Міністерство освіти і науки України

Національний технічний університет Україні

«Київський політехнічний інститут імені Ігоря Сікорського»

НМК ІПО

Кафедра автоматики та управління в технічних системах

Технічна кібернетика

ЗВІТ

про преддипломну практику

(назва практики)

на ФОП Миколаєнко С. А.

(назва підприємства, місто)

з « 15 » 04 2019 р. по « 19 » 05 2019 р.

Керівник практики: Слухач 4 курсу, групи ЗПІ-ЗП61

від підприємства

Миколаєнко С. А. Макіян С. А.

(прізвище, ініціали) (підпис) (прізвище, ініціали) (підпис)

(дата) (дата)

від інституту

Хмелюк М. С.

(прізвище, ініціали) (підпис)

(дата)

Захищено

(дата)

з оцінкою

Київ 2019 р.

**ЗМІСТ**

[РОЗРОБКА ЗАВДАННЯ ПРОЕКТУ 3](#_Toc9430196)

[ОГЛЯД ІСНУЮЧИХ РІШЕНЬ 4](#_Toc9430197)

[ПРОЕКТУВАННЯ ДОДАТКУ 6](#_Toc9430198)

[ТЕСТУВАННЯ 9](#_Toc9430199)

[ВИСНОВКИ 11](#_Toc9430200)

# РОЗРОБКА ЗАВДАННЯ ПРОЕКТУ

Метою преддипломної практики є створення кодеку для виправлення помилок, в системах передачі даних.

Перед розробкою вимог до системи необхідно проаналізувати кому буде корисна дана система, та в яких середовищах її доцільно використовувати.

На даний момент є багато методів передачі інформації, які використовуються для перевірки та виправки помилок. Серед них можна виділити: CRC match, котрий використовуються в фреймах TCP/IP; контроль та виправлення одиничної похибко кодами Хеммінга; коди Ріда-Соломона. Всі ці алгоритми широко використовуються в сучасній електроніці та програмному забезпеченні. Вони маленькі, швидкі та добре продокументовані. Кожен з них має свої вади. Наприклад, код Ріда-Соломона 15-11 дозволяє виправити 40 похибок на 10 блоків коду (600 біт), але, якщо вони локалізовані у різних блоках. CRC-match повторної пересилки даних, та не підходить для використовуванні на фізичних носіях (наприклад CD-Rom) або в космичній галузі де пересилка пакету може бути критично довгою. Код Хеммінга, також не дозволяє виправити велику кількість помилок, і є більше перевірочним кодом.

Якщо взяти код БЧХ, то він дозволяє реалізовувати виправлення великої (або малої кількості помилок, залежить від бажаного результату), є гнучким, та є породжуючим для коду Ріда-Соломона. Тобто його можна настроїти на будь яку бажану ситуацію: збереження на фізичних носіях, передачі даних в космічній та воєнній галузях. Саме його доцільніше всього реалізовувати як базовий алгоритм для дипломного проєкту.

# ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

В загальному виді БЧХ код не використовується, про те використовуються коди Ріда-Соломона заданої довжини. Також для реалізації блочних кодів, треба мати імплементацію поліноміальної математики полів Галуа, котрі також реалізовані або псевдокодом, або є закритими.

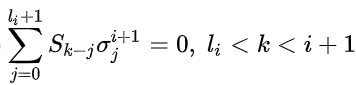
Серед алгоритмів реалізації кодування БЧХ коду використовують циклічні коди. Але серед алгорітмів декодування є декілька реалізацій, а саме:

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 використувати для реалізації програмного декодеру.

# 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 мультипликативну форму GF(q).



На таблиці віще ступеневе представлення – мультиплікативна форма, а поліноміальне представлення – адаптивна алгоритмічна. Мультиплікативного представлення треба зробити 4 арифметичні дії: додавання, віднімання, множення та ділення, так як ці арифметичні дії будуть використовуватись, при розкладенні на ступені поліноми з коефіцієнтами для кодеру БЧХ.

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, і котрий буде поділено на блоки, та передано до адресата.

Третій та останій модуль, це декодер, котрий має отримати кодове повідомлення, та виправити помилки, якщо вони є. Декодер є реалізацією алгоритму Берклемпа-Месі.

Для тестової програми, достатньо вносити помилки власноруч, але для подальшої дипломної роботи, слід зробити випадкову генерацію помилок.

Мова для тестової програми Python, але для дипломної роботи доцільно написати код на мові C++, котрий буде швидший, там має більш гнучкі можливості по оптимізації використовування процесорного часу та використаної пам’яті, так як метою є не працююча програма, а швидкий кодек для передачі даних.

Після реалізації всіх трьох модулів, треба зробити інтерфейс командного рядка для програми: ключ “--encode” для вводу повідомлення; ключ “--decode” для його декодування; та ключ “--lenght” для покажчика ступіню поля Галуа.

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.