CS 367 - Introduction to Data Structures
Week 6, 2017

Program 2

- Due 10 pm Friday, July 28th
- Take a look at clarifications posted and come talk to me if you have serious concerns about them.

Homework 5 released, complete as soon as possible. Due by 10 pm Sunday, July 30th

Rest of the assignments check out Piazza post 250. Come discuss with me if you have any concerns.

Exam anonymous feedback check out Piazza post 240. Midterm 2 will be incorporating the feedback received, so please volunteer to provide your feedback.

Last Week

General Trees, Classifying Binary Trees, Balanced Search Trees

This Week

Read: *Red-Black Trees*, *Graphs*

Red-Black Trees

- Tree properties
- Insert
- Complexity

ADTs/Data Structures Revisited

Graphs

- terminology
- implementation
- edge representations
- traversals
- applications of BFS/DFS
- more terminology
- topological ordering

Next Week (more Graphs and Hashing)

Read: continue *Graphs, Hashing*

- topological orderings
- Dijkstra's Shortest Path algorithm

Hashing

- terminology
- designing a good hash function
- choosing table size
- expanding a hash table
- handling collisions

Java Support for Hashing: Tree Map vs Hash Map

Sorting Intro

Red-Black Trees (RBT)

RBT:

Example:

Red-Black Tree Properties

root property

red property

black property

Red-Black Tree Operations

print
lookup

insert

delete

Inserting into a Red-Black Tree

Goal: insert key value K into red-black tree T
and ____.

If T is Empty

If T is Non-Empty

- step down tree as done for BST
- add a leaf node containing K as done for BST, and ___________________
-

→ **Which of the properties might be violated as a result of inserting a red leaf node?**

root property

black property

red property

Non-Empty Case 1: K's parent P is black

Non-Empty Case 2

Non-Empty Case 2: K's parent P is red

Fixing an RBT

Tri-Node Restructuring is done if P's sibling S is null

Recoloring is done if P's sibling S is red

Practice

→ 1. Starting with an empty RBT, show the RBT that results from inserting 7 and 14.

→ 2. Redraw the tree from above and then show the result from inserting 18.

→ 3. Redraw the tree from above and then show the result from inserting 23.

→ 4. Redraw the tree from above and then show the result from inserting 1 and 11.

→ 5. Redraw the tree from above and then show the result from inserting 20.

More Practice!

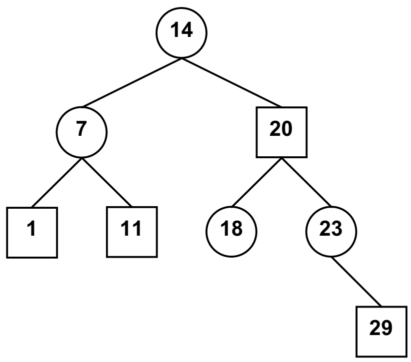
→ 6. Redraw the tree from the previous page and then show the result from inserting 29.

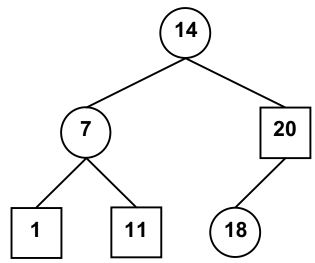
→ 7. Insert the same list of values into an empty BST: 7, 14, 18, 23, 1, 11, 20, 29

→ What does this demonstrate about the differences between a BST and RBT?

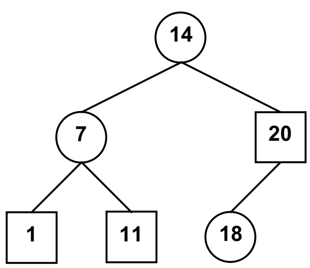
More Practice?

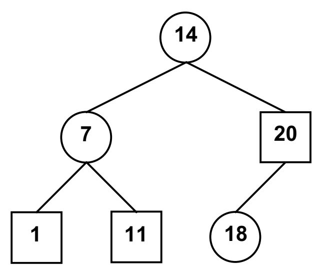
→ 8. Show the result from inserting 25 in the RBT below.





→ 9. Redraw the tree from above and then show the result from inserting 27.

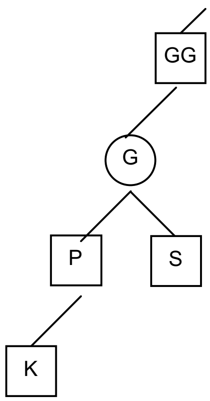




Cascading Fixes

Fixing an RBT UPDATED!

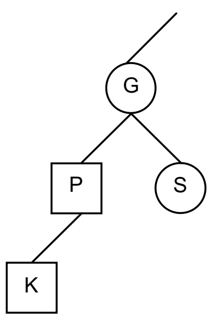
Recoloring is done if P's sibling S is red

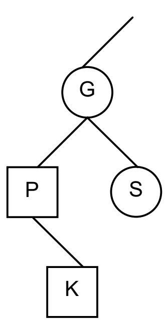


1. change P & S to black
2. if G is the root – done

   otherwise change G to red

Tri-Node Restructuring is done if P's sibling S null _______________





RBT Complexity

print

lookup

insert

ADTs/Data Structures

Linear (Lists, Stacks, Queues)

- predecessors: at most 1
- successors: at most 1

Hierarchical (Heaps, BSTs, Balanced Search Trees)

- predecessors: at most 1
- successors: 0 or more - general tree, at most two - binary tree

Graphical

- predecessors:
- successors:

Graph Terminology

Implementing Graphs

Graph ADT Ops

Graph Class

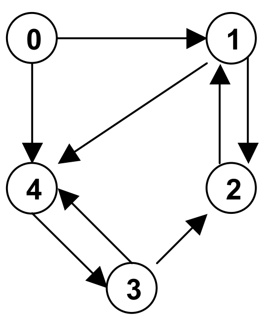
Graphnode Class

Representing Edges

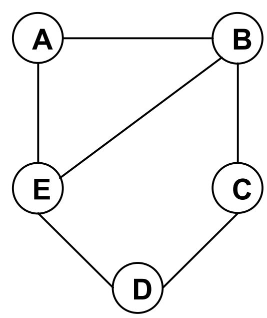
Adjacency Matrix

Given the following graphs:

Graph 1



Graph 2



→ **Show the adjacency matrix representation of the edges for each of the graphs:**

Graph 1

	0	1	2	3	4
0					
1					
2					
3					
4					

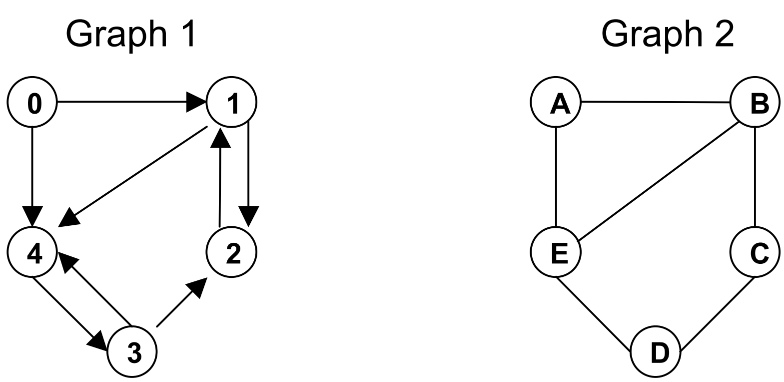
Graph 2

	A	B	C	D	E
A					
B					
C					
D					
E					

Representing Edges

Adjacency Lists

Given the following graphs:



→ Show an adjacency list representation of the edges for each of the graphs:

	Graph 1		**Graph 2**
0:		**A:**	
1:		**B:**	
2:		**C:**	
3:		**D:**	
4:		**E:**	

Using Edge Representations

→ **Write the code to be added to a `Graph` class that computes the degree of a given node in an undirected graph.**

1. Adjacency list:

```
public int degree( Graphnode<T> n) {
```

2. Adjacency matrix:

```
public int degree( Graphnode<T> n) {
```

Comparison of Edge Representations

Ease of Implementation

Space (memory)

AM

AL

Time (complexity of ops)

node's degree?

AM

AL

edge exists between two given nodes?

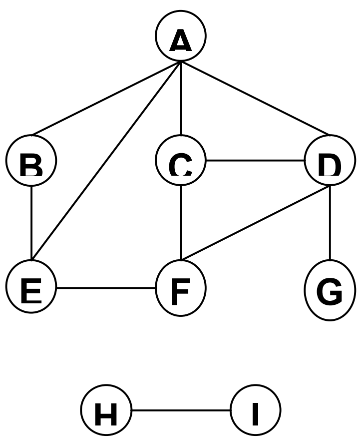
AM

AL

Searches and Traversals

Search

Traversal



→ **Which connected component in the graph above can produce the longest path?**

✵

Depth-First Search (DFS)

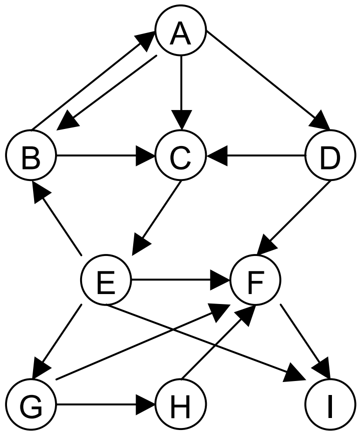
-
-

Algorithm

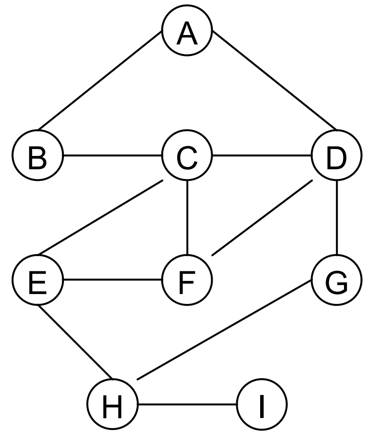
✵

DFS Practice

Graph 1



Graph 2



→ **Give the order that vertexes are visited for depth-first search (DFS) starting at A.**

Graph 1:

Graph 2:

→ **Give the DFS spanning tree starting at A.**

Graph 1:

Graph 2:

Breadth-First Search (BFS)

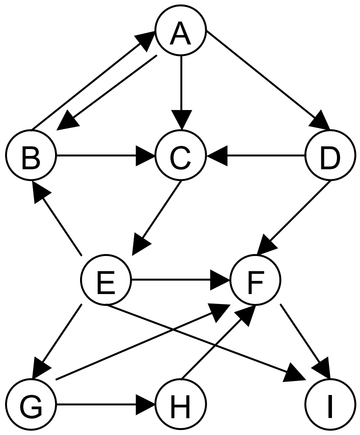
-
-

Algorithm

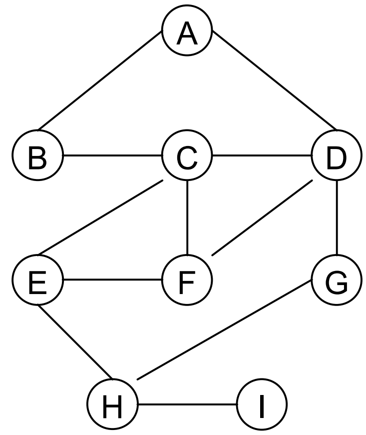
✵

BFS Practice

Graph 1



Graph 2



→ Give the order that vertexes are visited for breadth-first search (BFS) starting at A.

Graph 1:

Graph 2:

Give the BFS spanning tree starting at A.

Graph 1:

Graph 2:

Applications of DFS/BFS

Path Detection

Cycle Detection

More Graph Terminology

Weighted graph:

Network:

Complete graph:

Connected graph (undirected):

Connected graph (directed):

Length of a path: