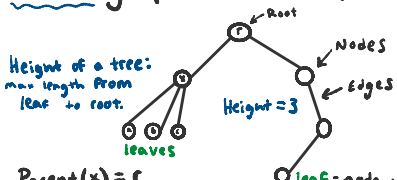
Notes #3

Thursday, April 2, 2020 1:00 PM

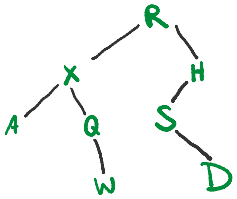
Trees: graph with no cycles



Parent(x) = r
Children(x) = {a, b, c}

Binary Trees: Every node has at most 2 children.

- Left and Right Children



level: RXHAQSWD
pre: RXAQWHSD
in: AXQWRSDH
post: AWQXDSHR

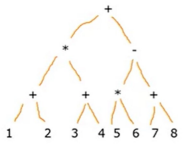
Tree Traversals:

a) Preorder traversal: "visit" each node before "visit" its children (for short: (node, left, right))

b) Postorder traversal: "visit" each node only after "visit" its children
or say: 1) visit its children first
2) visit the node (after visiting its children)
(For short: (left, right, node))

c) Inorder traversal:
1) visit the left child
2) visit the node
3) visit the right child
(for short: (left, node, right))
only for binary trees

d) Level order: left to right by level



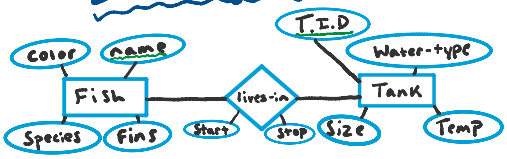
in: ((1+2)*(3+4)) +((5*6)-(7+8)) (infix)
pre: + * + 1 2 + 3 4 - * 5 6 + 7 8 (prefix)
post: 1 2 + 3 4 + * 5 6 * 7 8 + - + (postfix)

Brookshear Ch. 9

Database Systems:

- Entity - Relationship model
- Relational model
- SQL
- Other stuff

E-R Diagram:



Fish:

name	color	species	fins
Nemo	Orange	Clown	7

Lives-in:

name	TID	start	stop
Nemo	1	1/2	4/1

Tank:

TID	WT	Size	Temp
1	Salt	50	70

Relational Algebra - relational operators

projection - selects a column π
selection - choose a row based on a condition σ
join - combines tables on common attributes/columns $\bowtie$

List all fish colors
π_{color} (fish)

List fish with at least 5 fins
$\sigma_{fins>=5}$ (fish)

List the names of fish with at least 5 fins
π_{name} ($\sigma_{fins>=5}$ (fish))

List fish living in salt water.
π_{name} (lives-in $\bowtie$ $\sigma_{sw="s"}$ (tank))

Other topics:

concurrency
fault-tolerance/backups/checkpoints
optimization - indexes
query processing and optimization
distributed processing

SQL - Sequel

projection - select
selection - where clause
join - join

List all fish colors
select color from fish;

List fish with at least 5 fins
select * from fish where fins>=5;

List the names of fish with at least 5 fins
select name from fish
where fins>=5;

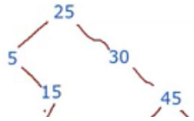
List fish living in salt water.
select name from (lives-in join
(select * from tank where sw="s"));

Binary Search Trees:

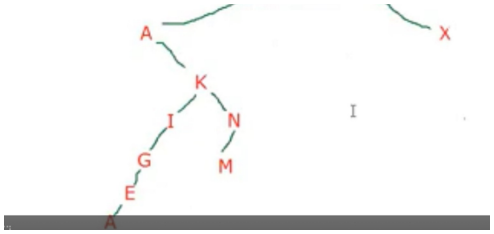
binary tree
For every node:
every item in the left subtree is < the item in that node
every item in the right subtree is >= the item in that node

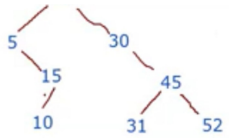
Insertion: TAKING EXAM

25 5 30 45 15 10 52 31



O(log n) average
insert, search times





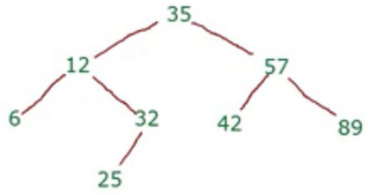
O(log n) average
insert, search times

Level order: 25 5 30 15 45 10 31 52
Preorder: 25 5 15 10 30 45 31 52
Inorder: 5 10 15 25 30 31 45 52
Postorder: 10 15 5 31 52 45 30 25

```
import java.io.*;

class BST implements Collection
{
 private class treenode
 {
  public Comparable data;
  public treenode left;
  public treenode right;
 }
 treenode root;
 int count;

 public BST()
 {
  root=null; count=0;
  Q=new QueueLL();
 }

 public void insert(Comparable x)
 {
  count++;
  root=inserthelp(root,x);
 }
 private treenode inserthelp(treenode r, Comparable x)
 // returns the new value of the root of this subtree (r)
 {
  // base case
  if(r==null)
  {
   treenode newnode=new treenode();
   newnode.data=x;
   newnode.left=newnode.right=null;
   return newnode;
  }
  // recursive cases
  // left subtree
  if(x.compareTo(r.data)<0)
   r.left=inserthelp(r.left,x);
  else
   r.right=inserthelp(r.right,x);
  return r;
 }

 public Comparable lookup(Comparable x)
 {
  return luh(root,x);
 }
 private Comparable luh(treenode r, Comparable x)
 {
  if(r==null) { return null; } // not in the tree!
  if(r.data.compareTo(x)==0) return r.data;
  if(x.compareTo(r.data)<0) return luh(r.left,x);
  return luh(r.right,x);
 }

 // traversal functions

 public static final int PREORDER=-1;
 public static final int INORDER=0;
 public static final int POSTORDER=1;

 private Queue Q;

 public void reset(int order)
 {
  Q.makeEmpty();
  traversal(root,order);
 }

 private void traversal(treenode r, int order)
 {
  // base case - empty
  if(r==null) return;

  // recursive case
  if(order==PREORDER) { Q.enqueue(r.data); }
  traversal(r.left,order);
  if(order==INORDER) { Q.enqueue(r.data); }
  traversal(r.right,order);
  if(order==POSTORDER) { Q.enqueue(r.data); }
 }

 public Comparable getNext() { return (Comparable) Q.dequeue(); }

 public boolean hasNext() { return !Q.isEmpty(); }

 // quick print function
 public void print() { printhelp(root); }
 private void printhelp(treenode r)
 {
  if(r==null) return;
  printhelp(r.left);
  System.out.println(r.data);
  printhelp(r.right);
 }


 public int size() { return count; }
 public boolean isEmpty() { return root==null; }
 public boolean isFull() { return false; }
 public void makeEmpty() { root=null; count=0; }

 private Comparable deleteditem;

 public Comparable delete(Comparable x) // like lookup
 {
  deleteditem=null;
  root=deletehelp(root,x);
  return deleteditem;
 }

 private treenode deletehelp(treenode r, Comparable x)
 {
  if(r==null) return null;
  if(r.data.compareTo(x)==0) // it matches!  Delete!
  {
   deleteditem=r.data;
   count--;
   // 0 children
   if(r.left==null && r.right==null)
    return null;
```

BST: traversals

35 12 57 6 89 32 25 42



level: 35 12 57 6 32 42 89 25
pre: 35 12 6 32 25 57 42 89
in: 6 12 25 32 35 42 57 89
post: 6 25 32 12 42 89 57 35

Use a Queue!!

reset
getNext
hasNext

Theory of Computation

watch Hidden Figures

Questions:
What is a computer?
What can and can it not do?
Complexity theory - how much difficulty is there?

Alan Turing, 1930s
invented the Turing Machine model of computing
turing machine=finite brain + infinite "memory" paper tape

Church-Turing thesis - all computing models are essentially equivalent

There are functions with no algorithm.

H(A,x) = true iff algorithm A halts on input x
= false if A(x) runs forever (infinite loop)

Theorem - H() has no algorithm

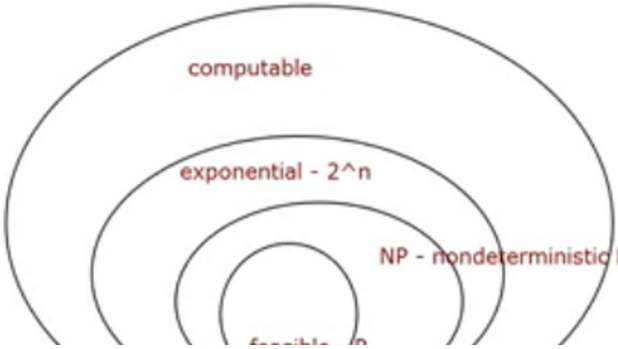
Proof that H(A,x) has no algorithm is by contradiction:

Assume H(A,x) has an algorithm, and get a contradiction.

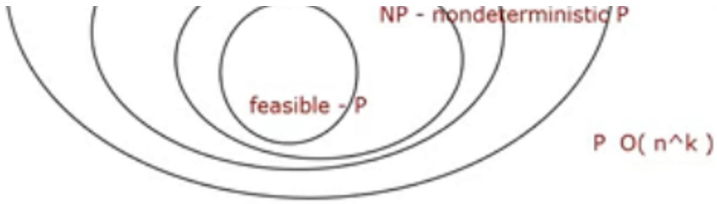
program R(A):
if H(A,A)==true, then while(true); // infinite loop
else { print("hi") and halt; }

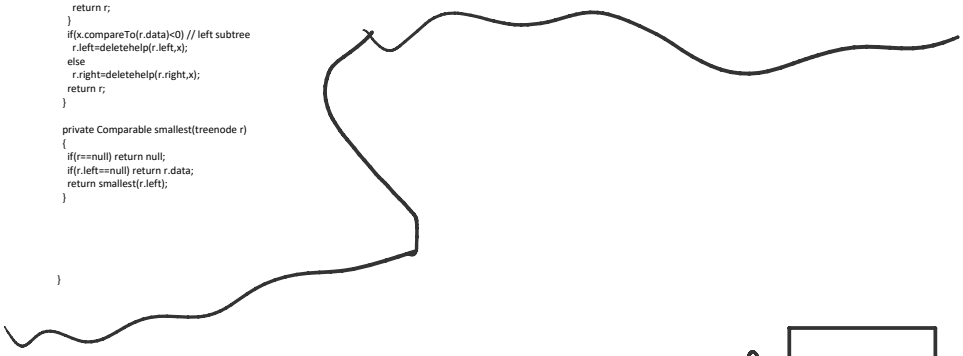
What is R(R)? What happens? Contradiction - R(R) neither halts nor runs forever.

How bad is it?



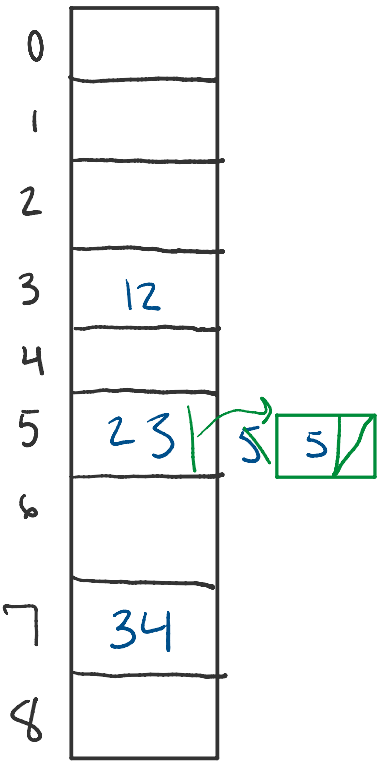
```
if(r.data.compareTo(x)==0) // it matches!  Delete!
{
 deleteditem=r.data;
 count--;
 // 0 children
 if(r.left==null && r.right==null)
  return null;
 // 1 child
 if(r.right==null) // we have a left child
  return r.left;
 if(r.left==null) // we have a right child
  return r.right;
 // 2 children!
 Comparable ios=smallest(r.right);
 Comparable deleteditem2=r.data;
 r.data=ios;
 r.right=deletehelp(r.right,ios);
 count++;
 deleteditem=deleteditem2;
 return r;
}
if(x.compareTo(r.data)<0) // left subtree
 r.left=deletehelp(r.left,x);
else
 r.right=deletehelp(r.right,x);
return r;
}

private Comparable smallest(treenode r)
{
 if(r==null) return null;
 if(r.left==null) return r.data;
 return smallest(r.left);
}


}
```





Hash Tables:

Operations:	BST:	Hash Table:
insert	log n	O(1)
delete	log n	O(1)
lookup	log n	O(1)
traversals	n	O(n log n)



Hash Function:

12, 24, 34, 5, 21

h(x)

input: primary attribute

output: table position (0...8)

$h(x) = x \% 9$

collision!

Collision Resolution:

1. linear probing
2. quadratic probing
3. Separate Chaining
4. buckets

Priority Queues: Heap Data Structure

Priority Queue:

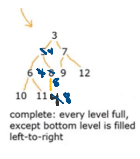
enqueue
dequeue
getFront
Collection

use Comparable objects

Heap Data Structure

binary tree

1. Complete →



2. heap condition

→ Heap Condition:
every node is <= its children

```
public interface PQ extends Collection
{
	public void enqueue(Comparable x);
	public Comparable dequeue();
	public Comparable getFront();
}
```

From <https://blue.butler.edu/~jsorenso/oop/code/11am/PQ.java>

```
import java.io.*;
import java.util.*;
class PQMain
{

	public static void main(String [] args) throws IOException
	{
		Scanner ff=new Scanner(new FileReader("fishfile.txt"));
		int fishcount;
		fishcount=ff.nextInt();

		PQ hof=new Heap(100);

		for(int i=0; i<fishcount; i++)
		{
			hof.enqueue(new Fish(ff));
		}

		while(!hof.isEmpty())
			System.out.println(hof.dequeue());
}
}
class Heap implements PQ
{
 Comparable [] h;
```

Use Comparable Objects

→ Heap Condition:
every node is <= its children

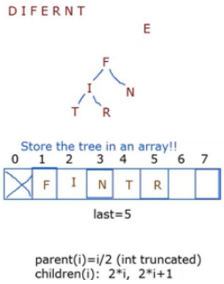
insert(x):
1. add x at the *last position*
(only place that preserves
the complete property)
2. bubble up

deletemin
1. copy out the root (return it later)
2. move the *last item* to the root
3. bubble down

Heap Condition:
every node is <= its children

complete: every level full,
except bottom level is filled
left-to-right

DIFERNT



parent(i)=i/2 (int truncated)
children(i): 2*i, 2*i+1

Artificial Intelligence

- Agents: making them intelligent
- Weak AI: weakly intelligent
- Strong AI: human-like, self-aware intelligence, the Singularity!! (alphaGo)

AI areas of research:
- Image processing
- Language recognition
- Robotics
- Neural networks/machine learning
- Genetic algorithms
- Games

How do we tell if an agent is intelligent? *State Space Search*
Turing Test (1950)

What is hard for AI? What's easy for people
Hard: walking, seeing, hearing, reading
Easy: chess, diagnosing (expert system)

GUI Programming

1. Get Java on Laptop
2. Write a GUI program (Graphical User Interface)
3. Write a GUI program that quits correctly

1. We need the Java Runtime Environment (JRE)

```
                while(!hof.isEmpty())
                        System.out.println(hof.dequeue());
}
}
class Heap implements PQ
{
 Comparable [] h;
 int last;
public Heap(int maxsize)
 {
  h=new Comparable[maxsize+1];
  last=0;
 }
public Heap() { this(100); }
private int parent(int pos) { return pos/2; }
 private int left(int pos) { return 2*pos; }
 private int right(int pos) { return 2*pos+1; }
 private void swap(int a, int b)
 {
  Comparable tmp=h[a];  h[a]=h[b]; h[b]=tmp;
 }
public void enqueue(Comparable x)
 {
  if(isFull()) return;
  // add the item in the last position
  h[++last]=x;
  bubbleup(last);
 }
private void bubbleup(int pos)
 {
  // base cases
  if(pos==1) return;
  if(h[parent(pos)].compareTo(h[pos])<=0) return;
  // recursive case
  swap(pos,parent(pos));
  bubbleup(parent(pos));
 }
public Comparable dequeue()
 {
  if(isEmpty()) return null;
  Comparable root=h[1];
  h[1]=h[last--];
  bubbledown(1);
  return root;
 }
private void bubbledown(int pos)
 {
  // base cases
  // no children
  if(left(pos)>last) // we have no children
    return;
  // find the smallest child
  int runt=left(pos);
  if(right(pos)<=last) // the right child exists
  {
   if(h[right(pos)].compareTo(h[left(pos)])<0)
     runt=right(pos);
  }
  if(h[pos].compareTo(h[runt])<=0) return; // parent is smaller
  swap(pos,runt);
  bubbledown(runt);
 }
public Comparable getFront()
 {
  if(isEmpty()) return null;
  else return h[1];
 }
public int size() { return last; };
 public boolean isEmpty() { return (last<=0); }
 public boolean isFull() { return (last>=h.length-1); }
 public void makeEmpty() { last=0; }
}
```

From <https://blue.butler.edu/~jsorenso/oop/code/11am/Heap.java>

Episode 5

Event-loop programming:
things happen in response to user-initiated events
(pressing buttons, clicking on things, changing text, etc.)

An event handler is code that responds to a user event
In Java-swing, these are sometimes called "Listeners"

1. write the code (as a method/function in an object)
2. connect the code to the event object (button, text area, etc.)

GuI 5

```
import java.io.*;
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;

class dad extends WindowAdapter
{
 public void windowClosing(WindowEvent e)
 {
  System.out.println("It's closing time.");
  System.exit(0);
 }
}

class gui5 extends JFrame implements
ActionListener
{
 JButton north;
 JButton east;
 JTextField south;
 int count;

 public void actionPerformed(ActionEvent e)
 {
  count++;
  if(e.getSource()==north)
  {
   south.setText("North Button Pressed!!  Victory!
count="+count);
  }
  else
  {
   south.setText("East Button Pressed!!  Even more
Victory! count="+count);
  }
 }

 public gui5()
 {
  count=0;

  setTitle("GUI5: Our new gooey program!");
  setSize(500,500);
```

GuI 6

```
import java.io.*;
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;

class dad extends WindowAdapter
{
 public void windowClosing(WindowEvent e)
 {
  System.out.println("It's closing time.");
  System.exit(0);
 }
}

class gui6 extends JFrame implements
ActionListener
{
 class art extends JPanel
 {

  int suny;

  public art()
  {
   setSize(1000,600);
   suny=50;
  }

  public void time() { suny++; }

  public void paintComponent(Graphics g)
  {
   // sky
   g.setColor(Color.blue);
   g.fillRect(0,0, 1000,600);
   // sun
   g.setColor(Color.yellow);
   g.fillOval(800,suny, 100,100);
   // grass
   g.setColor(Color.green);
   g.fillRect(0,300, 1000,300);
   // house
```

```
  count=0;

  setTitle("GUI5: Our new gooey program!");
  setSize(500,500);
  addWindowListener(new dad());

  // put stuff in the window
  Container glass=getContentPane();
  glass.setLayout( new BorderLayout() ); // layout
manager

  //glass.add( new JButton("Center"), "Center" );

  north=new JButton("Center");
  east=new JButton("E");
  south=new JTextField("S");

  north.addActionListener(this);
  east.addActionListener(this);

  JPanel people=new JPanel();
  people.setLayout(new BorderLayout());
  people.add( north, "North" );
  people.add( new JLabel("Center"), "South" );

  glass.add( people, "Center" );

  glass.add( new JLabel("Victim"), "North" );
  glass.add( south, "South" );
  glass.add( east, "East" );
  glass.add( new JTextArea("W"), "West" );

  setVisible(true);
 }

 public static void main(String [] args)
 {
  gui5 gelatinousCube=new gui5();
 }
}
```

```
    g.setColor(Color.green);
    g.fillRect(0,300, 1000,300);
    // house
    g.setColor(Color.red);
    g.fillRect(100,150, 500,300);
   }
 }


 JButton north;
 JButton east;
 JTextField south;
 int count;
 art picasso;

 public void actionPerformed(ActionEvent e)
 {
  count++;
  if(e.getSource()==north)
  {
   south.setText("North Button Pressed!!  Victory!
count="+count);
    picasso.time();
  }
  else
  {
   south.setText("East Button Pressed!!  Even more
Victory! count="+count);
  }
  // forces a redraw of the picasso art object
  picasso.repaint();
 }

 public gui6()
 {
  count=0;

  setTitle("GUI6: Our new gooey program!");
  setSize(1100,700);
  addWindowListener(new dad());

  // put stuff in the window
  Container glass=getContentPane();
  glass.setLayout( new BorderLayout() ); // layout
manager

  //glass.add( new JButton("Center"), "Center" );

  north=new JButton("Center");
  east=new JButton("E");
  south=new JTextField("S");

  north.addActionListener(this);
  east.addActionListener(this);

  JPanel people=new JPanel();
  people.setLayout(new BorderLayout());
  people.add( north, "North" );
  //people.add( new JLabel("Center"), "South" );
  picasso=new art();
  people.add(picasso, "Center");

  glass.add( people, "Center" );

  glass.add( new JLabel("Victim"), "North" );
  glass.add( south, "South" );
  glass.add( east, "East" );
  glass.add( new JTextArea("W"), "West" );

  setVisible(true);
```

```
  }

  public static void main(String [] args)
  {
    gui6 gelatinousCube=new gui6();
  }
}
```