CS 32 – Discussion 1B

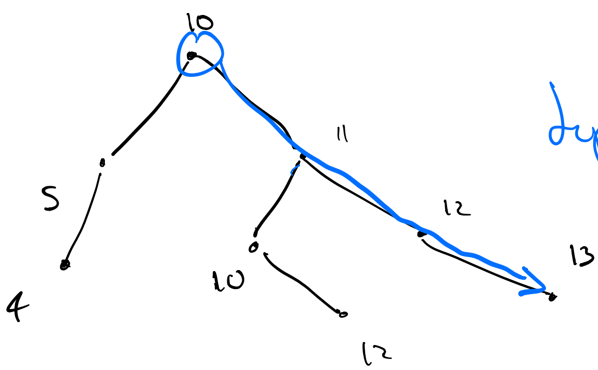
Week 8

Binary Search Trees (BSTs) and Hash Tables

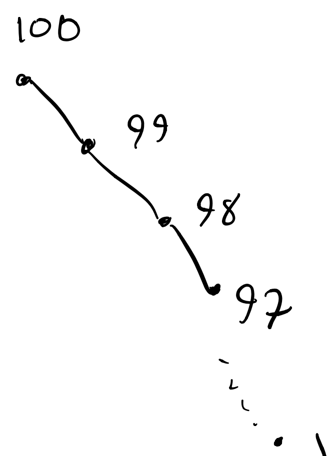
Basics of BSTs:

Binary Tree with a certain property:

> for every node p:
> - values in left subtree are ≤ value of p
> - values in right subtree are ≥ value of p



depth = 4



Cost of insertion / deletion / lookup : $O(\text{depth}(T))$

- if T is well-balanced : $O(\log n)$
- if T extremely unbalanced : $O(n)$

There are good ways to make sure BST stays balanced : e.g. red-black trees, 2-3 trees

Structure of BST allows us to do certain tasks efficiently:

```
displayInOrder (Node* p)
    if p==nullptr : return
    displayInOrder ( p -> left)
    cout p->val ;
    displayInOrder ( p -> right)
```

Cost: $O(n)$

STL container classes implemented with BSTs:

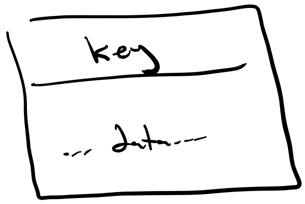
set, multi-set, map, multi-map

Q: Suppose we don't need to maintain order information. Can we design a data structure with $O(1)$ cost of deletion/insertion/lookup?

Yes

Hash Tables:

Setup: Want to store many Entries



① Allocate a large array A (size n)
(n buckets)

② Design a "hash function"

$h : \{keys\} \longrightarrow \{1, 2, 3, 4, \ldots\}$

∞

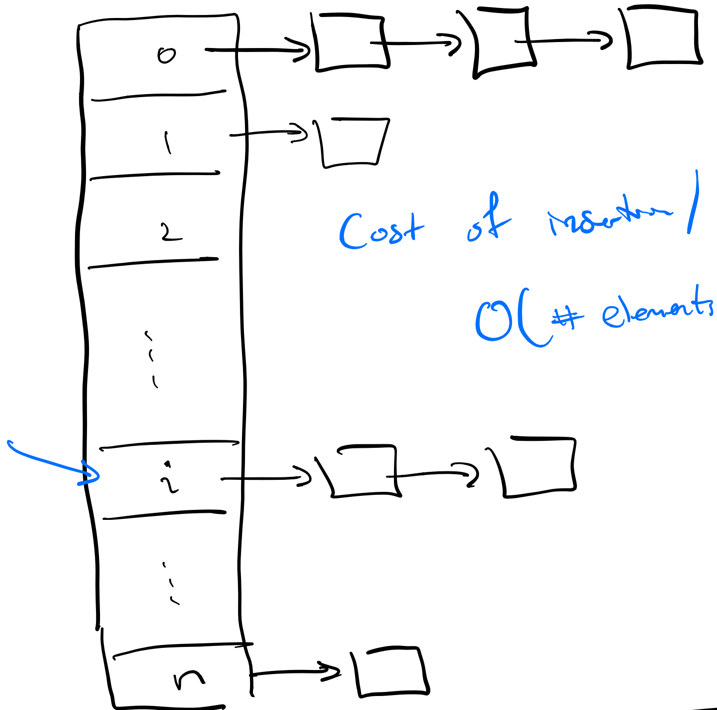
Entry e:
e gets assigned to index $(h(e.key) \bmod n)$

Things with key value "key" go to $A[h(key) \bmod n]$

Collision: $key1 \neq key2$ & $h(key1) \bmod n$
$= h(key2) \bmod n$

Fix: A is an array of lists.
$A[i]$ is the list of all entries e in hash table
for which $h(e.key) \bmod n = i$

A $\qquad$ $i = h(e.key) \bmod n$



Cost of insertion / deletion / lookup @ bucket i

$O(\#\text{ elements in Bucket } i)$

$$\boxed{\text{Key to hash tables being efficient is a hash function which distributes keys as uniformly as possible.}}$$

Typically, we can achieve $O(1)$-sized buckets.

STL classes that use hash tables :	unordered_set unordered_multi-set unordered_map unordered_multi-map