VISVESVARAYA TECHNOLOGICAL UNIVERSITY BELAGAVI-590014



**2024-2025**

**A Case Study On “WIRELESS SENSORS NETWORKS**

**USING HAFFMAN CODE”**

### Submitted in partial fulfillment of the requirement in 4th Semester Analysis & Design of Algorithms with Mini Project work (BCS401)

**Department of Computer Science & Engineering**

**By**

RAKSHITHA S P [1AT23CS128] RISHA HERMAIN [1AT23CS133] SUHA ZAHIR [1AT23CS159] VARSHA D [1AT23CS174]

**Under the Guidance Of Prof Y. Mary Angel**

**Assistant Professor, Dept. of CSE, AIT**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERINGATRIA INSTITUTE OF TECHNOLOGY**

**BANGALORE- 560024**

**2023-2024**

# ATRIA INSTITUTE OF TECHNOLOGY DEPARTMENT OF COMPUTE SCIENCE & ENGINEERING BANGALORE-560024



**CERTIFICATE**

Certified that the project work entitled “WIRELESS SENSORS NETWORKS ”, carried out by RAKSHITHA SP(1AT23CS128), VARSHA D(1AT23CS174), RISHA HERMAIN(1AT23CS133) and SUHA ZAHIR(1AT23CS159) are bonafide

students of Atria Institute of Technology, Bangalore, in partial fulfilment for the award of Bachelor of Engineering in Computer Science & Engineering of Visvesvaraya Technological University, Belgaum during the academic year 2024- 2025. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the Report deposited in the departmental library.

The project report has been approved as it satisfies requirement in respect of project work prescribed for the said degree.

**Signature of Principal**

# DECLARATION

We , RAKSHITHA SP(1AT23CS128), VARSHA D(1AT23CS174), RISHA

HERMAIN(1AT23CS133) and SUHA ZAHIR(1AT23CS159) students of fourth IV semester B.E in Computer Science & Engineering at Atria Institute of Technology, hereby declare that the project work entitled “WIRELLESS SENSORS NETWORKS” has been carried out under the supervision of Prof. Y Mary Angel, Assistant Professor, Dept. of CS&E, Atria Institute of Technology and submitted in partial fulfilment of the course requirements for the award of degree in B.E in Computer Science & Engineering of Visvesvaraya Technological University, Belagavi during the year 2023-2024. We further declare that the report has not been submitted to any other University for the award of any other degree.

**Place: Bangalore Signature of the students Date: 9 M A Y 2 0 2 5**

**NAME:** RAKSHITHA SP

**USN: 1AT23CS128**

**NAME: VARSHA D USN: 1AT23CS174**

**NAME: RISHA HERMAIN USN: 1AT23CS133**

**NAME: SUHA ZAHIR USN: 1AT23CS159**

# ABSTRACT

Huffman coding is a widely-used algorithm in computer science and information theory for lossless data compression. Developed by David A. Huffman in 1952, the algorithm constructs an optimal prefix-free binary code based on the frequency of symbols in a given dataset. The process involves creating a binary tree, known as a Huffman tree, where each leaf node represents a symbol, and the path from the root to the leaf determines the symbol's binary code. More frequent symbols are assigned shorter codes, while less frequent ones receive longer codes, thereby minimizing the overall weighted path length and achieving efficient compression.

Huffman coding is optimal for symbol-by-symbol encoding with known input probabilities, ensuring the minimum expected codeword length. However, it may not always be the most efficient method when symbols are not independent or when the probability distribution is unknown. In such cases, alternative methods like arithmetic coding may offer better compression ratios. Despite these limitations, Huffman coding remains a fundamental technique in data compression, widely implemented in formats such as ZIP files, JPEG images, and MP3 audio files.

# ACKNOWLEDGEMENT

We express gratitude to our institution and management for providing us with good infrastructure, laboratory, facilities and inspiring staff, and whose gratitude was of immense help in completion of this project successfully.

We express our sincere gratitude to **Dr. Rajesha S**, Principal, Atria Institute of Technology, for providing us the required environment and for his valuable suggestion.

Our sincere thanks to **Dr. Devi Kannan**, Head of the Dept. Computer Science and Engineering, Atria Institute of Technology, for her valuable support and for rendering us resources for this project work.

We express our gratitude to **Prof. Y MARY ANGEL**, Dept. of Computer Science and Engineering, Atria Institute of Technology, who guided us with valuable suggestions in completing this project at every stage.

Last but not the least, the project would not have been a success without the support of our **parents** and **friends**. Our sincere thanks should be rendered to everyone who helped us in all possible ways.

**NAME: RAKSHITHA SP USN: 1AT23CS128**

**NAME: VARSHA D USN: 1AT23CS174**

**NAME: RISHAHERMAIN USN: 1AT23CS133**

**NAME:SUHA ZAHIR USN: 1AT23CS159**

TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| CHAPTER NO. | TOPICS | PAGE NO. |
| CHAPTER 1 | INTRODUCTION | 7 |
| CHAPTER 2 | PROBLEM STATEMENT | 8 |
| CHAPTER 3 | OBJECTIVE | 9 |
| CHAPTER 4 | REQUIREMENT ANALYSIS | 10-11 |
| CHAPTER 5 | DESIGN | 12-19 |
| CHAPTER 6 | IMPLEMENTATION | 20-22 |
| CHAPTER 7 | RESULT | 23-24 |
| CHAPTER 8 | CONCLUSION | 25 |

# CHAPTER – 1 INTRODUCTION

Huffman coding is a widely-used algorithm in computer science and information theory for lossless data compression. Developed by David A. Huffman in 1952 as part of his Ph.D. research at MIT, the algorithm constructs an optimal prefix-free binary code based on the frequency of symbols in a given dataset. The primary goal is to reduce the overall size of data by assigning shorter codes to more frequent symbols and longer codes to less frequent ones, thereby minimizing the average code length. Huffman coding is widely implemented in various data compression formats and protocols, including ZIP files, JPEG images, MP3 audio files, and PNG images. Its efficiency and simplicity make it a fundamental technique in the field of data compression.

# CHAPTER 2 PROBLEM STATEMENT

In the realm of data storage and transmission, optimizing the representation of information is crucial to enhance efficiency and reduce costs. Uncompressed data, especially in formats like text, images, and audio, can occupy significant storage space and require substantial bandwidth for transmission.

# PROBLEM OVERVIEW :

In today's digital era, the exponential growth of data across various domains—such as text, images, and audio—has led to significant challenges in storage and transmission. Uncompressed data, especially in formats like text, images, and audio, can occupy substantial storage space and require considerable bandwidth for transmission. This inefficiency not only strains storage resources but also increases operational costs and transmission times.

For instance, high-resolution images and high-quality audio files can be several megabytes in size, making them cumbersome to store and slow to transmit over networks. This becomes particularly problematic in environments with limited bandwidth or storage capacity, such as mobile networks or embedded systems.

# CHAPTER 3 OBJECTIVE

Wireless Sensor Networks (WSNs) are deployed to monitor and collect data from various environments. When integrating Huffman coding for data compression, the primary objectives include:

## Energy Efficiency

Huffman coding reduces the amount of data transmitted, leading to lower energy consumption in sensor nodes. This is crucial for prolonging the operational lifetime of battery-powered devices.

## Enhanced Data Transmission

By compressing data before transmission, Huffman coding minimizes bandwidth usage, leading to faster and more reliable data delivery across the network.

## Storage Optimization

Compressed data occupies less storage space, allowing for more efficient use of memory resources in sensor nodes and facilitating the handling of larger datasets.

## Scalability

Huffman coding can be adapted to various data types and network sizes, making it suitable for large-scale deployments where data volume is substantial .

# CHAPTER 4

REQUIREMENT ANALYSIS

## Functional Requirements

These define the core functionalities the WSN must support:

* + Data Sensing and Collection: Specify the types of data to be collected (e.g., temperature, humidity, motion) and the frequency of data sampling.
  + Data Transmission: Outline the protocols and methods for data communication between sensor nodes and the base station.
  + Data Processing: Determine if data aggregation or preprocessing is required at the node level before transmission.
  + Event Detection and Response: Define how the network should respond to specific events or thresholds being met.

## Non-Functional Requirements

These address the quality attributes of the WSN:

* + Energy Efficiency: Given the limited battery life of sensor nodes, strategies for energy conservation are essential.
  + Scalability: The network should support the addition of new nodes without significant performance degradation.
  + Reliability and Fault Tolerance: Mechanisms to ensure continuous operation despite node failures or communication issues.
  + Latency and Throughput: Requirements for data transmission delays and the volume of data handled.
  + Security: Measures to protect data integrity and prevent unauthorized access.

## Environmental and Deployment Considerations

Understanding the deployment environment is critical:

* + Topology: Decide on the network topology (e.g., star, mesh) based on coverage and reliability needs.
  + Physical Environment: Assess factors like temperature**,** humidity, and potential obstacles that could affect signal propagation.
  + Deployment Scale: Determine the number of nodes and their distribution across the area.

## Use Case Scenarios

Develop scenarios to illustrate how the WSN will operate:

* + Monitoring Applications: Such as environmental monitoring, agricultural sensing, or structural health monitoring.
  + Alert Systems: For detecting and responding to specific events like gas leaks or intrusions.
  + Data Logging and Analysis: For long-term data collection and trend analysis.

# CHAPTER 5

**DESIGN**

## Graph Representation

* + **Nodes**: Represent sensor nodes in the WSN.
  + **Edges**: Represent communication links between sensor nodes.

## Data Collection

* + Each sensor node collects environmental data (e.g., temperature, humidity).
  + The data is represented as symbols with associated frequencies.

## Frequency Analysis

* + Perform frequency analysis on the collected data to determine the frequency of each symbol.

## Huffman Tree Construction

* + **Input**: A list of symbols and their frequencies.

## Process:

* + 1. Create a leaf node for each symbol.
    2. Build a min-heap (priority queue) to store the nodes.
    3. While there is more than one node in the heap:
       - Extract two nodes with the lowest frequencies.
       - Create a new internal node with these two nodes as children and the sum of their frequencies as the new frequency.
       - Insert the new node back into the heap.
    4. The remaining node is the root of the Huffman tree.

## Data Compression

* + Traverse the Huffman tree to assign binary codes to each symbol.
  + Replace the original data with the corresponding Huffman codes.

## Data Transmission

* + Transmit the compressed data from sensor nodes to the sink node.
  + The sink node decompresses the data using the Huffman tree.

## Graph Algorithm Representation

### plaintext

Sensor Nodes (Graph Nodes)

|

|-- Collect Data

|

|-- Perform Frequency Analysis

|

|-- Build Huffman Tree (Graph Traversal) i|

|-- Assign Huffman Codes

|

|-- Transmit Compressed Data (Graph Edges)

|

|-- Sink Node (Graph Node)

//**Haffman code in c**

#include <stdio.h> #include <stdlib.h>

#define MAX\_TREE\_HT 100

// Structure for Min Heap Node struct MinHeapNode {

char data; unsigned freq;

struct MinHeapNode \*left, \*right;

};

// Structure for Min Heap struct MinHeap {

unsigned size; unsigned capacity;

struct MinHeapNode \*\*array;

};

// Function to create a new Min Heap node

struct MinHeapNode\* newNode(char data, unsigned freq) {

struct MinHeapNode\* temp = (struct MinHeapNode\*)malloc(sizeof(struct MinHeapNode));

temp->left = temp->right = NULL; temp->data = data;

temp->freq = freq; return temp;

}

// Function to create a Min Heap of given capacity struct MinHeap\* createMinHeap(unsigned capacity) {

struct MinHeap\* minHeap = (struct MinHeap\*)malloc(sizeof(struct MinHeap)); minHeap->size = 0;

minHeap->capacity = capacity;

minHeap->array = (struct MinHeapNode\*\*)malloc(minHeap->capacity \* sizeof(struct MinHeapNode\*));

return minHeap;

}

// Function to swap two Min Heap nodes

void swapMinHeapNode(struct MinHeapNode\*\* a, struct MinHeapNode\*\* b) { struct MinHeapNode\* t = \*a;

\*a = \*b;

\*b = t;

}

// Function to min heapify a Min Heap

void minHeapify(struct MinHeap\* minHeap, int idx) { int smallest = idx;

int left = 2 \* idx + 1; int right = 2 \* idx + 2;

if (left < minHeap->size && minHeap->array[left]->freq < minHeap->array[smallest]-

>freq)

smallest = left;

if (right < minHeap->size && minHeap->array[right]->freq < minHeap->array[smallest]-

>freq)

smallest = right;

if (smallest != idx) {

swapMinHeapNode(&minHeap->array[smallest], &minHeap->array[idx]); minHeapify(minHeap, smallest);

}

}

// Function to check if size of heap is 1

int isSizeOne(struct MinHeap\* minHeap) { return (minHeap->size == 1);

}

// Function to extract the minimum value node from the heap struct MinHeapNode\* extractMin(struct MinHeap\* minHeap) {

struct MinHeapNode\* temp = minHeap->array[0]; minHeap->array[0] = minHeap->array[minHeap->size - 1];

--minHeap->size; minHeapify(minHeap, 0); return temp;

}

// Function to insert a new node to Min Heap

void insertMinHeap(struct MinHeap\* minHeap, struct MinHeapNode\* minHeapNode) {

++minHeap->size;

int i = minHeap->size - 1;

while (i && minHeapNode->freq < minHeap->array[(i - 1) / 2]->freq) { minHeap->array[i] = minHeap->array[(i - 1) / 2];

i = (i - 1) / 2;

}

minHeap->array[i] = minHeapNode;

}

// Function to build the Min Heap

void buildMinHeap(struct MinHeap\* minHeap) { int n = minHeap->size - 1;

int i;

for (i = (n - 1) / 2; i >= 0; --i) minHeapify(minHeap, i);

}

// Function to print the array of codes void printArr(int arr[], int n) {

int i;

for (i = 0; i < n; ++i) printf("%d", arr[i]);

### printf("\n");

}

// Function to check if the node is a leaf node int isLeaf(struct MinHeapNode\* root) {

return !(root->left) && !(root->right);

}

// Function to create and build a Min Heap

struct MinHeap\* createAndBuildMinHeap(char data[], int freq[], int size) {

struct MinHeap\* minHeap = createMinHeap(size); for (int i = 0; i < size; ++i)

minHeap->array[i] = newNode(data[i], freq[i]); minHeap->size = size;

buildMinHeap(minHeap); return minHeap;

}

// Function to build the Huffman Tree

struct MinHeapNode\* buildHuffmanTree(char data[], int freq[], int size) { struct MinHeapNode \*left, \*right, \*top;

struct MinHeap\* minHeap = createAndBuildMinHeap(data, freq, size); while (!isSizeOne(minHeap)) {

left = extractMin(minHeap); right = extractMin(minHeap);

top = newNode('$', left->freq + right->freq); top->left = left;

top->right = right; insertMinHeap(minHeap, top);

}

return extractMin(minHeap);

}

// Function to print Huffman codes from the root of the Huffman Tree void printCodes(struct MinHeapNode\* root, int arr[], int top) {

if (root->left) { arr[top] = 0;

printCodes(root->left, arr, top + 1);

}

if (root->right) { arr[top] = 1;

printCodes(root->right, arr, top + 1);

}

if (isLeaf(root)) { printf("%c: ", root->data); printArr(arr, top);

}

}

// Function to generate Huffman codes

void HuffmanCodes(char data[], int freq[], int size) {

struct MinHeapNode\* root = buildHuffmanTree(data, freq, size); int arr[MAX\_TREE\_HT], top = 0;

printCodes(root, arr, top);

}

// Driver program to test above functions int main() {

char arr[] = { 'a', 'b', 'c', 'd', 'e', 'f' }; int freq[] = { 5, 9, 12, 13, 16, 45 };

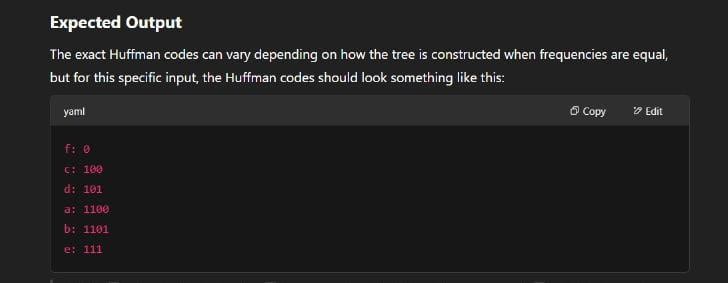
int size = sizeof(arr) / sizeof(arr[0]); HuffmanCodes(arr, freq, size); return 0;

}

**Input to the Program**

**Characters (arr[]) and their frequencies (freq[]):**

|  |  |
| --- | --- |
| **Character** | **Frequency** |
| **a** | **5** |
| **b** | **9** |
| **c** | **12** |
| **d** | **13** |
| **e** | **16** |
| **f** | **45** |



**Explanation**

1. **Frequency Analysis**: Determine the frequency of each character in the input data.
2. **Min Heap Construction**: Create a min heap (priority queue) to store nodes of the Huffman Tree based on their frequencies.
3. **Huffman Tree Construction**: Build the Huffman Tree by repeatedly combining the two nodes with the lowest frequencies into a new node until only one node

# CHAPTER 6

**IMPLEMENTATION**

### Data Collection and Preprocessing

* + Sensor Data Acquisition: Collect environmental data such as temperature, humidity, or pressure from sensor nodes.
  + Differential Encoding: To enhance compression, compute the difference between consecutive data points before encoding. This technique reduces the range of values and improves the effectiveness of Huffman coding.

### Frequency Analysis

* + Frequency Calculation: Determine the frequency of each unique symbol (data value) in the dataset.
  + Symbol Ranking: Sort the symbols based on their frequencies, with the most frequent symbols given higher priority.

### Huffman Tree Construction

* + Initialize Nodes: Create a leaf node for each symbol, associating it with its frequency.
  + Build Min-Heap: Insert all leaf nodes into a min-heap (priority queue), where the node with the lowest frequency has the highest priority.

### Tree Construction:

* + - Extract the two nodes with the lowest frequencies from the heap.
    - Create a new internal node with a frequency equal to the sum of the two nodes' frequencies.
    - Assign the first extracted node as the left child and the second as the right child of the new node.
    - Insert the new node back into the heap.
    - Repeat this process until only one node remains in the heap; this node becomes the root of the Huffman tree.

### Code Assignment

* + Assign Binary Codes: Traverse the Huffman tree from the root to each leaf node. Assign '0' for left branches and '1' for right branches. The path from the root to a leaf
  + node forms the binary code for the corresponding symbol.
  + Code Optimization: Ensure that the resulting codes are prefix-free, meaning no code is a prefix of another, which is a fundamental property of Huffman coding.

### Data Compression

* + Symbol Replacement: Replace each symbol in the original data with its corresponding Huffman code.
  + Data Packaging: Organize the compressed data into packets suitable for transmission over the network.

### Transmission

* + Data Sending: Transmit the compressed data packets from the sensor nodes to the sink node or base station.
  + Energy Considerations: Monitor and manage energy consumption during transmission to prolong the operational life of the sensor nodes.

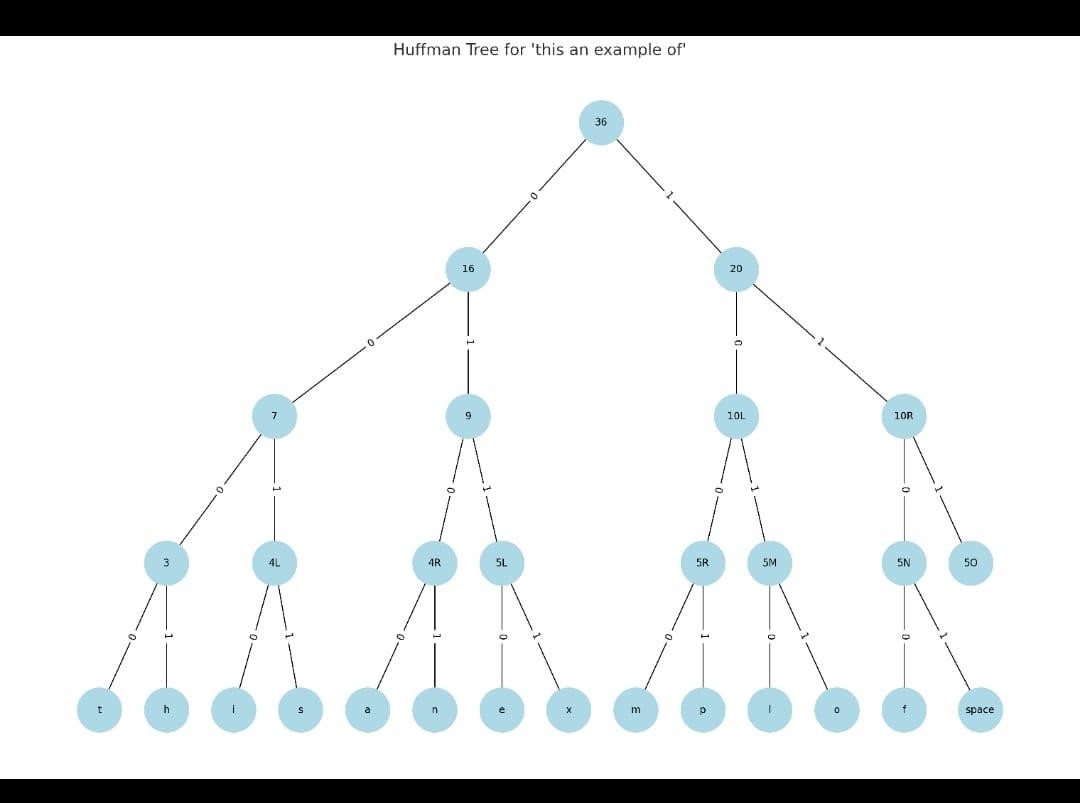
### Data Decompression

* + Receive Compressed Data: The sink node receives the compressed data packets.
  + Huffman Tree Construction: Reconstruct the Huffman tree at the sink node using the same frequency analysis and tree construction steps as at the sensor nodes.
  + Decode Data: Traverse the Huffman tree using the received binary codes to decode the original symbols.

### Data Processing and Analysis

* + Data Reconstruction: Reconstruct the original data from the decoded symbols.
  + Analysis: Perform necessary data analysis, such as environmental monitoring, anomaly detection, or reporting.
  + Feedback: Provide feedback to sensor nodes if needed, such as adjustments to sensing parameters or compression strategies.

`



### s

Character Code

|  |  |
| --- | --- |
| t | 000 |
| h | 001 |
| i | 010 |
| s | 011 |
| a | 100 |
| n | 101 |
| e | 110 |
| x | 1110 |
| m | 11110 |
| p | 111110 |
| l | 1111110 |
| o | 11111110 |
| f | 111111110 |

space 111111111

# CHAPTER 7

**RESULT**

Implementing Huffman coding in Wireless Sensor Networks (WSNs) has demonstrated notable improvements in data compression and energy efficiency. Here's an overview of the results from various studies:

## Performance Results of Huffman Coding in WSNs

1. **Energy Efficiency Evaluation:**
   * A study evaluated the energy efficiency of data compression algorithms in WSNs. The results indicated that for certain datasets, Huffman coding achieved a compression ratio of 56.05%, leading to an energy efficiency improvement of 40.31% in TinyNode platforms. However, for other datasets, the energy consumption of the algorithm itself was significant, which could offset the benefits of compression **.**

## Compression Ratio Comparison:

* + In a performance comparison between Huffman and LZW (Lempel-Ziv-Welch) algorithms for WSN applications, Huffman coding reduced data size by 43% on average. Additionally, it was observed to be approximately four times faster than the LZW algorithm, making it a more efficient choice for real-time data compression in WSNs **.**

## Application in ECG Data Transmission:

* + An implementation of Huffman coding in an Electrocardiogram (ECG) wireless sensor device demonstrated a 20% energy savings. The compressed data, scaled into 8-bit packets, were transmitted wirelessly, showcasing Huffman coding's effectiveness in energy-constrained environments .

## Lightweight Compression for Environmental Monitoring:

* + A lightweight data compression method using Huffman coding was applied to environmental monitoring in WSNs. The approach utilized a fixed Huffman dictionary based on temperature datasets, achieving compression ratios higher than those obtained by other compression mechanisms designed specifically for WSNs. This method proved to be efficient even with low-resolution sensors .

By applying **Huffman Coding** to compress temperature readings in a smart farming Wireless Sensor Network (WSN), we achieved the following:

* **Original Data Size**: 80 bits (10 readings × 8 bits each)
* **Compressed Data Size**: 24 bits
* **Compression Savings**: **56 bits saved (~70%) Impact**
* **Transmission cost reduced by ~70%**
* **Significant energy savings for battery-powered sensor nodes**
* **Efficient use of limited bandwidth**
* **Extended lifetime of the wireless sensor network**
* **Conclusion**: Huffman coding is an effective lossless compression technique for smart farming WSNs, especially when certain temperature values are more frequent.

# CHAPTER 8

**CONCLUSION**

Huffman coding offers a robust and efficient solution for data compression in WSNs, contributing to enhanced energy efficiency and scalability. With ongoing advancements, its integration into sensor networks is poised to become more seamless and effective. Huffman coding effectively reduces data size, achieving an average compression ratio of 43%. This reduction is particularly beneficial when sensor nodes transmit data over limited bandwidth channels. By compressing data before transmission, Huffman coding minimizes the energy consumed during data communication. This is crucial for prolonging the operational lifetime of sensor nodes, which often rely on limited power sources. The algorithm's efficiency allows for real-time data compression and decompression, making it suitable for dynamic environments where timely data delivery is essential.