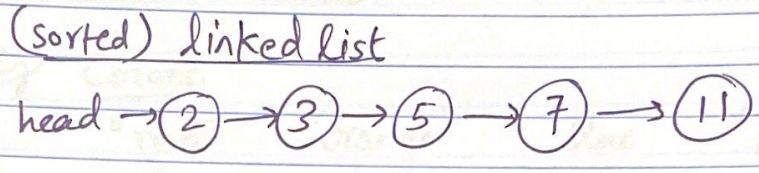
Lecture 18: Hashing and Hash Tables

Previously: (sorted) sets

1. (sorted) linked list



all ops $O(n)$

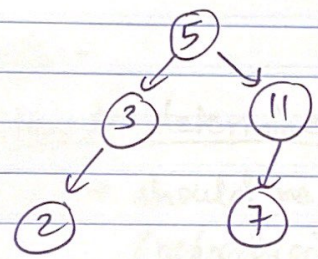
2. (Sorted) array

0	1	2	3	4
2	3	5	7	11

find in $O(\log n)$ binary search

other ops $O(n)$

3. (balanced) BST:

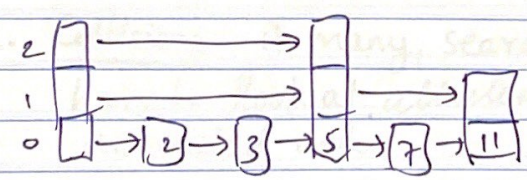


All ops in $O(h)$ (h = height)

$\Rightarrow$ unbalanced $\rightarrow h = O(n)$

$\Rightarrow$ balanced (AVL) $\rightarrow h = O(\log n)$

4. Skiplists



probablistic guarantees

Expected running times for ops $O(\log n)$

average over random choices of algorithm, not user.

⇒ <u>Idea:</u> use randomness to assign random numerical values to elements, "sort" according to values.

eg <u>Colors:</u>

"red" → 2, "orange" → 6, "blue" → 1, "pink" → 3.

Sort colors according to these values!

"blue" < "red" < "pink" < "orange"

Use sorting for Sorted Set implementation

<u>ISSUES:-</u>

1. <u>How to determine order from colors?</u>
   → should be deterministic so get same # (reproducible) from each color name.
   → assign values to chars in name of color?

2. <u>Collisions</u> – if many, searching is inefficient b/c have to look at all elements with same associated value.

3. <u>Semantic Equivalence</u> – need to associate same value to o & p if (o.equals(p) == true).

<u>Analogy:</u> Balls in Bins (Assignment 5)

objects — assigned values

"red" ⟶ 2

<u>Suppose:</u> associate a "random looking" value (an int) to each object wrt property:

1) getVal(obj) → gives value associated to obj;

2) values appear random – hard to find correlation between getVal(o) and getVal(p)

3) Values reproducible: multiple calls to getVal(o) will be same.

4) getVals respects semantic equivalence.
→ o.equals(p) then getVal(o) == getVal(p)

Q Given getVal method,
how to implement Unsorted set as efficiently as sorted set?

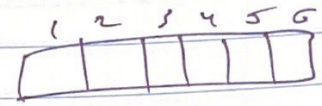
<u>Possibility:</u> Store pair of <obj, getVal(obj)>
↑ make comparable using this

Store pairs, in say, AVL tree.
→ add, find, remove in O(logn) time
(assume getVal is O(1)).

Q Can we do better?
– faster?
– simpler?

⇒ <u>Suggestion</u>: Have array, use getVal(o) to determine index at which O should be stored.

Example	values
"red"	5
"orange"	1
"yellow"	4
"green"	2
"blue"	3
"violet"	3



add("red") → getVal("red") = 5
add("blue") → ... = 3
find("green") → 2
add("violet") → oops!!

⇒ <u>Dealing with collisions</u>:

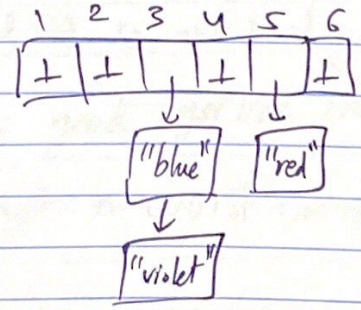
"Chaining" each index of array refers to linked list of elements

→ <u>add</u>: search for elt:
1. go to associated index
2. search list
3. append element if not found

→ <u>find</u>: do 1 & 2 above.

→ <u>remove</u>: 1 and 2; remove elt from list if found.

★ <u>prev. example</u>:



Each index of array refers to head of linked list.

~~do~~

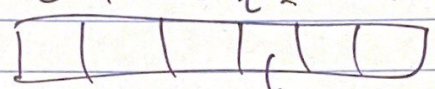
- to do op(x).
  - find, add, remove
  - perform that operation on list at index getVal(x)

⇒ add("green") ← go to index 2

⇒ remove("blue") { - go to index 3.
  - remove node from list

⊛ <u>Question</u>: If array of size (n) storing n elements, what is the expected time to find

arr: 0 1 - i - n-1



occupancy of $arr[i] = n_i$

$$E(\text{occupancy}) = \sum_{i=0}^{n} (\text{occupancy of } i)\, P(\text{get}(i))$$

$$= (n_0 + n_1 + \dots + n_{n-1}) \frac{1}{n} = n \cdot \frac{1}{n} = 1$$

<u>Conclusion</u>: expected time find/add/remove is O(1)

<u>Issue</u>: getVal(o) is not truly random.

→ defining good getVal methods is an art form

→ don't get n-worst case guarantees.