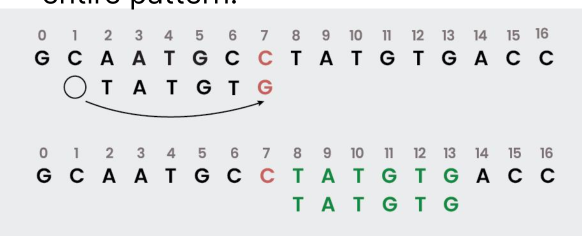
Boyer-Moore

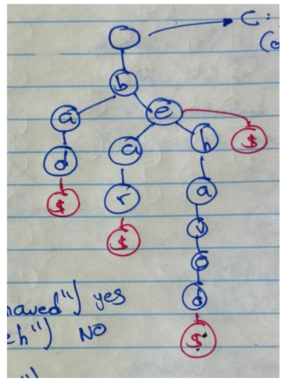
- search from last index of pattern
- keep a table of indexes in the pattern
- if mismatch, turn it into a match:

- if the last character mismatches, move entire pattern:



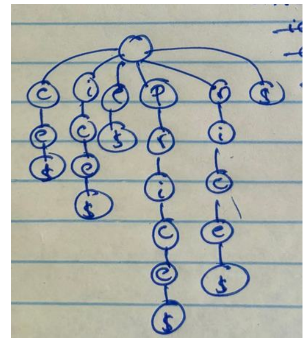
Tries

- structure for patterns using ordered alphabets
- optional '$' signifying pattern end (kinda like "\0")
- O(n) space complexity, n is the total length of all strings
- ~O(L) time complexity, L is the average length of a string



Suffix Trie

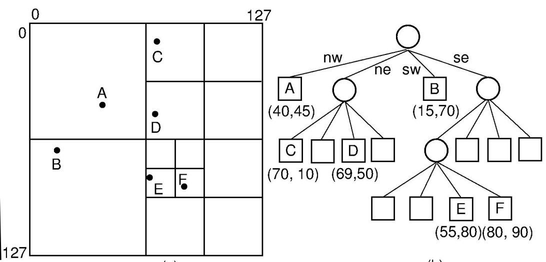
- form all possible suffixes from the string, terminate each with $
- if a path ends with $, it's a *suffix*. any path is a *substring*.



- to find the *longest repeated substring*, find the deepest node with more than one child.

Point-Region Quadtree

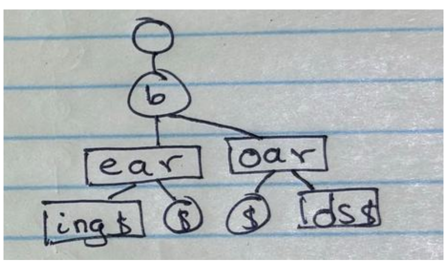
- each node has exactly 4 children: nw, ne, sw, se



(a) (b)

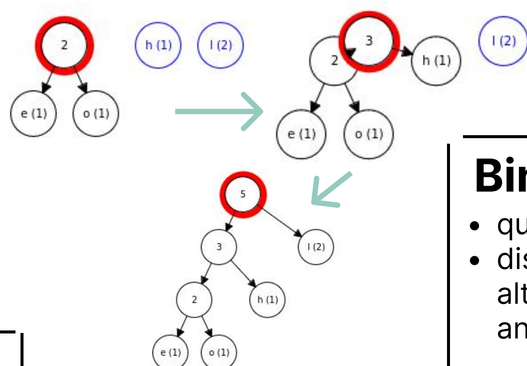
PATRICIA

- assume a node is redundant if it has one child and it's not the root.
- chain redundant nodes:



Huffman Coding

- repeatedly combine the 2 *least frequent* nodes into a subtree



Bintree

- quadtree but binary
- discriminator alternates between x and y coordinates

Bloom Filter

- pass search key through k hash functions.
- check if result of hash functions in your table is 1, then the searched key exists in your set.
- if any bit from the hashed result is 0---does not exist.
- collisions can occur, resulting in false positives.

n: # of items
f: false positive rate
m: # of bits
k: # of hash functions

$$k_{opt} = \frac{m}{n} \ln 2$$

$$f_{opt} = (\frac{1}{2})^{k_{opt}}$$

$$f \approx (1 - e^{\frac{k}{m/n}})^k$$

Count min-sketch

- Like a bloom filter, but instead of just storing either 1 or 0 in the table, we store multiple bits representing the integer count of the cell.

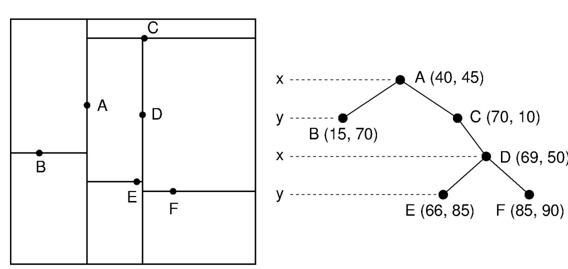
ε: band of overestimate
δ: failure probability

$$k = \ln(1/\delta)$$

$$m = \frac{e}{\varepsilon}$$

K-D Tree

- bintree but adaptive



(a) (b)