Thursday, October 15, 2009
3:31 PM

Midterm: min: 18
avg: 59
max: 91

① copy constructor / copy assignment

↓ new object
3 copies from another obj,
3 to initialize

↓ copies from one obj
to another but 3
dest. obj already exists

② $A3 = A1 + A2$

$A3.\text{operator}=(A1.\text{operator}+(A2))$

3 3 3

③ $O(T(n)) = F(n) \mid \exists c, n_0 : T(n) \leq cF(n),$ 2 2 2

$\forall n \geq n_0$ 2

2 upper bound $F(n)$

2 worst case

④ 3 $\Theta(T(n))$: average running time

3 $\Omega(T(n))$: lower bound (min. running time)

3 $\Theta(T(n)) = \Omega(T(n))$ on average, alg. performs optimally.

3 $O(T(n)) = \Omega(T(n)) \Rightarrow$ alg. always runs optimally (has been optimized)

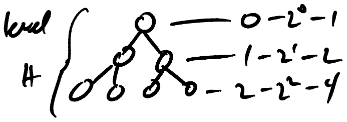
⑤ Alg. 1: optimal on average,
degenerated to quadratic time worst case
5 e.g. binary search tree

Alg. 2: quadratic time in both avg. & worst cases
5 e.g. bubble sort

Alg. 3: average case better than its best?
5 impossible
e.g. N/A

5 Alg. 4 optimized alg.
e. AVL

⑥ 2^i nodes per level



$$\underbrace{\sum_{i=0}^{H} 2^i = 2^{H+1} - 1}_{12}$$

leaves: nodes at bottom, or at level H

2^i at each level, so 2^H at leaf level

4

⑦ Strategy A

insert every item

$O(\lg n) \times n$

$= O(n \lg n)$

$+$

extract $(O(\lg n)) \times n$ each item

$= O(n \lg n)$

$O(\underbrace{n \lg n} + n \lg n)$

$\in O(n \lg n)$

Strategy B

toss $O(1) \times n = O(n)$

$+$

FixHeap $O(n)$ once $\times 1$

$= O(n)$

$+$

extract $O(\lg n) \times n$

$= O(n \lg n)$

$O(\underbrace{n + n} + n \lg n)$

$\in O(n \lg n)$

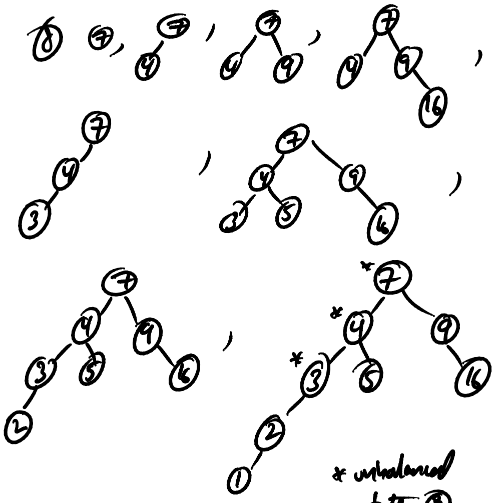
$2n < n \lg n$

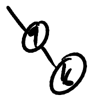
$n \lg n > 2n$

so B is just slightly better

but in same class

(both $O(n \lg n)$)

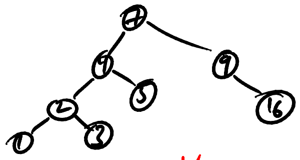




* unbalanced

rotate ③

8



4

Thursday, October 15, 2009
3:52 PM

Chp. 5 Hashing

- the data structure talked about here is the hash table

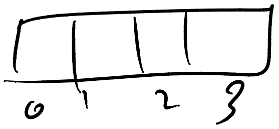
- basic array, one that supports insertion, deletion, find in $\Theta(1)$ (constant average time)

e.g. practical example:
Unix file system
insertion: mkdir, vi newfile
deletion: rm
occasionally: rehash

- findMin, findMax & sort are less efficient
- how to implement insert & find in $\Theta(1)$?
  - given key (data element), e.g. number compute its index into array via hash function (fancy formula)

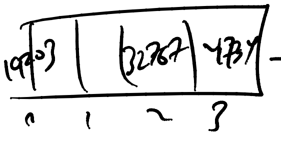
Note: size of hash table << no. of elements

e.g.



hash table of size 4

given keys 19203, 32767, 41734, would like to map these keys (values) to unique elements



- sooner or later you get collisions

1995		3210	497
0	1	2	3

- sooner or later you get <u>collisions</u>

a good hash function ? %

"compress" maps large range of numbers into small ones

mod function

hash fn: element % 4

for table of size 5, element % 5 would be ok except if all elements were multiples of 5

- so, good idea is to keep table size a prime number (1, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, ..., 10,007) how to generate ?

something like

```
for(i=1, i<n, i++)
  for(j=1, j<n-1, j++)
    if( i/j == 0)
      not a prime
```

??

- then element % prime number would be a good hash function

- text gives good hash function for string keys

$$h = \sum_{i=0}^{\text{keysize}-1} \text{key}[\text{keysize}-i-1] * 37^i$$

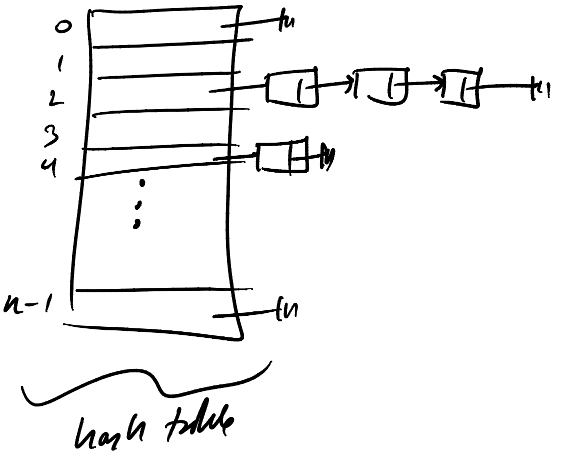
via Horner's rule

$$h_k = k_0 + 37k_1 + 37^2k_2$$

$$= ((k_2)*37 + k_1)*37 + k_0$$

```
h=0
for(int i=0; i< key.length(); i++)
    h = 37*h + key(i)
h = h % tableSize
if(h < 0) h+= tableSize
```

- what's left is collision resolution
- at each array position, keep a linked list of elements



- class HashTable

```
{
    private:
        vector<list<T> > elements;
}
```

using STL linked list

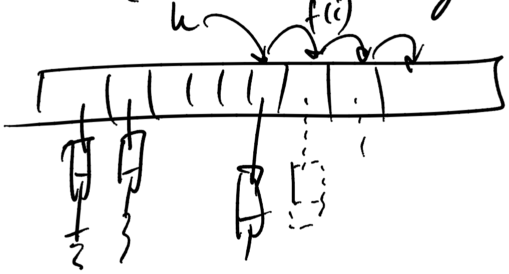
notice space (>> is input op.)

- so far, fairly decent ADT
- problem: linked lists can get long

linear probing : $h_i(x) = hash(x) + f(i)$

where $f(i) = i$

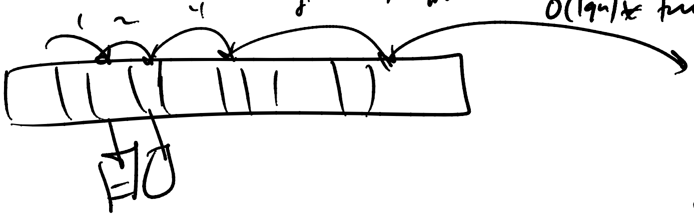
(advance down array linearly)



quadratic probing

$f(i) = i^2$ — 1, 2, 4, 16, ?

Analysis: time to search $O(\log n / x)$ time



to jump $\textcircled{x}$ $O(?)$

for find() upon landing, may need to search list

basic find() problem

$O(n)$ – keep list sorted?

– binary search on list?

↓

$O(\lg n)$

– analysis of hashing can be complicated sometimes dependent on distribution of data — can get into queuing theory, Poisson distributions, arrival times, networks, etc.

- analysis is mainly done on average case
- other variants:
  - double hashing: hash the results of first hash
  - why not having trees instead of linked lists?