pset 6: Huff'n Puff

Zamyla Chan | zamyla@cs50.net


cuteheaven.com

Toolbox

- Huffman trees
- distribution code
- pen and paper

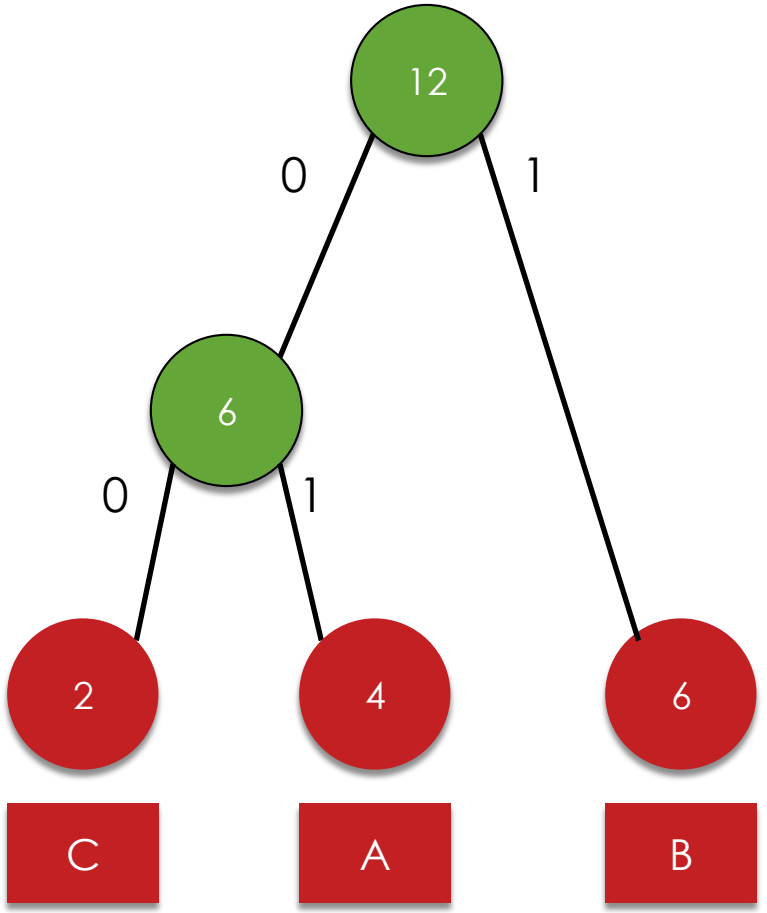
Huffman Trees

Huffman trees

- Most frequent values represented with fewest bits
- Iteratively:
  - join two lowest frequencies as siblings
  - parent's frequency is the sum of its childrens'
- Left branches represent 0s,
  Right branches represent 1s
- chars are always the "leaves"
  - i.e. they're always at the end of a branch

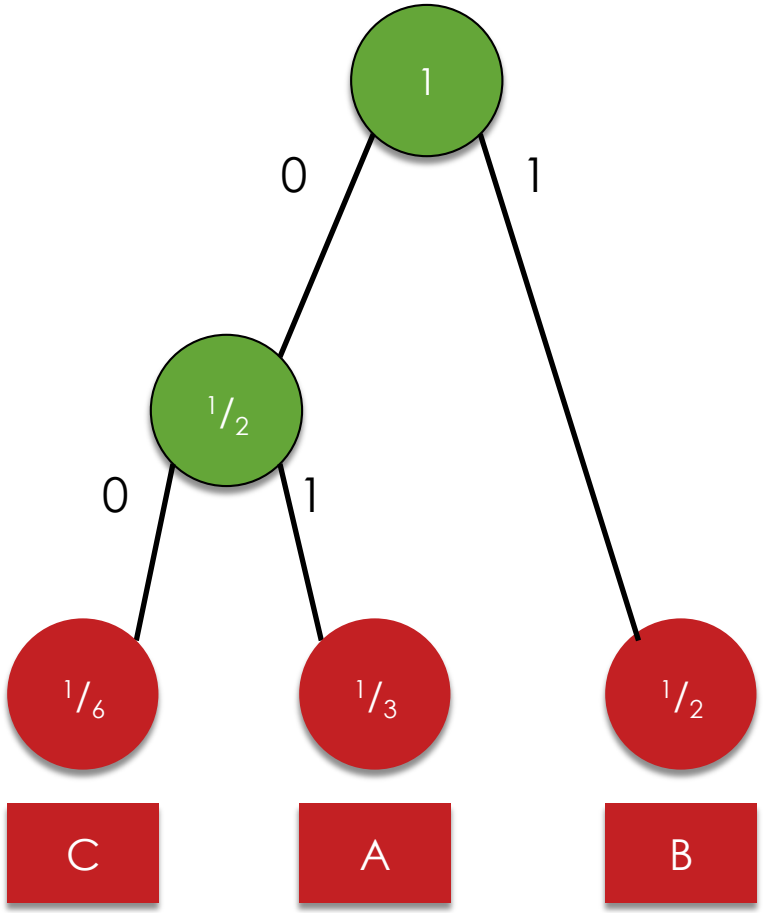
“AAAABBBBBBCC”

char	frequency
A	4
B	6
C	2



“AAAABBBBBBCC”

char	frequency
A	$^{4}/_{12} = {}^{1}/_{3}$
B	$^{6}/_{12} = {}^{1}/_{2}$
C	$^{2}/_{12} = {}^{1}/_{6}$



Huffman nodes

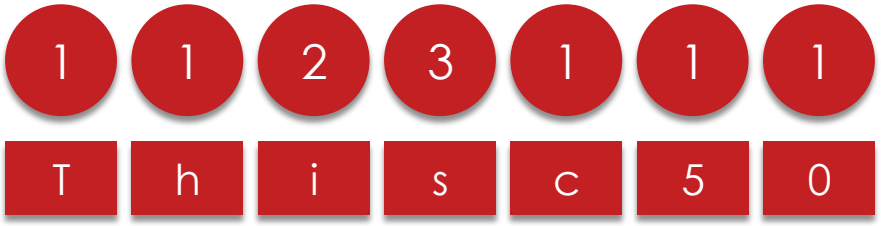
```
typedef struct node
{
    char symbol;
    float frequency;
    struct node* left;
    struct node* right;
}
node;
```

Huffman nodes

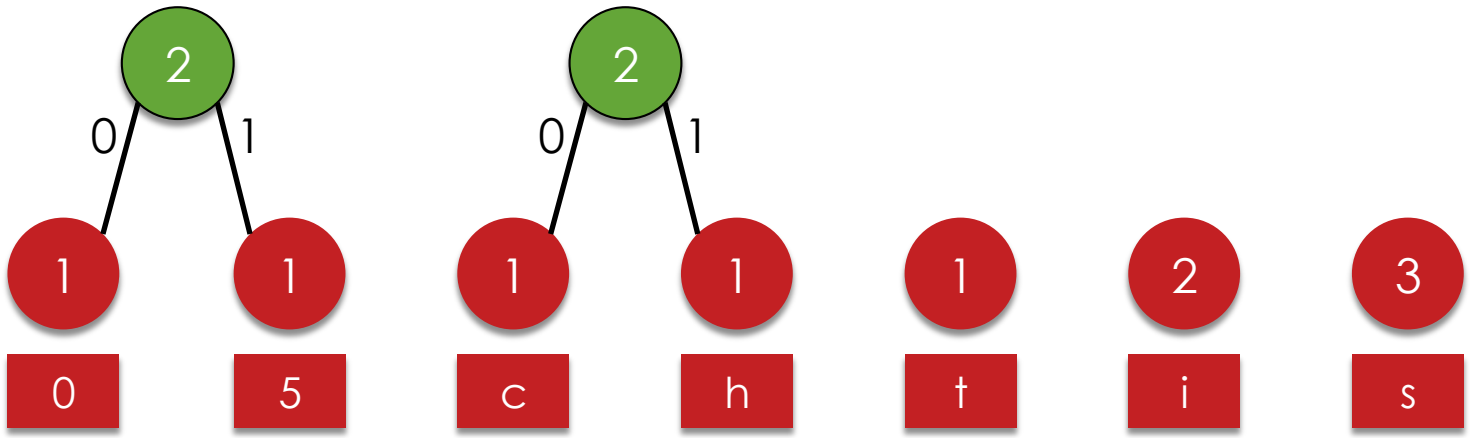
```
typedef struct node
{
    char symbol;
    int frequency;
    struct node* left;
    struct node* right;
}
node;
```

“This is cs50”

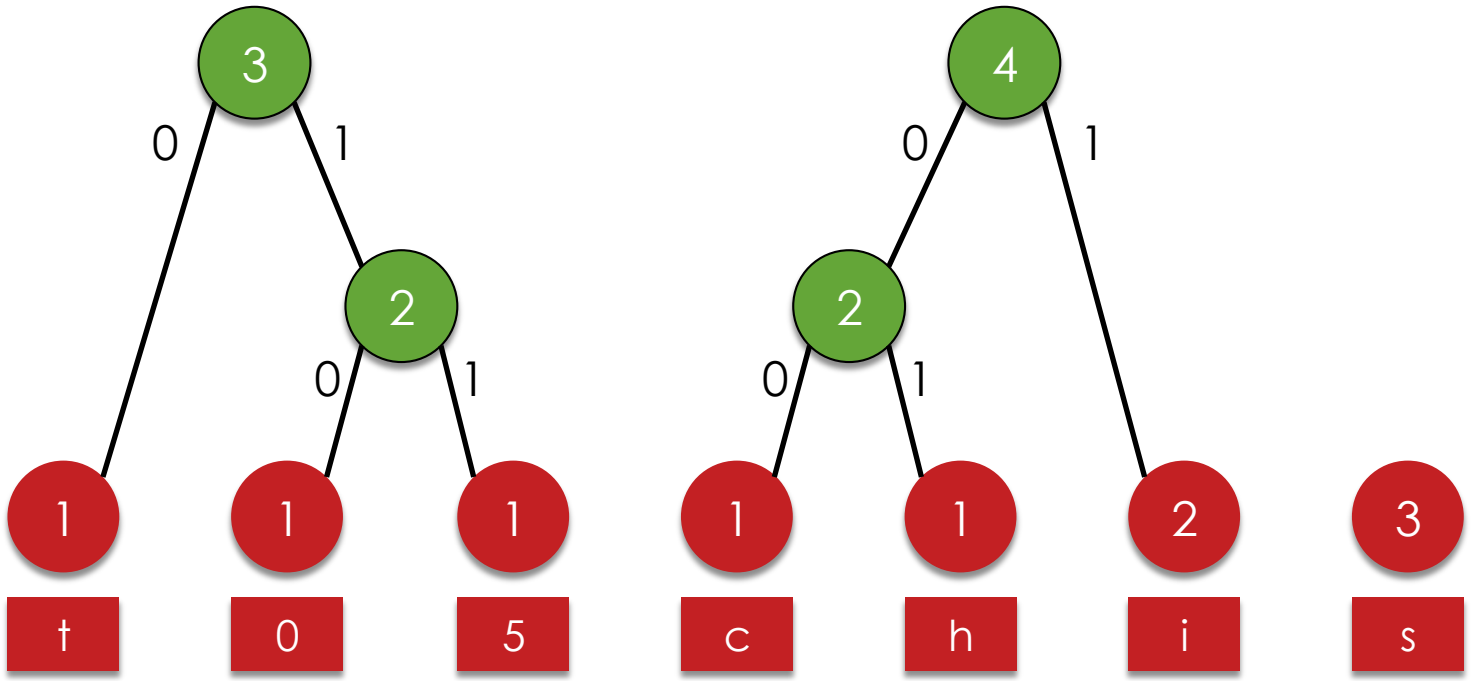
char	frequency
T	1
h	1
i	2
s	3
c	1
5	1
0	1



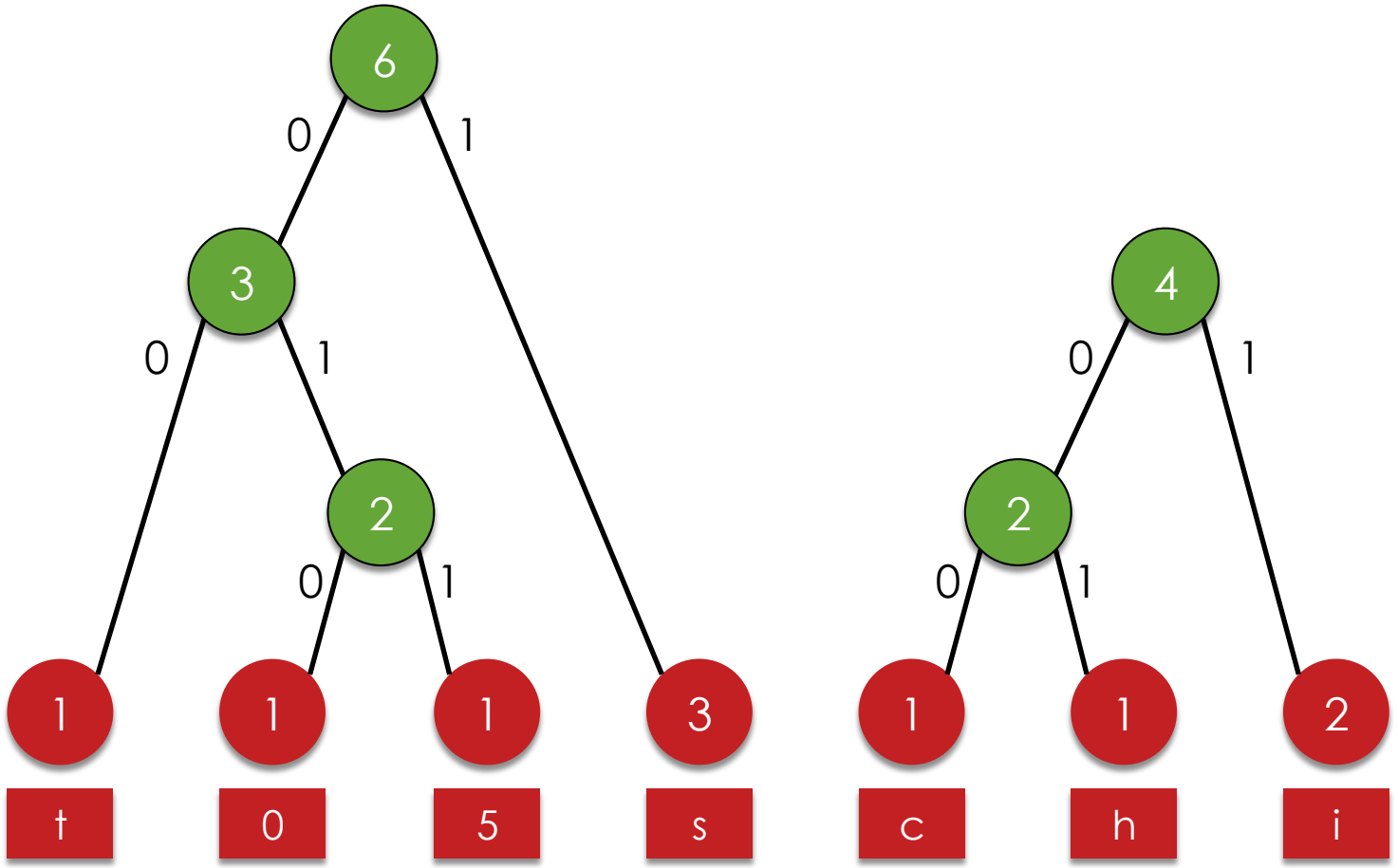
“This is cs50”



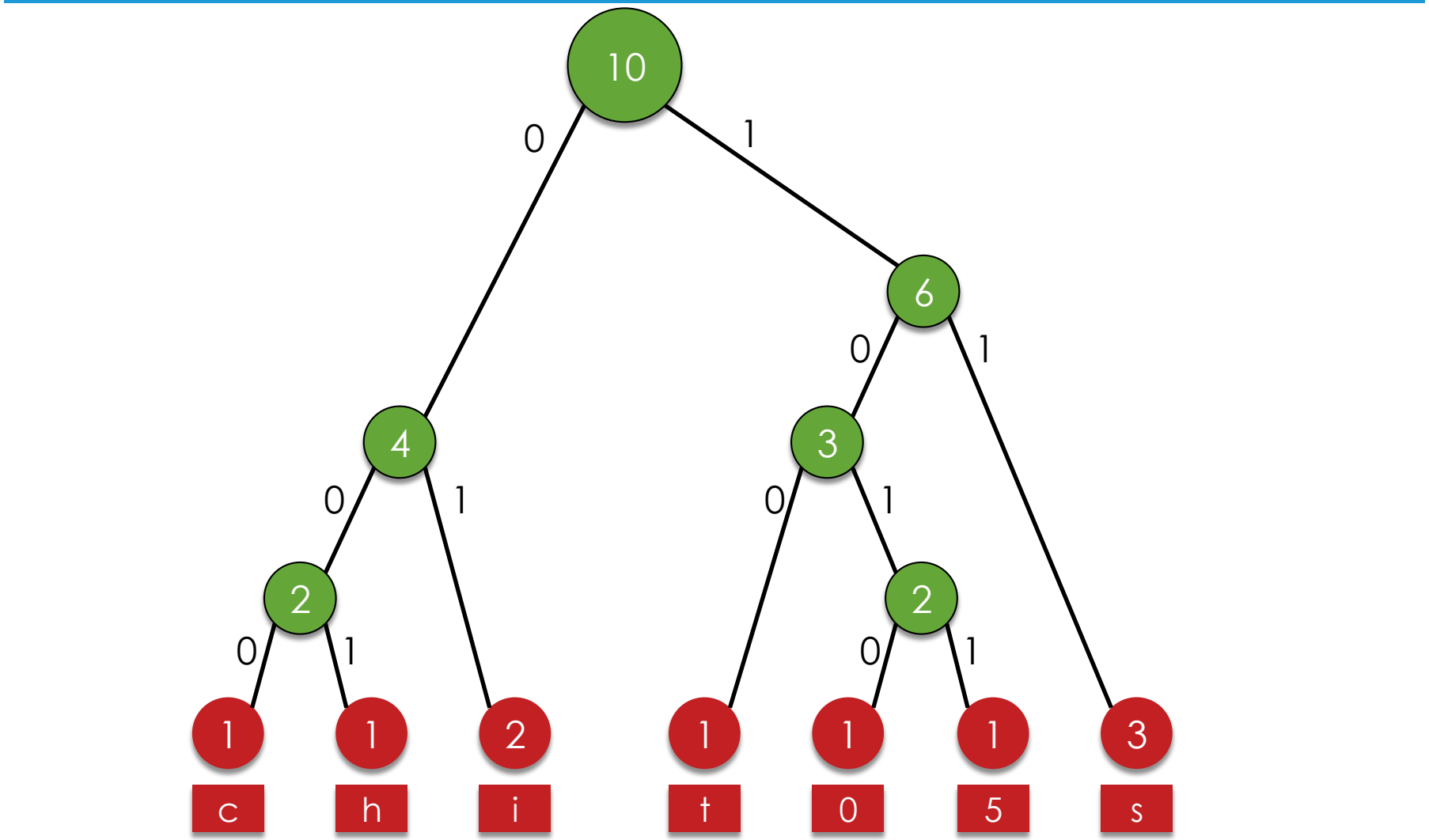
“This is cs50”



“This is cs50”

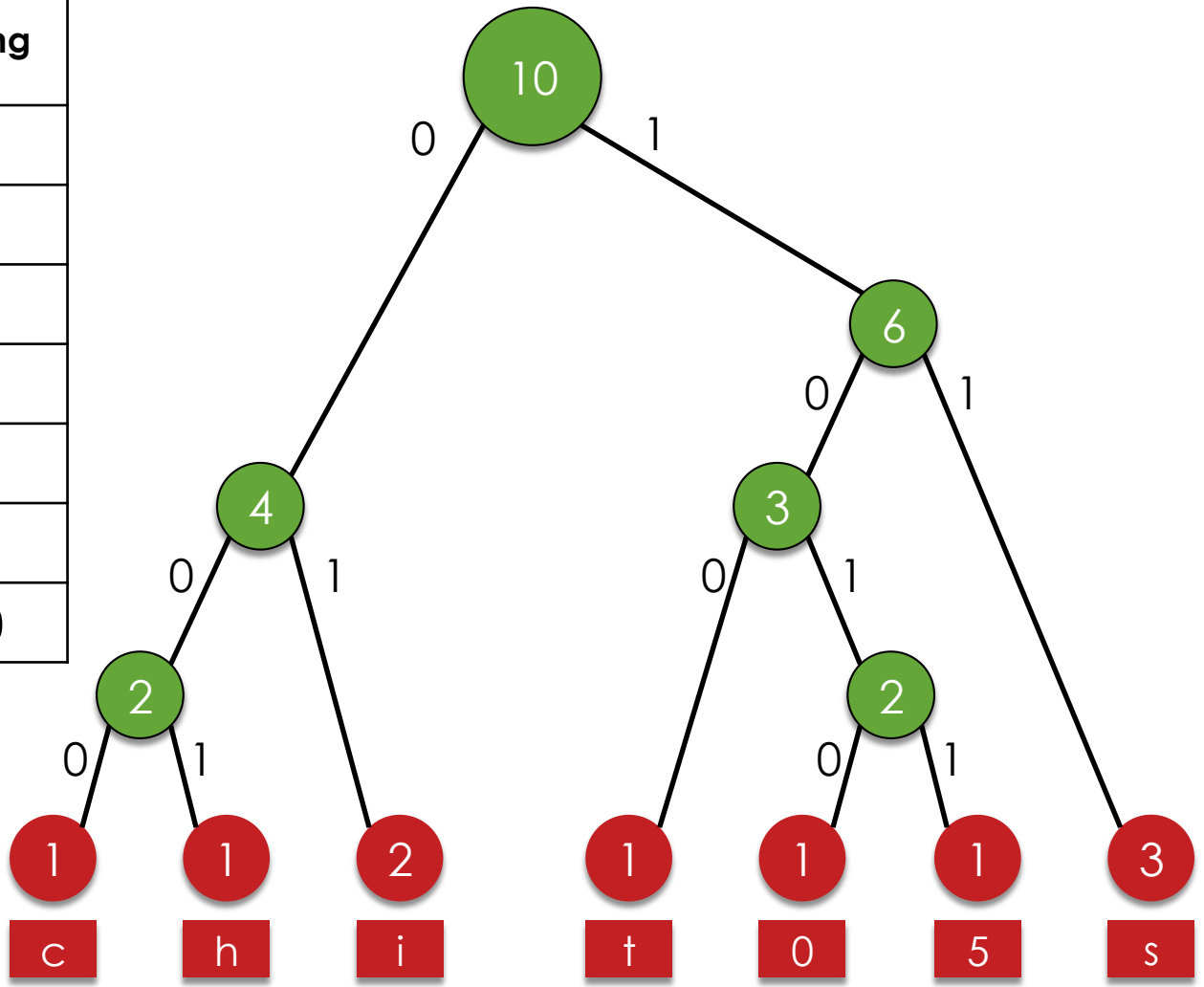


“This is cs50”



“This is cs50”

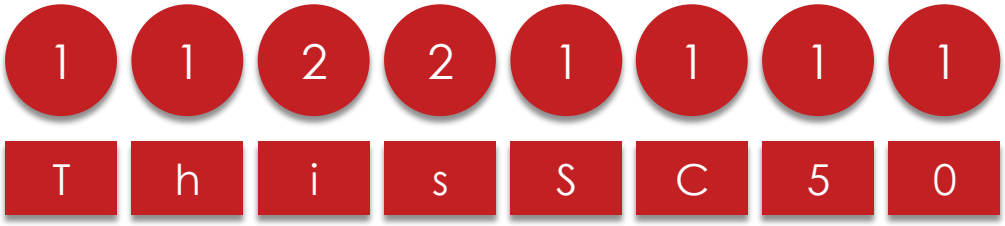
char	freq	encoding
t	1	100
h	1	001
i	2	01
s	3	11
c	1	000
5	1	1011
0	1	1010



“This is CS50”

char	frequency
T	1
h	1
i	2
s	2
S	1
C	1
5	1
0	1

CasE SeNsitiVitY!



Huff'n Puff

0. A Section of Questions
1. Huff'n Puff

https://762f6585f5-custmedia.vresp.com/4ccee98eda/Wolf%20Puppy.jpg

Huff'n Puff

Huff

text  0s and 1s, frequency counts

Puff

0s and 1s  text

Distribution Code

- `huffile.h, huffile.c`
- `tree.h, tree.c`
- `forest.h, forest.c`
- `dump.c`
- `Makefile`

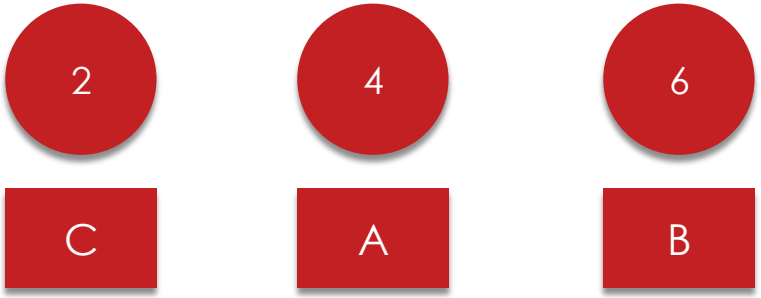
Huffman nodes

```
typedef struct tree
{
    char symbol;
    int frequency;
    struct tree* left;
    struct tree* right;
}
Tree;
```

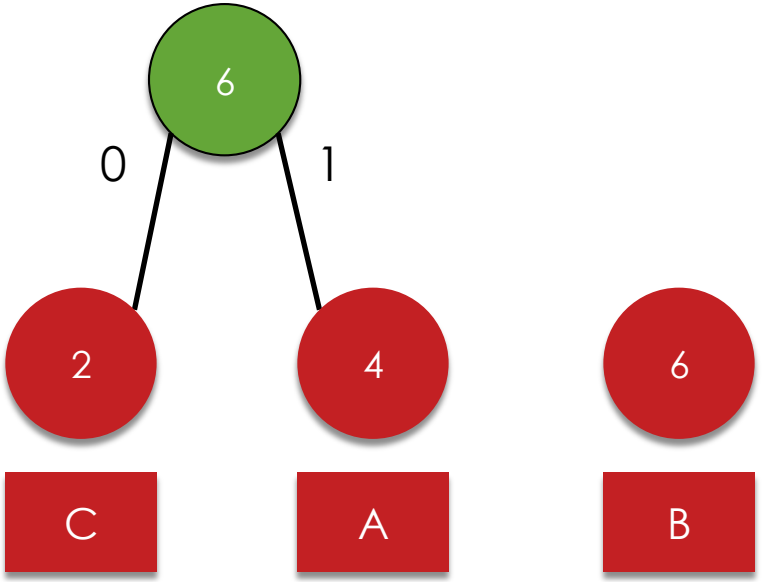
Forest

- A linked list of Trees
- Ultimately, we want our Forest to contain just 1 Tree

← Forest with 3 trees



Forest with 2 trees →



TODO

- Ensure proper usage
- Ensure input is huffed
- Build Huffman's tree
- Write message to file
- Free memory

Usage

- `./puff input output`
- input: name of file to puff
- output: name of file for puffed output

TODO

- ☑ Ensure proper usage
- ☐ Ensure input is huffed
- ☐ Build Huffman's tree
- ☐ Write message to file
- ☐ Free memory

Ensure input is huffed

- Huffeader
  - `huffile.h`
  - `dump.c`

TODO

- ☑ Ensure proper usage
- ☑ Ensure input is huffed
- ☐ Build Huffman's tree
- ☐ Write message to file
- ☐ Free memory

forest.h

- `bool plant(Forest* f, Tree* t)`
  - Plants a (non-0 frequency) tree in the forest
- `Tree* pick(Forest* f)`
  - removes a tree with lowest weight from forest

Build Huffman's tree

- Read in symbols/frequencies in order
- Ignore 0 frequencies
- Plant trees in forest
- Join trees as siblings
- End result: a single `Tree` in the `Forest`

TODO

- ☑ Ensure proper usage
- ☑ Ensure input is huffed
- ☑ Build Huffman's tree
  - Error handling
- ☐ Write message to file
- ☐ Free memory

Error handling

- Check for errors wherever there's chance of failure
- Using `bool` functions in your condition

```
if (foo(x))
```

  - has to evaluate `foo` in order to evaluate condition!

TODO

- ☑ Ensure proper usage
- ☑ Ensure input is huffed
- ☑ Build Huffman's tree
  - ☑ Error handling
- ☐ Write message to file
- ☐ Free memory

Write message to file

- Use `dump.c` as basis (if you haven't been already!)

```
int bit;
while ((bit = bread(input)) != EOF)
{
    …
}
```

Write message to file

- `bit == 0` → go left
- `bit == 1` → go right
- If you're at a leaf, print the symbol
  - How do you know you're at a leaf?

Free memory

- `hfclose`
- `rmtree`
- `rmforest`
- Careful! Not just at the very end of the main!
  - Can your code ever return prematurely?
  - (Yes.)

Tools: valgrind, diff, ls, gdb

- valgrind
  - valgrind ./puff hth.bin puffed.txt
  - valgrind -v --leak-check=full ./puff hth.bin puffed.txt
- diff
  - ~cs50/pset6/huff hth.txt hth.bin
  - ./puff hth.bin puffed.txt
  - diff hth.txt puffed.txt
- ls -l
- gdb

this was walkthrough 6