Data Structures extravaganza

Data Structures

- ✘ Store a collection of related data
- ✘ Operations that we can do:
  - `add(x)`
  - `remove(x)`
  - `empty(x)`
  - `find(x)`

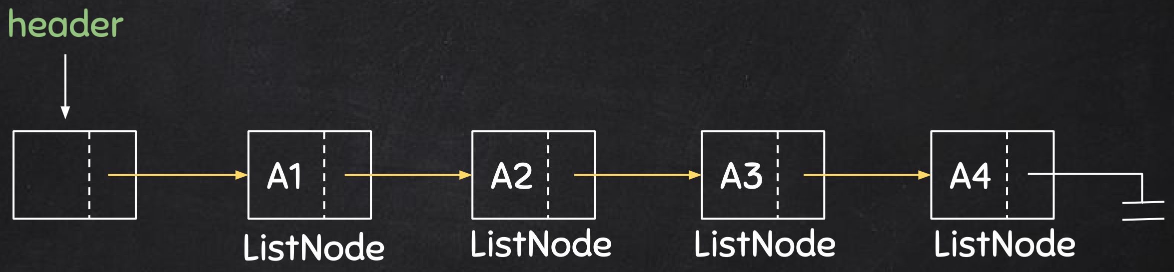
Linear Data Structures

List

Index 1 2 3 4

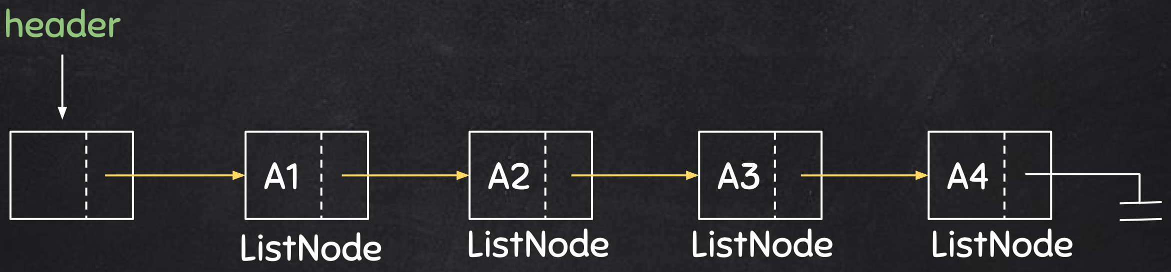
- ✘ A collection of items. Each item has a position.
- ✘ Can access any item using its index.
- ✘ Example: arrays
  - Allow random access
  - Insertion/deletion is expensive!

Linked List



- ✗ Using pointer or links among data in the collections.
- ✗ Random access is not allowed.
- ✗ Extra memory space
- ✗ Variants: sorted, doubly-linked list, circular linked list

Linked List




```python
class ListNode:

    def __init__(self, d=None, n=None):
        self.data = d
        self.next_node = n

    def get_data(self):
        return self.data

    def get_next(self):
        return self.next_node

    def set_next(self, new_next):
        self.next_node = new_next
```

```python
class LinkedList:

    def __init__(self):
        self.head = ListNode()

    def insert(self, data):
        new_node = ListNode(data, self.head.get_next())
        self.head.set_next(new_node)

    def printList(self):
        current = self.head.get_next()
        while current is not None:
            print current.get_data()
            current = current.get_next()
```

Superclass

```
class List:
	def __init__(self):
	def isEmpty(self):
	def makeEmpty(self):
	def insert(self,x,current):
	def remove(self,x)
```

```
class LinkedList(List):
	def __init__(self):
	def insert(self,x,current):
	def remove(self,x)
	...
```

Subclass

```
class DoublyLinkedList(List):
	def __init__(self):
	def insert(self,x,current):
	def remove(self,x)
	...
```

Subclass

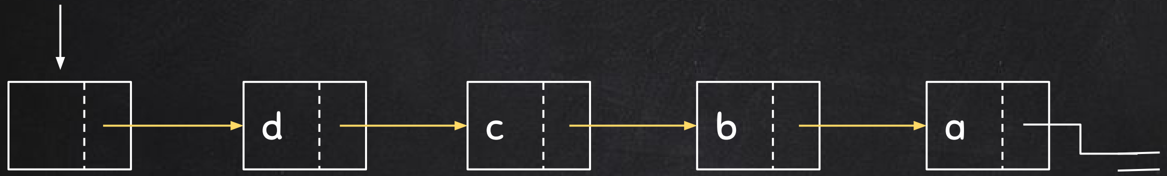
Stack

- Access is restricted to the most recently inserted item.
- Operations take a constant amount of time.

Stack

Implementation: Linked List

topOfStack



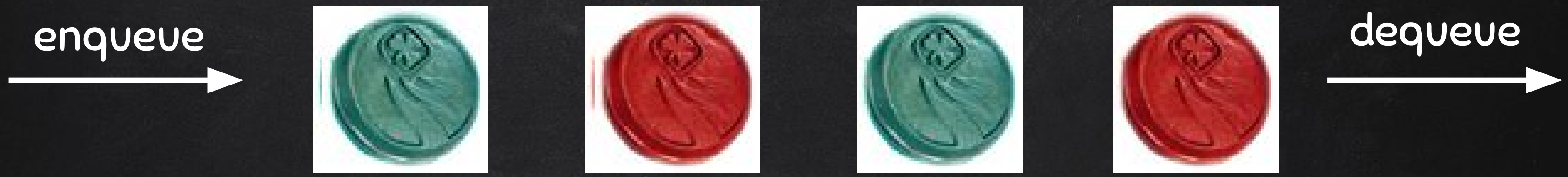

```python
class MyLinkedListStack:

    def __init__(self):
        self.topOfStack = ListNode()

    def push(self, data):
        new_node = ListNode(data, self.topOfStack)
        self.topOfStack = new_node

    def top(self):
        data = self.topOfStack.get_data()
        self.topOfStack = self.topOfStack.get_next
        return data
```

Queue



- ✘ Access is restricted to the least recently inserted item.
- ✘ Operations take a constant amount of time.

Priority Queue

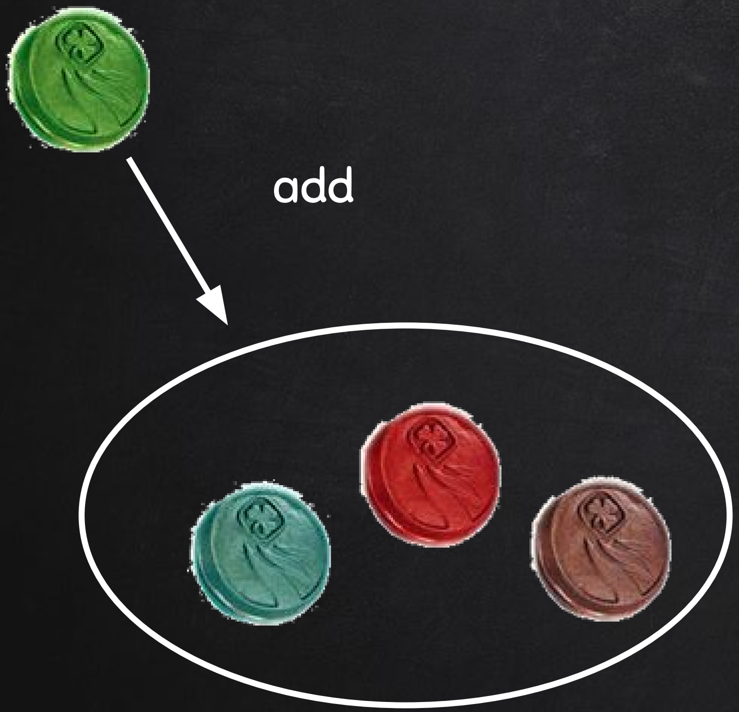
- ✘ A queue in which each item can have a priority value.
- ✘ Access is restricted to item with the highest priority.

Maps

Keys Z01 AB3 UY6 HAA

- A collection of (key, value) items.
- Keys must be unique, but values don't.
- Access to an item by using its key.

Set

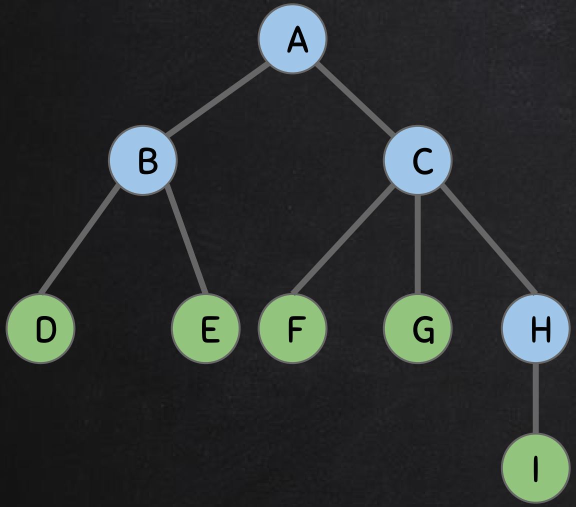


- ✘ A data structure that contains no duplicates.

Non Linear Data Structures

Trees & Graphs

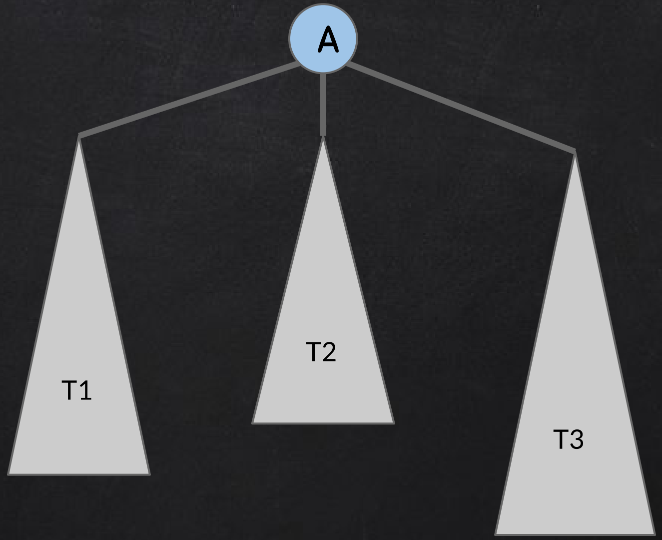
Trees



- **Root** node: A
- **Internal** nodes: A, B, C, H
- **External** nodes/leaves: D, E, F, G, I
- C is the **sibling** of B
- D and E are the **children** of B
- The **height** of tree is 3
- The **depth/level** of E is 2
- The **degree** of node B is 2

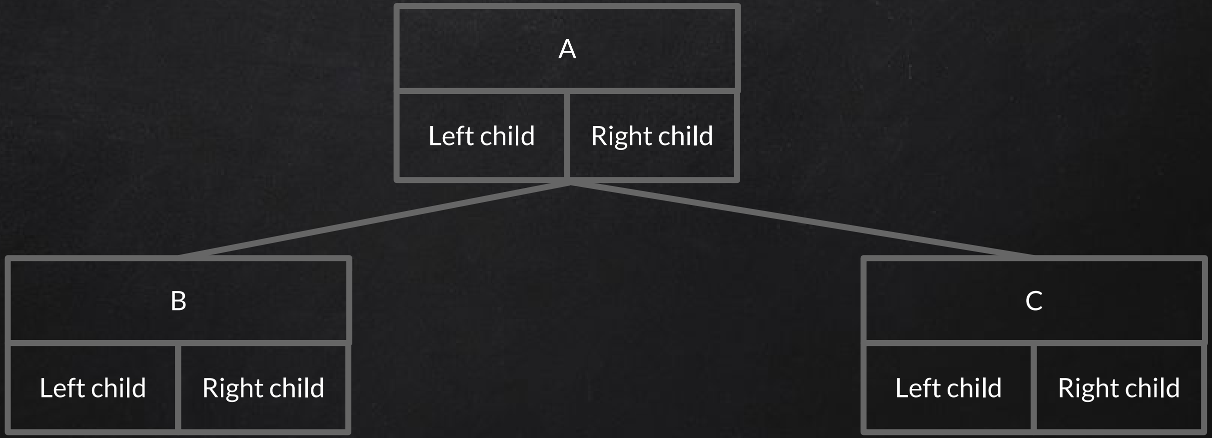
Solving Tree Problems

A subtree is also a tree!



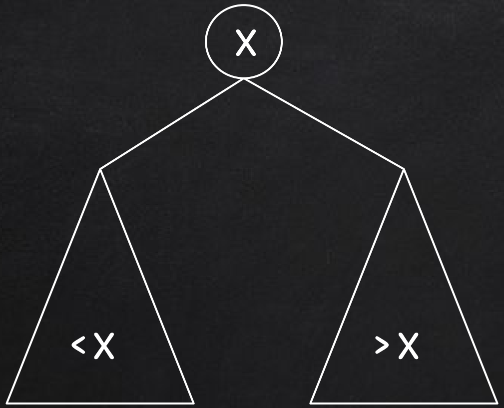
Tree Representation

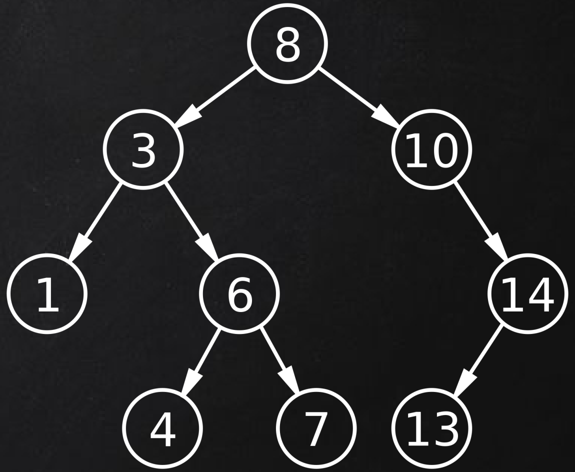
- We can use LinkedList to represent a tree.
- Example: binary tree



Binary Search Trees (BST)

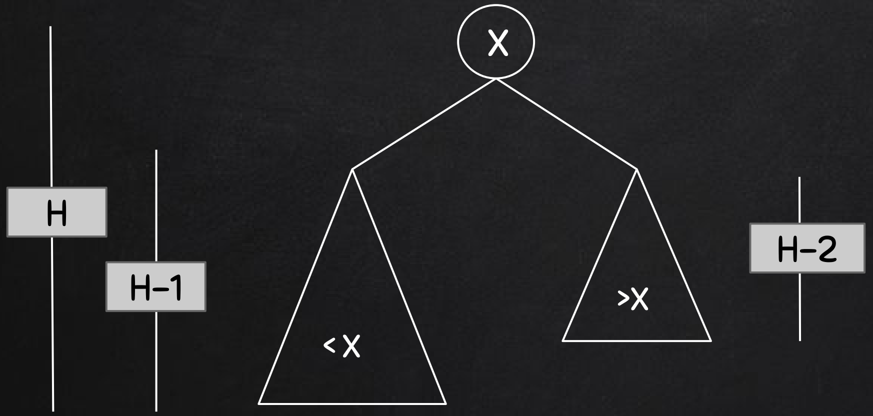
- Elements have keys (no duplicates allowed)
- Property:

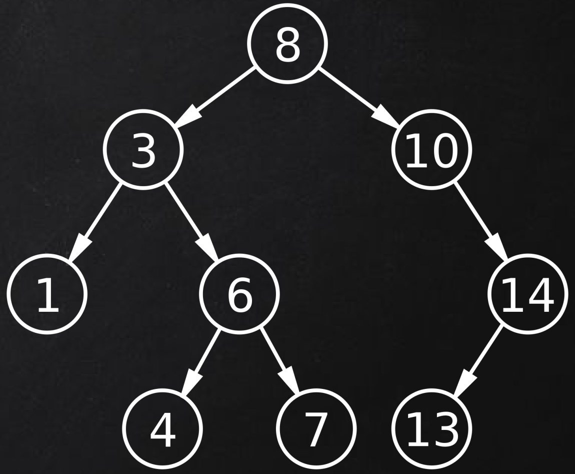




AVL Trees

- Unbalanced BST are inefficient.
  How to maintain balance?
- Property:

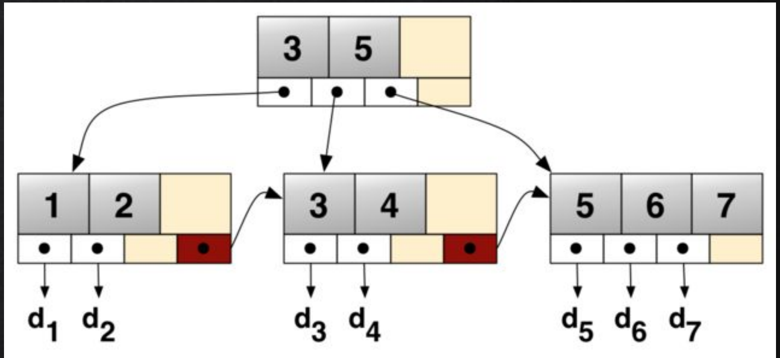




Demo: https://visualgo.net/bn/bst

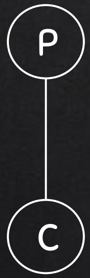
B+ Trees

- ✘ B stands for Balanced.
- ✘ All the leaves are the same distance from the root.
- ✘ B Tree of degree m:
  - All non-leaf nodes (except root) have between m/2 (ceil) and m non-empty children.

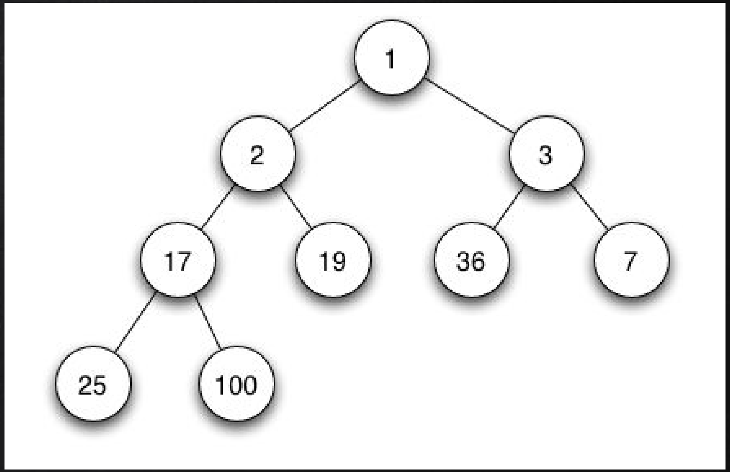


Binary Heap

- One technique to implement priority queue.
- The tree is balanced, so all operations are guaranteed O(log n) worst case.
- Property:

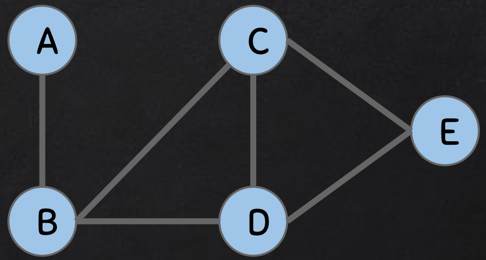


Parent <= Child
(minheap)



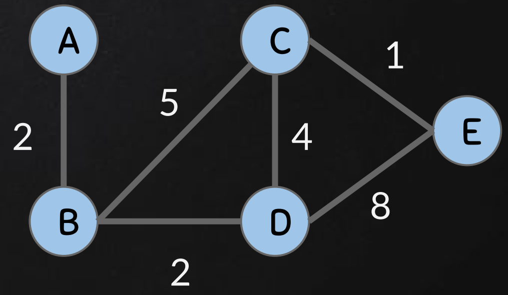
Graphs

- ✘ A collection of vertices of arcs which connects vertices in the graph.
- ✘ G = (V, E)
  - V is the set of vertices (similar with nodes in trees)
  - E is the set of edges



Graphs

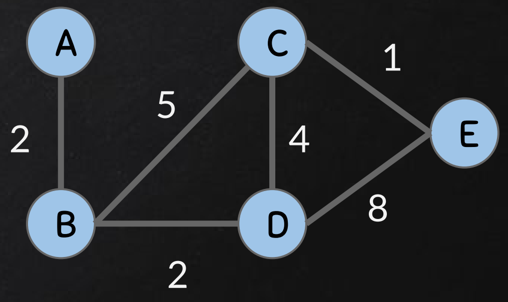
- There are three types of graphs:
  - Undirected graph
  - Directed graph
  - Weighted graph



Graph Representation

Adjacency Matrix

	A	B	C	D	E
A	0	2	0	0	0
B	2	0	5	2	0
C	0	5	0	4	0
D	0	2	4	0	8
E	0	0	0	8	0



Graph Representation

Adjacency List

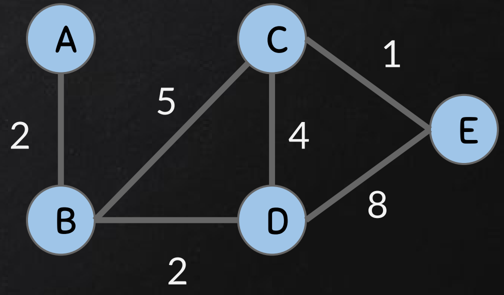
A → B

B → C, D

C → B, D, E

D → B, C, E

E → C, D



Graph Traversal

- ✘ Used to find all nodes that are reachable from the given root node.
- ✘ Algorithms:
  - Breadth First Search (BFS)
    - Explores nodes in the order of their distance from the root
  - Depth First Search (DFS)
    - Start from the root, we explore as long as possible along the path
    - We backtrack when we could not go any further

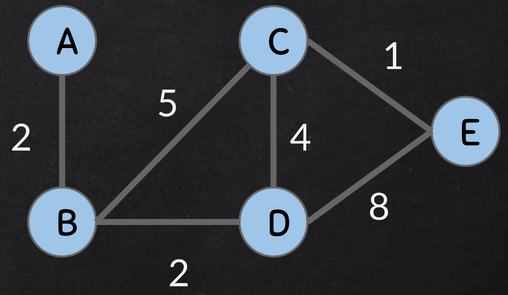
Graph Traversal

An example

Root node: A

BFS traversal: B, C, D, E

DFS traversal: A, B, D, C, E



Minimum Spanning Tree

A spanning tree is a subgraph that contains all the vertices and is a tree.

A minimum spanning tree (MST) is a spanning tree whose weight is no larger than the weight of any other spanning trees.

Algorithms to find MST: Prims, Kruskal, etc.

Preparing for Coding Interviews

Books:

- ✘ Cracking the Coding Interview
- ✘ Elements of Programming Interviews

Websites:

- ✘ https://leetcode.com/
- ✘ https://www.hackerrank.com/
- ✘ https://www.interviewbit.com/

References

- Data Structures and Algorithm course slides, *Faculty of Computer Science, University of Indonesia*, 2014.