UCLA Samueli Computer Science

CS32: Introduction to Computer Science II

Discussion Week 9

Junheng Hao, Arabelle Siahaan

May 31, 2019

Announcements

UCLA Samueli Computer Science

- Homework 5 is due 11PM next Thursday, June 6. (@A@)
- Project 4 is also due 11PM next Thursday, June 6. (@A@ >_< @A@)

Outline Today

UCLA Samueli Computer Science

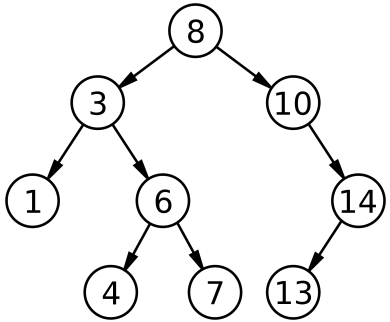
- Binary Search Tree
- Heap (Preview)
- Hash Tables
- Project 4 Guidelines

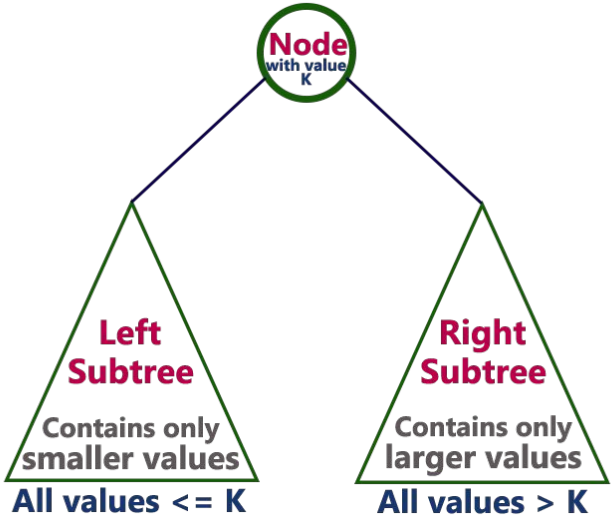
Binary Search Tree

Definition, Properties

UCLA Samueli Computer Science

- The left subtree of a node contains only nodes with keys lesser than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- The left and right subtree each must also be a binary search tree.





Binary Search Tree

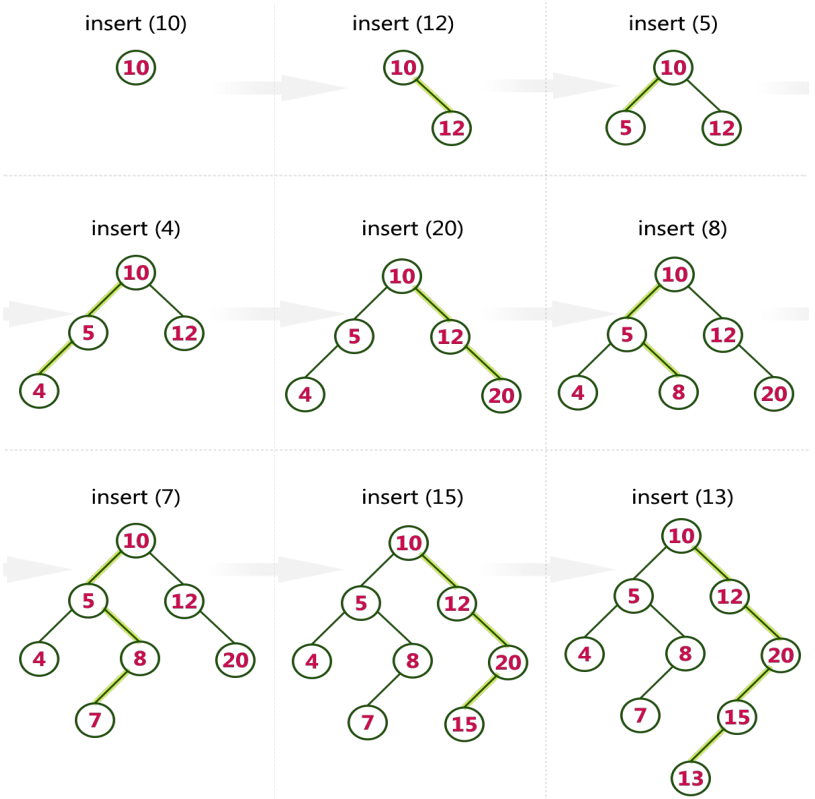
Insertion

UCLA Samueli Computer Science

```
node* insert(node* node, ItemType key)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(key);

    /* Otherwise, recur down the tree */
    if (key < node->key)
        node->left  = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);

    /* return the (unchanged) node pointer */
    return node;
}
```



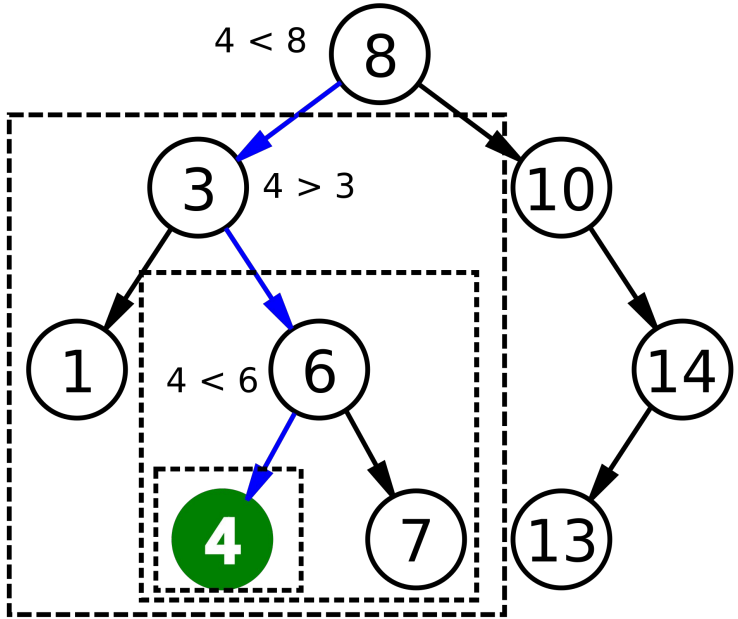
Binary Search Tree

Search

UCLA Samueli Computer Science

```
node* search(node* node, ItemType key)
{
    /* If the tree is empty, return null pointer */
    if (node == nullptr) return nullptr;

    /* compare with current node and decide*/
    if (key == node->key)
        return node;
    else if (key < node->key)
        return search(node->left, key);
    else
        return (node->right, key);
}
```



Binary Search Tree

Deletion

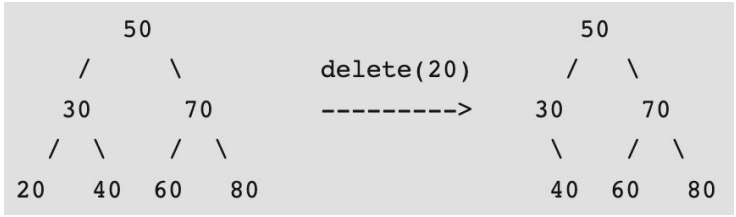
UCLA Samueli Computer Science

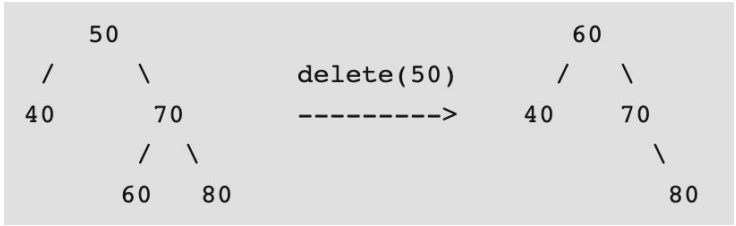
- Deletion is the most tricky one.
- 3 cases:
  - The node is a leaf node.
  - The node has one child.
  - The node have two children.

Super easy! Just delete that node!

Not difficult! Copy the child to the node and delete the child.

Find inorder successor of the node. Copy contents of the inorder successor to the node and delete the inorder successor.
Note that inorder predecessor can also be used.





Binary Search Tree

Deletion

UCLA Samueli Computer Science

```
node* delete(node* node, ItemType key)
{
  if (node == nullptr) return nullptr;
  if (key < node->key) { node->left = delete(node->left, key); }
  else if (key > node->key) { node->right = delete(node->right, key); }
  else{
    /* case 1 & case 2*/
    if (node->left == nullptr) { node *temp = node->right; delete(node); return temp; }
    else if (node->right == nullptr) {node *temp = node->left; delete(node); return temp;}
    /* case 3 */
    node* temp = minValueNode(node->right);
    node->key = temp->key;
    root->right = deleteNode(root->right, temp->key);
  }
}
```

Binary Search Tree

Analysis of BST

UCLA Samueli Computer Science

- Insertion
  - The (worst) case time complexity of search and insert operations is O(h) where h is height of Binary Search Tree. In worst case, we may have to travel from root to the deepest leaf node.
  - The height of a skewed tree may become n and the time complexity of search and insert operation may become O(n).
  - Average time complexity: O(log *n*)
- Deletion
  - Similar to insertion for complexity analysis

Binary Search Tree

FindMin and FindMax by using recursion

UCLA Samueli Computer Science

How to implement **`findMaxKey`** and **`findMinKey`** function?

We assume the root is not `nullptr`.

```
node* findMaxKey(const node* node)
{
  /* only need to check the right subtree*/
  if (node->right == nullptr) return node->key;
  return findMaxKey(node->right);
}
```

```
node* findMinKey(const node* node)
{
  /* only need to check the right subtree*/
  if (node->left == nullptr) return node->key;
  return findMinKey(node->left);
}
```

Binary Search Tree

FindMin and FindMax by using recursion

UCLA Samueli Computer Science

Question: How to test a tree is a valid BST?

One possible recursion solution by using **`findMinKey`** and **`findMaxKey`**.

```
bool isValidBST(const node* node)
{
  if (node == nullptr) return true;
  /* check left subtree and right subtree condition*/
  if (node->left != nullptr && findMaxKey(node->left) > node->key)
    return false;
  if (node->right != nullptr && findMinKey(node->right) < node->key)
    return false;
  /* further check subtree with left child and right child */
  return isValidBST(node->left) && isValidBST(node->right)
}
```

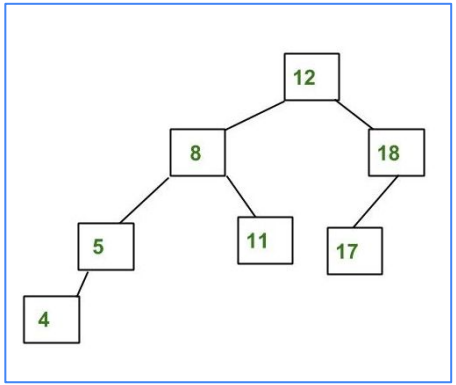
What is the complexity of this `isValidBST()` function?

Beyond Binary Search Tree

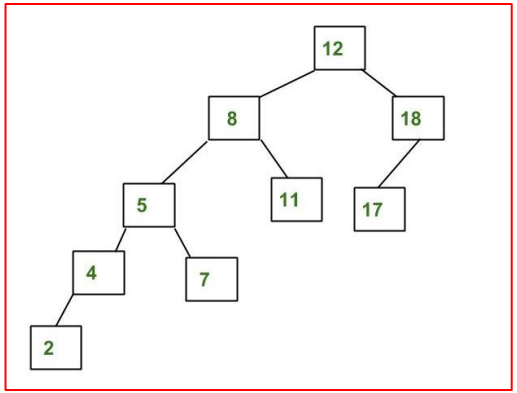
AVL Tree

UCLA Samueli Computer Science

- What is the drawback of naive BST? → It can be skewed! Not good!
- AVL (Adelson-Velsky and Landis) Tree is a self-balancing BST.



This is OK!



Just OK is not OK!

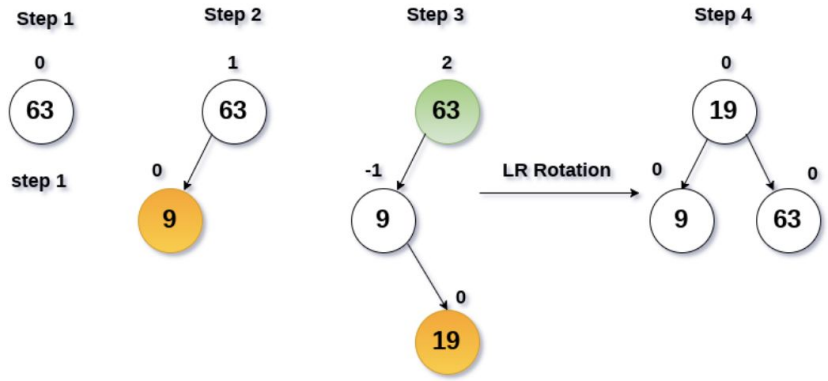
Beyond Binary Search Tree

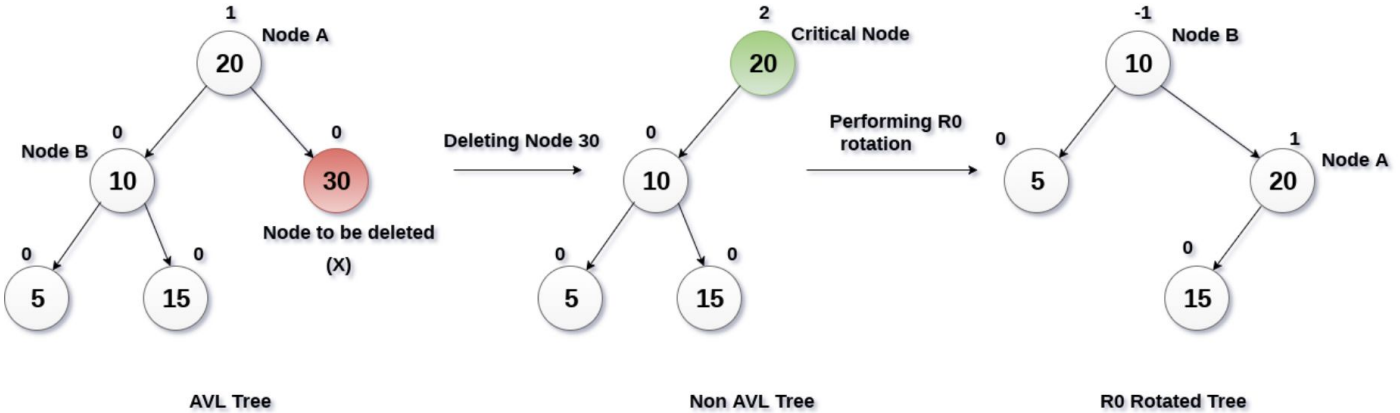
AVL Tree: Operations

UCLA Samueli Computer Science

- Insertion
- Deletion

Please check this interesting demo!





Beyond Binary Search Tree

The tree family (1)

UCLA Samueli Computer Science

There are many interesting tree structures such as:

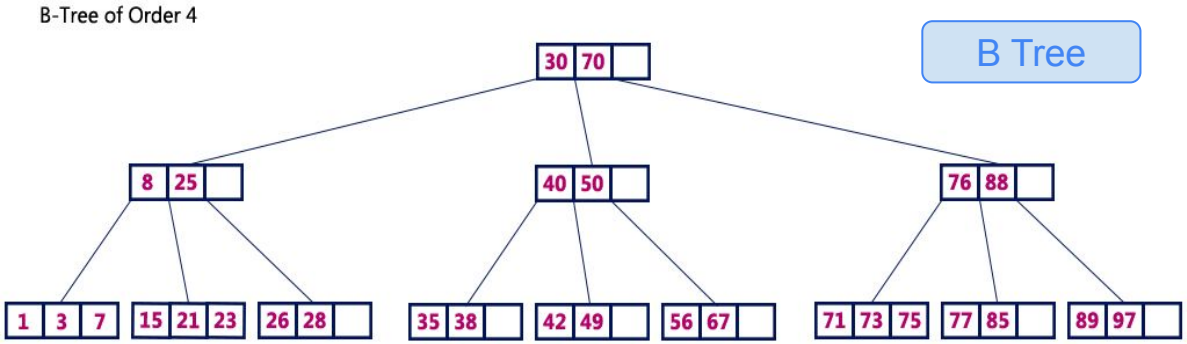
- B tree and B+ tree (I did not hear about A+ tree though)
- 2-3-4 tree
- R tree (spatial index tree)
- **Red-black tree**
- K-D tree

Please check the given links for more details!

Beyond Binary Search Tree

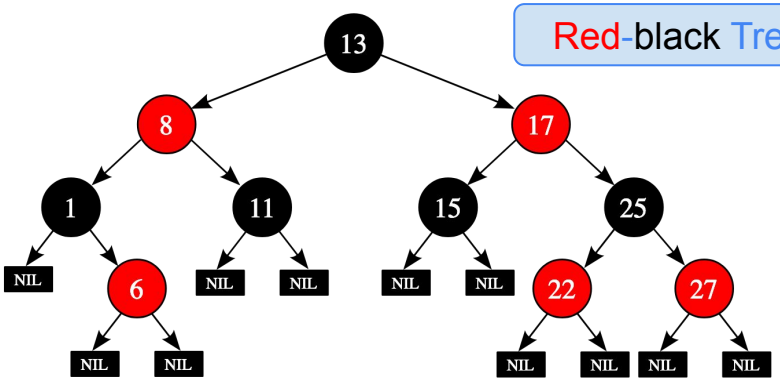
The tree family (2)

UCLA Samueli Computer Science

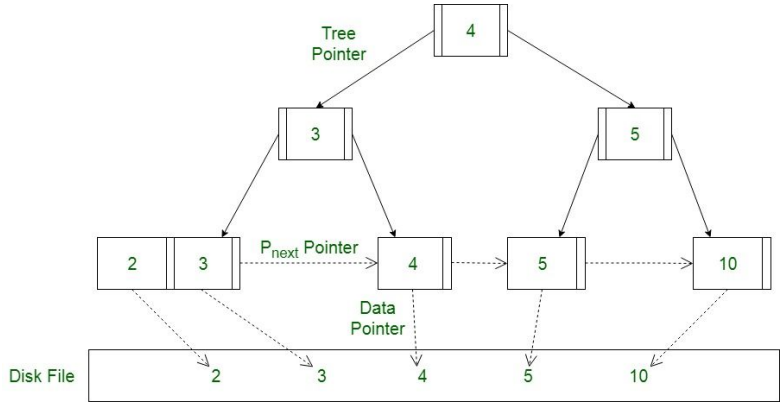


B Tree

B+ Tree



Red-black Tree

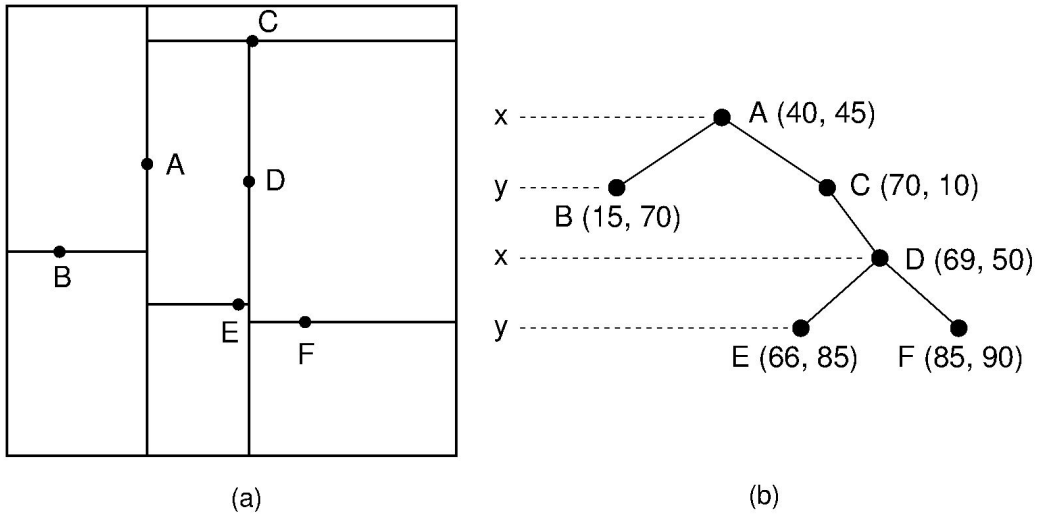


Beyond Binary Search Tree

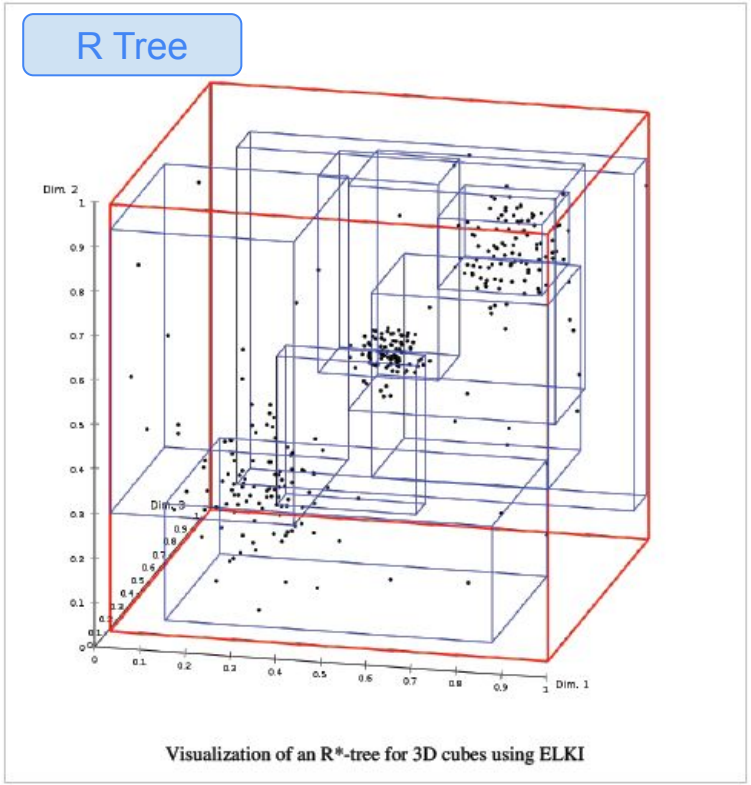
The tree family (3)

UCLA Samueli Computer Science

KD Tree



R Tree



Visualization of an R*-tree for 3D cubes using ELKI

Heap (preview)

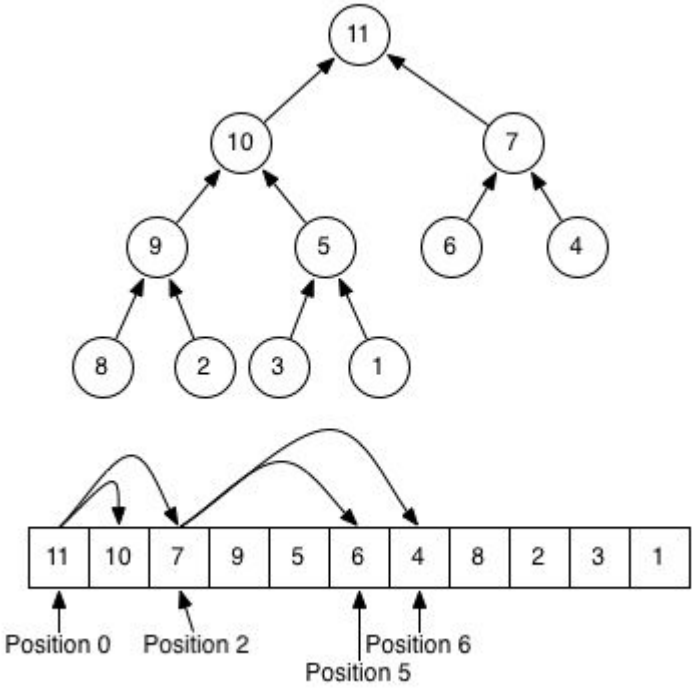
Definition and properties

UCLA Samueli Computer Science

- About heap
  - Heap is considered as complete binary tree.
  - Every nodes carries a value greater than or equal to its children (for MaxHeap).
  - Often implemented as an array.
  - Body structure of priority queue.

Stack

Heap



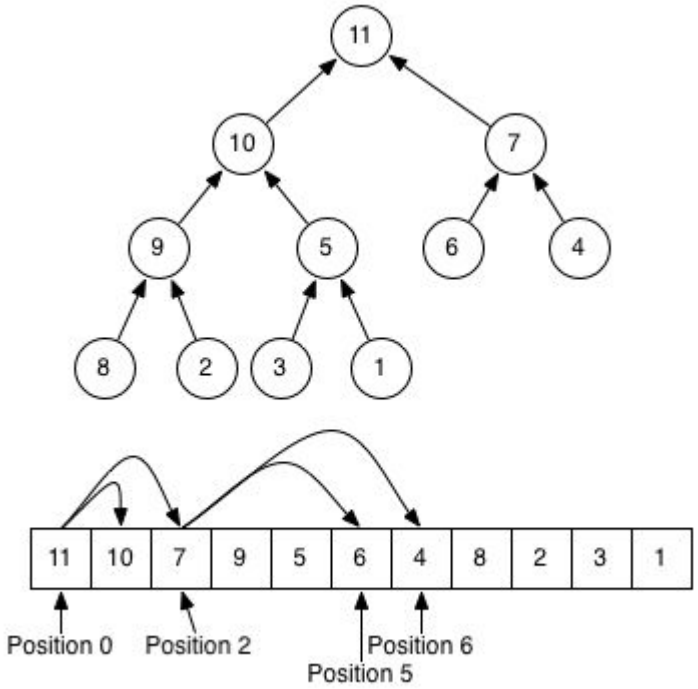
Heap (preview)

Standard operations

UCLA Samueli Computer Science

- Three operations of heaps
  - Find Max (search)
  - Insert Node (insert)
  - Delete Max (delete)

- How to implement `FindMax()` function of a heap?
  - Well, that is just too obvious!



Heap & Heapsort (preview)

Complexity of heap operations

UCLA Samueli Computer Science

- Find Max → O(1)
- Insert → O(log *n*)
- Delete Max Node → O(log *n*)

6 5 3 1 8 7 2 4

- Bonus: How can you sort based on heap?
  - Insert all elements into a heap.
  - Extract the maximum element from the heap one by one.
  - Check the example in Wikipedia
- What is the complexity of heapsort?
  - $O(n \log n)$

Hash Tables

Start from hashing and hash functions

UCLA Samueli Computer Science

- Hash functions: Take a “key” and map it to a number
- Requirement for hash function: should return the same value for the same key
- Good hash functions:
  - Spreads out the values: two different key are likely to results in different hash values. → Avoid confliction
  - Compute each value quickly.
- Example: FNV-1


```
unsigned int FNV-1(string s) {
  unsigned int h = 2166136261U;
  for (int k = 0; k != s.size(); k++)
  {
    h += s[k];
    h *= 16777619;
  }
  return h;
}
```

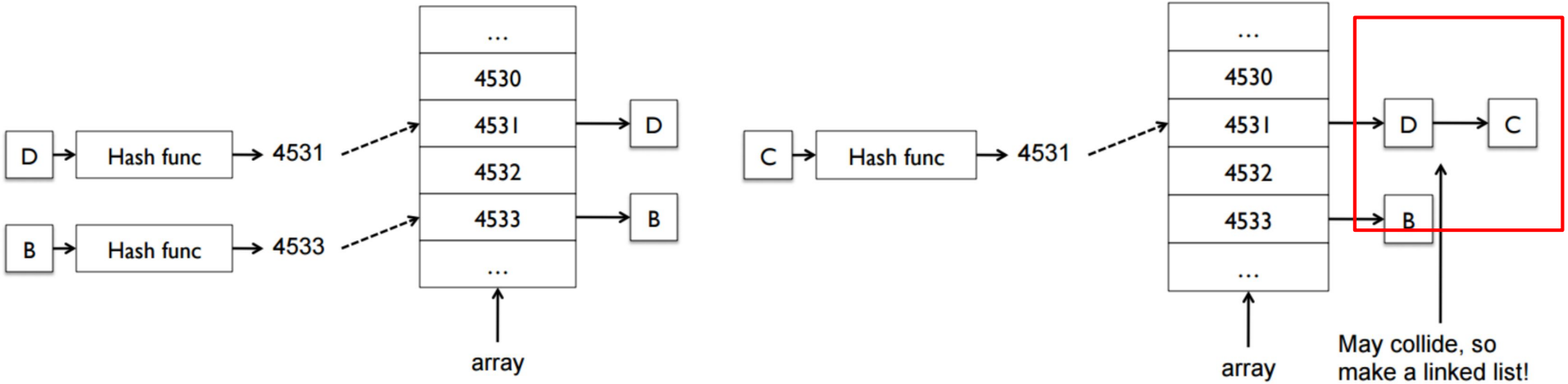
Hash Tables

Examples

UCLA Samueli Computer Science

- Example: Use a hash table to store people.
- Use a linked list to collision in the hash function.

*"You should almost **NEVER** assume that collisions are impossible!!!" --David Smallberg*



Hash Tables

Operations

UCLA Samueli Computer Science

- Insert
- Remove
- Search

- The complexity depends on your hash tables.
- Closed Hashing
  - Fixed number of buckets
  - All operations are **O(n)** with a small constant of proportionality
- Open Hashing
  - Consider *#entries / #buckets*
  - Almost **O(1)** for all operations

Hash Tables Problem 1

Word counting and sorting in a document

UCLA Samueli Computer Science

- Question: We have ***n*** words in a document, whose vocabulary size is ***v***. Count top k frequent words in a document.

Two steps: counting + sorting

- The most efficient way to count the frequency for all words takes **O(**n**)** time complexity.
- After getting the frequency of each word, the most efficient way to get the top k frequent words takes **O(**$v \log k$**)** time complexity.
- Totally the entire procedure takes **O(**$n + v \log k$**).**

Hash Tables Problem 2

The very first question in LeetCode - TwoSum

UCLA Samueli Computer Science

- Given an array of integers, return indices of the two numbers such that they add up to a specific target.
- You may assume that each input would have exactly one solution, and you may not use the same element twice.
- Example: Given `nums = [2, 7, 11, 15]` and `target = 9`. Because `nums[0] + nums[1] = 2 + 7 = 9`, return `[0, 1]`.

Hash Table in STL

Map, multimap, unordered_map, HashMap

UCLA Samueli Computer Science

- Useful functions of STL `map`, `multimap`, `unordered_map`
  - `size(), begin(), end(), empty()`
  - `insert(keyvalue, mapvalue)`
  - `find()`
  - `operator[]`
- Internal implementation:
  - `map/multimap`: Red-black tree
  - `unordered_map`: Hash table
- Pros and cons between `map` and `unordered_map`

Hints for Project 4

UCLA Samueli Computer Science

Note & Reminders:

1. Get File I/O done first.
2. Build BST or hashtable by `vector`, `list`, `stack` and `queue` (and `string`) instead of `map` or `unordered_map`.
   - Reason to build BST or hashtable: fast indexing the substrings
   - For complexity analysis
3. Algorithm improvements → Shorten the diff file

UCLA Samueli Computer Science

Break Time! (5 minutes)

Q & A

Group Exercises: Worksheet

UCLA Samueli Computer Science

- Exercise problems from **Worksheet 8** (see “LA worksheet” tab in CS32 website). Answers will be posted next week.
- Questions for today:
  - 1, 2, 4, 6

Question 1

UCLA Samueli Computer Science

Given an array arr[0..n-1] of distinct elements and a range [low, high], use a hash table to find all numbers that are in the range, but not in the array. Print out the missing elements in sorted order. Use the function header:

```
void inRange(int arr[], int size, int low, int high)
```

For example:

```
Input:     arr[] = {10, 12, 11, 15}, low = 10, high = 15
Output:    13, 14

Input:     arr[] = {1, 14, 11, 51, 15}, low = 50, high = 55
Output:    50, 52, 53, 54
```

Question 1: Solution

UCLA Samueli Computer Science

```
#include <unordered_set>
#include <iostream>
using namespace std;

void inRange(int arr[], int size, int low, int high)
{
    // Insert all elements of arr[] in set
    unordered_set<int> set;
    for (int i=0; i<size; i++)
        set.insert(arr[i]);

    // Traverse through the range and print all
    // missing elements
    for (int x=low; x<=high; x++)
        if (set.find(x) == set.end()) //or if (set.count(x) == 0)
            cout << x << " ";
}
```

Question 2

UCLA Samueli Computer Science

Given an array of integers and a target sum, determine if two integers in the array can be added together to equal the sum. The time complexity of your solution should be O(n) where n is the number of elements in the array. In other words, you cannot use the brute force method in which you compare each element with every other element using nested for loops. Example:

```
Array:   4   8   3   7   9   2   5
Target:  15
```

You can take 8 and 7 from the array, and their sum equals the target of 15. Thus, the function we will write will return true.

Use the following function header:

```
bool twoSum(const int arr[], int n, int target);
```

Question 2: Solution

UCLA Samueli Computer Science

```
bool twoSum(const int arr[], int n, int target) {
    unordered_set<int> numsFound;

    for (int i = 0; i < n; i++) {

        int complement = target - arr[i];

        if (numsFound.find(complement) != numsFound.end()) {
            return true;
        }
        else {
            numsFound.insert(arr[i]);
        }
    }
    return false;
```

Question 4

UCLA Samueli Computer Science

Given a string, find the first non-repeating character in it and return its index. If it doesn't exist, return -1. You may assume the string contain only lowercase letters. Use a hashtable to solve this problem.

Use the function header:

```
int firstUniqueChar(std::string s)
```

Example:

```
s = "hello"                  // return 0
s = “computer science”       // return 1
```

Question 4: Solution

UCLA Samueli Computer Science

```
int firstUniqueChar(std::string s) {
         // Map character to the frequency of occurrence
         unordered_map<char, int> counter;
        for(int i = 0; i < s.size(); i++) {
            counter[s[i]]++;
        }
        for (int i = 0; i < s.size(); i++) {
            if (counter[s[i]] == 1)
                  return i;
        }
        return -1;
}
```

Question 6

UCLA Samueli Computer Science

Implement the following function, given the following data structure:

```
struct Node {
   int val;
   Node* left;
   Node* right;
};

bool isMinHeap(const Node* head);
```

This function takes in the head of a binary tree and returns whether or not that binary tree represents a binary min heap. In other words, this tree must follow the min heap property, where a parent is less than or equal to its children.

Question 6: Solution

UCLA Samueli Computer Science

```
// Precondition: head points to a complete tree
bool isMinHeap(const Node* head) {
  if (head == nullptr)
    return true;

  Node* left = head->left;
  Node* right = head->right;
  if (left != nullptr && head->val > left->val)
    return false;
  if (right != nullptr && head->val > right->val)
    return false;

  return isMinHeap(head->left) && isMinHeap(head->right);
}
```

UCLA Samueli Computer Science

Thank you!

Q & A