МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний економічний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп'ютерної інженерії

КОЧЕНКО Аліна Олегівна

Алгоритми стиснення даних в безпровідних сенсорних мережах / Data Compression Algorithms in Wireless Sensor Networks

спеціальність: 123 - Комп'ютерна інженерія
магістерська програма - Комп'ютерна інженерія

Магістерська робота

Виконала студентка групи КІм-21
А. О. Коченко _____

Науковий керівник:
к.т.н., доцент, В. В. Яцків _____

Магістерську роботу допущено
до захисту:

"30" 01 2018 р.

Завідувач кафедри
_____ **О. М. Березький**

ТЕРНОПІЛЬ - 2018

Тернопільський національний економічний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп'ютерної інженерії
Освітній ступінь «магістр»
спеціальність: 123 - Комп'ютерна інженерія
магістерська програма - Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.М. Березький
"17" 10 2016 р.

ЗАВДАННЯ
НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ
Коченко Аліни Олегівни
(прізвище, ім'я, по батькові)

1. Тема магістерської роботи «Алгоритми стиснення даних у безпровідних сенсорних мережах/Data Compression Algorithms in Wireless Sensor Networks» керівник роботи доцент, к.т.н., В.В. Яцків
затверджені наказом по університету від 11 жовтня 2016 р. № 669.

2. Строк подання студентом роботи «15» січня 2018 року

3. Вихідні дані до магістерської роботи

Об'єкт дослідження – процеси обробки даних в безпровідних сенсорних мережах.

Предмет дослідження – алгоритми стиснення даних в безпровідних сенсорних мережах.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

– дослідити безпровідні сенсорні мережі їх характеристики та особливості;
– здійснити класифікацію основних методів та алгоритмів стиснення даних;
– дослідити програмні засоби для стиснення даних;
– розробити асиметричний за складністю алгоритм стиснення даних;
– здійснити програмну реалізацію алгоритму стиснення даних;
– провести тестування розроблених алгоритмів.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень):

– тема, мета, завдання, наукова новизна, практичне значення, актуальність;
– аналіз методів стиснення даних в безпровідних сенсорних мережах;
– об'єкт дослідження;

– вимоги до алгоритмів стиснення даних в безпровідних сенсорн мережах;
– приклад моделювання алгоритму стиснення даних;
– програмна реалізація алгоритму стиснення даних;
– експериментальне дослідження розроблених алгоритмів.

6. Консультанти розділів магістерської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
Антиплагіат	Мельник Г.М., доцент		
Нормо-контроль	Гураль І. В., викладач		

7. Дата видачі завдання « 13 » жовтня 20 16 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів магістерської роботи	Прим
1	Аналіз методів та алгоритмів стиснення даних в безпровідних сенсорних мережах	3.11.2016 – 1.01.2017	
2	Алгоритми кодування декодування в безпровідних сенсорних мережах	2.01.2017 – 31.05.2017	
3	Програмна реалізація та дослідження алгоритмів стиснення даних	1.06.2017 – 25.01.2018	
4	Нормоконтроль, попередній захист	16.01.2018 – 2.02.2018	
5	Захист	4.02.2018	

Студент _____ Коченко А. О.
(підпис)

Керівник магістерської роботи _____ д.т.н., доцент В. В. Яцків
(підпис)

ВСТУП

Темою даної дипломної роботи є дослідження алгоритмів стиснення у безпровідних сенсорних мережах.

Безпровідні сенсорні мережі на сьогоднішній день одна з швидко розвинених областей радіозв'язку. Не дивлячись на те,що використання безпровідних сенсорних мереж в різних сферах діяльності розпочалось не так давно, їх популярність дуже швидко розвивається.

Ідея використання безпровідної технології для передачі даних в різних системах для збору інформації та управління є дуже популярною за рахунок своєї низької ціни та високої надійності.

Стиснення даних – це алгоритмічне перетворення даних, яке виготовлене з ціллю зменшення їх об'єму. Стиснення даних використовується для більше раціонального використання пристроїв збереження та передачі даних.

Згідно відомих даних про алгоритми стиснення можна зрозуміти, що не існує методів чи алгоритмів, які здатні ефективно стискати будь-які файли, алгоритм стиснення має спочатку дослідити його, знайти в ньому надмірність та помилки, а потім постаратись видалити їх. Оскільки, надмірність залежить від типу даних (текст, графічний об'єкт, звук і т.д.), методи компресії мають розроблятися з урахуванням цього типу. Алгоритм буде краще всього працювати зі своїми даними.

Метою роботи є дослідження алгоритмів та методів стиснення даних та здійснення програмної реалізації алгоритму стиснення даних.

Для досягнення поставленої мети необхідно розв'язати такі завдання:

– дослідити безпровідні сенсорні мережі їх характеристики та особливості;

– здійснити класифікацію основних методів та алгоритмів стиснення даних;

– дослідити програмні засоби для стиснення даних;

– розробити асиметричний за складністю алгоритм стиснення даних;

– здійснити програмну реалізацію алгоритму стиснення даних;

– дослідити обчислювану складність алгоритму стиснення даних;

– дослідити функціональні параметри безпровідної сенсорної мережі з використанням алгоритму стиснення.

Об’єктом дослідження в даній роботі виступають безпровідні сенсорні мережі, а предметом дослідження – алгоритми стиснення даних.

Методи дослідження базуються на теорії алгоритмів для стиснення даних та функціональних параметрах безпровідної сенсорної мережі з використанням алгоритму стиснення.

Наукова новизна полягає у розробці асиметричного за складністю алгоритму стиснення даних на основі вже існуючих алгоритмів.

Практичне значення. Розроблено асиметричний за складністю алгоритм стиснення даних та здійснено програмну реалізацію даного алгоритму.

1 АНАЛІЗ МЕТОДІВ ТА АЛГОРИТМІВ СТИСНЕННЯ ДАНИХ У БЕЗПРОВІДНИХ СЕНСОРНИХ МЕРЕЖАХ

1.1 Безпровідні сенсорні мережі їх особливості та класифікація

Технологія безпровідних сенсорних мереж отримала широке застосування в різних сферах діяльності. Безпровідна сенсорна мережа – це розподілена самоорганізаційна мережа, яка стійка до відмови окремих елементів, об’єднана між собою за допомогою радіоканалу. Кожний елемент мережі має автономний ресурс живлення, мікрокомп’ютер, приймач/передавач. В залежності від типу модуля, антени та можливості ретрансляції повідомлення від одного елементу до іншого, область покриття такої мережі може складати від декілька метрів до декількох кілометрів. У випадку якщо відстань між двома кінцевими пристроями велика і не надає можливості для взаємної передачі даних, то можна використовувати ретранслятор. В зв’язку з цим, пристрої з малим радіусом дії за допомогою ретранслятора можуть спілкуватися один з одним [2].

До основних стандартів, які використовуються в малопотужних безпровідних сенсорних мережах можна віднести:

- IEEE 802.15.4;
- ZigBee;
- Wibree;
- Bluetooth.

Безпровідні сенсорні мережі спочатку почали використовувати воєнні для висліджування цілей в бойових умовах. Як приклад, велика кількість скинутих з повітря на місце бойових дій сенсорних вузлів, можуть прозондувати місцевість та пересилати дані, які стосуються бойової техніки та живих сил противника з місця проведення бою. З кожним роком БСМ все більше і більше використовуються в різних сферах діяльності. Основною задачею сенсорних мереж є моніторинг, збір даних за певний проміжок часу. БСМ проводять

промисловий моніторинг, моніторинг навколишнього середовища, транспорту, контроль руху об’єктів. Великим плюсом сенсорних мереж є: їх малий розмір, не потрібні провідні комунікації, можливість розгортати мережі в важких умовах. Все це робить технологію безпровідної сенсорної мережі досить таки гнучкою та затребуваною. Ці фактори вважаються вирішальними при побудові мережі, яка містить велику кількість інтелектуальних сенсорних вузлів та інтерфейсів передачі даних. Стійке зниження вартості рішень для безпровідного зв’язку та збільшення їх експлуатаційних параметрів дозволяють позиціонувати БСМ в якості ефективних та перспективних рішень для систем концентрації телеметричних даних, дистанційної діагностики віддаленого обміну інформацією [3].

В США нараховується декілька сотень комерційних фірм, які спеціалізуються в області створення та сервісу безпровідних сенсорних мереж. В 2010 році в світі було поставлено користувачам більше 500 мільйонів сенсорних датчиків на суму близьку до 7 млр. доларів.

Поява потужних вбудованих мікрокомп’ютерних систем для БСМ забезпечує хороший повід для створення нових розумних сенсорних систем, які можуть бути корисні для подальшого розвитку нових наукових досліджень та покращення життя. БСМ використовуються в дуже різних сферах діяльності. До найбільш популярних завдань і додатків, які реалізовуються на основі технології безпровідних сенсорних мереж належать:

- детектори знаходження та моніторингу пожеж лісу та торфу;
- сейсмічний моніторинг;
- моніторинг навколишнього середовища (виявлення та прогнозування стихійного лиха);
- автоматичний дистанційний контроль параметрів радіаційних об’єктів, газу та інших потенційно небезпечних промислових об’єктів;
- контроль дорожнього руху та транспортної інфраструктури;
- моніторинг несучих конструкцій будівель та споруд;

– місцезнаходження управління, сповіщення та організації надійного зв’язку для рятувальних операцій;

– моніторинг місцевого теплопостачання.

Ефективність БСМ – це не просто їх моніторинг, обчислювані та комунікаційні можливості з додатковою обробкою мікрокомп’ютерів, потужності, аналогових та цифрових портів, трансиверів та доступної пам’яті, вони мають можливість організовувати самостійний організм та спілкуватись в розверненій площині. Їх обчислювана потужність обмежена, проте БСМ зазвичай розгортаються у великій кількості та їх загальне навантаження відповідно.

1.1.1 Складові елементи безпровідного сенсорного вузла

Основними компонентами вузла безпровідного давача, як зображено на рисунку 1.1 є: давач/привід, вбудований контролер (мікрокомп’ютер), блок живлення, пам’ять та комунікаційний пристрій.

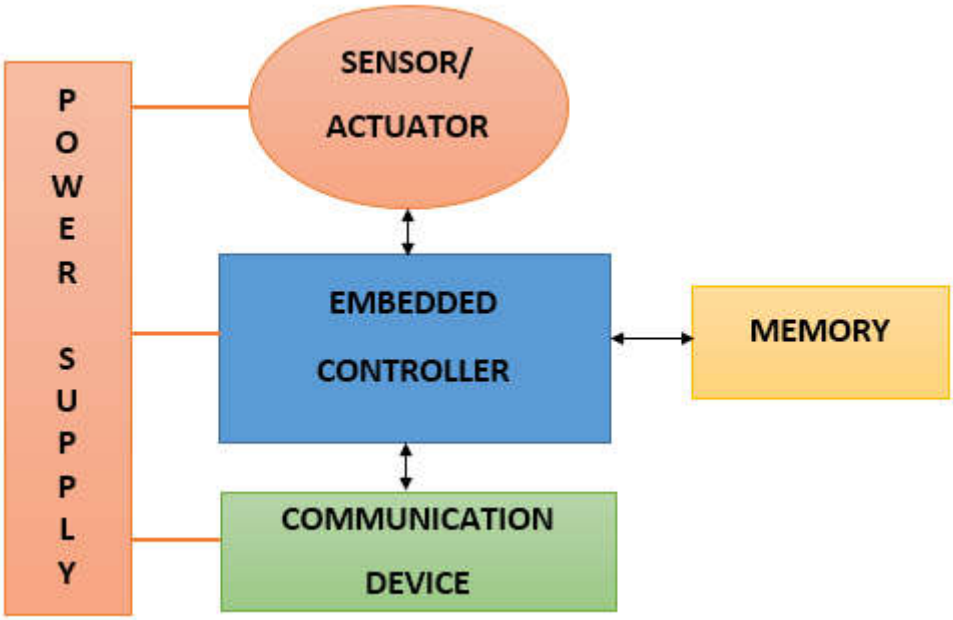

Рисунок 1.1 – Елементи безпровідного сенсорного вузла

Сенсорні вузли мають широке застосування в БСМ із-за їх низької вартості, малих габаритів, простоти розгортання та енерго-ефективності. Вибір архітектури сенсорних вузлів, у тому числі їх апаратно-програмної платформи, залежить від додатків та вимог. Через те, що давачі вузлів мають обмежену пропускну здатність, високу обчислювану потужність та обмежені енергетичні ресурси, одним важливим фактором в БСМ – є енергоефективність давача вузлів. Основними ресурсами енегровикористання давача вузлів являється мікроконтролер і прийомо-передавач. Звідти слідує, що використання енергії вузлів давача є важливим фактором при розгляді безпровідних давачів для конкретного додатку. Для того, щоб знизити використання енергії, давачі використовують роботу в різних режимах, а саме: активний, режим сну та режим очікування. В БСМ контроль топології є одним із підходів, котрі використовуються для вирішення проблеми енергетичної ефективності сенсорних мереж [4].

Частіше всього основною частиною блоку сенсорного вузла є мікроконтролери, через свою низьку вартість, з вбудованою пам’яттю, і різноманітним набором переферійних пристроїв та гнучкістю для підключення до інших пристроїв, мала кількість затрат на електроенергію порівняно з high-end процесором та багатий набір переферійних пристроїв, включає багато портів для вводу/виводу. Крім цього, мікроконтролери можуть бути запрограмовані велику кількість разів та існує багато розробок та комплектів засобів для створення прототипів та досягнення мети.

Пам’ять також відіграє важливу роль для повноцінної роботи вузла. існують різні види пам’яті, які використовуються в мікроконтролерах:

– RAM (Random Access Memory) – оперативна пам’ять, швидка, в основному використовується для передачі даних;

– ROM і EEPROM (Read only memory / в електронному вигляді Erasable Programmable ROM). Використовується для зовнішнього зберігання даних, програмного коду і програмування циклу (в байтах);

– флеш – пам’ять, використовується в програмі код пам’яті.

Трансивер – це комунікаційний пристрій, який містить в собі передавач та приймач, який дозволяє встановити безпровідний зв'язок між сенсорними вузлами. Основна задача, трансивера в сенсорних вузлах полягає в отриманні і передачі даних за допомогою радіохвиль на певній частоті. Основними компонентами трансивера являються радіочастоти building block (виконує обробку аналогових сигналів) та сигналів блоку будівлі (виконує цифрову обробку сигналу). Існує багато виробників, які пропонують трансивери, але особливо підходять для БСМ, які включають всі необхідні ланцюги для операцій прийому та передачі сигналу [5].

1.1.2 Операційна система сенсорного вузла

В наш час, найбільш використаною операційною системою БСМ є TinyOS, розроблена в університеті Берклі спеціально для використання в БСМ. TinyOS – ОС класу з відкритим початковим кодом, характерною якість яких є компонентна архітектура, подієве управління, моделі та статичне виділення пам'яті. Це забезпечує мінімальну кількість коду, який необхідний для вузлів БСМ, з строгими обмеженнями на об'єм пам'яті та енергозабезпечення малих автономних ресурсів. TinyOS – операційна система реального часу, призначена для роботи в умовах обмежених обчислюваних ресурсів. Забезпечуючи можливість встановлення автоматичного зв'язку вузлів з їх сусідами та формування даного сенсору топології мережі. TinyOS має бібліотеку компонентів, які складаються з мережевих протоколів, драйверів, давачів, утиліти генерації та збір інформації, яка може бути покращена в клієнтських додатках. Реалізована в TinyOS модель подій дозволяє контролювати потужність до найнижчого рівня, що дая можливість економити енергію.

Увага розробників TinyOS зосереджена на забезпеченні можливостей та мінімізації потреби енергії для мови програмування з високим рівним абстракції. В результаті чого, операційна система була створена з простою, але досить таки розвиненою компонентною архітектурою, специфіка полягає в

тому, що – надавання передових та надійних механізмів для паралельного виконання задач, маючи дуже обмежені ресурси.

TinyOS – це операційна система, яка підтримує модульність та події, які ґрунтуються на програмних парадигмах, тому вона підходить для безпровідних сенсорних вузлів. Тому, модульність TinyOS дозволяє їй бути основним драйвером для реалізації різних БСМ додатків. TinyOS не звичайна операційна система, а скоріше всього структура програмування (з базою коду менше 400 байт) для вбудованих систем (також, добре підходить для сенсорних вузлів) – набір компонентів, які дозволяють розробляти різні додатки, і повторне використання вже існуючих компонентів [6].

Класифікацію сенсорних вузлів наведено нижче.

Сенсорний вузол Mica2 (рисунок 1.2):

- маленька безпровідна платформа для сенсорної мережі;
- використовує батареї AA (більше чим на 1 год);
- може використовуватися в якості маршрутизатора;
- має багатоканальний радіо трансивер;
- облаштований роз’ємом розширення для зовнішніх датчиків.

Рисунок 1.2 – Сенсорний вузол Mica2

Сенсорний вузол XYZ (рисунок 1.3):

- open source wireless зондування платформи;
- забезпечений OKIML67 Q500x ARMTHUMB процесором та CC2420 Chipcon radio (IEEE 802.15.4);
- підтримує різні режими сну;

– може працювати на різних швидкостях і конфігурувати потужності.

Рисунок 1.3 - Сенсорний вузол XYZ

Сенсорний вузол Eyes:

– три роки вивчення та дослідження безпровідних сенсорних мереж (власна організація енергетичної ефективності сенсорних вузлів);

– невелика платформа сенсорного вузла, заснована на MSP430 контролері;

– забезпечений RFM10001low живленням радіо модуля.

Рисунок 1.4 – Сенсорний вузол Eyes

1.1.3 Характеристика безпровідної технології стандарту 802.15.4 ZigBee

Як і все сімейство стандартів 802.15, стандарт 802.15.4 призначений для організації бездротових персональних і сенсорних мереж WPAN, відмінною рисою яких є невеликий радіус дії мережевих пристроїв. Стандарт описує два нижніх рівні протоколів моделі взаємодії відкритих систем OSI:

– фізичний (PHysicalLayer, PHY);

– канальний підрівень доступу до середовища передачі (MediumAccessControl, MAC).

Мережевим вузлом є трансивер стандарту 802.15.4 з управлінням маршрутизацією стеком ZigBee та програмним профілем. Якщо до трансивера підключається сенсор, вузол отримує профіль сенсорного вузла. Цей профіль наказує йому збирати дані та відправляти вузлу, який являється центром збирання даних. Цей вузол має профіль центру збирання даних, який наказує йому чекати і збирати дані, які надходять від сенсорів. Вузол також може являтися координатором ZigBee, але такого може і не бути [7].

Функціональні можливості фізичних пристроїв дозволяють побудувати мережу з базовими топологіями (рисунок 1.5): Star (зірка), ClasterTree (кластерне дерево) та Peer-to-Peer або Mesh (кожен з кожним).

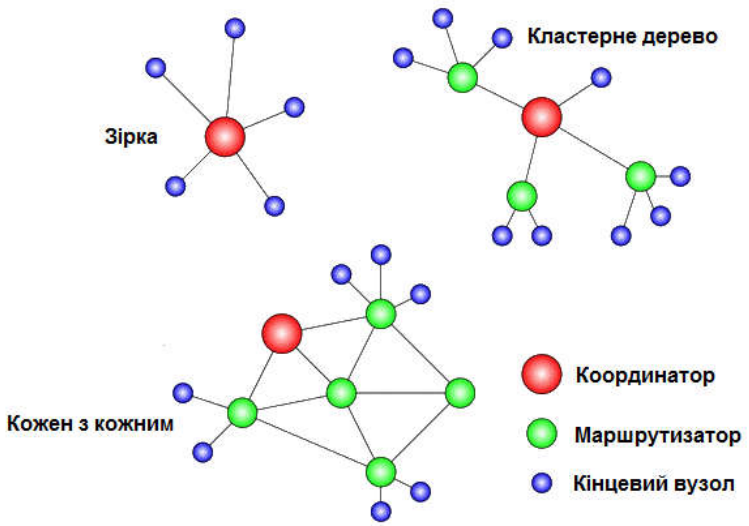

Рисунок 1.5 – Базові топології сенсорних мереж

1.1.4 Особливості побудови БСМ

У високошвидкісних мережах передачі даних механізм управління, як правило, розглядається в відриві від характеру процесів, які виникають при встановленні транспортних з’єднань. Для дослідження таких процесів в останній час застосовують підходи, засновані на аналізі їх фрактальності та

використанні концепції активного мережевого середовища. Сенсорна мережа може розглядатися як складний об’єкт, який знаходиться під впливом регулюючих стохастичних впливів, котрі формують локальні флуктуації інтенсивності телеметричного та іншого пакетного трафіку.

Для створення механізмів управління інформаційними процесами БСМ необхідно враховувати те, що вплив різних факторів може призвести до появи труднощів пов’язаних з динамікою процесів передачі даних. Зміни величини затримки, які виникають у віртуальному каналі при переході між різними протокольними стеками призводить до появи не стаціонарності в процесах комутації пакетів та MAC – адресів. Це в свою чергу впливає на характер зміни величини затримки в каналах передачі даних в кластерах сенсорної мережі, що призводить до відхилень від середніх значень [8].

Безпровідну сенсорну мережу , з точки зору складових фрагментів можна розділити на дві частини:

– кластерні мережі доступу з кінцевими сенсорними вузлами, які зв’язані з координатором. Тут використовується широкомовне опитування вузлів з підтримкою режимів управління енерговикористанням, фізична схема MAC – адресації, передача інформації без встановлення з’єднання з використанням логічних IP – адресів, фрагментація кадрів даних на комірки однакової довжини для синхронізації роботи вузлів.

– комірково – магістральні БСМ з ретрансляторами, які підтримують функцію маршрутизації, і шлюзами в інші мережі. Використовується підтримка технології логічної комутації комірок для синхронізації, алгоритми динамічної маршрутизації в проміжних ретрансляторах, принцип самоорганізації мережі, підтримка фізичних та логічних адресів, схем нумерації віртуальних машин.

Кластерні мережі мають більш просту реалізацію канального рівня, але не надають засобів безпечної доставки даних, так як для зниження затримки тут не використовується механізм підтвердження доставки кадрів та комірок, в силу чого затрати комірок формують основу для появи фрактальних властивостей

трафіку. З іншої сторони комутація комірок фіксованої довжини дає можливість забезпечити синхронізацію переходу вузлів з сонного в активний режим для управління їх енергозатратами, а також час збору сенсорної інформації, її обробки та ретрансляції, виходячи з цього, гарантує заданий доступний рівень затримок у випадку передачі ізохронних даних. Крім цього, можна використовувати великий об’єм управлінської інформації для підвищення ефективності алгоритмів розділення ресурсів та управління станом мережі.

Магістральна складова БСМ володіє розширеними можливостями щодо інтеграції різних інформаційних додатків та сервісів прикладного рівня, як чутливих, так і нечутливих до затримок даних. Визначення складності при синтезі моделей інформаційних додатків викликають різницю з організації каналів передачі, які керують впливом між різними підмережами.

1.2 Класифікація основних методів та алгоритмів стиснення даних

Стиснення даних – це алгоритмічне перетворення даних, яке виготовлене з ціллю зменшення їх об’єму. Стиснення даних використовується для більше раціонального використання пристроїв збереження та передачі даних. Синоніми – це пакування даних, компресія, стиснене кодування, кодування ресурсу. Обернена процедура називається відновленням даних (розпакування, декомпресія).

Стиснення засновано на усуненні надмірності, яке міститься у вихідних даних. Простим прикладом надмірності є повторення в тексті фрагментів (слова природньої чи машинної мови). Подібна надмірність зазвичай змінюється заміною повторним послідовним посиланням на уже закодований фрагмент з вказуванням його довжини.

Інший вид надмірності пов’язаний з тим, що деякі значення в стиснених даних зустрічаються частіше інших. Скорочення об’єму даних досягається за рахунок заміни даних, які часто зустрічаються короткими кодовими словами, а рідше – довшими (ентропійне кодування). Стиснення даних, які не володіють властивістю надмірності, принципово неможливо без втрат.

В основі будь – якого методу стиснення покладена модель джерела даних, точніше модель надмірності. Іншими словами, для стиснення даних використовуються деякі свідчення про те, якого роду дані стискаються. Не володіючи такими свідченнями про джерело, неможливо зробити ніяких припущень про перетворення, яке би дозволило зменшити об’єм повідомлення. Модель надмірності може бути статичною, незмінною для всього стискаючого повідомлення або будуватися на етапі стискання (і відновлення).

Методи, які дозволяють на основі вхідних даних змінювати модель надмірності інформації, називаються адаптивними. Неадаптивними є вузькоспеціалізовані алгоритми, які застосовуються для роботи з даними, які володіють добре визначеними та незмінними характеристиками. Переважна частина досить універсальних алгоритмів в тій чи іншій мірі є адаптивними[9].

1.2.1 Класичні алгоритми Зіва – Лемпела

Алгоритми словникового стиснення Зіва–Лемпела з’явилися у другій половині 1970-х років. Це були алгоритми LZ77 та LZ78, розроблені спільно Зівом (Ziv) та Лемпелом (Lempel). В подальшому початкові схеми піддавалися багатьом змінам, в результаті чого ми сьогодні маємо десятки самостійних алгоритмів и велику кількість модифікацій.

LZ77 та LZ78 являються універсальними алгоритмами стиснення, в яких словник формується на основі вже обробленої частини вхідного потоку. Відмінністю є лише спосіб формування виразів. В модифікаціях початкових алгоритмів ця властивість зберігається. Тому, словникові алгоритми Зіва-

Лемпела поділяються на два сімейства – алгоритми типу LZ77 і алгоритми типу LZ78. Часом згадуються словникові методи LZ1 та LZ2.

Публікації Зіва і Лемпела мали чисто теоретичний характер, так як ці дослідження справді займалися проблемою виміру «складності» стрічки, і застосування випрацюваних алгоритмів для стиснення даних є лише частковим результатом. Потрібний був деякий період часу, щоб ідея організації словника досягла розробників програмного і апаратного забезпечення. Тому, практичне використання алгоритмів почалось лише через декілька років.

З того часу методи даного сімейства незмінно являються найбільш популярними серед всіх методів стиснення даних, хоча в останній час ситуація починає мінятися на користь BWT та PPM, як забезпечувати краще стиснення. Крім того, практично всі використані словникові алгоритми відносять до сімейства Зіва-Лемпела [10].

Алгоритм LZSS дозволяє досить гнучко поєднувати у вихідній послідовності символи та вказівники, яка проявляється в регулярній передачі одного символу в прямому вигляді. Ця модифікація була запропонована у 1982 році Сторером та Жиманскі.

Ідея алгоритму основана на тому, що до кожного вказівника та символу додається однобітний префікс *f,* який розділяє ці об’єкти. Іншими словами, однобітний прапорець *f* вказує тип та довжину даних, які слідують за ним. Така техніка дозволяє:

– записує символи в явному вигляді, коли відповідний їм код має велику довжину, і відповідно словникове кодування тільки шкодить;

– обробляє символи, які раніше ні разу не зустрічались.

Алгоритм LZ78 був опублікований у 1978 році, і став «батьком» сімейства словникових методів LZ78.

Алгоритми даної групи не використовують ковзаючого вікна і в словник поміщають не всі стрічки, які зустрічаються при кодування, а лише перспективні з точки зору вірогідності повторного використання. На кожному кроці в словник додається нова фраза, яка являє собою щеплення однієї фрази з

S словника, яка має саме довше співпадання зі стрічкою буфера, і символу *s*. Символ *s* є символом, наступним за стрічкою буфера, для якої знайдена фраза співпадання S. У відмінності від сімейства LZ77, в словнику не може бути однакових фраз.

Кодувальник породжує тільки послідовність кодів фраз. Кожний код складається із номеру (індексу) *n*, «батьківської» фрази *S*, чи префікса, та символу *s*. В початку обробки словник є пустим.

Далі, теоретично, словник може зростати безкінечно, на зріст алгоритм не накладає обмежень. На практиці при досягненні певного об'єму пам'яті словник має очищуватися повністю чи частково.

В таблиці 1.1 наведено декілька характерних алгоритмів LZ. Вказаний список далеко не повний, реально існує в декілька разів більше алгоритмів, які засновані на LZ77, або на LZ78.

Таблиця 1.1 – Характерні алгоритми сімейства LZ

№	Назва	Автор, рік	Тип алгоритму
1	LZMV	Міллер та Вегнам, 1984	LZ78
2	LZW	Уелч, 1984	LZ78
3	LZB	Белл, 1987	LZSS
4	LZH	Брент,1987	LZSS
5	LZFG	Файєле та Гріні, 1989	LZ77
6	LZBW	Бендер та Вулф, 1991	LZ77
7	LZRW1	Уіллямс, 1991	LZSS
8	LZCB	Блум, 1995	LZ77
9	LZP	Блум,1995	LZ77
10	LZ77-PM, LZFG-PM, LZW-PM	Хоанг, Лонг, Віттер, 1995	LZ

Також, відомо гібридні схеми, які поєднують обидва алгоритми щодо побудови словника:

– LZMV – алгоритм сімейства LZ78, цікавий способом своєї побудови словника: нова фраза створюється за допомогою конкатенації двох останніх

фраз, а не конкатенації фрази та символу, тобто словник наповнюється «агресивніше»;

– LZW – модифікація LZ78, за рахунок попереднього занесення в словник всіх символів алфавіту вхідний результат роботи LZW складається тільки із послідовності індексів фрази словника;

– LZB – модифікація LZSS, зміни чіпляють кодування вказівників, зміщення задається змінною кількістю бітів в залежності від реального розміру словника;

– LZH – модифікація LZSS, аналогічна LZB, але для стиснення заміщення та довжини відповідності використовуються коди Хафмана;

– LZFG – модифікація LZ77, являє собою декілька алгоритмів. Ідея найбільш складного, складається в кодуванні парою <довжина, зміщення>, а індексом відповідним фразі вузла дерева цифрового пошуку;

– LZBW – спосіб модифікації алгоритмів сімейства LZ77, за рахунок обліку вже існуючих в словнику однакових виразів дозволяє зменшити кількість бітів, які необхідні для кодування довжини спів падання;

– LZRW1 – алгоритм є модифікацією LZSS, точніше алгоритму A1 групи LZFG, та розроблений з метою забезпечення максимальної швидкості стиснення та розтиснення;

– LZCB – це ціла група алгоритмів, які є тією чи іншою модифікацією LZ77, основна ідея в тому, що введення досить сильного обмеження на мінімальну довжину співпадання – від 4 символів і більше;

– LZP – заснований на LZ77, для кожного вхідного символу стрічка із попередніх символів розглядається, як контекст довжини. Цей алгоритм справедливий для алгоритму LZP1. Перевагою якого є висока швидкість;

– LZ77-PM, LZFG-PM, LZW-PM – модифікація алгоритмів LZ, використовується декілька контекстно – залежних словників, а не один словник. В контекстно – залежний словник потрапляють тільки стрічки, які зустрічаються після кінцевої послідовності символів. Кодування стрічки буфера відбувається за допомогою одного чи декількох словників, номера яких

визначаються останніми закодованими символами. Облік контексту дозволяє значно покращити стиснення вихідних словникових схем [11].

1.2.2 Кодування Хафмана

Кодування Хафмана є простим алгоритмом для побудови кодів змінною довжини, які мають мінімальну середню довжину. Цей досить популярний алгоритм є основою більшості комп'ютерних програм стиснення текстової та графічної інформації. Деякі із них використовують алгоритм Хафмана, а інші беруть його в якості одного із ступенів багаторівневого процесу стиснення. Метод Хафмана здійснює ідеальне стиснення, якщо вірогідність символів точно рівне негативним степенем числа. Алгоритм починає будувати кодове дерево знизу вверх, потім сковзає вниз по дереву, щоб добудувати кожний індивідуальний код справа на ліво. Починаючи з робіт Д. Хафмана 1952 року, цей алгоритм являється предметом багатьох досліджень.

Алгоритм починається складання списку символів алфавіту в порядку зменшення їх вірогідності. Потім, від кореня будується дерево, листям якого служать ці символи. Це відбувається покроково, крім цього на кожному кроці вибираються два символи з найменшими вірогідностями, і додаються наверх частинного дерева, видаляються із списку та замінюються допоміжним символом.

Допоміжному символу приписується вірогідність, рівна сумі вірогідностей, вибраних на цьому кроці символів. Коли список скорочується до одного допоміжного символу, який представляє цілий алфавіт, дерево оголошується побудованим. Алгоритм завершується спусканням з дерева і побудовою коду всіх символів [12].

Краще всього показати цей алгоритм на простому прикладі, є п'ять символів з вірогідностями, показано на рисунку 1.6.

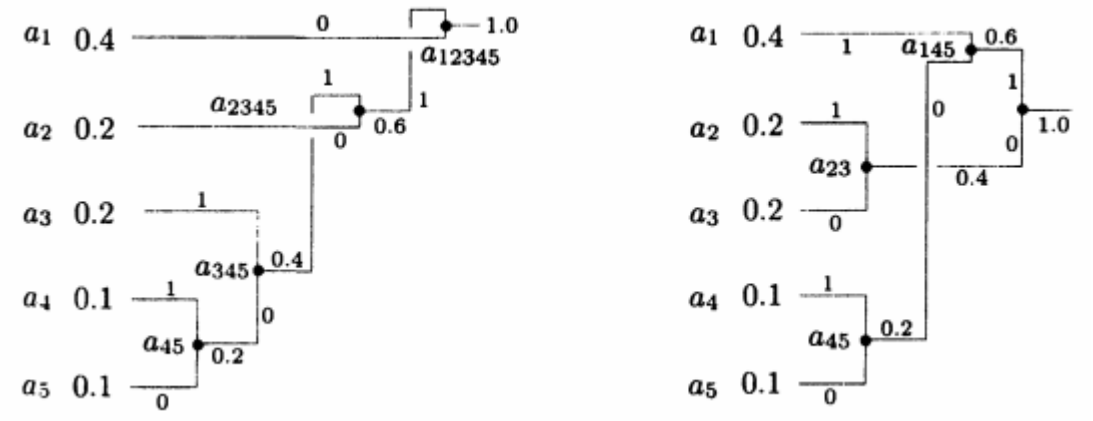

Рисунок 1.6 – Коди Хафмана

Перед тим, як розпочати стиснення потоку даних, компресор (кодер) має побудувати коди. Це робиться з допомогою вірогідностей (чи частоти появи) символів. Вірогідності і частоти потрібно записати в стиснений файл для того, щоб декомпресор (декодер) Хафмана міг здійснити декомпресію даних. Це легко зробити, так як частоти є цілими числами, а вірогідності також представлені цілими числами. Зазвичай це призводить до добавлення декількох сотень байтів у стиснутий файл. Можна записати в стиснутий файл і самі коди, але це вносить додаткові труднощі, оскільки коди мають різні довжини. Ще можна записувати в файл саме дерево Хафмана, але для цього буде необхідна більша кількість об’єму, чим простий запис частот.

В будь-якому випадку, декодер має прочитати початок файлу і побудувати дерево Хафмана для алфавіту. Тільки після цього він може читати і декодувати весь файл. Алгоритм декодування дуже простий. Потрібно почати з кореня і прочитати перший біт стиснення файлу. Якщо це нуль, потрібно рухатись по нижній вітці дерева; якщо це одиниця, то рухатися потрібно по верхній частині дерева. Далі прочитується другий біт і проходить рух по наступній вітці в напрямку до листя. Коли декодер досягне листя дерева, він дізнається код першого нестисненого символу. Процедура повторюється для наступного біта, починаючи знову з кореня дерева. Дана процедура зображена на рисунку 1.7 для алфавіту з 5 символів.

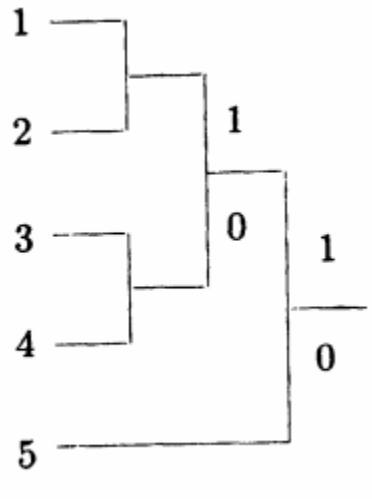

Рисунок 1.7 – Коди Хафмана з рівними вірогідностями

Метод Хафмана передбачає, що частоти символів алфавіту відомі декодеру. На практиці це буває дуже рідко. Одне можливе рішення для компресора – це прочитати вихідні дані два рази. В перший раз, щоб вирахувати частоти, а в другому випадку, щоб зробити стиснення. В проміжку компресор будує дерево Хафмана. Такий метод називається напівадаптивним і він працює повільно для практичного використання. На практиці застосовується метод адаптивного (чи динамічного) кодування Хафмана. Цей метод лежить в основі програми compact операційної системи UNIX [13].

Основна ідея полягає в тому, що і компресор, і декомпресор починають з пустого дерева Хафмана, а потім модифікує його по мірі читання та обробці символів. І компресор, і декомпресор повинні модифікувати дерево однаково, щоб весь час використовувати один і той самий код, який може змінюватися в ході процесу. Можна сказати, що копмресор і декомпресор синхронізовані, якщо їх робота погоджена. Слово «дзеркально» найкраще означає суть їхньої роботи. Декодер дзеркально повторює операції кодера.

Проте, декодер має знати чи є даний зразок просто нестисненим символом чи це код змінної довжини. Для подолання двозначності, кожному незжатому символу буде передувати спеціальний esc (escape) код змінної довжини. Коли декомпресор читає цей код, він точно знає, що за ним слідують 8 біт коду ASCII, який вперше з'явився на вході.

1.3 Програмні засоби для стиснення даних

Класичними форматами стиснення даних, які широко використовуються в щоденній роботі з комп'ютером, являються формати .ZIP та .ARJ. В останній час до них добавився популярний формат .RAR. Недивлячись на те, що засоби архівації, призначені для ОС MS-DOS, вони можуть працювати під управлінням Windows 9x (у вікні сеанс MS-DOS), користуватися ними не рекомендовано. В першу чергі, це пов'язано з тим, що при обробці файлів виникає втрата «довгих імен» фаулів та заміна їх іменами MS-DOS по специфікації 8.3. Це може створити користувачу документу певні незручності, а у випадку, коли архівація проходить з метою резервної копії, втрата «довгих імен» взагалі не доступна.

Сучасні програмні засоби для створення та обслуговування архівів відрізняються великим об'ємом функціональних можливостей, багато з яких виходять далеко за рамки простого стиснення даних і ефективно доповнюють стандартні засоби операційних систем. В даному випадку сучасні засоби архівації даних називають диспетчерами архівів. До базових функцій, які виконують більшість сучасних диспетчерів архівів відносять:

- створення нових архівів;
- додавання файлів в архів, який вже існує;
- створення розпределених архівів на носіях малого об'єму;
- тестування цілісної структури архіву;
- повне чи часткове відновлення пошкоджених архівів.

В тих випадках коли архівація виконується для передачі документу користувачу, потрібно врахувати те, чи у нього є необхідний програмний засіб для вилучення початкових даних із архіву. Якщо таких засобів у користувача немає створюють архів, який може сам розпакуватися. Саморозпаковуючий архів створюється на базі звичайного архіву шляхом приєднання до нього невеликого програмного модуля. Сам архів отримує розширення з ім'ям .EXE,

яке характерне для виконуючих файлів. Користувач може виконати його запуск як програми, після чого розпакування пройде на його комп'ютері автоматично.

В тих випадках, коли необхідна передача вкликого архіву на носіях малого об'єму, наприклад на гнучких дисках, можливе розпределення одного архіву у вигляді дрібних фрагментів на декількох носіях.

Деякі диспетчери виконують розбивання зразу на гнучкі диски, а деякі дозволяють виконувати попереднє розбивання архіву на фрагменти заданого розміру на жорсткому диску. Після чого їх можна перенести на зовнішній носій шляхом копіювання. При створенні рощпределених архівів диспетчер WinZip має неприємну особливість: кожний том несе файли з однаковими іменами. В результаті цього не має можливості встановити номеритомів, котрі зберігаються на кожному з гнучких дисків, по назві файлу.

Серед великої кількості LZ – архіваторів слід відмітити такі:

- 7-Zip, автор Ігор Павлов;
- ACE, автор Маркел Лемке;
- ARJ, автор Роберт Джанг;
- ARJZ, автор Булат Зіганшин;
- CABARC, корпорація Microsoft;
- Imp, фірма Technelysium Pty Ltd;
- JAR, автор Роберт Джанг;
- PKZIP, фірма PKWARE Inc;
- RAR, автор Євгеній Рошал;
- WinZip, фірма Nico Mak Computing;
- Zip, Info-ZIP group.

Ці архіватори являються чи одними із найефективніших в класі застосовуючих методів Зіва – Лемпела, чи користуються популярністю, чи зробили значний вплив на розвиток словникових алгоритмів, чи цікаві з точки зору деяких вказаних критеріїв. За винятком 7-Zip, словникові алгоритми всіх вказаних архіваторів можна розглядати як модифікацію LZH. В алгоритмі

LZMA, реалізованому в 7-Zip, спільно з словниковими замінами використовується контекстне моделювання і арифметичне кодування.

В таблиці 1.2 наведено результати порівняння деяких архіваторів в степені стиснення файлів набору CalgCC.

Таблиця 1.2 – Результати порівняння архіваторів

	ARJ	PKZIP	ACE	RAR	CABARC	7-Zip
Bib	3.08	3.16	3.38	3.39	3.45	3.62
Book1	2.41	2.46	2.78	2.80	2.91	2.94
Book2	2.90	2.95	3.36	3.39	3.51	3.59
Geo	1.48	1.49	1.56	1.53	1.70	1.89
News	2.56	2.61	3.00	3.00	3.07	3.16
Obj1	2.06	2.07	2.19	2.18	2.20	2.26
Obj2	3.01	3.04	3.39	3.38	3.54	3.96
Paper1	2.84	2.85	2.91	2.93	2.99	3.07
Paper2	2.74	2.77	2.86	2.88	2.95	3.01
Pic	9.30	9.76	10.53	10.39	10.67	11.76
Progc	2.93	2.94	3.00	3.01	3.04	3.15
Progl	4.35	4.42	4.49	4.55	4.62	4.76
Progp	4.32	4.37	4.55	4.57	4.62	4.73
Trans	4.65	4.79	5.19	5.23	5.30	5.56
Всього	3.47	3.55	3.80	3.80	3.90	4.10

Використані версії архіваторів: ARJ 2.50a, PKZIP 2.04g, WinRAR 2.71, ACE 2.04, 7-Zip 2.30 beta7. У всіх випадках застосовувався алгоритм LZ, який забезпечує найкраще стиснення. Слід замітити, що 7-Zip використовує спеціальні медоти препроцесінгу нетекстових даних, «відключити» які не вдалося, що трохи зіпсувало загальний вигляд. Тим самим, перевага цього архіву на даному текстовому наборі безперечна. У випадку WinRAR та ACE режим мультимедійної компресії спеціально не вмикався.

При порівнянні 7-Zip та RAR з іншими LZ-архіваторами необхідно слідкувати, щоб 7-Zip та RAR використовували алгоритм типу LZ, оскільки вони мають в своєму арсеналі алгоритм PPMII, що забезпечує високий степінь стиснення тексту.

При порівнянні слід враховувати, що швидкість стиснення ARJ та PKZIP була приблизно у чотири рази вище, чим у RAR та ACE, які в свою чергу були швидші CABARC та 7-Zip приблизно на 30%. Розмір словника в ARJ та PKZIP в десятки разів менший, ніж в інших програмах.

У більшості випадків захист архівів виконують за допомогою паролю, який запитується при спробі переглянути, розпакувати чи змінити архів. Теоретично, захист за допомогою паролю не досить добре виконує свою функцію і його не рекомендовано використовувати для важливої інформації. В той же час, необхідно відмітити, що основні програмні засоби, які використовуються для відновлення втраченого паролю, використовують методи прямого перебору. Роботу цих засобів можна значно погіршити, якщо розширити область перебору.

Паролі на базі символів англійського алфавіту і цифр дійсно знімаються дуже швидко. Однак навіть незначне збільшення числа використовуваних символів за рахунок розділових знаків багаторазово збільшує криптостійкість захисту, а використання також і символів російського алфавіту може повністю спростувати спроби зняти пароль шляхом перебору, зробивши терміни роботи неприйнятними.

1.4 Аналіз завдання дипломної роботи та постановка задач дослідження

В даному дипломному проекті проводиться дослідження алгоритмів стиснення даних, які використистовуються в безпровідних сенсорних мережах. Проведено комплексний аналіз технічних характеристик, програмного забезпечення для безпровідних сенсорних мереж з використанням алгоритмів стиснення даних.

Аналіз характеристик вибраної технології побудови безпровідної сенсорної мережі показав, що найбільш ефективним варіантом є використання

готових модулів. Розглянуто складові сенсорного вузла та їх технічні характеристики, приведено приклади деяких сенсорних вузлів та їх окремих частин. Наведено можливі топології сенсорних мереж. У безпровідних сенсорних мережах відбувається вільний обмін інформацією, цей обмін може бути як деструктивним та конструктивним.

Метою роботи є дослідження алгоритмів та методів стиснення даних та здійснення програмної реалізації алгоритмів стиснення даних.

Для досягнення поставленої мети необхідно розв’язати такі задачі:

– дослідити безпровідні сенсорні мережі їх характеристики та особливості;

– здійснити класифікацію основних методів та алгоритмів стиснення даних;

– дослідити програмні засоби для стиснення даних;

– аналізувати вимоги до алгоритмів стиснення даних в безпровідних сенсорних мережах;

– розробити асиметричний за складністю алгоритм стиснення даних;

– здійснити програмну реалізацію алгоритму стиснення даних;

– дослідити обчислювану складність алгоритму стиснення даних;

– дослідити функціональні параметри безпровідної сенсорної мережі з використанням алгоритму стиснення.

2 АЛГОРИТМИ КОДУВАННЯ ДЕКОДУВАННЯ В БЕЗПРОВІДНИХ СЕНСОРНИХ МЕРЕЖАХ

2.1 Аналіз алгоритмів кодування в безпровідних сенсорних мережах

Кодування – це зображення повідомлень (інформації) послідовністю елементарних символів. Символи в повідомленнях можуть відноситись до алфавіту, який включає n-кількість букв (буква – символ повідомлення). Проте кількість елементів коду k значно обмежене, тому n>k. Тому, якщо відношення сигнал/перепона для надійного розпізнавання рівня сигналу має бути не менше значення q, тоді найменша амплітуда для представлення одного з k символів має бути qg, де g – амплітуда перешкоди, а найбільша амплітуда відповідно qgk. Потужність передавача пропорційна квадрату амплітуди сигналу повинна перевищувати величину пропорційну $(qgk)^2$. У зв'язку з цим поширено двійкове кодування з k=2. При двійковому кодуванні повідомлень з n типами букв, кожна з n букв кодується певною комбінацією 1 та 0.

Теореми Шенона відіграють важливу роль в теорії кодування. Перша теорема Шенона надає можливість створення системи ефективного кодування повідомлень, у якій середнє число двійкових символів на один символ повідомлення асимптотично прагне до інформаційної ентропії джерела повідомлень (при відсутності перепон). Друга теорема Шенона відноситься до умов надійної передачі даних по ненадійним каналам [14].

Нехай, потрібно передати послідовність символів, які з'являються з відповідними можливостями, при цьому є деяка вірогідність того, що символ передавання в процесі передачі буде спотворений. Згідно теореми Шенона можна вважати, що можна вказати таке, залежне тільки від розглянутих можливостей додатне число v, що при малому числі e>0 існують способи передачі з швидкістю v'(v'<v), яка близька до v, яке дає можливість відновлювати початкову послідовність з вірогідністю помилки, менше e. в той самий час при швидкості передачі v', більшої v, це неможливо. Згадані способи

передачі даних використовують належні «перешкодостійкі» коди. Критична швидкість v визначається співвідношенням:

$$Hv=C, \tag{2.1}$$

де H – ентропія джерела на символ;

C – ємність (пропускна здатність) каналу в двійкових одиницях в секунду.

Однією з форм представлення даної теореми може слугувати відношення Хартлі – Шенона, що представлене формулою

$$C=2F\log_2 k, \tag{2.2}$$

де C – пропускна здатність (біт/с);

F – смуга пропускання лінії (Гц);

$k \leq 1+A$, A – відношення сигнал/перепона.

Для кодування текстових документів широко використовуються двійкові коди: EBCDIC (Extended Binary Coded Decimal Interchange Code) – символи кодуються вісьмома бітами, популярний через те, що широко використовується IBM; ASCII (American Standards Committee for Information Interchange) – семибітовий двійковий код.

Обидва коди включають бітові комбінації для друкованих символів та деяких популярних команд слів типу NUL, CR, ACK, NAK та інші.

Моментальний код – це код, в якому кодові слова не залежать від наступних символів повідомлення, тому символи можуть кодуватися та декодуватися зразу після їх приходу в кодек (пристрій чи програма, яка виконує операцію кодування та декодування даних). Код являється моментальним, якщо виконується нерівність Крафта

$$\sum_{i=1}^{m} r^{-q_i} \leq 1, \tag{2.3}$$

де r – кількість знаків в алфавіті коду, що використовується;

m – кількість символів джерела;

q_i – довжина і-го кодового слова.

Наприклад, у випадку двійкових блок-кодів, при r=2 та однакових q_i, рівних 8, отримуємо m≤64, отже восьмирозрядним двійковим кодом можна закодувати 64 різних символи [15].

Представлення повідомлень у вигляді коду зі зменшеною кількістю символів за рахунок зменшення надмірності чи за рахунок втрати малоістотної інформації називається стисненням (компресією) даних. Мірою надмірності є коефіцієнт надмірності повідомлення А, що визначається за формулою

$$r = \frac{I_{max} - I}{I_{max}}, \tag{2.4}$$

де I – кількість інформації в повідомленні А,

I_{max} – максимально можлива кількість інформації в повідомленні тієї ж довжини, що А.

Завдяки надмірності можливе стиснення інформації без її втрати в повідомленнях, які передаються. Для цього використовуються спеціальні алгоритми стиснення, які зменшують надмірність. Ефект стиснення оцінюють коефіцієнтом стиснення:

$$K = \frac{n}{q}, \tag{2.5}$$

де n – число мінімальних необхідних символів для передачі повідомлень (практично це число символів на виході еталонного алгоритму стиснення);

q – число символів в повідомленні, стиснутих даним алгоритмом.

Часто степінь стиснення оцінюють відношенням довжин кодів на вході та виході алгоритму стиснення.

З методами стиснення, які не зменшують кількість інформації в повідомленні, застосовують методи стиснення, які основані на втраті мало потрібної інформації.

Компресія та декомпресія даних здійснюється або на прикладному рівні за допомогою програм стиснення, або за допомогою апаратних засобів. Засоби чи програми, які застосовуються для компресії та декомпресії, називають кодеками. Слово «кодек» утворене з початкових складів слів «кодування-декодування».

Очевидний спосіб стиснення числової інформації, представленої в коді ASCII, полягає у використанні скороченого коду з чотирма бітами на символ замість восьми, так як передається набір, який включає тільки 10 цифр, символи «крапка», «кома» та «пробіл».

Одну з груп методів стиснення даних при аналого-цифровому перетворенні складають методи різничного кодування (кодування, яке опирається на передачу різниці чисел, які представляють сусідні відліки-значення амплітуд, величини, яка вимірюється). В цих методах передаються різниці чисел, які представляють сусідні відліки (значення амплітуд). Стиснення полягає в тому, що різниці амплітуд представлені меншим числом розрядів,чим самі амплітуди. Різничне кодування реалізовано в методах дельта-модуляції та її різновидами [16].

У випадку дельта-модуляції різниці амплітуд, що передаються представляються одно бітовими кодами, що зображують знаки «плюс» та «мінус», де «плюс» означає збільшення, а «мінус» - зменшення амплітуди порівняно з попереднім значенням на величину, яка відповідає одному рівню дискретизації.

Очевидно, що частота дискретизації при дельта-модуляції має відповідати швидкості зміни величини кодування, так як при недостатній частоті треба кодування більш чим одним розрядом.

Передбачувальні методи засновані на екстраполяції значень відліків амплітуди, і якщо виконано нерівність (формула 2.6), то відлік має бути переданий,. Інакше він являється збитковим.

$$\left|A_r - A_p\right| > d, \tag{2.6}$$

де A_r, A_p – амплітуди реального і передбачуваного відліків;

d – допустима похибка представлення амплітуд.

Графік передбачуваного методу з лінійною екстраполяцією зображено на рисунку 2.1.

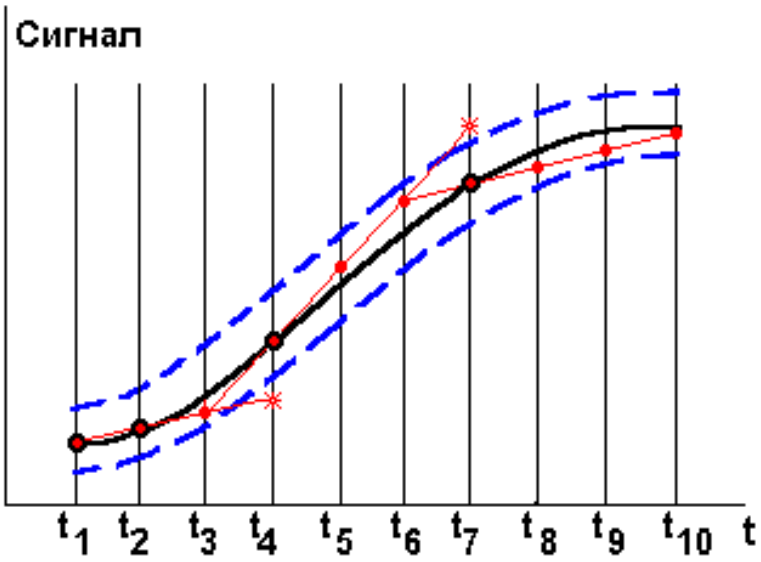

Рисунок 2.1 – Графік передбачуваного методу

На рисунку точками показано передбачувані значення сигналу. Якщо точка виходить за кордони «коридору» (допуску d), показаного пунктирними лініями, то відбувається передача відліку. Можна побачити, що відліки помічені темники кружками в моменти часу t_1, t_2, t_3, t_4. Якщо передачі відліку немає, то на приймальному кінці приймається екстраполярне значення.

Критерії порівняння алгоритмів стиснення даних використовуються для перегляду, аналізу та порівняння продуктивності, обмежень та невирішених питань кожного з алгоритмів стиснення:

– продуктивність стиснення;

– енергозбереження, яке залежить від передачі: даний критерій відстежує як коефіцієнт стиснення алгоритму впливає на передачу даних. Крім того, потужність, яка зберігається за рахунок стиснення даних;

– потужність, яка використовується алгоритмами при стисненні даних: критерій переглядає та порівнює складність кожного алгоритму стиснення в межах своєї категорії. Це відбувається за рахунок вимірювання чи аналізу алгоритму складності, або підраховуючи кількість базових операцій, що використовуються алгоритмом (додавання, порівняння та зміна). Тоді потужність використовується для виконання даних функцій;

– мережа енергозбереження: даний критерій можна виміряти за допомогою розрахунку різниці між енергозбереженням від зменшення передачі та потужністю, що використовується для виконання алгоритму. Результат є важливим фактором, який використовується для визначення того чи алгоритм стиснення підходить для використання в додатках безпровідних сенсорних мереж;

– розмір коду алгоритму: використовуючи датчик вузла пам’яті, даний критерій покликаний аналізувати та зіставляти розмір коду кожного алгоритму у своїй категорії;

– придатність використання класів стиснення даних (з втратами та без втрат): при обмеженнях датчиків точності цей критерій використовується для аналізу та порівняння того, правильно чи розроблений кожний алгоритм для відповідних даних;

– придатність для одного чи для різних типів даних: даний критерій розглядає кожний алгоритм, для визначення чи підходить придатність для різних типів даних для кожного вузла або для поодиноких типів даних кожного вузла.

2.1.1 Аналіз алгоритмів ентропійного кодування

Для стиснення даних необхідною умовою є вивчення одного теоретико – інформаційного поняття, а саме ентропію. Під ентропією символу a, яке має вірогідність P, розуміється кількість інформації, яка міститься в a, яка рівна - $\mathrm{P}\log_2 \mathrm{P}$. Якщо символи деякого алфавіту з символами від a_1 до a_n мають вірогідність від P_1 до P_n, то ентропія всього алфавіту рівна сумі $\sum_n -P_i \log_2 P_i$. Якщо задана стрічка символів цього алфавіту, то для неї ентропія визначається аналогічно.

За допомогою поняття ентропії теорія інформації зображує, як вираховувати вірогідності стрічок символів алфавіту, і передбачає її найкраще стиснення, тобто найменше в середньому число біт, яке необхідне для представлення даної стрічки символів.

Проаналізувавши ентропію алфавіту, який складається з двох символів a_1 і a_2 з вірогідностями P_1 і P_2. Оскільки, $\mathrm{P}_1 + \mathrm{P}_n = 1$, то ентропія цього алфавіту виражається числом – $\mathrm{P}_1 \log_2 \mathrm{P}_1 - (1 - \mathrm{P}_1) \log_2 (1 - \mathrm{P}_1)$. В таблиці 2.1 наведені різні значення величин P_1 і P_2 разом з відповідною ентропією.

Таблиця 2.1 – Вірогідності та ентропія двох символів

P_1	P_2	Ентропія
0,99	0,01	0,08
0,90	0,10	0,47
0,80	0,20	0,72
0,70	0,30	0,88
0,60	0,40	0,97
0,50	0,50	1.00

Коли $\mathrm{P}_1 = \mathrm{P}_2$, необхідно хоча б один біт для кодування кожного символу. Це означає, що ентропія досягла свого максимуму, надмірність буде рівна нулю, і дані стиснути буде неможливо. Проте, якщо вірогідність символів сильно відрізняється, то мінімальне число необхідних бітів на символ буде знижуватися.

Методи стиснення застосовуються не самостійно, а з використанням методів ентропійного кодування, що замінює символи їх кодовими словами – стрічками нулів та одиниць так, що символам, які зустрічаються найчастіше відповідають більш короткі слова [17].

Такі методи кодування є давно відомі, і їх можна розділити на дві великі групи: префіксні методи (метод Хафмана, Шенона, Шенона-Фано) та арифметичні.

Алгоритм Шенона – Фано – один з найперших розроблених алгоритмів стиснення. В основі даного алгоритму лежить ідея представлення більш частих символів за допомогою більш коротких кодів. При цьому коди, отримані за допомогою алгоритма Шенона-Фано, володіють властивістю префіксності, тобто ні один код не є початком іншого коду.

Алгоритм побудови коду Шенона-Фано:

– розбити алфавіт на дві частини, сумарні частоти символів в яких максимально близькі один до одного;

– у префіксний код першої частини символів добавити 0, в префекційний код другої частини символів добавити 1;

– для кожної частини (в якій не менше двох символів) рекурсивно виконати кроки 1-3.

Не дивлячись на простоту, алгоритм Шенона-Фано має недоліки, одним з яких є неоптимальність кодування. Хоча розділення на кожному кроці і є оптимальним, алгоритм, як і значна частина алгоритмів, не гарантує оптимального ресурсу в цілому.

На рисунку 2.2 наведено приклад дерева Шенона-Фано та отримані результати виконані на основі даного кодування.

Без використання даного кодування повідомлення буде займати 40 біт (при умові, що кожний символ кодується 4 бітами), а з використанням алгоритму Шенона-Фано 27 біт. Об’єм повідомлення зменшиться на 32,5%, але цей результат можна покращити з використанням алгоритму кодування за допомогою методу Хафмана.

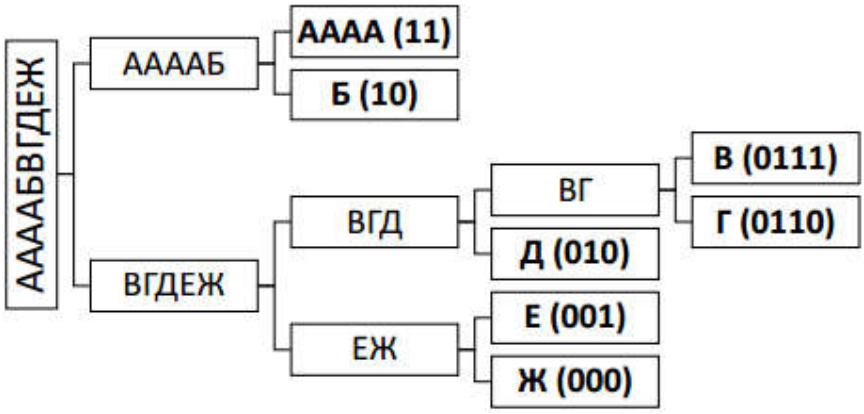

Рисунок 2.2 – Приклад дерева Шенона-Фано

Алгоритм кодування Хафмана, також володіє властивостями префіксності, крім цього, ще мінімальною кількістю надмірності, саме тому його дуже широко використовують. Для отримання кодів Хафмана використовують такий алгоритм:

– всі символи алфавіту наведені у вигляді вільних вузлів, при цьому вага вузла пропорційна частоті символів у повідомленні;

– із великої кількості вільних вузлів вибирають два вузла з мінімальною вагою та створюється новий (батьківський) вузол з вагою рівною сумі вазі вибраних вузлів;

– вибрані вузли видаляються з списку вільних, а створений на їх основі батьківський вузол додається в даний список;

– кроки 2 та 3 повторюються до того часу, коли в списку вільних більше одного вузла;

– на основі побудованого дерева кожному символу алфавіту присвоюється префікційний код.

Розглянемо дерево Хафмана та коди отримані для повідомлення на рисунку 2.3.

Легко підрахувати, що об’єм закодованого повідомлення складе 26 біт, що менше, чим в алгоритмі Шенона Фано.

Окремо потрібно відмітити, що крім популярності алгоритму Хафмана на даний момент існує велика кількість варіантів реалізації кодування Хафмана, в тому числі і адаптивне кодування, яке не потребує передачі частоти символів.

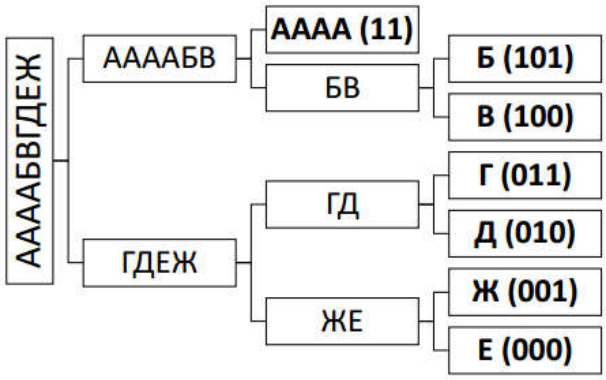

Рисунок 2.3 – Приклад дерева Хафмана

Серед недоліків даного кодування значну кількість складають проблеми, що пов'язані зі складністю реалізації. Використання для зберігання частоти символів змінних з'єднано з втратою точності, тому на практиці часто використовують цілочисельні змінні, але так як вага вузлів постійно зростає, тому рано чи пізно виникає переповнення.

Таким чином, не дивлячись на простоту алгоритму, його реалізація до цього часу може викликати деякі труднощі, особливо при кодуванні великої кількості елементів.

До методів ентропійного кодування можна віднести арифметичне кодування. Дане кодування одне з найбільш ефективних способів стиснення інформації. Арифметичне кодування дозволяє кодувати повідомлення з ентропією менше 1 біта на символ.

Якщо в алфавіті N символів $a_1, \dots, a_N$ з частотами $p_1, \dots, p_N$ відповідно. Кроки алгоритму арифметичного кодування мають наступний вигляд:

– в якості робочого напівінтервалу вибирається [0;1);

– робочий напівінтервал розбивається на N напівінтервалів, які не перериваються, при цьому довжина і-го напівінтервалу пропорційна p_1;

– якщо кінець повідомлення ще не досягнуто, то в якості нового робочого інтервалу вибирається і-й напівінтервал та відбувається перехід до кроку два. В іншому випадку, повернути будь – яке число з робочого напівінтервалу. Запис цього числа в двійковому коді і буде являти собою закодоване повідомлення.

На рисунку 2.4 наведено процес кодування повідомлення «АБААВ».

При декодуванні необхідно виконати аналогічну послідовність дій, тільки на кожному кроці необхідно додатково визначати, який саме символ був закодований.

Основним плюсом арифметичного кодування є його ефективність, а найбільшим недоліком – необхідність високоточної арифметики.

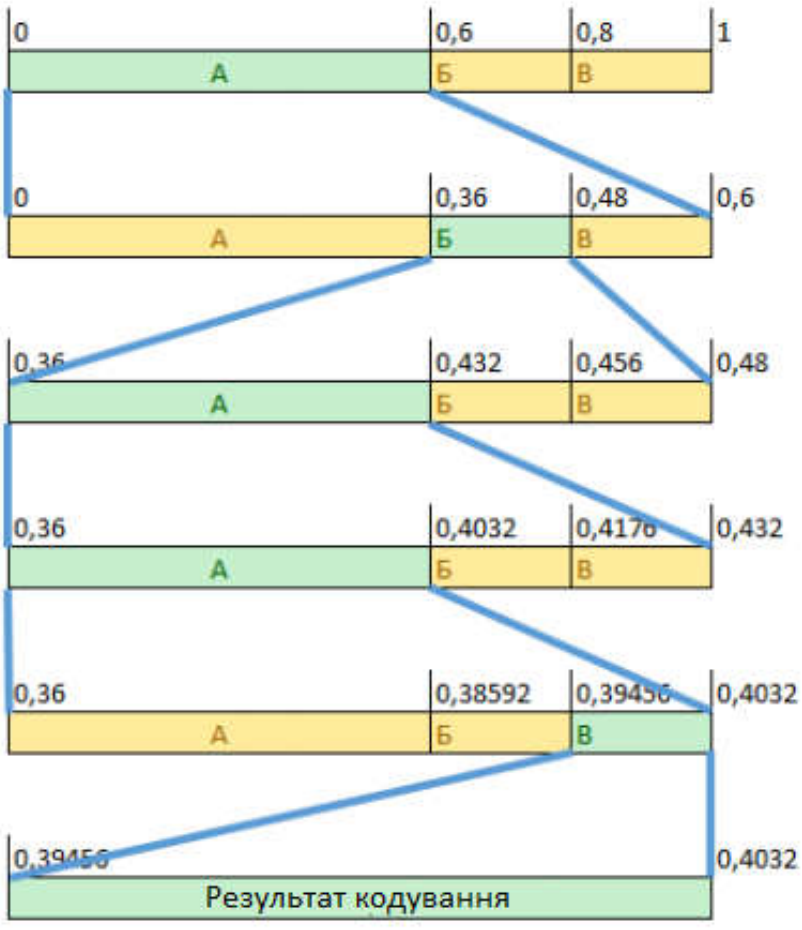

Рисунок 2.4 – Приклад арифметичного кодування

Отже, алгоритми словникового стиснення та стиснення за допомогою сортування певних блоків використовують для кодування виходу основного алгоритму стиснення кодів Хафмана, а арифметичне кодування зазвичай використовують в поєднанні з методами статистичного моделювання для кодування символів у відповідності з вірогідностями, які вже є відомі.

2.2 Алгоритм ентропійного кодування

Ентропійне кодування – це схема стиснення даних, яка привласнює коди символам так, щоб відповідати кодовій довжині з можливостями символів. Ентропійний метод стискає дані шляхом заміни даних символами, які представлені рівними по довжині кодовим словам, де довжина пропорційна негативному логарифму вірогідності.

Методика реалізується шляхом створення бінарного дерева вузлів. Вони можуть зберігатися у вигляді регулярної матриці, розмір якої залежить від кількості символів (n).

В теорії стиснення інформації та даних етап ентропійного кодування є останнім етапом алгоритму стиснення, де всі відомі результати від моделі починають реалізуватися.

Мета етапу ентропії це є зниження набору прапорців/символів їх оптимального простору з урахування ймовірностей. Якщо прапорець має 50% і ви хочете його закодувати використовуючи 1 біт, то якщо 25% - 2 біта.

Якщо в закодованому повідомленні символи зустрічаються в різних пропорціях значення, то ці числа є нормалізованими частотами q_i. N- це сума нормалізованих частот символів.

Для швидкої роботи алгоритму необхідно, щоб сума частот була степенем двійки. Якщо сума частот символів не рівна степені двійки, то частоти потрібно

нормалізовувати – пропорційно зменшувати (чи збільшувати) їх так, щоб в сумі вийшла степінь двійки [18].

На рисунку 2.4 представлено набір з N колонок, де записані символи повідомлення так, що символ зустрічається стільки разів, що рівна його частоті.

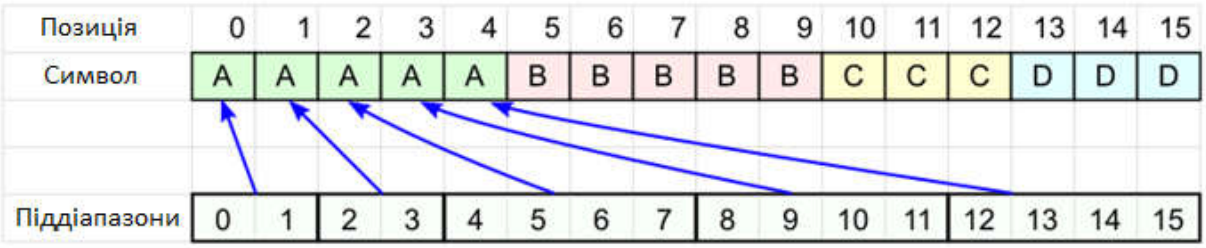

Рисунок 2.4 – Кодова таблиця

Кожному символу відповідає свій діапазон комірок: розмір кожного діапазону має бути степенем двійки, для вираховування швидкодії, в сумі вони мають покривати всі N комірки. Окремо для кожного з символів вибирається свій набір діапазонів.

2.2.1 Алгоритм кодування

Теорія кодування інформації є одним з розділів теоретичної інформатики. До основних задач, які можна вирішити за допомогою даної теорії можна віднести:

– розробка принципів найбільш економного кодування інформації;

– погодження параметрів інформації, яка передається з особливостями каналу зв’язку;

– розробка прийомів, які забезпечують надійність передачі інформації по каналу зв’язку, тобто відсутність втрати інформації.

Дві останні задачі пов’язані з процесами передачі інформації. Кодування інформації необхідне не тільки для передачі, але й для обробки та зберігання даної інформації.

Кодування інформації – процес перетворення сигналу з форми, яка зручна для використання інформації в форму, зручну для передачі, зберігання чи автоматичної переробки.

Процес перетворення повідомлення в комбінацію символів у відповідності з кодом називається кодуванням, процес відновлення повідомлення з комбінації символів називається декодуванням.

Для того, щоб зрозуміти принцип кодування потрібно звернути увагу на кодову таблицю, яка складається з символів та відповідним їм діапазонам.

Для здійснення кодування необхідно вибрати символ, тобто будь-яку комірку таблиці з даним символом, номер даної комірки – це поточний стан.

На рисунку 2.5 показано вибір символу з відповідним станом.

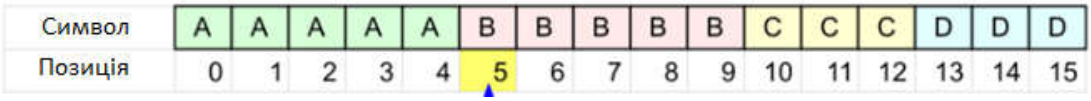

Рисунок 2.5 – Приклад символу з відповідним станом

Після цього необхідно обрати наступний символ та перевірити, в який із інтервалів попадає поточний стан. В даному випадку стан попадає в інтервал номер два (рисунок 2.6).

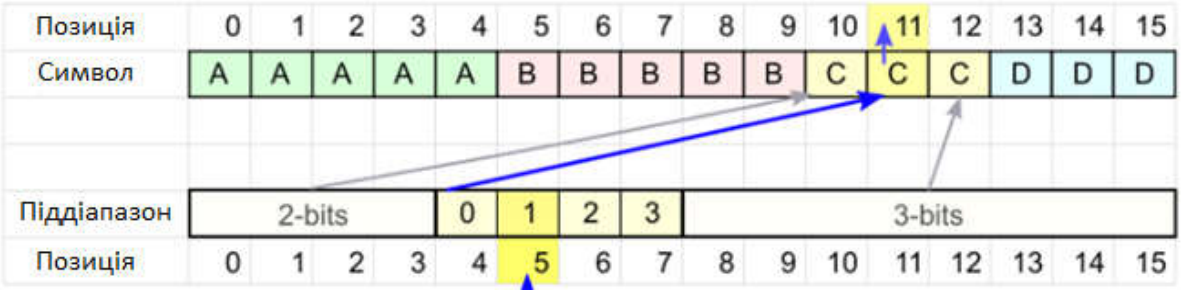

Рисунок 2.6 – Відповідність діапазонів зі станом символу

Запис зміщення відбувається від самого початку даного інтервалу, в даному випадку воно буде рівне одинці. Для запису буде необхідно використати 2 біта згідно розміру діапазону, у який попав наш поточний стан.

Поточний стан відповідає символу розташованому в одинадцятій комірці, тому новий стан буде рівний одинадцяти.

Для здійснення кодування потрібно виконувати дану дію для кожного наступного символу. Потрібно вибрати необхідну таблицю, визначити діапазон, записати зміщення, і отримуємо нове повідомлення.

Для кожного символу записується тільки одне зміщення в інтервалі. Для цього використовується різне число біт, в залежності від того, чи у великий чи малий інтервал ми попадаємо. В середньому кількість біт на символ буде слідувати значенню - $\log_2 \frac{q_i}{N}$. Після того, коли буде закодований останній символ, буде отримано кінцевий стан, який потрібно буде зберегти. За допомогою даного отриманого кінцевого стану можна буде розпочати процес декодування.

2.2.2 Алгоритм декодування

Для здійснення процесу декодування необхідно знати, що файл (зображення, текст чи інше) був стиснутий за допомогою кодів змінної довжини (префіксний код). Для виконання декодування потрібно знати префіксний код кожного символу. Для вирішення цього є три способи:

– велику кількість префіксних кодів вибрано один раз та використовується для процесів кодування та декодування. Даний метод використовується в факсимільному зв’язку. Для навчання та тренування алгоритму вибрано ряд еталонних документів. Навчання алгоритму є найпростішим підходом для стиснення даних, і його якість залежить від того, наскільки реальні файли для стиснення співпадають з вибраними для тренування та навчання зразкам;

– процес кодування відбувається у два етапи. На першому етапі він зчитує стиснутий файл та збирає необхідні статистичні дані. На другому етапі відбувається саме стиснення. В перерві між етапами декодер на основі зібраної інформації створює найкращий префіксний код для даного файлу. Такий метод

надає результати по стисненню, але в нього низька швидкодія для практичного використання. Даний підхід в стисненні називають напівадаптивною компресією;

– адаптивне стиснення застосовується для процесу кодування та декодування. Кодування відбувається не знаючи статичних властивостей об'єкту стиснення. Тому перша частина даних стикається не оптимальним способом, в залежності від стис тиснення та зборі статистики, кодувальник покращує префікційний код, що призводить до покращення компресії. Алгоритм має бути розроблений, таким чином, що декодер міг повторити кожен крок кодувальника, зібрати статистику та покращити префікційний код, тим же методом.

Декодування виконується в зворотному порядку - від останнього закодованого символу до першого. На рисунку 2.7 зображено процес декодування з відповідними символами та позицією відповідною для нього.

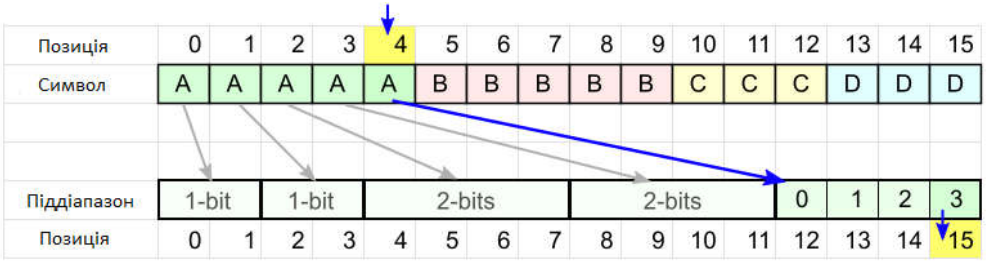

Рисунок 2.7 – Процес декодування

Кінцевий стан символу, який визначається позицією відповідає останньому закодованому символу. Кожен стан відповідає інтервалу (піддіапазон) в кодовій таблиці. Необхідно зчитати необхідну кількість біт згідно розміру інтервалу – 2 біта. В результаті цього отримуємо новий стан, який визначає наступний символ. Процес зчитування повторюється до того часу, коли всі символи будуть декодовані.

Декодування завершується, коли закінчуються закодовані дані, або коли зустрічається спеціальний стоп-символ, який потребує додаткового символу в алфавіті.

2.3 Алгоритм комбінаторного кодування для стиснення даних

Комбінаторне кодування, названо згідно аналогії з арифметичним кодуванням, використовує статистику вхідних даних, та є різновидом статистичних (ентропійних) методів стиснення інформації, до такого методу відносять арифметичне кодування, метод Хафмана та ряд інших менш популярних методів.

Суть комбінаторного методу полягає в тому, що з відомого алфавіту та таблиці частот, можна сформувати певне число різних послідовностей даних. Тому, кожній послідовності даних, відповідає певний (комбінаторний) номер (код) і навпаки, кожному комбінаторному номеру відповідає певна послідовність даних. Для даного методу неможлива адаптивна реалізація, але це в значній мірі окуповується компактністю коду. Метод рекомендований як основний у випадку невеликих вибірок даних, це полягає в тому, що чим більша вибірка, тим довша арифметика, яка використовується при стисненні, чи як допоміжний метод. Для великого об'єму даних можлива блокова реалізація [19].

Вхідну послідовність даних можна представити у вигляді масиву d[N], де N – кількість елементів вибірки стиснення. При скануванні масиву d[N] можна побудувати статистику частоти зустрічі символів у розглянутих даних. Вона може бути представлена двома масивами: a[M] – символи алфавіту, n[M] – частоти зустріч символів алфавіту в розглянутих даних, де M – розмір алфавіту.

Згідно отриманої статистики та вхідних даних можна вирахувати комбінаторний номер, який буде завжди меншим можливого числа комбінацій:

$$C = \frac{N!}{\prod_{i=0}^{M=1} n_i!}. \qquad (2.7)$$

Мінімальне число біт необхідних для зберігання комбінаторного коду визначається виразом $B =]\log_2 C[$. в програмній реалізації. Вирахування логарифмів можна замінити скануванням числа C до першої значущої двійкової одиниці, подібно до команди асемблера bsr [20].

На відміну від арифметичного стиснення та методу Хафмана, для яких подією є наступний символ послідовності даних з інформаційною ентропією

$$H_0 = -\sum_{i=0}^{M=1}\left(\frac{n_i}{N}\log_2\left(\frac{n_i}{N}\right)\right). \quad (2.8)$$

Для комбінаторного стиснення подією являється вся послідовність даних. Оскільки всі можливі послідовності даних рівно вірогідні, то подія для комбінаторного кодування має сумарну умовну ентропію N порядку $H_{SN} = \log_2(C)$, а затрати на зберігання комбінаторного коду можна записати, як $B =]H_{SN}[$.

Відомо, що розмір даних, стиснутих будь-яким ентропійним кодером (без використання методів контекстно-залежного моделювання), який використовується в якості події символ послідовності даних, завжди більший чи рівний сумарній ентропії послідовності $H_{SN} = H_0 * N$ розглянутих даних. Згідно властивостям умовної ентропії, які можна представити у вигляді нерівності $0 \le H_N \le H_{N-1} \dots \le H_1 \le H_0$, чим вищий її порядок, тим вона менша, в гіршому випадку не більша, тому для комбінованого кодування вірне твердження, що розмір коду B завжди менший чи рівний $]H_{S0}[$.

Кількість можливих комбінацій C необхідно вираховувати під час побудови статистики, по мірі надходження даних:

$$C_i = C_{i=1}\frac{i+1}{f(d_i)+1}, \quad (2.9)$$

де $f(d_i)$ – кількість символів еквівалентних d_i, у вже розглянутих даних, і змінюється від 0 до $N-1$, при цьому $C_0 = 1$.

Такий підхід дозволяє легко контролювати розмір комбінаторного коду B, для обмеження довжини арифметики при блочній реалізації.

При кодуванні та декодуванні зручно розглядати кожний символ алфавіту незалежно від решти символів, тому поєднання символів алфавіту для розглянутої статистики буде мати вигляд:

$$C = \prod_{i=0}^{M=1} K_i, \tag{2.10}$$

$$K_i = \frac{(n_i + g_i)!}{n_i g_i!}, \tag{2.11}$$

де $g_i = \sum_{j=0}^{i-1} n_j$, при цьому $K_0 = 1$.

K_i можна розглядати, як число можливих перестановок символу алфавіту a_1 з усіма попередніми символами з урахуванням таблиці частот, таким чином, значення комбінаторного номеру $\bar{C}$ можна показати формулою:

$$\bar{C} = \sum_{i=1}^{M-1} \left(\bar{c}_i \prod_{j=0}^{i-1} K_j \right), \tag{2.12}$$

де $\bar{c}_i$ – номер перестановки a_1 в кількості n_i з послідовністю даних, які складаються з попередніх $a_{j-1}, a_{j-2}, \dots, a_0$ символів алфавіту відповідно в кількості $n_{j-1}, n_{j-2}, \dots, n_0$.

Таким чином, необхідно вирахувати M-1 номерів перестановок $\bar{c}_i$, щоб отримати комбінаторний номер $\bar{C}$ відповідний розглянутій послідовності даних. Алгоритм вирахування комбінаторного номеру (кодування) зображено на рисунку 2.8.

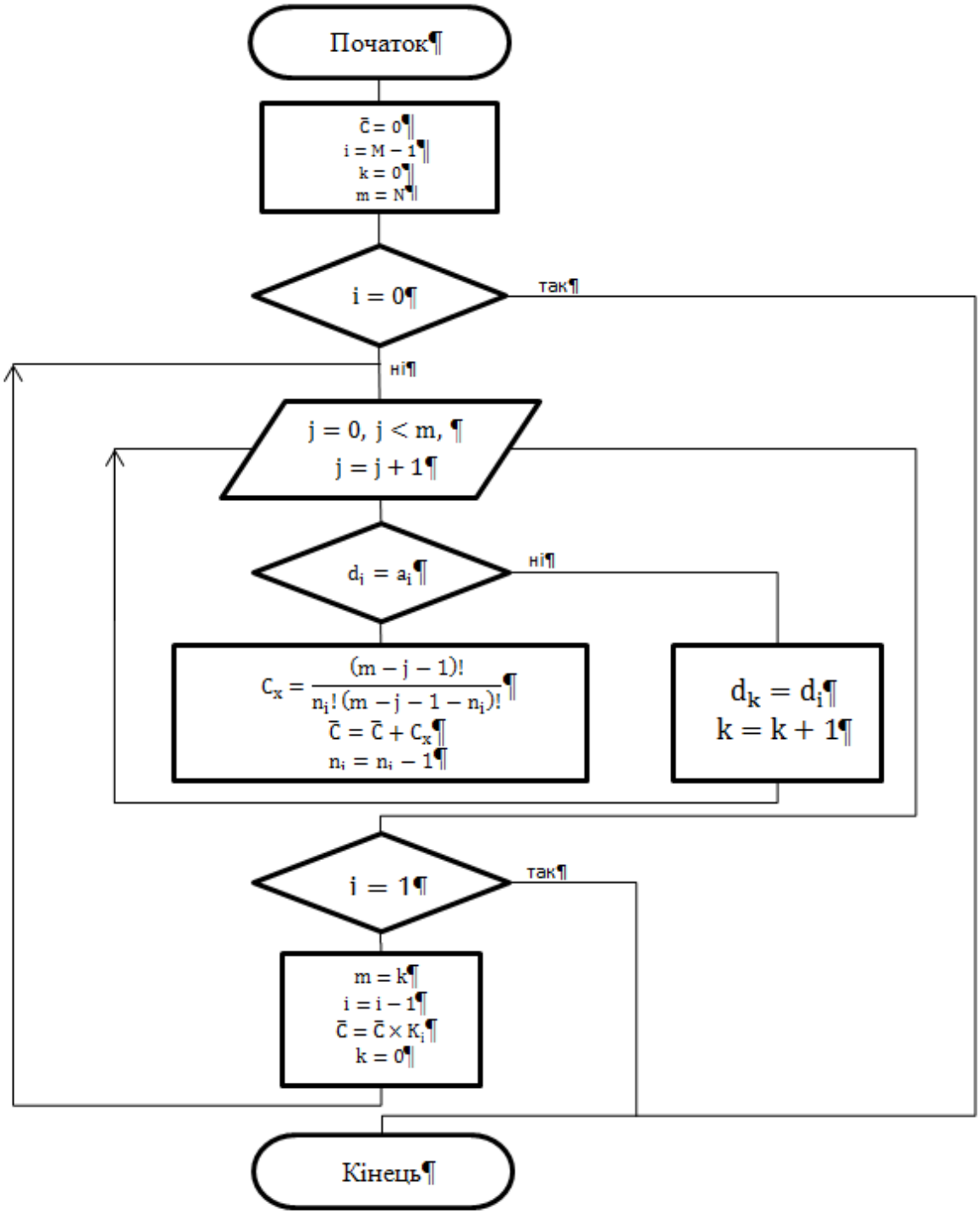

Рисунок 2.8 – Алгоритм комбінаторного кодування даних

Алгоритм декодування представлений на рисунку 2.9, застосовані j, k та m є допоміжними і не мають асоціюватися з однойменними змінними в алгоритмі кодування.

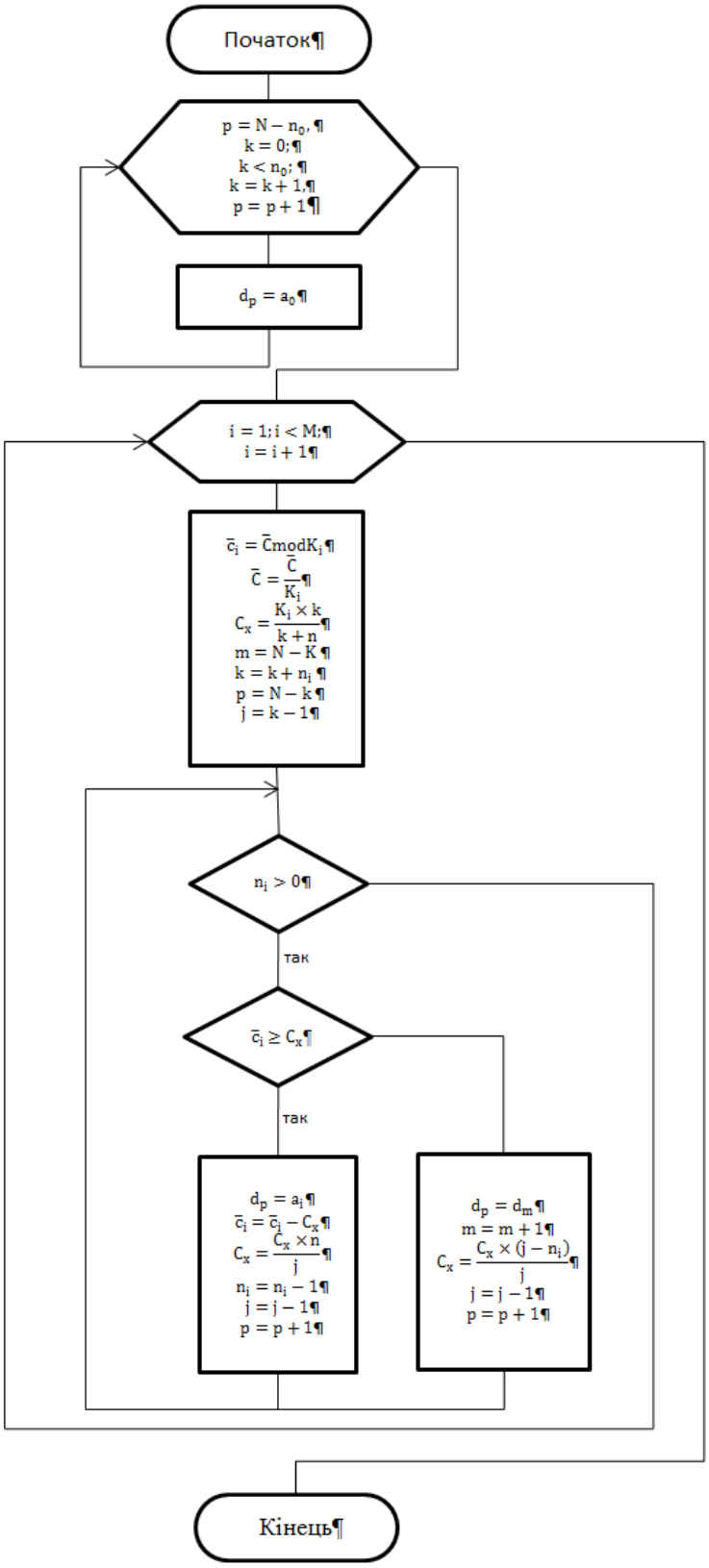

Рисунок 2.9 – Алгоритм комбінаторного декодування даних

Обчислення $\bar{C}$, починається з останнього символу алфавіту a_{M-1} і по мірі розгляду вибірки даних,вона зменшується за рахунок виключення символів

еквівалентних поточному символу алфавіту, що розглядається. Значення C_x визначає кількість комбінацій, які могли би бути, якщо би поточний символ алфавіту не співпадав з значенням у вибірці, а зустрівся би пізніше. Потрібно відмітити. Що факторіал нуля рівний одиниці. Якщо алфавіт складається з одного єдиного символу, комбінаторний номер займає 0 біт [21].

Процес декодування номеру будується зворотно процесу кодування. Для успішного декодування потрібно алфавіт, таблиця частот та комбінаторний номер, при цьому розмір комбінаторного номера легко вирахувати по таблиці частот.

В процесі декодування значення C_x необхідно знати на кожному кроці обчислення. Для зменшення часу декодування, значення C_x вираховується з використанням факторіалів тільки при зміні поточного символу алфавіту, а потім коригується в залежності від результату порівняння з поточним номером перестановки $\bar{c}_i$. Аналогічним чином можна зробити при кодуванні.

В ході дослідження було проведено аналіз ефективності методу Хафмана, комбінаторного та арифметичного кодувань, отримана статистика показана у вигляді графіків на рисунку 2.10 та 2.11.

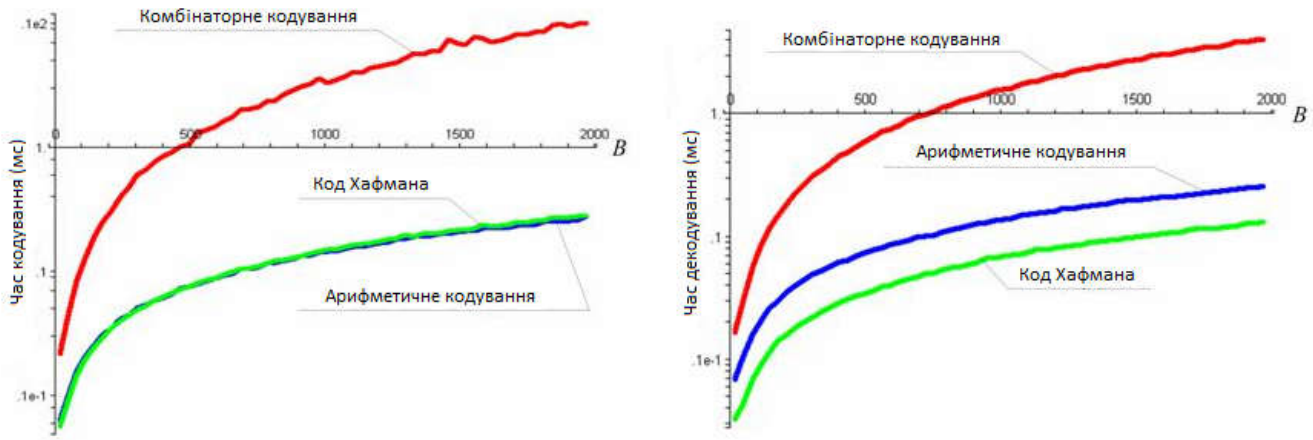

Рисунок 2.10 – Час роботи алгоритмів відносно розміру комбінаторного коду

Час комбінаторного кодування та декодування (рисунок 2.10) швидко росте зі збільшенням самого коду, тому у випадку кодування значних об’ємів даних потрібно використовувати блочну реалізацію алгоритму, що все ж не

дозволить перегнати класичні методи і з точки зору швидкості вони залишаються переважними [22].

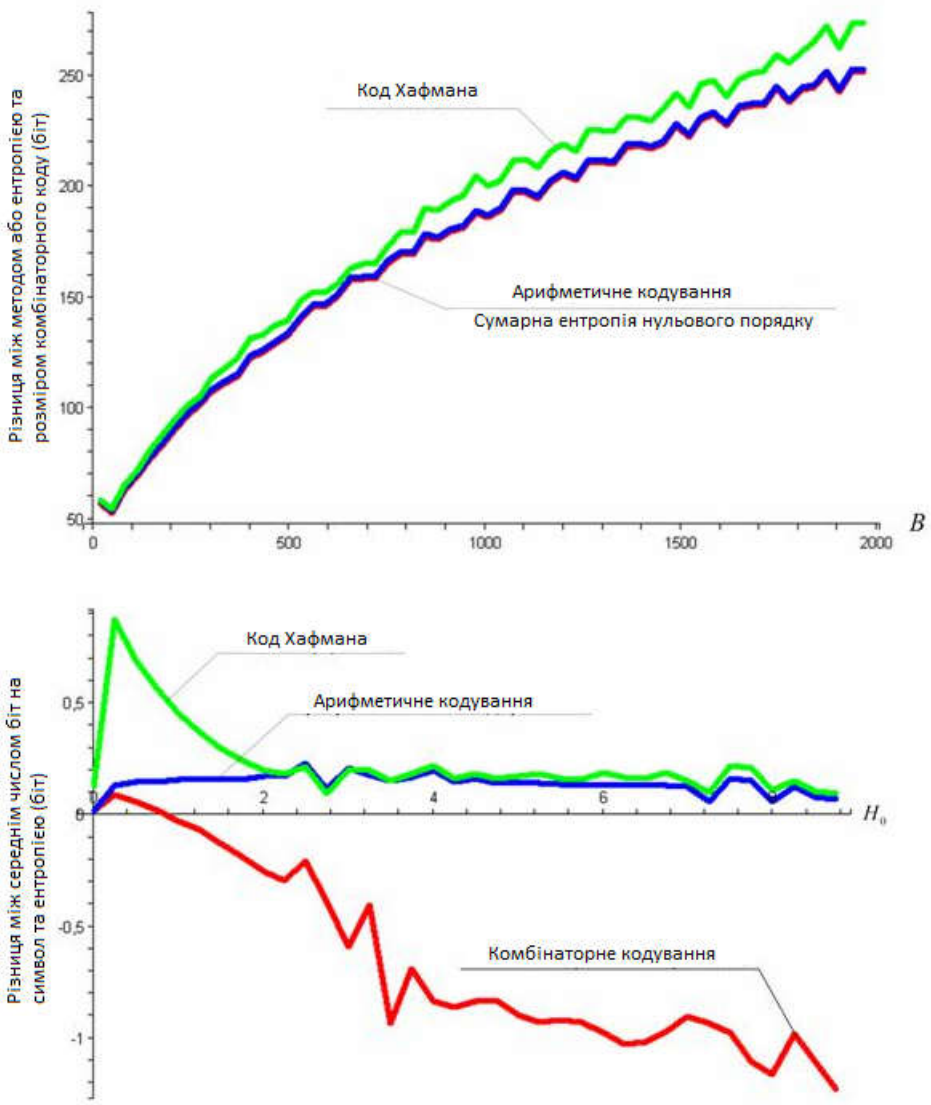

Рисунок 2.11 – Порівняльна ефективність алгоритмів кодування

В розрахунках ефективності кодування враховувався лише розмір відповідного коду без врахувань затрат на таблицю частот та алфавіт, оскільки вони однакові для всіх вказаних методів. Якщо розглядати ефективність кодування відносно комбінаторного коду (перший графік), видно, що на рівних умовах комбінаторний метод дозволяє отримати більш компактний код в порівнянні з іншими алгоритмами, межею для яких є сумарна ентропія H_{S0}. Якщо розглядати адаптивні варіанти арифметичного кодування і метод

Хафмана, то отриманої переваги може буди достатньо, щоб компенсувати неможливість адаптивної реалізації комбінаторного кодування та зберегти перевагу в степені стиснення даних.

Розглядаючи ефективність кодування відносно ентропії H_0 (другий графік), потрібно відмітити, що значний стрибок на початку графіка відповідає методу Хафмана пов'язаний з тим, що даний метод на відміну від арифметичного кодування не може кодувати символи менше чим один біт. Середнє число бітів на символ при використанні комбінаторного кодування лише на початку незначно більше ентропії, це внаслідок того, що розмір комбінаторного коду B являється цілим значенням.

Комбінаторне кодування є найбільш ефективним при стисненні невеликих об'ємів даних, а також випадкових або важко передбачуваних даних. Ефективним середовищем застосування може бути використання в якості додаткового методу. Наприклад, при використанні в парі з алгоритмом, який в силу властивостей подальшого використання інформації стискає її невеликими порціями та формує деякі коефіцієнти, які потребують додаткового статистичного стиснення. Це можуть бути алгоритми стиснення різних потоків, таких як відео, звук чи просто дані, стиснуті блоками для прискорення довільного доступу до архіву [23].

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ДОСЛІДЖЕННЯ АЛГОРИТМІВ СТИСНЕННЯ ДАНИХ

3.1 Оптимізація алгоритмів стиснення даних

Для виконання процесу оптимізації необхідно визначити, як під час кодування розрахувати в який інтервал попадає поточний стан та скільки біт необхідно записати.

Для кодування необхідно розбити таблицю розміром N (кратним степені два) на q_1 інтервалів. Розміри даних інтервалів мають бути кратними степені два: 2^{maxbit} та $2^{(\text{maxbit}-1)}$ – «великі» та «малі». Нехай, малі інтервали розташовані зліва, а великі – справа, зображено на рисунку 3.1.

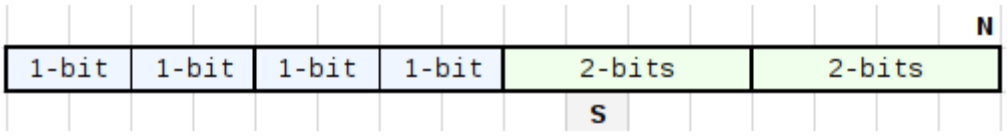

Рисунок 3.1 – Розташування інтервалів кодування

Наприклад, якщо N=16 і потрібно розбити на шість інтервалів, то воно матиме вигляд: 2+2+2+2+4+4 – чотири «малих» інтервали розміром 2^1 та два «великих» розміром 2^2. Потрібно просто вирахувати розмір великого інтервалу – це найменша степінь двійки, така що $2^{\text{maxbit}} \geq \frac{N}{q}$. Відповідно, $\text{maxbit} = \log_2 N - \text{highbit}$, де highbit – номер старшого ненульового біту. Для отримання q інтервалів спочатку потрібно розбити таблицю на «великі» інтервали: $N/2^{\text{maxbit}}$ штук, потім декілька з них розділити навпіл. Розділення інтервалу збільшує їх загальну кількість на один, тому розділити необхідно буде q – $(N/2^{\text{maxbit}})$ великі інтервали. Відповідно кількість малих буде рівна (q – $(N/2^{\text{maxbit}})$)*2.

При цьому необхідно визначити умови стану «позиція», щоб вона попала у «великий» інтервал. Для того. Щоб визначити межу між малими та великими

інтервалами, помножимо кількість малих інтервалів на їх розмір: (q – $(N/2^{maxbit})$)*2*$2^{(maxbit-1)}$=(q*2^{maxbit}) – N.

Стан «позиція» попадає у великий інтервал при умові N+state≥q*2^{maxbit}, якщо ні, то в малий.

Розмір інтервалу визначає скільки молодших бітів від поточного стану потрібно записати при кодуванні. Якщо всі умови виконуються, то nbBitsOut=maxbit, в іншому випадку (maxbit-1). Решта бітів стану будуть визначають номер інтервалу.

Замість операції порівняння можна застосувати прийом з зсувом різниці на досить більше число: nbBitsOut=((maxbit<<16)+N+state – (count<<maxbit))>>16. Відсутність умови в даному виразі дозволяє процесору ефективніше задіяти внутрішній паралелізм.

В наслідок, функція кодування має вигляд:

```
//Розрахунок кількості бітів для запису
nbBitsOut=(( maxbit<<16)+N+state – (count<<maxbit))>>16;
//Запис nbBitsOut молодших бітів від state у bitStream
bitStream.WriteBits(state.nbBitsOut);
interval_number=(N+state)>>nbBitsOut;
//Новий стан
state=nextStateTable[deltaFindsState[symbol]+interval_number];
```

Все, що можна розрахувати раніше, виносимо в таблицю:

```
symbolTT[symbol].deltaNbBits=(maxBitsOut<<16)-(count<<maxBitsOut);
```

Треба відмітити, що state весь час використовується у вигляді N+state. Якщо в nextStateTable зберігати N+next_state, то функція матиме вигляд:

```
nbBitsOut=(N_plus_state+symbolTT[symbol].deltaNbBits)>>16;
bitStream/WriteBits(N_plus_state, nbBitsOut);
N_plus_state=nextStateTable[symbolTT[symbol].deltaFindState+(N_plus_state>>nbBitsOut)];
```

Декодування матиме вигляд:

```
//Запис декодованого символу
```

```
outputSymbol (decodeTable [state] .symbol);
//Кількість бітів взятих з таблиці, які необхідно зчитати
nbBits = decodeNable [state] .nbBits;
//Прочитані біти визначають зміщення в інтервалі
offset = readBits (bitStream, nbBits);
//Вираховуємо наступний стан: початок діапазону + зміщення
state = decodeTable[state] .subrange_pos + offset
```

З вище сказаного відомо, що сума частот символів рівна степені два. В загальному випадку це є не так. Тоді потрібно пропорційно зменшити ці числа, щоб вони в сумі давали степінь двійки.

При цьому треба взяти до уваги, що сума частоти символів рівна розміру кодової таблиці. Чим більша таблиця, тим точніше будуть представлені частоти символів і тим краще буде виконуватися операція стиснення. З іншої сторони, сама таблиця також займає багато місця. Для оцінки розміру стиснення даних можна використовувати формулу Шенона, вважаючи, що стиснення близьке до ідеалу, і таким чином підібрати оптимальну таблицю.

В процесі нормалізації частот може виникнути велика кількість малих проблем. Наприклад,, якщо у деяких символів дуже мала вірогідність, то найменша частота може виявитися дуже неточним приближенням, але менше уже не буде.

Одним з підходів в даному випадку – зразу призначити таким символом одиницю, та помістити їх в кінець кодової таблиці (так оптимальніше), і потім перейти до решти символів. Заокруглення значень до найближчого цілого також може виявитись не кращим варіантом. Отже, існують як швидкі евристичні алгоритми нормалізації частот, так і повільні, але більш точні.

Якщо у різних частин даних різний частотний розподіл, то для їхнього кодування краще використовувати різні таблиці.

При цьому різнотипні дані можна кодувати, почергово використовуючи різні таблиці. Стан із однієї таблиці використовується для кодування

наступного символу іншої таблиці. Основне, щоб повторити той самий порядок при виконанні декодування. Таблиці мають бути однакового розміру.

Якщо кодування даних відбувається від початку до кінця, то при декодування від кінця до початку, тоді буде зрозуміло, що за чим буде йти, і відповідно, яку таблицю необхідно використовувати для декодування наступного елементу.

3.2 Моделювання стиснення даних з використанням апаратних засобів безпровідних сенсорних мереж

Для зібрання даних для процесу стиснення за допомогою алгоритмів стиснення даних було вирішено побудувати схему з використанням датчиків вологості та температури. Після проведення даного дослідження ми отримаємо дані показників для, які будуть записані у файл для подальшого стиснення за допомогою програми написаної на мові програмування Python.

Перш за все для проектування фізичної моделі пристрою для моніторингу температури повітря та вологості необхідно підібрати елементну базу. На сьогоднішний день існує безліч мікроконтролерів і платформ на основі яких створюються пристрої з подібним функціоналом. Перед тим як зробити вибір, потрібно визначити який саме функціонал він повинен виконувати щоб задовільнити наші вимоги. Для фізичної моделі пристрою для моніторингу температури повітря та вологості запропоновано два контролери, а саме:

- мікроконтролерна плата Arduino Uno;
- міні ПК Raspberry Pi.

Arduino Uno – цей пристрій оснований на мікроконтролері ATmega328. У його склад входить все необхідне для зручної роботи з мікроконтролером: 14 цифрових входів / виходів 6 яких можуть використовуватися в якості ШІМ-виходів, 6 аналогових входів, кварцовий резонатор на 16 МГц, роз'єм USB,

роз'єм живлення, роз'єм для внутрисхемного програмування (ICSP) і кнопка скидання. Для початку роботи з пристроєм досить просто подати живлення від AC / DC-адаптера або батарейки, або підключити його до комп'ютера за допомогою USB-кабелю. Застовується для створення електронних пристроїв з можливістю прийому сигналів від різних давачів, які можуть бути підключені до нього і виконувати різний функціонал. Окрім того його низька ціна, просте і зрозуміле середовище програмування з можливістю розширення і відкритим кодом, це все в сукупності дає ряд переваг використання Arduino.

Платформа Arduino представлено на рисунку 3.2, а також наведений перелік ключових елементів, такі як:

– світлодіод №13 - це світлодіод який з'єднаний з цифровим виходом №13;

– цифрові входи/виходи;

– кнопка скидання - здійснює скидання мікроконтролера і повторний запуск програми;

– Мікроконтролер - це аналог мікропроцесора в звичайному ПК;

– аналогові входи;

– виходи живлення;

– живлення +9В - додаткове живлення від зовнішнього джерела;

– USB - призначений для зв'язку з ПК та забезпечує живленням самий пристрій.

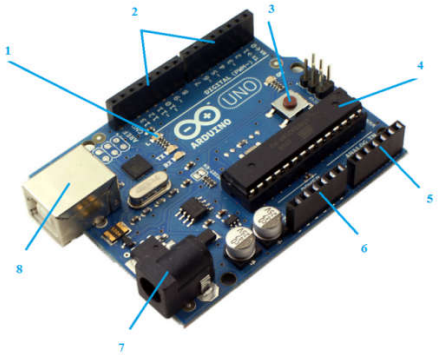

Рисунок 3.2 – Мікроконтролер Arduino

Детальна характеристика Arduino Uno наведена в таблиці 3.1. Для того щоб увімкнути мікроконтролер, достатньо просто подати на нього живлення безпосередньо від персонального комп’ютера через USB кабель, або через спеціальний роз'єм зовнішнього живлення. При підключенні через роз’єм зовнішнього живлення, напруга живлення може коливатись від + 7В до + 12В. Системи створені на основі Arduino можуть бути як самостійними, так і взаємодіяти з програмним забезпеченням, що працює на персональному комп'ютері.

Таблиця 3.1 – Характеристика платформи Arduino Uno

Мікроконтролер	ATmega328
Робоча напруга	5В
Напруга живлення (рекомендований)	7-12В
Напруга живлення (граничне)	6-20В
Цифрові входи / виходи	14
Аналогові входи	6
Максимальний струм одного виведення	40мА
Максимальний вихідний струм виводу 3.3V	50мА
Flash-пам'ять	32 КБ (ATmega328) з яких 0.5 КБ використовуються загрузчиком
SRAM	2 КБ (ATmega328)
EEPROM	1 КБ (ATmega328)
Тактова частота	16МГц

Інший контролер який було вибрано для порівняння це одноплатний комп’ютер Raspberry Pi. Він розроблений британським фондом Raspberry Pi Foundation. Побудований на чіпі (SoC) Broadcom BCM2835, яка включає в себе процесор ARM із тактовою частотою 700 МГц, графічний процесор VideoCore IV, і 512 чи 256 мегабайт оперативної пам'яті. Твердий диск відсутній, натомість використовується SD карта. Така начинка дозволяє виконувати безліч завдань, наприклад, працювати з текстом, використовувати доступ до Інтернету або грати в комп'ютерні ігри. Так само може показувати відео високої роздільної здатності а саме відео формату H.264 в роздільній здатності 1080p.

Цей міні ПК може працювати під управлінням операційних систем Debian, Fedora, Gentoo, ОС Raspbian яка заснована на Debian і оптимізована під Raspberry Pi. Raspberry Pi зображений на рисунку 3.3.

Рисунок 3.3 – Міні ПК Raspberry Pi

Детальна характеристика Raspberry Pi наведена в таблиці 3.2.

Таблиця 3.2 – Характеристика платформи Raspberry Pi

Процесор	Broadcom BCM2835 на 700 МГц
Графічний процесор	Broadcom VideoCore IV, OpenGL ES 2.0, 1080p30 h.264/MPEG-4 AVC high-profile decoder
Оперативна память	512 Мб SDRAM
Відео виходи	Composite RCA, HDMI
Аудіо виходи	3.5 мм джек, HDMI
Роз’єми	SD, MMC, SDIO, 10/100 Ethernet RJ45, 4 x USB, Живлення micro-USB, 40пін інтерфейс GPIO

Проаналізувавши інформацію по Arduino та Raspberry Pi можна зробити деякі висновки. Ці дві платформи сприймають як аналоги і як конкуруючі апаратні платформи. Розміри цих пристроїв приблизно одинакові, це мабуть єдиний фактор, що робить їх схожими. Серед відмінностей на які потрібно звернути увагу це:

Raspberry Pi це повнофункціональний комп'ютер на якому може бути запущена операційна система Linux. До USB-портів можна підключати різні пристрої, наприклад для бездротового підключення до мережі Інтернет. Загалом ця невеличка плата є досить потужною для того, що функціонувати в якості повноцінного комп'ютера, на відміну від Arduino якого не можливо

назвати комп'ютером . По тактовій частоті Raspberry Pi в 40 разів швидший за Arduino. Ще більш вражаючим фактом є оперативна пам'ять: Raspberry Pi має в 128000 разів більше оперативної пам'яті ніж Arduino.

Великим плюсом Arduino є те, що він може краще ніж Raspberry Pi здійснювати зчитування аналогових сигналів в реальному часі. Ця гнучкість дозволяє Arduino працювати практично з будь-яким видом датчиків або чіпів. Raspberry Pi не така гнучка, для читання аналогових датчиків потрібні додаткові апаратні засоби.

Порівнявши ці дві плати можна зробити висновки, що не зважаючи на великі переваги Raspberry Pi над Arduino, саме мікроконтролер Arduino доцільніше застосувати для проектування фізичної моделі моніторингу температури повітря у будинку, оскільки необхідно тільки підключити необхідний давач та почати роботу.

Для фізичної моделі моніторингу температури повітря у будинку потрібно вибрати один з давачів, який буде підключатися до Arduino, а саме:

– DHT11;

– DHT22.

Давач DHT11 дозволяє визначати температуру повітря і відносну вологість в будинку чи в любиш приміщеннях, де температура не опускається нижче нуля, давач зображено на рисунку 3.4.

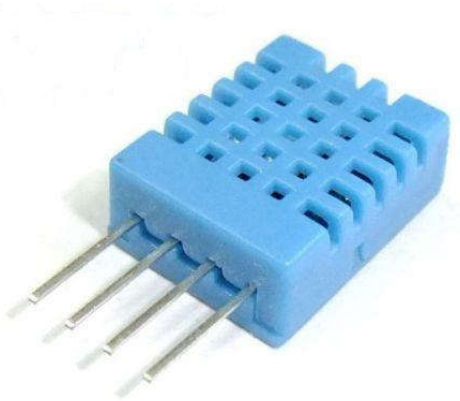

Рисунок 3.4 – Давач DHT11

Дані про температури і вологості поставляються по сигнальному проводу у вигляді цифрового сигналу. І температура і вологість відправляються по одному сигнальному проводу (S). DHT11 працює з приймаючою стороною, такою як Arduino за власним протоколом. Комунікація в загальному виглядає так:

– мікроконтролер говорить про те, що хоче зчитати показники. Для цього він встановлює сигнальну лінію в 0 на деякий час, а потім встановлює її в 1;

– сенсор підтверджує готовність віддати дані. Для цього він аналогічно спочатку встановлює сигнальну лінію в 0, а потім в 1;

– після цього сенсор передає послідовність 0 і 1, послідовно формують 5 байт (40 біт). У перших двох байтах передається температура, в третьому і четвертому – вологість, в п'ятому – контрольна сума, щоб мікроконтролер зміг переконатися у відсутності помилок зчитування.

Завдяки тому, що сенсор робить вимірювання тільки за запитом, досягається енергоефективність: поки запиті на зчитування немає, датчик споживає дуже невеликий струм. Більше деталей по давачу DHT11 наведено в таблиці 3.3.

Таблиця 3.3 – Характеристики давача DHT11

Напруга живлення	3-5 В
Визначення вологості	20-80% з 5% точністю
Визначення температури	0-50 град. з 2% точністю
Частота запиту	Не більше 1Гц (не більше разу в 1 сек.)
Розміри	15.5мм × 12мм × 5.5мм

Давач DHT22 складається з датчика температури та датчика вологості. Датчик температури цифровий, побудований на основі чіпа DS18B20. Датчик вологості ємнісного типу відносної вологості (RH), чутливим елементом якого є полімерний конденсатор. Давач DHT22 дозволяє визначати температуру повітря і відносну вологість в будинку чи в любих приміщеннях, і на відміну

від DHT11 навіть у приміщеннях з температурою нижче нуля, давач зображено на рисунку 3.5.

Дані про температури і вологості поставляються по сигнальному проводу у вигляді цифрового сигналу. І температура і вологість відправляються по одному сигнальному проводу (S). DHT11 працює з приймаючою стороною, такою як Arduino за власним протоколу.

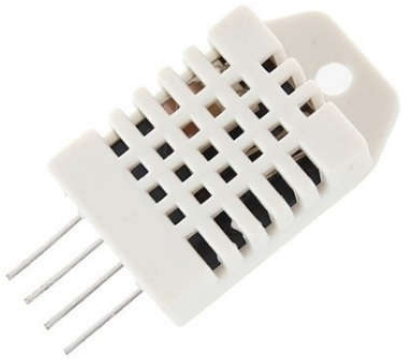

Рисунок 3.5 – Давач DHT22

Комунікація в загальному виглядає так:

– мікроконтролер говорить про те, що хоче зчитати показники. Для цього він встановлює сигнальну лінію в 0 на деякий час, а потім встановлює її в 1;

– сенсор підтверджує готовність віддати дані. Для цього він аналогічно спочатку встановлює сигнальну лінію в 0, а потім в 1;

– після цього сенсор передає послідовність 0 і 1, послідовно формують 5 байт (40 біт). У перших двох байтах передається температура, в третьому-четвертому - вологість, в п'ятому - контрольна сума, щоб мікроконтролер зміг переконатися у відсутності помилок зчитування.

Завдяки тому, що сенсор робить вимірювання тільки за запитом, досягається енергоефективність: поки запиті на зчитування немає, датчик споживає дуже невеликий струм. Більше деталей по давачу DHT22 наведено в таблиці 3.4.

Таблиця 3.4 – Характеристики давача DHT22

Напруга живлення	3.3-6 В
Визначення вологості	0-100% з 2-5% точністю
Визначення температури	-40-125 град. з точністю ±0.5 град.
Частота запиту	Не більше 0.5Гц(не більше разу в 2сек)
Розміри	15.1мм × 25мм × 7.7мм

Для створення системи моніторингу температури повітря у будинку було прийнято рішенням про використання мікроконтролера Arduino Uno. Raspberry Pi це повнофункціональний комп'ютер на якому може бути запущена операційна система Linux. До USB-портів можна підключати різні пристрої, наприклад для бездротового підключення до мережі Інтернет. Загалом ця невеличка плата є досить потужною для того, що функціонувати в якості повноцінного комп'ютера, на відміну від Arduino якого не можливо назвати комп'ютером . По тактовій частоті Raspberry Pi в 40 разів швидший за Arduino. Ще більш вражаючим фактом є оперативна пам'ять: Raspberry Pi має в 128000 разів більше оперативної пам'яті ніж Arduino. Великим плюсом Arduino є те, що він може краще ніж Raspberry Pi здійснювати зчитування аналогових сигналів в реальному часі. Ця гнучкість дозволяє Arduino працювати практично з будь-яким видом датчиків або чіпів. Raspberry Pi не така гнучка, для читання аналогових датчиків потрібні додаткові апаратні засоби.

Порівнявши ці дві плати можна зробити висновки, що не зважаючи на великі переваги Raspberry Pi над Arduino, цей вражаючий функціонал Raspberry Pi не потрібен для нашої системи, а тому буде недоцільно застосувати Raspberry Pi для проектування фізичної моделі моніторингу температури повітря у будинку. Також Arduino набагато простіший у використанні оскільки необхідно тільки підключити необхідний давач та почати роботу.

Давачем було вибрано датчик DHT22 оскільки він на відміну від DHT11 може працювати у приміщеннях з температурою нижче нуля. Також DHT22 з більшою точністю визначає температуру повітря і відносну вологість.

3.2.1 Фізична модель пристрою для моніторингу температури та вологості

Фізична модель пристрою для моніторингу температури повітря та вологості було змодельовано в програмі «Fritzing». Fritzing - унікальне програмне забезпечення від University of Applied Sciences Potsdam для перекладу прототипу в фізичну модель. Додаток являє собою утиліту з відкритим вихідним кодом, що полегшує процес роботи фахівців різного рівня. Програма дозволяє створювати моделі друкованої плати і перетворювати їх в реальні.

Зручні інструменти покликані максимально полегшити перетворення ідеї в цифровий проект. Створювані схеми виходять надійними і не потребують доопрацювання для запуску виробництва плат. Інтерфейс програми вимагає хороших навичок роботи з ПК. Основне вікно Fritzing - це робочий стіл з можливістю проектування плати. Щоб створити макет, досить вибрати бажане розташування потрібних мікросхем і способи їх з'єднання з платою. Робоче середовище «Fritzing» зображено на рисунку 3.6.

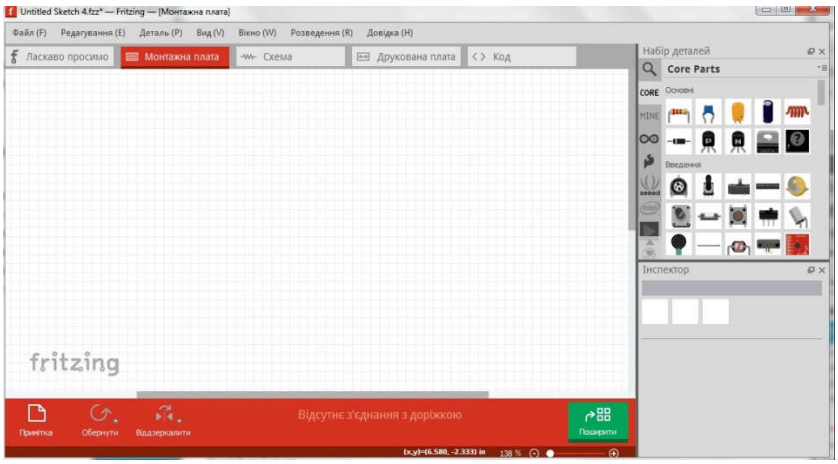

Рисунок 3.6 – Робоче середовище «Fritzing»

Фізичну модель пристрою для моніторингу температури повітря у будинку у середовищі «Fritzing» зображено на рисунку 3.7, а схему моделі наведено на рисунку 3.8.

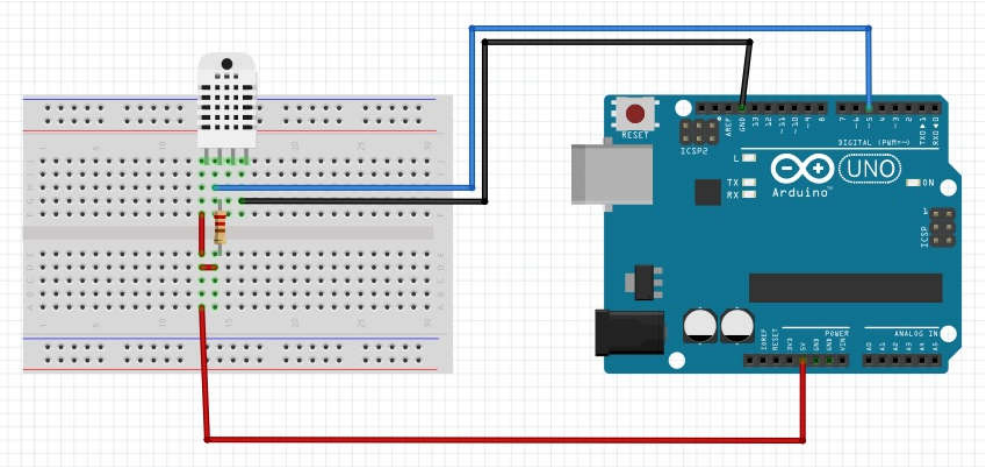

Рисунок 3.7 – Фізична модель пристрою для моніторингу температури повітря в середовищі «Fritzing»

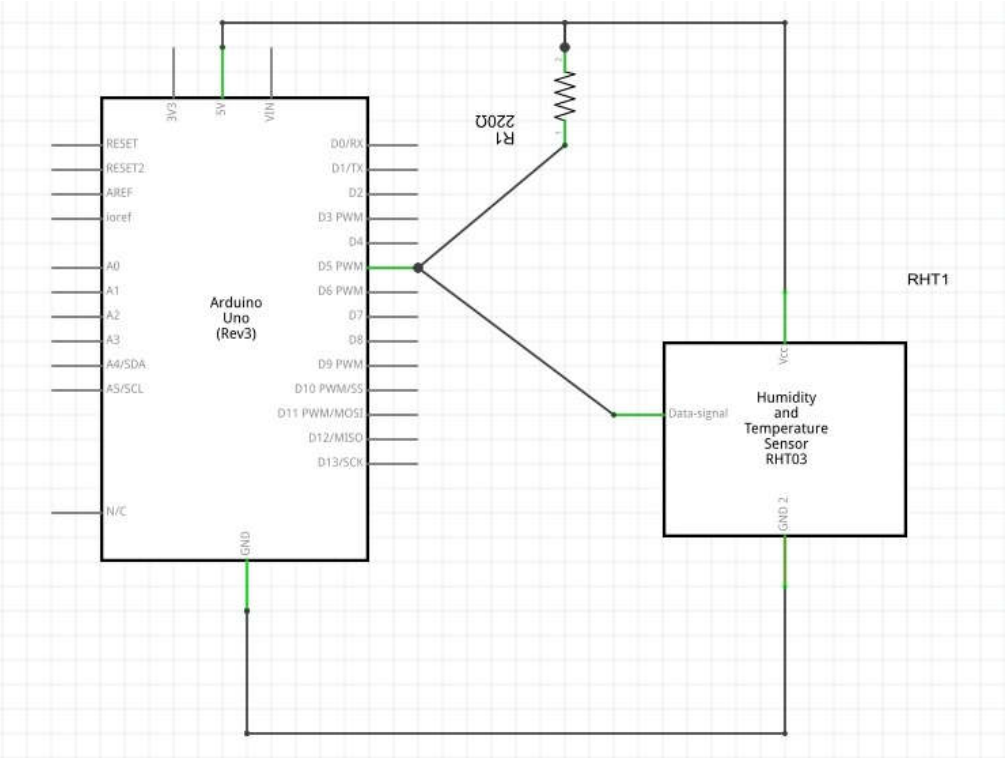

Рисунок 3.8 – Схема фізичної моделі пристрою для моніторингу температури повітря в середовищі «Fritzing»

3.2.2 Програмно-апаратна реалізація фізичної моделі пристрою для моніторингу температури повітря та вологості

Програмне забезпечення Arduino IDE розроблено для взаємодії користувача з спроектованим стендом, інтерфейс допомагає швидше опанувати

систему користувачу. Arduino IDE дозволяє комп’ютеру взаємодіяти з Arduino для передачі даних і для прошивки коду в контролер.

Основними елементами в Arduino IDE є:

– панель інструментів - кнопки на панелі інструментів призначені для створення, відкриття, збереження і прошивки програм в пристрій;

– текстовий редактор який має стандартні інструменти копіювання, вставки, пошуку і заміни тексту і в якому пишуть так званні скетчі (іноді програми написані в Arduino IDE називають скетчами, вони написані в текстовому редакторі і збережені з розширенням «.ino»);

– текстова консоль – відображає потік вихідних даних середовища Arduino у вигляді тексту;

– область для виведення повідомлень - це зворотним зв'язком для користувача який інформує його про події що виникають в процесі запису або експорту коду.

Вікно редактора зображено на рисунку 3.9.

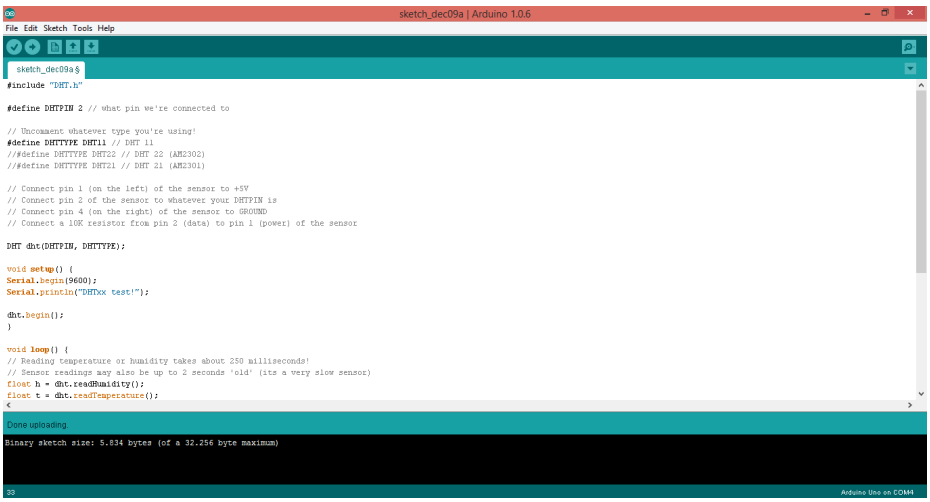

Рисунок 3.9 – Вікно редактора Arduino IDE

В результаті компіляції програми на рисунку 3.11 та 3.11 представлено дані показники температури та вологості при різних умовах.

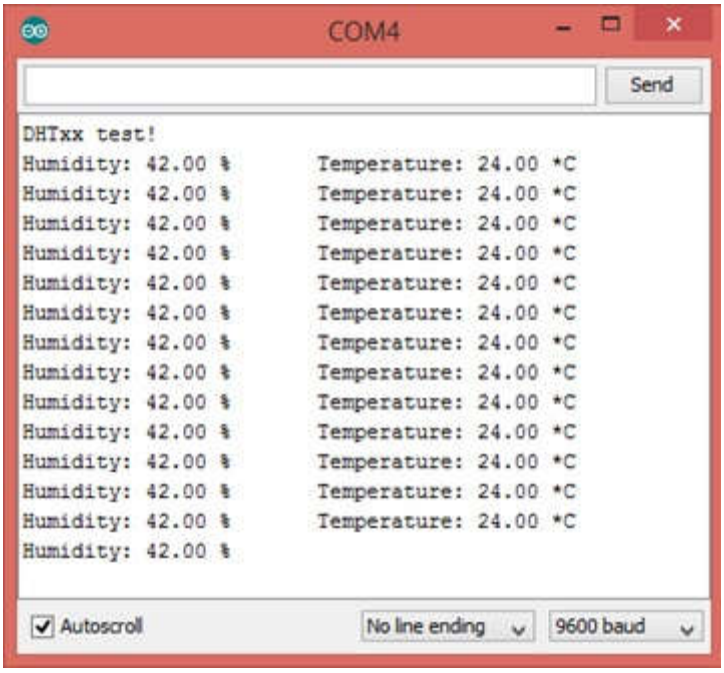

Рисунок 3.10 – Вигляд вікна показників складеної схеми

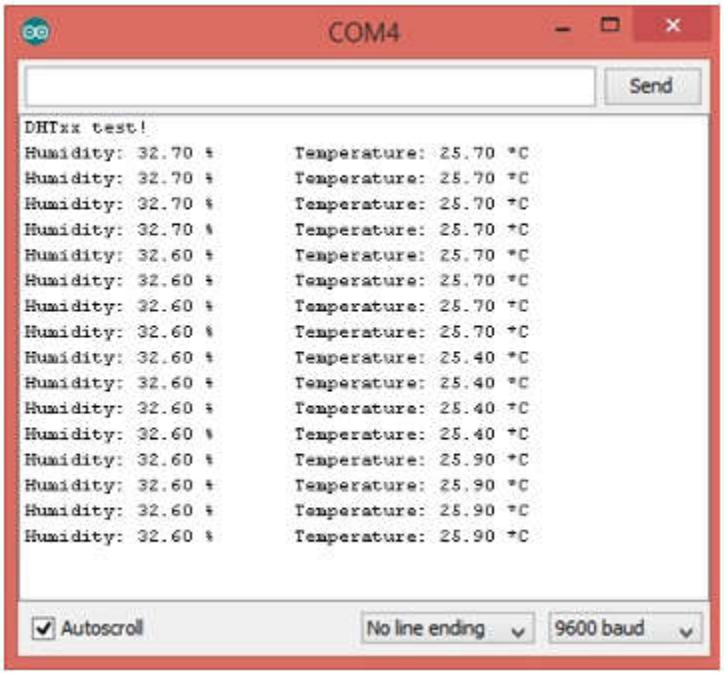

Рисунок 3.11 – Вигляд вікна показників складеної схеми

В результаті проведення даного дослідження ми отримали дані вологості та температури, які записані у файл *txt для подальшого стиснення даних за допомогою відомих алгоритмів.

3.3 Програмна реалізація та тестування алгоритмів стиснення даних

Python – високорівнева мова програмування загального значення, орієнтована на збільшення продуктивності розробника та читабельності коду. Дана мова програмування є обєктно – орієнтованою розширеною мовою програмування дуже високого рівня. Python універсальна мова, яка широко використовується у всьому світі для самих різних цілей: бази даних та обробка тексту, вбудова інтерпретатора в ігри, програмування GUI та швидке створення прототипів, для програмування Internet та WEB додатків. Python володіє багатою стандартною бібліотекою, великим набором модулів. Python та додатки написані на мові, використовують популярні та великі фірми – IBM, GOOGLE, RED HAT та MICROSOFT [24, 25].

Python - це проста мова, але і в той час, потужна обєктно – орієнтована мова програмування. Дана мова програмування представляє структури даних високого рівня, має зручний синтаксис та використовує динамічний контроль типів, що робить її ідеальною мовою для швидкого написання різних додатків, які працюють на більшості поширених платформах [26].

Для реалізації програми було використано такі алгоритми стиснення даних, а саме: алгоритм Хафмана, алгоритм RLE та алгоритм Зіва – Лемпела. В додатку Г наведено реалізацію алгоритму RLE для стиснення даних, а в додатку Д представлено реалізацію алгоритму Зіва – Лемпела.

Кодування Хафмана – це метод стиснення даних без втрат, він призначає коди змінної довжини для вхідних символів, виходячи з частоти їх появи. Найбільш частому символу дано найменший код довжини.

Для реалізації програми стиснення даних були визначені основні пункти:

– створення часописного словника;

– виділення двох мінімальних символів та об’єднання їх кілька разів, використовується Min Heap;

– створення дерева процесу, для цього необхідно створити клас HeapNode та використовувати об'єкти для збереження структури дерева;

– призначення коду для символів, рекурсивно перетинає дерево та призначає відповідний код;

– кодування введеного тексту;

– запис результату у вигляді двійкового коду, що і є стиснутим файлом.

Після запуску декількох тестових зразків було встановлено, що значення розміру стисненого файлу зменшується у два рази. Реалізовано функцію компресії та декомпресії.

Метод компресії використовується для стиснення даних, в результаті виконання якого отримуємо стиснутий файл.

```
def compress(self):
        filename, file_extension = os.path.splitext(self.path)
        output_path = "compressed.hc"
        with open(self.path, 'r+') as file:
                text = file.read()
                text = text.rstrip()
                frequency = self.make_frequency_dict(text)
                self.make_heap(frequency)
                self.merge_nodes()
                self.make_codes()
                encoded_text = self.get_encoded_text(text)
                padded_encoded_text = self.pad_encoded_text(encoded_text)
                b = self.get_byte_array(padded_encoded_text)
                with open('compressed.hc', 'wb') as f:
                        pickle.dump(b, f, pickle.HIGHEST_PROTOCOL)
        print("Compressed")
        return output_path
```

При виконанні даного методу застосовуються такі дії:

- виконання частотного словника;
- виділення основних символів;
- з’єднання вузлів та створення дерева;
- кодування тексту та виведення масиву байтів у бінарний файл.

В результаті виконання методу декомпресії стиснутий файл повертається в початковий стан, без втрати даних.

```
def decompress(self, input_path):
        filename, file_extension = os.path.splitext(self.path)
        output_path = filename + "_decompressed" + ".txt"
        with open(input_path, 'rb') as file, open(output_path, 'w') as output:
            bit_string = ""
            byte = file.read(1)
            while(byte != ""):
                byte = ord(byte)
                bits = bin(byte)[2:].rjust(8, '0')
                bit_string += bits
                byte = file.read(1)
            encoded_text = self.remove_padding(bit_string)
            decompressed_text = self.decode_text(encoded_text)
            output.write(decompressed_text)
        print("Decompressed")
        return output_path
```

Після виконання методу декомпресії виконуються такі дії:

- считування вмісту бінарного файлу;
- видалення відступів та розшифрування тексту;
- виведення декодованого тексту у файл з розширенням txt.

В додатку В наведено реалізацію алгоритму Хафмана для стиснення даних.

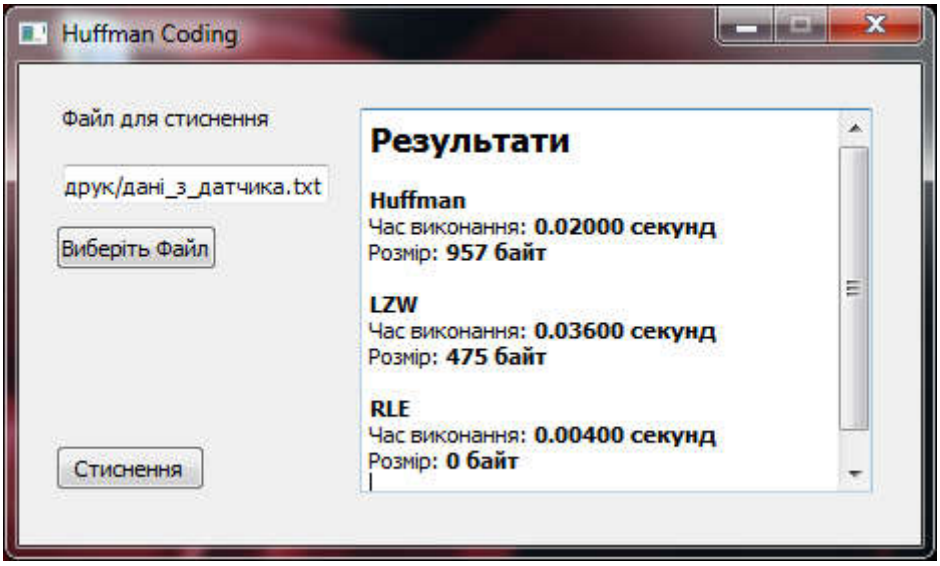

Рисунок 3.12 – Вікно програми

На рисунку 3.12 представлено результати виконання програми стиснення даних на основі алгоритмів Хафмана, Зева – Лемпела та RLE . Отже, можна зробити висновок, що алгоритм Хафмана показав найкращий час виконання, а саме 0,02000 секунд, алгоритм Зіва – Лемпела показав найкраще стиснення, при розмірі файлу 2кб, отримано файл розміром 475 байт.

ВИСНОВКИ

В результаті виконання дипломної роботи розроблено асиметричний за складністю алгоритм стиснення даних на основі алгоритмів Хафмана, Зіва – Лемпела та RLE, а також здійснено програмну реалізацію на мові програмування Python. При цьому отримано такі результати:

1) проведено аналіз методів та алгоритмів стиснення даних, та здійснено порівняльний аналіз обчислюваної складності алгоримів стиснення даних;

2) досліджено вимоги до алгоритмів стиснення даних в безпровідних сенсорних мережах;

3) розроблено асиметричний за складністю алгоритм стиснення даних;

4) здійснено програмну реалізацію алгоритму стиснення даних на мові програмування Python.