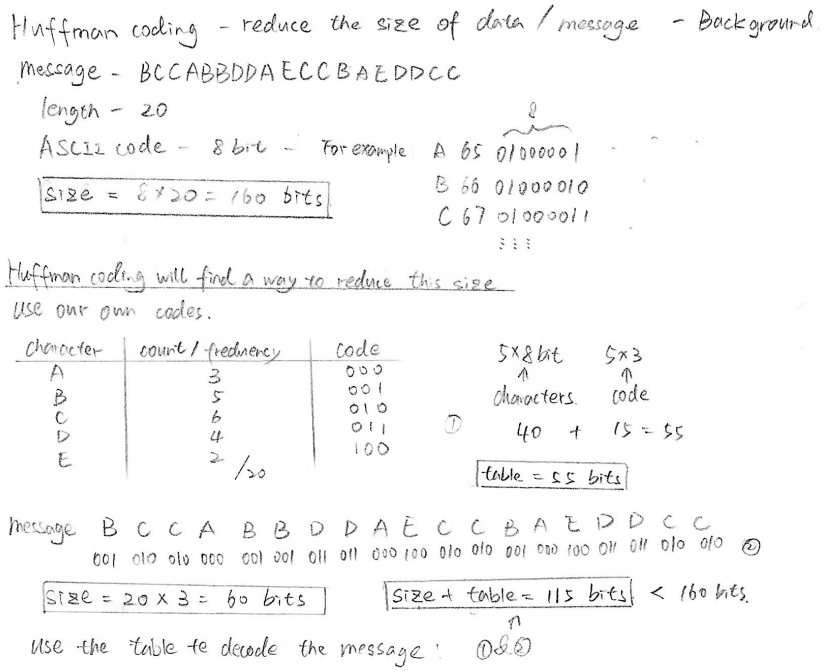
Assignment 6 DESIGN.pdf

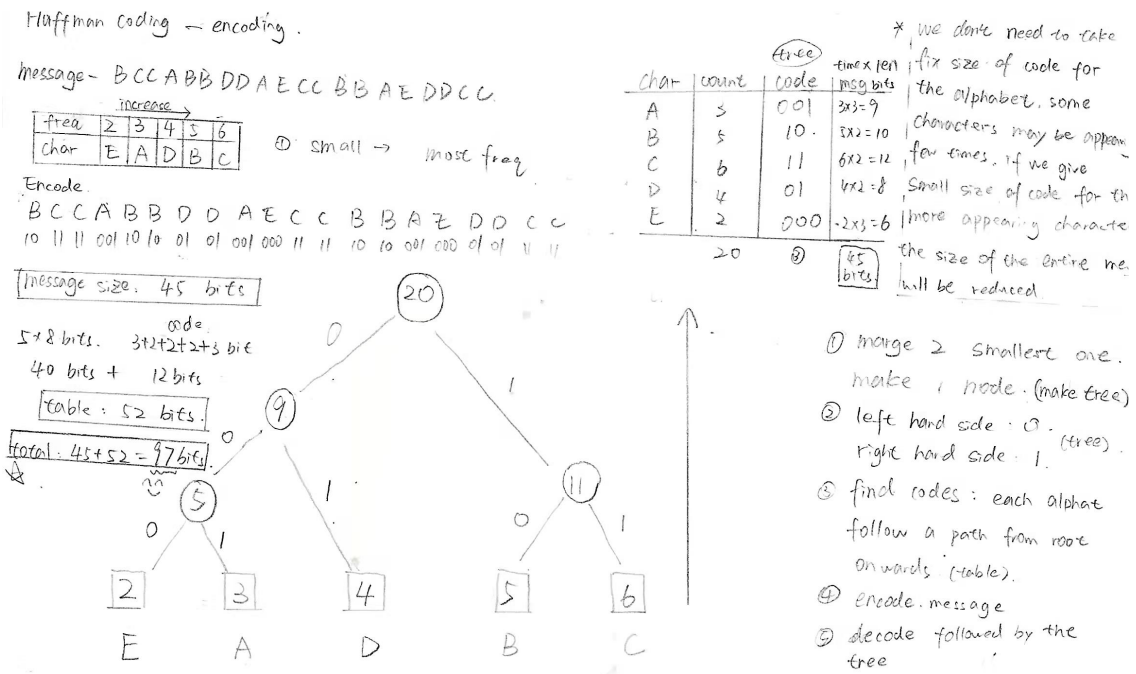
Ning Jiang

Description of Program: For this assignment, we mainly need to do huffman coding which is a way to reduce the size of data/message. The logical steps are:

1. Merge 2 smallest one, make 1 node. (make tree)
2. Left hand side is 0 and the right side is 1. (tree)
3. Find codes: each alphabet follow a path from root onwards. (table)
4. Encode message
5. Decode followed by the tree.

Basic rules and background:





Files to be included in directory “asgn6”:

1. encode.c: This file will contain your implementation of the Huffman encoder.
2. decode.c: This file will contain your implementation of the Huffman decoder.
3. defines.h: This file will contain the macro definitions used throughout the assignment. You may not modify this file.
4. header.h: This will will contain the struct definition for a file header. You may not modify this file.
5. node.h: This file will contain the node ADT interface. This file will be provided. You may not modify this file.
6. node.c: This file will contain your implementation of the node ADT.
7. pq.h: This file will contain the priority queue ADT interface. This file will be provided. You may not modify this file.
8. pq.c: This file will contain your implementation of the priority queue ADT. You must define your priority queue struct in this file.

9. code.h: This file will contain the code ADT interface. This file will be provided. You may not
modify this file.
10. code.c: This file will contain your implementation of the code ADT.
11. io.h: This file will contain the I/O module interface. This file will be provided. You may not modify
this file.
12. io.c: This file will contain your implementation of the I/O module.
13. stack.h: This file will contain the stack ADT interface. This file will be provided. You may not
modify this file.
14. stack.c: This file will contain your implementation of the stack ADT. You must define your stack
struct in this file.
15. huffman.h: This file will contain the Huffman coding module interface. This file will be provided.
You may not modify this file.
16. huffman.c: This file will contain your implementation of the Huffman coding module interface.
16. Makefile
17. README.md
18. DESIGN.pdf

2. Structure & explanation / Pseudocode:
Explanations for some of the equations are cited from asgn6.pdf.
encode.c
Command line options:
-h: Prints out a help message
-i infile
-o outfile
-v: Prints compression statistics to stderr.
Algorithm
1. Compute a histogram of the file. In other words, count the number of occurrences of each unique
symbol in the file.
2. Construct the Huffman tree using the computed histogram. This will require a priority queue.
3. Construct a code table. Each index of the table represents a symbol and the value at that index the
symbol's code. You will need to use a stack of bits and perform a traversal of the Huffman tree.
4. Emit an encoding of the Huffman tree to a file. This will be

done through a post-order traversal of
the Huffman tree. The encoding of the Huffman tree will be referred to as a tree dump.
5. Step through each symbol of the input file again. For each symbol, emit its code to the output file.

decode.c
Command line options:
-h: Prints out a help message
-i infile
-o outfile
-v: Prints decompression statistics to stderr.

Algorithm
1. Read the emitted (dumped) tree from the input file. A stack of nodes is needed in order to recon-
struct the Huffman tree.
2. Read in the rest of the input file bit-by-bit, traversing down the Huffman tree one link at a time.
Reading a 0 means walking down the left link, and reading a 1 means walking down the right link.
Whenever a leaf node is reached, its symbol is emitted and you start traversing again from the root.

node.c
crate a node to know the node in the message to make a tree. Each node(except leaf) contain a pointer to its left child, a pointer to its right child, a symbol, and the frequency of that symbol. We will make function to create, delete, join(two nodes to a parent), print the node.

pq.c
Priority queue is used to store nodes to make a tree. Priority queue is actually a circular queue which will use insertion sort after enqueue a Node. we will make functions to create, delete, check whether the queue is empty, check whether the queue is full, count the size, enqueue, dequeue and print the priority queue.

My pq:
LSB MSB
Front _ _ _ _ ... _ _ _ rare
Pop push

Enqueue:
If queue is fill, return false
Else, rare = rare + 1
Node[rare] = n
Do the insertion sort and return true

Dequeue:
If queue is empty, return false
Else: Node[front] = n, front = front + 1 and return true

Code.c
a stack of bits while traversing the tree in order to create a code for each symbol. We willcreate functions for initing the code, calculating the size, checking whether the code is empty, checking whether the coed is full, setting bit at index i in code, clearing bit i in code, getting the bit, pushing a bit in the code, poping a bit from the code, print the code.
Stack:

_ _ _ ……_ _ _
Top (push and pop from there)

Io.c
// this file is too hard I am still figuring out.

Stack.c
This is also a stack for decoder. Like code.c
Stack:

_ _ _ ……_ _ _
Top (push and pop from there)
We need to createm delete, check whether the stack is empty, check whether the stack is full, calculate the stack size which is the top, push and pop.

Hufmman.c
Haven’t write yet.

Error Handing:

Credit: