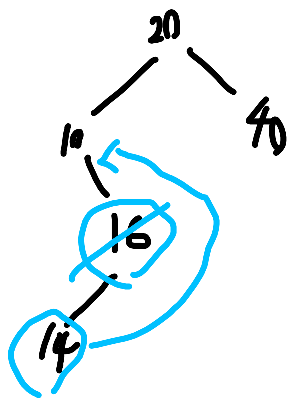
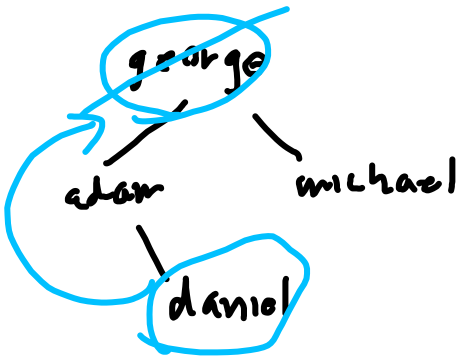
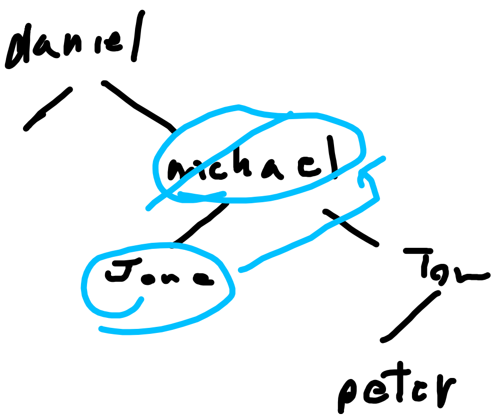
deleting nodes from BST

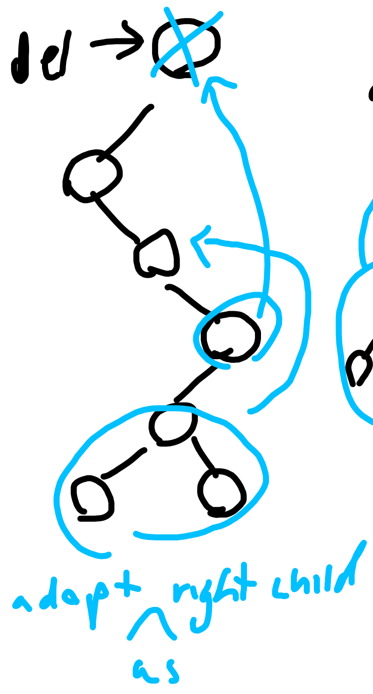


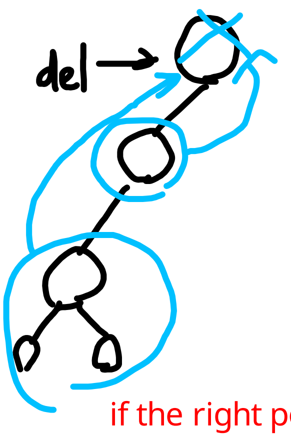




general case

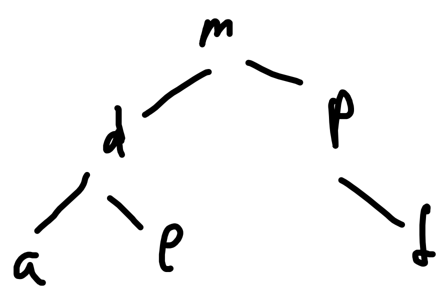
special case





if the right pointer of
the parent of rightmos
is the rightmost, (direc
connected), then want
to adopt the rightmos
node as a left child
(special case, instead
of as right child)

BST path function



$\text{path}(e) = \{m, d, e\}$

returns a pointer to a vector of pointers to nodes
use pointers (dynamically allocated array) so that you dont have to call
the copy constructor

# Huffman coding tree

s1
mississippi
s2

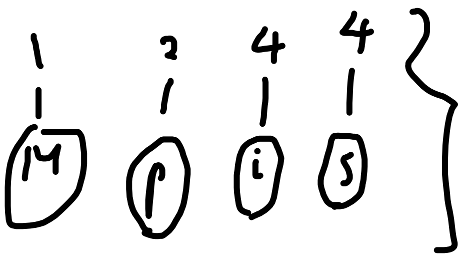
1. want to minimize bandwidth when sending messages b/w stations

2. try to send the word "mississipi" --> 11 char * 8 bits/char = 88 bits
can we minimize this?

3. want to generate var-length encoding scheme: more often occuring chars, are shorter strings to represent, but less often, use longer string

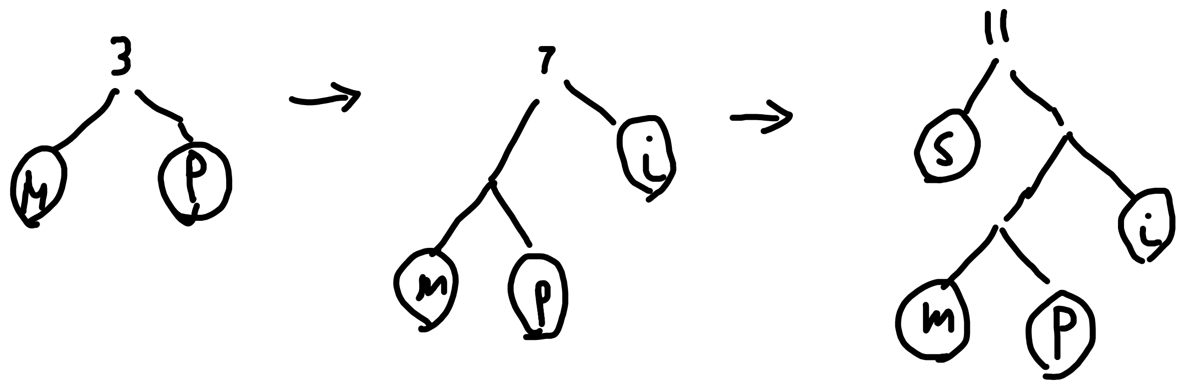
Mississippi

| | |
|---|---|
| M | - x1 |
| i | - x4 |
| s | - x4 |
| p | - x2 |



trees for each char

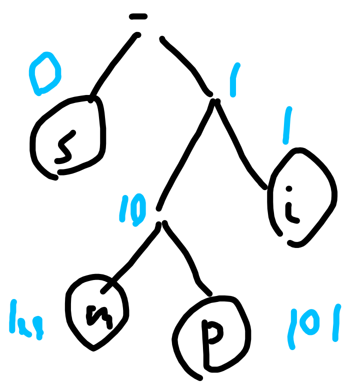
4. huffman coding tree allows to assign encoding to every char in a string
merge the trees
1. merge the smallest 2 trees first



2. merge the 2 smallest trees continually until all trees are merged, left and right tree do not matter when merging
3. put the weights of the tree at the root
4. frequently occurring chars are closer to the root of the tree

5. encoding

1. start from the root, if you go left, append a 0, if you go right, append a 1



num occurrences of char in string * num of bits

| | | | |
|---|---|---|---|
| s | 0 | → | 4×1 = 4 |
| m | 100 | → | 1×3 = 3 |
| p | 101 | → | 2×3 = 6 |
| i | 11 | → | 4×2 = 8 |

only need 21 bits to represen
the entire string!

100110011001110110111

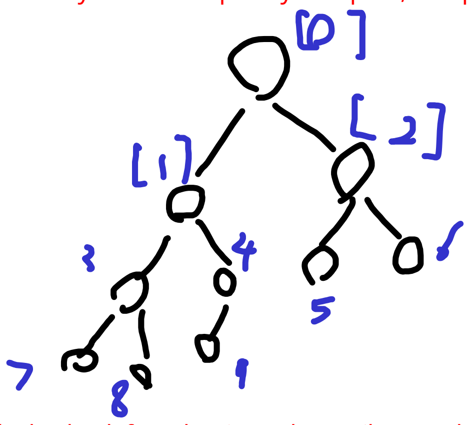
$\xrightarrow{}$

mi

self terminating! don't need delimiters

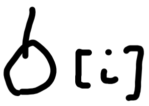
complete binary tree, heap, max heap

binary tree = always has root element w/ right child and left child
complete binary tree = every level is completely occupied, except for the last, all elements flushed to left on the last level



want all elements flushed to left so that it can be easily stored in array

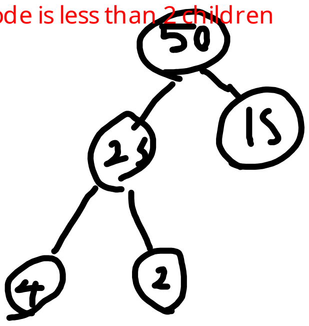
if have some node index i



parent = at index [(i-1) / 2]

left child = at index [2i+1]
right child = at index[2i+2]

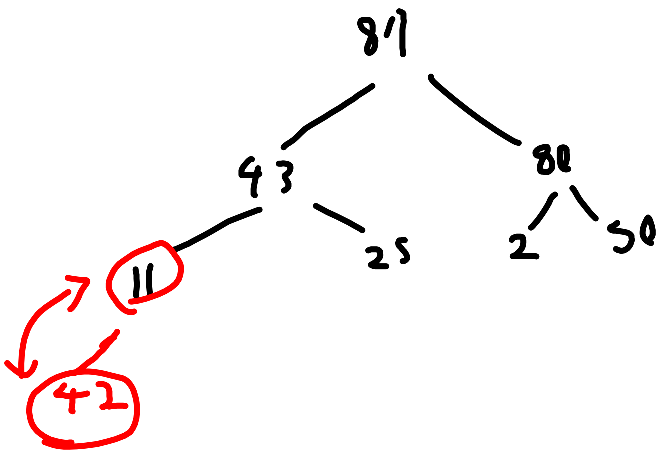
heap = complete binary tree + either max or min (if no specify, assume max)
max heap = each node is greater than its 2 children
min heap = every node is less than 2 children



max heap

inserting into max heap

ADD 42



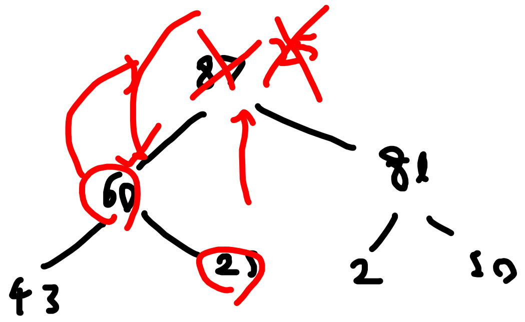
swimming: need to continually swapping with your parent (know what the parents index is) until you are less than your parent

time complexity: O(logn) --> based on height of the tree

building the heap: O(nlogn)

removing from heap

1. when removing the root, replace the root with the last element (rightmost element on last level), and it will sink until it settles



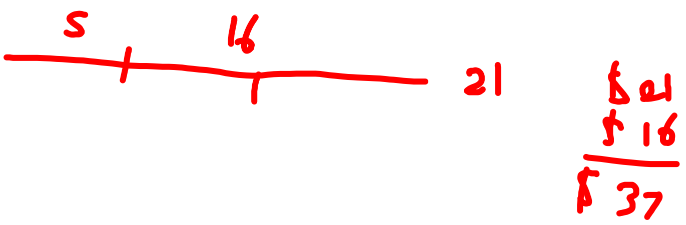
keep swapping with elements larger than yourself until you settle with a parent larger than you
only ever remove the root of a heap!! like a stack

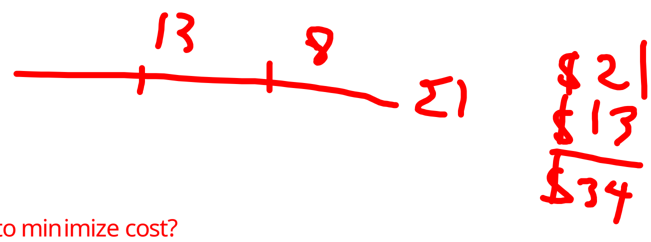
using the heap as a huffman coding tree

1. want to have a heap of pointers to all the trees of chars, so its a tree of trees
1.5. to create a small tree, only care about the element and weight vars of the tree class
2. want to compare the trees so you can always pull out the 2 smallest trees, merge, and put back into heap
3. bc starting with a max heap, just change the meaning of "compare to," change meaning of <,
if less than, return +1, if greater than, return -1 (behaves opposite of max heap, good!)
4. invoking remove x2 gives the 2 smallest trees in the heap
5. major performance improvement compared to sorting and merging everytime (logn vs nlogn)
6. if try to find the length of the encoding, do the tracing num of occurences of char +

fence hw problem

1. all the fences fell down in a fence
2. he has 3 planks of wood: 5, 8, and 8 feet long (user input)
3. the plank comes in one piece: 5+8+8 = 21 ft long
4. everytime he cuts the plank, he sill be charged the length of the plank





5. how to minimize cost?

e.g. 2, 2, 3, 3, 4

think about: if you end up with n pieces of wood, need (n-1) cuts
think about how you can form the original piece of wood, glue piece bak together
huffman coding problem
once you do (n-1) glues, you get back to original plank of wood