Huffman Coding

Project 5

Driver Program

- Name your file huffman.c.
- A driver program with a main function that takes `argv[1]` as `input_file_name` and `argv[2]` as `output_file_name`.
- The program will compress the input file into the output file with ".z" extension.
- Include pack.h which contains the definition of the node structure:

```
struct node {
    struct node *parent;
    int count;
};
```

Goal

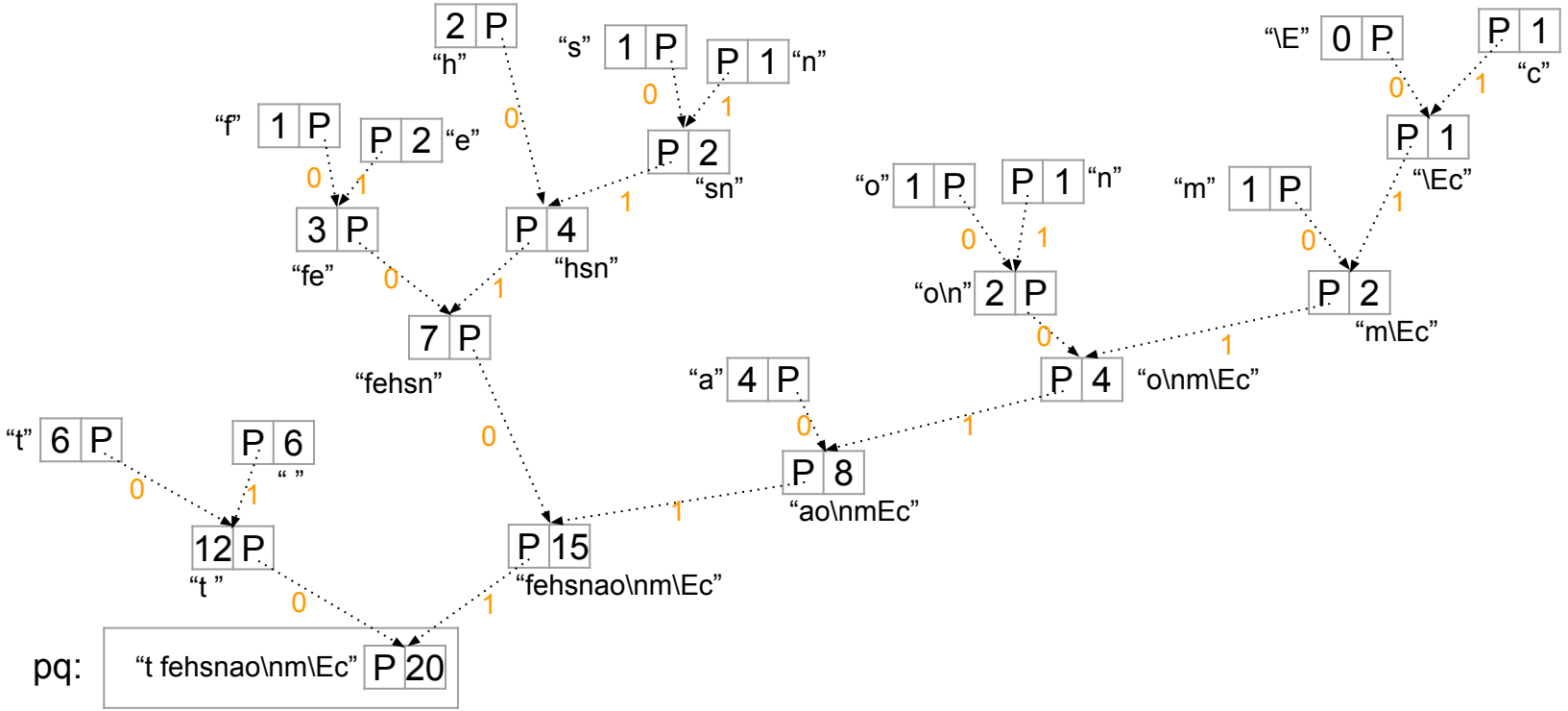
- Our goal is to compress a text file (input.txt):

```
the fat cat sat on the mat
```

- 27 characters including spaces and ‘\n’
  - 27*8 bits = 216 bits
  - Can we use less bits to store this sentence?
  - Maybe use less bits for higher frequency characters?

‘ ‘	‘a’	‘c’	‘e’	‘f’	‘h’	‘m’	‘n’	‘o’	‘s’	‘t’	‘\n’
6	4	1	2	1	2	1	1	1	1	6	1

Huffman Tree



Huffman Step 1

- Create an integer array to store the frequencies for the characters.
  - `counts[256 + 1]`, last one is for the `EOF` with frequency 0.
- Create another array `struct node *nodes[257]` with each of them pointing to `NULL` first.

	…	‘ ‘	…	‘a’	‘b’	‘c’	‘d’	‘e’	‘f’	‘g’	‘h’	…	‘m’	‘n’	‘o’	…	‘s’	‘t’	…	EOF
counts:	…	0	…	0	0	0	0	0	0	0	0	…	0	0	0	…	0	0	…	0
nodes:	…	N	…	N	N	N	N	N	N	N	N	…	N	N	N	…	N	N	…	N

Huffman Step 2

- Example: `the fat cat sat on the mat`
- Count the number of frequencies of each character in the file. Keep track of these counts in an array.
  - `getc(fp)`

ASCII:	…	10	…	32	…	97	98	99	100	101	102	103	104	…	109	110	111	…	115	116	…	256
character:	…	‘\n’	…	‘ ’	…	‘a’	‘b’	‘c’	‘d’	‘e’	‘f’	‘g’	‘h’	…	‘m’	‘n’	‘o’	…	‘s’	‘t’	…	EOF
counts:	…	1	…	6	…	4	0	1	0	2	1	0	2	…	1	1	1	…	1	6	…	0
nodes:	…	N	…	N	…	N	N	N	N	N	N	N	N	…	N	N	N	…	N	N	…	N

Before Huffman Step 3

- We need a function to create a node.
  - Create a last generation node (no children and no parent yet).
  - Create a parent node pointed by two children nodes.
- We need to tell createQueue how to compare two nodes

Private Functions

- Create a private function to make a new node:

```
mknode(count, left_node, right_node):
    Malloc for a new_node pointer;
    new_node→count=count; (sum of the two children.)
    new_node→parent=NULL; (new node has no parent yet.)
    If left_node not NULL, left_node→parent=new_node;
    If right_node not NULL, right_node→parent=new_node;
    Return new_node;
```

- NO children for the last generation nodes.

Private Functions

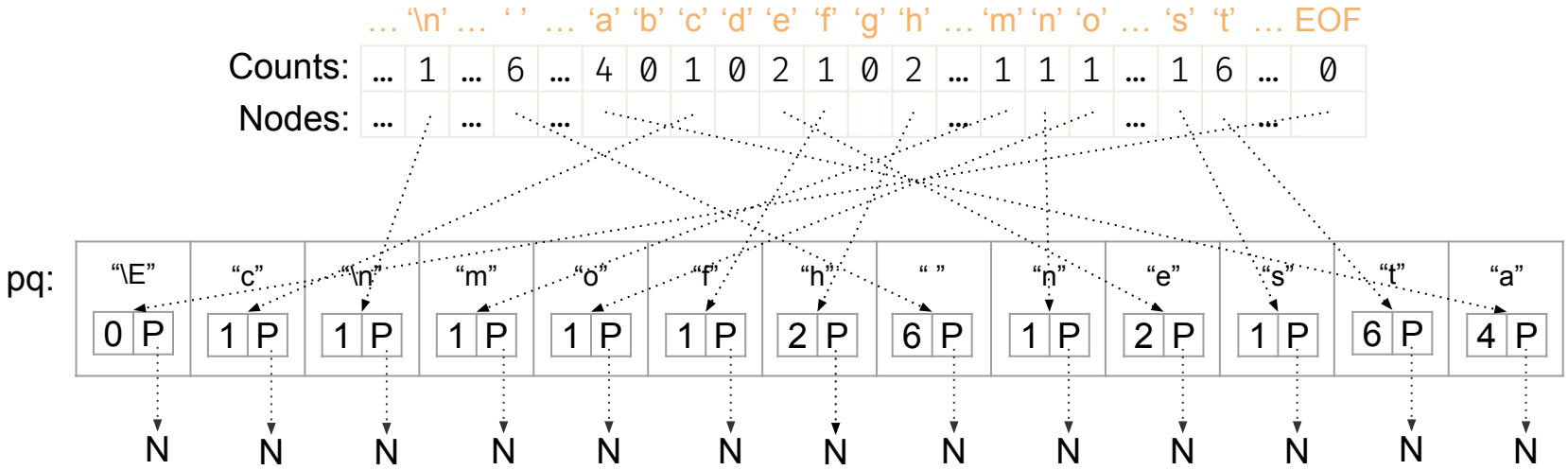
- We need a min heap to sort the nodes by its frequency so we are going to use the priority queue algorithm from last week but we need to tell it how to compare.
- Create a comparison function for createQueue(cmp):

```
(t1→count < t2→count) ? -1 : (t1→count > t2→count)
```

  - t1<t2: -1
  - t1==t2: 0
  - t1>t2: 1

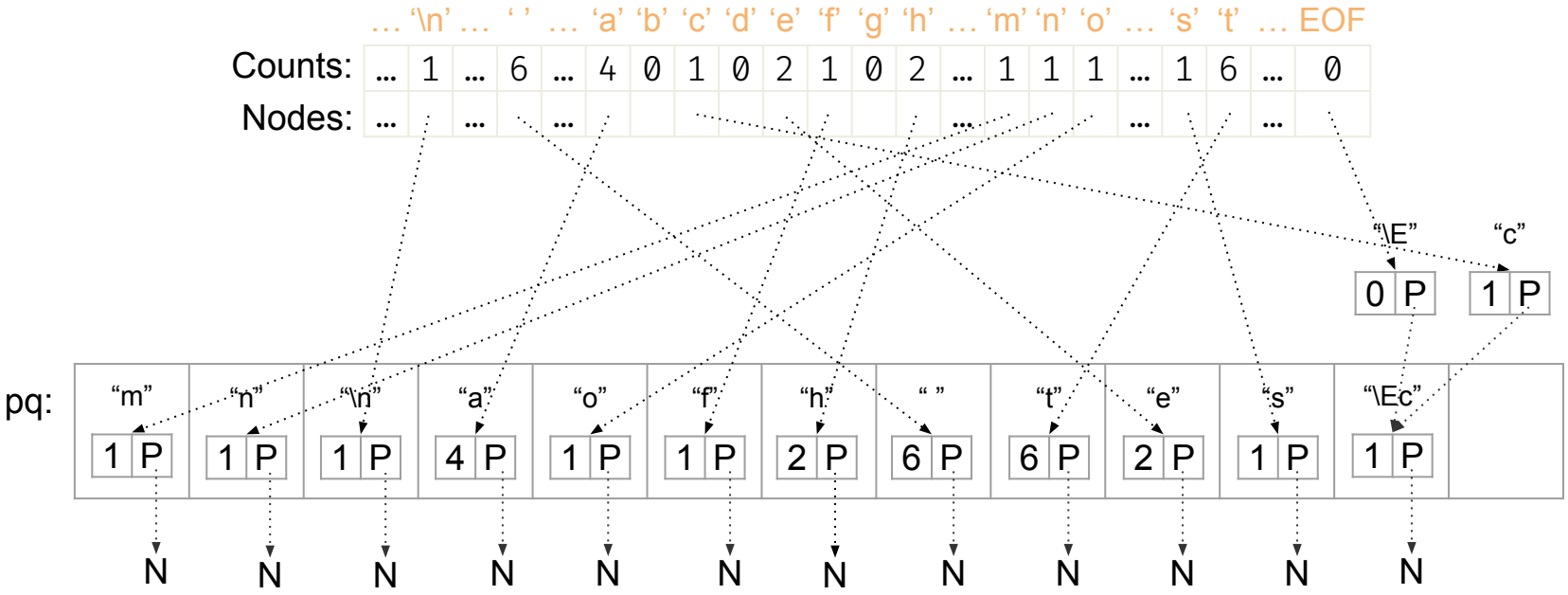
Huffman Step 3

- Populate the priority queue with the last generation nodes.
  - Call `mknode` for the each character with nonzero frequency and insert the node to the pq. And, create one extra for the `EOF` with zero count.



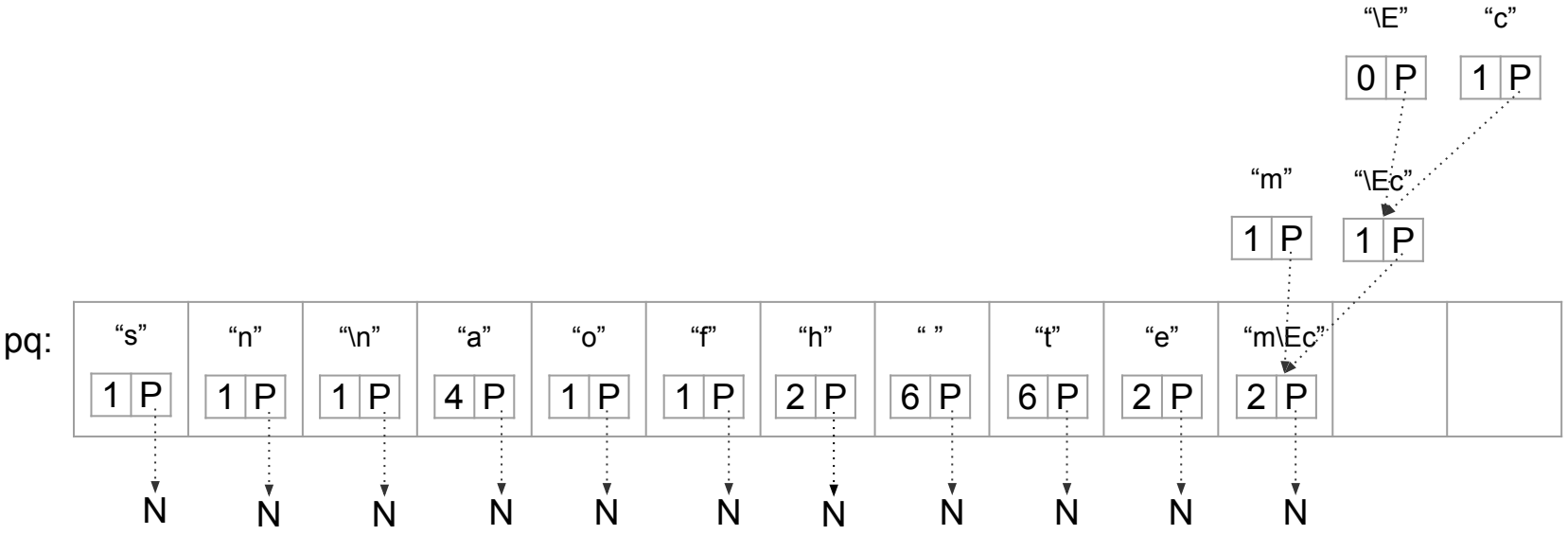
Huffman Step 4

- Building the tree, by taking first two out and create a new one with the count of the sum of the two, then put the new one back.

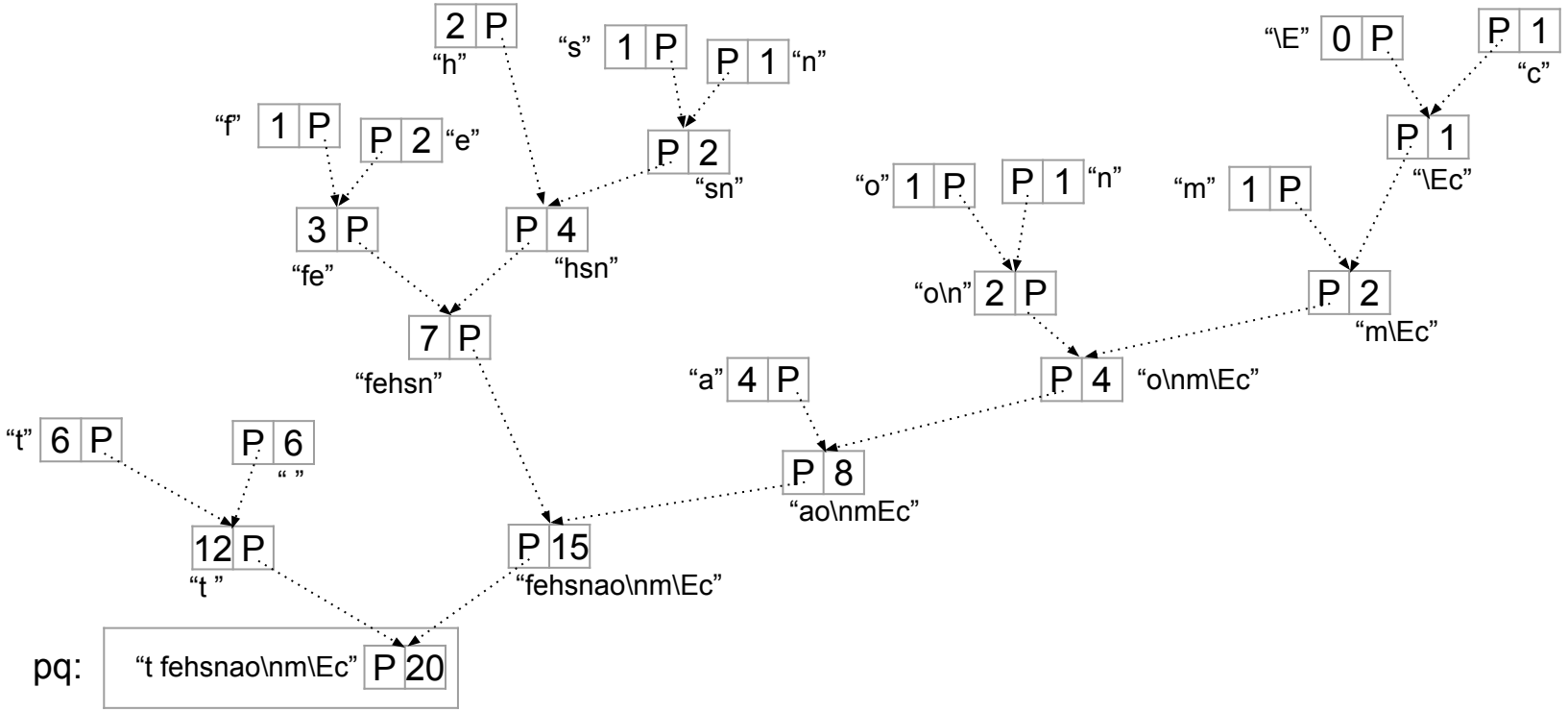


Huffman Step 4 (repeat)

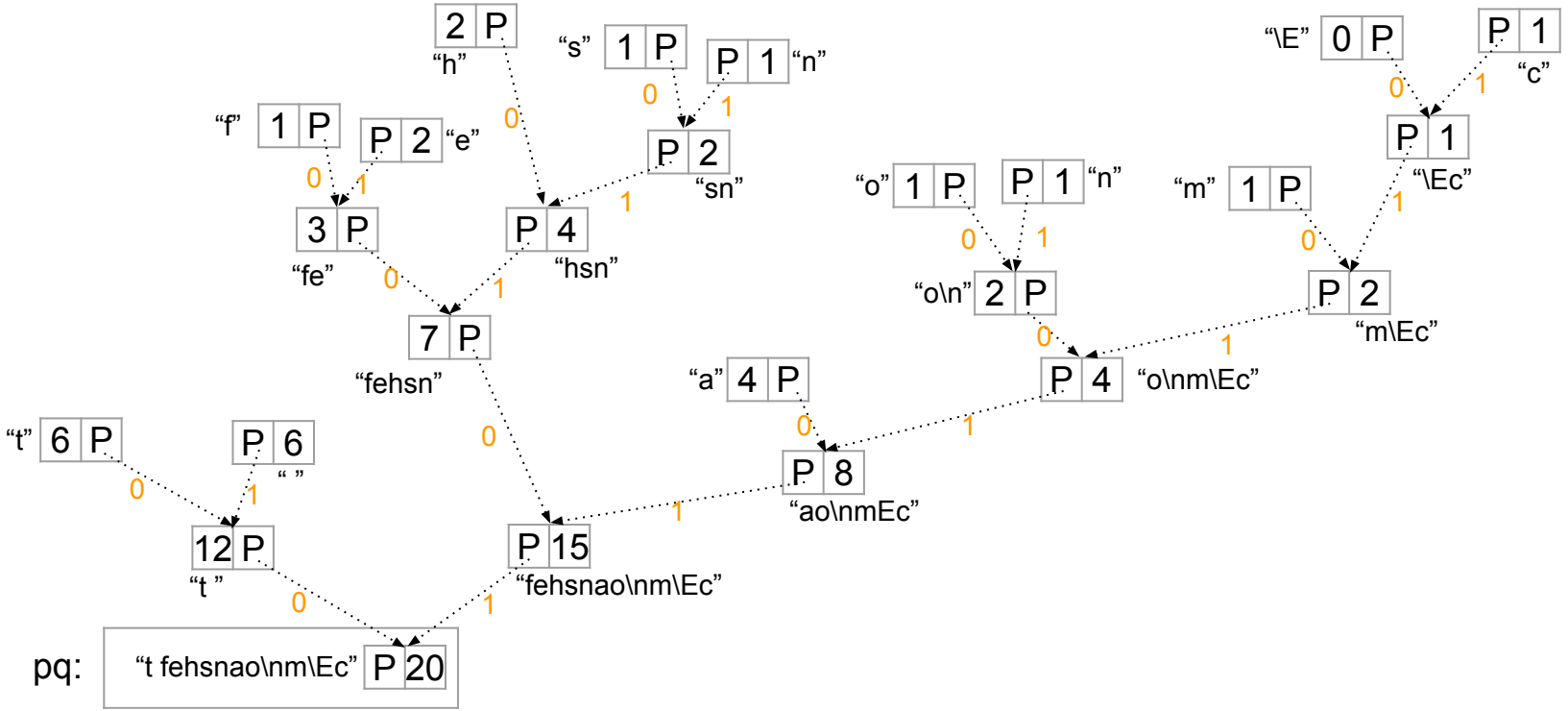
- Repeat Step 4 until only one left in the pqueue.



Huffman Tree



Huffman Tree



Huffman Code

\n	“11101” 5 bits * 1 occurrences = 5 bits
	“01” 2 * 6 = 12 bits
a	“110” 3 * 4 = 12 bits
c	“111111” 6 * 1 = 6 bits
e	“1001” 4 * 2 = 8 bits
f	“1000” 4 * 1 = 4 bits
h	“1010” 4 * 2 = 8 bits
m	“11110” 5 * 1 = 5 bits
n	“10111” 5 * 1 = 5 bits
o	“11100” 5 * 1 = 5 bits
s	“10110” 5 * 1 = 5 bits
t	“00” 2 * 6 = 12 bits
EOF	“111110” 6 * 0 = 0 bits
Total	87 bits (vs. 216 bits)

“the fat cat sat on the mat”:
00101010010110001100001111111111000
01101101100001111001011101001010101
01011111011000

Huffman Step 5

- Print out occurrences and length of bits for each character
  - One more private function `depth(node)` to calculate the number of bits.
  - If `isprint(c)` is False:

    `printf("%03o",c)`
  - Print `counts[c]`, `depth(nodes[c])`, and `counts[c] * depth(nodes[c])`

```
012: 1 x 5 bits = 5 bits
' ': 6 x 2 bits = 12 bits
'a': 4 x 3 bits = 12 bits
'c': 1 x 6 bits = 6 bits
'e': 2 x 4 bits = 8 bits
'f': 1 x 4 bits = 4 bits
'h': 2 x 4 bits = 8 bits
'm': 1 x 5 bits = 5 bits
'n': 1 x 5 bits = 5 bits
'o': 1 x 5 bits = 5 bits
's': 1 x 5 bits = 5 bits
't': 6 x 2 bits = 12 bits
400: 0 x 6 bits = 0 bits
total bits required = 87
```

Huffman Step 6

- Call pack(input_file_name, output_file_name, nodes_array) to generate the compressed file.
- For our example, pack() should print out:

```
total bits required = 87 bits
```

File Decompression

- Command to run the program to compress the file:

  ./huffman input.txt output.z
- Command to decompress the compressed file:

  gunzip output.z
- Command to check the decompressed file:

  cat output

Submission

- tar -czvf project5.tar folder_path
  - folder_path is the directory of the folder that contains both pqueue.c, huffman.c and all other files.
- Submission deadline: Sunday, 11:59pm.
- Late Submission deadline: Monday, 11:59pm.
- Demo deadline: the end of the lab section next week.