# ВТУП

Обмін інформацією є однією з центральних ланок в роботі інформаційних систем. До інформації в таких системах ставляться наступні вимоги: об’єктивність, достовірність, повнота, точність, актуальність, корисність.

При передачі інформації через канали з’єднання або при збереженні на фізичні носії є ймовірність її пошкодження, що впливає на достовірність інформації. Через це інформація потребує захисту від помилок та фізичних вад каналів передачі або носіїв. Є багато різноманітних способів знаходження та виправлення помилок. При передачі інформації через мережу інтернет за допомогою протоколу TCP/IP вона поділяється на пакети, де кожен пакет на прикінці має хеш суму інформаційної частки пакету. В разі коли обчислена хеш сума пакету не співпадає з бажаною, іде повторна пересилка повідомлення. Тим самим користувач уникає помилок в отриманої інформації.

Але що робити коли повторна пересилка пакета неможлива або є надто повільною. В таких випадках доцільно використовувати коди виправлення помилок. Серед них доцільно виділити коди Хеммінга та коди Ріда-Соломона. Код Хемінга використовується для виправлення одиночної помлки. Код Ріда-Соломона дозволяє коректувати декілька помилок без повторної пересилки даних, та застосовується, наприклад, при збереженні даних на CD накопичувачі.

Ці коди є добре задокументованими та широко використовуються в електроніці, як на рівні ПЗ так і на рівні інтегральних схем. Але вони не підходять для виправлення великої кількості помилок. Наразі код Ріда-Соломона 15-11 дозволяє виправити 40 похибок на 10 блоків коду (600 біт), але, якщо вони локалізовані у різних блоках. Але вони перестають працювати при концентрації помилок в одному місці. Електричні реалізації цих кодів пришвидшують процес кодування/декодування, але їх не можно змінити, якщо параметри канала даних змінились.

Метою дипломної роботи є реалізація поміхо стійкого кодеку для збереження даних на фізичних носіях та в системах передачі даних де зустрічаються поміхи.

Для реалізації цієї задачі доцільно використовувати циклічні коди. Коди Хеммінга та коди Ріда-Соломона мають під собою однаковий математичний апарат – коди БЧХ, та обидва види кодів є частною реалізацією кодів БЧХ. Саме на кодах БЧХ доцільно реалізувати такий кодек. Це дозволить гнучко підстроювати коди під конкретну задачу.

Такий кодек біло би зручно використати для збереження інформації на фізичних носіях або для передачі інформації в космічний та військових галузях.

# АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

## 1.1 Введення в алгебру полів Галуа

На початку історії цифрової електроніки машини оперували машинними словами, котрі потім почали називати байтами. Байти в свою чергу реалізовувались як група бітів. Біт – мінімальна одиниця інформації, що може приймати два значення: 1 або 0. Були окремі спроби реалізувати трити – одинця інформації що може мати 3 значення: 0, 1 або2. Але таки системи не знайшли популярності, через свою повільність та складність в реалізації. З часом інженери виявили, що схеми котрі працюють в байтами, котрі мають кількість рівну цілим ступеням 2-ки працюють набагато швидше, ніж системи, з іншою кількістю бітів. Після років експериментів майже всі обчислювальні машини використовують байт довжиною 8 бітів.

Математичні операції над бітами є реалізацією булевої алгебри. Розділ математики що розглядає властивості таких типів даних називається – дискретна математика. Якщо роздивлятись машинне слово довжиною 8 бітів то ми побачимо, що їх варіативність кінцева, а саме є лише 28=256 варіантів байту. Що означає що люба інформація котрою оперують обчислювальні машини є комбінаціями цих байтів. Пошкодження цієї інформації означає – зміну бітів в байті на протилежний.

Циклічні коди також працюють на кінцевих полях. Хоча в загальному виді кінцеві поля реалізовують математику для різних основ та ступенів, для реалізації кодеку доцільно використати лише поля з основою – 2 та цілими ступенями, починаючи з 2-х

Поля Галуа є різновидами кінцевих полів, на основі котрих реалізовані циклічні коди. З визначення терміну «поле» - ми знаємо, що для поля існують детерміновані операції додавання, віднімання, множення і ділення. Також є операції відведення в ступінь, котру можна розглядати як окремий випадок множення.

Операції віднімання та додавання в полях Галуа є еквівалентними та представляють з себе додавання за модулем 2: (a+b)mod(2). Машинний еквівалент цієї операції є оператор XOR. Для цієї операції характерні наступні властивості: рефлективність, симетричність та транзитивність.

Операції множення має наступну реалізацію: знайти ступеневу форму чисел Галуа для множників; скласти ступені; знайти число відповідно до цього ступеня. Якщо ступень виходить за межі простору поля Галуа, то його слід поділити за модулем на максимальну ступінь цього поля.

Операція ділення робіться так аналогічно, але перед тим, ступінь дільника слід змінити на протилежну.

Маючи реалізацію цих 4-х операцій мі можемо кодувати бінарну інформацію в коди БЧХ.

Коди БЧХ (коди Боуза — Чоудхурі — Хоквінгема, Bose–Chaudhuri–Hocquenghem codes, BCH codes) – це клас циклічних кодів котрі використовуються для поміхо стійкого кодування інформації. Ключовим для цих кодів є те, що в них є інформація для виправлення заданої кількості помилок.

На відміну від коду Хеммінга або кодів Ріда-Соломона, БЧХ код можна налаштовувати на завдану довжину та кількість помилок(але с завданими обмеженнями).

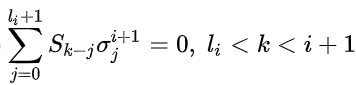
Для декодування кодів БЧХ є декілька алгоритмів, а саме:

1. Алгоритм Берлекемпа - Месі
2. Євклідів алгоритм
3. Пряме рішення (алгоритм Пітерсона - Горенстейна - Цирлера, ПГЦ)
4. Пошук Ченя
5. Алгоритм Форні

Роздивмося декілька алгоритмів.

Алгоритм ПГЦ: історично є першим алгоритмом декодування БЧХ коду. Цей алгоритм заснован на прямому рішенні системи поліноміальних рівнянь, де ведеться пошук коефіцієнтів локаторів помилок. Реалізація цього алгоритму підходить тільки для систем з маленькими полями Галуа.

Євклідів алгоритм. Через високу регулярніть популярний для рішення апаратного декодування кодів Ріда-Соломона.

Алгоритм Берклемпа-Месі. Є високопотожним алгоритмом. Його слід роздивлятись як ітеративний процес генерації реєстру ссуву, для генерації заданої послідовності синдромів. Його мета знайти найменшу ступінь поліному  котра буде задовільнять рівнянню . Його доцільніше використати для реалізації програмного декодеру.

## 1.2 Алгоритм створення БЧХ коду

* Визначити ступінь до поля Галуа, котре буде використовуватись для кодування.
* Построїти таблицю чисел Галуа, для завданої форми (десятина форма, двійкова форма, мультиплікативна форма.
* До цих чисел знайти ступеневу форму та мінімальний зворотній поліном.
* На основі попередніх обчислень, обчислити мінімальний зворотній поліном для коду БЧХ с завданою кількістю помилок. Знайти ступінь мінімального зворотного поліному.
* Знайти довжину інформаційного повідомлення, та поділити вхідну інформацію на блоки з цією довжиною.
* Закодувати інформацію блок за блоком, та зберегти її.

## 1.3 Алгоритм декодування БЧХ коду

* Визначити ступінь до поля Галуа, котре буде використовуватись для кодування.
* Построїти таблицю чисел Галуа, для завданої форми (десятина форма, двійкова форма, мультиплікативна форма.
* До цих чисел знайти ступеневу форму та мінімальний зворотній поліном.
* Взяти блок закодованої інформації.
* Обчислити синдроми. Якщо вони нульові – або інформація ціла, або помилки компенсували друг друга.
* Якщо синдроми не нульові – обчислити мінімальний полином локаторів помилок.
* Визначити ступінь цього пліному. Якщо вона перевищує кількість помилок, то декодування неможливе, якщо співпадає, або менша – продовжити декодування.
* Знайти корні поліному локаторів (методом повного перебору)
* Виправити вхідний поліном згідно до коренів. Перевірити новий поліном. Якщо синдроми дорівнюють нулю, то декодування пройшло вдало.

## Визначення вимог для кодеку

Так як кінцевим продуктом дипломної роботи має бути кодек для кодування/декодування кодів БЧХ, треба визначитись з тим, в якому виді цей кодек буде постачатись.

В класичному виді кодек має бути динамічною бібліотекою, котру можна підключити до програмного продукту, та використовувати з клієнтського додатку.

Також однією з вимог до кодеку є те, що він має бути легким для обчислювання на процесорі та не використовувати багато оперативної пам’яті.

1. **ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ ТА ТЕХНОЛОГІЙ**

## Порівняльні характеристики мов програмування

Насамперед треба означитись з мовою програмування. Згідно з вимог для кодеку та особливостей реалізації алегбри для кінцевих полів мова програмування має відповідати наступним характеристикам:

* Має бути достатньо швидкою.
* Має доволяти робити низькорівневу математику.
* Має мати весь перелік булєвих операцій.
* Використовувати мало оперативної пам’яті.

Згідно с характеристик можемо зробити висновок що мова програмування має бути достатньо низького рівня для ефективного використовування оперативної пам’яті та швидкого обчислення поліномів. З одного боку було б зручніше зробити програмну реалізацію на мові Python, так як математика там реалізована на векторах, що дозволяє нам працювати з дуже великими цифрами без переповнення, що характерні для низькорівневих статично типізованих мов програмування, але через векторну реалізацію математики, обчислення будуть проводитися довго. Також мова Python через особливості своєї реалізації є повільною, використовує багато оперативної пам’яті та працює лише в одному потоці, що наскладує обмеження на швидкість реалізації алгоритмів.

З статично типізованих мов програмування доцільно розглядати кандидатів серед C++, C# та Java, через те що вони є найпоширеніші серед мов для побудови клієнтських та серверних додатків.

Серед переваг Java та C# можна виділити наступні:

* Не має ручного менеджменту пам’яті.
* Багато бібліотек, котрі йдуть з SDK.
* Відносно легке створення коду, через попередні пункти.

Серед недоліків:

* Потрібних для роботи бібліотек не має, що нівелює користність існуючих бібліотек.
* Автоматичний менеджмент пам’яті приводить до великої кількості оперативної пам’яті в додатках.
* Є повільними відносно C++

Якщо розглядати мову C++, то можна побачити наступні переваги:

* Швидке виконання
* Доступ до низькорівневих операцій
* Можливість керувати користуванням оперативної пам’яті.
* Потужна реалізація стандартних колекцій/

Серед недоліків:

* Складна реалізація через специфічний набор стандартних бібліотек (але в нашому випадку, їх досить)
* Великий поріг входу для програмістів.

Зваживши всі плюси і мінусі мов програмування було вирішено використати мову C++.

## 2.2 Огляд існуючих програмних рішень

Через складну реалізацію кодів БЧХ та літературу до неї, котра орієнтована більше на науковців теоретиків гарних універсальних імплементацій БЧХ алгоритму для двійкового коду в відкритому доступі не має.

В реалізація полів Галуа в відкритому доступі більшість реалізована для GF(23) або GF(24), що не є гнучким. Також в цих реалізаціях не має мінімальної зворотної функції для GF.

## 2.3 Порівняльні характеристики способів доставки коду

Кодек для кодування декодування треба використовувати в клієнтських додатках як сторонню бібліотеку. Згідно з принципами побудови бібліотек в мові С++ є 3 способи це зробити:

* Створити дінамічну бібліотеку.
* Створити статичну бібліотеку.
* Віддати код всього пакету для компілювання код як частину додатку.

Розглянем переваги та недоліки кожного варіанту.

**Динамічна бібліотека.**

Переваги:

* Інкапсуляція коду бібліотеки, та захист її від модифікацій.

Недоліки:

* Клієнтська програма має бути скомпільована с тими же ключами що і динамічна бібліотека. Через це треба мати інфраструктуру для побудови бібліотеки під всі бажані платформи, та версії операційних систем.
* Складність в підключені динамічних бібліотек для новачків.

**Статична бібліотека.**

Переваги:

* Інкапсуляція коду бібліотека, та захист її від модифікацій.

Недоліки:

* Трохи легша в підключенні ніж динамічна бібліотека.

**Доставка коду пакетом.**

Переваги:

* Легкий спосіб доставки коду.
* Легкий спосіб підключення коду в проект.
* Легка реалізація кросплатформеності коду.
* Компіляція коду з бажаними ключами.
* Клієнт може подивитися в код та модифікувати його в разі потреби на це, що дозволяє розробляти менший об’єм документації

Недоліки:

* Клієнт може модифікувати та передивлятись код, що може бути критичним для бізнесу.

Дивлячись на ті обмеження котрі стоять перед нами в рамках дипломної роботи, біло вирішено доставляти код, як набір файлів, котрі треба самостійно включати до проекту.

2.4 Огляд існуючих алгоритмів декодування

Як вказувалось раніше є декілька методів декодування БЧХ коду. Розглянемо ближче кожен з них.

**Евклідів алгоритм.**

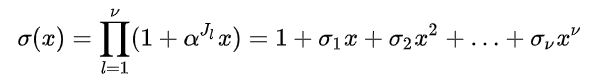
Евклідів алгоритм – в нашому випадку це класичний евклідів алгоритм, проте замісць знаходження найбільшого спільного дільника (НСД), відбувається пошук НСД для двох поліномів.

Прийняте кодове слово *r(x)=c(x)+e(x)*, де *e(x)* – поліном помилок, а *c(x)* кодове слово + породжуючий поліном для перевірки. Згідно з цим в нас може бути *u≤t=(d-1)/2* помилок на позіціях *i1,i2,…, iu*, де *t* максимальна кількість помилок, котрі можна виправити. З цього слідує *e(x)=ei1xi1+ ei2xi2 +ei3xi3+…+ eiuxiu,* а *ei1, ei2, …, eiu –* це помилки передачі.

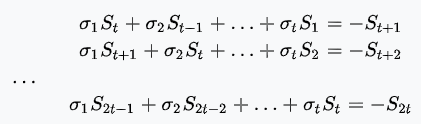
Визначимо поліном значення помилок *Λ=σ(x)S(x).* Де синдромний поліном дорівнює:



Згідно з визначенням локаторів помилок, він дорівнює:



де корені рівняння є зворотними величинами локаторів помилок. Тоді буд вірне наступні відносини між коефіцінтами полінома локаторів помилок і синдромами:



Це рівняння є основним та лежить в основі декодування БЧХ коду. Згідно з основної системи рівняннь:



Задача зводиться к тому щоби знайти всі *Λ(x)* так, що би корені не були вищи за td. По суті це є розширений алгоритм Евкліда. Має комплексну складність O(n3).

**Алгоритм ПГЦ**.

Алгоритм ПГЦ - алгоритм заснован на прямому рішенні системи поліноміальних рівнянь, де ведеться пошук коефіцієнтів локаторів помилок. Це алгоритм прямого рішення системи рівнянь, де ми з одного боку маємо обчислені синдроми, а з іншого боку нам відома максимальна кількість помилок котрі мі можемо виправити.

Нехай БЧХ код на полем GF(q) довжини n і конструктиною відстаннь d задається породжуючим поліномом g(x), який серед своїх коренів має елементи:



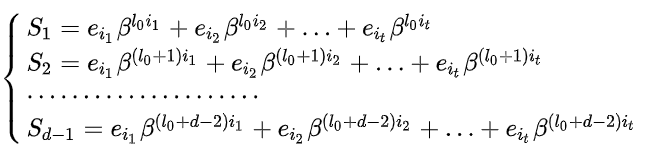
де l0 ціле число (наприклад 0 або 1).

Щыгдно з основним рывнянням можна виділити *j-*й синдром Sj прийнятого повідомлення *r(x):*

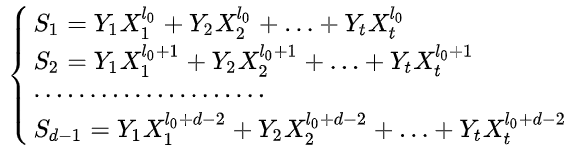
**

Задача полягає в тому, що треба знайти всі комбінації кількості помилок, та їх позіцій (через те що, в контексті дипломної роботи мі розглядаємо лише бінарні коди, то значення помилки завжди 1).

Припустимо, для початку, що *u* точно дорівнює *t*. Запишемо систему лінійних рівнянь:



Визначим *Xk=bik* локатор *k-ї* помилки, а через *Yk=eik* зачення помилки, *k=1,2, ..., t.*

**

Зробимо поліном локаторів помилок:



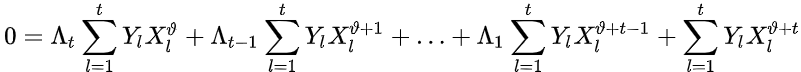
Коренями цього поліному елементи, зворотні локаторам помилок. Помножимо обидві сторони на . Отримане рівняння буде справедливо для *ν=l0, l0+1, l0+2, l0+2, …, l0+d-1, l=1, …,t:*

**

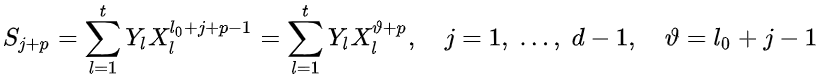
Помножимо , таким чином отримаємо:



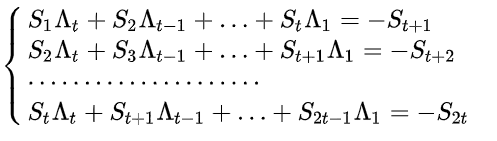
Приведемо рівняння до виду:



Враховуючи що:



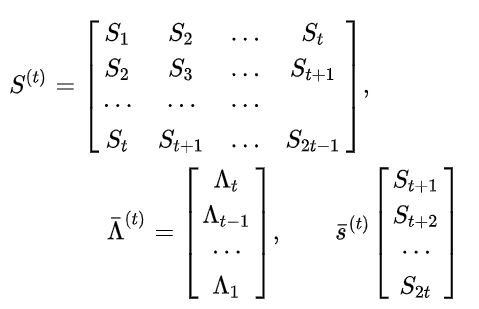
Таким чином, можемо отмиати систему лінійних рівнянь:



Або в матричній формі:



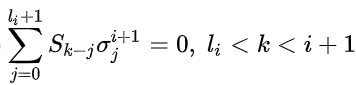
Де:



Якщо кількість помилок дорівнює *t,* то система вирішається для коєфіцієнтів *Λ1, Λ2, Λ3, …, Λt*. Якщо ні визначник матриці *S(t)* системи буде дорівнювати 0. Що значить, що кількість помилок меньша. Тому слід повторити процес з кількістю помилок *t-1,* і так далі до 1.

Як ми бачимо такий алгоритм є складним для побудови в програмному виді, та дуже затратним за ресурсами.

Євклідів алгоритм. Через високу регулярніть популярний для рішення апаратного декодування кодів Ріда-Соломона.

Алгоритм Берклемпа-Месі. Є високопотожним алгоритмом. Його слід роздивлятись як ітеративний процес генерації реєстру ссуву, для генерації заданої послідовності синдромів. Його мета знайти найменшу ступінь поліному  котра буде задовільнять рівнянню .

**Алгоритм Берлекемпа – Мессі.**

Алгоритм Берлекемпа – Мессі – його ми будемо реалізовувати, тому розглянемо його далі. Слід зазначити, що цей алгоритм найшвидший серед інших алгоритмів декодування, тому саме його доцільно вживати для декодування.

# ПРОЕКТУВАННЯ ДОДАТКУ

Додаток доцільно поділити на 3 окреми частини: модуль арифметики полів Галуа, кодер БЧХ, та декодер БЧХ.

Перша частина буде реалізовувати поліноміальну алгебру для заданих кодових слів розміром q ( GF(q)). В цій частині треба реалізувати таблицю алфавіту векторну форму, алгоритмічно адаптивну форму та мультиплікативну форму GF(q), та знайти зворотній поліном.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **N** | **Векторне подання** | **Адитивна група** | **Ступенева форма** | **Зворотній поліном** |
| 0 | 0 | 0 | - |  |
| 1 | 1 | 1 | x0 | x+1 |
| 2 | 10 | x | x1 | x4+x+1 |
| 3 | 11 | x+1 | x4 | x4+x+1 |
| 4 | 100 | x2 | x2 | x4+x+1 |
| 5 | 101 | x2+1 | x8 | x4+x+1 |
| 6 | 110 | x2+x | x5 | x3+x+1 |
| 7 | 111 | x2+x+1 | x10 | x3+x+1 |
| 8 | 1000 | x3 | x3 | x4+x3+x2+x+1 |
| 9 | 1001 | x3+1 | x14 | x4+x3+1 |
| 10 | 1010 | x3+x | x9 | x4+x3+x2+x+1 |
| 11 | 1011 | x3+x+1 | x7 | x4+x3+1 |
| 12 | 1100 | x3+x2 | x6 | x4+x3+x2+x+1 |
| 13 | 1101 | x3+x2+1 | x13 | x4+x3+1 |
| 14 | 1110 | x3+x2+x | x11 | x4+x3+1 |
| 15 | 1111 | x3+x2+x+1 | x12 | x4+x3+x2+x+1 |
| 1 | 1 | 1 | x15 |  |

Для побудви БЧХ коду треба реалізувати повну арифметику для чисел полів Галуа, а саме 4 арифметичні дії: додавання, віднімання, множення та ділення, так як ці арифметичні дії будуть використовуватись, при кодуванні та декодуванні БЧХ. Також для реалізації кодування та декодування, треба зробити арифметику множення чисел Галуа на невизначені коефіцієнти.

Наступний модуль програми, кодер БЧХ: користувач має зазначити розмірність кодового повідомлення, та кількість помилок, що слід виправити в кодері. Також на вході в цей модуль має буде повідомлення що слід закодувати, або декодувати. Декодер має бути реалізацією алгоритму Берклемпа-Месі.

Для тестової програми, достатньо вносити помилки власноруч, щоби перевірити, здатність декодеру локалізувати ці помилки, та правильно їх обробляти.

## 3.1 Реалізація алгебри для чисел полів Галуа

## 3.2 Реалізація алгоритму знаходження мінімальної зворотної функції

## 3.3 Реалізація кодування БЧХ

3.4 Реалізація декодування БЧХ

Наприклад: python b4x.py --encode 00010000 00100000 00001100 01010110 01100001 10000000 11101100 00010001 11101100 00010001 11101100 00010001 11101100 00010001 11101100 00010001 –lenght 8

Результат: Encoded result for give input is: 00010000 00100000 00001100 01010110 01100001 10000000 11101100 00010001 11101100 00010001 11101100 00010001 11101100 00010001 11101100 00010001 ::: 10100101 00100100 11010100 11000001 11101101 00110110 11000111 10000111 00101100 01010101. Де 10100101 00100100 11010100 11000001 11101101 00110110 11000111 10000111 00101100 01010101 це результат обчислення додаткового коду для нашого повідомлення.

Після перейти до тестування та знаходження помилок в реалізації програми.

# ТЕСТУВАННЯ

Для тестування можна взяти приклад з попереднім результатом, та подати його на вход до самої програми, використовуючи ключ “--decode": python b4x.py --decode 00010000 00100000 00001100 01010110 01100001 10000000 11101100 00010001 11101100 00010001 11101100 00010001 11101100 00010001 11101100 00010001 10100101 00100100 11010100 11000001 11101101 00110110 11000111 10000111 00101100 01010101 --length 8. На виході отримаємо результат: No errors found. Your input code is: 00010000 00100000 00001100 01010110 01100001 10000000 11101100 00010001 11101100 00010001 11101100 00010001 11101100 00010001 11101100 00010001.

Перевіряємо виданий результат з повідомленням, котре ми вводили з самого початку: результати співпадають. Робимо теж саме, декілька разів, бачимо, що програма працює з кодами без помилок успішно.

Після цього міняємо перший біт в 4-ому байті з нуля но одиницю 01010110 -> 11010110. Вводимо змінений код до програми. Бачимо в результаті дії програми: 1 error found: bit #25. Your input code is: 00010000 00100000 00001100 01010110 01100001 10000000 11101100 00010001 11101100 00010001 11101100 00010001 11101100 00010001 11101100 00010001. Перевіряємо виданий результат. Програма успішно зробила декодування 16-ті байтів, з одиничною помилкою.

Робим ще одну помилку, збільшив кількість помилок до 2-х. Перевіряємо результат. Додаємо ще одну, знову перевіряємо результат. Робимо це, поки програма не видасть: Cannot decode: to many errors**.** Кількість помилок для цього повідомлення зазначила 31 помилок, але в залежності де саме та як близько будуть знаходитися помилки кількість помилок може відрізнятись: невелика кількість помилок поспіль, може пошкодити код більше, ніж велика рівномірно по всьому коду.

# 

# ВИСНОВКИ

В результаті практики було отримано програму, яка реалізовує БЧХ кодування та декодування повідомлень, як з помилками в повідомленні для декодування, так і з помилками. Програма коректо знаходить локатори синдромів (помилок) та успішно виправляє похибки в достатньої кількості. Але так як програма реалізована на мові Python, та є прототипом, то в неї є декілька недоліків, а саме: невелика швидкість роботи, порівняно з низько- рівневими мовами програмування; не реалізована передача даних – повідомлення для декодування потрібно вводити користувачеві через термінал; також не реалізовано графічний інтерфейс користувача, що ускладнює роботу з програмою.