Class Notes for November 02, 2021

Agenda

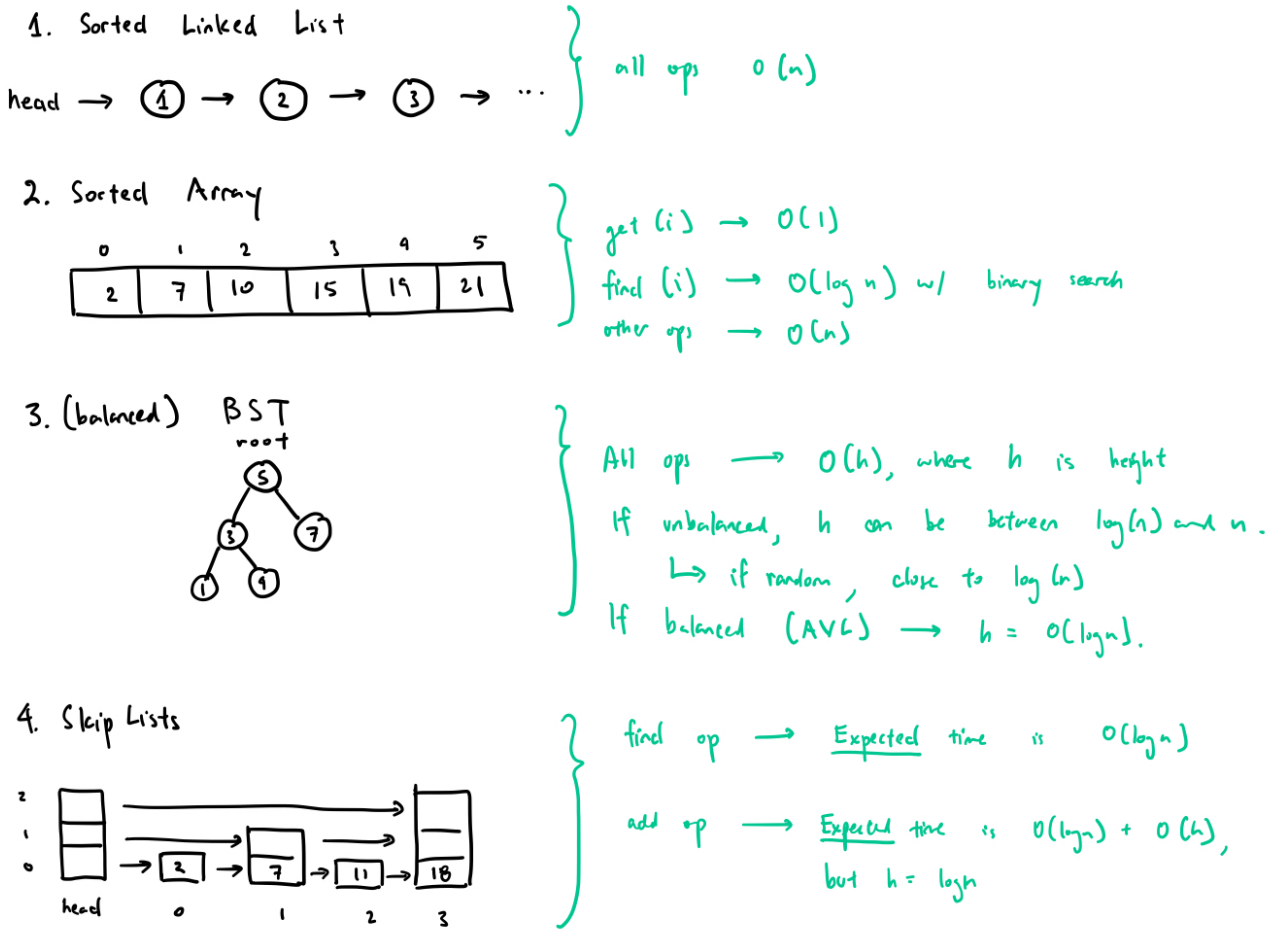
- Hashing and Hash Tables

Announcements

- Midterm 2 : Moved to next week, will be take home
- Final Exam : also take home (there is scheduled time if you want on the last Friday of Finals week @ 9:00am)

Hashing and Hash Tables

Summary of Previous Sorted Sets



Limitations

- fundamental to all of these implementation is that we are able to *sort* elements. That there is some ordering of elements.
- how can we store elements that can't be sorted?

  Idea: Use randomness to assign random numerical values to elements, and then "sort" according to these values.
- example : lets try mapping colors to random numbers
  - [`red`, `orange`, `blue`, `pink`]
  - [2,6,1,3]
  - the ordering would then be `blue` < `red` < `pink` < `orange`, and now we can sort according to the random values we chose
- we can then use this ordering with some sorted set implementation

Issues:

- how do you determine the orders from the colors?
  - say we add `brown`, we'd have to check all colors to see what the value is. Then we're essentially looping through a whole list and having $O(n)$ operations
  - we ideally want the ordering to be deterministic so we get the same value from each color name.
  - if we want a deterministic result, we could maybe convert each character into a number, and then concatenate them!
  - but then what if we have very large names, or non-strings (like objects)
- how do we give things random values to avoid collisions (repetitions of values)?
  - if there are lots of collisions, then we have to spend alot of time to look at all elements with the same value
- we might have semantic equivalence issues (ex: we need `Blue` and `blue` to be associated to the same value)
- we can relate this to the balls in bins assignment that showed us that collisions happen very quickly, but the expected number of collisions is predictable

Suppose we associated a "random looking" value (an integer) to each object instance with the properties:

1. `getVal(obj)` gives value associated to object
2. values appear to be random - it is hard to find correlation between `getVal(0)` and `getVal(p)`
3. values are reproducible : multiple calls to `getVal(0)` will be the same
4. `getVal()` respects semantic equivalence, ie if `0.equals(p)` then `getVal(0) == getVal(p)`

Question : Given `getVal` method, how could we implement an unsorted set as efficiently as a sorted set?

- store a pair of <object, `getVal(obj)`>, and compare on the output of `getVal`
- store these pairs in an AVL tree, and then we get all operations in $O(\log(n))$ time assuming that `getVal` runs in $O(1)$

Can we do better? faster? simpler?

- have an array and use `getVal(x)` to determine the index at which `x` should be stored.

  - example with colors

  - suppose we have `red`; `orange`; `yellow`; `green` ; `blue`; `violet`

  - if we `add(red)`, we just do `getVal(red) = 5`, and then store red at index 5

  - the issue is if we have a collision, (like we get the same value for `violet`), we could maybe just store it like a Skiplist

  - **chaining** each index of an array to refer to a linked list of elements

  - `add` : search for the element

  1) go to associated index

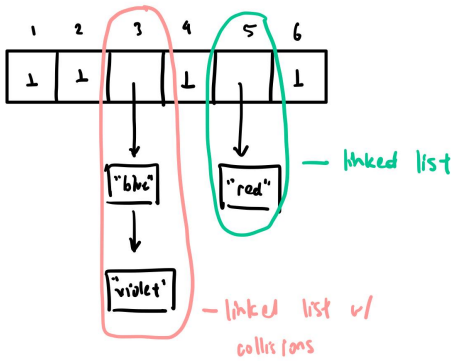
  2) search list

  3) append element if not found

  - `find` : simply do 1 + 2 as above

  - `remove`: do 1+2 as above, but remove element from list if found

  - the hope is that if our array is large enough, the number of collisions will be very small



find (violet):
- getVal("violet") = 3
- go to array @ index 3
- then search the linked list of ["blue"; "violet"]

Question: if an array of size `n` stores `n` elements, what is the expected time to find?

- if we throw `n` balls into `n` bins, the longest "list" (list of collisions) will be `log(n)`
- the expected occupancy is going to be `1`

$$E(n) = n_0 * P(0) + n_1 P(1) + \ldots + n_{n-1} * P(n-1) *$$

$$E(n) = (n_0 + \ldots n_{n-1}) 1/n$$

$$E(n) = (n) * 1/n = 1$$

conclusion: Expected time to `find` / `add` / `remove` is $O(1)$!!!!

- this falls if we expect there to be lots of collisions
- the next step is to figure out how to define the `getVal` function, because it needs to be truly random, but that is difficult!

This data structure is called a hash table and the function `getVal` is a hash function, and the general process of assigning values to objects is called hashing.