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;
};
```

Huffman Step 1

- Create an integer array to store the occurrences 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	…	\E
counts:	…	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

Huffman Step 2

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

	…	97	98	99	100	101	102	103	104	…	109	110	111	…	115	116	…	256
	…	a	b	c	d	e	f	g	h	…	m	n	o	…	s	t	…	\E
counts:	…	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

Private Functions

- Create a private function to make new node:

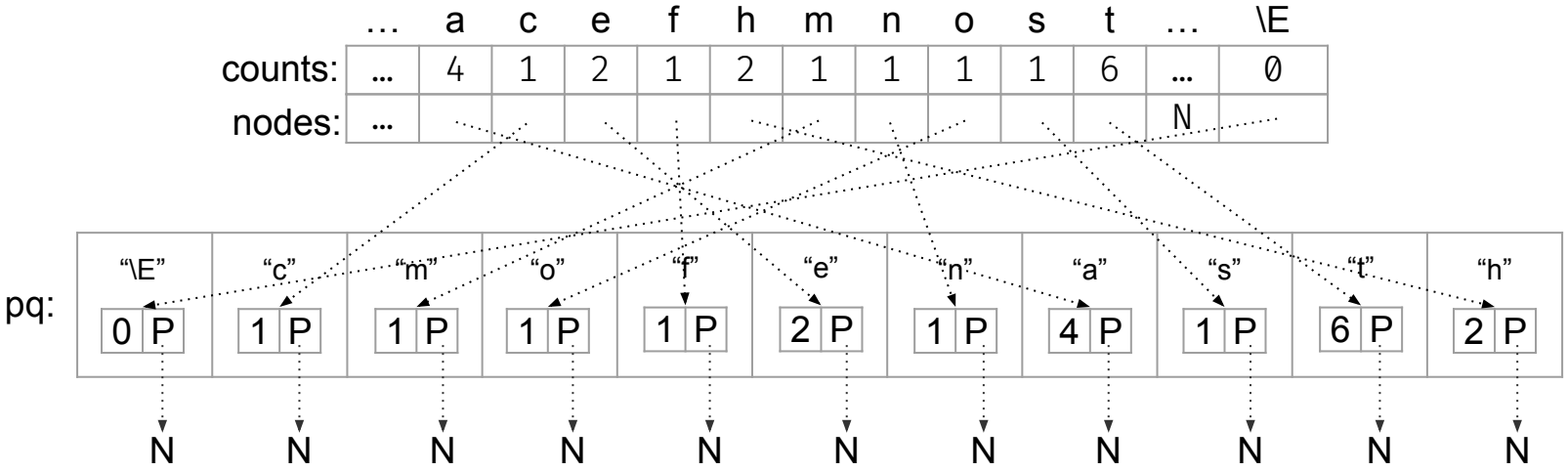
```
mknode(data, left_node, right_node):
    Malloc for a new_node;
    new_node→count=data;
    new_node→parent=NULL; (new node has no parent yet.)
    If left_node not NULL, left_node→parent=new_node;
    If right_node not NULL, right_node→parent=new_node;
    Return new_node;
```

- Create a comparison function for createQueue(cmp):

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

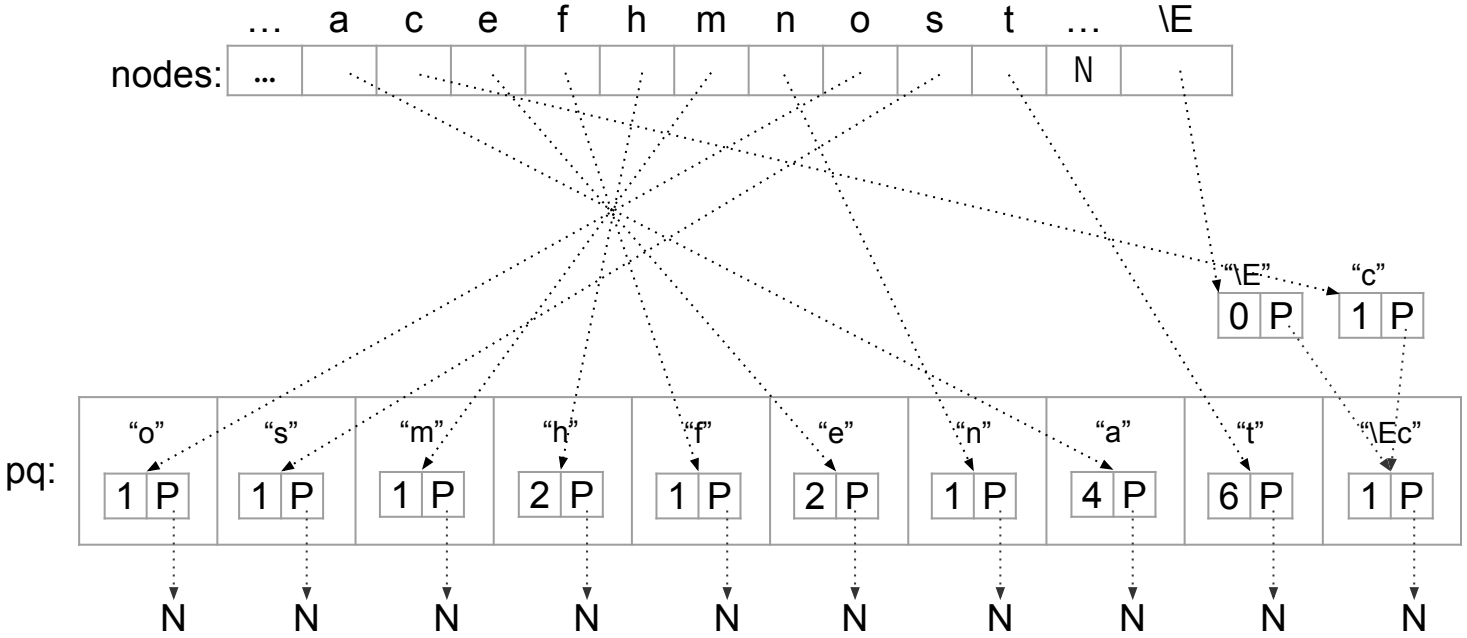
Huffman Step 3

- Create a priority queue to build huffman tree bottom up (leaves to root)
  - Call `mknode` for the each character with nonzero count and insert to the pq. And, create one extra for the `EOF` with zero count.



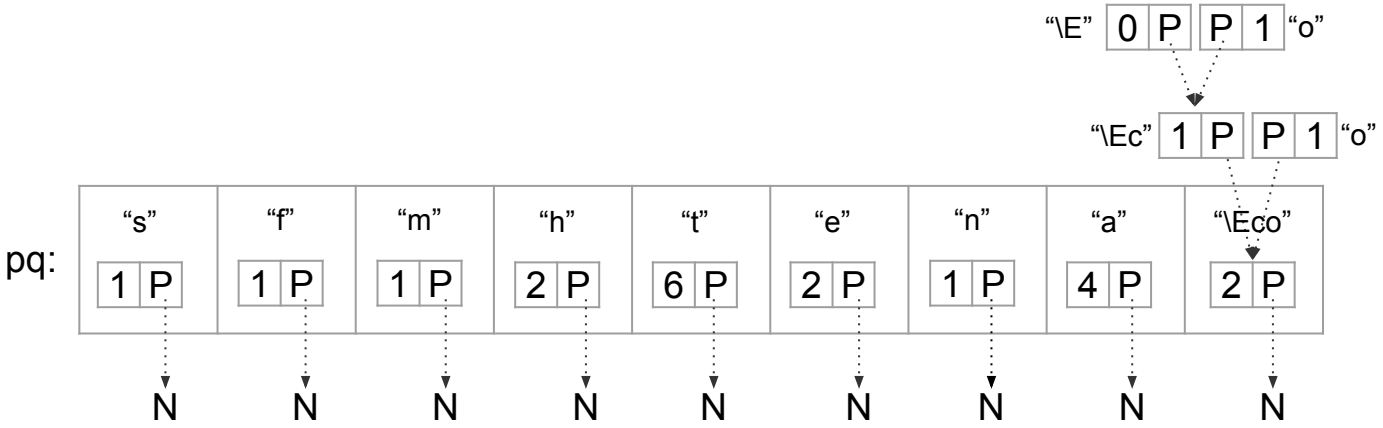
Huffman Step 4

- 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.

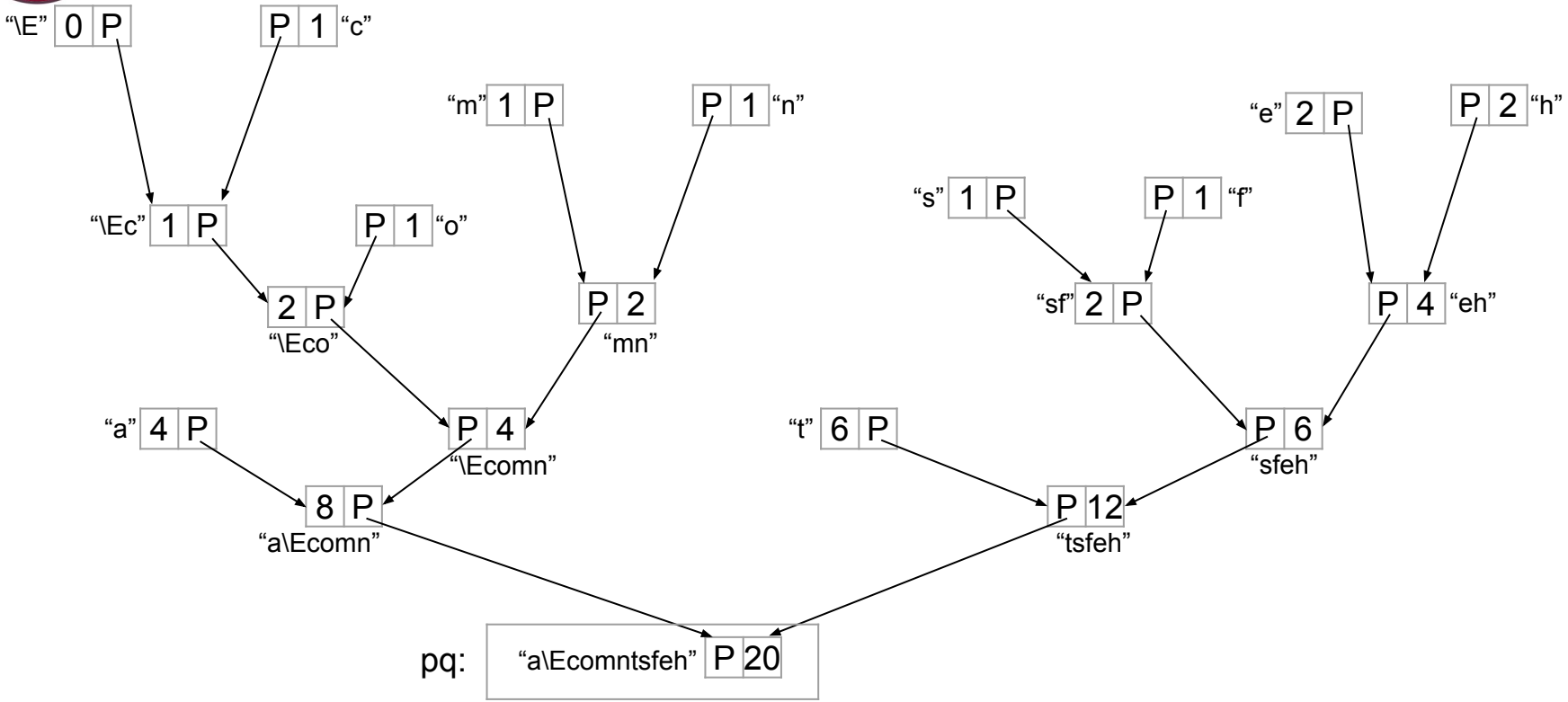


Huffman Step 4

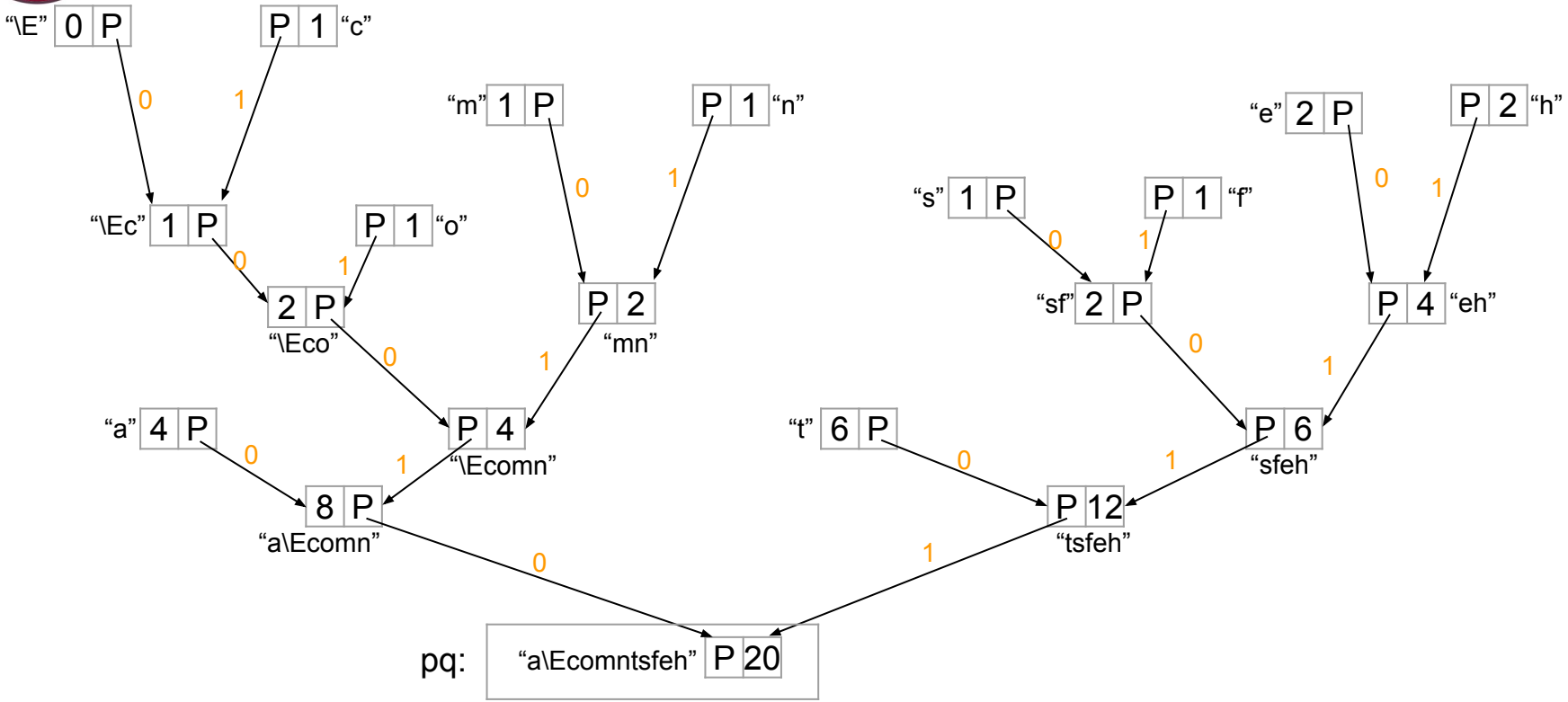
- Repeat Step 4 until only one left in the pqueue.



Huffman Tree



Huffman Tree



Huffman Code

a	“00” 2 bits * 4 occurrences = 8 bits
c	“01001” 5 * 1 = 5 bits
e	“1110” 4 * 2 = 8 bits
f	“1101” 4 * 1 = 4 bits
m	“0110” 4 * 1 = 4 bits
n	“0111” 4 * 1 = 4 bits
o	“0101” 4 * 1 = 4 bits
s	“1100” 4 * 1 = 4 bits
t	“10” 2 * 6 = 4 bits
EOF	“01000” 5 * 0 = 0 bits

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
```

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, March 12th, 11:59pm.
- Late Submission deadline: Monday, March 13th, 11:59pm.
- Demo deadline: the end of the lab section next week.