення, класифікацію квитків і перевірку орфографії.

Візьмемо, наприклад, аналіз настроїв, який використовує обробку природної мови для виявлення емоцій у тексті. Це завдання класифікації є одним із найпопулярніших завдань NLP, яке часто використовується компаніями для автоматичного виявлення настроїв бренду в соціальних мережах.

Однією з головних причин, чому обробка природної мови є настільки важливою, є те, що її можна використовувати для аналізу великих обсягів текстових даних, як-от коментарів у соціальних мережах, запитів у службу підтримки клієнтів, онлайн-оглядів, новин тощо. Інструменти НЛП обробляють дані в режимі реального часу, цілодобово та без вихідних, і застосовують однакові критерії до всіх ваших даних, тож ви можете гарантувати, що результати, які ви отримуєте, точні – і не містять суперечностей.

Коли інструменти NLP зможуть зрозуміти, про що йдеться в тексті, і навіть виміряти такі речі, як настрої, підприємства зможуть розставляти пріоритети та організовувати свої дані відповідно до своїх потреб.

Незважаючи на те, що в обробці природної мови виникає багато проблем, переваги NLP є величезні, що робить NLP відмінним механізмом для допомоги при аналізі та обробці текстової інформації.

Людська мова складна, неоднозначна, неорганізована та різноманітна. У світі існує понад 6500 мов, і всі вони мають свої синтаксичні та семантичні правила.

Отже, щоб машини могли розуміти природну мову, її спочатку потрібно перетворити на щось, що вони можуть інтерпретувати.

У NLP синтаксис і семантичний аналіз є ключовими для розуміння граматичної структури тексту та визначення того, як слова співвідносяться одне з одним у певному контексті. Але перетворення тексту на щось, що може обробити машина, є складним.

Науковці повинні навчити інструменти NLP виходити за межі визначень і порядку слів, розуміти контекст, неоднозначність слів та інші складні поняття, пов’язані з людською мовою.

Під час обробки природної мови людська мова поділяється на фрагменти, щоб граматичну структуру речень і значення слів можна було проаналізувати й зрозуміти в контексті. Це допомагає комп’ютерам читати та розуміти усний або письмовий текст так само, як люди.

Ось кілька фундаментальних завдань NLP попередньої обробки даних, які науковці мають виконати, перш ніж інструменти NLP зможуть зрозуміти людську мову (рисунок 2.1):

Задачі попередньої обробки текстової інформації

Токенізація

Теги частин мови

Створення коренів і лемматизація

Видалення не інформативних

Рисунок 2.1 – Класифікація за критерієм доступу до ПЗ

Розглянемо більш детально кожну з задач попередньої обробки такстів:

* **Токенізація:** розбиває текст на менші семантичні одиниці або окремі речення;
* **Теги частин мови**: позначення слів як іменників, дієслів, прикметників, прислівників, займенників тощо;
* **Створення коренів і лемматизація:** стандартизація слів шляхом зведення їх до кореневих форм;
* **Видалення не інформативних слів**: відфільтрування загальних слів, які додають мало унікальної інформації або взагалі не додають її, наприклад, прийменники. Тільки тоді інструменти NLP зможуть перетворити текст на те, що може зрозуміти машина.

Наступним кроком є побудова алгоритму NLP. Алгоритми обробки природної мови. Після попередньої обробки даних настав час переходити до наступного кроку: створення алгоритму NLP і його навчання, щоб він міг інтерпретувати природну мову та виконувати конкретні завдання.

Є два основних алгоритми, які можна використовувати для вирішення проблем NLP, які наведені на рисунку 2.2:

Алгоритми обробки природної мови

Системне програмне забезпечення

Прикладне програмне забезпечення

Рисунок 2.2 – Класифікація алгоритмів природної мови

Підхід на основі правил.Системи, засновані на правилах, спираються на граматичні правила, створені вручну, які повинні бути створені експертами з лінгвістики або інженерами знань. Це був найперший підхід до створення алгоритмів NLP, і він все ще використовується сьогодні.

Алгоритми машинного навчання.З іншого боку, моделі машинного навчання базуються на статистичних методах і навчаються виконувати завдання після того, як їм надають приклади (навчальні дані).

Найбільшою перевагою алгоритмів машинного навчання є їх здатність навчатися самостійно. Вам не потрібно визначати правила вручну – натомість машини навчаються на попередніх даних, щоб робити прогнози самостійно, що забезпечує більшу гнучкість.

Алгоритми машинного навчання подають навчальні дані та очікувані результати (теги), щоб навчити машини створювати асоціації між певним входом і відповідним виходом. Потім машини використовують методи статистичного аналізу, щоб створити власний «банк знань» і визначити, які функції найкраще представляють тексти, перш ніж робити прогнози для невидимих даних (нових текстів). Послідовність кроків алгоритму навчання та отримання масиву ознак та тегів наведено на рисунку 2.3.

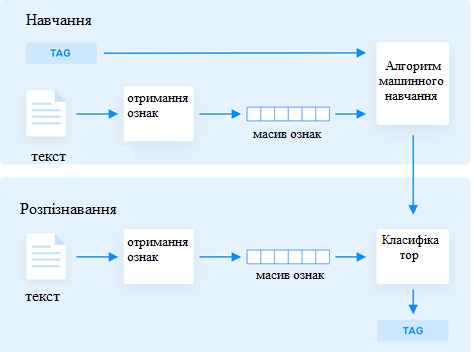


Рисунок 2.3 – Послідовність кроків при машинному навчанні аналізу текстової інформації

Класифікація тексту – це техніка машинного навчання, яка призначає відкритому тексту набір попередньо визначених категорій. Текстові класифікатори можна використовувати для організації, структурування та класифікації майже будь-якого типу тексту – від документів, медичних досліджень і файлів, а також у всьому Інтернеті.

Наприклад, нові статті можна впорядкувати за темами; квитки підтримки можуть бути організовані в терміновому порядку; розмови в чаті можна організувати за мовами; згадування брендів можна впорядкувати за настроями і так далі.

Класифікація тексту є одним із основних завдань у обробці природної мови з широкими застосуваннями, такими як аналіз настроїв, маркування тем, виявлення спаму та виявлення намірів. Ось приклад того, як це працює:«Інтерфейс користувача досить простий і зручний у використанні». Текстовий класифікатор може прийняти цю фразу як вхідні дані, проаналізувати її вміст, а потім автоматично призначити відповідні теги, такі як ПіЗ для словосполучення «простий і зручний».

Етап класифікації є одним з головних при аналізу текстової інформації. За підрахунками, близько 80% усієї інформації є неструктурованою, а текст є одним із найпоширеніших типів неструктурованих даних. Через заплутану природу тексту аналізувати, розуміти, упорядковувати та сортувати текстові дані важко й забирає багато часу, тому більшість компаній не можуть використовувати його повною мірою.

Саме тут на допомогу приходить класифікація тексту за допомогою машинного навчання. Використовуючи класифікатори тексту, системи можуть автоматично структурувати будь-який відповідний текст, починаючи з електронних листів, юридичних документів, соціальних мереж, чат-ботів, опитувань тощо, у швидкий і економічно ефективний спосіб. Це дозволяє компаніям економити час на аналізі текстових даних, автоматизувати бізнес-процеси та приймати бізнес-рішення на основі даних.

Серед основних переваг використання автоматизованих аналізаторів текстових документів є:

Масштабованість. Аналіз і систематизація вручну є повільними та менш точними. Машинне навчання може автоматично аналізувати мільйони опитувань, коментарів, електронних листів тощо за незначну ціну, часто всього за кілька хвилин. Інструменти класифікації тексту можна масштабувати відповідно до потреб будь-якого бізнесу, великого чи малого.

Аналіз в реальному часі. Існують критичні ситуації, які компаніям необхідно якомога швидше виявити та негайно вжити заходів (наприклад, PR-кризи в соціальних мережах). Текстова класифікація за допомогою машинного навчання може постійно стежити за згадками вашого бренду в режимі реального часу, тож ви зможете визначити важливу інформацію та мати змогу негайно вжити заходів.

Послідовні критерії. Люди-анотатори роблять помилки під час класифікації текстових даних через відволікання, втому та нудьгу, а людська суб’єктивність створює непослідовні критерії. З іншого боку, машинне навчання застосовує той самий об’єктив і критерії до всіх даних і результатів. Коли модель класифікації тексту належним чином навчена, вона працює з неперевершеною точністю.

Класифікацію тексту можна виконати двома способами: ручним або автоматичним.

Класифікація тексту вручну залучає людину-анотатора, яка інтерпретує зміст тексту та класифікує його відповідно. Цей метод може дати хороші результати, але він трудомісткий і дорогий.

Автоматична класифікація тексту застосовує машинне навчання, обробку природної мови (NLP) та інші методи, керовані ШІ для автоматичної класифікації тексту швидшим, економічнішим і точнішим способом.

Існує багато підходів до автоматичної класифікації тексту, але всі вони поділяються на три типи систем:

* Системи на основі правил;
* Системи на основі машинного навчання;
* Гібридні системи.

Системи на основі правил. Підходи на основі правил класифікують текст на організовані групи за допомогою набору ручних лінгвістичних правил. Ці правила наказують системі використовувати семантично релевантні елементи тексту для ідентифікації релевантних категорій на основі його змісту. Кожне правило складається з антецеденту або шаблону та передбаченої категорії.

Якщо необхідно розділити статті новин на дві групи: Спорт і Політика. По-перше, потрібно буде визначити два списки слів, які характеризують кожну групу (наприклад, слова, пов’язані зі спортом: футбол, баскетбол, чемпіонат тощо, і слова, пов’язані з політикою вибори, Верховна рада, депутат тощо).

На етапі класифікації нового вхідного тексту потрібно буде підрахувати кількість слів, пов’язаних зі спортом, які з’являються в тексті, і зробити те саме для слів, пов’язаних із політикою. Якщо кількість слів, пов’язаних із спортом, перевищує кількість слів, пов’язаних із політикою, тоді текст класифікується як спорт, і навпаки.

Наприклад, ця система на основі правил класифікуватиме заголовок «Коли відбудеться перша гра футбольного чемпіонату України?» як «Спорт», тому що тут присутні два терміни пов’язаний зі спортом (футбол та чемпіонат), і не враховував жодного терміну, пов’язаного з політикою.

Системи, розроблені на правилах, зрозумілі людині та можуть бути вдосконалені з часом. Але такий підхід має деякі недоліки. Для початку ці системи вимагають глибоких знань домену. Вони також забирають багато часу, оскільки створення правил для складної системи може бути досить складним завданням і зазвичай потребує багато аналізу та тестування. Системи на основі правил також важко підтримувати, і вони погано масштабуються, оскільки додавання нових правил може вплинути на результати вже існуючих правил.

Системи на основі машинного навчання. Замість того, щоб покладатися на правила, створені вручну, класифікація тексту машинного навчання вчиться створювати класифікації на основі минулих спостережень. Використовуючи попередньо позначені приклади як навчальні дані, алгоритми машинного навчання можуть вивчати різні асоціації між фрагментами тексту та те, що певний вихід (тобто теги) очікується для певного введення (тобто тексту). «Тег» — це заздалегідь визначена класифікація або категорія, до якої може потрапити будь-який текст.

Першим кроком до навчання NLP-класифікатора машинного навчання є виділення ознак: використовується метод для перетворення кожного тексту в числове представлення у формі вектора. Одним із найбільш часто використовуваних підходів є пакет слів , де вектор представляє частоту слова в попередньо визначеному словнику слів.

Наприклад, якщо ми визначили, що словник має такі слова {вивчати, студент, оцінка, університет, дисципліна}, і необхідно векторизувати текст «Студент прийшов вивчати мову пригорумування С++ в університеті», то на виході буде наступне векторне представлення цього тексту: (1, 0, 1, 0, 0, 0, 1).

Потім на вхід алгоритму машинного навчання подається масив з навчальними даними, які складаються з пар наборів функцій (векторів для кожного текстового прикладу) і тегів (наприклад, спорт, політика), щоб створити модель класифікації:

Коли модель машинного навчання навчена достатньою кількістю навчальних зразків, вона може почати робити точні прогнози. Той самий екстрактор функцій використовується для перетворення невидимого тексту в набори функцій, які можна ввести в модель класифікації, щоб отримати прогнози щодо тегів (наприклад, спорт, політика).

Класифікація тексту за допомогою машинного навчання зазвичай набагато точніша, ніж системи правил, створені людиною, особливо для складних завдань класифікації NLP. Крім того, класифікатори з машинним навчанням легше підтримувати, і ви завжди можете позначати нові приклади, щоб вивчати нові завдання.

Деякі з найпопулярніших алгоритмів класифікації тексту включають сімейство алгоритмів Байєса, опорні векторні машини (SVM) і глибоке навчання.

Сімейство статистичних алгоритмів Байєса є одними з найбільш використовуваних алгоритмів у класифікації та аналізі тексту в цілому.

Одним із членів цього сімейства є мультиноміальний метод Байєса (MNB) з величезною перевагою в тому, що ви можете отримати дійсно хороші результати, навіть якщо ваш набір даних не дуже великий (~ пара тисяч зразків з тегами), а обчислювальні ресурси обмежені.

Алгоритм Байєса базується на теоремі Байєса, яка допомагає обчислити умовні ймовірності появи двох подій на основі ймовірностей появи кожної окремої події. Отже, можна обчислити ймовірність кожного тегу для певного тексту, а потім виводимо тег із найвищою ймовірністю.

Імовірність A, якщо B є істинним, дорівнює ймовірності B, якщо A є істинним, помноженому на ймовірність того, що A є істинним, поділеному на ймовірність того, що B є істинним.

Це означає, що будь-який вектор, який представляє текст, повинен містити інформацію про ймовірність появи певних слів у текстах даної категорії, щоб алгоритм міг обчислити ймовірність належності цього тексту до категорії.

Метод опорних векторів (Support Vector Machines (SVM)) – це ще один потужний алгоритм машинного навчання класифікації тексту, оскільки, як і алгоритм Байєса, SVM не потребує багато навчальних даних, щоб почати надавати точні результати. Проте SVM потребує більше обчислювальних ресурсів, ніж алгоритм Байєса, але результати ще швидші та точніші.

SVM малює лінію або «гіперплощину», яка розділяє простір на два підпростори. Один підпростір містить вектори (теги), які належать до групи, а інший підпростір містить вектори, які не належать до цієї групи.

Оптимальною гіперплощиною є та з найбільшою відстанню між кожним тегом. У двох вимірах це виглядає так (рисунок 2.4):

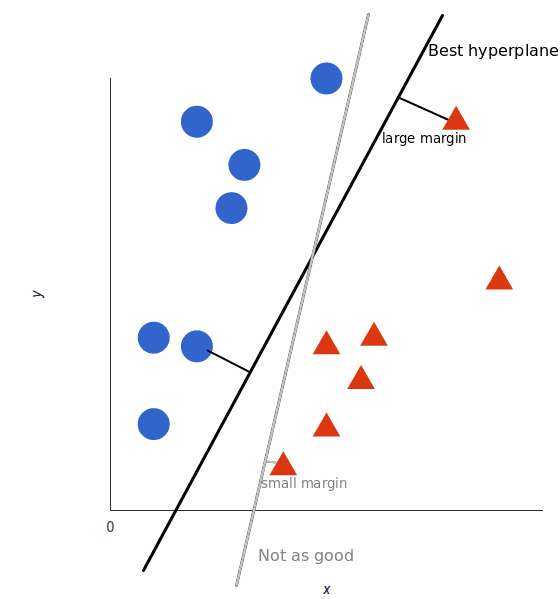


Рисунок 2.4 – Приклад класифікації на основі вектору опорних векторів

Ці вектори представляють навчальні тексти, а група – це тег, яким позначили свої тексти.

Оскільки дані стають складнішими, неможливо класифікувати вектори/мітки лише за двома категоріями. Але це чудова риса алгоритмів SVM – вони «багатовимірні». Отже, чим складніші дані, тим точнішими будуть результати. Уявіть наведене вище в трьох вимірах із доданою віссю Z, щоб створити коло. Повернувшись до двох вимірів, ідеальна гіперплощина виглядає так (рисунок 2.5):

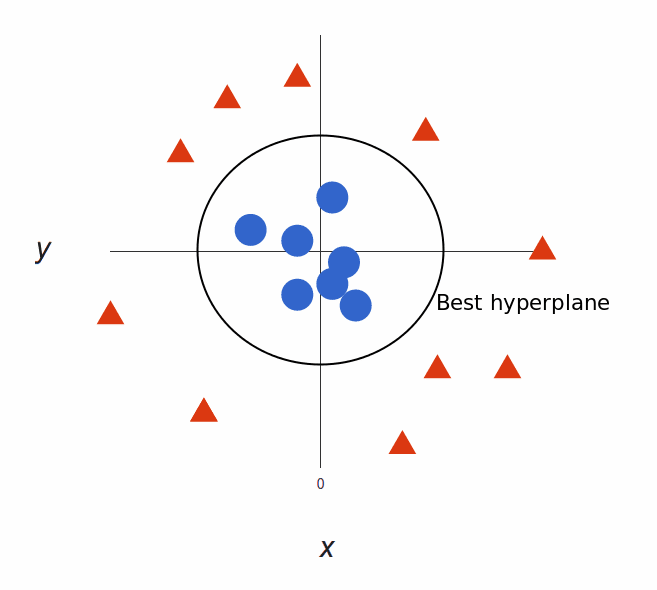


Рисунок 2.5 – Приклад класифікації на основі вектору опорних векторів у випадку декількох критеріїв

Глибоке навчання – це набір алгоритмів і методів, створених за принципом роботи людського мозку, які називаються нейронними мережами. Архітектури глибокого навчання пропонують величезні переваги для класифікації тексту, оскільки вони працюють із надвисокою точністю з нижчим рівнем розробки та обчислень.

Двома основними архітектурами глибокого навчання для класифікації тексту є згорткові нейронні мережі (CNN) і рекурентні нейронні мережі (RNN).

Глибоке навчання – це ієрархічне машинне навчання з використанням кількох алгоритмів у послідовному ланцюжку подій. Це схоже на те, як працює людський мозок під час прийняття рішень, використовуючи різні методи одночасно для обробки величезних обсягів даних.

Алгоритми глибокого навчання вимагають набагато більше навчальних даних, ніж традиційні алгоритми машинного навчання (принаймні мільйони прикладів з тегами). Однак у них немає порогу для навчання з навчальних даних, як традиційні алгоритми машинного навчання, такі як SVM і NBeep, навчальні класифікатори стають кращими, чим більше даних ви їм надаєте:

Алгоритми глибокого навчання, такі як Word2Vec або GloVe, також використовуються, щоб отримати кращі векторні представлення слів і підвищити точність класифікаторів, навчених традиційними алгоритмами машинного навчання.

Гібридні системи.Гібридні системи поєднують базовий класифікатор, навчений машинним навчанням, із системою на основі правил, що використовується для подальшого покращення результатів. Ці гібридні системи можна легко налаштувати, додавши спеціальні правила для конфліктуючих тегів, які не були правильно змодельовані базовим класифікатором.

Метрики та оцінка. Перехресна перевірка є поширеним методом оцінки продуктивності текстового класифікатора. Вона працює шляхом поділу навчального набору даних на випадкові набори прикладів однакової довжини (наприклад, 4 набори з 25% даних). Для кожного набору текстовий класифікатор навчається з рештою зразків (наприклад, 75% зразків). Далі класифікатори роблять прогнози для своїх відповідних наборів, і результати порівнюються з анотованими людиною тегами. Це визначить, коли прогноз був правильним (справжні позитивні та справжні негативні), а коли він був помилковим (хибно-позитивні, хибно-негативні).

За допомогою цих результатів ви можете створити показники ефективності, які стануть у пригоді для швидкої оцінки того, наскільки добре працює класифікатор:

Точність: відсоток текстів, які були класифіковані з правильним тегом.

Якість: відсоток прикладів, отриманих класифікатором із загальної кількості прикладів, які він передбачив для даного тегу.

Передбачуваність: відсоток прикладів, які класифікатор передбачив для даного тегу, від загальної кількості прикладів, які він мав передбачити для даного тегу.

Оцінка F1: середнє гармонійне значення точності та передбачуваність.

## 2.2 Алгоритми предекативного введення текстів

Інтелектуальний набір. Ефективність набору – це поєднання точності, актуальності та швидкості. Хоча сама людська майстерність і талант відіграють важливу роль у підтримці високої продуктивності, інтелектуальні інструменти набору тексту виводять їх на абсолютно новий рівень. Одним із найважливіших удосконалень у тому, як людина може друкувати, є Lightkey, програмне забезпечення на основі штучного інтелекту, яке не тільки вгадує, що вводитиме користувач далі. Цей потужний інструмент має можливості прогнозування кількох слів, які дозволяють передати на 12 слів у різних мовах, включаючи знаки пунктуації.

Люди називають Lightkey та подібні інтелектуальні інструменти набору під різними іменами, як-от програмне забезпечення для прогнозування введення тексту та програмне забезпечення для інтелектуального введення тексту. Основна відмінність полягає в тому, що прогнозоване введення тексту стосується мобільного набору тексту, а інтелектуальне введення стосується використання клавіатури ноутбука чи ПК. Незважаючи на це, вони обидва стосуються здатності програми подавати пропозиції слів на основі тем і шаблонів введення.

Інтелектуальний набір тексту – це загальний термін, який використовується для позначення всіх форм інтелектуального тексту та процесів набору тексту, керованих ШІ. Його назвали «інтелектуальним» за його здатність передбачати літери та слова, які йдуть після введення певної клавіші на основі шаблонів натискання клавіш і тем. Крім того, інтелектуальне введення тексту може запускати письмові та вбудовані пропозиції, щоб допомогти користувачам швидше створювати речення та абзаци.

Інтелектуальний набір тексту не обмежується передбачуваним текстом. Його можливості глибокого навчання також дозволяють йому завчасно виправляти орфографічні та граматичні помилки та неправильно вжиті слова. Якщо користувач введе математичне рівняння, програмне забезпечення також може обчислити та запропонувати відповідні відповіді. Чим частіше користувач використовує інтелектуальний текст, тим краще він адаптується до моделей введення та швидше реагує на натискання клавіш.

В теорії автоматизованого доповнення/прогнозування текстової інформації є ряд термінів та визначень, для більш чіткого розуміння сфери використання того чи іншого алгоритму (рисунок 2.6).

Автоматичне доповнення тексту

Інтелектуальний текст

Інтелектуальний набір тексту

Прогнозування тексту

Прогнозування слів

Рисунок 2.6 – Класифікація за критерієм доступу до ПЗ

Інтелектуальний текст – термін, що стосується технології прогнозування в мобільних пристроях.

Інтелектуальний набір тексту: термін, що стосується технології передбачення в ПК, де фізична клавіатура використовується для введення тексту замість текстових повідомлень або натискання на сенсорному екрані.

Прогнозування тексту: термін, який Microsoft використовує, щоб говорити про досвід передбачення. Це дозволяє користувачам швидше висловлюватися, передбачаючи відповідні слова та фрази на основі контексту та тем.

Прогнозування слів: термін, подібний до передбачення тексту, але пропонує або виконує інтелектуальне написання одного слова.

Окрім цих основних термінів є ще ряд допоміжних (рисунок 2.7):

Сфери використання механік автоматичного доповнення слів

Автоматичне виправлення

Вбудована пропозиція слова

Пропозиції щодо написання

Персоналізоване передбачення

Рисунок 2.7 – Класифікація за критерієм доступу до ПЗ

Автоматичне виправлення: це досить суперечлива технологія передбачення тексту, яка перекриває введені користувачем дані за власним вибором і часто викликає небажані побічні ефекти.

Вбудована пропозиція: це стосується способу, у який передбачення передаються візуально. Клавіатури всіх мобільних пристроїв пропонують кілька передбачень для кожного натискання на екрані. Вбудоване передбачення пропонує пропозиції та виправлення під час редагування тексту. Вони відображатимуться над текстом у дужках.

Пропозиції щодо написання: це термін, який Google використовує для опису своєї технології передбачення тексту в продукті Smart Compose. Його створено, щоб допомогти користувачам швидше писати електронні листи в Gmail.

Персоналізоване передбачення: цей термін стосується передбачень тексту та виправлень у реальному часі, які випливають із унікального стилю написання користувача. Наприклад, коли користувач пише матеріал про програмне забезпечення передбачення штучного інтелекту, введення «pred» призведе до того, що система запропонує «інтелектуальний текст». Однак, якщо писати твір, пов’язаний зі світом медицини, і написати «pred», то можна отримати назву препарату «преднізон».

Інтелектуальне програмне забезпечення для введення тексту призначене не лише для того, щоб допомогти друкувати швидше й точніше, але й для зменшення когнітивного навантаження. Когнітивне навантаження означає просто кількість енергії, яку користувач споживаєте, намагаючись вибрати серед пропозицій передбачення, представлених ШІ.

Більшість програм для інтелектуального набору тексту не працюють, тому що користувачі надто відволікаються багатьма рекомендаціями, які виникають. Негативний ефект від таких технологій виникає, коли пропонуються слова чи фрази, які абсолютно не пов’язані з темою вмісту та ходом наших думок.

Використання технологій інтелектуального введення тексту має плюси та мінуси. Порівняльна зарактеристика наведена в таблиці 2.1.

Таблиця 2.1 – Переваги та недоліки технології інтелектуального введення

|  |  |
| --- | --- |
| Переваги | Недоліки |
| * Прискорює набір; * Упорядковує слова та фрази; * Ефективний; * Виправляє прості помилки; * З часом адаптує вибір слів. | * Зменшує використання когнітивних навичок; * Обмежує словниковий набір; * Робить користувачів залежними; * Порушує набір тексту; * Схильний до помилок. |

## 2.3 Алгоритми автоматичного доповнення слів в програмному коді

При розробці алгоритму предекативного доповнення слівв програмному коді написаному на мові програмування java за основу були обрані ряд параметрів які будуть безпосередньо впливати на отриманий результат. Серед явних параметрів будуть масив заразервованих команд мови програмування, окрім того слід враховувати назви класів та їх методів, що будуть підключатись під час роботи. Додаткове в масив молживих варіантів необхідно додати назви змінних, методів, класів та інших службових слів та виразів які може теоретично використати програміст. Додатково слід враховувати на якій позиції від початку стрічки розташоване слово або чи є перед ним необхідне ключове слово з яким вони сформують сталу структуру. Аналіз та доповнення проводиться з структурованим текстом, а тому очевидно правила написання програмного коду на відповідній мові програмуання теж слід враховувати. Окрім того, для отримання більш точнішого підбору потенційного слова чи словосполучення необхідно враховувати індивідуальні звички користувача. Кожних з програмістів має унікальну матеру написання коду, вибору імен для змінних чи методів, команди які використовує частіше, підходи до вирішення тих чи інших задач. Оскільки даний програмни додаток повинен працювати в режимі реального часу, то і вимоги до швидкодії алгоритму автоматичного доповнення тексту ставляться достатньо високими. Для цього пропонується розпочинати процес доповнення після введення як мінімум 2 літер з слова, це дозволить виключити випадки коли слово є невеликим або в якості змінної використовується одна літера. Окремо слід зазначити, що при подібності початкових секторів слів та словосполучень вищий приорітет буде мати те слово, що має більшу частоту появи для відповідного розробника. Запропонований алгоритм використовує інформацію про загальні принципи написання програмного коду на мові високого рівня, а також враховує індивідуальні риси. Блок-схема алгоритму наведена на рисунку 2.8.

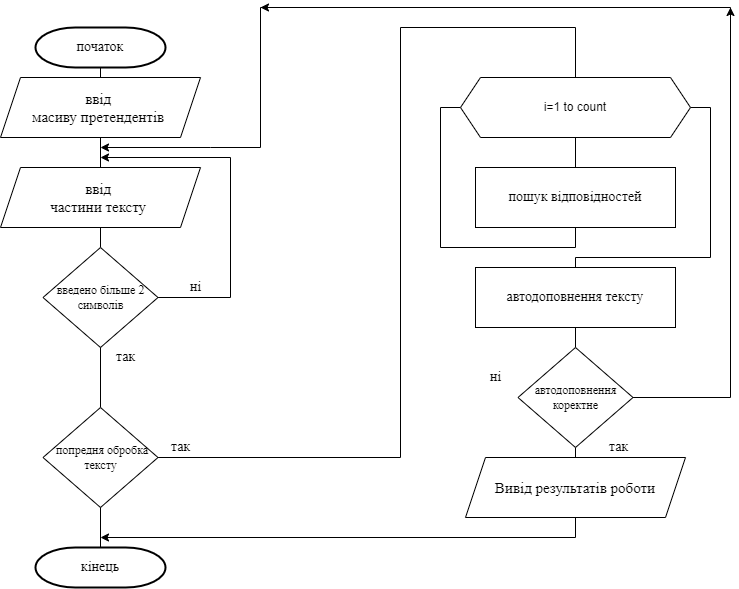


Рисунок 2.8 – Блок-схема алгоритму інтелектуального вводу тексту програмного коду

Основна ідея даного алгоритму полягає в тому, що при введенні тексту програмного коду програміст буде частіше використовувати слова, словосполучення чи команди які притаманні його стилю написання коду. В такому випадку маючи наперед створений масив можливих слів та частоту їх використання при попередніх запусках програмних додаток з може не тільки запропонувати можливий варіант доповнення деякого слова чи терміна, а й з підвищеною ймовірністю запропонувати більш очікуваніший варіант.

Розроблений алгоритм інтелектуального доповнення текстів програмних кодів на основі статистичного аналізу та аналізу індивідуальних особливостей користувача містить наступні кроки:

1. Завантаження масиву слів та словосполучень (тегів) з попередніх запусків.
2. Отримання символа з вхідного потоку.
3. Якщо символ не «пропуск» і він по рахунку менший 3 або це «пропуск» переходимо на крок 2.
4. Перевірка масиву претендентів з введеною послідовністю символів.
5. Якщо в масиві тегів є співпадіння, то формуємо масив претендентів та преходимо на крок 7.
6. Якщо співпадінь не має, то запам’ятовуємо всі введені слова до введення «пропуска». Вважаємо, що було створено нову змінну чи назву метода, для перевірки гіпотези проводимо перевірку чи перед введеним словом є слово яке відповідає за тип даних. Якщо є то вносимо нове значення в масив тегів та переходимо на крок 2. Якщо відсутнє, то вважаємо, що це коментар, ігноруємо введене слово та повертаємось на крок 2.
7. Проводимо визначення позиції слова в стрічці.
8. Ранжуємо масив претендентів на основі позиції слова в стрічці та ваги самого слова.
9. Перевіряємо чи введена частина слова не може бути частиною деякого словосполучення, для цього проводимо перевірку попереднього слова.
10. Виводимо перші 5 варіантів для вибору користувача. Якщо користувач обрав запропонований варіант, то замінюємо введену частину слова на запропонований вираз, якщо ні – переходимо на крок 2 для отримання більш точнішого опису словаякий хоче користувач.

Проведений аналіз введеного слова не лише на основі співпадіння слова з масивом можливих варіантів, а й з використанням оцінки частоти їх використання користувачем дозволяє значно підвищити точність прогнозованого результату. Окрім того, враховуючи, що програмни текст все таки не є повністю не стурукторованим, то використання конструкцій мови для прогнозування дій користувача підвищує заральний відсоток правильно запропонованих варіантів.

До основних переваг розробленого алгоритму відносяться:

* висока швидкодія за рахунок мінімальної кількості перевірок;
* можливість роботи на комп’ютерах з низькими апаратними характеристиками;
* простота та зрозумілість реалізації;
* можливість налаштування робти під конкретного користувача.

Недоліки:

* результати прогнозування залежать від повноти масива тегів та часу аналізу роботи конкретного користувача;
* при великій подібності значень в масиві тегів алгоритму необхідно більш довга вхідна частина шуканого слова для підбору варіантів.

В результаті моделювання робти запропонованого алгоритму він показав хороші результати по швидкості роботи та точності при автомазованому доповненні тексту.

## 2.4 Висновки до розділу

Проведно аналітичний огляд алгоритмів аналізу та обробки текстової інформації з використанням статистичного підходу, що дозволило обрати підхід на основі вибору значення на основі вагових коефіцієнтів для алгоритму інтелектуального доповнення тексту програмного коду.

Розроблено алгоритм предекативного доповнення текстів на основі статистичного аналізу та дослідхення поведінки користувача, що дозволило реалізувати програмний додаток інтелектуального доповнення програмного коду на мові Java.

# 3 Програмний додаток інтелектуалізованого введення програмних кодів

## 3.1 Структура програмного додатку предекативного введення текстів

Інтелектуальне введення тексту – це популярна технологія для аналізу та автоматизованого формування тексту. Коли користувач пишете текст в пошуковому терміналі Google, то система автоматично видає рекомендації щодо поточного чи наступного слова. Це приклад інтелектуального доповнення тексту. Система намагається зрозуміти, що користувач скажете далі та запропонувати йому можливі закінчення його думок. Коли система працює правильно, це економить час і зусилля.

В більшості випадків інтелекттуальнне введення базується на використання технології N-грамам. N-грами – це, в основному, набір слів, що зустрічаються разом у певному контексті. Вони використовується для ідентифікації шаблонів послідовності слів. Отже, на першому етапі проводиться навчання на основі набуру речень які були обрані як набір таких, що максимально відповідають стулю написання програмного коду відповідного користувача та для специфічного контексту.

На цьому етапі для виділення частин та ключових слів використаємо кластеризацію та класифікацію. Кластеризація – це техніка машинного навчання, яка допомагає групувати подібні елементи на основі їхніх атрибутів. Кластеризація є чудовим кандидатом для використання неконтрольованого навчання. У неконтрольованому навчанні немає навчального набору даних із попередньою класифікацією. Замість цього проводиться групування елементів на основі подібності атрибутів. У кластеризації доступна низка методів, таких як k-середнє кластеризування та k-найближчі сусіди. Під час роботи зі текстом слова в документі стають функціями. Документи зі схожими словами групуються разом. Алгоритми кластеризації використовують лише числові дані. Отже, текстові дані потрібно перетворити на числові представлення. Частота тексту в документах у словах або tf-idf є найпопулярнішим методом, який використовується для цієї мети. Він перетворює масив документів на числову матрицю, де документи представляють рядки, а слова представляють стовпці. Кластеризацію тексту можна використовувати для групування таких документів, як огляди, новинні статті та твіти, на основі слів, які використовуються в цих документах.

Класифікація – це техніка машинного навчання для навчання під наглядом. Алгоритми класифікації створюють моделі на основі цільової змінної в наборі даних. Для побудови цих моделей використовуєються інші змінні функції, доступні в наборі даних. Модель використовується для прогнозування класу нових даних. Вона передбачає цільову змінну на основі інших змінних ознак, доступних у нових даних. На вході відбувається поділ вихідних даних на навчальні та тестові дані, навчальні дані використовуються для створення моделі, а тестові дані використовуються для перевірки її точності.

Під час аналізу тексту слова в документі стають змінними функції. Для цілей навчальних моделей кожен документ має бути позначений певним класом. Потім це використовується як цільова змінна для побудови моделей. Більшість алгоритмів класифікації вимагають, щоб ознака та цільові змінні мали числові значення.

На наступному етапі необхідно буде використати техніку N-грам, для того щоб створити базу даних попередніх і можливих наступних слів. Потім до цієї бази даних N-грамів надсилається запит, щоб передбачити наступне можливе слово. Щоб побудувати точну базу даних, рекомендується створити спеціальний масив на основі контексту, яким є конкретний користувач або програма.

На сонові виділених етапів роботи алгоритму предекативного введення текстів була розроблена структура програмного додатку для інтелектуального введення текстової інформації. Спрощену структуру програмного додатку наведено на рисунку 3.1.

Центральний модуль програмного додатку

Модуль отримання текстової інформації

Модуль навчання програми

Модуль попередньої обробки

Модуль аналізу введеного слова

Модуль доповнення текстів

Модуль роботи з БД

Модуль встановлення атрибутів

Модуль виводу даних

Рисунок 3.1 – Узагальнена структура програмного додатку для інтелектуального введення текстової інформації

Запропонована структура дозволяє розбити процес реалізації програмного додатку на окремі етапи та проводити розробку окремих модулів незалежно один від одного. Даний підхід є ресурсо ефективним та дозволяє чітко підкреслити преваги запропонованого алгоритму предекативного ведення тексту. Окрім того модульна структура надає широкий простір для внесення змін до архітектури програмного рішення без значної її зміни. Серед головних структурних частин слід відмітити такі:

Центральний модуль програмного додатку – це основний модуль програмного додатку який виконує функції встановлення та підтримки інтерфейсів взаємодії між різними частинами програмнного додатку. Окрім того в даному модулі предбачено молижвість перекодування параметрів вхідної інформації у формат який може бути реалізований в додаткових модулях при нарощуванні системи. В даному блоці реалізовані алгоритми перевірки цілісності програмного додатку, наявність поточних оновлень модулів та версії або від’єднання непотрібних або малоінформативних блоків програми, конвертація та перевірка коректності повідомлень в середині системи та при обміні з зовнішніми структурами. Вхідними даними даного модуля є вектор параметрів роботи програми. Вихідними – активні параметри роботи активного модуля, а також запити до користувача у випадку винекнення помилкової ситуації або відсутності/ не справності окремо взятого модуля або зовнішнього анаратного засобу.

Модуль навчання програми – використовується з метою отримання первинної інформації для встановлення початкових параметрів робти програмного додатку. А саме шляхом опрацювання вхідної вибірки отримати набір слів та словосполучень для масиву тегів, що максимально характеризують та є характеними для тексту програмного коду на обраній мові програмування. Окрім того, даний модуль має можливість завантаження масиву тегів з результатами попередніх запусків програмного додатку. Окремим етапом є додавання до масиву тегів нових елементів під час підключення зовніхніх бібліотек та класів, оскільки вони містять набір зарезервованих назв які необхідно додати до вже існуючиго набору. Окрім того ще однією функцією даного модуля є можливість підрахунку частоти появи того чи іншого слова в програмному коді, що є необхідною умовою для отримання більш точніших (очікуваніших) результатів під час проонозування можливого результату вводу. Вхідні дані – це набір навчальної вибірки текстових файлів з програмними кодами, набір структорованого опису методів та параметрів відповідних класів, а також масив тегів отриманих при попередньому запуску програмного додатку, а вихідною – масив тегів які описують слова та словосполучення які можуть зустрітись в тексті програмного коду на мові програмнування високого рівня..

Модуль навчання програмної системи

Функції тестування коректності навчальної вибірки

Функції оцінки частоти використання

Функції конвертації

Функції користувацького інтерфейсу

Функції контролю отриманих даних

Функції кластеризації

Функції класифікації

Рисунок 3.2 – Узагальнена структура модуля навчання програми

Модуль попередьої обробки. В даному модулі згруповані функції для організації етапу вводу текстової інформації користувачем під час написання програмного коду. Функції даного модуля прехоплюють натискання клавіш від користувача, проводять їх попередні аналіз та формують запити для подальшої робои програмної системи. Основні функції для робти з клавіатурую є стандартними та забезпечують безперебійну взаємодію між користувачем та середовищем написання програмних кодів. Функції даного модуля працюють в режимі реального часута не створюють перешкод або затримок для роботи програмного додатку в цілому. На вхід даного модуля передаютсья коди натиснутих на клавіатурі клавіш, а результатом роботи даного модуля є сформована користувачем частина слова яка передається для подальшого аналізу іншими складовими програмної системи.

Модуль встановлення атрибутів – модуль, що не відноситься до основної групи програмних модулів і виконує допоміжні функції для підвищення роботи користувача з програмою. Серед можлвих параметрів які дозволяє коригувати даний модуль є:

* можливість встановлення затримок перед активацією автозаміни;
* кількість можливих варіантів під час автодоповнення;
* налаштування шрифтів для виводу підказок на екран тощо.

Модуль попередньої обробки – основне завдання модуля коректно опрацювати введені користувачем дані та перетворити їх у зручний для подальшого опрацювання формат. Функції даного модуля забезпечуються можливість інгорування повторного введення символів «пропуск», введені користувачами одностірчкові та багатострічкові коментарі, слова які не відповідать тегам в база даних або критеріям відповідності введених слів до вимог додавання в базу даних. Коректна робота даного модуля забезпечує швидку роботу програмного додатку в цілому за рахунок зменшення кількості запитів для подальшої обробки. Реалізовані в системі алгоритми дозволяють сформувати попередній опис слова яке хоче ввести користувач та в подальшому передати дану інформацію на наступний програмний модуль системи. На вхід даний модуль отримує послідовність символів з яких в результаті внутрішніх перевірок формує початкову версію слова або словосполучення яке хоче ввести користувач.

Модуль роботи з БД – допоміжний модуль який допомагає реалізувати програмний інтерфейс з сервером бази дани для отримання набору тегів які мають найбільшу подібність до введених символів та частоту використання конкретним користувачем. Результати роботи данго модуля є масив можливих слів претендентів які можуть відповідати введеному користувачем початку слова.

Модуль аналізу введеного слова – після отримання початкового значення вхідного слова чи словосполучення даний модуль формує запит до да бази даних тегів. В результаті роботи попередні модулів функції даного програмного блоку отримають початкову частину можливого слова та масив можливих слів та словосполучень. Після цього користувачеві надається можливість обрати один з запропонованих варіантів з запропонованої множини. Якщо жоден з варіантів не є коректним, то користувачу пропонується продовжити подальше введення тексту програми. Варіант з відсутністю коректного варіанту може виникнути у двох варіантах. По-перше, у випадку якщо користувач під час введення інформації допустив граматичну або орфографічну помилку, в такому випадку програма буде рахувати що це є нова назва змінної чи методу та після введення символу «пропуск» додасть її в база даних тегів. По-друге – це можливо створення користувачем нової змінної назва якої ще не зустрічалась під час попередніх запусків. В даному випадку дане слово теж буде добавлене в масив.

Розроблена та спроектована структура на основі ієрархічного підходу з використанням окремих модулів на кожному рівні ютакий підхід є гнучким та універсальним та дозволяє зберегти працездатність системи навіт при пошкодженні окремих програмних блоків. Окрім того використаний модульний підхід дозволяє вносити корективи в роботу програмного додатку додаючи різні нові модулі чи модіфікуючи вже присутні, при цьому робота системи та цілісність не порушується.

Наступним етапом створення програмного додатку є процес реалізації запропонованих алгоритмів,проте перед цим доцільно провести процес моделювання можливості коректної взаємодії між різними модулями програмного додатку та функціональними модивостями запропонованих алгоритмів виконувати поставлені перед системою завдання. Моделюваня проводилось за допомогою механізмів універсальної мови моделювання (UML), а семе ввикористовувалисьь діаграми прециденів та діаграма послідовності Серед недоліків застосування універсальної мови моделювання слід відмітити:

* часові затрати. Для проведення грунтовного моделювання необхідно затратити час для того, щоб діаграма була синхронізована з фактичним кодом та розумною. Діаграми UML не запускаються, але вимагають багато часу. Тому вони корисні тільки в тому випадку, якщо розробник може ними керувати.
* можливі втрати інформації. Оскільки моделювання це процес якиї огоплює основні етапи створення, але при цьому ігноруються дрібні деталі, то не завжди можливо передбачити усі труднощо які виникнуть під час функціонування програмного.
* затрати фінансових ресурсів системи. Для проведення якісного модулювання необхідне відповідне програмне забезпечення, а отже потрібні кошти на його закупівлю та підтримку у робочому стані.

Моделювання на основі діаграми предецентів дозволить оцінити рівень взаємодії окремих функції та вирахувати відсотки їх важливості в загальній структурі проекту. Додатково розглядається функціональні можливості які отримає користувач при роботі з системою. Приклад діаграми прецеденів наведено на рисунку 3.3.

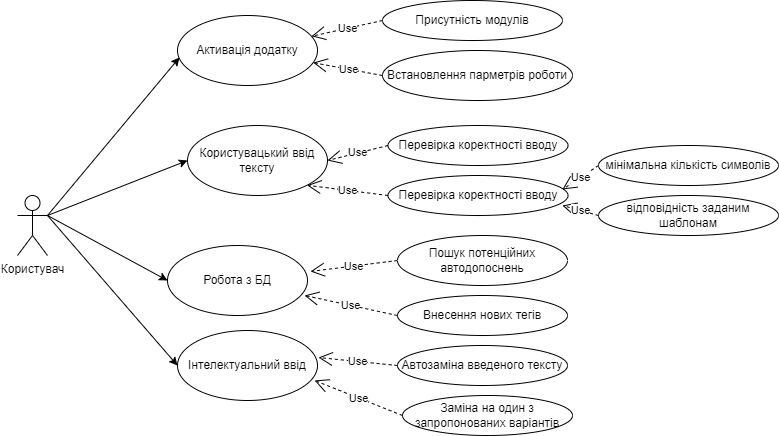


Рисунок 3.3 – Діаграма прецедентів програмної системи

Як показують результати моделювання, що наведені на діаграмі прецедентів запропонована архітектура програмного додатку дозволяє виконувати усі закладені на етапі функції. Користувач отримує доступ до мінімально необхідної кількостю функцій які безпосередньо не впливають на робту програмного додатку в цілому, проте дозволяють коректно йому функціонувати. Дане моделювання показало правильність обраної структури та набору реалізованих функцій. Крім того слід зазначити, що закладені механізми автоматизованої роботи програмних додатків дозволяють використовувати дане програмне забезпечення і користувачам з мінімальним досвідом роботи, оскільки в процесі свого функціонування усю необхідну інформацію програма буде отримувати автоматично. Процес опанування програмою є достатньо короткий та не ввимагає від нових кристувачів грибоких попередніх знань комп’ютерної техніки.

Наступний етап моделювання – це аналіз запропонованих кроків на їх коректність які необхідно зробити користувачеві для досягнення результатів. Аналізувалась взаємодія процесів та об’єктів, що будуть відбуватись під час роботи програми. Результат моделювання наведено на рисуноку 3.4.

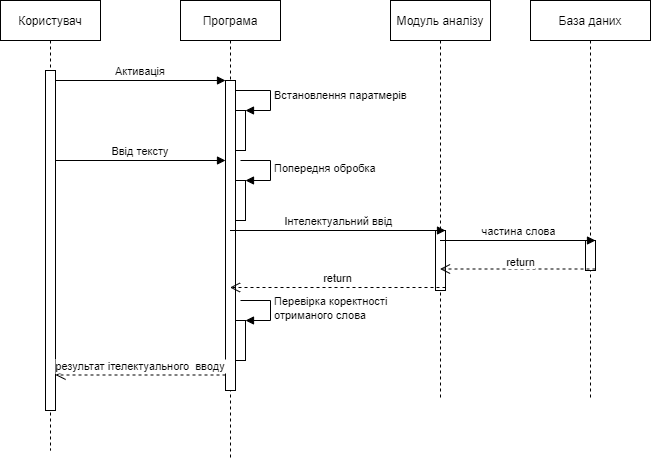


Рисунок 3.4 – Діаграма послідовності програмної системи

Навдена діаграма у повній мірі дозволяє проілюстурвати можливості запропонованої архітектури програмного додатку. На діаграмі можна побачити, що усі процеси активовуються в чіткій послідовності, при цьому між ними не викликають взіємної колізії, а також відсютнє взаємоперехоплення управніння ресурсами у випадку занятості іншими процесами. Лінійна послідовність активації відповідних функцій у деякій мірі сповільнює роботу програмного додатку, проте тут слід враховувати, що даний програмний додаток постійно буде очікувати введену з клавіатури інформацію. Ввід інформації з клавіатури є відносно повільним, тому дана проблема не є актуальною. А враховуючи, що програмний додаток повністю виконує поставлені задачі за відведені часові межі, то даний фактор можна вважати не суттєвим.

Єтап модулювання дозволив оцінити основні фактиро функціонування програмного забезпечення, а також внести корективи в місця де могли виникати потенційні помилкові ситуації. Успішне завершення аного етапу дозволило перейти до наступного, а саме проектування інтерфейсу взаємодії користувача та програмного додатку.

Розроблений програмний додаток має допоміжну функцію при роботі з середовищами розробки програмних засобів, а тому при проектуванні графічного інтерфейсу було принято рішення використати мінімальстичний стиль. Оскільки користувач всеодно не буде безпосередньо постійно взаємодіяти з програмою, тому на головному вікні передбачено використання невеликої кількості перемикачів які забезпечать неохідний мініму для коректної роботи програми.

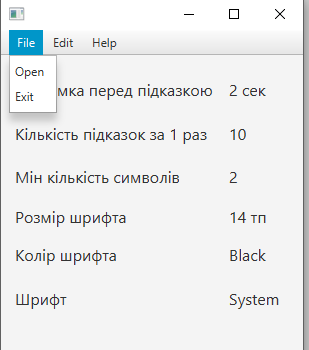


Рисунок 3.5 – Графічний інтерфейс користувача

Графічний інтерфейс користувач було розроблено з врахуванням сучасних вимог до програмних інтерфейсів, а саме використано принцип мінімальзму. Принцип мінімальзму полягає в тому, щоб зменшити кількість активних елементів на основних вікнах програмних додатків. Це зумовлено тим, що на сьогоднішній десь все більше користувачів без достатнього рівня досвіду використовують спеціалізоване програмне забезпечення. А отже встановлення некоректних параматрів роботи може призвести до помилкової роботи програми або до виникнення критичних та непередбачуваних ситуацій.

Для роботи з програмою користувачеві необхідно виконати просту послідовність кроків. Даний набір чітко ілюструється підказками під час запуску програмного додатку, а також може бути проілюстрований в процесі роботи програми у вигляді підказок які з’являються. Послідовність необхідних дій відображено на рисунку 3.6:

Активація системи

Встановлення параметрів роботи

Користувацький ввід тексту

Попередня обробка

інтелектуальни ввід тексту

Оцінка автодоповнення тексту

Рисунок 3.6 – Послідовність дій користувача при роботі з інтелектуального вводу тексту програмного коду

Як показано зі схеми дій користувача, при роботі користувач повинен виконати три основі дії з програмною системою:

* активація та корекція парамемтрів роботи;
* ввід тексту програмного коду на мові високого рівня;
* оцінка запропонованого варіанту автозаповнення.

Окрім того користувач може виконати ще дві додаткові дії, які необхідні у більшій мірі для підвищення рівня коректності роботи програми.

## 3.2 Програмні модулі додатку інтелектуального вводу тексту

Для початку необхідно створити конвеєр обробки тексту, додаючи анотатори, необхідні для виконання аналізу тексту, наприклад tokenize, ssplit, parse та sentiment. З точки зору Stanford CoreNLP, анотатор – це інтерфейс, який працює з об’єктами анотації, де останні представляють проміжок тексту в документі. Наприклад, анотатор ssplit потрібен для розбиття послідовності токенів на послідовності.

Після того, як текст було розбито на послідовності, анотатор синтаксичного аналізу виконує синтаксичний синтаксичний аналіз залежностей, генеруючи представлення залежностей для кожної послідовності. Потім анотатор обробляє ці представлення залежностей, перевіряючи їх на відповідність базовій моделі, щоб побудувати двійкове дерево з мітками (анотаціями) для кожної послідовності.

Простіше кажучи, вузли дерева визначаються токенами кожної послідовності та містять анотації, що вказують на передбачуваний клас. На основі цих прогнозів анотатор обчислює вагу кожної послідовності.

Метод init() ініціалізує інструмент оцінки в конвеєрі CoreNLP, який створюється, а також ініціалізує токенизатор, аналізатор залежностей і роздільник речень, необхідні для використання цього інструменту оцінки. Щоб ініціалізувати конвеєр, необхідно передати об’єкт Properties із відповідним списком анотаторів у конструктор CoreNLP(). Це створює налаштований конвеєр, який готовий виконувати аналіз введеного тексту.

У методі estimatingSentiment() класу nlpPipeline викликається метод process() об’єкта конвеєра, створеного раніше, передаючи текст для обробки. Метод process() повертає об’єкт анотації, який зберігає аналіз поданого тексту.

Далі буде задіяний об’єкт анотації який отримуючи об’єкт CoreMap рівня слова або словосполучення на кожній ітерації. Для кожного з цих об’єктів необхідно оримати об’єкт Tree, що містить анотації для визначення потенційно можливих варіантів для заміни.

Приклад лістингу підключення відповідних модулів для автивації роботи програми наведено на рисуноку 3.7.



Рисунок 3.7 – Приклад встановлення початкових параметрів роботи програми

Щоб перевірити наведені вище функції, необхідно реалізувати клас із методом main(), який викликає метод init(), а потім викликає метод estimatingSentiment() класу nlpPipeline, передаючи зразок тексту останньому.

Потім відбувається передача об’єкту Tree методу getPredictedClass() класу RNNCoreAnnotations, щоб отримати кількісну вагу передбаченого слова або словосполучення. Потім, при умові підтвердження користувачем, проводиться автозаміна введеного тексту на запропонований.

У Java String – це об’єкт, представлений у вигляді послідовності символів. Обробка "рядків" символів є фундаментальним аспектом більшості мов програмування. Реалізація рядків як вбудованих об'єктів дозволяє Java надавати повний набір функцій, які роблять обробку рядків зручною. Наприклад, Java має методи для порівняння двох рядків, пошуку підрядка, об’єднання двох рядків і зміни регістру літер у рядку. Клас String має 11 конструкторів і понад 40 службових методів. Рядкові об’єкти є незмінними, це означає, що після створення об’єкта String ви не можете змінити символи, які складають цей рядок. Це не є обмеженням класу String, але це буде дуже корисно в багатопоточних програмах. користувач можете виконувати всі операції з вашим об’єктом String, лише змінені речі – щоразу створюватимуться нові об’єкти String. Цей підхід використовується тому, що фіксовані, незмінні рядки можуть бути реалізовані ефективніше, ніж змінні.

У Java рядки є об’єктами. Подібно до інших об’єктів, ви можете створити екземпляр String за допомогою ключового слова new. Об’єкт String можна створити за допомогою конструктора з використанням ключового слова new, а також можемо використовувати літерали String. Існує принципова різниця між обома способами створення String.

Однією з ключових цілей будь-якої хорошої мови програмування є ефективне використання пам’яті. У міру зростання додатків рядкові літерали дуже часто займають великі обсяги пам’яті програми, і часто існує багато надмірностей у всесвіті рядкових літералів для програми. Щоб підвищити ефективність пам’яті Java, JVM виділяє спеціальну область пам’яті під назвою «пул констант рядків». Коли компілятор стикається з літералом String, він перевіряє пул, щоб побачити, чи вже існує ідентичний рядок. Якщо збіг знайдено, посилання на новий літерал спрямовується на наявний рядок, і новий об’єкт літералу String не створюється.

Клас String надає багато корисних методів, один із них – метод дорівнює. Метод equals використовується для порівняння значення об'єкта. Існує невелика різниця між оператором “==” і методом equals(). Порівняння з використанням «==» називається поверхневим порівнянням, тому що == повертає істину, якщо посилання на змінну вказує на той самий об’єкт у пам’яті. Порівняння за допомогою методу equals() називається глибоким порівнянням, оскільки воно порівнює значення атрибутів. Давайте розберемо цю концепцію за допомогою програми Java.

У java String — це об’єкт, представлений послідовністю символів Послідовність символів, узятих у подвійні лапки, є літеральним рядком. Крористувач може створити об’єкт String за допомогою ключового слова new і конструктора String.

Порівняння рядкових об’єктів можна виконати за допомогою оператора ==, який називається поверхневим порівнянням. Порівняння рядкових об’єктів за допомогою методу equals() називається глибоким порівнянням.

У Java String є незмінним об’єктом, що означає, що після створення його не можна змінити, посилання вказуватиме на новий об’єкт. Якщо наша програма маніпулює рядками, у пам’яті купи буде багато відкинутих рядкових об’єктів, що може вплинути на продуктивність. Щоб обійти ці обмеження, можна використовувати клас StringBuilder або StringBuffer. Якщо використовуєте один із цих класів, які є альтернативою класу String, коли знаєте, що String буде змінено; зазвичай розробник можете використовувати об’єкт StringBuilder або StringBuffer у будь-якому місці, де б використовували String. StringBuffer може мати символи та підрядки, вставлені в середину або додані в кінець. StringBuffer буде автоматично збільшуватися, щоб звільнити місце для таких доповнень, і часто має більше символів, попередньо виділених, ніж насправді потрібно, щоб забезпечити простір для зростання. Подібно до класу String, ці два класи є частиною пакета java.lang і автоматично імпортуються в кожну програму. Класи ідентичні, за винятком наступного:

* StringBuilder більш ефективний;
* StringBuffer є потокобезпечним через усі синхронізовані методи.

Класи StringBuffer і StringBuilder мають одну різницю: у StringBuffer усі методи синхронізовані, тоді як StringBuilder не є потоково безпечним, тому продуктивність краща. Проаналізуємо методи StringBuilder, які також можна застосувати до StringBuffer.

Видалення StringBuilder (int startIndex, int endIndex). Цей метод видаляє частину послідовності символів StringBuilder. Для роботи йому потрібно надати два аргументи int startIndex і endIndex.

StringBuilder insert(int offset, String s ). Цей метод використовується, коли ми хочемо вставити певну послідовність символів, рядок або будь-які примітивні типи за певним індексом у послідовність символів StringBuilder.

Заміна StringBuilder (int початок, int кінець, String s) Цей метод замінить певну частину послідовності символів третім аргументом (String), згаданим у виклику методу.

StringBuilder reverse(). Як випливає з назви, цей метод змінює послідовність символів у цільовому об’єкті StringBuilder.

void setCharAt(int index, char ch). Цей метод слід використовувати, коли ми хочемо замінити один символ у певному індексі другим аргументом методу. приклад синтаксису,

При використання текстових змінних на мові програмування java, слід враховувати наті особливості:

* Рядкові об’єкти є незмінними, що трохи сповільнює роботу програми.
* Java надає класи StringBuffer і StringBuilder для покращеної роботи з рядками.
* StringBuffer і StringBuilder є подібними класами, за винятком того, що StringBuffer має всі методи синхронізовані, а StringBuilder має несинхронізовані методи.

Обрана мова програмування та обрані динамічні цифрові бібліотеки є попюлярними на сьогоднішній день та мають широке застосування під час проектування та реалізації програмних засобів аналізу та інтелектуалізованого вводу текстових дани. Наведени перелік програмний функцій у повній мірі дозволяє оцінити результати реалізації окремих програмних модулів системи, а також ілюструє рівень роботи окремих модулів та обраних програмних рішень. Тому реалізований програмний додаток у повній мірі використовує усі переваги, що надає йому мова програмування java, а також особливості запропонованих алгоритмів обробки та аналізу текстових.

## 3.3 Тестування та аналіз реалізованого програмного додатку

Для проведеня тестових випробувань запропонованого програмного додатку було обрано апаратне забезпечення виходячи з наступних особливостей проведення тестування:

* програма повинна працювати в режимі реального часу або з мінімальними часовими затримками;
* програма повинна працювати паралельно с середовищем розробки програмного забезпечення
* програма повинна працювати на робочих станціях без додаткових вимог до технічних характеристи апаратного забезпечення.

В результаті для проведення тестування було обрано робочу станцію з «офісними» параметрами, що є достатніми для повсякденного використання та дозволяє створювати програмні коди довільної складності. Може використовуватися як робоче місце програміста, тестера, менеджера проекту тощо.. Робоча станція має такі технічні характеристики:

* Процесор: Intel Core i3-7100 1 шт.
* Чіпсет: Intel® B250.
* Оперативна пам’ять: 8GB (2x4GB) DIMM DDR4 (частота шини 2400MHz) Crucial.
* Носій інформації: 500GB (1x500GB) SATA 6Gb/s 3.5" Seagate.
* Відеокарта: ASUS, серія: GeForce GT 730, об'єм відео-пам'яті 2GB, GT730, встановлено 1 шт.
* Мережа: на мат. платі: 1Gb/s 1шт.
* Блок живлення: 1шт. 450W.
* Фірма виробник: Gigabyte.

Технічні характеристики пристрою відображення зображень (DELL P2018H 19.45″ ):

* матриця: TN;
* роздільна здатність: 1600×900 (16:9);
* частота поновлення кадрів: 75 Гц;
* яскравість: 250 кд/м²;
* підсвічування: WLED;
* входи: HDMI 1.4, DisplayPort 1.2, VGA (D-Sub);
* розміри: 570 \* 446 \* 226 мм.

Технічні параметри обраної робочої станції є достатніми для проведення запланованих тестових випробувань та для отримання достовірної інформації про функціональні моливості розробленого програмного забезпечення.

Для обєктивного проведення тестування було запропоновано проведення ряду тестів. При цьому проводилось суб’єктивне оцінюваня роботи програми на основі опитування користувачів з різним рівнем досвіду роботи з середовищем розробки програмних засобів. Окрім того, під час проведення тестування було виставлено ряд додаткових критеріїв, на основі яких було проведено поділ типів програмних кодів на окремі групи:

* звичайний код – в даному випадку дозволялось використовувати тільки класичні методи, а кількість змінних повинна була бути мінімальною, прицьому їх назви повинні були значно відрізнятись одна від одної. Написання назв змінних які містять різницю в 1-3 літери або відрізняються номером неможна;
* користувацький код – в даному випадку розробник міг використовувати класи власного проектування, обмеження на кількість змінних не накладалась, проте зберігалось обмеження на подібність або малу варіативність назв в змінних;
* довільний програмний код – в даному випадку користувачі прауювали без будь яких обмежень, при цьому додатково дозволялось використовувати різні мови та коментарів в програмному коді, назви змінни могли відрізнятись тільки однією.

Для тестування програми було залучено користувачів з різним досвідом розробки програмних додатків і умовно їх можна поділити на наступні групи: початківець (досвід програмування менше пів року), досвідчений (досвід – до 3 років) та професіонал (активна розробка більше 5 років). Результати їх роботи та подальше анкетування були узагальнені та оформленні у вигляді таблиці. Даний підхід забезпечив більш об’єктивний результат оскільки дозволоив зменшити вплив сторонніх похибок під час проведення експериментів. Іншим важливим чинником який впливав на результати тестування є завдання яке ставилось перед розробниками. Оскільки в тестуванні приймали участь користувачі з невеликим досвідом розробки програмних додатків, то було прийнято рішення що завдання будуть носити простий зарактер. Тестерам було запропоновано вирішити класичні завдання по вводу/виводу інформації на екран, створення програмного додатку з моживістю множинного вибору та сортування масиву заданого розміру. Це дозволило зменшити вплив досвіду саме проектування програмних додатків та збільшити вагу оцінки саме автотизованому доповненню слів.

При написанні програмних кодів, що відповідали першій групі, тобто в коді можна було використовувати тільки стандартні засоби та мінімальна кількість змінних відгуки усіх груп користувачів були позитивні. Програмний додаток швидко та коректно пропонував варіанти завтозаміни. При створенні нових назв змінних безпомилково вихначав їх та додавав до масиву тегів. Отож можна зробити висновки, що для написання простих програмних кодів які використовують стандартні засоби мови програмуванн даний програмний помічник інтелектуального вводу текстів показав відмінні результати.

При виконанні тих же завдання, про те при умові що можна використовувати влясні розробки зроблені позамежами проведення тестування було отримано наступні відгуки. В цілому програмний додаток працює з очікуваною продуктивністю та точністю результатів. Часові затримки візуально не відчуваються та дозволяють комфортно працювати в режимі реального часу. Проте були помічений один недолік, при використанні великої кількості нестандартних класів, структур даних та змінних відбувалась невеличка затримка на початковому етапі роботи. Це пояснюється тим, що програмі необхідний деякий час для опрацювання нових програмних кодів та виділення тегів. Проте, при подальшій роботі, програмний додаток оперативно вносив зміни в дазу даний і затримки не повторювались.

При проведенні тестування без додаткових обмежень в загальному програмний додаток отримав більшість позитивних відгуків. Швидкість опрацювання текстів без затримок, відкидання коментарів відбувалось коректним, відділення команд від звичайних стрічок теж відбувалось без проблем. Серед недоліків слід відмітити затримки під час використання зовнішніх/попередньо невідомих класів, а також складність коректного прогнозування назв змінних при умові їх практичної ідентичності (відрізняються одним символом або цифрою). Проте це пояснюється тим, що завдання вибору подібних змінних є не тривіальним та потребує більш детального аналізу не стільки введеної інформації, але й аналізу цілого блоку коду в якому було ці змінні задіяно.

Як видно з наведених результатів тестування, розроблена програмна система успішно пройшла етап тестуваннята показала високі результати роботи при різних задачах, умовах тестування та з групами користувачів, що мають різними досвід роботи з програмами даного типу. Робота програми відбувалась в автоматичному режимі, а результати роботи зберігались у текстовому файлі.

Отримані результати підтвердили припущення, які були зроблені на етапі моделювання, що розроблений програмний код інтелектувального введення тексту має високу швидкодію, та пороводить опрацювання введеного тексту при різних вхідних параметрах, при цьому забезпечуючи відповідну якість результатів роботи.

В результаті проведених експерементів та на основі аналізу отриманих даних була сформована підсумкова таблиця де відображаються отримані результати у згрупованому вигляді. Результати оцінювались якісно та носять невеликий субєктивний характер, проте в загальному можна вважали їх достатньо достовірними. Результати наведено в таблиці 3.1

Таблиця 3.1 – Узагальнена таблиця результатів тестування

|  |  |  |  |
| --- | --- | --- | --- |
| Групи обмежень/Тестери | Початківці  (до пів року стажу) | Досвідчені  (1-3 року стажу) | Досвідчені  (>5 років стажу) |
| Простий код | Оцінка – відмінно  Переваги – швидкість та коректні передбачення  Недоліки – не помічено | Оцінка – відмінно  Переваги – швидкість та коректні передбачення  Недоліки – не помічено | Оцінка – відмінно  Переваги – швидкість та коректні передбачення, не вносить дискомфорт в процес написання програмного коду  Недоліки – не помічено |
| Користувацький код | Оцінка – відмінно  Переваги – швидкість та коректні передбачення  Недоліки – не помічено | Оцінка – відмінно  Переваги – коректність роботи  Недоліки – невелика затримка під час запуску | Оцінка – відмінно  Переваги – коректність роботи, моливість налаштування параметрів  Недоліки – невелика затримка під час запуску. |
| Довільний програмний код | Оцінка – відмінно  Переваги – швидкість  Недоліки – велика кількість пропонованих варіантів | Оцінка – відмінно  Переваги – коректність роботи  Недоліки – невелика затримка під час запуску | Оцінка – відмінно  Переваги – автоматичний аналіз програмних кодів  Недоліки – стартова затримка, велика кількість варіантів при подібності назв |

Отримані результати в процесі тестування показали правильність обраного шляху розробки програмного додатку.

## 3.4 Висновки до розділу

Розроблено та проведено теоретичне моделювання програмного додатку інтелектуального введення тексту, що надало можливість програмно реалізувати програмних додаток написання текстів програмних кодів.

Здійснено тестування розробленого додатоку написання текстів програмних кодів, на основі використання n-грам, що підтвердило правильність у виборі механізму реалізації програми та розробленого алгоритму.

# Висновки

На основі аналізу сучасних програмних систем написання та обробки текстів програмних кодів та алгоритмів предекативного вводу текстів програмних кодів можна зробити такі висновки:

1. Проведено аналіз та класифікацію існуючих груп програмних файлів, що надало можливість окреслити основні завдання та особливості внутрішньої архітектури виконуваних програмних файлів в різних операційних системах.
2. Проаналізовано аналіз та класивікацію характеристик поширених мов програмування, що дозволило виділити основні структурні елементи та особливості написання програмних кодів на мовах високого рівня.
3. Проведено аналіз наявних програмних засобів для написання та вілагодження програмних кодів який дозволив виділити основні структурні модулі та встановити інтерфейси обміну даними між окремими модулями.
4. Проведно аналітичний огляд алгоритмів аналізу та обробки текстової інформації з використанням статистичного підходу, що дозволило обрати підхід на основі вибору значення на основі вагових коефіцієнтів для алгоритму інтелектуального доповнення тексту програмного коду.
5. Розроблено алгоритм предекативного доповнення текстів на основі статистичного аналізу та дослідхення поведінки користувача, що дозволило реалізувати програмний додаток інтелектуального доповнення програмного коду на мові Java.
6. Здійснено тестування розробленого додатоку написання текстів програмних кодів, на основі використання n-грам, що підтвердило правильність у виборі механізму реалізації програми та розробленого алгоритму.

# Список використаної літератури

1. Сергійчук В.В., Демчук Ю.І. Алгоритм предекативного вводу текстів програмних додатків. Збірник тез VІ Науково-практична конференція молодих вчених і студентів «Інтелектуальні комп’ютерні системи та мережі»., Тернопіль, 10 листопада 2022 р. с. 28.
2. Сергійчук В.В., Демчук Ю.І. Аналіз алгоритмів предекативного вводу текстів. Збірник тез VІ Науково-практична конференція молодих вчених і студентів «Інтелектуальні комп’ютерні системи та мережі»., Тернопіль, 10 листопада 2022 р. с. 29.
3. Bradski G. Learning OpenCV - Computer Vision with the OpenCV Library, 2018. 580 с.
4. Гонсалес Р., Вудс Р., Эддинс С. Цифровая обработка изображений в среде MATLAB, М.: Техносфера, 2016. 616 с.
5. Сафонов В. О. Параметризованные типы данных. История, теория, реализация и применение. М.: Издательство Санкт-Петербургского университета, 2019. 116 c.
6. Секунов Н.Ю. Самоучитель Visual C++ 6.0. М.: СПб: BHV, 2018. 960 c.
7. Страуструп Б. Язык программирования C++. М.: Радио и связь,2019. 350 c.
8. Страуструп Бьерн Дизайн и эволюция С++ М.: ДМК Пресс, 2016. 446c.
9. Сойфер В.А. Методы компьютерной обработки програмних кодов М.: Физматлит, 2020. 784 с.
10. Оппенгейм А. Цифровая обработка сигналов. М.: Техносфера,2016. 856с.
11. Прэтт У. Цифровая обработка изображений. В 2-х книгах: пер. с англ. М.: Мир,2019. 792 с.
12. Грузман И.С. Цифровая обработка изображений в информационных системах: учеб, пособие. 2002. 352 с.
13. Яне Б. Цифровая обработка сигналов. М.: Техносфера, 2017. 584 с.
14. Селянкин В.В. Компьютерное зрение. Анализ и обработка сигналов: учебное пособие. 2019. 152с.
15. Форсайт Д. Компьютерное зрение. Современный подход. 2014. 928 с.
16. Поляков А. Методы и алгоритмы компьютерного анализа. 2019. 545 с.
17. Безрядин С. Н. Преобразование яркости в программном обеспечении. 2016. 211с.
18. Баженова И. Ю. Язык программирования С++ . 2017. 366 с.
19. Бартлетт Н. Программирование на Delphi Путеводитель. 2016. 116с.
20. Вебер Дж. Технология программирования в подлиннике. 2017. 256с.
21. Волш А. И. Основы программирования на С++для World Wide Web. 2016. 458с.
22. Абрамов С. А. Задачи по программированию. 2018. 256 с.
23. Березин Б.И. Начальній курс программирования на языках высокого уровня. 2016. 331с.
24. Бондарев В.М. Основы программирования. 2017. 446с.
25. Вирт Н. Алгоритмы и структуры данных. 2019. 345с.
26. Гладков В. П. Задачи по информатике на вступительном экзамене в вуз и их решения: Учебное пособие. 2021. 516с.
27. Грогоно П. Программирование на языке Java. 2022. 216с.
28. Дагене В.А. 100 задач по программированию. 2019. 106с.
29. Джамса К. Библиотека программиста Java. 2019. 656с.
30. I.Merino, J. Azpiazu 2D Features Detector And Descriptor Selection Expert System. 2019. – С. 51–61.
31. Заварикин В.М. Основі информатики и вычислительной техники. 2019. 556с.
32. Кен А. Язык программирования С++. 2017. 378с.
33. Керниган Б. Язык программирования Java. 2022. 391с.
34. Ляхович В.Ф. Руководство к решению задач по основам информатики и вычислительной техники. 2014. 127с.
35. Мейнджер Дж. С++ Осноы программирования. 2017. 346с.
36. Миков А. И. Информатика. Введение в компьютерные науки. 2018. 442с.
37. Могилев А. В. Информатика: Учеб. пособие для студ. 2019. 629с.
38. Нотон П. JAVA:Справ.руководство. 2016. 447с.
39. Нотон П. Полный справочник по Java. 2017. 556с.
40. Ренеган Э.Дж. 1001 адрес WEB для программистов:Новейший путеводитель программиста по ресурсам World Wide Web. 2021. 512с.
41. Томас М. Секреты программирования для Internet на Java. 2017. 396с.
42. Семакина И. Г. Информатика. Задачник-практикум. 2019. 476с.
43. Сокольский М.В. Все об Intranet и Internet. 2018. 254с.
44. Тассел Д. Стиль, разработка, эффективность, отладка и испытание программ. 2021. 56с.
45. Тюрин Ю.Н. Анализ данных на компьютере. 2019. 384с.
46. Флэнэген Д. Java in a Nutshell. 2018. 473с.
47. Чен М.С. Программирование на С++:1001 совет:Наиболее полное руководство по С++ и Visual С++. 2017. 640с.
48. Эферган М. С++: справочник. QUE Corporation.2018. 256с.
49. Sinha P. Perceiving and Recognizing threedimensional forms. 2016. 278р.
50. Pablo F. Fast Explicit Diffusion for Accelerated Features in Nonlinear Scale Spaces. 2013. 187с.
51. Березький О.М., Дубчак Л.О., Мельник Г.М. Методичні рекомендації до виконання кваліфікаційної роботи з освітнього ступеня “Магістр”. Спеціальність: 123 - Комп’ютерна інженерія. Магістерська програма - Комп’ютерна інженерія". Тернопіль: ЗУНУ, 2022. 32 с.
52. Гураль І.В., Дубчак Л.О. Методичні вказівки до оформлення курсових проектів, звітів про проходження практики, випускних кваліфікаційних робіт для студентів спеціальності «Комп’ютерна інженерія» Тернопіль: ТНЕУ, 2019. 33 с.