Exam 1 – Sample Answers

October 26, 2015

Variables whose domain is not specified explicitly range over the reals. So for example, if a function is defined as $f(x) = x^2$, then $\operatorname{dom} f = \operatorname{cod} f = \mathbb{R}$.

Discrete math

Sets

True or false:

- If $x \in A$ then $x \in A \cup B$ $\boxed{\textit{T by def. of union}}$
- The empty set has no subsets. $\boxed{\textit{F, because } \varnothing \subseteq \varnothing}$
- The empty set is a proper subset of every set. $\boxed{\textit{F, no set is proper subset of itself}}$
- $5 \in \{2x + 1 : x \in \mathbb{N}\}$ $\boxed{\textit{T with } x = 2}$
- The power set of the set $\{1, 2, 3, 4, 5\}$ has 25 elements. $\boxed{\textit{F, it has } 2^5 = 32}$

Relations

Which of these relations are reflexive? Symmetric? Anti-symmetric? Transitive?

- $\{(x,y) : x < 2y\}$

  - *not reflexive, counterexample* $-1 \not< 2(-1)$
  - *not symmetric, counterexample* $2 < 2(5)$ *but* $5 \not< 2(2)$
  - *not transitive, counter example* $5 < 2(3)$ *and* $3 < 2(2)$ *but* $5 \not< 2(2)$

- $\{(x,y) : x, y \in \mathbb{N}, 2x \leq y\}$

  - *not reflexive, counterexample* $2(1) \not\leq 1$
  - *not symmetric, counterexample* $2(2) \leq 5$ *but* $2(5) \not\leq 2$
  - *transitive; suppose* $2x \leq y$ *and* $2y \leq z$. *Then* $4x \leq 2y$. *Note that for non-negative numbers (and* $\mathbb{N}$ *does not include negatives),* $2x \leq 4x$. *Thus by transitivity of* $\leq$ *twice,* $2x \leq 4x \leq 2y \leq z$.

- For $x, y \in \text{SetOfAllHumans}$, $x\,R\,y$ iff x and y have the same birthday.

  All three. Everyone who shares the same birthday is R-related to everyone else who shares that birthday and no one else.

- $\{(x,y) : x, y \in \mathbb{Z}, x^2 = y^2\}$

  All three for the same reason.

Functions

Which of these relations are functions?

- $\{(x,y) : x^2 = y\}$

  Is a function, because there is only one y *for each* x, *its square.*

- $\{(x,y) : x^2 = y^2\}$

  Is not a function; counter example: both $(2,2)$ *and* $(2,-2)$ *are in the relation.*

- $\{(x,y) : x = y^2\}$

  Is not a function; counter example: both $(4,2)$ *and* $(4,-2)$ *are in the relation.*

Which of these functions are injective? Surjective? Bijective?

- $f(x) = x/2$

  Bijective, with inverse $f^{-1}(y) = 2y$

- $f : \mathbb{Z} \to \mathbb{Z}$ where $f(x) = 2x$.

  Injective, because each integer is twice as much as only one other integer; not surjective because odd numbers are not in the image.

- $f : [0, 1] \to [0, 1]$ where $f(x) = x^2$.

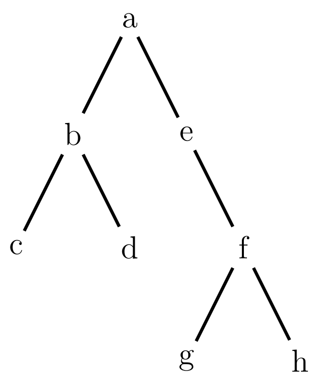
  Bijective, with inverse $f^{-1}(y) = \sqrt{y}$. However, if the codomain were made wider without increasing the domain to compensate, then it would no longer be surjective. And if the domain included negative numbers then it would no longer be injective.

Huffman coding

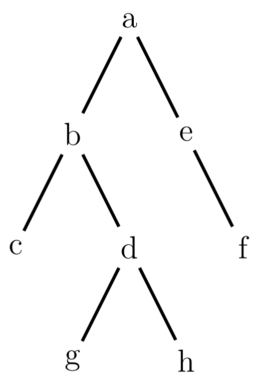
Trees

Consider these three binary trees:

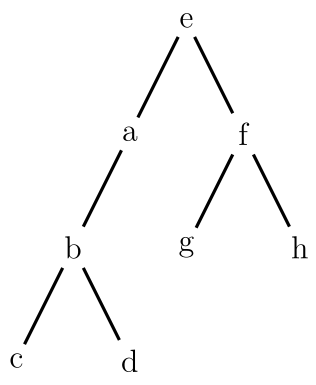
$T =$



$U =$



$V =$



Which pairs of them have the same:

- pre-order traversal? *none*
- in-order traversal? *T and V (compare below)*
- post-order traversal? *none*
- level traversal? *T and U*

Write out the letters in the order they would be visited by each kind of traversal.

	T	U	V
pre-order	abcdefgh	abcdghef	eabcdfgh
in-order	cbdaegfh	cbgdhaef	cbdaegfh
post-order	cdbghfea	cghdbfea	cdbaghfe
level	abecdfgh	abecdfgh	eafbghcd

True or false:

- A binary tree node has exactly two children. $\boxed{\textit{F, could have 0, 1, or 2}}$
- A binary tree node has at most two children. $\boxed{T}$
- A binary tree has edges labeled with 0s and 1s. $\boxed{\textit{F, tree edges needn't have labels}}$

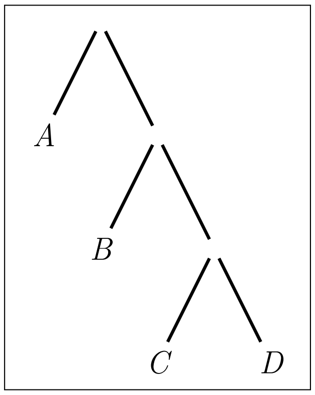
Prefix codes

Suppose we want to encode the symbols $\{A, B, C, D\}$ in binary. Is this a valid prefix code?

$$\begin{aligned} A &\mapsto 0 \\ B &\mapsto 10 \\ C &\mapsto 110 \\ D &\mapsto 111 \end{aligned}$$

$\boxed{\textit{Yes, because we can buld a prefix code tree (below)}}$

If it is, build the prefix code tree; if not, explain why you can't.



Encode this message: “BADCAB” *100111110010*

Decode this message: “11101110” *DADA*

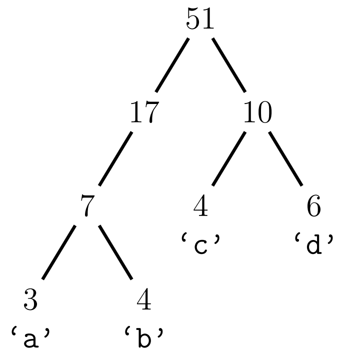
True or false:

- A prefix code is a code in which every code word is a prefix of some other code word. *F, the opposite is true*
- Every code word in a prefix code must have the same length. *F, the point of a prefix code is to allow variable length*
- If all the code words in a prefix code are reversed, the result is still a prefix code. *F, code above is counterexample*

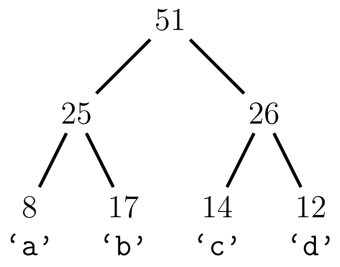
Huffman trees

Consider these two trees:

$H =$



$I =$

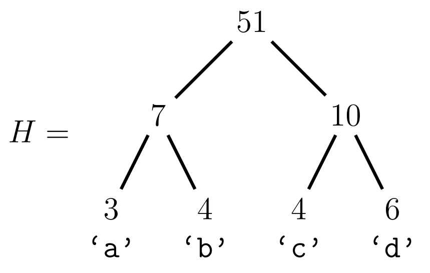


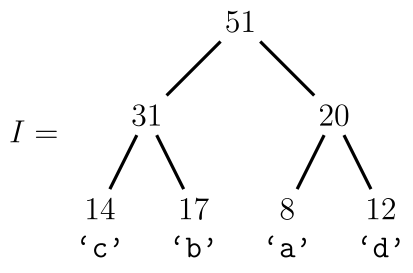
Which of these are valid Huffman trees? *neither*

For those that aren’t, what’s the problem?

H is not, because the node with weight 17 has only one child and the weights are wrong. I is not because the nodes with weights 8 and 12 should have been combined first.

Fix them.

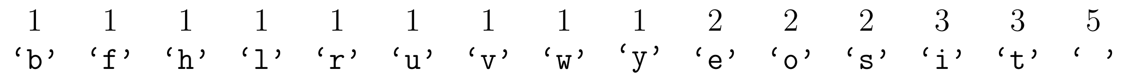




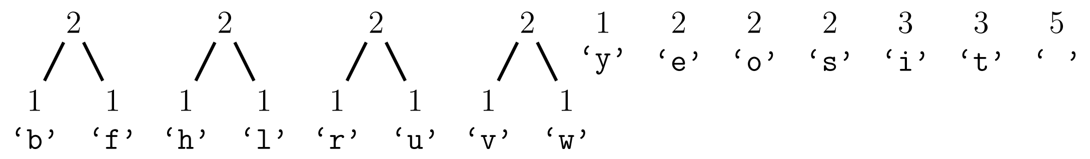
Huffman code this message: “`brevity is the soul of wit`”. Construct a Huffman tree based on the frequencies, and include the tree in your answer.

Frequences: b:1 f:1 h:1 l:1 r:1 u:1 v:1 w:1 y:1 e:2 o:2 s:2 i:3 t:3 ’ ’:5

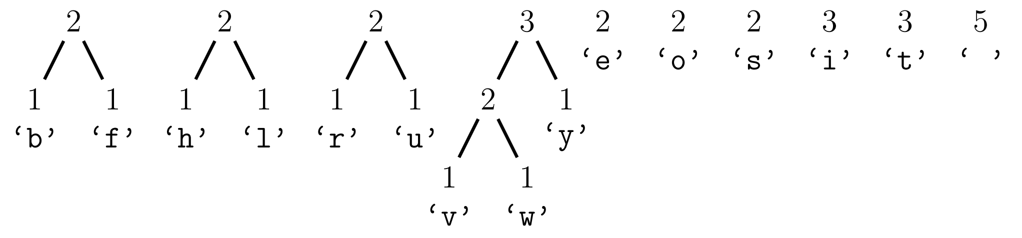
Tree construction:



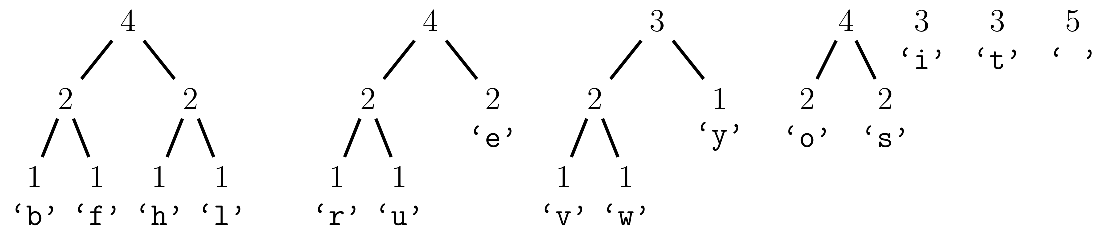
There are nine trees of weight 1, so after four steps eight of them are paired:



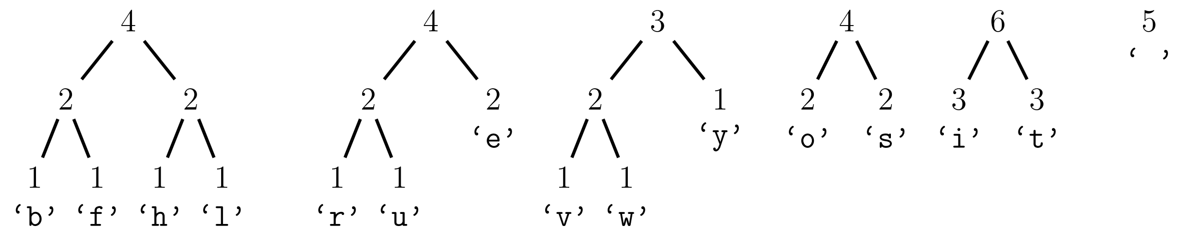
Now combine the 1 with a 2:



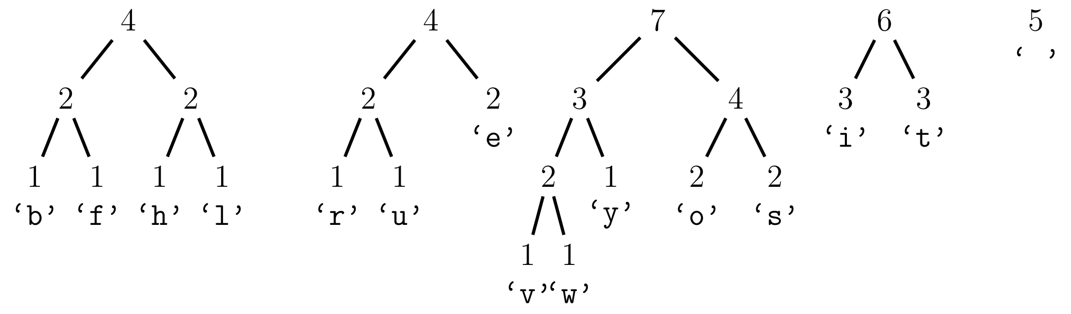
There are now six trees of weight 2, so after three steps we’ve paired all of them:



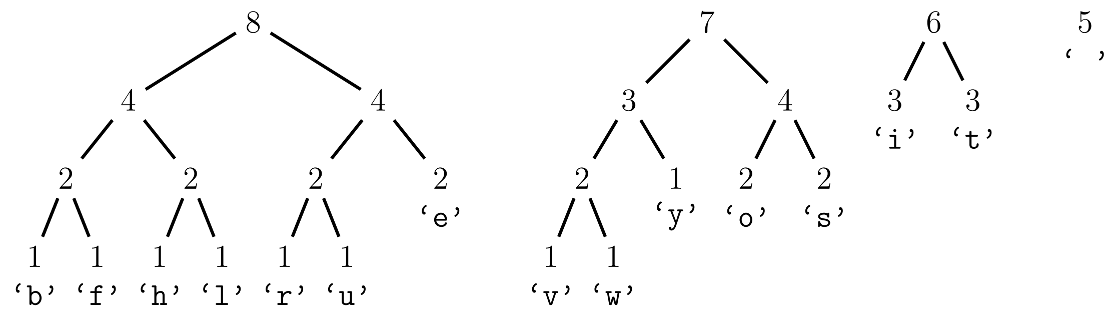
Now combine two of the 3s:



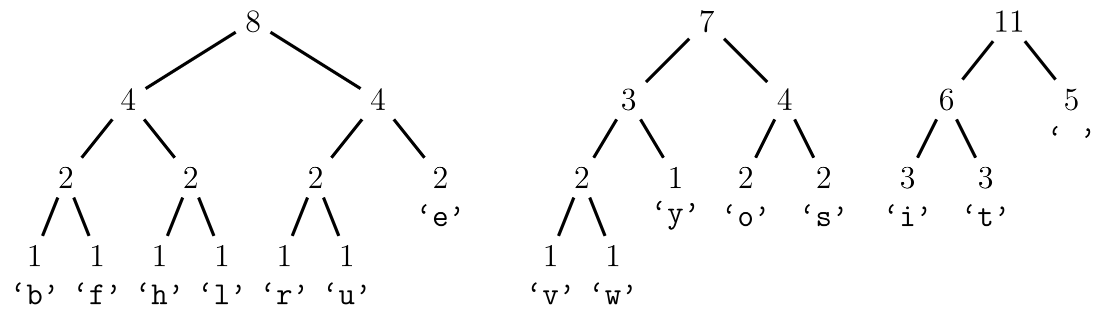
Now the 3 with a 4:


4 2 2 1 1 1 1 ‘b’ ‘f’ ‘h’ ‘l’
4 2 2 1 1 ‘r’ ‘u’ ‘e’
7 3 4 2 1 2 2 1 1 ‘v’ ‘w’ ‘y’ ‘o’ ‘s’
6 3 3 ‘i’ ‘t’
5 ‘ ’

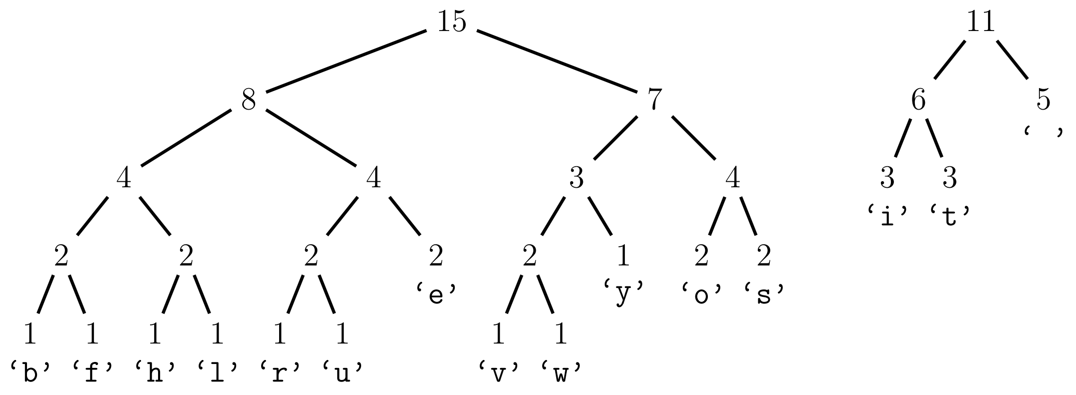
Now combine the remaining 4s:


8 4 4 2 2 2 2 1 1 1 1 1 1 ‘b’ ‘f’ ‘h’ ‘l’ ‘r’ ‘u’ ‘e’
7 3 4 2 1 2 2 1 1 ‘v’ ‘w’ ‘y’ ‘o’ ‘s’
6 3 3 ‘i’ ‘t’
5 ‘ ’

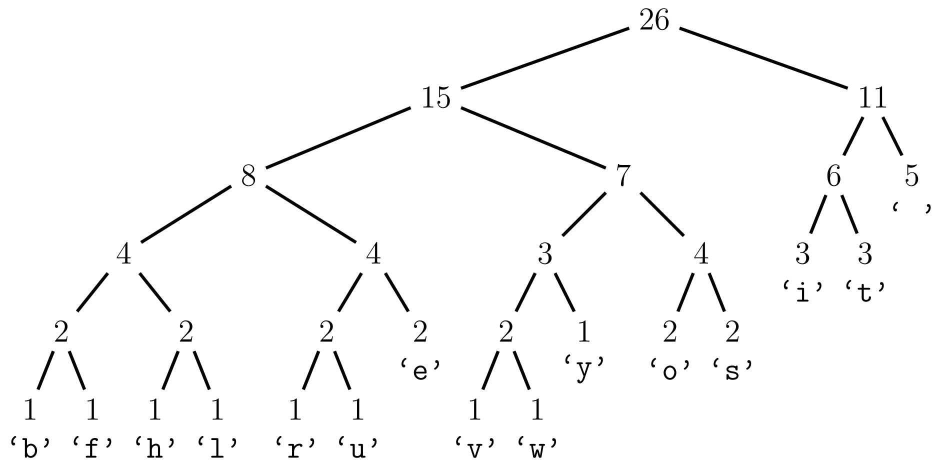
Now combine the 6 with the 5:


8 4 4 2 2 2 2 1 1 1 1 1 1 ‘b’ ‘f’ ‘h’ ‘l’ ‘r’ ‘u’ ‘e’
7 3 4 2 1 2 2 1 1 ‘v’ ‘w’ ‘y’ ‘o’ ‘s’
11 6 5 3 3 ‘i’ ‘t’ ‘ ’

Now 8 with 7:



One last combining step to get the Huffman tree:



Now we can encode the message: `brevity is the soul of wit`

With spaces between codewords for clarity: 00000 00100 0011 01000 100 101 0101 11 100 0111 11 101 00010 0011 11 0111 0110 00101 00011 11 0110 00001 11 01001 100 101

Arrays and linked lists

True or false:

- The time to get an element of an array, given its index, does not depend on the size of the array. $\boxed{\textit{T, constant-time indexing}}$
- One advantage of arrays over linked lists is that their capacity can grow in place without having to reallocate or copy elements. $\boxed{\textit{F, they can't do that}}$

- Unlike lists, the size of an array must be known at compile time. $\boxed{\textit{F, size needs to be known when array is created}}$
- The time to get an element of a linked list, given its ordinal position in the list, does not depend on the position of the element. $\boxed{\textit{F, need to follow links}}$
- An array can use less space than a linked list because it does not need extra space for pointers. $\boxed{\textit{T, array elements are packed adjacently}}$

A stack is a kind of data structure that allows adding and removing elements, and keeps track of the order that they are added so that they are removed in the reverse order. It has two main operations:

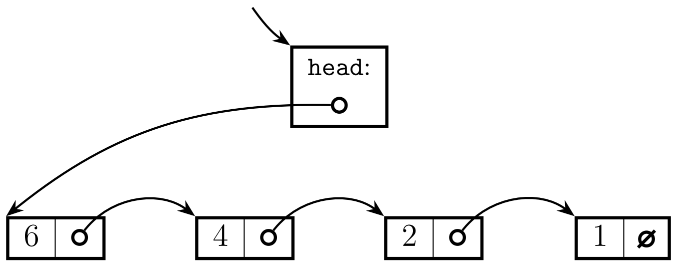
- The *push* operation adds an element to the queue.
- The *pop* operation the most recently *push*ed element that has not yet been popped.

For example, here is a sequence of interactions with a stack:

```
s = new Stack();        // s is empty

s.push(1);              // s is 1
s.push(2);              // s is 2 1
s.push(3);              // s is 3 2 1

a = s.pop();            // s is 2 1, a is 3

s.push(4);              // s is 4 2 1
s.push(5);              // s is 5 4 2 1

b = s.pop();            // s is 4 2 1, b is 5

s.push(6);              // s is 6 4 2 1
```

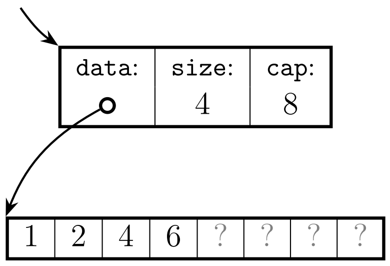
Now consider these two representations of a stack:

Linked list representation. The elements are stored in newest-to-oldest order in a singly-linked list. A one-element struct with one field, `head`, stores a pointer to the head of the list, in order to allow updating the head pointer. For example, here is how the stack `s` from the sample interaction would be represented:



The next element to be popped would be 6, followed by 4, 2, and then 1.

Array representation. The elements are stored in oldest-to-newest order in an array, along with two additional pieces of information: the current number of elements in the stack (labeled `size`) and the capacity of the array (labeled `cap`). For example, here is stack `s` in the array representation:



As in the other representation, the next element to be popped would be 6, followed by 4, 2, and then 1. The light gray question marks (?) indicate locations in the array whose values don't matter, either because those values have been popped or because those locations haven't been used yet.

Questions:

- What are some advantages and disadvantages of each representation?

  (Answer omitted; properties to consider: space, time for operations, flexibility)

- In your preferred language (or clear pseudocode), implement these operations:
  - *push* for the linked list representation
  - *pop* for the array representation

  If the stack is empty then *pop* should signal an error. (If you don't know how, writing a comment that says "`signal empty stack error here`" is sufficient.) You may assume that each structure or class has a constructor that takes all the fields and stores them in the new object. Clearly state any other assumptions that you make.

  Using C++, assuming classes named ***`liststack`*** *and* ***`arraystack`****, and assuming that element type is* ***`int`****.*

```
void liststack::push(int element)
{
    node *new_head = new node(element, this->head);
    this->head = new_head;
}

int arraystack::pop()
{
    if (this->size == 0) throw empty_stack_exception;
    return this->data[--this->size];
}
```