CSC148H Assignment Intro

March 13, 2017

What is it?

- Assignment 2 is a compression and decompression program
  - You can take any file and compress it
  - You can take one of the compressed files that you made and uncompress it back to the original file
- Huffman's algorithm is used to organize the file compression
- It gives short codes to frequent symbols and longer codes to infrequent symbols

Binary Files and Bytes

- We want to be able to compress and uncompress any kind of file: text, sound, picture, executable, etc.
- We therefore open files in binary mode, so that we can read bytes (not text) from the files
- A byte is an integer in the range 0-255
- We use Python `bytes` objects to work with bytes

```
>>> b1 = bytes([80, 90, 100])
>>> b2 = bytes([5])
>>> list(b1 + b2)
[80, 90, 100, 5]
```

Huffman's Algorithm

Let's run through an example of Huffman's algorithm on this frequency table.

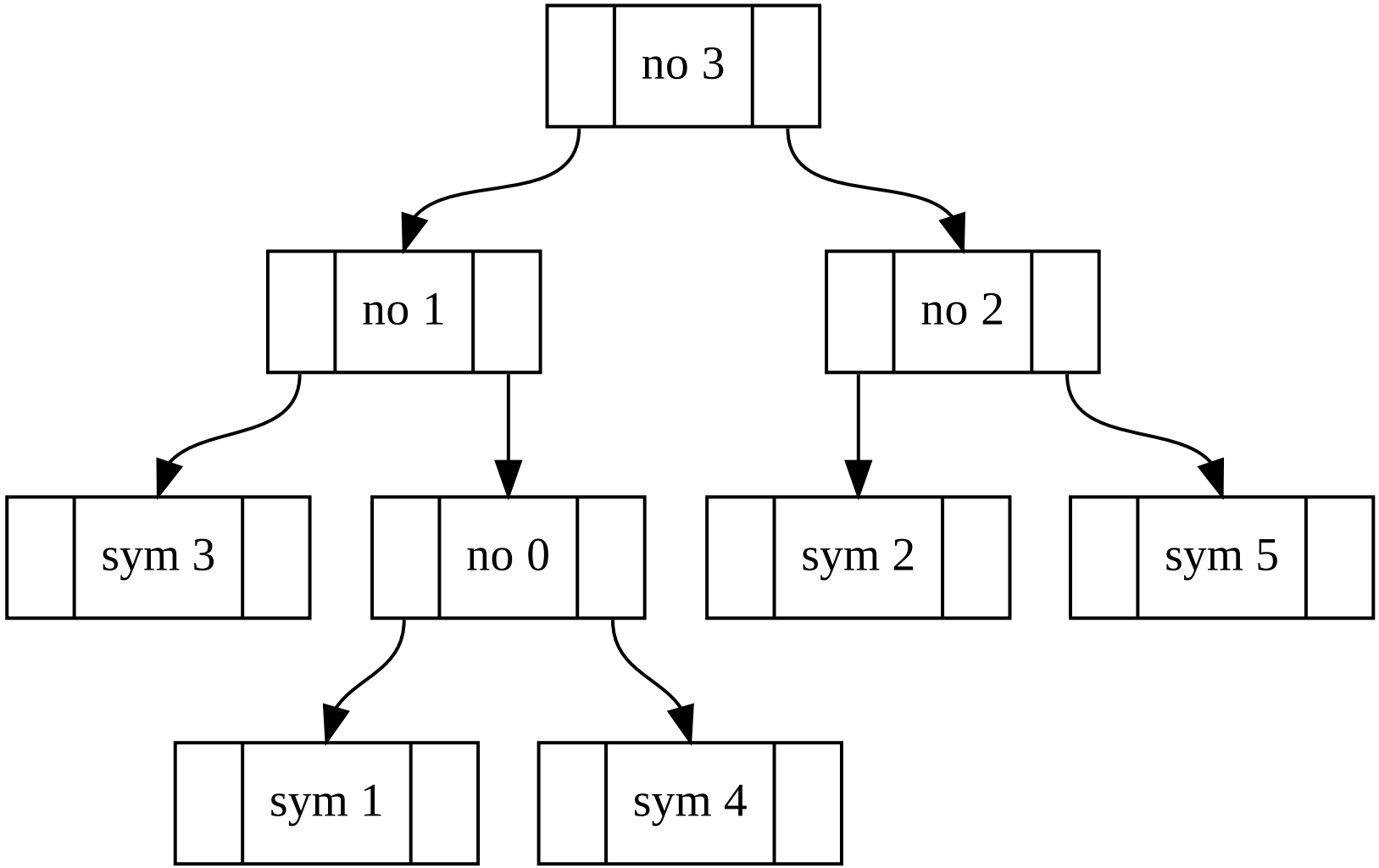
byte	frequency
1	12
2	33
3	15
4	8
5	32

The rule: at each step, merge the two smallest-frequency trees.

Numbering Nodes

- We're going to have to write a representation of a Huffman tree to a compressed file
- To do that, we will want a way to refer to nodes
- But internal nodes have no name, so there is no way to refer to them
- This is why we **number** nodes
- We have decided on a postorder numbering (we could have chosen any numbering for the assignment)

Sample Numbered Huffman Tree



Codes for Symbols

Each symbol gets a code from the tree.

byte	code
1	010
2	10
3	00
4	011
5	11

Use that to compress this list of bytes: `[5, 2, 1, 3]`

Tree to Bytes

- With the tree numbered, we can now output a representation of the tree to a file
- If there are n internal nodes, then we write $4n$ bytes to the file to represent the tree
  - Each internal node takes 4 bytes
  - A 0 or 1 is written to signify a leaf or non-leaf subtree, respectively

Uncompressing

- To uncompress a file, we first have to obtain the Huffman tree that was used to compress it
- Without the tree, we have no idea of the mapping between codes and symbols
- The Huffman tree is stored in the file in postorder
- And we have the node numbers in there to help us reconstruct the tree
- We are asking you for two **different** functions for recovering a tree from a file ...

Recovering Tree in General

`generate_tree_general`

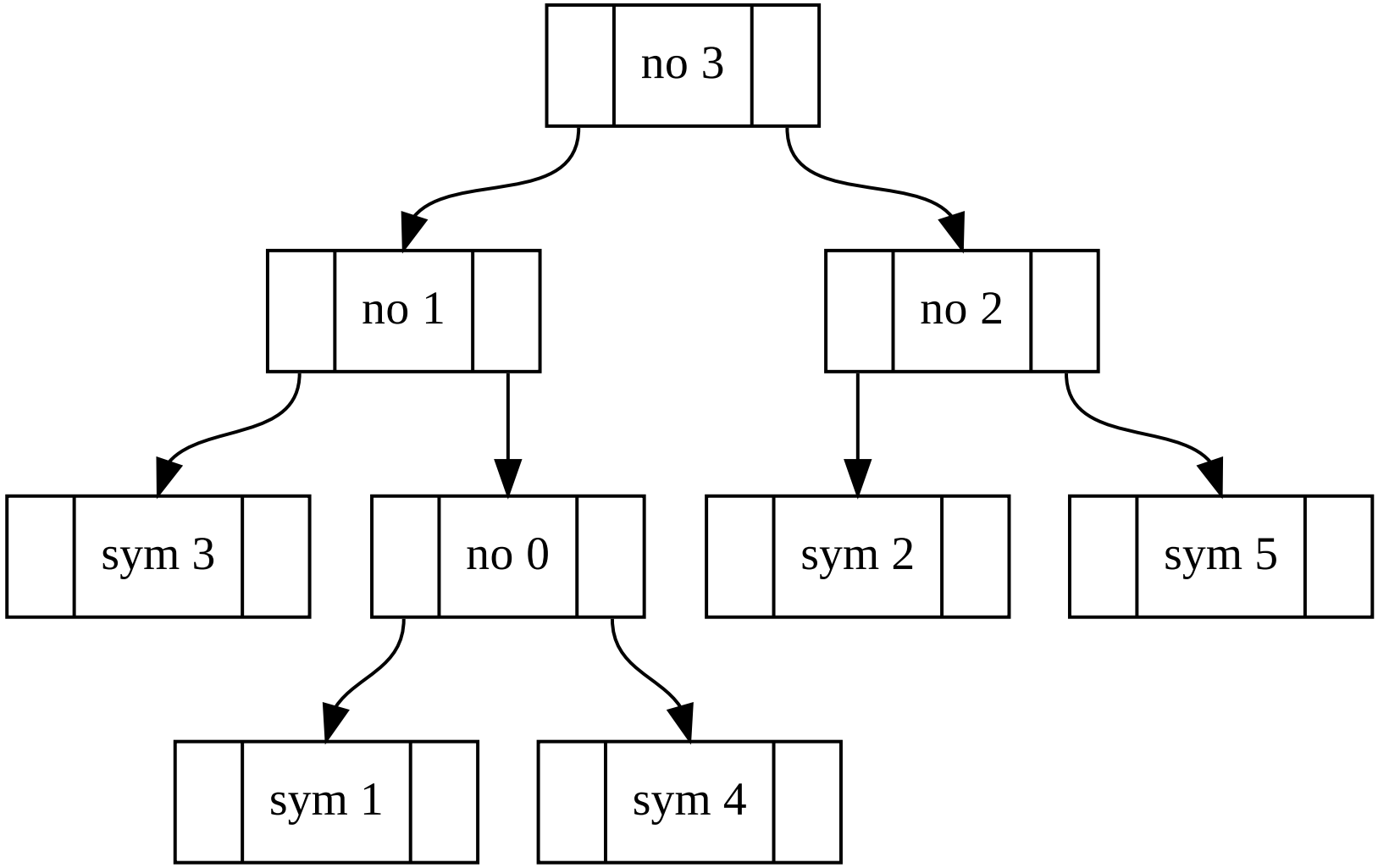
- In here, you must reconstruct a tree **no matter what** order it was written to the file
- It could be in postorder, or preorder, or inorder, or whatever ... you have to get the tree back
- Your assignment always writes trees in postorder, so you'll have to test with your own functions that write trees in other orders
- You're going to have to use the node numbers for sure

Recovering Tree in Postorder

`generate_tree_postorder`

- In this function, you are not allowed to use the node-numbers at all
- But, you are allowed to assume that the tree is written in postorder
- Tip: you know which left/right subtrees are leafs and which are not. This is crucial information!

Uncompressing Using a Tree



Uncompress 110110000