Lecture 1

Instructors: Subrahmanyam Kalyanasundaram
Karteek Sreenivasaiah

3rd September 2020

About the course

- This is the standard undergraduate course to learn nontrivial data structures

About the course

- This is the standard undergraduate course to learn nontrivial data structures
- Mode:
  - For some topics (most initial topics), we shall point you to available video lectures, with online meetings for Q&A.
  - For the other topics, we shall make available pre-recorded video lectures, with online meetings for Q&A.

About the course

- This is the standard undergraduate course to learn nontrivial data structures
- Mode:
  - For some topics (most initial topics), we shall point you to available video lectures, with online meetings for Q&A.
  - For the other topics, we shall make available pre-recorded video lectures, with online meetings for Q&A.
- We will use Google Classroom and emails to communicate. Everyone is expected to enrol in Google Classroom.

Evaluation

- Programming Assignments: Around 60%
- Short Quizzes: Around 20%
- Descriptive Exams: Around 20%

We are figuring out appropriate modes to do each of the above. This will be communicated as and when necessary.

Topics to be covered

- Binary Search Trees. Balanced Binary Search Trees like AVL and Red-Black Trees.
- Connection between Randomized Quicksort and Insertion in BST.
- (2,3,4)-Trees, and extension to B-Trees.
- Heaps. Binary Max Heaps and Min Heaps.
- Graphs. Graph Data Structures. Breadth First Search Traversal. Shortest Path and Minimum Spanning Trees.
- Disjoint Set Data Structure.
- Selected advanced topics from: Amortized Analysis, Skip Lists, Hashing, Binomial/Fibonacci Heaps.

Plagiarism

- Plagiarism means copying work (either from another person or from the internet) of another person and passing it as your own.
- Your submissions are expected to be your own.
- We are very strict and will have a zero tolerance policy on plagiarism.
- If detected, you will receive an F grade for the course, and potentially other penalties.
- Check out https://cse.iith.ac.in/academics/plagiarism-policy.html

Runway Reservation System

- Airport with a single busy runway (Mumbai)

Runway Reservation System

- Airport with a single busy runway (Mumbai)
- Reservation requests for future landings
  - Need to land at time t

Runway Reservation System

- Airport with a single busy runway (Mumbai)
- Reservation requests for future landings
  - Need to land at time t
- We can approve landing request if no other landing within k minutes

Runway Reservation System

- Airport with a single busy runway (Mumbai)
- Reservation requests for future landings
  - Need to land at time t
- We can approve landing request if no other landing within k minutes
- Once approved, we can add t to the set R of landing times

Runway Reservation System

- Airport with a single busy runway (Mumbai)
- Reservation requests for future landings
  - Need to land at time t
- We can approve landing request if no other landing within k minutes
- Once approved, we can add t to the set R of landing times
- Remove t from the set after plane lands

Runway Reservation System

- Let $|R| = n$
- Ideally, all the operations to be done in $O(\log n)$ time

What are the options?

- Unsorted List/Array:

What are the options?

- Unsorted List/Array:
- Almost every operation is $O(n)$, except insertion

What are the options?

- Unsorted List/Array:
- Almost every operation is $O(n)$, except insertion

- Sorted Array:

What are the options?

- Unsorted List/Array:
- Almost every operation is $O(n)$, except insertion

- Sorted Array:
- Search is $O(\log n)$, but insertion is $O(n)$

What are the options?

- Unsorted List/Array:
- Almost every operation is $O(n)$, except insertion

- Sorted Array:
- Search is $O(\log n)$, but insertion is $O(n)$

- Sorted List:

What are the options?

- Unsorted List/Array:
- Almost every operation is $O(n)$, except insertion
- Sorted Array:
- Search is $O(\log n)$, but insertion is $O(n)$
- Sorted List:
- Insertion is $O(1)$, but search is $O(n)$

What are the options?

- Hash Tables:
- Heaps:

What are the options?

- Hash Tables:
- Heaps:

- Not good for search in a range

What are the options?

- Hash Tables:
- Heaps:

- Not good for search in a range

Fast insertion into a sorted array

Set

A set has supports the following features:

- INSERT(*val*) – Inserts *val* into the set.
- SEARCH(*val*) – Search for *val* in the set.
- SUCC(*val*) – Returns the smallest element greater than *val* in the set.
- PRED(*val*) – Returns the largest element lesser than *val* in the set.
- DELETE(*val*) – Deletes *val* from the set.

Trees

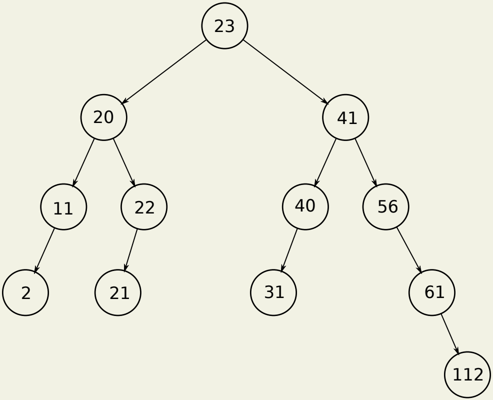
- Root
- Parent, Child
- Ancestor, Descendant
- Sibling
- Leaves, Internal Nodes
- Depth, Height

Trees

- Organization Structure
- File System
- Family Tree

Binary Trees

A binary tree is an ordered tree in which every node has at most 2 children.

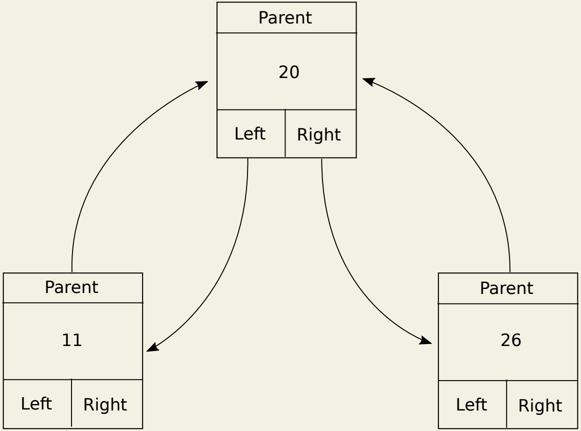


Implementation

Similar to a node in a linked list, each node in a Binary Tree has the following:

- int *val* – holds the data/value of the node.
- Left child pointer.
- Right child pointer.
- Parent node pointer.

Data Structure



Questions

1. What is the maximum height of a Binary Tree with n nodes?
2. What is the minimum height of a Binary Tree with n nodes?

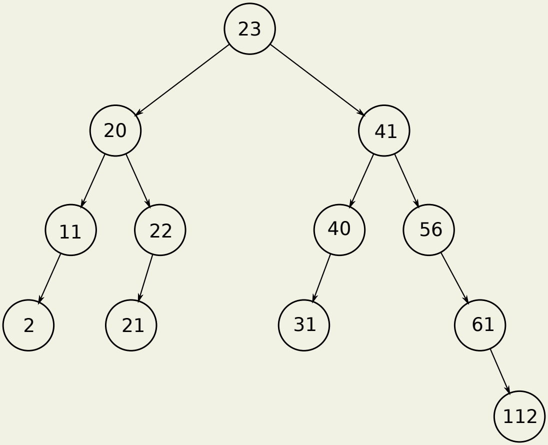
Binary Search Tree

A Binary Search Tree (BST) is a tree that satisfies the following:

- For every node X in the BST, we have:

  $$\text{Values in left subtree} \leq \text{value}(X) \leq \text{Value in right subtree}$$

Example BST



Data Structure - BST

A BST supports the following functions:

- INSERT(*node*, *val*) – Inserts *val* into the BST rooted at *node*.
- SEARCH(*node*, *val*) – Returns True of *val* exists in the BST rooted at *node*. False otherwise.
- SUCC(*val*) – Returns the smallest element greater than *val* in the BST.
- PRED(*val*) – Returns the largest element lesser than *val* in the BST.
- DELETE(*val*) – Deletes *val* from the BST.

Example BST

The order in which elements are inserted makes a difference!
Consider two different sequences of values:

Sequence A:
23, 11, 20, 21, 2, 56, 40, 41

Sequence B:
2, 11, 20, 21, 23, 40, 41

INSERT procedure

The INSERT($node, x$) procedure:

- If $node = \text{NULL}$, create new node with x and attach to parent.
- Else If $x < \text{value}(node)$,
  - INSERT($node \rightarrow left, x$)
- Else If $x > \text{value}(node)$ Then,
  - INSERT($node \rightarrow right, x$)

Binary Search Trees

Recall that a Binary Search Tree (BST) has the following crucial property:
For every node X in the BST, we have:

- Every node in the left subtree of X contains a value smaller than that of X.
- Every node in the right subtree of X contains a value larger than that of X.

Questions

- Write the SEARCH(*node*, *x*) procedure.

- SEARCH(*node*, *x*):

Questions

- Write the SEARCH(*node*, *x*) procedure.

- SEARCH(*node*, *x*):
- If $node = \text{NULL}$, then return NULL

Questions

- Write the SEARCH(*node*, x) procedure.

- SEARCH(*node*, x):
- If $node = \text{NULL}$, then return NULL
- Else If $x = \text{value}(node)$, then return *node*

Questions

- Write the $\textsc{Search}(node, x)$ procedure.

- $\textsc{Search}(node, x)$:
- If $node = \text{NULL}$, then return NULL
- Else If $x = \text{value}(node)$, then return $node$
- Else If $x < \text{value}(node)$, then
  - Return $\textsc{Search}(node \to left, x)$

Questions

- Write the $\textsc{Search}(node, x)$ procedure.

- $\textsc{Search}(node, x)$:
- If $node = \text{NULL}$, then return NULL
- Else If $x = \text{value}(node)$, then return *node*
- Else If $x < \text{value}(node)$, then
  - Return $\textsc{Search}(node \rightarrow left, x)$
- Else
  - Return $\textsc{Search}(node \rightarrow right.x)$