Basic Data Structures

September 28, 2023

What are programs made of?

Programs = Data Structures + Algorithms

Data Abstraction

- Separation of a data type's logical properties from its implementation.
- Logical Properties
  - What are the possible values?
  - What operations will be needed?
- Implementation
  - How can this be done in Java, C++, or any other programming language?

Abstract Data Type (ADT)

An Abstract Data Type (ADT) is a set of objects together with a set of operations.

- A data type that does not describe or belong to any specific data, yet allows the specification of organization and manipulation of data
- In general, without implementation, you first design your ADT in an abstract level according to your needs
- A data type whose properties (domain and operations) are specified independently of any particular implementation
- A data type that specifies and can share its logical properties without giving specifics of the implementation code
- A way of thinking about data types, often outside the constraints of a programming language
- **Actual data type is added later in an implementation**

ADT Implementation

- A Java class allows for implementation of ADTs, with appropriate hiding of implementation details.
- A program needs to call appropriate methods to perform an operation on ADT.
- If the implementation needs to be changed, the class can be updated without an effect on the program that uses the ADT (in an ideal world).

Basic ADTs

- List
- Stack
- Queue
- Tree

What is List ADT?

A list is a group of homogeneous items (objects) of the form
$A_0, A_1, A_1, ...A_{N-1}$

Abstract Definition/Constraints

- The size of the list is N
- The special list of size 0 is called an **empty list**
- For any list except empty list, we say A_i follows (or succeeds) $A_{i-1}(i < N)$ and A_{i-1} precedes $A_i(i > 0)$.
- The first element of the list is A_0 and last element is A_{N-1}
- Predecessor of A_0 and successor of A_{N-1} is not defined.
- The position of element A_i in a list is i.

Possible List Operations

- Find – Returns the position of the first occurrence of an item
- Insert – Insert some object to a position in the list
- Remove – Remove some object from a position in the list
- findKth – Returns the element in some position
- MakeEmpty – Sets list to an empty state.
- PrintList – Prints elements of the list.

Implementation of List ADT

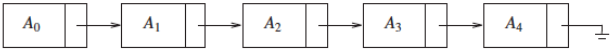
- Simple Array Implementation
- Simple Linked List Implementation

Simple Array Implementation of List ADT

Uses a simple array to store the items.

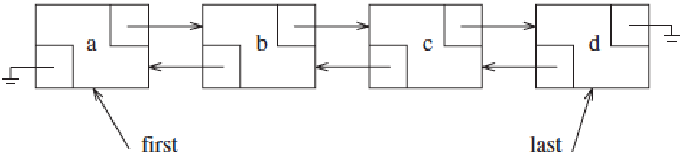
- PrintList
- FindKth
- Insert (what if the array is full)
- Remove

Simple Linked List Implementation of List ADT



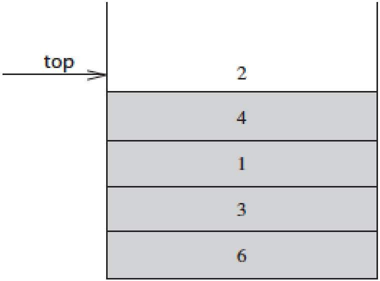
- PrintList
- FindKth
- Insert
- Remove

Doubly Linked List Implementation



What is a Stack?

- A stack is a list with the restriction that insertion and deletions can be performed only at the end of the list, called **top**
- A stack is a LIFO "Last In, First Out" list.



Stack Operations

- Push – Inserts and item to the top of the stack.
- Pop – Removes the most recently inserted item.
- Top – Returns the top item
- MakeEmpty – Sets stack to an empty state.

Some of Stack Use Cases

- Balancing Symbols
  - {[]}
- Postfix Expressions
  - Infix: a + b * c + (d * e + f) * g
  - Postfix: a b c * + d e * f + g * +
- Function calls

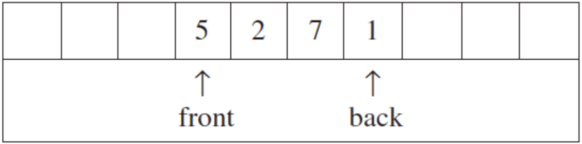
What is a Queue?

- A queue is a list with the restriction that insertion is done at one end, whereas deletion is performed at the other end.
- A queue is a FIFO "First In, First Out" list.

Queue Operations

- Enqueue – Adds an element to the end of the queue.
- Dequeue – Removes (and returns) en element from the front of the queue.
- MakeEmpty – Sets queue to an empty state.

Array Implementation of Queue



- If we hit the end of array (while there are 3 empty spaces at the start of the array), what can we do to add a new item?
- Circular array ??

Applications of Queues

- There are many algorithms that use queues to give efficient running times (topological sort, shortest path algorithms in graph theory etc, more on this later in the course, stay tuned)
- Some simple examples
  - Jobs in the printer
  - Real life lines (in a bank etc)
  - Calls to a call center

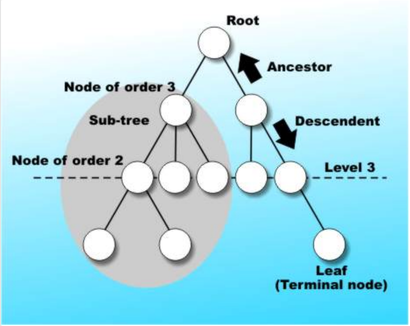
Tree

A tree is a collection of nodes such that:

- The collection can be empty, otherwise there is a specially designated node called the root.
- The remaining nodes are partitioned into $n \geq 0$ disjoint sets $T_1, \ldots, T_n$, where each of these sets is also a tree.
- We call $T_1, \ldots, T_n$ the subtrees of the root.
- Each subtree is connected by a directed edge from root.

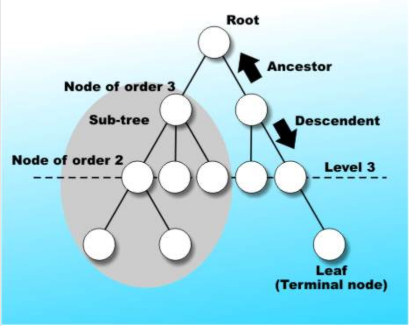
Terminology

- The degree of a node is the number of subtrees of the node
- The node with degree 0 is a leaf or terminal node.
- A node that has subtrees is the parent of the roots of the subtrees.
- The roots of these subtrees are the children of the node.
- Children of the same parent are siblings.
- The ancestors of a node are all the nodes along the path from the root to the node.

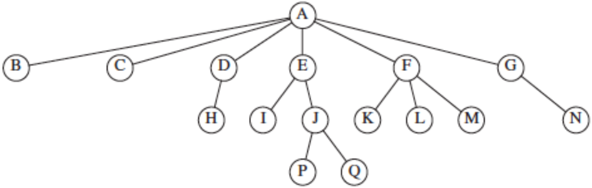


Terminology

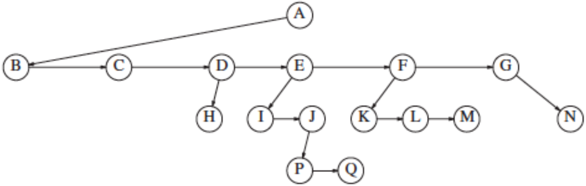
- A path from $node_1$ to $node_k$ is defined as a sequence of nodes $node_1$, $node_2$ ... $node_k$ such that $node_i$ is the parent of $node_{i+1}$ for $1 \leq i < k$
- For any node $node_i$ the depth of $node_i$ is the length of the unique path from the root to $node_i$
- The height of $node_i$ is the length of the longest path from $node_i$ to a leaf.
- The height of a tree is the height of root.



A Tree



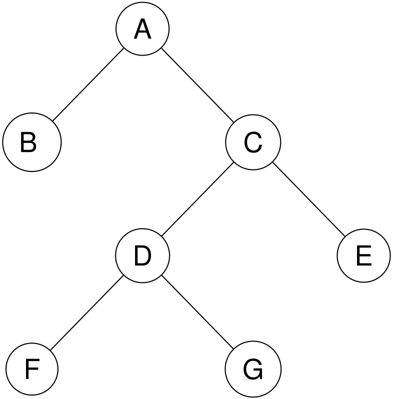
An Implementation of Tree



Binary Tree

- A binary tree is a tree in which each node can have at most two children.
- The two children of a node are called the left child (T_L) and the right child (T_R), if they exist.

Binary Tree Example



An Implementation of Binary Tree Node

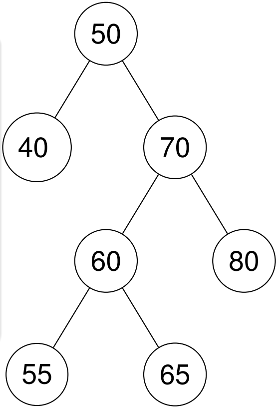
```
class BinaryNode<T1 extends Comparable> implements Comparable {
    private T1 data;
    private BinaryNode left;
    private BinaryNode right;

    @Override
    public int compareTo(BinaryNode<T1> node){
        return this.data.compareTo(node.getData());
    }
}
```

Binary Search Tree

A special kind of binary tree in which:

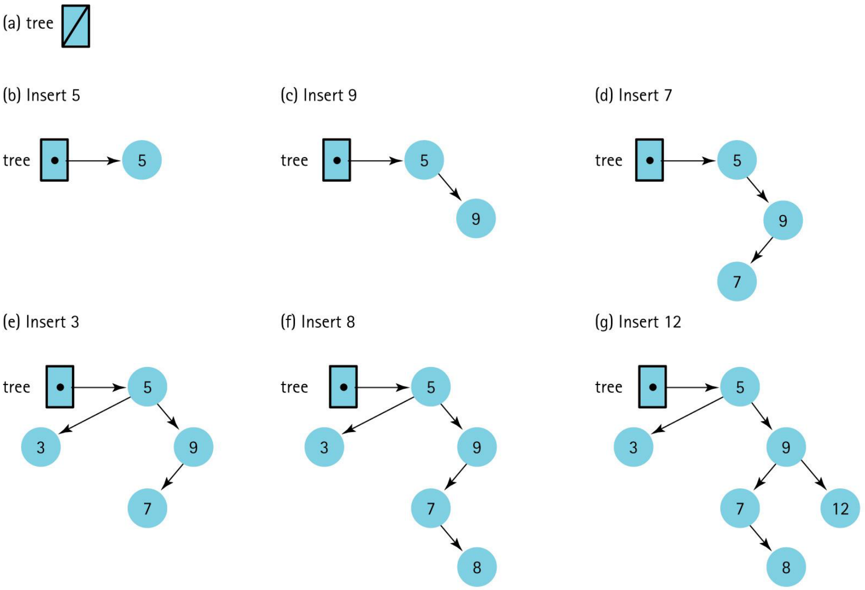
- Each node contains a distinct data value,
- The key values in the tree can be compared using "greater than" and "less than" operators
- **The key value of each node in the tree is less than every key value in its right subtree, and greater than every key value in its left subtree**



Insertion into a Binary Search Tree

Recursive Code In PS

Binary Search Tree Insertion Example

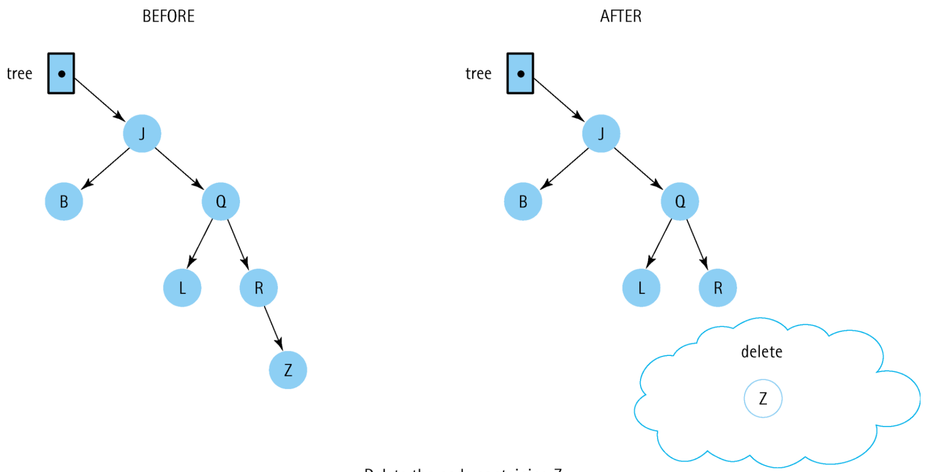


Remove Operation

PseudoCode

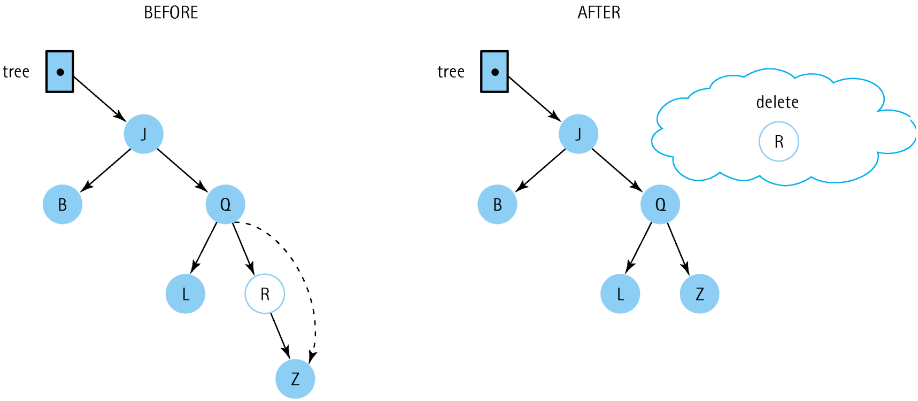
```
void remove( Comparable x, BinaryNode t ) {
    if( t == empty )
        return;   // Item not found; do nothing
    if( x < t->element )
        remove( x, t->left );
    else if( t->element < x )
        remove( x, t->right );
    else if( Equal & Two children ) {
        t->element = findMaxInLeftSubTree();
        deleteLeafElement();
    }
    else if( Equal & One child ) {
        t->element = singleChild;
        deleteRemovedElement();
    }
}
```

Removing a Leaf Node



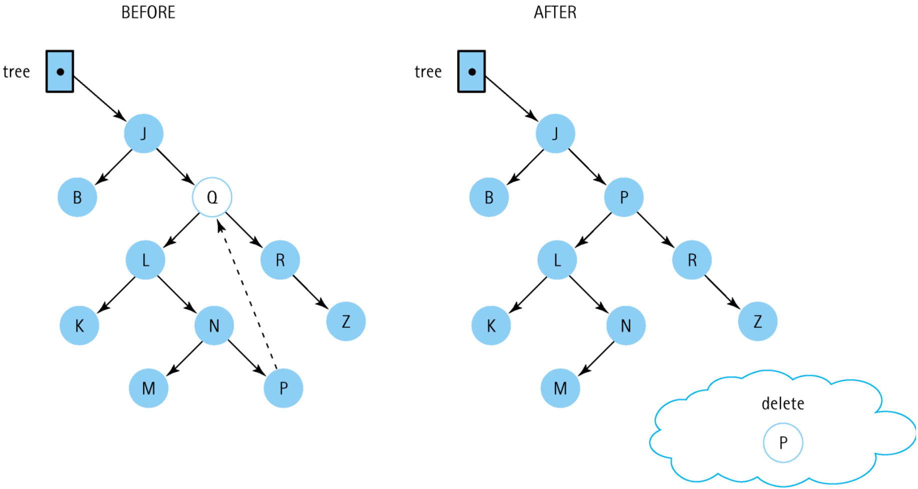
Delete the node containing Z

Removing a Node with One Child



Delete the node containing R

Removing a Node with Two Children



Delete the node containing Q