Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет «Комп'ютерні технології і системи»

Кафедра «Комп'ютерні інформаційні технології»

# Пояснювальна записка
до кваліфікаційної роботи бакалавра

на тему: «Відновлення граматик ДНК - ланцюгів»
за освітньою програмою: «Інженерія програмного забезпечення»
зі спеціальності: «121 Інженерія програмного забезпечення»
Виконав: студент групи «ПЗ19130»

_____ /Денис ВЕДЕРНИКОВ/
(підпис студента) (Ім'я ПРІЗВИЩЕ)

Керівник: _____ /проф. Віктор ШИНКАРЕНКО/
(підпис) (посада, Ім'я ПРІЗВИЩЕ)

Нормоконтролер: _____ /доц. Олена КУРОП'ЯТНИК/
(підпис) (посада, Ім'я ПРІЗВИЩЕ)

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

Студент _____
(підпис)

Дніпро – 2022 рік

Ministry of Education and Science of Ukraine
Ukrainian State University of Science and Technologies

Faculty «Computer technologies and systems»

Department «Computer information technology»

# Explanatory Note
to Bachelor's Thesis

on the topic: «Restoration of grammars of DNA chains»
according to educational curriculum «Software engineering»
in the Speciality: «121 Software engineering»

Done by the student of the group PZ19130: /Denys VEDERNYKOV/

Scientific Supervisor: /Viktor SHYNKARENKO/

Normative controller: /Olena KUROPIATNYK/

Dnipro – 2022

Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет: «Комп'ютерні технології і системи»
Кафедра: «Комп'ютерні інформаційні технології»
Рівень вищої освіти: бакалавр
Освітня програма: «Інженерія програмного забезпечення»
Спеціальність: «121 Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри КІТ
_____ /Вадим ГОРЯЧКІН/
(підпис)
Дата 22.12.2021

# ЗАВДАННЯ

на кваліфікаційну роботу бакалавра
студенту Ведерникову Денису Сергійовичу
1. Тема роботи: «Відновлення граматик ДНК - ланцюгів»
Керівник роботи: Шинкаренко Віктор Іванович, професор
затверджені наказом № 77 ст від 08.12.2021
2. Строк подання студентом роботи: __.06.2022 р.
3. Вихідні дані до роботи: _____

4. Зміст пояснювальної записки (перелік питань до розробки): Призначення, сфера застосування, функціональні вимоги, предметна область, огляд аналогів, функціональне та експлуатаційне призначення, вхідні та вихідні дані, опис зовнішнього інформаційного середовища, статична архітектура, поведінка системи, проєктування графічного інтерфейсу користувача, архітектура системи, вибір мови програмування, принципи розробки, метод рішення, тестування та налагодження, висновки.
5. Перелік графічного матеріалу: Презентація, яка містить такі основні слайди: ціль розробки, актуальність теми, відомості про предметну область, метод рішення, приклад роботи алгоритму, можливості алгоритму та відео-демонстрація розробленого додатку.

# КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Постановка задач, збір інформації, вибір та обгрунтування критеріїв розробки. Попередній вибір методів рішення задач. Визначення вимог до технічних засобів. Узгодження і затвердження технічного завдання. | 31.01.22 – 18.02.22 | |
| 2 | Програмування та відлагодження програми. | 19.02.22 – 01.05.22 | |
| 3 | Тестування програми. | 02.05.22 – 24.05.22 | |
| 4 | Розробка, узгодження і затвердження програмної документації. | 25.05.22 – 12.06.22 | |
| 5 | Подання кваліфікаційної роботи до кафедри | 13.06.22 | |
| 6 | Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії | 23.06.22 | |

Студент _____ Денис ВЕДЕРНИКОВ
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник роботи _____ проф. Віктор ШИНКАРЕНКО
(підпис) (Ім'я ПРІЗВИЩЕ)

# РЕФЕРАТ

Темою дипломної роботи є «відновлення граматик ДНК–ланцюгів».

Об’єкт дослідження є процеси відновлення граматик ДНК–ланцюгів та особливості структури ДНК–ланцюгів.

Предмет дослідження є дослідження алгоритмів відновлення граматик ДНК–ланцюгів.

Мета кваліфікаційної роботи є створення власного алгоритму відновлення граматик ДНК–ланцюгів, який був би повністю автоматизованим.

Результати та їх новизна: новизна полягає у тому, що було створено власний алгоритм та підхід до відновлення граматик ДНК–ланцюгів. Вирішенні проблеми складності алгоритмів та їх реалізації.

Пояснювальна записка складається зі вступу, 4 частин, додатків:

- у вступі для ознайомлення висвітлюються основні моменти, актуальність, тема, мета, предмет дослідження. Складається з однієї сторінки;
- у першому розділі проведено збір та аналіз вимог для можливості встановити вимоги до алгоритму. Складається з 3 сторінок;
- у другому розділі описано проєктування системи та визначено призначення та вимоги. Складається з 8 сторінок;
- у третьому розділі обґрунтовано вибір мови програмування, принципи розробки, метод рішення та наведено приклад. Складається з 13 сторінок;
- у четвертому розділі представлено тести, описано процес тестування та налагодження додатку. Складається з 6 сторінок;
- додатки містять технічне завдання та робочий проект.

Рисунків – 19, таблиць – 10, джерел – 7.

Перелік ключових слів: ДНК, ДНК–ланцюг, формальна граматика, алгоритм, метод, відновлення граматики.

ЗМІСТ

## ПЕРЕЛІК УМОВНИХ ПОЗНАК, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Формальна граматика – четвірка $G = (V_T, V_N, P, S)$.

$V_T$ – алфавіт термінальних символів.

$V_N$ – алфавіт нетермінальних символів.

P – множина правил підстановки.

S – початковий символ.

ОС – операційна система.

ПЗ – програмне забезпечення.

ПК – персональній комп’ютер.

ПП – програмний продукт.

МДО – мінімальна довжина опису.

ДНК(Дезоксирибонуклеї́нова кислота́) – один із двох типів природних нуклеїнових кислот, що забезпечує зберігання, передачу з покоління в покоління і реалізацію генетичної програми розвитку й функціонування живих організмів. Основна роль ДНК в клітинах — довготривале зберігання інформації про структуру РНК і білків.

# ВСТУП

В наш час все більшої популярності набирає тренд здорового способу життя та довголіття. Проводяться тисячі досліджень в цих напрямках, публікуються дані в відкритий доступ, щоб дослідникам було ще легше та швидше роботи нові відкриття, тому аналіз ДНК-ланцюгів – це актуально як ніколи раніше.

Тема дипломного проєкту «відновлення граматик ДНК–ланцюгів». Об'єктом дослідження власне і є ДНК–ланцюги людини.

Метою дипломної роботи є створення власного алгоритму відновлення граматик ДНК–ланцюгів, який був би повністю автоматизованим та не мав потреби в додаткових діях зі сторони користувача алгоритму.

Щоб досягнути мети було створено план дій:

– спробувати відразу створити власний алгоритм – це єдиний спосіб для проявлення справжньої творчості та правильний шлях до створення власного авторського алгоритму, який би пішов своїм власним шляхом та не був заснованим на старих застарілих поглядах(це зіграло ключову роль).
– після творчої діяльності дізнатись, які взагалі існують алгоритми в цій предметній області;
– проаналізувати існуючі алгоритми на можливість обробки ДНК–ланцюгів та їх теоретичну можливість за конкретну кількість операцій відновити граматику;
– по можливості виявити та перейняти всі сильні сторони знайдених алгоритмів;
– створити новий підхід до вирішення задачі.

На практиці використання алгоритму дозволить розширити існуючі можливості дослідників. Може дати краще розуміння закономірностей людської ДНК. Наприклад, порівнювати граматичні правила ДНК здорових та хворих людей та звісно створити нові метрики та індикатори здоров'я.

# 1 ЗБІР ТА АНАЛІЗ ВИМОГ

## 1.1 Призначення та сфера застосування

Розроблюваний ПП призначений для відновлення граматик ДНК–ланцюгів людини. Новий інструмент для більш детального дослідження ДНК–ланцюгів з метою виявлення нових закономірностей та структур. ПП, а особливо створений алгоритм відновлення граматик повинен стати фундаментальним інструментом для нових досліджень на основі отриманих ним результатів.

Перш за все розроблюваний ПП призначається для дослідницьких центрів та повинен відкрити наступні перспективні можливості:

– пошук часто повторюваних місць в ланцюгах з метою пошуку однакових механізмів в різних системах організмів;

– порівняння ДНК-ланцюгів;

– виявлення різних мутацій та порушень;

– покращення розуміння структури та принципів побудови ДНК;

– створення нових метрик та індикаторів здоров’я.

Основними перевагами ПП «Відновлення граматик ДНК-ланцюгів» є:

– скорочення часу на аналіз ДНК;

– фільтрація однакових механізмів ДНК;

– фіксація однакових місць з метою подальшого дослідження;

– зручний та простий інтерфейс.

## 1.2 Функціональні вимоги

Функціональними вимогами для ПП такого типу можна вважати:

– відновлення граматик ДНК–ланцюгів;

– відновлення граматики погодженої з істинною;

– швидка обробка великих об’ємів даних;

– підтримка роботи з файлами великого розміру;

– можливість збереження результатів в файл.

## 1.3 Дослідження предметної області

До предметної області можна віднести: формальні граматики, алгоритми відновлення граматик та ДНК–ланцюги як об’єкт з яким працює алгоритм.

В загальному випадку будь–яка формальна граматика – це четвірка $G = (V_T, V_N, P, S)$, де:

$V_T$ – алфавіт термінальних символів.

$V_N$ – алфавіт нетермінальних символів.

P – множина правил підстановки.

S – початковий символ.

Відновлення граматики – це певна скінченна послідовність дій, яка призводять до створення алфавіту термінальних та нетермінальних символів, множини правил підстановки та визначає початковий символ.

Існують різні підходи до відновлення граматики, тому якогось загального алгоритму не існує.

В простому розумінні ДНК – це молекула, що зберігає біологічну інформацію у вигляді генетичного коду, що складається з послідовності нуклеотидів. У ДНК зустрічається чотири види азотистих основ: аденін (A), гуанін (G), тимін (T) та цитозин (C). Крім того, зустрічаються ділянки, що належать "генетичним паразитам" та мутації.

Генетичний код складається з трьох буквених "слів", які називаються кодонами, що складаються з трьох нуклеотидів (тобто ACT, CAG, TTT тощо). Один з трьох кодонів, які розташовуються в кінці мРНК, не означає амінокислоту і визначає кінець білка, це «стоп» або «нонсенс» кодони — TAA, TGA, TAG.

Співвідношення, виявлене для аденіну (А), тиміну (Т), гуаніну (Г) і цитозину (Ц), виявилося наступним: кількість аденіну дорівнює кількості тиміну, а гуаніну — цитозину: А=Т, Г=Ц.

Короткий приклад, ДНК – ланцюга: GCCCATCAGTCCTCTGAGACAGGTGAA GAACCTGAGGTCGCAGGAGGACACCCAGAAGGTCCAGCCGCGGTCCCTGCAG TCCCTCCCTGGCGGCTGCGCAGCCGTCCCACGACAGGGGCC.

## 1.4 Огляд аналогів

Існує декілька алгоритмів аналогів. Розглянемо декілька з них більш детально.

Припустимо, що є існує погоджена граматика з істинною граматикою для даного ланцюга. Ціль: знайти граматику, узгоджену з істинною граматикою для даного ланцюга.

Вимога: наявність повної інформаційної послідовності (при теоретичному дослідженні, тому що на практиці повних не зустрічається).

Методом перебору. Суть методу: виконується перебір всіх граматик з деякого заданого класу до тих пір, доки не буде знайдено погоджену граматику з інформаційною послідовністю мови.

При повному переборі можна гарантовано відновити істину граматику але на практиці з’являється проблема комбінаторного вибуху при переборі всіх можливих граматик. Тому цей метод мало прийнятний через обчислювальну складність.

Метод індукції. На відміну від методу перерахування не вимагає знаходження істинної граматики та перебору всіх варіантів. Цей метод опирається на поняття моделі, яка признається принципово неточною тобто, яка має імовірнісний характер. Суть методу полягає в поступовому спрощенню граматики сумісної з даним зразком мови. Індукційні методи не мають суворого обґрунтування. Тому, отримати не те щоб оптимальний, але й хоч якусь прийнятну відповідь немає гарантії.

Метод на основі принципу мінімальної довжини опису. Тут граматика – це генеративна модель початкових даних. У генеративної моделі при породженні правил застосування наступних правил не залежить від того, які правила були застосовані до цього. Принцип МДО каже, що слід шукати компроміс між складністю моделі та її точністю, при цьому і те, й інше може бути виражено у термінах кількості інформації.

Висновки до розділу 1

Визначивши призначення, сферу застосування, дослідивши предметну область та покроковий розбір методів відновлення формальних граматик, можемо встановити вимоги до алгоритму. Слід відмітити, що процес відновлення граматик є досить складним і потребує постійного поліпшення, наприклад, через проблеми складності алгоритмів, складності реалізації.

# 2 ПРОЄКТУВАННЯ

## 2.1 Зовнішнє проєктування

### 2.1.1 Функціональне призначення

Функціональне призначення – ПП призначається для обробки ДНК-ланцюгів з метою відновити їх граматику. Далі на основі отриманих ним результатів будуть проводитись інші дослідження.

### 2.1.2 Експлуатаційне призначення

Експлуатаційне призначення — за допомогою створеного додатку працівник отримує можливість користуватися додатковим інструментом для аналізу ДНК-ланцюгів з метою отримання додаткових даних, які допоможуть виявляти нові закономірності в ДНК-ланцюгах. Порівнювати ланцюги за різні інтервали часу, виявляти різні мутації та порушення в ДНК пацієнтів. Порівнювати метрики з середніми нормами та перевіряти ключові індикатори здоров’я. Отримувати результати, ставити діагнози в короткі строки.

### 2.1.3 Функціональні вимоги

Функціональними вимогами для ПП такого типу можна вважати:

– відновлення граматик ДНК–ланцюгів;
– відновлення граматики погодженої з істинною;
– швидка обробка великих об’ємів даних;
– підтримка роботи з файлами великого розміру;
– можливість збереження результатів в файл.

Також виконаємо специфікацію функціональних вимог у вигляді діаграми прецедентів (рисунок 2.1).

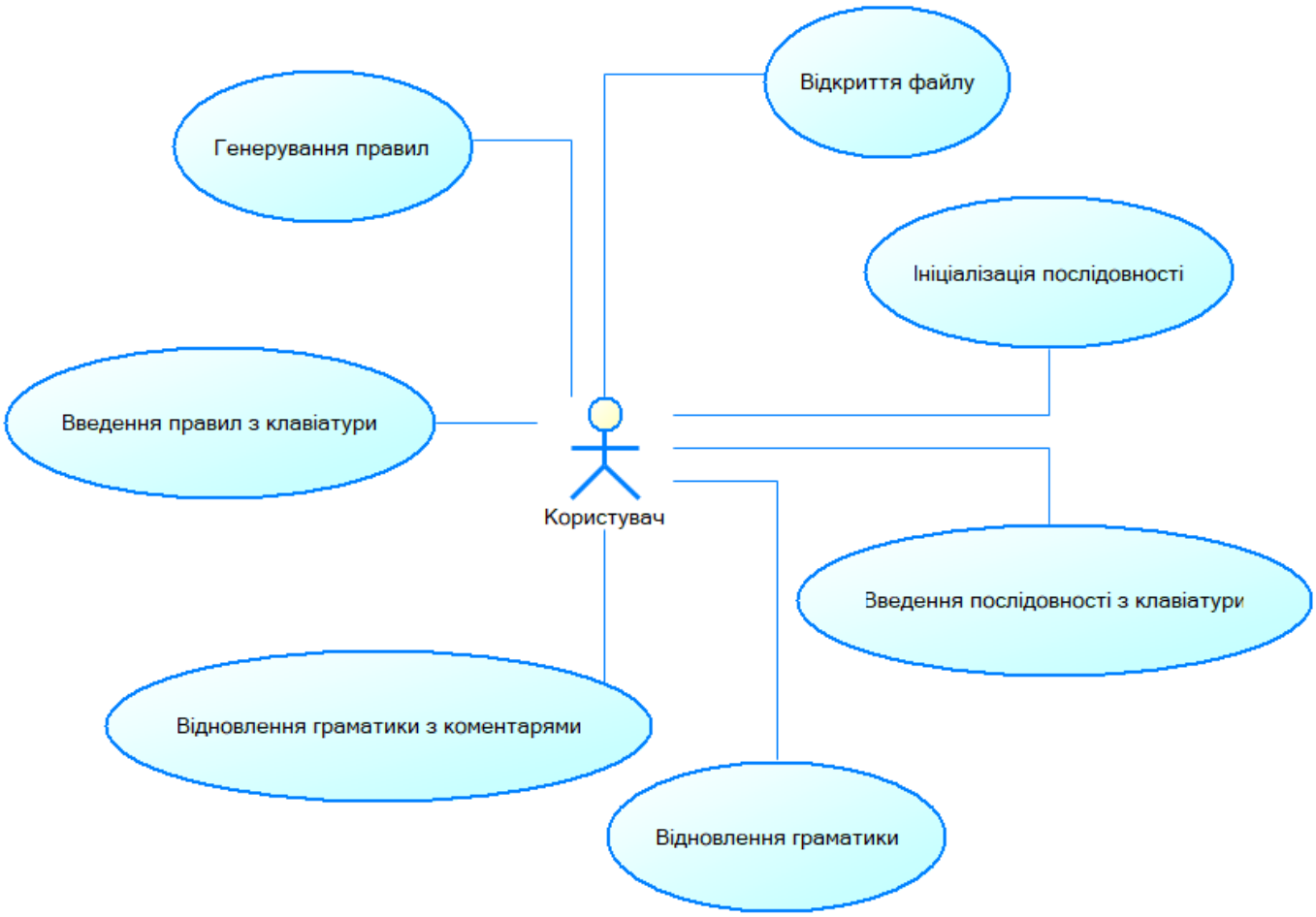


Рисунок 2.1 – Діаграма прецедентів

### 2.1.4 Вхідні дані

Вхідні дані: безперервний ДНК–ланцюг у вигляді послідовності азотистих основ(аденін (A), гуанін (G), тимін (T) і цитозин (C)), які об’єднані по троє і утворюють кодони. Всі дані вводяться за допомогою системної клавіатури та діалогового вікна.

### 2.1.5 Вихідні дані

Вихідні дані: текстовий файл, який містить в собі відновлену граматику ДНК–ланцюга, яка в свою чергу представлена у вигляді четвірки $G = (V_T, V_N, P, S)$, де:

– $V_T$ – алфавіт термінальних символів;
– $V_N$ – алфавіт нетермінальних символів;
– P – множина правил підстановки;
– S – початковий символ.

2.1.6 Опис зовнішнього інформаційного середовища

Для роботи програми непотрібно встановлювати додаткові програми. Послідовності ДНК–ланцюгів буде вноситись в програму через текстовий документ та клавіатуру. Результати роботи зберігаються в текстовому файлі.

Для використання програми необхідний потужний комп’ютер з операційною системою Windows та має наступні характеристики:

– процесор з частотою не нижче 2.4 ГГц;

– розрядність процесору 64 біти;

– 8 Гб оперативної пам’яті;

– 1 Гб вільної внутрішньої пам’яті.

Для початку роботи з програмою непотрібно здійснювати додаткові налаштування.

**2.2 Внутрішнє проєктування**

2.2.1 Статична архітектура

Для відображення архітектури системи здійснено моделювання архітектури системи на основі діаграм компонентів, артефактів і розгортування.

Артефакт GrammarRecoveryApp – файл, який містить код для основного запуску додатку.

Артефакт GrammarRecoveryForm – файл, який містить код візуальної частини додатку.

Артефакт GrammarRecoveryClass – файл, який містить всі функції, які відповідають за відновлення граматики.

Артефакт VectorMethodsClass – файл, який містить код для роботи з векторними даними.

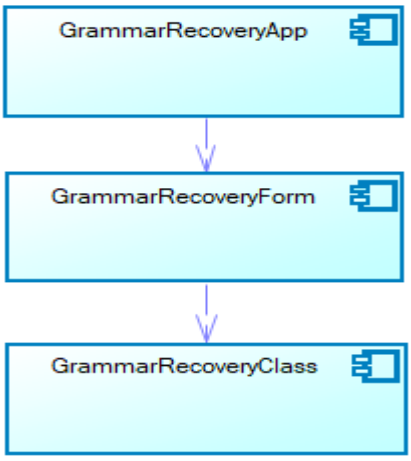


Рисунок 2.2 – Модель архітектури системи на основі діаграми артефактів

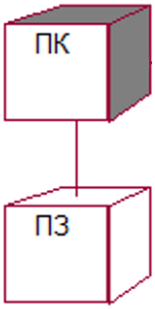


Рисунок 2.3 – Діаграма розгортування додатку

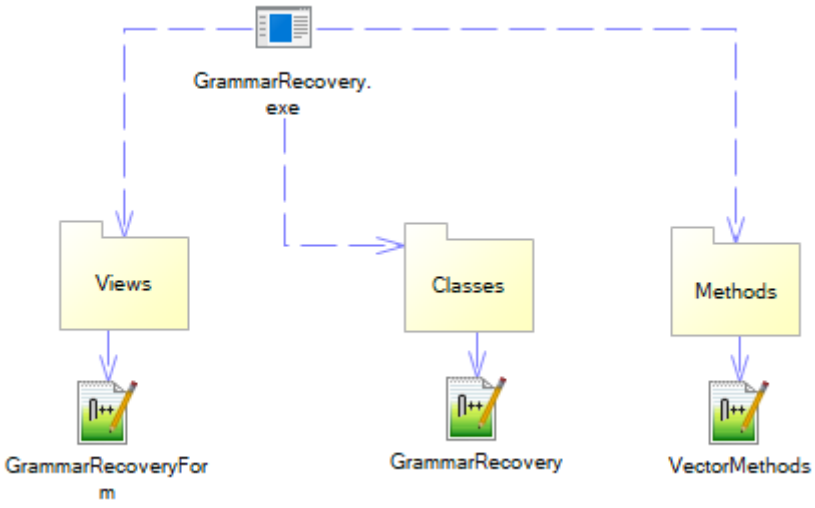


Рисунок 2.4 – Діаграма компонентів проекту

### 2.2.2 Поведінка системи

В системі є головний прецедент – це відновлення граматики, він виникає після відкриття файлу з ДНК–ланцюгом користувачем і відіграє головну роль в ПП, розглянемо його діаграму діяльності.

Діаграми діяльності є аналогом блок-схеми будь-якого алгоритму. Вони, як і діаграми станів та переходів, відображаються у вигляді орієнтованого графу, вершинами якого є дії, а ребрами – переходи між діями.

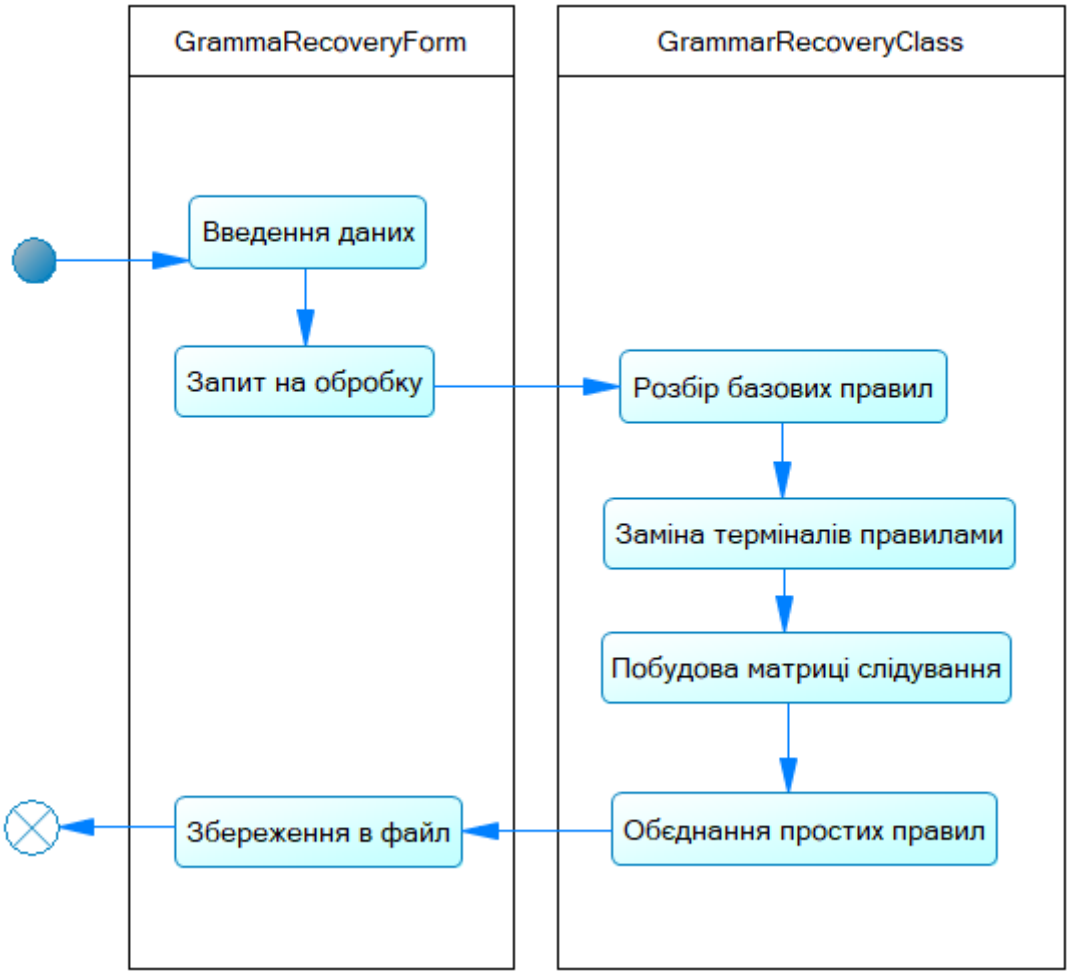


Рисунок 2.5 – Діаграма діяльності для прецеденту «відновлення граматики»

### 2.2.3 Проєктування віконного інтерфейсу користувача

Після запуску програми на екрані повинна з’явитися основна форма додатку. Основна форма складається з поля для ручного вводу даних та поля для відображення повідомлень користувачу. Для виконання якихось дій справа розміщена панель з

кнопками: відкриття файлу, ініціалізації послідовності, генерування базових правил, відновлення граматики та відновлення граматики з додатковими поясненнями.

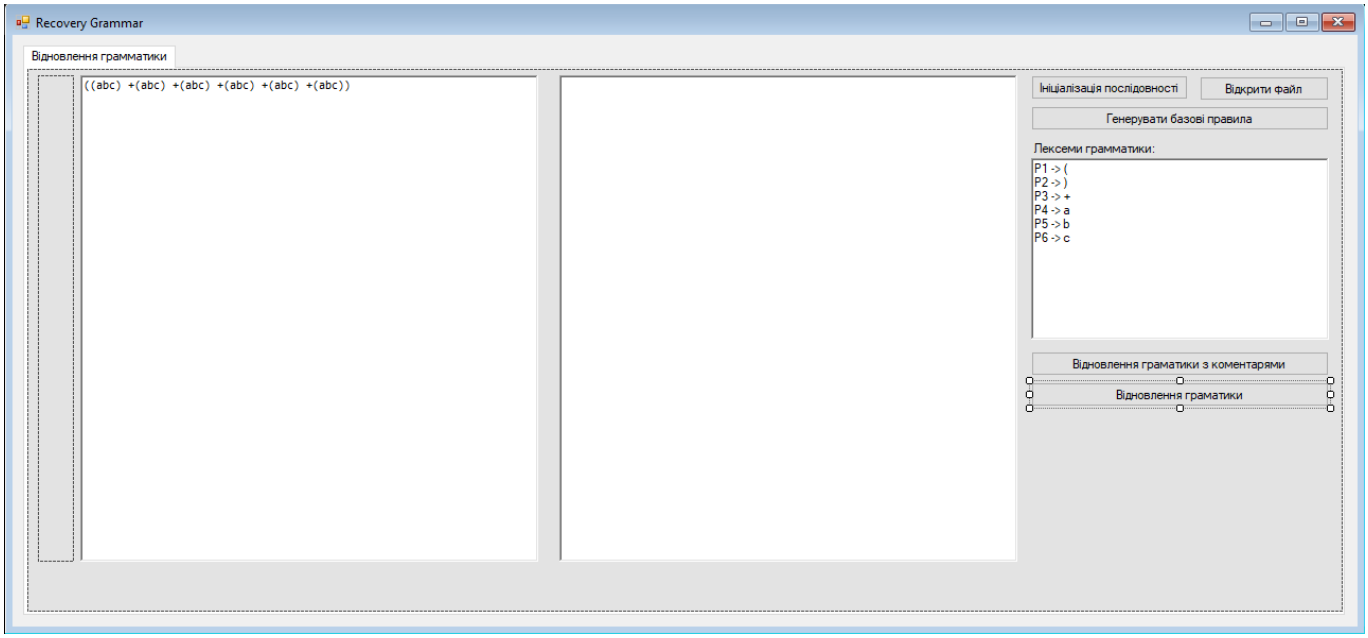


Рисунок 2.6 – Головне вікно програми

Елементи вікна згруповані за логікою використання. Наприклад, поле для вводу розміщено зліва, результати посередині, а панель управління справа. Таке розміщення елементів на формі дозволяє користувачеві мати швидкий доступ до всіх необхідних йому областей, фокусувати увагу на важливому та швидко знаходити потрібний елемент.

## 2.3 Проєктування архітектури системи

Для демонстрації архітектури системи за видом взаємодії класів між собою була розроблена діаграма класів, яка зображена на рисунку 1.

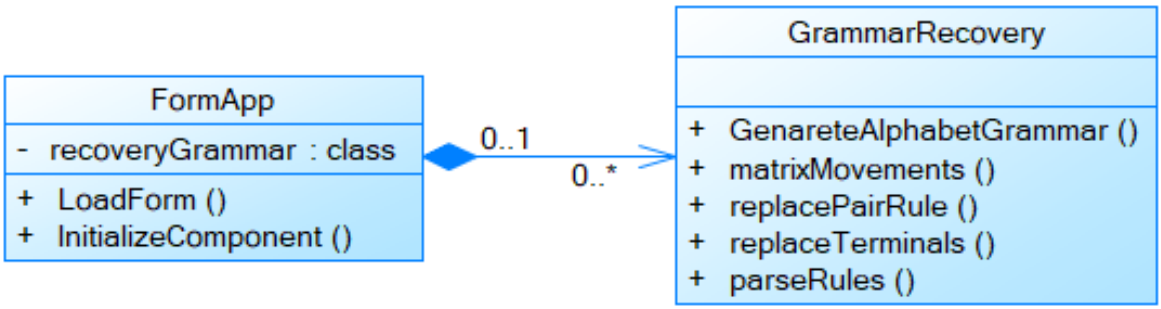


Рисунок 2.7 – Діаграма класів

Визначені класи були специфіковані за допомогою CRC-карток:

Таблиця 2.1 – Картка класу «FormApp»

| Базовий клас | Похідні класи (нащадки) |
|---|---|
| System::Windows::Forms::Form | – |
| Обов’язки | Зв’язки |
| Завантаження додатку, ініціалізація та відображення інтерфейсу | Клас GrammarRecovery |

Таблиця 2.2 – Картка класу «GrammarRecovery»

| Базовий клас | Похідні класи (нащадки) |
|---|---|
| – | – |
| Обов’язки | Зв’язки |
| Виконує всі обов’язки по відновленню граматики | Клас FormApp |

Атрибути та операції, необхідні для виконання обов'язків кожної сутності-класу наведемо у таблиці 2.3.

Таблиця 2.3 – Сутності, атрибути та методи

| Сутність | Атрибути | Методи |
|---|---|---|
| FormApp | linkGrammarRecovery | LoadForm, InitializeComponent |
| GrammarRecovery | Txt, grammarAlphabet | GenareteAlphabetGrammar, matrixMovements, replacePairRule, replaceTerminals, parseRules |

Клас GrammarRecovery є головним класом, він виконує всі необхідні дії для відновлення граматики, а саме:

– зберігає посилання на послідовність ДНК–ланцюга;

- генерує алфавіт для послідовності;
- виконує розбір правил;
- будує таблицю слідування;
- замінює термінали на правила;
- об’єднує пари правил в нові правила.

Висновки до розділу 2

Було визначено можливі прецеденти в програмі під час взаємодії користувача з нею. Представлено опис функціональних вимог, вхідних та вихідних даних, зовнішнього середовища. Розкрито функціональне та експлуатаційне призначення в зрозумілій формі, що має виконувати ПП.

Внутрішнє проєктування дозволило визначити складові системи, що створювалась. Описати, який життєвий цикл вони проходять. Вказати на особливості інтерфейсу та описати його складові, які й будуть використовуватись користувачем для взаємодії з ПП.

Проєктування архітектури системи дозволило розподілити обов’язки, виділити класи, описати їх структуру та поведінку за допомогою CRC-карток та діаграми класів.

# 3 РОЗРОБКА ПРОГРАМИ

## 3.1 Вибір мови програмування

Вибір мови програмування – це важливий етап, тому що від нього залежать швидкість розробки, швидкодія та якість програми.

Тому визначимо наступні критерії до мови:

- ефективність виконуваного коду за часом – це дуже важливий критерій тому, що існують високі вимоги до часу відповіді на запит користувача та програма працює з великими обсягами даних. Висока швидкість обробки даних необхідна для звільнення ресурсів ПК для інших операцій;
- розробка об’єктних програм мінімального розміру – це вимога, яка випливає з необхідності програми в великих ресурсах оперативної та фізичної пам’яті для обробки запитів, тому об’єктний код запиту повинен бути мінімально можливого розміру;
- мінімальний час компіляції програми – це необхідний критерій тому, що процес розробки та налагоджування програми передбачають велику кількість спроб, тому цей показник повинен бути мінімально можливим, щоб швидко розробити та налагодити програму.

Для розробки ПП обрана мова C++ тому, що вона найбільше задовольняє потреби ПП. Мова демонструє найкращу швидкість виконання коду за часом – це дуже важливий критерій тому, що створювана програма застосовується для обробки великих обсягів даних в реальному часі та існують високі вимоги до часу відповіді на запит користувача.

Скомпільовані програми написані на мові C++ мають дуже малий розмір, то не будуть даремно витрачати місце, матимуть високу швидкість завантаження в оперативну пам'ять.

Також програми написані на мові C++ володіють високою надійністю та демонструють стабільне поводження під час роботи, тому що мова підтримується та вдосконалюється передовими компаніями в цій галузі.

Час розробки ПП на цій мові не дуже значний тому, що існують великі спільноти де можна задати питання, багато книг, довідкової інформації та потужні інструменти для швидкої відладки.

### 3.2 Принципи розробки

Крім, правильного вибору мови програмування та ефективних алгоритмів важливі і принципи написання програми. Вони впливають на швидкість написання програми, легкість її зміни, доповнення, повторного використання коду – це також зменшує затрати часу на налагодження та тестування. Далі розглянемо основні принципи.

При розробці ПС використання принципів SOLID допомагає написанню системи, яку просто підтримувати та розширювати протягом всього життєвого циклу.

Принципи SOLID:

- принцип єдиної відповідальності – для кожного класу має бути визначено одне призначення. Всі ресурси, необхідні для його здійснення, мають бути інкапсульовані в цей клас і підпорядковані тільки цьому завданню;
- принцип відкритості / закритості – програмні сутності(класи, структури) мають бути відкриті для розширення, але закриті для зміни;
- принцип підстановки Лісков – функції, які використовують базовий тип, мають мати можливість використовувати підтипи базового типу, не знаючи про це;
- принцип розподілу інтерфейсу – багато інтерфейсів, спеціально призначених для клієнтів, краще, ніж один інтерфейс загального призначення;
- принцип інверсії залежностей – залежність від абстракцій. Немає залежності від чогось конкретного.

KISS (Keep It Simple – будь простіше) – цей принцип говорить, що найпростіші системи працюватимуть краще і надійніше. Не вигадуйте до задачі складнішого рішення, ніж їй потрібно.

DRY (Don’t Repeat Yourself – не повторюйте) – дублювання коду, це марна трата часу та ресурсів. Вам доведеться підтримувати ту саму логіку і тестувати код

відразу в двох місцях, причому якщо ви зміните код в одному місці, його потрібно буде змінити і в іншому.

YAGNI (You Aren’t Gonna Need It – вам це не знадобиться) – коли пишете код, будьте впевнені, що він вам знадобиться. Не пишіть код, якщо думаєте, що він стане в нагоді пізніше.

### 3.3 Метод рішення

Алгоритми основних методів рішення задачі представлені схемами Нассі–Шнейдермана.

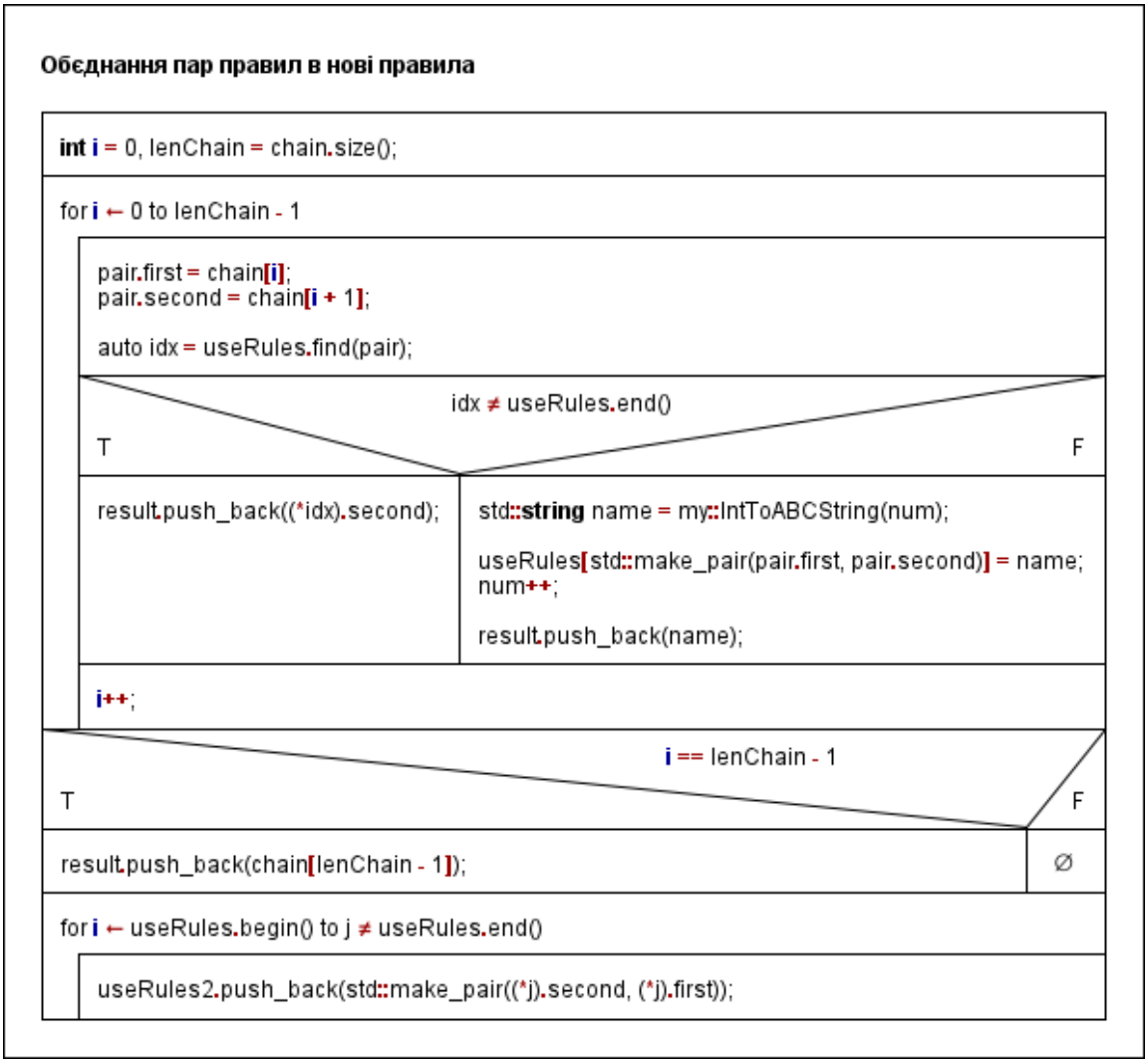


Рисунок 3.1 – Алгоритм об’єднання пар правил в нові правила

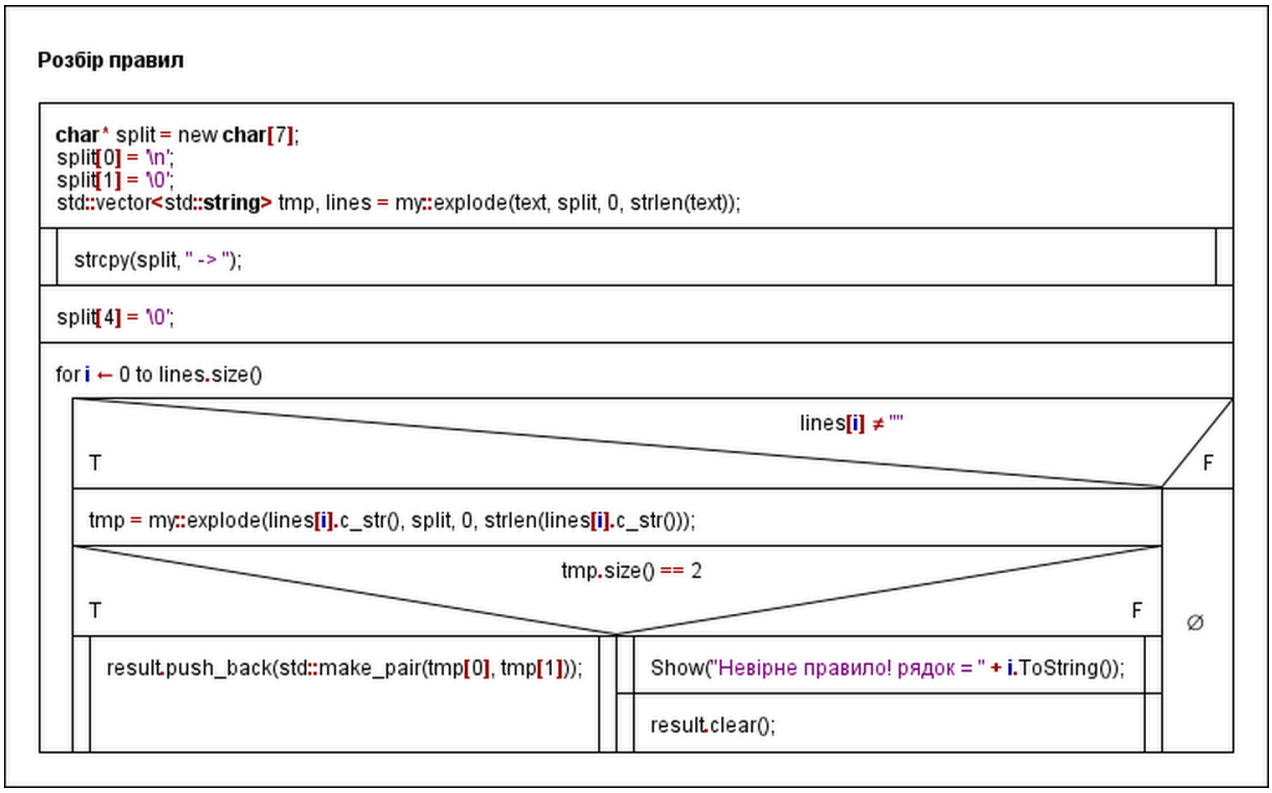


Рисунок 3.2 – Алгоритм створення матриці слідування

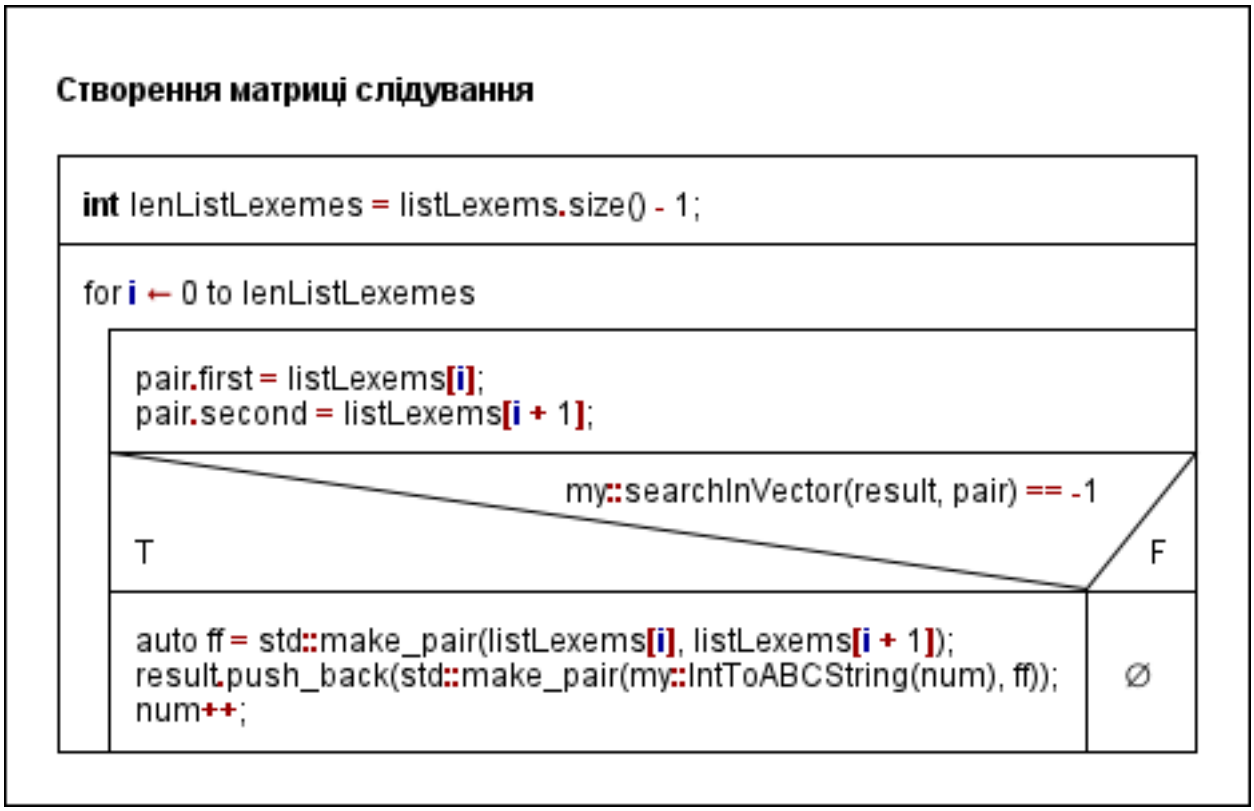


Рисунок 3.3 – Алгоритм створення матриці слідування

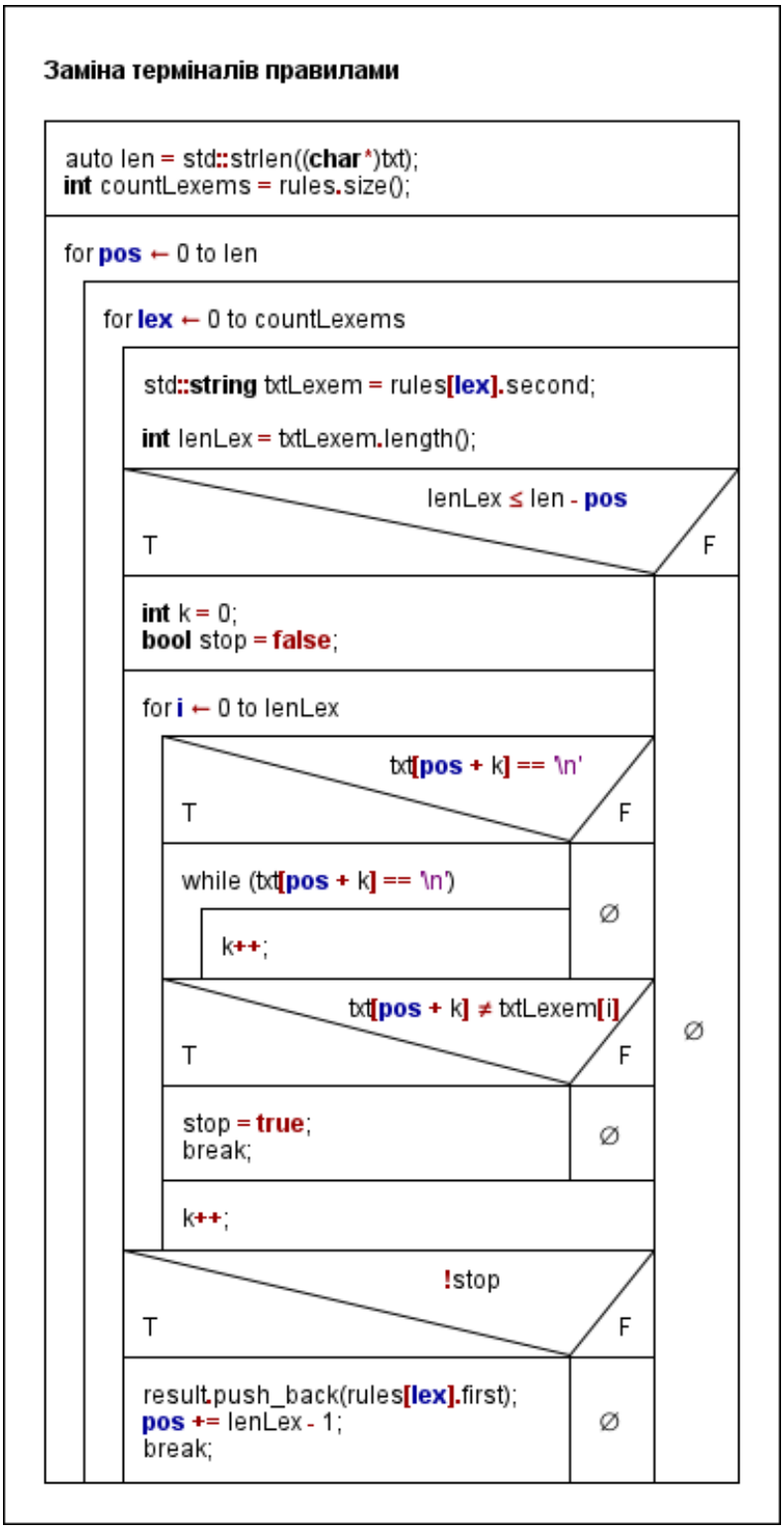


Рисунок 3.4 – Алгоритм заміни терміналів правилами

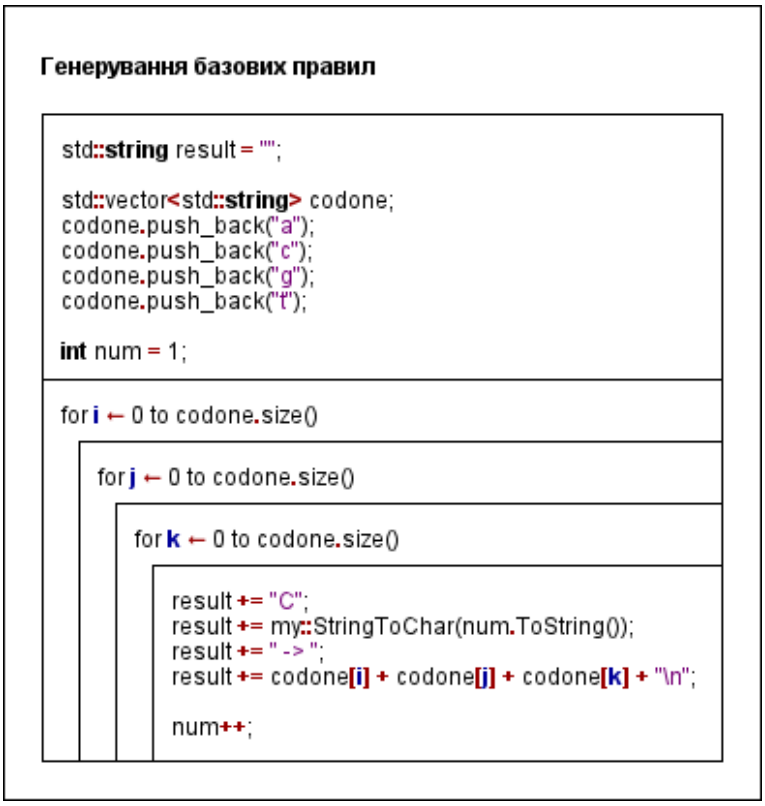


Рисунок 3.5 – Алгоритм генерування базових правил

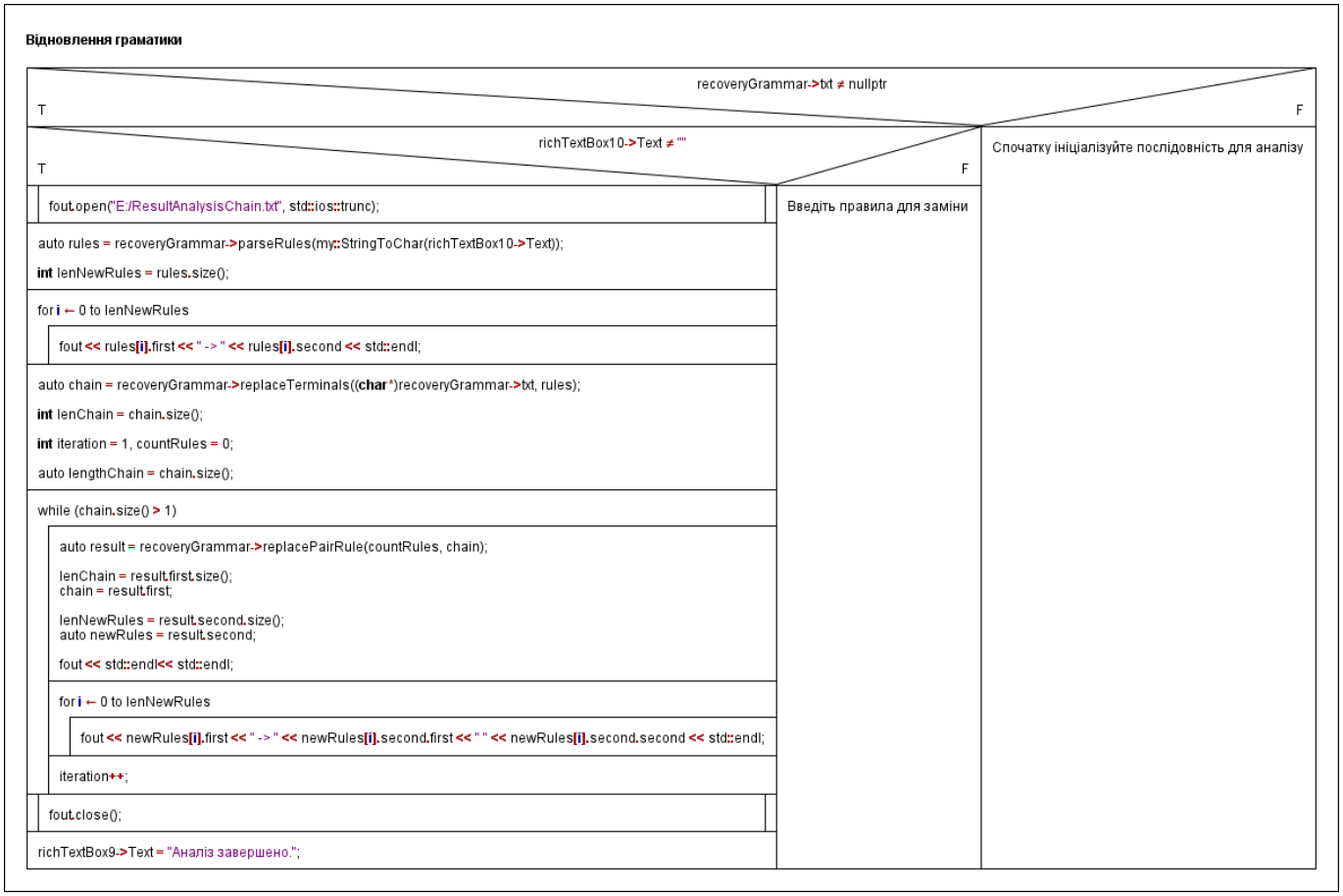


Рисунок 3.6 – Алгоритм відновлення граматики

Розглянемо алгоритм рішення більш детально. Головний принцип алгоритму від простого до складного, тобто об’єднання простих правил в більш складні з виключенням повторень.

Розглянемо та обґрунтуємо принцип побудови правил.

Нехай дано послідовність: gcccatcagtcctctgagacaggt.

Тоді побудуємо можливі переходи, тобто заповнимо таблицю слідувань. Такий підхід дозволяє побудувати всі можливі переходи і головне граматичні конструкції не дублюються, тобто повторення поглинаються – такий підхід дуже ефективний при великій кількості повторень фрагментів в послідовності.

Таблиця 3.1 – Таблиця слідування

| | a | c | g | t |
|---|---|---|---|---|
| a | | 1 | 1 | 1 |
| c | 1 | 1 | | 1 |
| g | 1 | 1 | 1 | 1 |
| t | | 1 | 1 | |

Побудуємо правила по таблиці:

P1 → ac

P2 → ag

P3 → at

P4 → ca

P5 → cc

P6 → ct

P7 → ga

P8 → gc

P9 → gg

P10 → gt

P11 → tc

P12 → tg

З отриманих правил формується нова послідовність і над нею повторно виконуються аналогічні дії доки не залишиться остання пара правил.

Алгоритм позбавлений від рекурсивних викликів, тобто він ітеративний – це означає, що алгоритм має скінчену кількість ітерації, прогнозовану обчислювальну складність та завжди відновлює граматику.

Отримані правила схожі на бінарне дерево, але не являються ним, тому що правила розміщуються в ньому по іншому принципу, описаному раніше та вершини об’єднують прості правила в більш складні, тобто структура однакова, а принципи інші.

Розглянемо приклад в якому правила будуть повторно використовуватися. Для цього представимо правила в вигляді дерева.

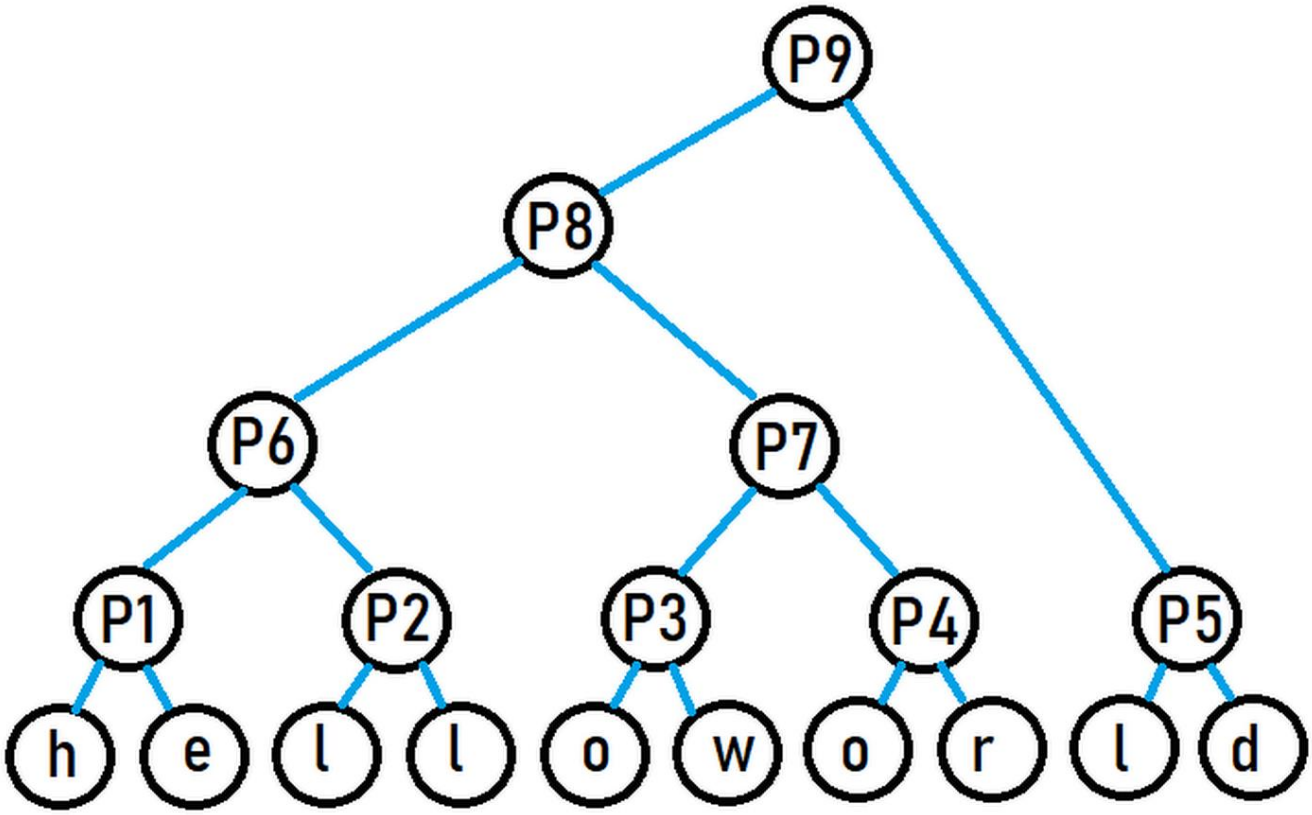


Рисунок 3.7 – Приклад правила, що утворює послідовність “hello world”

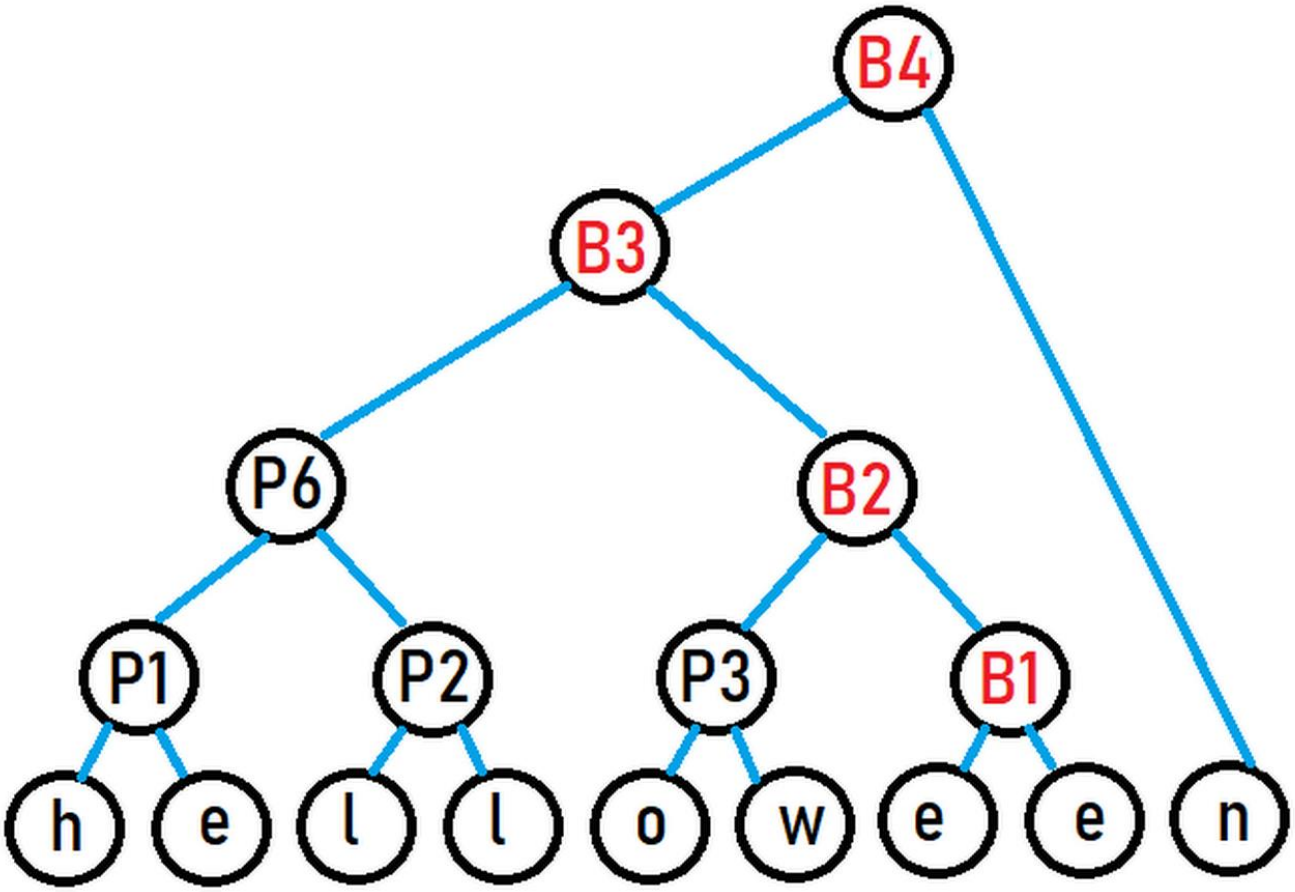


Рисунок 3.8 – Приклад правила, що утворює послідовність “helloween”

На рисунках 3.7 та 3.8 можна легко помітити, що правила P1, P2, P3, P6 були повторно використані, а для підтримки граматикою нового слова довелось її розширити чотирма новими правилами.

### 3.4 Приклад

Як гарний приклад можна розглянути послідовність, яка містить математичний вираз. Цей приклад більш зрозуміліший ніж ДНК–ланцюг та може бути легко перевірений будь ким, така заміна принципово нічого не змінює.

Нехай є послідовність: ((abc) +(abc) +(abc) +(abc) +(abc) +(abc)).

Першим кроком є підготовка: побудова алфавіту та базових правил.

Алфавіт → (, ), +, a, b, c.

Побудова базових правил:

A1 → (

A2 → )

A3 → +

A4 → a

A5 → b

A6 → c

Наступний крок заміна всіх терміналів на прості правила: S → A1 A1 A4 A5 A6 A2 A3 A1 A4 A5 A6 A2 A3 A1 A4 A5 A6 A2 A3 A1 A4 A5 A6 A2 A3 A1 A4 A5 A6 A2 A3 A1 A4 A5 A6 A2 A2.

Далі ітераційним шляхом відбувається виведення правил.

Перша ітерація: будуємо можливі переходи.

Таблиця 3.2 – Таблиця слідування

| | A1 | A2 | A3 | A4 | A5 | A6 |
|---|---|---|---|---|---|---|
| A1 | 1 | | | 1 | | |
| A2 | | 1 | 1 | | | |
| A3 | 1 | | | | | |
| A4 | | | | | 1 | |
| A5 | | | | | | 1 |
| A6 | | 1 | | | | |

Формулюємо нові правила по таблиці слідування:

R1 → A1, A1

R2 → A1, A4

R3 → A4, A5

R4 → A5, A6

R5 → A6, A2

R6 → A2, A3

R7 → A3, A1

R8 → A2, A2

Заміна на нові правила: S → R1 R3 R5 R8 R3 R5 R8 R3 R5 R8 R3 R5 R8 R3 R5 R8 R3 R5 A2.

Видалення невикористаних правил: R2, R4, R6, R7.

Друга ітерація: будуємо можливі переходи.

Таблиця 3.3 – Таблиця слідування

| | R1 | R3 | R5 | R8 | A2 |
|---|---|---|---|---|---|
| R1 | | 1 | | | |
| R3 | | | 1 | | |
| R5 | | | | 1 | 1 |
| R8 | | 1 | | | |
| A2 | | | | | |

Формулюємо нові правила по таблиці слідування:

P1 → R1, R3

P2 → R3, R5

P3 → R5, R8

P4 → R5, A2

P5 → R8, R3

Заміна на нові правила: S → P1 P3 P2 P5 P3 P2 P5 P3 P2 A2.

Видалення невикористаних правил: P4.

Третя ітерація: будуємо можливі переходи:

Таблиця 3.4 – Таблиця слідування

| | P1 | P2 | P3 | P5 | A2 |
|---|---|---|---|---|---|
| P1 | | | 1 | | |
| P2 | | | | 1 | 1 |
| P3 | | 1 | | | |
| P5 | | | 1 | | |
| A2 | | | | | |

Формулюємо нові правила по таблиці слідування:

F1 → P1, P3

F2 → P2, P5

F3 → P2, A2

F4 → P3, P2

F5 → P5, P3

Заміна на нові правила: S → F1 F2 F4 F5 F3.

Кінцева граматика:

S → F1 F2 F4 F5 F3

F1 → P1, P3

F2 → P2, P5

F3 → P2, A2

F4 → P3, P2

F5 → P5, P3

P1 → R1, R3

P2 → R3, R5

P3 → R5, R8

P5 → R8, R3

R1 → A1, A1

R3 → A4, A5

R5 → A6, A2

R8 → A2, A2

Висновки до розділу 3

Вибір мови програмування – це важливий етап в розробці від нього залежить ефективність виконуваного коду за часом, розмір розроблювального додатку, час компіляції, час розробки, тому що якщо мова має велику підтримку зі сторони розробників, то вона має велику кількість книг, уроків, прикладів та документацію, що значно пришвидшує час розробки. Обрано мову C++ тому, що вона найбільше відповідає визначеним критеріям. Наступним важливим етапом є визначення

принципів розробки. Від них залежить майбутнє додатку тому, що вдало написаний додаток буде легко підтримувати та розширювати протягом всього життєвого циклу.

Наприклад, з застосуванням принципів SOLID зникнуть проблеми пов’язані зі складністю модифікації додатки тому, що кожна функція, клас будуть мати одну відповідальність, тобто змінивши одне ми не ламаємо інше та не змінюємо очікувань до результатів виконання функції, а принцип інверсії залежностей дозволяє змінювати одні модулі на інші без необхідності корегувати код по всьому додатку. Інші принципи дозволяють не дублювати код, писати простіше та не писати зайве, яке можливо і не знадобиться або буде реалізовано в іншому вигляді. Було висвітлено основні моменти щодо реалізації та особливостей алгоритму. Головний принцип алгоритму від простого до складного, тобто об’єднання простих правил в більш складні з виключенням повторень. Такий підхід дозволяє створювати правила, які можна буде максимально повторно використовувати та уникати їх дублювання, тобто правила не будуть розширюватися без необхідності, а додання підтримки нових послідовностей або дуже великих послідовностей не буде суттєво збільшувати кількість правил за рахунок великої кількості простих правил, які легко комбінуються в нові.

# 4 ТЕСТУВАННЯ ТА НАЛАГОДЖЕННЯ

Для проведення тестування розробленого програмного додатку обрано метод еквівалентних розбиттів.

Метод еквівалентного розбиття розділяє множину вхідних значень на набори даних так, щоб в кожний набір потрапляли значення, принципово еквівалентні одне одному.

Кожний набір даних таких класів об’єднуються за принципом знаходження однакових помилок. Для кожного класу еквівалентності достатньо одного тесту.

Таблиця 4.1 – Класи еквівалентності

| Вхідні умови | Правильні класи еквівалентності | Неправильні класи еквівалентності |
|---|---|---|
| Значення послідовності ланцюга ДНК | Послідовність складається з символів acgt(1) | Порожня послідовність(2) |
| | Послідовність має термінали, що не входять в алфавіт граматики(3) | |
| Значення базових правил | Правила за дані у вірному форматі. Правила починаються з великої літери, термінал з маленької через роздільник -> . Приклад, C1 -> aaa (4) | Порожні правила(5) |
| | | Некоректно введені правила(6) |
| | | Правила мають термінали, що не входять в алфавіт граматики(7) |

Таблиця 4.2 – Тести за методом еквівалентних розбиттів.

| Клас, що покривається | Тест | Вхід | Вихід |
|---|---|---|---|
| 1, 4 | 1 | acactcgattctgta<br>C5 -> aca<br>C30 -> ctc<br>C36 -> gat<br>C45 -> gta<br>C56 -> tct | C5 -> aca<br>C30 -> ctc<br>C36 -> gat<br>C45 -> gta<br>C56 -> tct<br>B -> C36 C56<br>A -> C5 C30<br>C -> A B<br>D -> C C45 |
| 2, 7 | 2 | Порожня послідовність<br>C1 -> a56<br>C2 -> 8-c<br>C3 -> aag | Повідомлення про помилку |
| 3, 5 | 3 | acactcgaa689875454<br>Порожні правила | Граматика не буде побудована |
| 1, 6 | 4 | acactcgattctgta<br>C1 = aaa<br>C2 - aac<br>C3 -> aag | Повідомлення про помилку<br>Невірне правило! рядок = 0 |

Через невелику кількість тестів було проведено ручне тестування, результати представлені на скріншотах(рисунок 4.1 – 4.4) та відповідають очікуваним результатам.

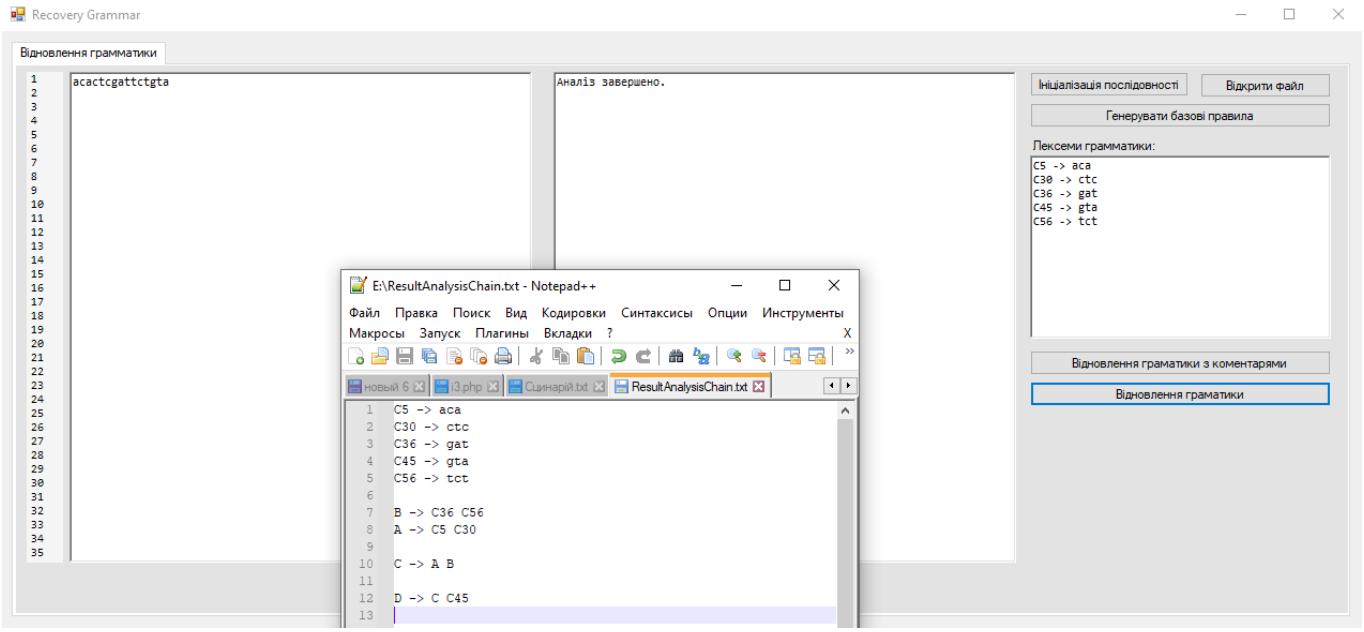


Рисунок 4.1 – Результат тесту 1

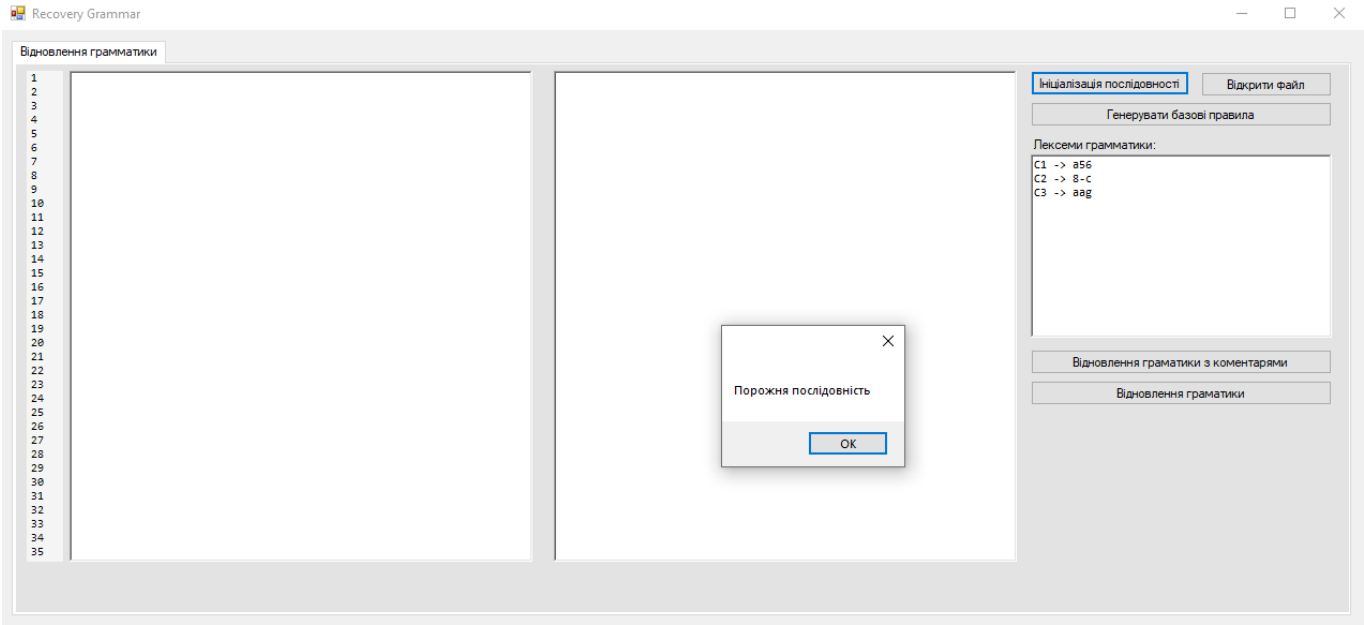


Рисунок 4.2 – Результат тесту 2

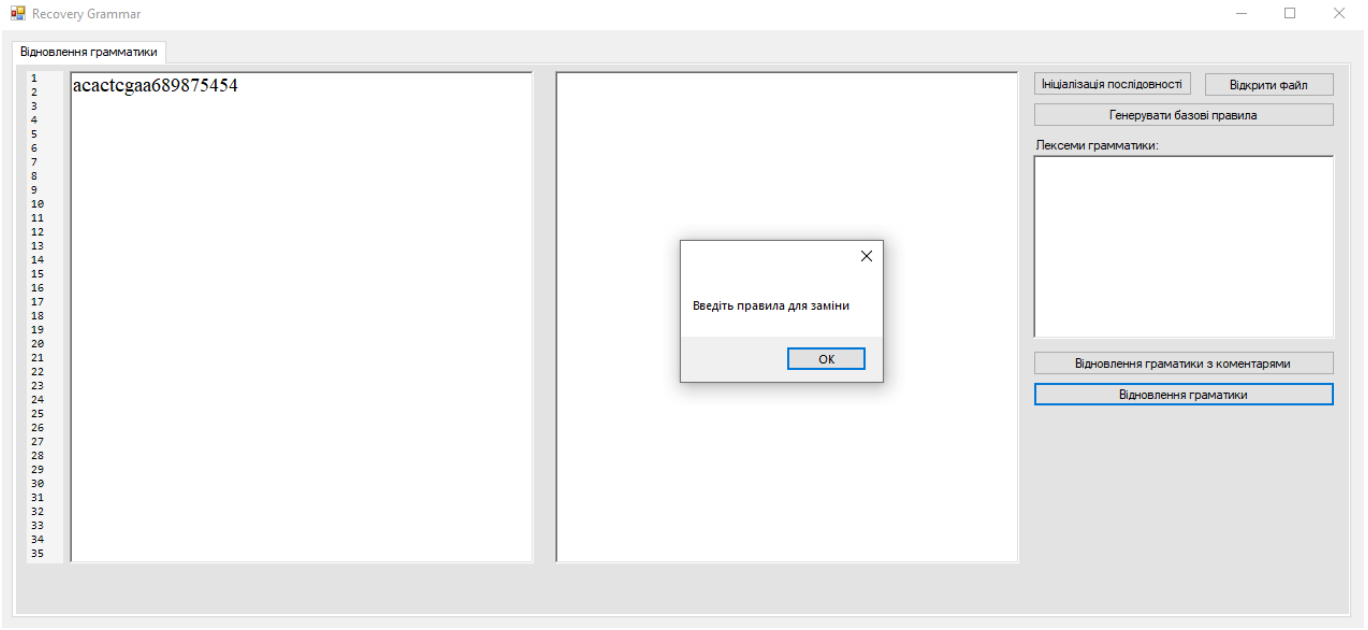


Рисунок 4.3 – Результат тесту 3

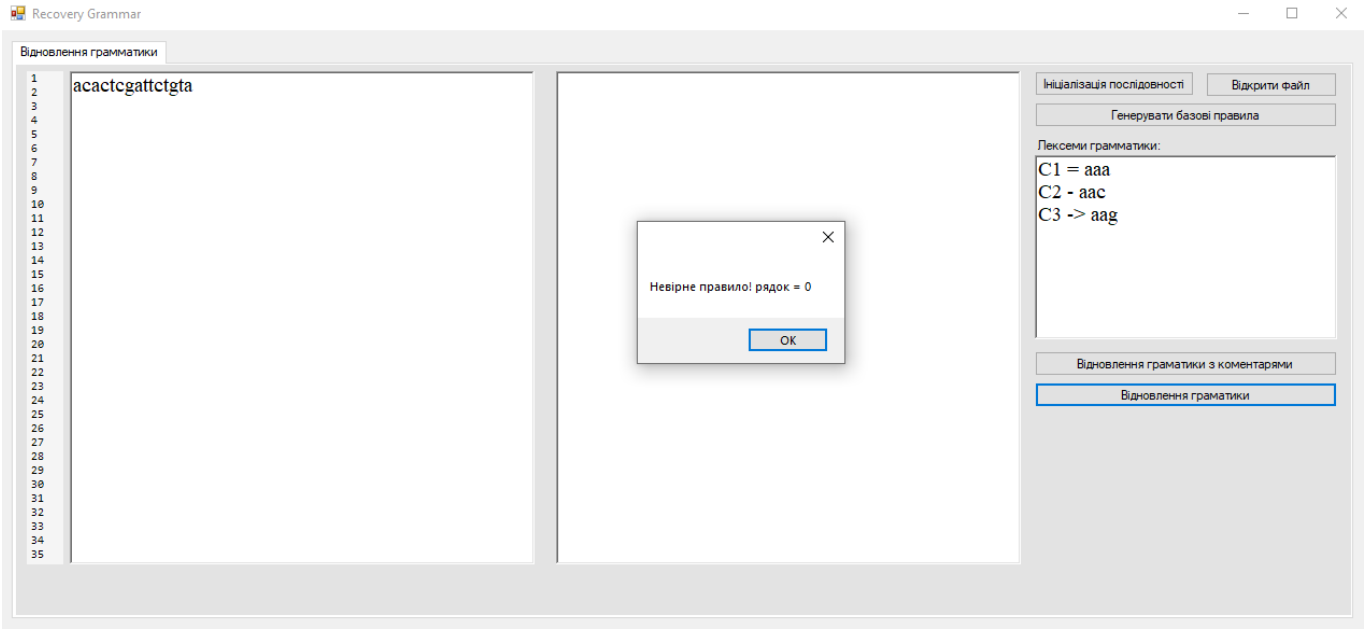


Рисунок 4.4 – Результат тесту 4

В процесі розробки програми було допущено деякі логічні, синтаксичні помилки, щоб позбутися від них було проведено налагодження. Для локалізації помилок було встановлено точки зупинки в необхідних місцях, а деякі помилки компілятор виявив самостійно.

В ході налагодження програми були виправлені наступні помилки представлені в таблиці 4.3.

Таблиця 4.3 – Протокол налагодження програми

| № з/п | Опис помилки | Ситуація, коли проявляється | Вид помилки | Дії для локалізації | Можливі дії з усунення помилки | Застосовані дії усунення помилки |
|---|---|---|---|---|---|---|
| 1 | Зайва дужка | При збірці проєкту | Синтаксична | Компілятор самостійно її виявив | Стирання зайвої дужки | Було стерто зайву дужку |
| 2 | Вихід за межі виділеної пам’яті | Під час виконання | Логічна | Встановлення точок зупинки в місцях звернення до пам’яті | Виправлення формул обчислення індексів; Обмеження кількості ітерацій в циклах | Виправлено метод знаходження індексу |
| 3 | Циклічне переміщення символів | Під час виконання | Логічна | Встановлення точок зупинки в місцях звернення до пам’яті | Виправлення умови в циклі | Виправлено умови в циклі |
| 4 | Дублювання символів | Під час виконання | Логічна | Встановлення точок зупинки в місцях звернення до пам’яті | Виправлення умов | Додано вихід з циклу в потрібний момент |

Висновки до розділу 4

Тестування ПП відбувалось методом еквівалентних розбиттів для цього було виділено правильні та неправильні класи еквівалентності, розроблено для них набори тестів та проведено тестування.

Метод еквівалентного розбиття розділяє множину вхідних значень на набори даних, тобто класи, так щоб кожний набір описував значення, принципово еквівалентні одне одному. Для кожного класу еквівалентності розроблено тест та проведено тестування. В ході розробки програми було допущено логічні, синтаксичні помилки, щоб їх виправити було проведено налагодження. Для виявлення місць з помилками було встановлено точки зупинки в необхідних місцях, а деякі помилки компілятор виявив самостійно.

Розроблені тести можуть бути використанні повторно при якихось вдосконаленнях в ПП. При збільшені кількості тестів можна застосувати автоматичне модульне тестування для більш швидкого тестування.

# ВИСНОВКИ

В рамках дипломної роботи було розроблено ПП відновлення граматик ДНК–ланцюгів, щоб досягнути цієї мети було виконано ряд важливих заходів, таких як: збір та аналіз вимог, проєктування, розробка алгоритмів, написання коду та його налагодження з тестуванням.

Під час збору вимог перш за все було визначено сферу застосування від якої залежать деякі критерії до ПП та функціональні вимоги, які встановлюють, що ПП має вміти виконувати. Так сферою застосування є медицинські дослідження направленні на виявлення нових закономірностей, порушень, мутації та структур в ДНК. Для цієї сфери є важливим скорочення часу на аналізи, фіксація однакових місць в ланцюгах з метою подальшого більш детального дослідження.

В процесі проєктування було визначено функціональне та експлуатаційне призначення, вхідні та вихідні дані, описано зовнішнє інформаційне середовище, архітектура додатку, поведінка системи.

В функціональному призначені було визначено, що головна функція ПП є відновлення граматик ДНК–ланцюгів, а далі з ними вже працюють дослідники.

В експлуатаційному призначені описано як користувач буде експлуатувати ПП.

Вхідні та вихідні дані описали, які дані від користувача очікує ПП та що користувач хоче отримати в результаті, а саме четвірку $G = (V_T, V_N, P, S)$.

Опис зовнішнього інформаційного середовища теж важливий пункт через необхідність додатку в великих обчислювальних потужностях.

В процесі проєктування архітектури системи було визначено ряд ключових моментів з урахуванням принципів розробки: з яких файлів буде складатись додаток, які класи буде містити, яка у них буде поведінка, зв’язки та за що вони будуть відповідати. Правильне проєктування дозволяє позбутися від зайвого та прискорити розробку.

Під час самої розробки було обрано мову програмування, встановлено принци розробки, обґрунтовано метод рішення та наведено приклади.

Було обрано мову C++ тому, що вона найбільше відповідає визначеним критеріям: ефективність виконуваного коду за часом, розробка об’єктних програм

мінімального розміру, мінімальний час компіляції програми тому, що це важливо як під час експлуатації так і самої розробки.

Принципи розробки впливають на швидкість написання програми, легкість її зміни, доповнення, повторного використання коду, що також зменшує затрати часу на налагодження та тестування. Тобто, від них залежить чи доведеться переписати додаток з нуля при зміні вимог.

Особливу увагу було приділено алгоритму відновлення граматик він виконує головну роль в додатку. Перед його створенням було виконано огляд існуючих рішень, взято по можливості краще від них та створено власний алгоритм. Алгоритм використовує простий підхід, беремо прості правила об’єднуємо в складні і так ітерація за ітерацією доки вся граматика не буде відновлена.

Особливістю алгоритму є унікальний підхід до побудови простих правил. Для цього використовується таблиця слідування, вона дозволяє поглинати правила, що повторюються і при цьому генерувати всі можливі правила відразу. Такий підхід відкидає необхідність використання рекурсій, робить його ітеративним та інтуїтивно зрозумілим навіть людині без специфічного досвіду та надає можливості робити алгоритм гнучким, наприклад, об’єднувати правила не тільки парами, а і трійками, четвірками, спрощувати правила, які використовуються тільки в одному місці. Тобто алгоритм дозволяє виконувати безліч налаштувань та оптимізацій, наприклад, алгоритм придатний для паралельного виконання – це надає алгоритму великої конкурентної переваги над аналогами.

## БІБЛІОГРАФІЧНИЙ СПИСОК


1 Context-free grammars: covers, normal forms, and parsing, Lecture Notes in Computer Science[Текст]: монографія / Nijholt, Anton – Springer Verlag, 1990. – 253 с.

2 Context-free recognition via shortest paths computation: a version of Valiant's algorithm[Текст]: монографія / Rytter, Wojciech : Theoretical Computer Science, 1995. – 143 с.

3 Визначення терміну SOLID [Електронний ресурс]. Матеріал з Вікіпедії. Доступна енциклопедія. – Режим доступу: https://ru.wikipedia.org/wiki/ SOLID_(объектно-ориентированное_программирование)

4 Визначення терміну ДНК [Електронний ресурс]. Матеріал з Вікіпедії. Доступна енциклопедія. – Режим доступу: https://uk.wikipedia.org/wiki/ Дезоксирибонуклеїнова_кислота

5 Приклад ДНК–ланцюгів [Електронний ресурс]. Матеріал з національної бібліотеки медицини. Відкрита бібліотека. https://www.ncbi.nlm.nih.gov/grc/human/data?asm=GRCh38

6 Формальные языки и грамматики: учбовий посібник / Соколов В.А. – Ярославль : Ярославський державний університет, 2000. – 152 с.

7 Якість програмного забезпечення та тестування [Текст]: методичні вказівки до лабораторних робіт / уклад.: В.І. Шинкаренко, О.С. Куроп'ятник, Г.В. Забула, Д.О. Пєтін, Є.В. Лукін, Дніпропетр. нац. ун-т залізн. трансп. ім. акад. В. Лазаряна. – Д.: Вид-во ПФ «Стандарт-Сервіс», 2018. – 50 с.

# ДОДАТКИ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Перший проректор Українського
державного університету
науки і технологій
Анатолій РАДКЕВИЧ
18.02.2022

# ВІДНОВЛЕННЯ ГРАМАТИК ДНК-ЛАНЦЮГІВ

Технічне завдання
ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.01252-ЛЗ

Представники
підприємства-розробника
Завідувач кафедри КІТ
Вадим ГОРЯЧКІН
18.02.2022

Керівник розробки
Віктор ШИНКАРЕНКО
18.02.2022

Виконавець
Денис ВЕДЕРНИКОВ
18.02.2022

Нормоконтролер
Олена КУРОП'ЯТНИК
18.02.2022

2022

ЗАТВЕРДЖЕНО
44165850.01252-ЛЗ

# ВІДНОВЛЕННЯ ГРАМАТИК ДНК-ЛАНЦЮГІВ

Технічне завдання

44165850.01252

Листів 13

2022

ЗМІСТ

# ВСТУП

Програмний додаток, «Відновлення граматик ДНК-ланцюгів» розробляється, щоб надати новий інструмент відновлення граматик ДНК-ланцюгів для подальшого більш детального дослідження з метою виявлення нових закономірностей в ДНК-ланцюгах. Додаток, а особливо алгоритм відновлення граматик стане фундаментальним інструментом для нових досліджень на основі отриманих ним результатів.

Розроблюваний програмний продукт призначається для дослідницьких центрів та відкриває наступні перспективні можливості:

– пошук часто повторюваних місць в ланцюгах з метою пошуку однакових механізмів в різних системах організмів;

– порівняння ДНК-ланцюгів;

– виявлення різних мутацій та порушень;

– покращення розуміння структури та принципів побудови ДНК;

– створення нових метрик та індикаторів здоров’я.

Основними перевагами додатку «Відновлення граматик ДНК-ланцюгів» є:

– скорочення часу на аналіз ДНК;

– фільтрація однакових механізмів;

– фіксація однакових місць з метою подальшого дослідження;

– зручний та простий інтерфейс.

Причина виникнення — існуючі аналоги не задовольняють функціональні потреби замовника, є платними, обмеженими та не забезпечують конфіденційність інформації, тому результати можуть бути передані третім особам.

Додаток призначений для використання в медичних та наукових закладах.

# 1. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є наказ від 08.12.21 №77 ст. ректора Українського державного університету науки і технологій «Про призначення керівників та затвердження тем бакалаврських робіт» за спеціальністю 121 «Інженерія програмного забезпечення» факультету «Комп’ютерних технологій і систем» кафедри «Комп’ютерні інформаційні технології».

# 2. ПРИЗНАЧЕННЯ РОЗРОБКИ

Функціональне призначення — програмний продукт призначається для обробки ДНК-ланцюгів з метою відновити їх граматику.

Експлуатаційне призначення — за допомогою створеного додатку працівник отримує можливість користуватися додатковим інструментом для аналізу ДНК-ланцюгів з метою отримання додаткових даних, які допоможуть виявляти нові закономірності в ДНК-ланцюгах. Порівнювати ланцюги за різні інтервали часу, виявляти різні мутації та порушення в ДНК пацієнтів. Порівнювати метрики з середніми нормами та перевіряти ключові індикатори здоров’я. Отримувати результати, ставити діагнози в короткі строки.

Додаток «Відновлення граматик ДНК-ланцюгів» має високі обчислювальні потреби, необхідне достатньо потужне обладнання для швидкого отримання результатів з операційною системою Windows, процесором з тактовою частотою від 2,4 ГГц та 8 Гб оперативної пам’яті.

# 3. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

## 3.1. Вимоги до функціональних характеристик

Вимоги до функціональних характеристик наступні:

– можливість обробляти файли великого розміру;

– можливість зберігати результати в файл;

– можливість відновлювати граматику в автоматичному режимі.

Вхідні дані: безперервний ДНК-ланцюг у вигляді послідовності азотистих основ(аденін (A), гуанін (G), тимін (T) і цитозин (C)), які об’єднані по троє і утворюють кодони. Всі дані вводяться за допомогою файлу.

Вихідні дані: файл, який містить в собі відновлену граматику ДНК-ланцюга, яка в свою чергу представлена у вигляді четвірки $G = (V_T, V_N, P, S)$, де:

– $V_T$ – алфавіт термінальних символів;

– $V_N$ – алфавіт нетермінальних символів;

– P – множина правил підстановки;

– S – початковий символ.

## 3.2. Вимоги до надійності

Надійність системи визначатиме її безпосередньою точністю виконання відновлення граматик ДНК–ланцюгів, адже недостовірність вихідних даних може призвести до фінансових збитків або помилкових результатів досліджень зроблених на основі отриманих результатів.

Вимоги до надійності наступні:

– наявність архівної копії тексту програми на зовнішньому носії;

– наявність резервної копії вхідних даних на зовнішньому носії;

– обмеженість доступу до даних;

– отримана граматика повинна бути істиною для вхідної послідовності;

– кількість помилок неповинна перевищувати одну помилку на 10 тисяч правил.

3.3. Умови експлуатації

Для забезпечення стабільного функціонування програмного продукту необхідно дотримуватись таких умов:

– допустима температура повітря від -20 до +40 град. С, допустима відносна вологість повітря — від 40% до 60%(залежить від водопроникного захисту пристрою);

– користувач має ознайомитись з керівництвом користувача;

– програмний продукт повинен використовуватись в кліматичних умовах, які підходять для стабільної роботи комп’ютера на якому виконуються обчислення;

– для повноцінного функціонування програмного продукту, комп’ютер повинен мати достатньо вільної пам'яті для збереження отриманих результатів;

– комп’ютер повинен мати достатньо вільної оперативної пам’яті для збереження проміжних обчислень;

– користувач повинен мати достатній досвід роботи з комп’ютером.

3.4. Вимоги до складу та параметрів технічних засобів

Розроблюваний програмний продукт розрахований для використання на потужному комп’ютері з операційною системою Windows та має наступні характеристики:

– процесор з частотою не нижче 2.4 ГГц;

– розрядність процесору 64 біти;

– 8 Гб оперативної пам’яті;

– 1 Гб вільної внутрішньої пам’яті.

3.5. Вимоги до інформаційної та програмної сумісності

Програмний продукт розробляється для операційної системи Windows.

Середовище розробки: Microsoft Visual Studio 2019 Community.

Мова програмування: C++.

3.6. Вимоги до маркування і упаковки

Упаковка програмного продукту, включаючи документацію повинна бути захищена від пошкоджень різного роду (механічних, кліматичних). На упаковці повинно бути вказана назва продукту, номер версії (якщо вона змінювалась),

мінімальні системні вимоги. На зворотній стороні упаковки вказується розробник та його юридична адреса.

Приклад маркування приведений на рисунку 3.1.

| | |
|---|---|
| Програмний додаток<br>«Відновлення граматик ДНК-ланцюгів»<br><br>Мінімальні системні вимоги:<br>– мінімальна версія ОС — Windows 7;<br>– процесор з частотою не нижче 2.4 ГГц;<br>– 8 Гб оперативної пам’яті;<br>– 1 Гб вільної внутрішньої пам’яті. | Розробник: Ведерников Д. С.<br>Кафедра «КІТ»,<br>УДУНТ<br>м. Дніпро, вул. Лазаряна 2<br>2022 |

Рисунок 3.1 – Приклад маркування

3.7. Вимоги до транспортування та зберігання

Транспортування повинне забезпечувати збереження програмного продукту, його цілісність і запобігання несанкціонованого доступу до нього. Програмний виріб міститься на оптичному носії даних типу CD-R і повинно мати відповідну упаковку для захисту від механічних ушкоджень та атмосферного впливу (пластиковий футляр або паперовий конверт).

## 4. ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

До складу програмної документації мають входити:

– специфікація;

– текст програми;

– опис програми;

– керівництво користувача. Керівництво з відновлення граматик ДНК.

Вся документація до програмного засобу повинна задовольняти вимогам державного стандарту до оформлення програмної документації [1].

## 5. ТЕХНІКО-ЕКОНОМІЧНІ ПОКАЗНИКИ

Додаткові затрати на розробку не передбачені. Економічні зиски не очікується.

# 6. СТАДІЇ І ЕТАПИ РОЗРОБКИ

Стадії та етапи розробки програмного продукту приведені в таблиці 6.1.

Таблиця 6.1 – Стадії та етапи розробки

| Стадія | Зміст | Строки виконання |
|---|---|---|
| Технічне завдання | Постановка задач, збір інформації, вибір та обґрунтування критеріїв розробки. Попередній вибір методів рішення задач. Визначення вимог до технічних засобів. Узгодження і затвердження технічного завдання. | 31.01.22 – 18.02.22 |
| Робочий проект | Програмування та відлагодження програми. | 19.02.22 – 01.05.22 |
| | Тестування програми. | 02.05.22 – 24.05.22 |
| | Розробка, узгодження і затвердження програмної документації. | 25.05.22 – 12.06.22 |

# 7. ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Контроль за виконання роботи здійснює керівник розробки.

Прийом здійснює комісія у складі визначеною університетом.

# БІБЛІОГРАФІЧНИЙ СПИСОК

1. Івченко, Ю.М. Основи стандартизації програмних систем [Текст]: методичні вказівки до дипломного проектування та лабораторних робіт / уклад.: Ю.М. Івченко, В.І. Шинкаренко, В.Г. Івченко; Дніпропетр. нац. ун-т залізн. трансп. ім. акад. В.Лазаряна. – Д.: Вид-во Дніпропетр. нац. ун-ту залізн. трансп. ім. акад. В. Лазаряна, 2009. - 38 с.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Перший проректор Українського
державного університету
науки і технологій
Анатолій РАДКЕВИЧ
10.06.2022

# ВІДНОВЛЕННЯ ГРАМАТИК ДНК–ЛАНЦЮГІВ

Специфікація
ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.01252-ЛЗ

Представники
підприємства-розробника
Завідувач кафедри КІТ
Вадим ГОРЯЧКІН
10.06.2022

Керівник розробки
Віктор ШИНКАРЕНКО
10.06.2022

Виконавець
Денис ВЕДЕРНИКОВ
10.06.2022

Нормоконтроллер
Олена КУРОПʼЯТНИК
10.06.2022

2022

ЗАТВЕРДЖЕНО
44165850.01252-ЛЗ

# ВІДНОВЛЕННЯ ГРАМАТИК ДНК–ЛАНЦЮГІВ

Специфікація

44165850.01252

Листів 3

2022

# 1. ДОКУМЕНТАЦІЯ

1.1. Специфікацію приведено в таблиці 1.1.

Таблиця 1.1 – Специфікація

| Позначення | Найменування | Примітка |
|---|---|---|
| 44165850.01252-ЛЗ | Лист затвердження | |
| 44165850.01252 12 01-ЛЗ | Лист затвердження | |
| 44165850.01252 12 01 | Текст програми | |
| 44165850.01252 13 01-ЛЗ | Лист затвердження | |
| 44165850.01252 13 01 | Опис програми | |
| 44165850.01252 І3 01-ЛЗ | Лист затвердження | |

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Перший проректор Українського
державного університету
науки і технологій
Анатолій РАДКЕВИЧ
10.06.22

# ВІДНОВЛЕННЯ ГРАМАТИК ДНК-ЛАНЦЮГІВ

Текст програми
ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.01252 12 01-ЛЗ

Представники
підприємства-розробника
Завідувач кафедри КІТ
Вадим ГОРЯЧКІН
10.06.2022

Керівник розробки
Віктор ШИНКАРЕНКО
10.06.2022

Виконавець
Денис ВЕДЕРНИКОВ
10.06.2022

Нормоконтролер
Олена КУРОП'ЯТНИК
10.06.2022

2022

ЗАТВЕРДЖЕНО
44165850.01252 12 01-ЛЗ

# ВІДНОВЛЕННЯ ГРАМАТИК ДНК-ЛАНЦЮГІВ

Текст програми

44165850.01252 12 01

Листів 20

2022

# АНОТАЦІЯ

Документ 44165850.01252 12 01-ЛЗ «ВІДНОВЛЕННЯ ГРАМАТИК ДНК-ЛАНЦЮГІВ». Опис програми» входить до складу програмної документації на програму, що реалізує відновлення граматик ДНК–ланцюгів.

У даному документі представлений опис програми. Додаток був розроблений на мові C++. Об'єм пам'яті, необхідний для функціонування програми, складає 1 Гб. Конфігурація – персональний комп’ютер. Програмний комплекс функціонує на платформі Windows.

ЗМІСТ

# 1. СХЕМА ВЗАЄМОДІЇ МОДУЛІВ

На рисунку 1.1 приведені схема взаємодії модулів програми.

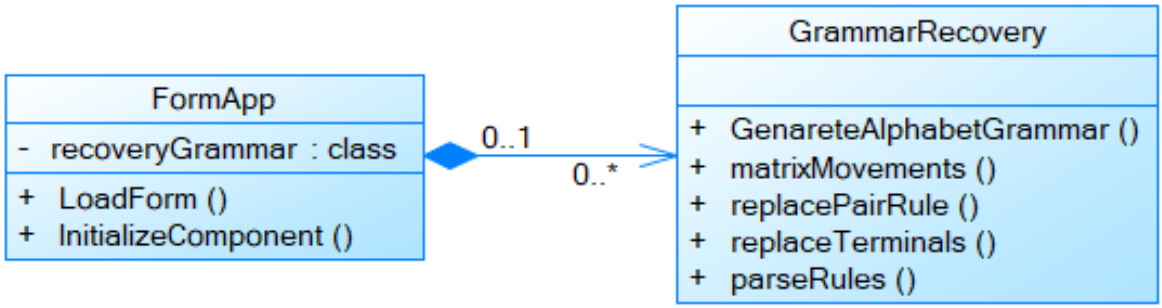


Рисунок 1.1 – Схема взаємодії модулів програми

# 2. ТЕКСТ ПРОГРАМИ

## 2.1. Модуль MyForm.h

// клас для відображення графічного інтерфейсу. Розробник Ведерников Д.С.

```
#pragma once
#define _CRT_SECURE_NO_WARNINGS

#include <fstream>
#include <windows.h>
#include <iostream>
#include "TypeChar.h"
#include "LexicalAnalysis.h"
#include "createTableAutomat.h"
#include "stateMachine.h"
#include "textRule.h"
#include "RecoveryGrammar.h"
#include "MyMethods.h"
#include <list>

HMODULE hLib;

namespace Project1 {

	using namespace System;
	using namespace System::ComponentModel;
	using namespace System::Collections;
	using namespace System::Windows::Forms;
	using namespace System::Data;
	using namespace System::Drawing;

	/// <summary>
	/// Сводка для MyForm
	/// </summary>
	public ref class MyForm : public System::Windows::Forms::Form
	{
	public:
		MyForm(void)
		{
			InitializeComponent();
			//
			//TODO: добавьте код конструктора
			//
		}

	protected:
		/// <summary>
		/// Освободить все используемые ресурсы.
		/// </summary>
		~MyForm()
		{
			if (components)
			{
				delete components;
			}
		}

	private: RecoveryGrammar* recoveryGrammar;

	private:
		/// <summary>
```

```
            /// Обязательная переменная конструктора.
            /// </summary>
            System::ComponentModel::Container^ components;

#pragma region Windows Form Designer generated code
            /// <summary>
            /// Требуемый метод для поддержки конструктора — не изменяйте
            /// содержимое этого метода с помощью редактора кода.
            /// </summary>
            void InitializeComponent(void)
            {
            }
#pragma endregion
private: System::Void MyForm_Load(System::Object^ sender, System::EventArgs^ e) {
	hLib = LoadLibrary(L"user32.dll");
	if (!hLib) {
		MessageBox::Show("Error LoadLibrary!");
	}

	recoveryGrammar = new RecoveryGrammar();
	lexicalAnalysis = new LexicalAnalysis;
	genTable = new createTableAutomat;
	machine = new stateMachine;
	machine->init();
}
private: System::Void VScroll(System::Windows::Forms::RichTextBox^ richTextBox,
System::Windows::Forms::PictureBox^ pictureBox) {
	//прокрутка номерів рядків
	if (hLib) {
		//оголошення функції
		int(__cdecl * GetScrollPos)(IntPtr, int) = (int(__cdecl*)(IntPtr,
int))GetProcAddress(hLib, "GetScrollPos");

		if (!GetScrollPos) {
			MessageBox::Show("Error load Function = " +
GetLastError().ToString());
			return;
		}
		else {
			int vPos = GetScrollPos(richTextBox->Handle, SBS_VERT);
			int start = floor(vPos / 13.0) + 1;
			int n = richTextBox->Height / 13;

			Color^ clr = gcnew Color();
			Brush^ br = gcnew SolidBrush(clr->Black);

			System::Drawing::Graphics^ draw = pictureBox->CreateGraphics();
			draw->Clear(clr->WhiteSmoke);

			for (int i = 0; i <= n; i++) {
				draw->DrawString((i + start).ToString(), richTextBox->Font, br,
2, i * 13);
			}
		}
	}
}
private: System::Void pictureBox2_Paint(System::Object^ sender,
System::Windows::Forms::PaintEventArgs^ e) {
	//початкове рисування номерів рядків
	Color^ clr = gcnew Color();
	Brush^ br = gcnew SolidBrush(clr->Black);

	System::Drawing::Graphics^ draw = e->Graphics;
	draw->Clear(clr->WhiteSmoke);
```

```
	int n = richTextBox3->Height / 13;

	for (int i = 0; i <= n; i++) {
		draw->DrawString((i + 1).ToString(), richTextBox3->Font, br, 2, i * 13);
	}
}
private: System::Void richTextBox3_VScroll(System::Object^ sender, System::EventArgs^ e) {
	VScroll(richTextBox3, pictureBox2);
}
private: System::Void button16_Click(System::Object^ sender, System::EventArgs^ e) {
	//відкритя файлу

	if (openFileDialog1->ShowDialog() == System::Windows::Forms::DialogResult::OK) {
		recoveryGrammar->txt = (unsigned
char*)my::StringToChar(System::IO::File::ReadAllText(openFileDialog1->FileName,
System::Text::Encoding::UTF8));

		if (strlen((char *)recoveryGrammar->txt) > 10000) {
			richTextBox8->Text =
my::charToString((char*)my::copySubString(recoveryGrammar->txt, 0, 10000));
		}
		else {
			richTextBox8->Text = my::charToString((char *)recoveryGrammar->txt);
		}
	}
}
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e) {
	//ініціалізація послідовності
	if (richTextBox8->Text == "") {
		System::Windows::Forms::MessageBox::Show("Порожня послідовність");
	}
	else {
		recoveryGrammar->txt = (unsigned char*)my::StringToChar(richTextBox8->Text);
	}
}
private: System::Void button17_Click(System::Object^ sender, System::EventArgs^ e) {
	//генерувати правила

	std::string result = "";

	std::vector<std::string> codone;
	codone.push_back("a");
	codone.push_back("c");
	codone.push_back("g");
	codone.push_back("t");

	int num = 1;

	for (int i = 0; i < codone.size(); i++) {
		for (int j = 0; j < codone.size(); j++) {
			for (int k = 0; k < codone.size(); k++) {
				result += "C";
				result += my::StringToChar(num.ToString());
				result += " -> ";
				result += codone[i] + codone[j] + codone[k] + "\n";

				num++;
			}
		}
	}

	richTextBox10->Text = my::charToString(result.c_str());
}
private: System::Void button23_Click(System::Object^ sender, System::EventArgs^ e) {
	//повний цикл
```

```
	if (recoveryGrammar->txt != nullptr) {
		if (richTextBox10->Text != "") {

			std::ofstream fout;
			fout.open("E:/ResultAnalysisChain.txt", std::ios::trunc);

			auto rules = recoveryGrammar-
>parseRules(my::StringToChar(richTextBox10->Text));

			fout << "Розібрано базові правила:" << std::endl;

			int lenNewRules = rules.size();

			for (int i = 0; i < lenNewRules; i++) {
				fout << rules[i].first << " -> " << rules[i].second <<
std::endl;
			}

			auto chain = recoveryGrammar->replaceTerminals((char*)recoveryGrammar-
>txt, rules);

			fout << std::endl << std::endl << "Замінено всі термінали на правила."
<< std::endl << "S -> ";

			int lenChain = chain.size();

			for (int i = 0; i < lenChain; i++) {
				fout << chain[i] << " ";
			}

			fout << std::endl;

			int iteration = 1, countRules = 0;

			auto lengthChain = chain.size();

			while (chain.size() > 1) {
				fout << std::endl << std::endl << "ІТЕРАЦІЯ: " << iteration <<
std::endl;

				auto newRules = recoveryGrammar->matrixMovements(countRules,
chain);

				fout << std::endl << "Побудовано нові правила:" << std::endl;

				lenNewRules = newRules.size();

				for (int i = 0; i < lenNewRules; i++) {
					fout << newRules[i].first << " -> " <<
newRules[i].second.first << " " << newRules[i].second.second << std::endl;
				}

				auto result = recoveryGrammar->replacePairRule(chain,
newRules);

				fout << std::endl << "Замінено пари правил на нові:" <<
std::endl << "S -> ";

				lenChain = result.first.size();
				chain = result.first;

				for (int i = 0; i < lenChain; i++) {
					fout << chain[i] << " ";
				}
```

```
                        fout << std::endl << std::endl << "Використані правила:" <<
std::endl;

                        lenNewRules = result.second.size();
                        newRules = result.second;

                        for (int i = 0; i < lenNewRules; i++) {
                                fout << newRules[i].first << " -> " <<
newRules[i].second.first << " " << newRules[i].second.second << std::endl;
                        }

                        iteration++;
                }

                fout << std::endl << "Аналіз завершено." << std::endl;
                fout.close();

                richTextBox9->Text = "Аналіз завершено.";
            }
            else {
                System::Windows::Forms::MessageBox::Show("Введіть правила для
заміни");
            }
        }
        else {
            System::Windows::Forms::MessageBox::Show("Спочатку ініціалізуйте
послідовність для аналізу");
        }
}
private: System::Void button24_Click(System::Object^ sender, System::EventArgs^ e) {
        //повний цикл оптимізований

        if (recoveryGrammar->txt != nullptr) {
            if (richTextBox10->Text != "") {

                std::ofstream fout;
                fout.open("E:/ResultAnalysisChain.txt", std::ios::trunc);

                auto rules = recoveryGrammar-
>parseRules(my::StringToChar(richTextBox10->Text));

                int lenNewRules = rules.size();

                for (int i = 0; i < lenNewRules; i++) {
                        fout << rules[i].first << " -> " << rules[i].second <<
std::endl;
                }

                auto chain = recoveryGrammar->replaceTerminals((char*)recoveryGrammar-
>txt, rules);

                int lenChain = chain.size();

                int iteration = 1, countRules = 0;

                auto lengthChain = chain.size();

                while (chain.size() > 1) {
                        auto result = recoveryGrammar->replacePairRule(countRules,
chain);

                        lenChain = result.first.size();
                        chain = result.first;
```

```
                        lenNewRules = result.second.size();
                        auto newRules = result.second;

                        fout << std::endl<< std::endl;

                        for (int i = 0; i < lenNewRules; i++) {
                                fout << newRules[i].first << " -> " <<
newRules[i].second.first << " " << newRules[i].second.second << std::endl;
                        }

                        iteration++;
                }

                fout.close();

                richTextBox9->Text = "Аналіз завершено.";
        }
        else {
                System::Windows::Forms::MessageBox::Show("Введіть правила для
заміни");
        }
    }
    else {
        System::Windows::Forms::MessageBox::Show("Спочатку ініціалізуйте
послідовність для аналізу");
    }
}
};
}
```

## 2.2. Модуль RecoveryGrammar.cpp

// містить всі ключові методи для відновлення граматики. Розробник Ведерников Д.С.

```
#include "RecoveryGrammar.h"
#include "MyMethods.h"

void RecoveryGrammar::GenareteAlphabetGrammar() {
    grammarAlphabet.clear();

    int* lexems = new int[256]{ 0 };

    auto len = std::strlen((char*)txt);

    for (int i = 0; i < len; i++) {
        auto symbol = txt[i];

        lexems[symbol] = lexems[symbol] + 1;
    }

    for (int i = 0; i < 256; i++) {
        if (lexems[i] > 0) {
            grammarAlphabet[i] = lexems[i];
        }
    }
}

std::map<std::string, int> RecoveryGrammar::frequencyLexems(std::vector<std::string>
&listLexems) {
    std::map<std::string, int> result;
```

```
	auto len = std::strlen((char*)txt);
	int countLexems = listLexems.size();

	for (int pos = 0; pos < len; pos++) {

		for (int lex = 0; lex < countLexems; lex++) {
			std::string txtLexem = listLexems[lex];

			int lenLex = txtLexem.length();

			if (lenLex <= len - pos){

				int k = 0;
				bool stop = false;

				for (int i = 0; i < lenLex; i++) {

					if (txt[pos + k] == '\n') {
						while (txt[pos + k] == '\n') k++;
					}

					if (txt[pos + k] != txtLexem[i]) {
						stop = true;
						break;
					}

					k++;
				}

				if (!stop) {
					result[txtLexem] = result[txtLexem] + 1;
					pos += lenLex - 1;
					break;
				}
			}
		}
	}

	return result;
}

std::map<std::string, int> RecoveryGrammar::frequencyLexems(char** listLexems) {
	std::map<std::string, int> result;

	auto len = std::strlen((char*)txt);
	int countLexems = _msize(listLexems)/sizeof(char*);

	int* counts = new int[countLexems] {0};

	for (int pos = 0; pos < len; pos++) {

		for (int lex = 0; lex < countLexems; lex++) {
			char* txtLexem = listLexems[lex];

			int lenLex = _msize(txtLexem) - 1;

			if (lenLex <= len - pos){

				int k = 0;
				bool stop = false;

				for (int i = 0; i < lenLex; i++) {

					if (txt[pos + k] == '\n') {
						while (txt[pos + k] == '\n') k++;
```

```
                                }

                                if (txt[pos + k] != txtLexem[i]) {
                                        stop = true;
                                        break;
                                }

                                k++;
                        }

                        if (!stop) {
                                counts[lex] = counts[lex] + 1;
                                pos += lenLex - 1;
                                break;
                        }
                }
        }
    }

    for (int lex = 0; lex < countLexems; lex++) {
        result[listLexems[lex]] = counts[lex];
    }

    return result;
}

std::vector<std::pair<std::string, std::pair<std::string, std::string>>>
RecoveryGrammar::matrixMovements(int &num, std::vector<std::string> &listLexems) {
    std::pair<std::string, std::string> pair;
    std::vector<std::pair<std::string, std::pair<std::string, std::string>>> result;

    int lenListLexemes = listLexems.size() - 1;

    for (int i = 0; i < lenListLexemes; i++) {
        pair.first = listLexems[i];
        pair.second = listLexems[i + 1];

        if (my::searchInVector(result, pair) == -1) {
            auto ff = std::make_pair(listLexems[i], listLexems[i + 1]);
            result.push_back(std::make_pair(my::IntToABCString(num), ff));
            num++;
        }
    }

    return result;
}

std::pair<std::vector<std::string>, std::vector<std::pair<std::string,
std::pair<std::string, std::string>>>>
RecoveryGrammar::replacePairRule(std::vector<std::string>& chain,
std::vector<std::pair<std::string, std::pair<std::string, std::string>>>& rules) {
    std::vector<std::string> result;
    std::pair<std::string, std::string> pair;

    int idx, i = 0, lenChain = chain.size(), lenRules = rules.size();

    bool* isUseRules = new bool[lenRules] {false};

    for (; i < lenChain - 1; i++) {
        pair.first = chain[i];
        pair.second = chain[i + 1];

        idx = my::searchInVector(rules, pair);

        if (idx > -1) {
```

```
                        isUseRules[idx] = true;

                        result.push_back(rules[idx].first);
                        i++;
                }
        }

        if (i == lenChain - 1) {
                result.push_back(chain[lenChain - 1]);
        }

        std::vector<std::pair<std::string, std::pair<std::string, std::string>>> useRules;

        for (int i = 0; i < lenRules; i++) {
                if (isUseRules[i]) {
                        useRules.push_back(rules[i]);
                }
        }

        return std::make_pair(result, useRules);
}

std::pair<std::vector<std::string>, std::vector<std::pair<std::string,
std::pair<std::string, std::string>>>> RecoveryGrammar::replacePairRule(int& num,
std::vector<std::string>& chain) {
        std::vector<std::string> result;
        std::pair<std::string, std::string> pair;
        std::map<std::pair<std::string, std::string>, std::string> useRules;

        int i = 0, lenChain = chain.size();

        for (; i < lenChain - 1; i++) {
                pair.first = chain[i];
                pair.second = chain[i + 1];

                auto idx = useRules.find(pair);

                if (idx != useRules.end()) {
                        result.push_back((*idx).second);
                }else{
                        std::string name = my::IntToABCString(num);

                        useRules[std::make_pair(pair.first, pair.second)] = name;
                        num++;

                        result.push_back(name);
                }

                i++;
        }

        if (i == lenChain - 1) {
                result.push_back(chain[lenChain - 1]);
        }

        std::vector<std::pair<std::string, std::pair<std::string, std::string>>> useRules2;

        for (auto j = useRules.begin(); j != useRules.end(); j++) {
                useRules2.push_back(std::make_pair((*j).second, (*j).first));
        }

        return std::make_pair(result, useRules2);
}
```

```
std::vector<std::string> RecoveryGrammar::replaceTerminals(char *txt,
std::vector<std::pair<std::string, std::string>> &rules) {
	std::vector<std::string> result;

	auto len = std::strlen((char*)txt);
	int countLexems = rules.size();

	for (int pos = 0; pos < len; pos++) {

		for (int lex = 0; lex < countLexems; lex++) {
			//текст терміналу
			std::string txtLexem = rules[lex].second;

			int lenLex = txtLexem.length();

			if (lenLex <= len - pos) {

				int k = 0;
				bool stop = false;

				for (int i = 0; i < lenLex; i++) {

					if (txt[pos + k] == '\n') {
						while (txt[pos + k] == '\n') k++;
					}

					if (txt[pos + k] != txtLexem[i]) {
						stop = true;
						break;
					}

					k++;
				}

				if (!stop) {
					result.push_back(rules[lex].first);
					pos += lenLex - 1;
					break;
				}
			}
		}
	}

	return result;
}

std::vector<std::pair<std::string, std::string>> RecoveryGrammar::parseRules(char* text) {
	std::vector<std::pair<std::string, std::string>> result;

	char* split = new char[7];
	split[0] = '\n';
	split[1] = '\0';

	//отримання рядків
	std::vector<std::string> tmp, lines = my::explode(text, split, 0, strlen(text));

	strcpy(split, " -> ");
	split[4] = '\0';

	for (int i = 0; i < lines.size(); i++) {
		if (lines[i] != "") {
			//отримання пар із рядків ->
			tmp = my::explode(lines[i].c_str(), split, 0,
strlen(lines[i].c_str()));
```

```
                        if (tmp.size() == 2) {
                                result.push_back(std::make_pair(tmp[0], tmp[1]));
                        }
                        else {
                                System::Windows::Forms::MessageBox::Show("Невірне правило!
рядок = " + i.ToString());

                                result.clear();
                                break;
                        }
                }
        }

        return result;
}
```

## 2.3. Модуль MyMethods.cpp

// містить код для роботи з векторними даними. Розробник Ведерников Д.С.

```
#include <vector>
#include "textRule.h"
#include "MyMethods.h"

//пошук в векторі
int my::searchInVector(std::vector<std::pair<textRule, std::vector<textRule>>>& vec,
std::string text) {
        for (int i = 0; i < vec.size(); i++) {
                if (vec[i].first.text == text) {
                        return i;
                }
        }

        return -1;
}

//пошук в векторі
int my::searchInVector(std::vector<std::pair<std::string, std::vector<std::string>>>& vec,
std::string text) {
        for (int i = 0; i < vec.size(); i++) {
                if (vec[i].first == text) {
                        return i;
                }
        }

        return -1;
}

//пошук в векторі
int my::searchInVector(std::vector<std::pair<std::string, std::string>>& vec, std::string
text) {
        for (int i = 0; i < vec.size(); i++) {
                if (vec[i].first == text) {
                        return i;
                }
        }

        return -1;
}

//пошук в векторі
```

```
int my::searchInVector(std::vector<std::pair< std::string, bool>>& vec, std::string text)
{
	for (int i = 0; i < vec.size(); i++) {
		if (vec[i].first == text) {
			return i;
		}
	}

	return -1;
}

//пошук в векторі
int my::searchInVector(std::vector<std::pair< std::string, std::pair<bool, bool>>>& vec,
std::string text) {
	for (int i = 0; i < vec.size(); i++) {
		if (vec[i].first == text) {
			return i;
		}
	}

	return -1;
}

//пошук в векторі
int my::searchInVector(std::vector<std::pair< std::string, std::vector<
std::pair<std::string, std::pair<bool, bool> > > > >& vec, std::string text) {
	for (int i = 0; i < vec.size(); i++) {
		if (vec[i].first == text) {
			return i;
		}
	}

	return -1;
}

//пошук в векторі
int my::searchInVector(std::vector<std::pair<std::string, std::pair<std::string,
std::string>>>&vec, std::pair<std::string, std::string> pair) {
	for (int i = 0; i < vec.size(); i++) {
		if (vec[i].second.first == pair.first && vec[i].second.second == pair.second)
{
			return i;
		}
	}

	return -1;
}

std::string my::IntToABCString(int n)
{
	int part1, part2;
	std::string result = "";

	if (n > 25)
	{
		do
		{
			part1 = n / 26;
			part2 = n - part1 * 26;
			n = part1;

			result += (char)(part2 + 65);

			if (part1 < 25)
			{
```

```
                        result += (char)(part1 + 65);
                        break;
                    }

                } while (part1 > 25);
            }
            else
            {
                result += (char)(n + 65);
            }

            std::reverse(result.begin(), result.end());

            return result;
}

char* my::StringToChar(System::String^ txt) {
            return
(char*)(void*)System::Runtime::InteropServices::Marshal::StringToHGlobalAnsi(txt);
}

//конвертування даних з масива символів char* в String^
System::String^ my::charToString(const char* str) {
            return
System::Runtime::InteropServices::Marshal::PtrToStringAnsi((System::IntPtr)(void*)str);
}

//в нижній регістр
void my::toSmall(unsigned char* str, int start, int end) {
            for (int i = start; i <= end; i++) {
                        if (str[i] >= 65 && str[i] <= 90) {
                                    str[i] += 32;
                        }
                        else if (str[i] >= 192 && str[i] <= 223) {
                                    str[i] += 32;
                        }// Ґ
                        else if (str[i] == 165) {
                                    str[i] = 180;
                        }// Ё
                        else if (str[i] == 168) {
                                    str[i] = 184;
                        }// Ї
                        else if (str[i] == 175) {
                                    str[i] = 191;
                        }// І
                        else if (str[i] == 178) {
                                    str[i] = 179;
                        }
            }
}

//в верхній регістр
void my::toLarge(unsigned char* str, int start, int end) {
            for (int i = start; i <= end; i++) {
                        if (str[i] >= 97 && str[i] <= 122) {
                                    str[i] -= 32;
                        }
                        else if (str[i] >= 224 && str[i] <= 255) {
                                    str[i] -= 32;
                        }// ґ
                        else if (str[i] == 180) {
                                    str[i] = 165;
                        }// ё
                        else if (str[i] == 184) {
                                    str[i] = 168;
```

```
			}// ї
			else if (str[i] == 191) {
				str[i] = 175;
			}// і
			else if (str[i] == 179) {
				str[i] = 178;
			}
		}
}

//копіювання підрядок
unsigned char* my::copySubString(unsigned char* str, int start, int end) {
		unsigned char* newStr = new unsigned char[end - start + 2];
		if (newStr) {
			for (int count = 0, i = start; i <= end; i++, count++) {
				newStr[count] = str[i];
			}
		}
		newStr[end - start + 1] = '\0';
		return newStr;
}

//пошук підрядка
Result my::isSubString(unsigned char* str1, int idx, int n1, unsigned char* str2, int n2)
{
		//поточний індекс
		int index = idx;

		int count = 0;
		bool res = true;

		if (n1 - idx == n2) {
			for (; count < n2 && index < n1; index++, count++) {
				if (str1[index] != str2[count]) {
					res = false;
					break;
				}
			}

			if (count != n2) {
				index = idx;
				res = false;
			}
		}
		else {
			res = false;
		}

		return Result(index, res);
}

//пошук в рядку
int my::strfind(const char* src, const char* find, int start, int stop) {
		int findlen = strlen(find);
		for (int i = start; i < stop; i++) {
			if (src[i] == find[0]) {
				for (int k = i, d = 0; (d <= findlen && k <= stop); k++, d++) {
					if (d == findlen) {
						return i;
					}
					if (src[k] != find[d]) {
						break;
					}
				}
			}
```

```
	}

	return -1;
}

//копіювання частини рядку
char* my::copySubString(const char* str, int start, int end) {
	char* newStr = new char[end - start + 2];
	if (newStr) {
		for (int count = 0, i = start; i <= end; i++, count++) {
			newStr[count] = str[i];
		}
	}

	newStr[end - start + 1] = '\0';

	return newStr;
}

//розділення рядку
std::vector<std::string> my::explode(const char* str, const char* metka, int start, int
end) {
	std::vector<std::string> headers;

	int posMetka = start;
	int lastPosMetka, end2;

	char* part;

	int len = strlen(metka);

	do {
		lastPosMetka = posMetka;
		posMetka = strfind(str, metka, lastPosMetka, end);

		if (posMetka > -1) {
			end2 = posMetka - 1;
		}
		else {
			end2 = end;
		}

		part = copySubString(str, lastPosMetka, end2);
		posMetka += len;

		headers.push_back(part);

		delete[] part;
	} while (posMetka > len - 1);

	return headers;
}
```

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Перший проректор Українського
державного університету
науки і технологій
Анатолій РАДКЕВИЧ
10.06.2022

# ВІДНОВЛЕННЯ ГРАМАТИК ДНК–ЛАНЦЮГІВ

Опис програми
ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.01252 13 01-ЛЗ

Представники
підприємства-розробника
Завідувач кафедри КІТ
Вадим ГОРЯЧКІН
10.06.2022

Керівник розробки
Віктор ШИНКАРЕНКО
10.06.2022

Виконавець
Денис ВЕДЕРНИКОВ
10.06.2022

Нормоконтроллер
Олена КУРОП'ЯТНИК
10.06.2022

2022

ЗАТВЕРДЖЕНО
44165850.01252 13 01-ЛЗ

# ВІДНОВЛЕННЯ ГРАМАТИК ДНК–ЛАНЦЮГІВ

Опис програми

44165850.01252 13 01

Листів 16

2022

# АНОТАЦІЯ

Документ 44165850.01252 13 01-ЛЗ «Відновлення граматик ДНК–ланцюгів. Опис програми» входить до складу програмної документації на програму, що реалізує відновлення граматик ДНК–ланцюгів.

У даному документі представлений опис програми. Додаток був розроблений на мові C++. Об'єм пам'яті, необхідний для функціонування програми, складає 1 Гб. Конфігурація – персональний комп’ютер. Програмний комплекс функціонує на платформі Windows.

ЗМІСТ

# 1. ЗАГАЛЬНІ ВІДОМОСТІ

Програмний продукт «Відновлення граматик ДНК–ланцюгів» інструмент, який призначений для відновлення граматик ДНК–ланцюгів.

Для функціонування додатку необхідно, що на персональному комп’ютері було встановлено операційну систему Windows.

Програма реалізована на мові програмування C++ в середовищі MS Visual Studio 2019.

# 2. ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Даний програмний продукт призначається для обробки ДНК-ланцюгів з метою відновити їх граматику. Далі на основі отриманих ним результатів будуть проводитись інші дослідження.

Обмеженням програми є не можливість виконати обчислення без достатньої кількості вільної оперативної пам’яті.

# 3. ОПИС ЛОГІЧНОЇ СТРУКТУРИ

## 3.1. Алгоритм програми

Головний принцип алгоритму від простого до складного, тобто об’єднання простих правил в більш складні з виключенням повторень. Розглянемо та обґрунтуємо принцип побудови правил.

Нехай дано послідовність: gcccatcagtcctctgagacaggt.

Тоді побудуємо можливі переходи, тобто заповнимо таблицю слідувань. Такий підхід дозволяє побудувати всі можливі переходи і головне граматичні конструкції не дублюються, тобто повторення поглинаються – такий підхід дуже ефективний при великій кількості повторень фрагментів в послідовності.

Таблиця 3.1 – Таблиця слідування

| | a | c | g | t |
|---|---|---|---|---|
| a | | 1 | 1 | 1 |
| c | 1 | 1 | | 1 |
| g | 1 | 1 | 1 | 1 |
| t | | 1 | 1 | |

Побудуємо правила по таблиці:

P1 → ac

P2 → ag

P3 → at

P4 → ca

P5 → cc

P6 → ct

P7 → ga

P8 → gc

P9 → gg

P10 → gt

P11 → tc

P12 → tg

З отриманих правил формується нова послідовність і над нею повторно виконуються аналогічні дії доки не залишиться остання пара правил.

Алгоритм позбавлений від рекурсивних викликів, тобто він ітеративний – це означає, що алгоритм має скінчену кількість ітерації, зрозумілу обчислювальну складність та завжди відновлює граматику.

Отримані правила схожі на бінарне дерево, але не являються ним, тому що правила розміщуються в ньому по іншому принципу, описаному раніше та вершини об’єднують прості правила в більш складні, тобто структура однакова, а принципи інші.

3.2. Використані методи

Використано наступні методи:

– METHOD [1]: «GenareteAlphabetGrammar» – генерує алфавіт для послідовності;

– METHOD [2]: «matrixMovements» – будує таблицю слідування;

– METHOD [3]: «replacePairRule» – об’єднує пари правил в нові правила;

– METHOD [4]: «replaceTerminals» – здійснює заміну терміналів на правила;

– METHOD [5]: «parseRules» – здійсює розбір правил.

3.3. Структура програми

Програмні модулі виконують наступні функції:

– MyForm.h – клас для відображення графічного інтерфейсу;

– RecoveryGrammar.cpp – головний клас, який містить всі ключові методи для відновлення граматики;

– MyMethods.cpp – містить код для роботи з векторними даними.

3.4. Зв’язки програми з іншими програмами

Програма немає ніяких зв’язків з іншими програмами.

# 4. ВИКОРИСТАНІ ТЕХНІЧНІ ЗАСОБИ

Розроблюваний програмний продукт розрахований для використання на потужному комп’ютері з операційною системою Windows, який має наступні характеристики:

– процесор з частотою не нижче 2.4 ГГц;

– розрядність процесору 64 біти;

– 8 Гб оперативної пам’яті;

– 1 Гб вільної внутрішньої пам’яті;

– наявність CD-R дисководу.

Для початку роботи з програмою непотрібно здійснювати додаткові налаштування.

# 5. ВИКЛИК І ЗАВАНТАЖЕННЯ

Для користування програмним продуктом необхідно:

– завантажити проєкт з диску на комп’ютер;

– встановити додаток на комп’ютер;

– відкрити єдиний файл додатку.

Об’єм програми складає 1 Мб. Програма може бути виконуватися лише у середовищі Windows.

Після завантаження програми з’являється головна форма, яка зображена на рисунку 5.1.


Рисунок 5.1 – Головна форма програми

## 6. ВХІДНІ ДАНІ

Вхідні дані: безперервний ДНК–ланцюг у вигляді послідовності азотистих основ(аденін (A), гуанін (G), тимін (T) і цитозин (C)), які об’єднані по троє і утворюють кодони. Всі дані вводяться за допомогою системної клавіатури та діалогового вікна.

# 7. ВИХІДНІ ДАНІ

Вихідними даними програми є текстовий файл, який містить в собі відновлену граматику ДНК–ланцюга, яка в свою чергу представлена у вигляді четвірки $G = (V_T, V_N, P, S)$, де:

- $V_T$ – алфавіт термінальних символів;
- $V_N$ – алфавіт нетермінальних символів;
- P – множина правил підстановки;
- S – початковий символ.

# 8. ОПИС ІНТЕРФЕЙСУ КОРИСТУВАЧА

Користувач має можливість інтуїтивно та зручно працювати з програмою. Після запуску програми «ВІДНОВЛЕННЯ ГРАМАТИК ДНК–ЛАНЦЮГІВ» з’являється графічний інтерфейс додатку представлений на рисунку 8.1.


Рисунок 8.1 – Головна форма

Головна форма дозволяє ввести та ініціалізувати послідовність ДНК–ланцюга з клавіатури або відкривши файл, генерувати базові правила, приведено на рисунку 8.2.

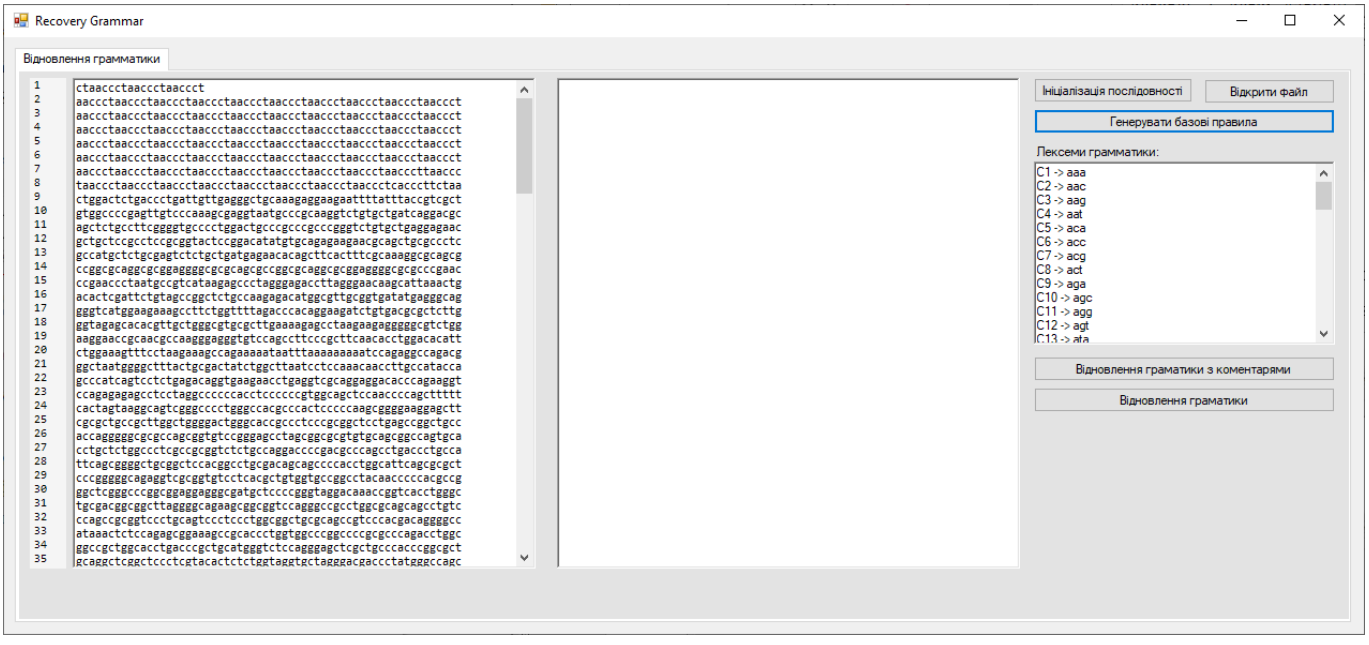


Рисунок 8.2 – Введено послідовність та базові правила

Результати відновлення граматики представлено на рисунку 8.3 та 8.4.

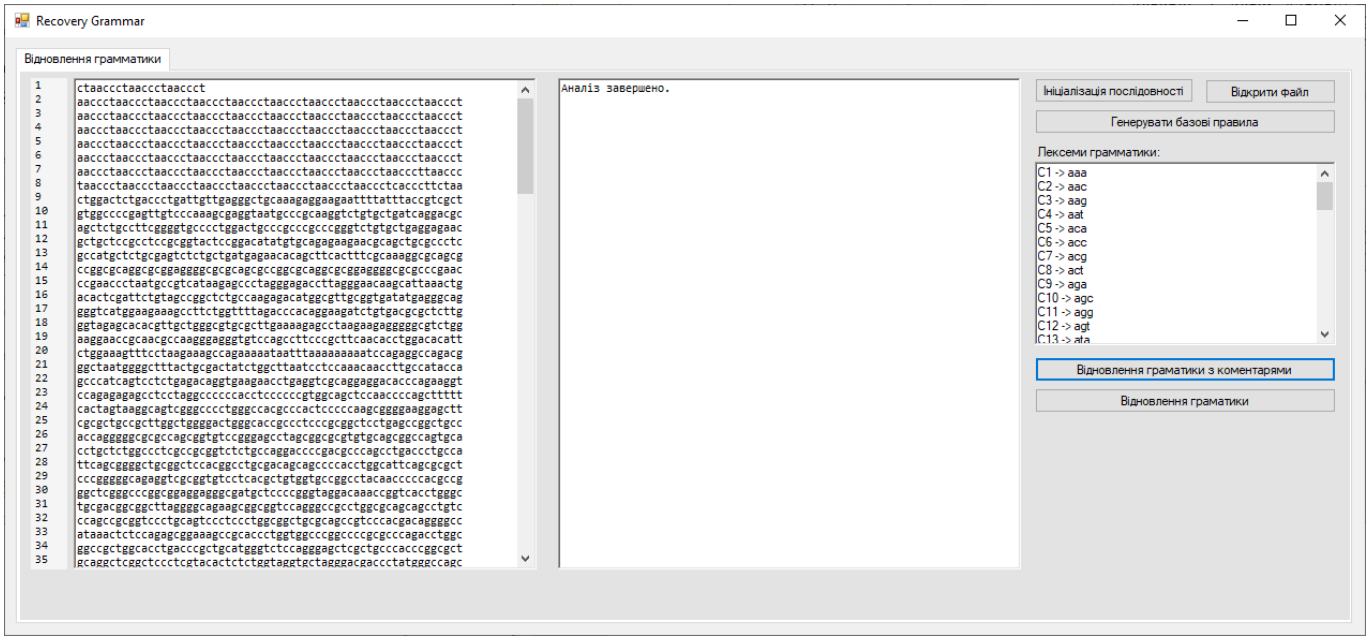


Рисунок 8.3 – Результат відновлення граматики на формі

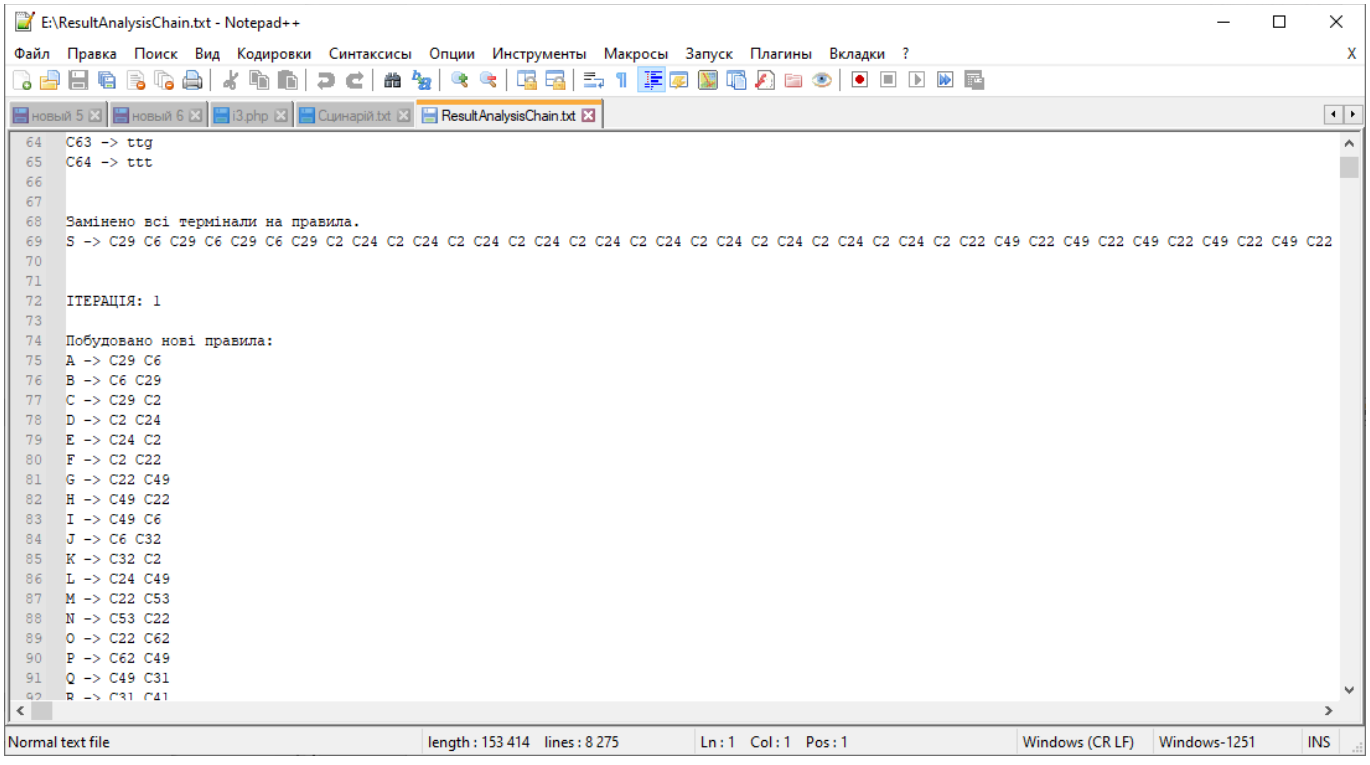


Рисунок 8.4 – Результат відновлення граматики

# 9. ПОРЯДОК РОБОТИ З ПРОГРАМОЮ

Перед початком роботи користувачеві необхідно перевірити персональний комп’ютер в справності та відповідності вимогам. Після чого можна перейти запуску додатку.

1. Запуск додатку (користувач) 09:00 — 09:01.
2. Відкриття файлу з ДНК послідовністю (користувач) 09:01 — 09:02.
3. Генерування базових правил (користувач) 09:02 — 09:03.
4. Відновлення граматики (користувач) 09:03 — 09:18.
5. Вихід з програми (користувач) 09:18 — 09:19.

# 10. ПОВІДОМЛЕННЯ

В таблиці 10.1 приведені повідомлення користувачу.

Таблиця 10.1 – Повідомлення користувачу

| Текст повідомлення | Опис ситуації | Рекомендовані дії |
|---|---|---|
| Аналіз завершено | Програма завершила аналіз | Продовжити роботу з програмою |
| Спочатку ініціалізуйте послідовність для аналізу | Спроба виконати якусь дію перед ініціалізацією послідовності | Ввести і ініціалізувати послідовність |
| Порожня послідовність | Спроба ініціалізувати порожню послідовність | Ввести послідовність |
| Введіть правила для заміни | Спроба відновити граматику без введення базових правил | Ввести базову граматику |