Govind Pillai
gopillai@ucsc.edu
5/13/2021

CSE 13s Spring 2021
# Assignment 6: Huffman Coding
Design Document

## PURPOSE:

The purpose of this lab is to compress and decompress files using Huffman Coding. This program uses a histogram in order to build a priority queue which is used to make a Huffman tree, This Huffman tree will then be used to create codes for different characters in the file based on where they are placed in the tree. The decoder will then use what is called the "tree dump" to reconstruct the tree and use recursion in order to build whatever was in the original file.

## DEFINITIONS:

Histogram: A table the counts the frequency of each symbol that is in the file

Priority Queue: A normal queue that uses insertion sort to have nodes with less frequency at the top. Elements at the top of the queue have higher "priority"

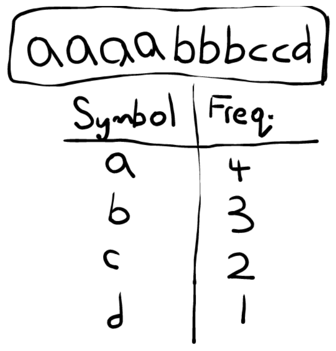
Huffman tree: A tree holding all the nodes from the histogram. It is created by popping two nodes from the priority queue, calling them the "left" and "right" nodes, and created a parent node with a "$" symbol and a frequency which is equal to the sum of the left and right frequencies

Code table: A table which holds each symbol and they code created for each symbol

Tree dump: An encoded version of the tree which is used by the decoder to recreate the tree when decoding the file itself

## DIAGRAMS:

### Histogram:



aaaabbbccd

| Symbol | Freq: |
|---|---|
| a | 4 |
| b | 3 |
| c | 2 |
| d | 1 |

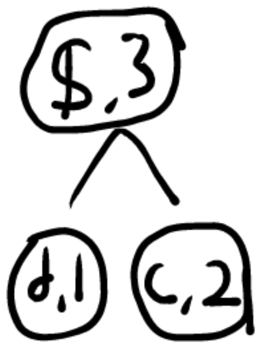
The table counts the frequency of each symbol. 'a' comes up 4 times, 'b' 3 times, etc.

Priority Queue:

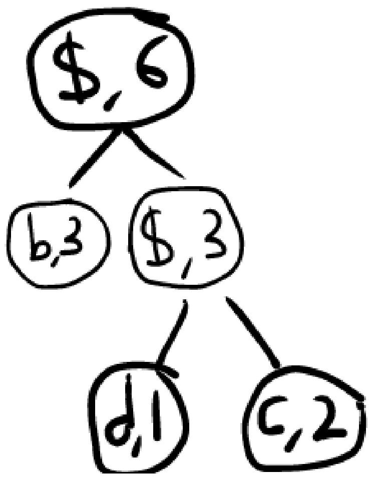
Queue: {(d, 1), (c, 2), (b, 3), (a, 4)}

This queue holds each symbol along with their frequency. As shown in the diagram, elements with lower frequency have higher priority and are near the top of the queue.

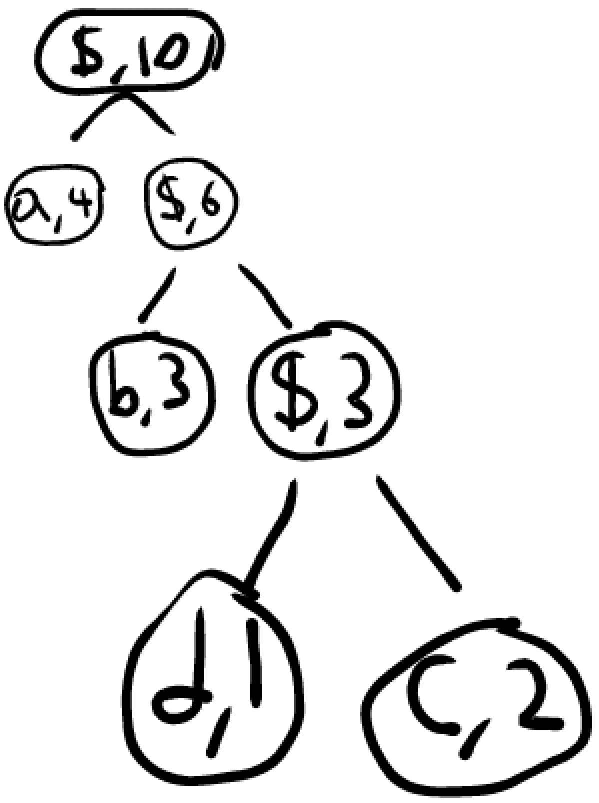
Huffman Tree:



Queue: {(b,3),($,3),(a,4)}

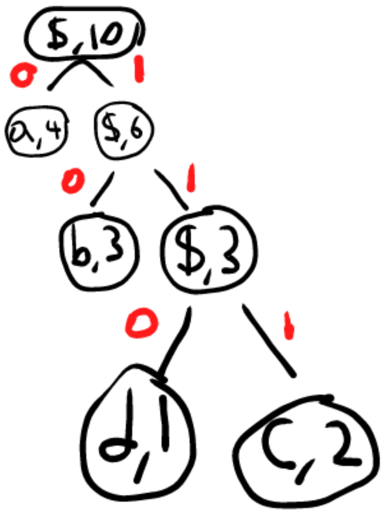


Queue: {(a,4),($,6)}



These diagrams show how to build the huffman tree. First, you pop off two items from the priority queue. These are the “left” and “right nodes respectively. The parent of these nodes will have a frequency that is equal to the sum of its direct children’s frequencies. This new parent node is then pushed onto the priority queue. This process is continued until there is only one item left on the priority queue.

Code table:



| Symbol | Code |
|---|---|
| a | 0 |
| b | 10 |
| c | 111 |
| d | 110 |

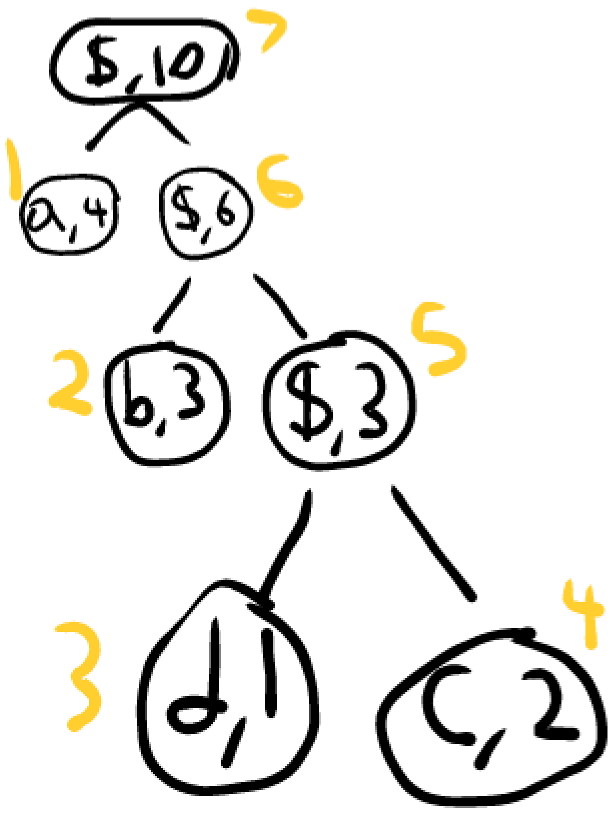
This table includes the codes for each table. This is created by using the huffman tree. When looking through the tree for a symbol, you add “0” to the code if you move down to the left and “1” to the code if you move down to the right. Because ‘a’ is just one move to the left, its code is just “0”. This is used for all other symbol elements in the tree.

## Tree Dump:

When traversing through a tree, you use a recursive function:

```
traverse(n):
    If n is not null:
        left(n);
        right(n);
        do_something(n);
```

If we use this function to go through our example tree, here is the order of the elements we visit:



We then use this order to create our tree encoding. If there is a leaf node(a node without any children), we denote it with an L. If it is not a leaf, we denote it with an I. Here is our tree dump:

LaLbLdLcIII

This code is then used to re-create the tree.