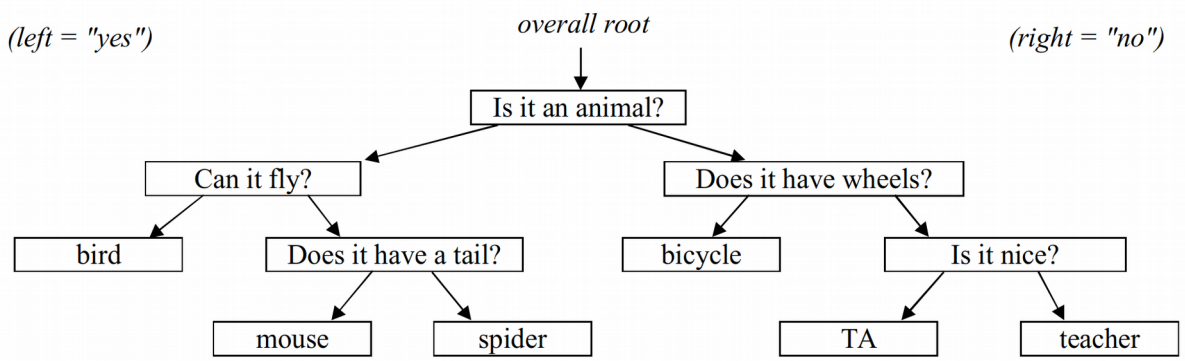
CS 106X
Homework 6, Binary Trees:
21 Questions, Huffman Encoding

Part A: 21 Questions

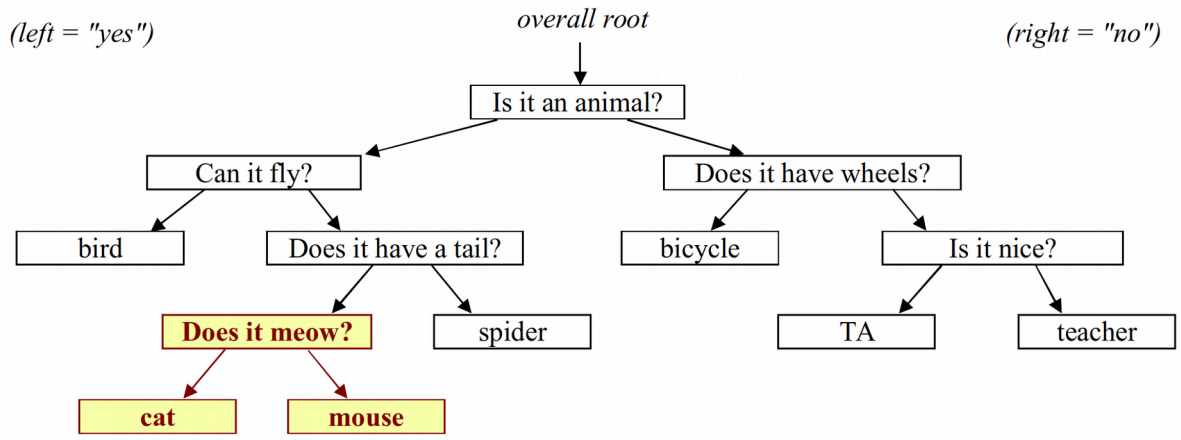
- "sequel" to past **20 Questions** recursion problem:
  - stores question/answer data in a **binary tree**
  - "**learns**" after losing a game by asking player for new data

```
// questions.txt
Q:Is it an animal?
Q:Can it fly?
A:bird
Q:Does it have a tail?
A:mouse
A:spider
Q:Does it have wheels?
A:bicycle
Q:Is it nice?
A:TA
A:teacher
```



Growing question tree

- when computer loses, asks human player for a new Q/A node




```
// log of execution
Is it an animal? y
Can it fly? n
Does it have a tail? y
Is your object: mouse? no
Drat, I lost. What is your object? cat
Type a yes/no question to distinguish cat from mouse: Does it meow?
And what is the answer for cat? y
```

Code they will write

- game state is now stored in a QuestionTree class:

```
class QuestionTree {
public:
    QuestionTree();
    ~QuestionTree();
    int getGamesLost() const;
    int getGamesWon() const;
    void playGame();
    void readData(istream& input);     // save to file
    void writeData(ostream& output);   // load from file

private:
    QuestionNode* root;   // node = {string data, node* yes/no}
    ...
};
```

Part B: Huffman encoding

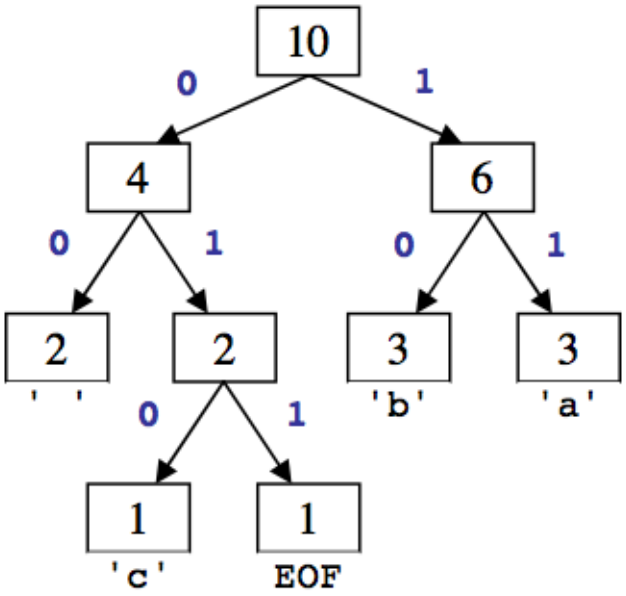
- Uses variable lengths for different characters to take advantage of their relative frequencies.

Char	ASCII value	ASCII (binary)	Hypothetical Huffman
' '	32	00100000	10
'a'	97	01100001	0001
'b'	98	01100010	01110100
'c'	99	01100011	001100
'e'	101	01100101	1100
'z'	122	01111010	00100011110

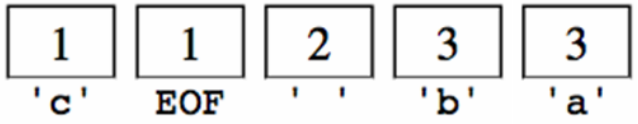
Huffman compression

1. Count occurrences of each char in file

```
{' ':2, 'a':3, 'b':3, 'c':1, EOF:1}
```



2a. Place chars, counts into **priority queue**



2b. Use PQ to create **Huffman tree** →

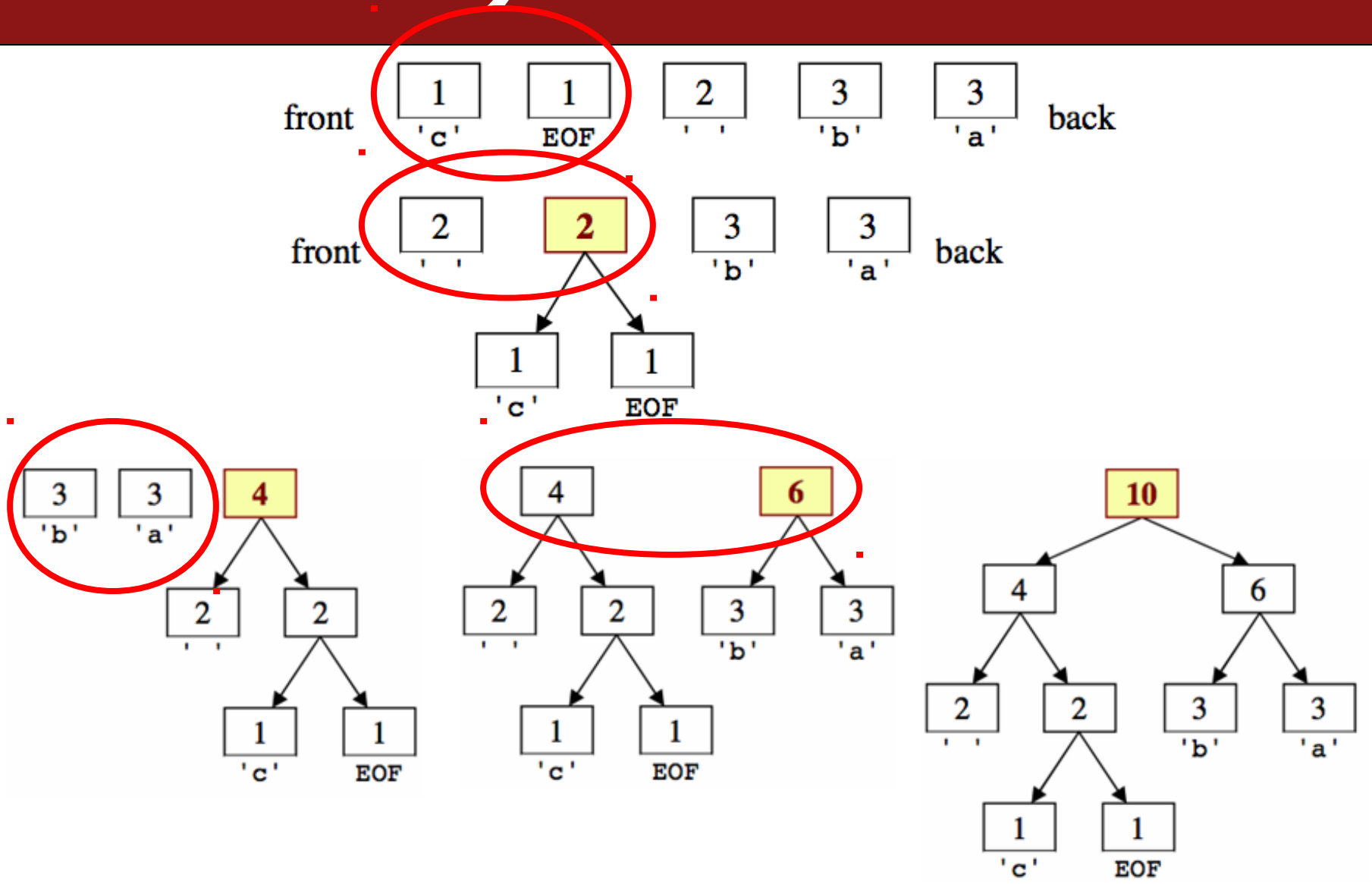
3. Traverse tree to find (char → binary) **map**

```
{' ':00, 'a':11, 'b':10, 'c':010, EOF=011}
```

4. Convert to binary (For each char in file, look up binary rep in map)

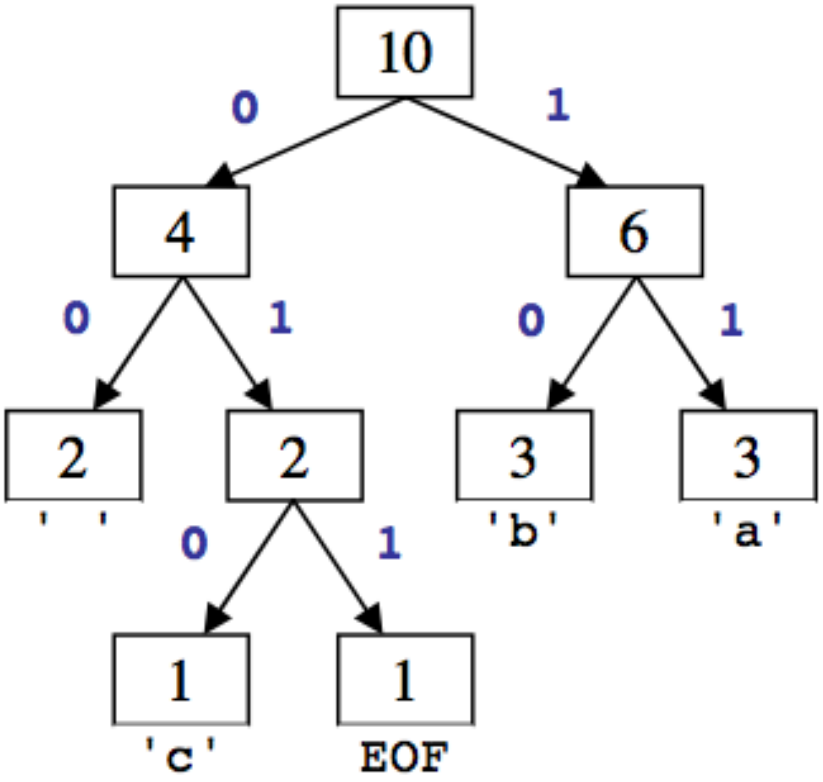
```
11 10 00 11 10 00 010 1 1 10 011 00
```

2b) Build tree



3) Tree to binary encodings

- The Huffman tree tells you the binary encodings to use.
  - left means **`0`**, right means **`1`**
  - example: `'b'` is `10`
  - example: `'c'` is `010`



4) Encode the file

- Based on the preceding tree, we have the following encodings:

  `{' ':00, 'a':11, 'b':10, 'c':010, EOF:011}`

  - The text "`ab ab cab`" would be encoded as:

char	'a'	'b'	' '	'a'	'b'	' '	'c'	'a'	'b'	EOF
binary	11	10	00	11	10	00	010	11	10	011

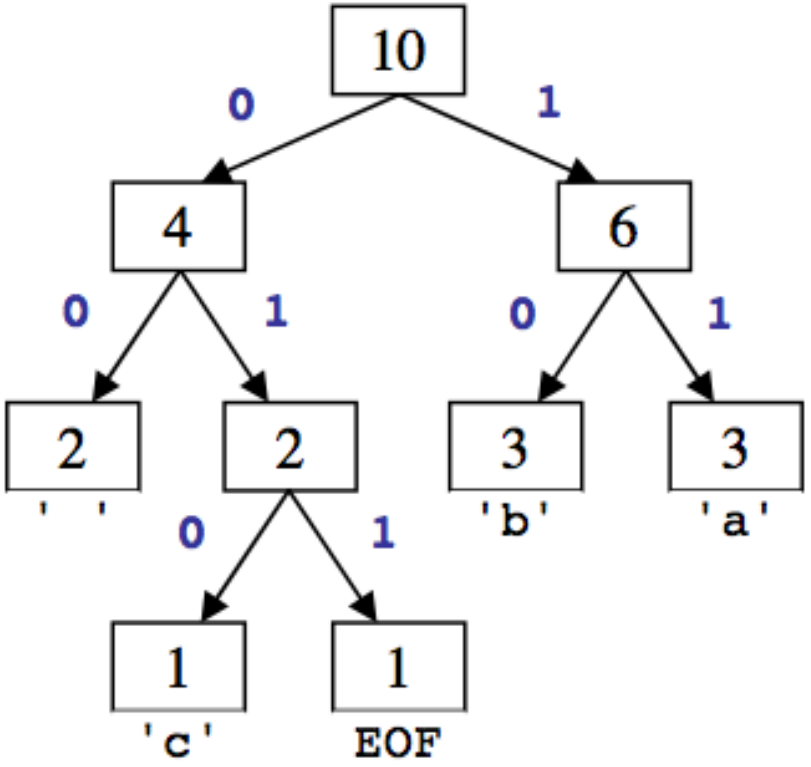
  - Overall: `1110001110000101110011`, (22 bits, ~3 bytes)

byte	1	2	3
char	a b a	b c a	b EOF
binary	11 10 00 11	10 00 010 1	1 10 011 00

Decompressing

```
10110100011010110 11
 b a   c _ a   c a
```

- Read each bit one at a time.
- If it is 0, go left; if 1, go right.
- If you reach a leaf, output the character there and go back to the tree root.



- Output:

```
bac aca
```

Bit I/O streams

- `ibitstream`: Reads one bit at a time from input.

`int` **`readBit`**`()`	Reads a single 1 or 0; returns -1 at end of file

- `obitstream`: Writes one bit at a time to output.

`void` **`writeBit`**`(int bit)`	Writes a single bit (must be 0 or 1)

  - `i/obitstream` also contain the members from `i/ostream`.
    - open, read, write, fail, close