CS 240

Input for sorting algs

- list of ints
- no particular order

eg: 2, 5), 23, 14.8

what size is this instance?

n = 5 ints or n = 5×4 = 20 bytes

eg: Matrix Mult

$$\begin{bmatrix} c_{11} & \cdots & c_{1n} \\ \vdots & & \vdots \\ c_{n1} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & & \vdots \\ b_{n1} & \cdots & b_{nn} \end{bmatrix}$$

C A B

Size of input?

- # of ints in both matrices
- # of bytes
- dimension of matrices

eg. matrices that are 10 x 10

Dimension: $n = 10$

200 ints, 800 bytes.

n^2

$c_{11} = a_{11} \cdot b_{11} + \cdots a_{1n} \cdot b_{n1}$

n mults

n-1 adds

2n-1 ops

扫描全能王 创建

n^2 positions in C.

Total $n^2(2n-1) = 3n^3 - n^2 \in O(n^3)$

2017.9.12

Big-O

- upper bound on growth rate
- "grows no faster than

Relation	Basic idea	functions
$4 \leq 7$		$f(n) \in O(g(n))$
$8 \geq 7$		Ω
$7 = 7$		Θ
$3 < 7$		o
$7 > 3$		ω

eg. $0 \leq \underbrace{2n^2+3n+11}_{f(n)} \leq c \cdot \underbrace{n^2}_{g(n)} \quad \forall n \geq n_0$

Find an n_0 and c.

✓ You must ~~decide~~ derive c and

✗ Do not simply guess $c = 16$

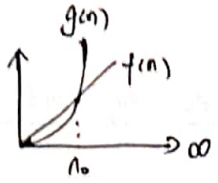
$g(n) = n^2$

Start $\underbrace{2n^2+3n+11}_{f(n)} \leq 2n^2+3n^2+11n^2 \quad \forall n \geq 1$

$\cdot n^2 \geq n \quad \forall n \geq 0$ ← derive this

$\cdot 11n \geq 11 \quad \forall n \geq 1$ ← justify

Big-Oh.



Typically we think $g(n)$ must dominate $f(n)$.

Eg. Prove $\underbrace{(n+1)^5}_{f(n)} \in O(\underbrace{n^5}_{g(n)})$.

$\therefore g(n)$ will never dominate $f(n)$

$\Rightarrow c > 1$

Note, not all values of c will work.

$(n+1) \leq 2 \cdot n \quad \forall n \geq 1$

$(n+1)^5 \leq (2 \cdot n)^5 = 2^5 \cdot n^5 = \underbrace{32}_{c} \underbrace{n^5}_{g(n)} \quad \forall n \geq \underbrace{1}_{n_0}$

eg $\underbrace{n^2 + n\log n + n}_{f(n)} \in O(\underbrace{n^2}_{g(n)})$

$n^2 + n\log n + n \leq n^2 + n^2 + n^2 \quad \forall n \geq 1$

since $n^2 \geq n\log n \quad \forall n \geq 1$

$n^2 \geq n \quad \forall n \geq 0$

$n^2 + n^2 + n^2 = \underbrace{3}_{c} \underbrace{n^2}_{g(n)}$.

Don't get lazy in proofs!

- take small steps. justify your statements.

From "First Principles" we explicitly find c, n_0.

Properties: Assume $f(n) \geq 0, g(n) \geq 0, \forall n \geq 0$

1. $af(n) \in O(f(n))$ for any constant $a > 0$

   pf: $c = a$, $n_0 = 1$.

2. if $f(n) \in^{①} O(g(n))$ and $g(n) \in^{②} O(h(n))$

   then $f(n) \in^{③} O(h(n))$

From

① $f(n) \leq c_1 \cdot g(n) \quad \forall n \geq n_1$

② $f(n) \leq c_2 h(n) \quad \forall n \geq n_2$

Substitude ② into ①

Show ③ $f(n) \leq c \cdot h(n) \quad \forall n \geq n_0$

$c = c_1 \cdot c_2$

$n_0 = \max(n_1, n_2)$

3. $\max(f(n), g(n)) \in O(f(n) + g(n))$

   pf: $c = 1$, $n_0 = 1$

Exercise: Show $f(n) + g(n) \in O(\max(f(n), g(n)))$

---

4. $a_0 + a_1 x^1 + a_2 x^2 + \cdots + a_n x^n \in O(n^n)$

5. $n^x \in O(a^n)$, $x > 0$, $a > 1$

   eg. $n^{1000000} \in O(1.000001^n)$

6. $(\lg n)^x \in O(n^y)$ $\quad x, y > 0$

Draft:

Show $f(n)+g(n) \in O(\max(f(n), g(n)))$.

3. $f(n) \le f(n)+g(n)$.

$g(n) \le f(n)+g(n)$

$\max(f(n), g(n)) \in O(f(n)+g(n))$

$f(n)+g(n) \le 2\max(f(n), g(n))$.

Thus $f(n)+g(n) \in O(\max(f(n), g(n)))$ $\forall c > 1$, $n_0 = 1$

a1. 1.

a) $n > \log n$ $\forall n_0 > 0$

$2016\, n > 2016$ $\forall n > 1$.

$27n^7 + 17n^3 \log n + 2016 \le 27n^7 + 17n^7 + 2016n^7 = 2060n^7$

$= \frac{2060}{n^2} n^9$

$\le cn^9$.

Since $c \ge \frac{2060}{n^2}$

$n^9 \ge \sqrt[9]{\frac{2060}{c}}$

For any c we choose $\forall n \ge \sqrt[9]{\frac{2060}{c}}$, thus $n^{27} - \cdots \le cn^9$, which means ...

b. Since $(\log n)^{1.001} \ge 1$ $\forall n \ge 2$

$n^2 (\log_2 n)^{1.001} > n^2$ when $c = 1$, $\forall n \ge 2$

c.

a. $\frac{n^2}{n+\log n} = \frac{n}{1+\frac{\log n}{n}}$ $\forall n \geq 1$ $\because \log n \leq n$ $\therefore \frac{\log n}{n} \leq 1$.

$1 \leq \frac{\log n}{n} + 1 \leq 2$

$\frac{n}{2} < \frac{n}{\frac{\log n}{n}+1} \leq n$

d. $n^{20} \leq 20n^{20}$

$\log_n n \leq 20$.

$\leq \frac{20}{n^{n-20}} n^n$

n^{n-20}

$n^n \geq \frac{1}{n^{n-20}} \cdot n^{20}$.

$\geq cn^{20} \geq \frac{1}{n^{n-20}} n^{20}$.

n^{20}

$c \geq \frac{1}{n^{20}} \cdot n^{n-20}$

$\log_n c \geq n-20$

$n^n \geq cn^{20}$

$\log n^n \geq \log cn^{20}$.

$n \geq \log c + 20$.

$n \log n \geq \log c + 20 \log n$

h.

$(n-20)\log n \geq \log c$.

$n^{20} < n^n \cdot n^{20}$ $n \geq 1$

$20 \log n \leq c \log n \leq c \log n$

$\log n + \frac{n-20}{n}$

$\log n + 1 - \frac{20}{n}$

$n^n > cn^{20}$

$n \log n > \log c + 20 \log n$

$(n-20)\log n > \log c$

$n^{n-20} > c$

$n > c$

$n-20 > 1$

$n > 21$

eg. $(\log n)^{10000000} \in O(n^{0.0000000001})$

Ω Big-Omega "grows no slower than"

lower bound.

eg. $n^3 \log n \in \Omega(n^3)$

$\underbrace{1}_{c} \cdot \underbrace{n^3}_{g(n)} \leq \underbrace{n^3 \log n}_{f(n)} \quad \forall n \geq 2$

Since $\log n \geq 1 \quad \forall n \geq 2$

Θ - Big-Theta $f(n)$ grows at same rate as $g(n)$.

Show $f(n) \in \Theta(g(n)) \Rightarrow$ show $f(n) \in O(g(n))$
and $f(n) \in \Omega(g(n))$

$0 \leq c_1 \cdot n^3 \leq 2n^3 - n^2 \leq 2n^3 \quad \forall n \geq n_0$

---

o - Little-O $f(n)$ grows slower than $g(n)$

· key difference "$\forall c$"

eg $n \in o(n^2)$

Given any c, we can always find n_0. $c = \frac{1}{10}, \frac{1}{100}, \frac{1}{100000}$.

ω Little-omega $f(n)$ grows faster than $g(n)$

eg $n^2 \in \omega(n)$ no matter how large & we choose c, $\exists n_0$.

$2010n^2 + 1388n \in O(n^3)$

$\forall c > 0 \quad \exists an \quad n_0 > 0 \quad st \quad 2010n^2 + 1388n < cn^3$

Proof: Let $c > 0$ be given
Note $0 < c < 1$ is possible

$$2010n^2 + 1388n < 5000n^3 \quad \forall n \geq 1$$
$$\leq \frac{5000}{n} n^3 \quad \forall n \geq 1$$
$$\leq c \cdot n^3$$

$\forall n \geq n_0$ Note $\frac{5000}{n} \leq c$

$$\forall n \geq \frac{5000}{c}$$

* ~~the~~ no ~~m~~ may depend on c.
can go as far as required.

For any c we choose $n_0 = \frac{5000}{c}$

2017.9.14

Insertion sort:

Instance: Given an ~~area~~ array of length n.

Best case: array ~~all~~ already sorted $\Rightarrow \Theta(n)$

Worst case: array reverse sorted $\Rightarrow \Theta(n^2)$

partially sorted - included in average case

average case:

- must consider all inputs of a certain size n
  - all possible permutations of array.

• Sometimes algs have bad worst case but perform well in practice

we often focus on worst case.

- insertion sort
  - worst case $\Theta(n^2)$
  - average case $\Theta(n^2)$

$$f(n) = \begin{cases} 3^n + \lfloor \frac{n^2}{2} \rfloor & \text{for } n \le 10 \\ 5.3n^3 + 2n^2 + 12 & \text{for } n \ge 11 \end{cases}$$

$f(n) \in O(n^3)$ $\quad$ $f(n) \in \Theta(n^3)$

$f(n) \in O(3^n)$

n	nlogn	n^2	n^3	2^n
4	8	16	64	16
16	64	256	4096	65,536
64	384	4096	262,144	1.84×10^{19}

Matrix Mult

A_1: Standard : $2n^3 - n^2 \in \Theta(n^3)$

A_2: Strassen : $42n^{2.807\ldots} \in \Theta(n^{\log_2 7})$

$n = 10$ $\quad A_1 = 1900$

$A_2 = 24955$

Let $i > 0$

$$\lim_{n\to\infty} \frac{\log n}{n^i} = \lim_{n\to\infty} \frac{\frac{1}{n}}{i n^{i-1}} = \lim_{n\to\infty} \frac{1}{i n^i} = 0$$

So $\log(n) \in o(n^i)$ for any $i > 0$.

$n \le n(2 + \sin\frac{n\pi}{2}) \le 3n$

Arithmetic Sequence - a, d are constants.

eg. $\sum_{i=1}^{n} i = 1+2+\cdots+(n-1)+n = (n+1)\left(\frac{n}{2}\right) = \frac{n^2+n}{2}$

eg. $S_1 = \sum_{i=1}^{\infty} \frac{1}{2^i} = 1+\frac{1}{2}+\frac{1}{4}+\cdots$

Geometric series: $a=1, r=\frac{1}{2}$ $0<r<1$ case 3.

$\Rightarrow a\frac{1-r^n}{1-r} = \frac{1-(\frac{1}{2})^n}{1-\frac{1}{2}} = 2\left(1-(\frac{1}{2})^n\right)$

"closed form" $\lim_{n\to\infty} \sum_{i=0}^{n-1} (\frac{1}{2})^i = \lim_{n\to\infty} 2(1-(\frac{1}{2})^n) = 2$

$S_2 = \sum_{i=0}^{\infty} \frac{i}{2^i} = \frac{0}{1} + \frac{1}{2} + \frac{2}{2^2} + \frac{3}{2^3} + \cdots$

How to solve this?

$2\cdot S_2 - S_2$

$$\begin{array}{r} 1+\frac{2}{2}+\frac{3}{2^2}+\frac{4}{2^3}+\cdots \\ -\frac{1}{2}-\frac{2}{2^2}-\frac{3}{2^3}-\cdots \\ \hline 1+\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\cdots \end{array}$$

$\Rightarrow 2\cdot S_2 - S_2 = S_1$

$\Rightarrow S_2 = 2$

$H_n = \sum_{i=1}^{n} \frac{1}{i}$ · harmonic series · diverges very slowly

$\sum_{i=1}^{10^{100}} \frac{1}{i} \simeq 230.84$

$\lg 2^n + n^2 2^{3\lg n} = n\lg 2 + n^2 2^{\lg n^3}$

$= n\cdot 1 + n^2 \cdot n^{3\lg 2}$

$= n + n^5$

Page 41

- each iteration of inner loop $\Theta(1)$
- How many times do the loops iterate

- work from from innermost to outermost.

Direct method

$$\#\text{ iterations} = \sum_{i=1}^{n}(n-i+1) = \sum_{i=1}^{n} i = (n+1)\left(\frac{n}{2}\right) = \frac{n^2+n}{2} \in \Theta(n^2)$$

$\Rightarrow$ complexity: $\Theta(n^2)$

Sloppy Method to get O-bound.

$$\#\text{ iter} = \sum_{i=1}^{n}(n-i+1) \leq \sum_{i=1}^{n} n = n^2$$

$\Rightarrow O(n^2)$

Caveat: still need to prove $\Omega(n^2)$

$$\sum_{i=1}^{n}(n-i+1) \geq \sum_{i=\frac{n}{2}}^{n}\frac{n}{2} = \frac{n}{2}\cdot\frac{n}{2} = \frac{n^2}{4}$$

- start halfway
- use smallest value.

Page 42 each iteration has cost $\Theta(1)$

Determin # iterations

Direct Method:

$$\#\text{ iter} \sum_{i=1}^{n}\sum_{j=1}^{i}(j-i+1)$$

- seperate out part of sum that depends on the index.

$$= \sum_{i=1}^{n}\left[\sum_{j=1}^{n} j - \sum_{j=1}^{i}(i-1)\right] \boxed{(n-i+1)(i-1)}$$

$$= \sum_{i=1}^{n}\left[\sum_{j=1}^{n} j - \sum_{j=1}^{i-1} j - (n-i+1)(i-1)\right]$$

$$= \sum_{i=1}^{n} \left[\frac{n(n+1)}{2} - \frac{(i-1)i}{2} - [(n-i+1)(i-1)] \right]$$

seperate terms, expand, simplify.

$$= \sum_{i=1}^{n} \left[\frac{1}{2} i^2 + (-n - \frac{3}{2}) i + \frac{n(n+1)}{2} + n + 1 \right]$$

$$= \frac{1}{2} \sum_{i=1}^{n} i^2 + (-n - \frac{3}{2}) \sum_{i=1}^{n} i + n \left[\frac{n(n+1)}{2} + n + 1 \right]$$

$\sum_{i=1}^{n} i^2 = n(n+1)(2n+1)/4$; $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

$$= \frac{1}{4} n^3 + \frac{1}{2} n^2 + \frac{1}{3} n \quad \Rightarrow \text{\# iteration is } \Theta(n^3)$$

Easy to get O-bound.

$$\sum_{i=1}^{n} \sum_{j=i}^{n} (j - i + 1) \leq \sum_{i=1}^{n} \sum_{j=1}^{n} n = n^3 \Rightarrow O(n^3)$$

Page 43.

Floors and ceilings are difficult to work with!

Fact: $\lfloor j/2 \rfloor \leq j/2 \quad \forall j \geq 1$

create a modified alg.

- replace $(j \leftarrow \lfloor j/2 \rfloor)$ with $j \leftarrow j/2$

=> O-bound.

For a given i, find # iter of inner loop.

Solve for k: $i * (\frac{1}{2})^k = 1$ ~ while condition.

$\lg i + k \lg(\frac{1}{2}) = 0$ => $\lg i = k$

$k <= \lg i$

Is k # iterations?

No, but it close

- real number $\lg 7 \approx 2.8$
- out by one $\lg 8 = 3$ but loop 4 times.

iters $\lfloor k \rfloor + 1 \le k+1$

$\forall i \ge 1 \quad \sum_{i=1}^{n} (\lfloor \lg i \rfloor + 1) \le \sum_{i=1}^{n} (\lg i + 1) \le \sum_{i=1}^{n} (\lg n + 1)$

$= n \lg n + n \in O(n \log n)$

2017.9.19

Divide and conquer

· Always recurse on a smaller problem

Method 1: Expand "Plug & Chug"

$T(n) = 2T(\frac{n}{2}) + cn$ $\quad$ $T(k) = 2T(\frac{k}{2}) + cn$

$= 2[2T(\frac{n}{4}) + c \cdot \frac{n}{2}] + cn$

$= 4T(\frac{n}{4}) + c[2(\frac{n}{2}) + n]$

$= 4[2T(\frac{n}{8}) + c \cdot \frac{n}{4}] + c[2(\frac{n}{2}) + n]$

$= 8T(\frac{n}{8}) + c[4(\frac{n}{4}) + 2(\frac{n}{2}) + n]$

...

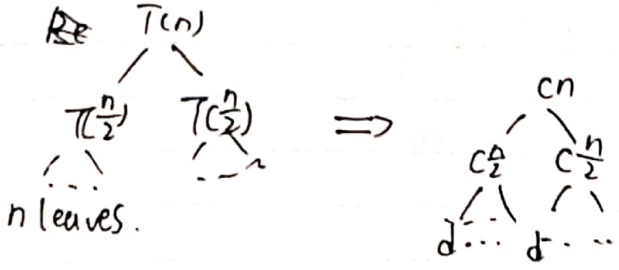
$= nT(1) + c[n + 2(\frac{n}{2}) + 4(\frac{n}{4}) + \cdots + (\frac{n}{2})(\frac{n}{\frac{n}{2}})]$

$= n \cdot d + cn(\lg n - 1) \in \Theta(n \log n)$

?? $\lg(\frac{n}{2})$

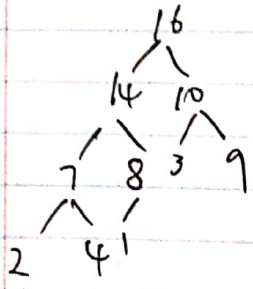
Method 2: Recursion tree with costs

· indicate non-recursive cost at each node.



Heap: - Not a BST

shape △ ← last raw may not full.



level	0	1	node 2^0
level	1	2	nodes 2^1
level	2	4	nodes 2^2

Lemma: Height with n nodes is $\Theta(\log n)$

Notation: height of node : # of edges in longest simple path down to leaf.

depth of node: # of edges in longest path up to the root

eg. node (14) height: 2
depth: 1

pf: Suppose height of tree is h.

$$n \geq 2^0 + 2^1 + 2^2 + \cdots 2^{h-1} + 1 = 2^h$$

$$n \geq 2^h \Rightarrow \lg n \geq h \lg 2$$

$$h \leq \lg n \quad \text{so} \quad h \in O(\log n)$$

Lower Bound is smaller

$$n \leq 2^0 + 2^1 + 2^2 + \cdots 2^h = 2^{h+1} - 1$$

$$\Rightarrow h \geq \lg(n+1) - 1$$

So $h \in \Omega(\log n)$

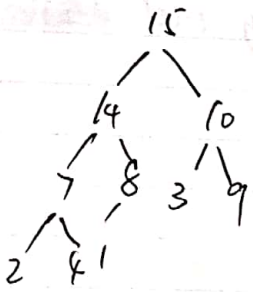
$$\Rightarrow h \in \Theta(\log n)$$

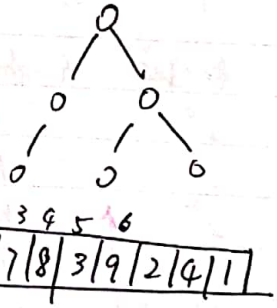
Insert is
- maintain structure
- order

Bubble-up $\in O(\log n)$

DeleteMax

Bubble-down $O(\log n)$

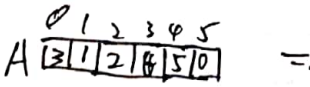




root A[0]
- Left

Bottom up Heapify

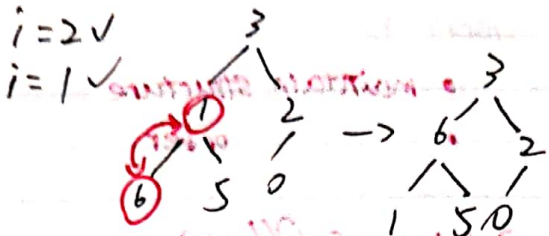
for i=n-1 down to 0
  bubble down at position i



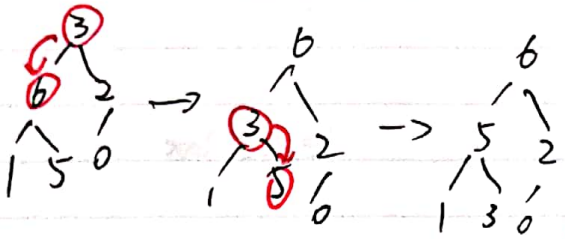
n=6

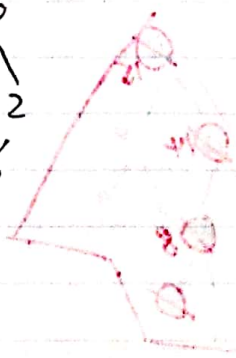
i = 5
  = 4
  = 3

* We will never bubble down on a leaf node (or node at level h)
  => use $\lfloor \frac{n}{2} \rfloor$ in loop



i=0





2017.9.21.

Runtime: Bottom up Heapify. consider nodes by level ~~#nodes~~.

level	#nodes	height of node
0	2^0	h
1	2^1	$h-1$
2	2^2	$h-2$
⋮	⋮	⋮
h	2^h	0

nodes with height $i \leq 2^{h-i}$

swaps for bubble down from height i is $\leq i$

Total # swaps: $\sum_{i=0}^{h} i \cdot 2^{h-i} \leq \sum_{i=0}^{h} n \cdot \frac{i}{2^i} \leq n \sum_{i=0}^{\infty} \frac{i}{2^i} = 2n \in O(n)$

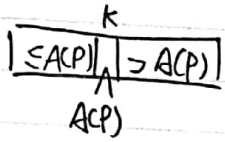
Given a number. can we find its index?

[9] 3 2 8 7 4 11 12 22 1

=> 9 belongs at index 6 if sorted

Given a index, what # belongs at that index?

Index k: more difficult or at least not as straight forward.



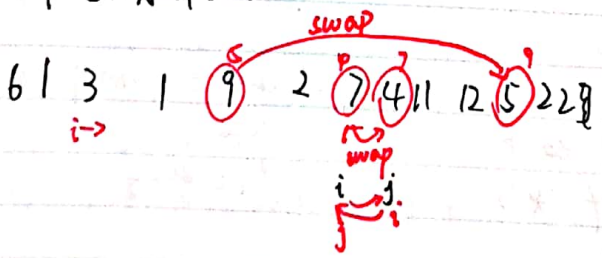
input A[* * ... * A[p] * ... *]
=x

after A[* ... * x * ... *]
<x >x

- P may not be the same as q.
- participating is done "in place"

A[7 3 1 9 2 6 4 11 12 5 22]

p=5, A[p]=6



- finally swap A[0] with A[j]

A[4 3 1 5 2 6 7 11 12 9 22]

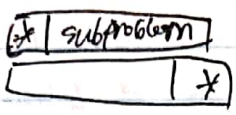
L i R

Quick sort selected . choose pivot

- given k
- partition

Figure out how to proceed.

① if $i=k$ found element at k, done.
② if $i>k$, recurse, find k in L.
③ if $i<k$, recurse, find $k-i-1$ in R.

worse case: 

• one subproblem of size $n-1$ to recurse on.

$$T(n) = T(n-1)+cn$$
$$= [T(n-2)+c(n-1)]+cn$$
$$= T(n-3)+c(n-2)+c(n-1)+cn$$
$$= \overbrace{d}^{T(1)} + c\cdot 2 + c\cdot 3 + \cdots c(n-1)+c(n)$$
$$= d + c\left[\frac{(n+2)(n-1)}{2}\right] \in \Theta(n^2)$$

Base case: we're lucky - find the item in the first pass.
• still must partition once $\in \Theta(n)$

Average case:?

```
foo1(k)
  for i=1 to k
    printf("*")
```

calls to printf on average?
$1 \le k \le 6$
$\frac{1}{6}\sum_{i=1}^{6} i = \frac{21}{6} = \frac{7}{2}$

• consider all inputs
• all inputs are "equally likely"
• take average.

Assumption:
• each input is equally likely - "uniform distribution"

Need to consider
• all inputs of a certain size, then take average.
• sum up runtime.
• devide by # inputs.

Assumption

- no items are repeated.
- behavior of alg depends on relative ordering of keys
  - not actual values.

eg [2 4 6 8] & [11 12 13 14] both worse case.

=> may assume keys are $1, 2, \cdots, n$.

=> need to consider all orderings: $n!$

We'll count # comparisons.

Remember: n possible positions for pivot

- each one has $(n-1)!$ permutation of non-pivot elements.
- each pivot location is "equally-likely"
  => devide by n.

Case: ① $i=k$ done

② $i>k$ Search $L = A[0 \ldots i-1]$ for item k

③ $i<k$ search $R = [i+1 \cdots n-1]$ for item $k-i-1$

$$T(n,k) = (n) + \frac{1}{n}\left[\sum_{i=0}^{k} T(n-i-1, k-i-1) + \sum_{i=k+1}^{n-1} T(i,k) + 0\right]$$

Let's be lazy: Define $T(n) = \max_{0 \le k < n} T(n,k)$

A [ ] [ ] [ ]

$\frac{1}{2}$ time pivot is here
- subproblem size at most n.

$\frac{1}{2}$ time pivot is here
- subproblem size at most $\lceil \frac{3}{4}n \rceil$

$$T(n) \le \begin{cases} cn + \frac{1}{2}\left(T(n) + T(\lceil \frac{3n}{4} \rceil)\right) & n \ge 2 \\ d & n = 1 \end{cases}$$

$T(n) \le 2cn + T(n) + T(\frac{3}{4}n)$

$Th) \le 2cn + T(\frac{3}{4}n)$

$\le 2cn + 2c(\frac{3}{4}n) + 2c(\frac{9}{16}n) + \cdots + d$

$\le d + 2n \sum_{i=0}^{\log_{\frac{4}{3}} n} (\frac{3}{4})^i \le 4 \in O(n)$ Average case

Best case : $\Omega(n)$ $\Rightarrow \Theta(n)$ Average case

worse case : $\Theta(n^2)$

Module 3 Page 10

```
foo1(k) // 1 ≤ k ≤ 6
    for i=1 to k
        printf("*")
```

average case : $\frac{1}{6}\sum_{i=1}^{6} i = \frac{7}{2}$

```
foo2
    k ← rdl a 6 sided die
    foo1(k)
```

(random)

"expected" number of calls to printf.

$\sum_{i=1\ldots6} (\frac{1}{6}) \cdot i = \frac{7}{2}$

probability · uniform distribution — cost of case.

requires A[0...n-1] is a permutation of : 1, 1, 1, ... 1, n

n-1 ones

```
foo3(A, n)
    k=0
    for i=1 to A[k]
        printf("*")
```

prints in
best case?
worst case?

```
foo4(A, n)
    k ← random int in 0...n-1
    for i=1 to A[k]
        printf("*")
```

Let I_1 be fixed input $A[1\ldots n]$ $n \ge 4$ $a_{1\ldots n}$

$T^{(exp)}(I_1) = \sum_{k=0\ldots3} T(I_1, k) \cdot \frac{1}{4}$

$$T^{(exp)}(I_1) = \sum_{k=0\ldots3} T(I_1,k) * \frac{1}{4}$$

$$= 1\cdot\frac{1}{4} + 1\cdot\frac{1}{4} + 4\cdot\frac{1}{4} + 1\cdot\frac{1}{4} = 1\frac{3}{4}$$

let I_n be input $A[n, 1, 1, \ldots, 1]$

$$T^{(exp)}(I_n) = \sum_{k=0\ldots n-1} T(I_n,k)\cdot\frac{1}{n}$$

- input is fixed
- k is random
- probability of each case uniform dist

$$= 1\cdot\frac{n-1}{n} + n\cdot\frac{1}{n}$$

$$= \frac{2n-1}{n} < 2$$

Worst case Expected runtime

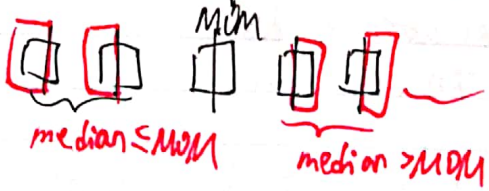
$$T^{(exp)}(n) = \max_{size(I)=n} T^{(exp)}(n)$$

- max of all inputs of a certain size
- each input has $\begin{cases} n-1 \text{ ones} \\ \#1\ n \end{cases}$

~~BER~~ BFPRT

- partition the array into groups of 5 items
  - add number, few left over
- take median of each groups of 5
- take median of medians

~ 2 items smaller
~ 2 items bigger



~ blocks of 5 items

$$T(n) \le T(\lceil \tfrac{n}{5} \rceil) + T(\tfrac{4}{5}n) + O(n)$$

median of medians · recurse · partition

Worst case : array sorted.

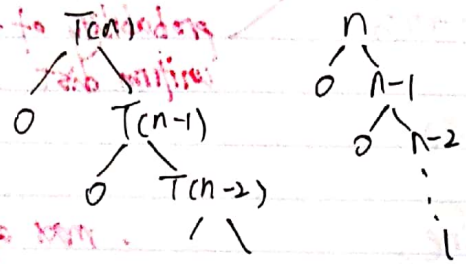
choose index 0 as pivot.

after partition A[0 | 1 2 ... n]

pivot

subproblem 1 | subproblem 2

Recursion Tree:

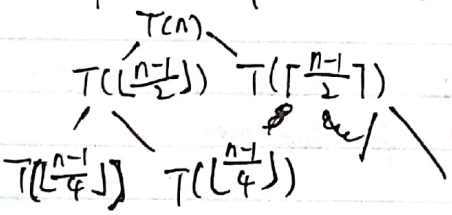


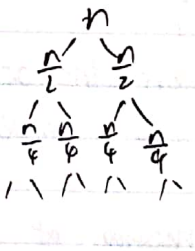
$T^{worst}(n) = T^{worst}(n-1) + \Theta(n)$

$\Rightarrow \Theta(n^2)$

Ideally want Balance Partitioning

- subproblems of size $\lfloor \frac{n-1}{2} \rfloor$ and $\lceil \frac{n-1}{2} \rceil$





upper bound on comparisons

n
n
n
.
.
.
1

Best Case $\Rightarrow \Theta(n \log n)$

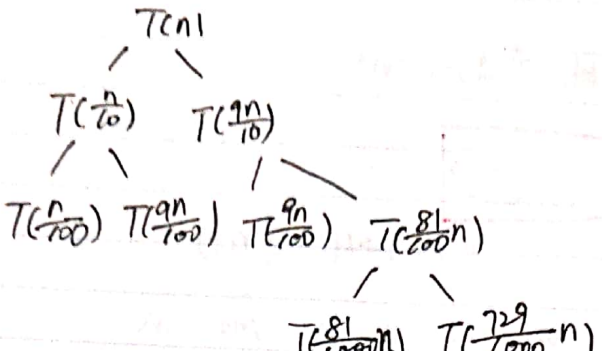
Average case : — not worst or best.

- maybe $\frac{1}{4}n$ and $\frac{3}{4}n$

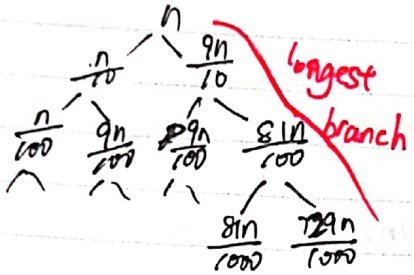
Suppose partitioning always split $\frac{1}{10}n$ and $\frac{9}{10}n$.

$$\#\text{comparisons } T(n) = \begin{cases} T(\frac{9n}{10}) + T(\frac{n}{10}) + n & \text{if } n > 1 \\ 0 & \text{if } n \leq 1 \end{cases}$$

Recursion Tree



Work



comparisons

n
n
n
n
⋮

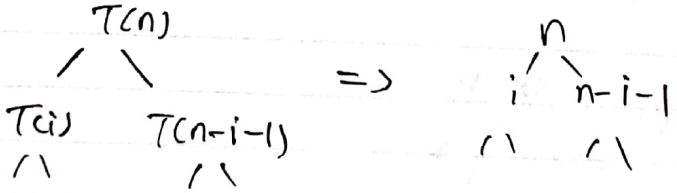
Deepest leaf

Height of tree is minimal integer h such that

$$\left(\frac{9}{10}\right)^h \cdot n \leq 1 \Rightarrow h \lceil \log_{\left(\frac{10}{9}\right)} n \rceil \in O(\log n)$$

Total # comparisons $\in O(n \log n)$

Average cost



$$T(n) = \frac{1}{n} \sum_{i=0}^{n-1} [T(i) + T(n-i-1)] + \theta(n)$$

Let $H(n)$ be the average height of the recursion tree.

$H(n) \le 1 + \frac{1}{2}H(\lfloor \frac{3}{4}n \rfloor) + \frac{1}{2}H(n)$

$2H(n) \le 2 + H(\lfloor \frac{3}{4}n \rfloor) + H(n)$

$H(n) \le 2 + H(\lfloor \frac{3}{4}n \rfloor)$

$\le 2 + 1 + H(\lfloor \frac{9}{16}n \rfloor)$

$\le 2 \cdot h$ where h is minimal s.t.

$(\frac{3}{4})^h \cdot n < 2$

$\Rightarrow H(n) \in O(\log n)$ · height.

$\Rightarrow$ Average case : $O(n \log n)$

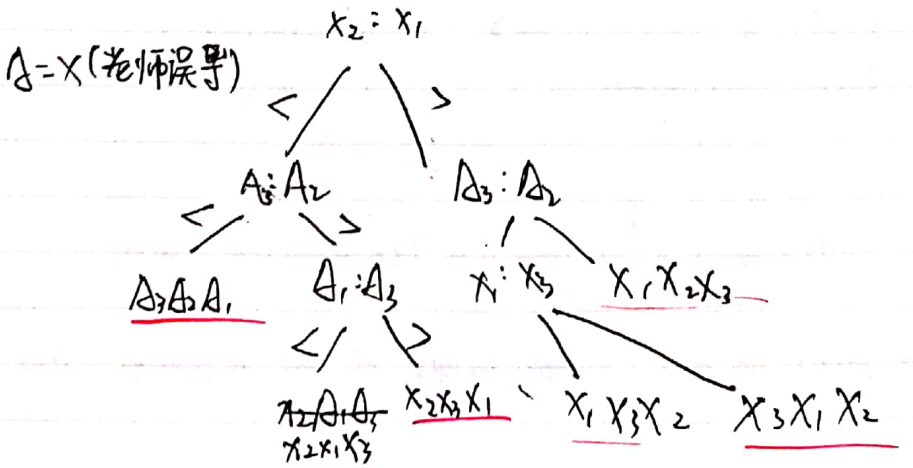
Best case : $\Theta(n \log n)$ $\Rightarrow$ Average case : $\Theta(n \log n)$

Comparisons: $<$, $>$, $=$

Decision tree:

eg. $n=3$ $[A_1, A_2, A_3]$

$\{[3\ 1\ 2]\ [2\ 1\ 3]\ [1\ 2\ 3]$
$[3\ 2\ 1]\ [2\ 3\ 1]\ [1\ 3\ 2]\}$



leaves : n!

leaves = # internal nodes +1

n! = n!-1 +1

height ≥ lg(n!)

≥ Ω(nlgn)

Page 23

Input A[4 5 10 5 5 10] R=11 n=7

1. C[0 0 0 0 0 0 0 0 0 0 0]
   1 1 3 1 1

2. C[0 1 1 1 2 5 5 5 5 5 5]

3. B[4 5 7 0 5 5 10]
   0 1 2 3 4 5 6
   A[0 4 5 7 10]

#	start index
0	0
4	1
5	2
7	5
10	6

A[1] = 4
A[2] = 5
A[5] = 7
~~A[6] =~~
A[0] = 0
A[3] = 5

"stable sort"

θ(n+R)

what if n = R => θ(n)

{3, 14, 159, 9265432}

8 comparisons.

n=4 R = 9265432

MSD Radix sort.

A[829 331 457 330 436 720 355]

n=7 R=10 3-digit number (m=3)

MSD-Radix sort (A, 0, 6, 1)
· order by 1st digit.

331]
330]
355]

331]
330]
355]

330
331
355

457]
436]

434]
457]

434
457

720]

720]

720

829]

829

recursively using 2nd digit sort each digit

MSD
· leftmost to right
· sort by digit.
"stable"

· many recursions

· works on variable-length

LSD-Radix Sort.

A[829 331 457 330 436 720 355]
n=7, R=10, m=3

330 720 331 355 436 457 829

720 829 330 331 436 355 457

330 331 355 436 457 720 829

LSD to MSD
right to left

$\Theta(m(n+R))$ → count sort
↑ m digits.
calls to count sort

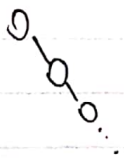
~~Bye~~

2017. 10.3. Module 04

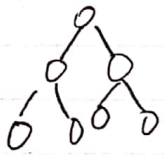
What are we counting for height?

- vertices or edges?
  => edges.

Worst case:
n nodes
n-1 edges

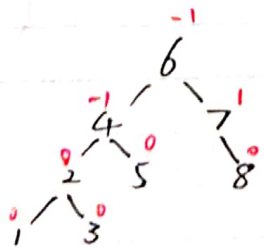


Best case:
a balanced

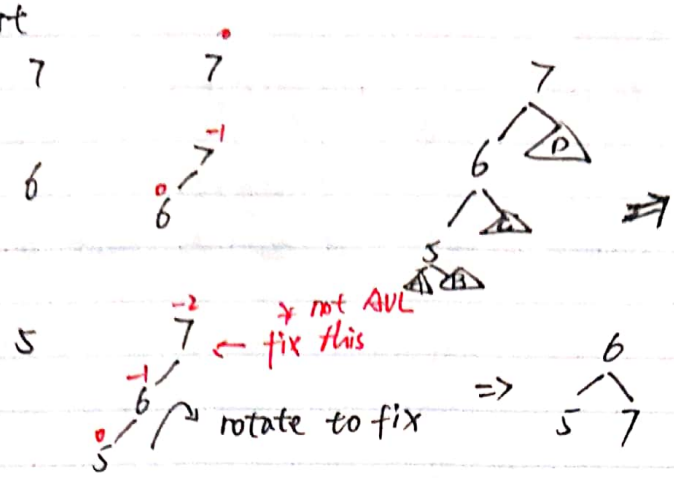


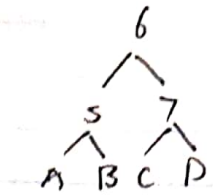
average case: $h \in \Theta(\log(n))$
- over all n! tree
- similar to quicksort analysis

AVL
- "balanced" - slight inperfection
- every node is "balanced".
- store ~~banla~~ balance factor in the node ???

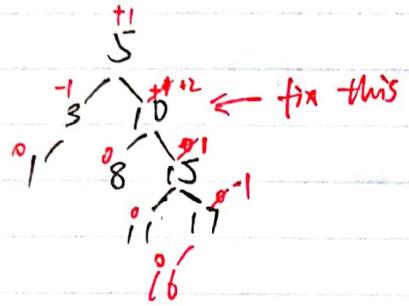


Insert

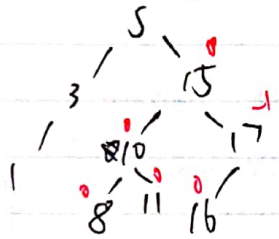




ordering properties maintained
still a BST



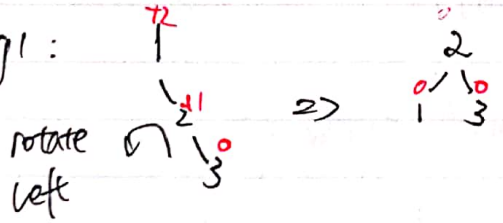
Insert 16
- update balance
- fix any that are not +1, 0, -1



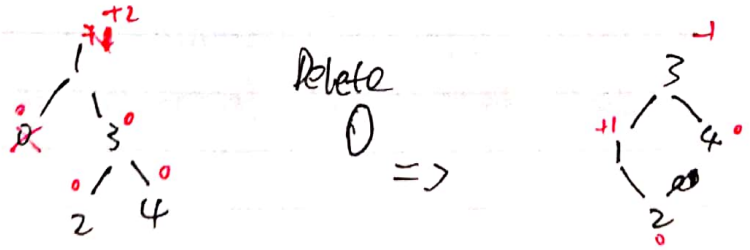
fixed: continue upwards fixing balance

* Note: functions return pointer to the new tree.

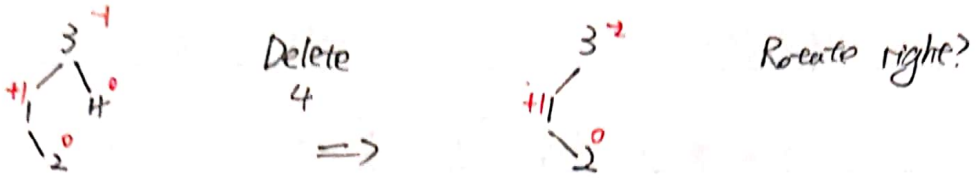
eg1:



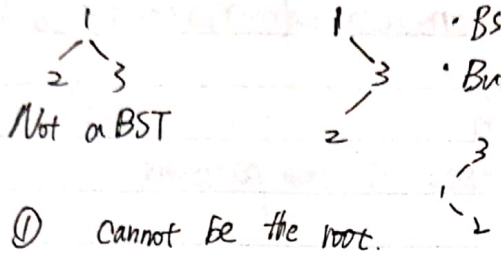
eg2:



Note: Balance factors are different for case (+2, 0) then for (+2, 1)

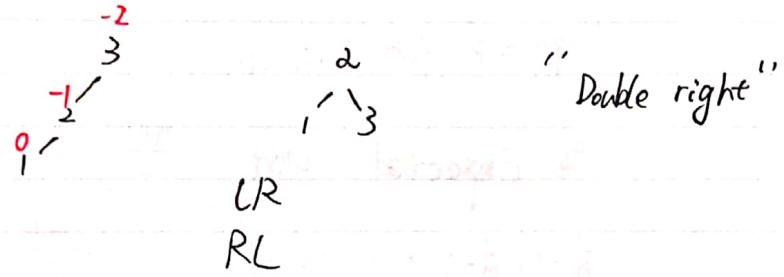


Can 1 be the root?



=> ① Cannot be the root.

Idea: do left rotation at ① first then right rotation



2017. 10.5.

Self-Organizing Search.

- Think of a messy desk with papers all over it.
- The things that are frequency used or recently used are conviently located on top.
- Could sort by frequency of access
  - sort items by their ~~prop~~ probability of access

Model.

- Search items 1, 2, 3.
- only count comparisons
  - all unsuccessful searches, cost n.

successful search.

- Worst case: n comparisons.
- p_i – probability to search for item x_i
  what if uniform distribution?

$p_i = \frac{1}{n}$ for $i = 1 \dots n$.

Expected cost: $\frac{n+1}{2}$

$\frac{1}{n} \cdot 1 + \frac{1}{n} \cdot 2 + \dots \frac{1}{n} \cdot n = \dots = \frac{n+1}{2}$

Non-uniform Distribution.

order $A_1 \dots A_n$ where $p_1 \geq \dots \geq p_n$.

Uniform Distribution: Expect $O(n)$

What if $p_i \geq \frac{1}{2^i}$ for $i = 1 \dots n-1$

and $p_n = 2^{-n+1} = \frac{1}{2^{n-1}}$

Expected $= \sum_{i=1}^{n-1} \frac{i}{2^i} + n \cdot 2^{1-n} = 2 - 2^{1-n} < 2$

In practice, Distribution is usually unknown

In practice, 80% of searches are on 20% of keys.

Optical static Ordering.

proper oder $\frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 1$

Move-to-front

Given: 1 2 3 4 5 6.

Search 3 3 1 2 4 5 6
4 4 3 1 2 5 6
6 6 4 3 1 2 5

- quick to steady state
- bad in an array
- affected by a rare lookup
  - lots of work to shuffle it back.

Can show $C_{MTF} \leq 2 C_{OPT}$

C_{MTF} cost of MTF
C_{OPT} cost of optical ordering

Transpose:

Given	1	2	3	4	5	6
Search 3	1	3	2	4	5	6
4	1	3	4	2	5	6
6	1	3	4	2	6	5

- slow to steady state
- unaffected by a rare lookup.
- good in an array.

Given	0	1	2	3	4	5	6	7
Accesses	5	3	5	6	4	6	5	0 3 5 6 4

MTF: 4 6 5 3 0 1 2 7.
- 54 comparisons.

Transpose: 0 1 5 3 6 4 2 7
- 62 comp.

2017.10.12

Skip Lists - Binary search or Linked list.

- sorted arrays: binary search.
  - must shift items - insert & delete
- Linked Lists:
  - easy to move items around if we know where they are.
  - no binary search

AVL:

Discovered by Bill Pugh (1989)

Competes with balanced tree structures.
- other after wins.

Randomized - random element as part of alg.

Idea: Given singly linked list - search down S_0

add an express lane S_1

Search(23)

Implementation: LinkedNodes

- different # pointers per node? S_0 vs $S_1 S_2 S_3 \ldots$
- pointers will change with delete and insert.

Doubly Linked list? - not necessary lots of nodes
lots of waste space.

Ideally, want shortcut (towers)

Search 52.

- get stack S from search
- Random element: height of tower for new node.

coin toss: H T => i=1 (2 levels)

- add 52 to S_0 and S_1
- add 52 after 44 on S_0
  after 37 on S_1

Insert 600.

coin toss H H H T => i=3 (4 levels)

- insert new level
- high increases.

* top level is always $-\infty \rightarrow +\infty$ by defn.

How tall can a tower be? very tall.

Stats: probability of H (or T): $\frac{1}{2}$

probability of tower of height 3.

$\frac{1}{16}$ => HHH T

probability of 10 Hs followed by one T:

$(\frac{1}{2})^{11} = \frac{1}{2048} \leq 0.05\%$

controlling Height:

1) Set max height to be a constant
as $n \to \infty$, not enough levels.

2) have height $h \leq f(n)$, eg $f(n) = \lceil 3 \log n \rceil$ works well.

3) allow $h \to \infty$, not likely to get too tall.

Delete

- update links
- free nodes
- remove any extra top levels.
  - may decrease height.

space: expect number of kess of level i:

- probabily of getting i or more H in a row.

n node on S_0

Expected # nodes on S_1 : $\frac{n}{2}$

S_2 : $\frac{n}{4}$

S_3 : $\frac{n}{8}$

S_i : $\frac{n}{2^i}$

Expected # levels : $\theta(\log n)$

Expected # nodes at level $S_i = \frac{n}{2^i}$

Sum over all levels: $\sum_{i=0}^{h} \frac{n}{2^i} = 2n - \frac{n}{2^h} \approx 2n \in \Theta(n)$

Time:

worst case for search/insert/delete: $O(n+h)$

Expected height: $O(\log n)$
Expected number: $O(n+\log n)$?

- Don't usually scan all n elements

Refined analysis:

Idea: Backwards analysis: follow path in reverse. always move up if you can.

How long is backwards path before rising k levels.

Probability Node will have a tower to go up?

$C(n)$ $\frac{1}{2}$

$C(n)$ = # steps to go up n levels.

- 1 step then either:
  - go up, prob $\frac{1}{2}$: $C(n-1)$
  - go left, prob $\frac{1}{2}$: $C(n)$

=> Expected # steps to go up n levels is

$$C(n) = 1 + \frac{1}{2}C(n-1) + \frac{1}{2}C(n)$$
$$2C(n) = 2 + C(n-1) + C(n)$$

$$C(n) = 2 + C(n-1)$$
$$C(n) = 2 + 2 + C(n-2)$$
$$\cdots$$
$$= 2n \quad \text{where } n \text{ is \# of levels}$$

Expected
O(log n) levels => O(log n) steps
· constant time at each level (expected)

2017.10.17.

Tries "try"

Binary Trees
· store collection of binary strings (key)
· runtime analysis based on length of string not number of string in trie.

Compressed Trie (Patricia Trie).
· eliminate unflagged nodes with only one child.

=> save space

Multiway - Tries

· extension to ~~et~~ arbitrary alphabet.

· keys can be variable length.
· keys are not explicitly stored in trie. / could stop associated value.
· Edge labels are implicit: left: 0 } we show them for convenience
right: 1 }

Module 6 Slides 3/27

Slides 18/27
Search (0 1 0 0 1) (1 2 3 4)
Check Bit
0 0 => go left
1 1 => go right
2 0 => go left

At leaf: compare stored key (in file) with search key 0/001.

: match (still $\Theta(1 \times 1)$)

Search (101)

Check Bit

0 1 => go right

2 1 => go right.

At leaf: compare stored - seach key

• x match

Search (1110)

Check Bit

0 1 => go right

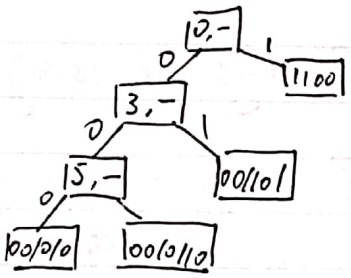
1 1 => go right

At leaf: compare search - stored key.

• x match (arrive at some leaf with search (101))

Slide 25/27.

Bit	0	1	2	3	4	5	6
	0	0	1	1	0	1	
	0	0	1	0	1	0	
	0	0	1	0	1	1	0
	1	1	0	0			



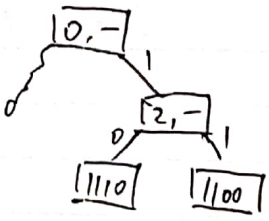
Try to find bits where strings disagree.

Insert a new key (1110)

• search ends at leaf [1100]

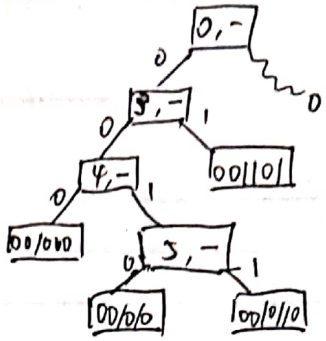
• disagree at index 2.

• insert a new node



Insert (00/000)

- search ends at leaf [00/010]
- disagree at index 4
- insert a new node
- goes between nodes with index 3 ~~start~~ and 5.

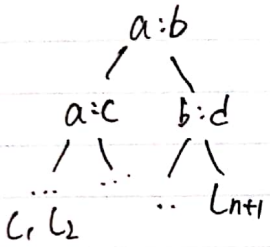


Module 07.

- n keys

Thm: comparison model $\Omega(\log n)$ comparisons required.

- each comparison has two outcomes,
- any algorithm can be (2) modeled by binary tree.



- Total # of outcomes is: n+1
  - one of n keys.
  - or "not found"

=> need n+1 leaves

- binary tree => n interval nodes

=> height $\Omega(\log n)$

Dictionary as an array - Problems?

① · Keys are usually not simple eg strings.

② · storage if $n \ll M$

often # keys used $\ll$ # possible keys.

Direct Address is OK if we can allocate array with one ~~position~~ position for every key.

Idea: try to reduce amount of wasted space.

Let M be # buckets in hash table T → array T[0, ... M-1]
Let n = # actual KVPs stored in table

Goal: · reduce storage to $\Theta(n)$
· still get $O(1)$ time for all operations at least in average
· not necessarily 1 but a constant #.

· a key K "hashes" to index $h(K)$ in T
If $M < |U|$ then ... Probem? · two or more keys hash to the same bucket.
=> Collision

· more possible keys than indices
· if $n \leq M$ ~ may or may not happen
· if $n > M$ ~ definitely happen. pegeon hole principle

What is the likelihood of a collision?
Birthday Paradox suppose n people with "random" birthdays.
· $M = 365$.

What is the probability that 2 people have same birthdy?
· if $n > 365$ => probability is 1 100%

Probability of a shared birthday is : $1 - \frac{\binom{365}{n}}{365^n}$

$n = 2$ => 0.2%
$n = 10$ => 12%
$n \geq 23$ => 50%
$n = 50$ => 97%

Conclusion : collisions are very likely even when $n \ll M$

· must be prepared to handle them

Ehard Challenges: · good hash funetion

· dealing with collisions $h(K_1) = h(K_2)$, $K_1 \neq K_2$

"Good" Hash function
- be unrelated to patterns in the data.
- depend on all parts of the key
- be efficient to compute

~~Chaining~~ Chaining ~~analysis~~ analysis: n KVPs in table T.
M ~~buckets~~ backets in table T.

Load factor: $\alpha = \frac{n}{M}$

Search: cost depends on length of particular lish that we hash to ~~calculate~~ unsuccessful search.

- assume ~~complete~~ compute hash value in O(1)
- assume uniform hashing — each bucket in T is equally likely
- average length of list $\frac{n}{M} = \alpha$

=> α ~~comparisons~~ comparisons on average
=> $\Theta(1+\alpha)$ why not $\Theta(\alpha)$? It's possible that $\alpha < 1$, $= 1$, > 1?
hash to a bucket

Worst case! $\Theta(n)$

Successful Search
- # comparions to find k.
- compute hash value + average search time on average length list.

eg. list of size 4:
Average # searches: $\frac{1+2+3+4}{4} = 2.5$

$= 1 + \frac{1}{2}(\frac{n}{M} + 1) = 1 + \frac{\alpha}{2} + \frac{1}{2}$
$= \Theta(1+\alpha)$

($\frac{n}{m} = \alpha$)

$\frac{1}{2}(\frac{n}{M}+1)$, half list (average) search

Search (average case) : $\Theta(1+\alpha)$
Insert : $O(1)$
Delete : same as search

if $n \in \Theta(M)$
=> average cost are all $O(1)$

Open Addressing - Linear ~~Rodde~~ Probing.

Delete · don't simply delete item.
* must flag as deleted - lazy deletion

Insert : scan until empty or flagged (deleted) ~~slot~~ slot found.
Search : scan until empty (continue if flagged (deleted))
or found

Many deletions => many slots flagged

Adv : · space efficient
· easy to implement

Thm : The average # Probes required to search in hash table of size M with n keys is approx :
· successful search : $\frac{1}{2}(1+\frac{1}{1-\alpha})$ $(\frac{1}{2}(1+\frac{1}{1-\alpha}))$
· unsuccessful : $\frac{1}{2}(1+\frac{1}{(1-\alpha)^2})$ $(\frac{1}{2}(1+\frac{1}{(1-\alpha)^2}))$
for α not too close to 1.

Primary Clustering :
· a sequence of slots gets full
· the bigger the cluster gets, the quicker it continues to grow.
· the larger area it spans
=> larger chance that new ~~item~~ insertions will go in the cluster
=> cluster expands

long search/insert

α	$\frac{1}{2}$	$\frac{2}{3}$	$\frac{3}{4}$	$\frac{9}{10}$
$\frac{1}{2}(1+\frac{1}{1-\alpha})$ hit	1.5	2.0	~~3.0~~ 2.5	5.5
$\frac{1}{2}(1+\frac{1}{(1-\alpha)^2})$ miss	~~2.5~~ 2.5	5.0	~~8.0~~ 8.5	~~50.5~~ 50.5

Double Hashing

Idea: Stop formation of clusters by using 2 independent hash functions.

- if $h_1(k_1) = h_1(k_2)$ collide
  then $h_2(k_1) \neq h_2(k_2)$ unlikely but possible.

eg. $h_1 = k \bmod 11$
$h_2 = \lfloor 10(A \cdot k - \lfloor A \cdot k \rfloor) \rfloor + 1$ — some floating point constant.

$h_1 : U \to \{0, 1, \dots M-1\}$
$h_2 : U \to \{1, \dots M-1\}$

- h_2 is an offset from original hash location

Double hashing: $h(k,i) = [h_1(k) + i \cdot h_2(k)] \bmod M$.

- try $i = 0, 1, 2, \dots$ until find open slot.

$M = 11$ eg $k = 11$ probe sequence: 0, 8, 5, 2, ... linear probing
$k = 22$ 0, 6, 1, 7, ... both are 0, 1, 2 ...

eg $M = 2^{10} = 1024$ Suppose $h_1(k) = 10$ and $h_2(k) = 256$ for some k
probes: 10, 266, 522, 778, 10, 266, 522.
h_2 should be relatives prime to Hash Table size.

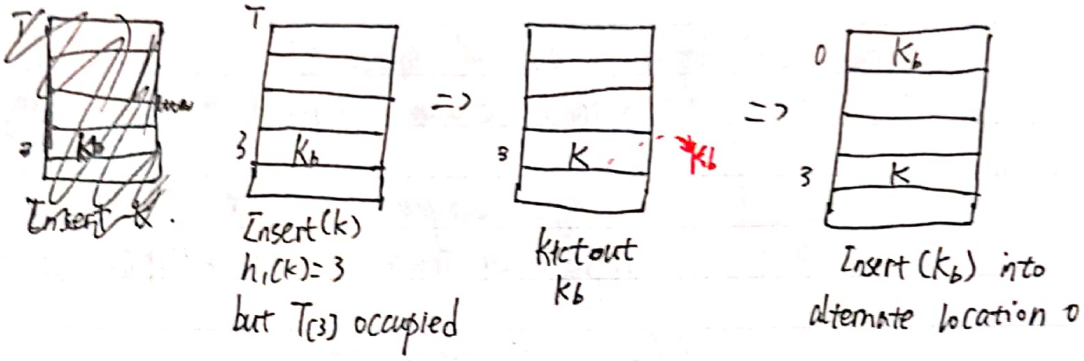
=> no common factors.
=> Choose prime # for M. and $1 < h_2(k) < M$.
- no common factors

Analysis: Average # probes:
- successful: $\frac{1}{2}\ln(\frac{1}{1-\alpha})$
- unsuccessful: $\frac{1}{1-\alpha}$

α	$\frac{1}{2}$	$\frac{2}{3}$	$\frac{3}{4}$	$\frac{9}{10}$
hit	1.4	1.4	1.8	2.6
miss	1.5	2.0	3.0	5.5

Cuckoo Hashing.

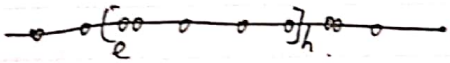
* One KVP per slot
* 2 independent hash functions
* always insert k to $T[h_1(k)]$
  - may "kick out" another item
    - if so, insert "kick out" item at its alternate location



Keys: • letters
• strings "apple" => ~~inside~~ index in hash table
each letter has an ASCII value.

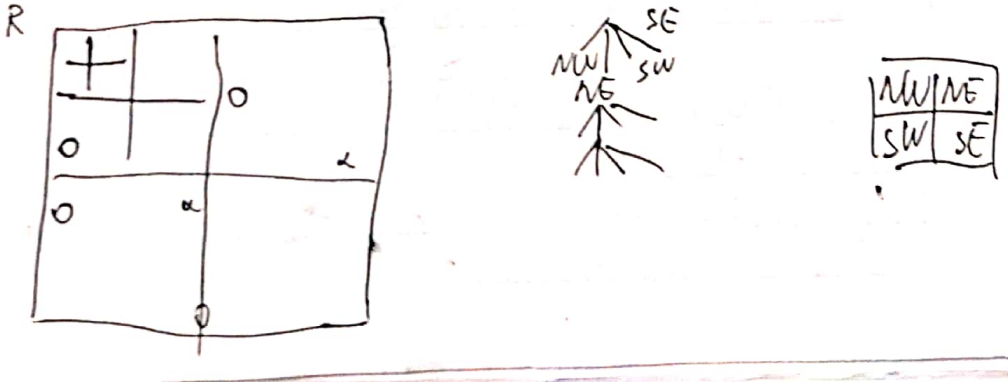
Module 8.

Range Search 1-D



Query [e, h]

- binary search for l — O(log n)
- go right until > h
  - ~~this~~ let k be size of output
    => O(log n + k) * ~~different~~ output sensitive



If point $p \in P$ on x mid, then p in NW ∪ SW

If point $p \in P$ on y mid, then p in SW ∪ SE

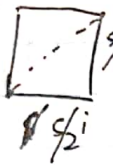
Lemma - Depth of Quadree is at most $\lg(s/c) + \frac{3}{2}$

where · s ($= d_{max}$) is side length between of intial square

· c ($= d_{min}$) is minimum distance between any two points in P.

Proof: at depth i, size of square: $\frac{s}{2^i}$

=> max distance between 2 points in the same region:



length diagnal $\sqrt{2(s/2^i)^2} = \frac{\sqrt{2}s}{2^i}$

· depth i of internal nodes satisfies $\frac{\sqrt{2}s}{2^i} \geq c \Rightarrow \frac{s\sqrt{2}}{c} \geq 2^i$

Solve for i: $\log(\frac{s}{c}) + \frac{1}{2}$

2011. 10.31

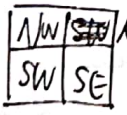
Quadtree

· Define a bounding square => root node

· all points

· if region contains > 1 pt, => each square is a child

subdevide region into 4 squares · if no points in region, do not add child.



convention! left & lower · order matter

· keep subdividing until only 1 pt per region => all points are leaves

· not based on n.

Depth of Quadtree $\leq \log(\frac{d_{max}}{d_{min}}) + \frac{3}{2}$

Build tree: $O(nh)$

Range Search: $O(nh)$ ~ follow branches of tree that intersect quary rectangle

Often performs well in practice.

- easy to implement
- easy to extend to higher dimensions

main problem: based on splitting region equally regardless of distribution of ~~point~~ points.
- Some region will have many points => region many subdivisions.

=> bad performance for non-uniformly distributed points.

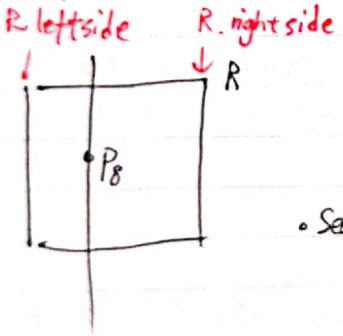
---

kd-tree

Idea: split points instead of by area
=> always ~ half points on each side of dividing line
- alternate between vertical & horizontal divisions.

Convention: "other" points on vertical split line => left
horizontal => ~~right~~ lower

make sure to sort points by x and/or y coord ONLY ONCE!



- Search left and right

search only left

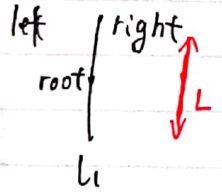
search right only

Analysis: We want better than $\Theta(n)$
Runtime is bounded by # calls to
① reporting a point in R
+ ② recursing on C because $C.region \cap R \neq \emptyset$

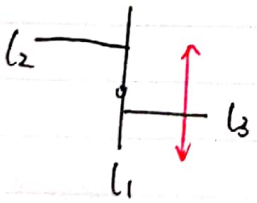
① let k be # points reported $\Rightarrow \Theta(k)$
② $\leq$ # nodes C in T:
$S \cup C.point \in R$
+ $U \cup$ number of regions we ~~the~~ check unsuccessfully
- regions that intersect R but not fully in R.
- points not in R but are close.

$|S| \in \Theta(k)$
Need to ~~boot~~ bound U
$|U| \leq$ # nodes in T such that C.region is intersected by an edge of R

Let L be a fixed vertical line (eg an edge of R)
Let Q(n) be # of nodes in T s.t. C.region is intersected by L

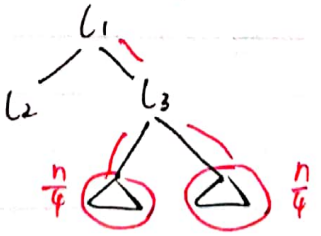


- L intersects exactly one region
- either left or right.



- L intersects both upper & lower region of l_3 but neither for l_2

$\Rightarrow$



$$Q(n) = \begin{cases} O(1) & \text{if } n=1 \\ \Theta(1) + 2Q(\frac{n}{4}) & \text{if } n>1 \end{cases}$$

Solve:

$$Q(n) = 2Q(\tfrac{n}{4}) + \Theta(1) = 2Q(\tfrac{n}{4}) + c$$

a constant c

$$= 4Q(\tfrac{n}{16}) + 2c + c$$
$$= 8Q(\tfrac{n}{64}) + 4c + 2c + c$$
$$\cdots$$
$$= 2^{\log_4 n} \cdot Q(1) + \sum_{i=0}^{\log_4 n} 2^i \cdot c = O(2^{\log_4 n})$$

$$2^{\log_4 n} = 2^{\frac{\log_2 n}{\log_2 4}} = (2^{\log_2 n})^{\frac{1}{\log_2 4}} = (n^{\log_2 2})^{\frac{1}{\log_2 4}}$$
$$= n^{1/2} = \sqrt{n}$$

- Same holds for all 4 edges of R: $O(4\sqrt{n}) \Rightarrow O(\sqrt{n})$

Runtime: $O(k + \sqrt{n})$ where k is number points reported.

2017.11.2

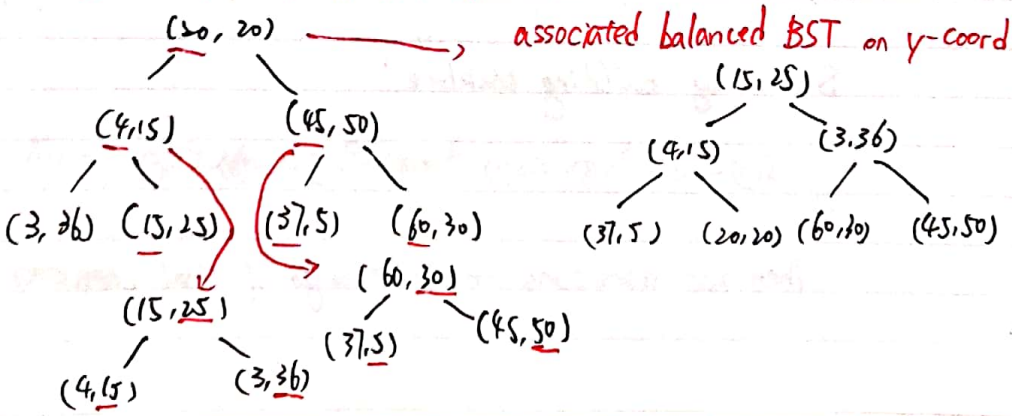
Range Trees 2-D

Let $P(v)$: points stored in subtree rooted at v

- for every initial v, point to an associated BST (balanced) on y-coords storing points $P(v)$

eg. given points: (3,36) (4,15) (15,25) (20,20) (37,5) (45,50), (60,30)

Balanced BST on x-coord



Inside: "top" inside node v
=> perform range search on y-coords for $[y_1, y_2]$ in Yassr(v)
- do not need to check x-coords.

Space: each point is stored in $O(\log n)$ y-BSTs
- in each tree on path up to root

- n points
=> $O(n \log n)$

Runtime: at most 2 logn internal "top" nodes (subtree)
- do range search on y-coords BSTs ~ $O(\log n + k)$ for each.

=> $O((\log n)^2 + k)$

Note: Can be improve to $O(\log n + k)$
Idea: We search the same y range $[y_1, y_2]$ over & over
- only do it once & use the result many times

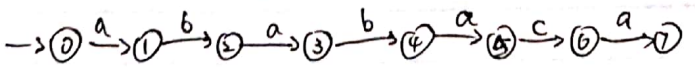
=> fractional cascading.

Pattern matching.

eg. => check $\Theta(m)$ characters
=> $\Theta(n-m+1)$ times
=> total $\Theta(nm)$ usually $m \ll n$.

DFA - given a pattern, the DFA is a pattern matching checker

Start by building backbone:



Then add transitions on where to go if final character found.

• for a string s. $l(s) \in \{0, \dots, m\}$ is the length of the longest prefix of P that is also a suffix of s.

eg. P = ababaca
s = ababaa

• a is the longest profix of P that is a suffix of s.
=> $l(s) = 1$ $\delta(s, a) = 1$
from state s on symbal a goto state 1.

eg. P = ababaca
s = ababab
=> $l(s) = 4$
$\delta(s, b) = 4$
b

T has size n
P has size m
Goal: O(n) runtime in worst case

KMP
- Compare pattern left to right
- when a mismatch occurs shift pattern "intelligently"
  - similar to mismatch transitions in DFA

eg. Guess: i-j check: i

T:	...	b	a	c	b	a	b	a	b	a	a	bc
P:					a	b	a	b	a	ac		
					0	1	2	...				
							j					

Main loop invariant: if checking index j of P with i of T
then P[0..j-1] = T[i-j .. i-1]
- previous characters of P match T.

Case to consider when checking T[i] with P[j]

cases: T[i] = P[j]
T[i] ≠ P[j]

Boundary cases: j=0, j=m-1

While i<n do

case1: T[i] = P[j] and j < m-1
- continue matching => i++, j++.

Case2: T[i] = P[j] and j == m-1
- whole pattern matched
- return i-j, first occurence of P in T

case 3: T[i] ≠ P[j] and j == 0

eg. T: ... cb a b ab ...
P: ~~~~ a b a b c a
j

- mismatch on 1st character of pattern
=> i++, keep j=0

case4: T[i] ≠ P[j] and j > 0

T: .... b a cb a b ab a a cba ...
P: d b ab a c a
0 1 2 3 4 5 6
i
j

- mismatch not at the first character.

Brute Force: go back in text to i-j+1 and start checking pattern at j=0.

KMP - shift pattern more intelligently.
- keep i same
- decrement j by correct amount ← intelligently.

Must maintain loop invariant: P[0,... j-1] = T[i-j, ... i-1]

failureArray(P)

P = abacaba

F[0] = 0
i = 1
j = 0

1 < m
P[1] ≠ P[0] (a≠b)
j = 0 =>
F[1] = 0
i = 2

2 < m
P[2] = P[0] (a=a)
F[2] = 1
i = 3
j = 1

3 < m
P[3] ≠ P[1] (b≠c)
j > 0 =>
j = F[1-1] = F[0] = 0

3 < m
P[3] ≠ P[0] (a≠c)
j = 0 =>
F[3] = 0
i = 4

4 < 7
P[4] = P[0] (a=a)
F[4] = 0+1 = 1
j = 1
i = 5

5 < 7
P[5] = P[1]
F[5] = 1+1 = 2
i = 6
j = 2

6 < 7
P[6] = P[2]
F[6] = 2+1 = 3
i = 6
j = 3

end

How do we update j?

- Without moving to backwards in T · currently at index i
- still need previous characters T[i-1] T[i-2] ... to match pattern

=> we need to choose a maximal $j_{new} < j$ such that:

$P[0 \dots j_{new}-1] = T[i-j_{new} \dots i-1]$

Define F[j-1] as the index we will have to check in P after we bring the pattern to its next possible ~~possib~~position.

i.e. $F[j-1] = j_{new}$

eg: T: ... b a c b a ba b a | a c b a

P: a b a b a c a

0 1 2 3 4 5 6

i

j

j=5 mismatch

F[4] = ?

Find maximal prefix of P[0...j-1]: ababa

that matches suffix of P[1...j-1] ba ba (length 3) * proper suffix

F[4] = 3

P=abacaba

j=5

P[0,...j]: abacab

P[1,...j]: bacab (length 2)

F[5]=2

Confusing part: Suppose there is a mismatch at j=6

Look up F[5], $j_{new} = 2$

Shift pattern to right.

Case 4: T[i] = P[j] and j > 0
- update j = F[j-1]

Last Occurence
- mapping for all characters in alphabet Σ

eg. Σ = {a, b, c, d, ..., z}

c	a	b	c	d	...	z
loc	4	5	3	-1	...	-1

|Σ|

Good suffix Arrays or Suffix skip array S
- similar to KMP failure array + extra condition
- Tougher to understand.

To compute: S[i]
- mismatch occured at P[i]
- Shift pattern to right by positive amount so that
① any pattern characters below already match text characters must match up.
And ② P[j] does not match P[i]
- return j = i - shift.amount.

Find the largest j s.t. P[i+1, ..., m-1] = P[j+1, ..., j+m-1-i] ①

and P[j] ≠ P[i] ②

ex: P = bonobobo (0 1 2 3 4 5 6 7) => m = 8

i = 7 P[8..7] = P[j+1, ... j] => j = 7 but P[j=7] = P[i=7] ✗ try again
Σ - no characters => j = 6 so P[6] ≠ P[7]

i = 6 P[7...7] = P[j+1, ... j+1] => 'o' at index 5
'o' P[j=5] = P[i=6] try again

=> 'o' at index 3
P[2] ≠ P[6] ← j=2

i=5 P[6...7] = P[j+1 ··· j+2] => "bo" at index 4|5, j=3
"bo" P[3] = P[5] try again

=> "bo" at index 0|1, j=-1
P[-1] ✓

i=4 P[5...7] = P[j+1 ··· j+3] => "obo" at 3|4|5 => j=2
"obo" P[2] ≠ P[4] ✓

P = 0 1 2 3 4 5 6 7
b o b o b o b o
i=4 match "obo" 5|6|7
mismatch at 'b' "4"
try: j=2 × "obo" at 3|4|5
j=0 × "obo" at 1|2|3

-4 -3 -2 -1 0 1 2 3 4 5 6 7
× × × × b o b o b o b o

j=-2 ✓

Rabin-Karp Fingerprint Alg

Idea: don't compare string
- compare hash values of string instead
fingerprint

Don't explicitly create the hash table
- but need prime numbers (table size) for hash function.

Recall: Good Hash function properties

=> if 2 hash values are the same, they are likely the same key.

Idea: Find hash value for Pattern $h(x) = x \bmod 97$

P: 59265
h(p) = 95

- only compute this once
- then use h(p) to find possible matches.

Search Text:

- computing hash values for strings of similar length to P

Graph①: T: 3 1 4 15 9 2 6 53...

compute hashvalue

"rolling hash"
- keeps on shifting right by 1 character

- compute hash for substring in text compare with h(p)
  - if match: "probably" the string we are looking for
    - use native approach to varify match $\Theta(m)$

Pattern has length m => $\Theta(m)$ to hash pattern.
If we must compute a hash for all possible entries of text.
- n-m+1 possible places to check for a match.

=> $\Theta(nm)$ to search for a miss.

Robind karp found a way to roll hash more efficiently.

2017.11.1F

Update: how to compute the next hash value of text from previous.

Suppose Initial value is 41592 mod 97 = ~~96~~ 76

$\underbrace{4\times10^4}_{①} + \underbrace{1\times10^3 + 5\times10^2 + 9\times10^1 + 2\times10^0}_{②}$

=> Graph①

If the next hash is: 15926 mod 97

41592
- 4×10^4 - O(1)
- *(15926) by 10 ~ O(1)?
- + 6 - O(1)
- mod 97 - O(1)

or to keep numbers smaller
work with mod values
$10^4 \bmod 97 = 9$
old hash value: $(76 - (4*9))*10 + 6 = 406 \bmod 97$
=> 18

Work with binary keys - strings of {0,1}

$H(T[i+1,\dots,i+m]) = H(T[i,\dots i+m-1])*2 - T[i]*2^m + T[i+m] \bmod M$ / bitwise operations

Table size:

- prime number
- big enough so collisions are likely

When do we get collisions?

- $x = y \Rightarrow H(x) = H(y)$
- Also can have $H(x) = H(y)$ but $x \neq y$

iff M devides $|x-y|$ => naive check of keys = failure, no match

---

From Number Theory: # primes that devide $A < 2^n$ is $\frac{n}{\ln n}$

Pick $M = \approx n^2$ "large enough"

size of text

then Prob{failure} $\frac{n}{\ln n} \cdot \frac{\ln n^2}{n^2} \doteq \frac{2}{n}$

if $n = 2000$, Prob{failure} $\leq 10^{-3}$

Module 10

Reliability · very important for hard drives, signal transmission

=> error correcting codes

- store & send some extra info to check if packet arrived ok

eg. transmit 7 bit characters: 0110100, 1101100, 0101110

parity bit { 1 if odd # of 1s
0 if even # of 1s

=> 0110100 1
=> 1101100 0
=> 0101110 0

Consider the following code:

E=1010
S=11
O=1011
Y=01
N=0110

- all codes are distinct
- ask the Oracle a question
  It will give you an encoded answer

Will I pass CS 246?
Answer: 01101011
Y E S

0110 1011
N O

2017.11.16

Huffman coding

- Decoding Dictionary - a binary trie
  => left branch: 0
  right branch: 1

encoding :

- From tree, build dictionary of letter-codes : $\Sigma \to \{0,1\}^*$

␣ : 000
N : 001
A : 01
O : 100
E : 101
T : 11

- AN␣ANT : 0|001|0000|00111
  - * Output : is a steam of bits!
    - * No delimits

need 2 bit per char in a fixed length approach

Compression Ration "LOSSLESS" - 4 different characters $4 = 2^2$

S = LOSSLESS

C = 0|00|1110|0|0011

$\Sigma_S = \{E, O, L, S\}$ $|\Sigma_S| = 4$ $\log_2^4 = 2$

$\Sigma_C = \{0, 1\}$ $|\Sigma_C| = 2$ $\log_2^2 = 1$

$\frac{|C| \cdot \log|\Sigma_C|}{|S| \log|\Sigma_S|} = \frac{14 \cdot 1}{8 \cdot 2} = \frac{14}{16} = 88\%$

* English Text - reduced by up to 50%

RLE:

- Binary length of k, $len(k) = \lfloor \lg k \rfloor + 1$
  - encode $len(k) - 1 \Rightarrow \lfloor \lg k \rfloor$

Number k	$\lfloor \lg k \rfloor$	Binary rep of k.
1	0	1
RLE 1		
3	$\lfloor \lg 3 \rfloor = 1$	11
=> 011		
5	$\lfloor \lg 5 \rfloor = 2$	101
=> 00|01		
23	$\lfloor \lg 23 \rfloor = 4$	10111
=> 0000|0111		

Indicate what first bit is : 0 or 1, then it alternates runs

...00100... = 4

#100

Expansion: encoded version is bigger than the source run.

11.2

Compel-Ziv-Welch (≥W.

- fixed length code
  - typically 12-bits $2^{12}=4096$ possible patterns can be encoded.
  - Dictionary will have 2^{12} entries

Dictionary

- adaptive - non static
- start with an initial dictionary D_0 such as all ASCII values - fixed codes
- will add new entries as we encode $D_0 \Rightarrow D_1 \Rightarrow D_2 \Rightarrow \ldots$
- as we find common patterns in the given input, add them to dictionary
- Remember, after we encode the input, it will also need to be decoded.
  - Dictionary will be rebuild by decoder

Advantages

1-pass → add patterns to dictionary as we are encoding

- streaming, we encode on the fly
- for later "text" we will hopefully have better codes to use - get better compression.

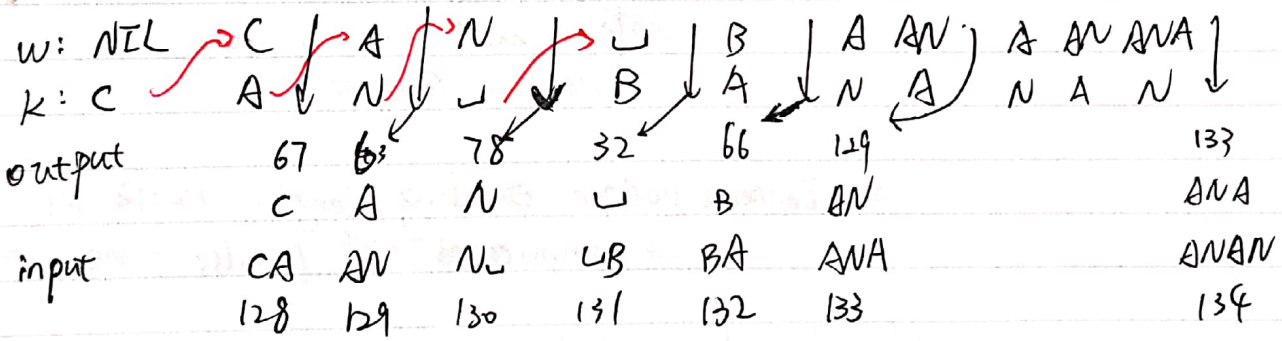
- for large files, space for encoder ~~and~~ decoder is small
- typically better compression than Huffman

Disadvantage:

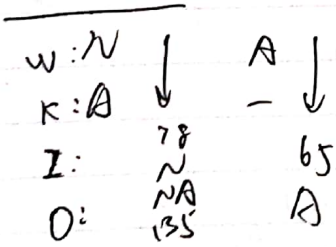
- early text of input will not have much compression
  - generally better than ASCII codes

eg. from 22: Y O ! Later YO YOU
stored as 12-bits 79 89 33 Still 12-bits => 128 132
of binary

CAN-BANANA



Final 67 65 78 32 66 129 133 78,65
=> each stored as 12 bits of binary



Decoding:

- Decoder is "one step behind"
  - Encoder added w with look ~~forward~~ ahead char k
    but decoder doesn't know k until one step later

LZW

Decoding: • any dictionary implementation will do

Encoding • need to find longest prefix of s in D
• trie insert & lookup in time proportional to length of string read from input

What if Dictionary fills up?

1. Stop adding • fast
• bad if data changes structure

2. Clear and start over with only ASCII • temporary poor compression again

3. ~~Discar~~ Discard less frequently used patterns
• may be expensive
⇒ find min value
store a counter

4. Increase pattern encoding length 12 → 13 bits
o Complicated but possible - new codes are longer

Note: Decoder must do exactly the same thing as encoder.

MTF

• Note: Encoder & Decoder are very similar
=> they must ~~update~~ update the Dictionary in some way

---

n characters: $\lceil \log_2 n \rceil$ bits to represent n characters
if alphabet is ASCII, need 7-bit for each # in C
⇒ same as each letter for c.

prefix-free integer encoding

- good because small integers take fewer bits than large integers.

Huffman Coding

- good because we might expect small integers to appear more frequently.

Burrows-Wheeler Transform

- permutes input, but is reversible

① cyclic shift

the␣quick ... dog $ ← end-of-word character.
he␣quick...dog$t * assume this has lowest ASCII value
e␣quick...dog$th so '$' < '␣'
␣quick...dog$the
⋮

lexicalgraphic Order: compare characters, if tie. check next one.

eg. A = aaab => tie, tie, tie. 'b' < 'c'
B = aaac A comes before B.

C = asff$f␣e␣lllaaaata

eg S = bubba$

bubba$
ubba$b
bba$bu
ba$bub
a$bubb

① List all cyclic shifts

=>

$ b u b b a
a $ b u b b
b a $ b u b
b b b $ b u
b u b b a $
u b b a $ b

② sort lexicographically

=> C = abbu$b

③ extract last character of each row

Decode: what do we know? How to fill colume 1

0	1	2	3	4	5
$					a
a					b
b					b
b					u
b					$
u					b

s	0	1
a	$	b
b	a	$
b	b	a
u	b	b
$	b	u
b	u	b

Colume 2?

s	0	1
a	$	b
b	a	$
b	b	a
u	b	b
$	b	u
b	u	b

stable sort =>

0	1	2
$	b	u
a	$	b
b	a	$
b	b	a
b	u	b
u	b	$

Decode Alg

j = index of $ in C
S = empty string
j = N[j]
while C[j] ≠ $
  append C[j] to S
  j = N[j]

C = abbu$b
0 1 2 3 4 5

A = (a,0) (b,1) (b,2) (b,3) ($,4) (b,5) => stable sort

$, 4	N
a, 0	0 4
b, 1	1 0
b, 2	2 1
b, 3	3 2
b, 5	4 5
u, 3	5 3

j = 4 5 3 2 1 0 4
S = ε b a b b a $

Ex 2 S = banana$

banana$		$banana
anana$b		a$banan
nana$ba		ana$ban
ana$ban	=> Sort	anana$b
na$bana		bana$an
a$banan		nana$ba
$banana		na$bana

c = annb$aa

Decode: c = annb$aa

0 1 2 3 4 5 6

		N	
A =	a 0	$0 4	j = 4 3 6 2 5 1 0 4
	n 1	a1 0	
	n 2	a2 5	s = $ b a n a n a $
	b 3	a3 6	
	$ 4	b4 3	
	a 5	n5 1	
	a 6	n6 2	

=> stable sort

Decode c = nkoe op m$

2017. 11. 28

photo in phone.

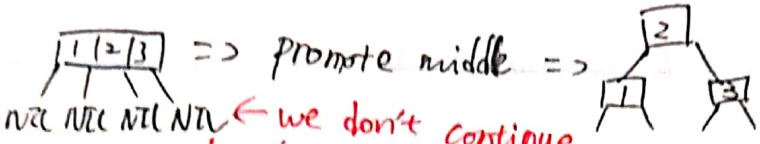
2-3 trees

- each non-leaf node: 2 or 3 children (1 or 2 kVP)
- all leaves are at the same level

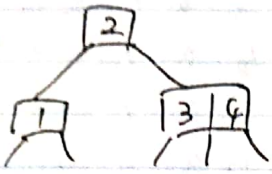
Insert

- find lowest internal node
- if overflow, promote middle

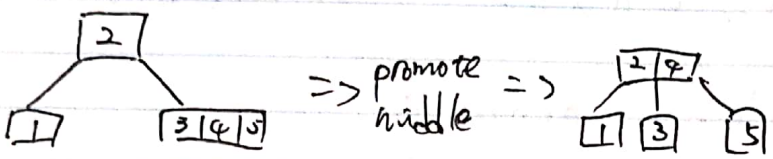
Insert 1 2

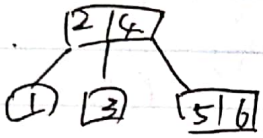
insert 3: 

← we don't continue to show this but remember the links are there.

insert 4: 

=> height of tree grows iff root overflows.

5: 

6. 

7. 

Delete

- Swap with inorder successor and delete
  - if no underflow, down
  - else · first try transfer from immediate sibling
    - If transfer not possible, merge.

---

tree decrease in height iff root node underflow.

2017.11.30

B-tree of order M
- each node has at most M-1 KVP (M-children)
- each non-root node has at ~~most~~ least $\lceil M/2 \rceil - 1$ KVPs ($\lceil M/2 \rceil$ children)

Suppose Hard drive sector has size 200 bytes
- KVP is 12 bytes
- child pointer (64-bit OS) 8 bytes

How many KVPs per sector?

$m \cdot 12 + (m+1) \times 8 \le 200$

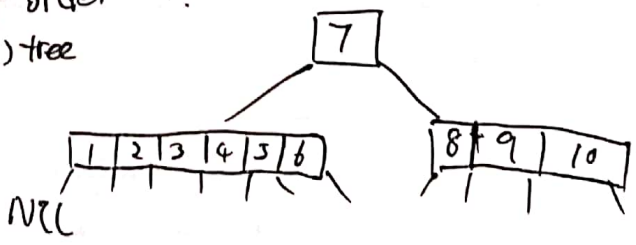
$(m-1)$ m

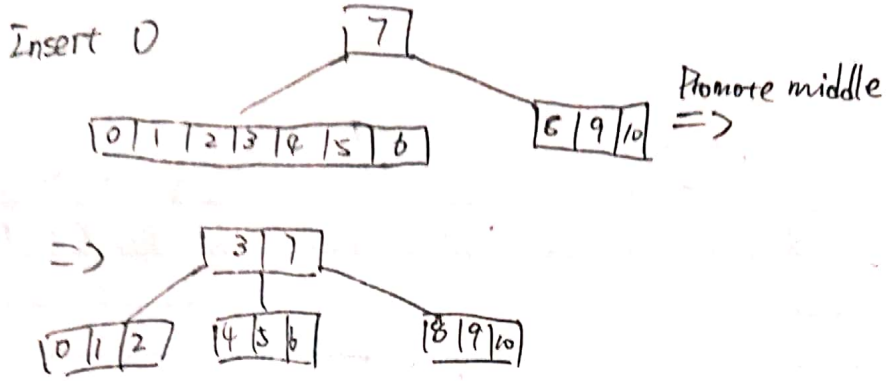
find m
$m = 9$ KVPs
or 10 children

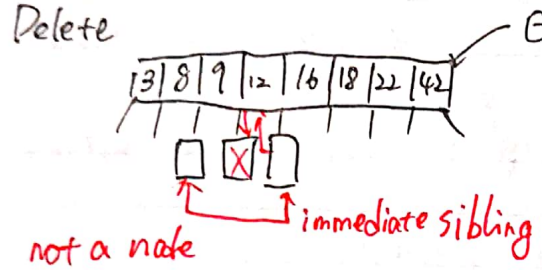
(5, 10) Tree of order 10

B^+
B^*
- Search is easy
- * insert may involve splitting => might increase height
- * delete may involve transfer from immediate sibling or merge (possibly repeated all the way up to root) => may decrease the height

B-tree of order 7.
(4, 7) tree







$\Theta(M)$ kVPs
$O(\log M)$

$$\text{Search} = \left(\frac{\log n}{\log M} \cdot (\log M)\right) = O(\log n)$$

Extendible hashing

- hash table doesn't fit ~~in~~ in RAM → store on hard drive

Which hashing method might be Okay?

- cuckoo & double hashing — jump all over the hash table
  - => each one could be a hard drive access
- none so far

Idea: want to fill up entire sector

- similar to B-tree
- similar to radix-sort - use bits to split up the data into tables.

Use a tree of height 1

=> directory stored in RAM

- root node
- contains hash table of size 2^d ("order d") — some power of 2

=> each entry of ~~directo~~ directory points to a block on HD

Hash function h: Key -> hash value $= \{0, 1, \dots 2^L - 1\}$ for some $L > 0$.

- represent hash value in binary as a bit string
- use part of hash value to look up in directory to determine the block on HD where key is stored

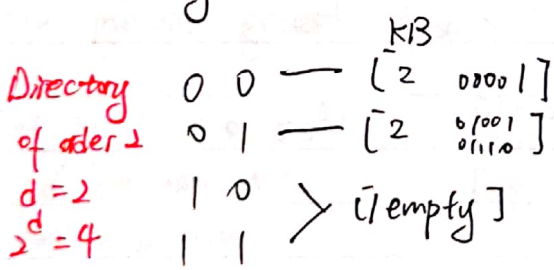
=> "extendible": grows and shrinks as needed.

Block on HD
- each block contains at most S items · stored by hash value.

S - Size of block # items per block
d - order of directory
K_B - local depth of block B

- Directory stored in RAM
- Block stored in HD
- Hash values: $\{0, 1, \dots 2^L - 1\}$
  - bit strings of length L.

Idea: Store keys: 00001, 01001, 01110.



$L = 5$ # bits in hash value
$S = 2$ # items per block
- block stores $\leq S$ items.

$d = 2 \leq L$ "order d"
- directory indexed by first d bits of hash values.

↑ pointers to blocks
- pointers may go to the same block
- directory has 2^d pointers to blocks

Local block depth
- "local depth" of each block B is $K_B \leq d$
- n_B is # items currently stored in Bblock B · $n_B \leq S$

Search
- Hash (K)
- use first d bits to look up in directory
- follow pointer to corresponding block B

- ~~look~~ Load block into RAM
- Search B for matching key (hash(K))
  ↑ eg. binary search

Ex $L=5$, $S=2$ 2 entries per block allowed
Insert 01001

$d=0$
$2^0=1$ [ ] – [K_B: 0 | 01001]

- order 0
- one block
- $K_B = 0$

Insert 00001
[ ] – [0 | 00001, 01001]

~~Insert~~
Insert 01110
- no room =>
- But $K_B = d$ => double ~~hashing~~ directory size

$d=1$ 0 – [K_B: 1 | 00001, 01001]
$2^1=2$ 1 – [1 | empty]

where does 01110 go?
- no room and $K_B = d$
=> split block and double directory

$d=2$ 00 – [2 | 00001]
01 – [2 | 01001, 01110]
10 \
11 – [1 | empty]

Exercise: 1