Lecture 14 - March 4

General Trees, Binary Trees

Initializing a Generic Array
Recursive Definitions of (Binary) Trees
Trees in Java: Construction, Depth

Announcements/Reminders

- ProgTest1 results to be released by Friday, Mar 14
- Makeup Lecture (on ADTs, Stacks) posted
- WrittenTest guide and example questions to be release
- Assignment 3 (on linked Trees) to be released
- Lecture notes template, Office Hours, TA Contact

General Trees: Recursive Definition

- root
- size

Case 0: Empty Tree

ϕ

root → null

size == 0

Case 1: A singleton tree

root → c. → null

implementation decision.

e.g. empty ArrayList of tree nodes

Case 2: >1 nodes

TreeNode

root → c. → 0 1 2 ... MAX

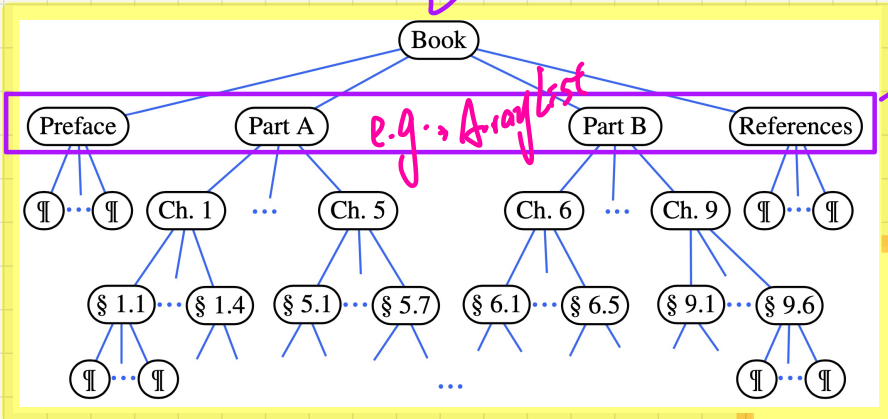
TreeNode[ ]

subtree rooted of 1st c.

subtree rooted at 2nd c.

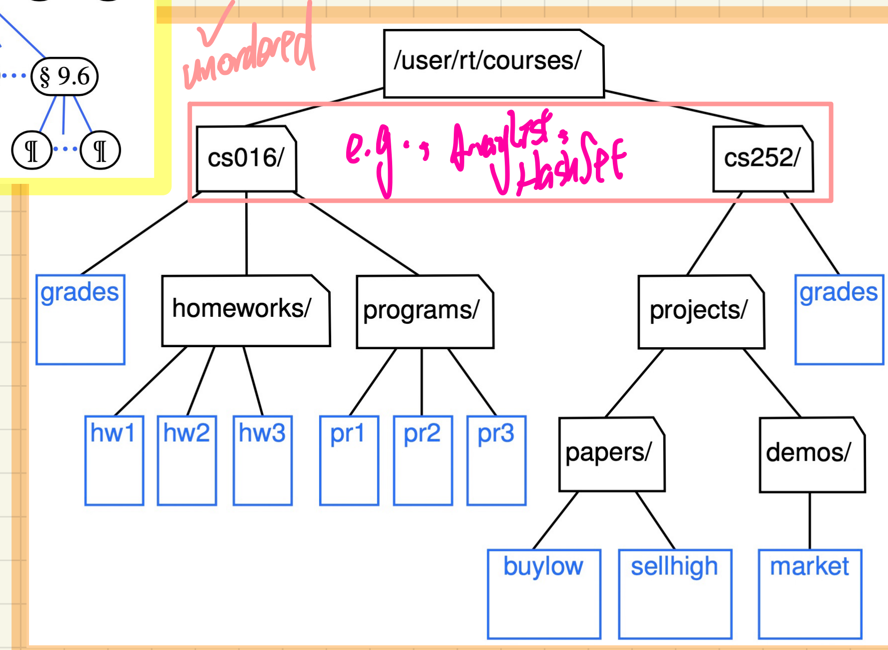
recursive structure.

General Trees: Ordered vs. Unordered Trees



→ there's a linear & logical order between a node's children.

unordered



red

↳ type

Generic, General Tree Nodes

n.children.length 10

vs.

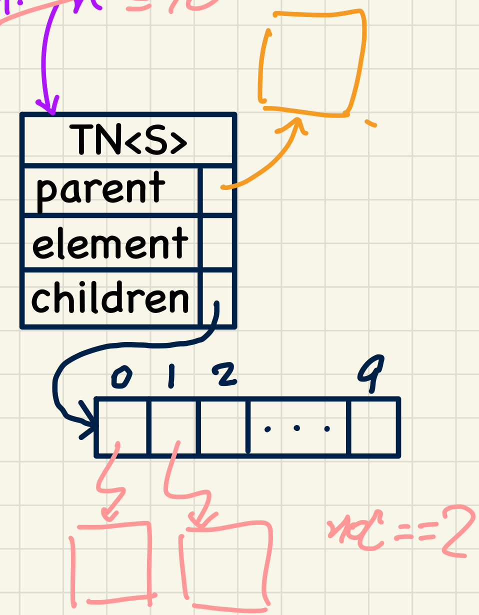
n.noc ≤ 10

```java
public class TreeNode<E> {
  private E element; /* data object */
  private TreeNode<E> parent; /* unique parent node */
  private TreeNode<E>[] children; /* list of child nodes */

  private final int MAX_NUM_CHILDREN = 10; /* fixed max */
  private int noc; /* number of child nodes */

  public TreeNode(E element) {
    this.element = element;
    this.parent = null;
    this.children = (TreeNode<E>[])
      Array.newInstance(this.getClass(), MAX_NUM_CHILDREN);
    this.noc = 0;
  }
  public E getElement() { ... }
  public TreeNode<E> getParent() { ... }
  public TreeNode<E>[] getChildren() { ... }

  public void setElement(E element) { ... }
  public void setParent(TreeNode<E> parent) { ... }
  public void addChild(TreeNode<E> child) { ... }
  public void removeChildAt(int i) { ... }
}
```



Compare:

+ prev ref.
+ next ref.

in a DLN.

Instantiating Generic Structures in Java

```
class ArrayStack<E> {
  private E[] data;
  ...
  public ArrayStack<E>() {

  }
}
```

```
class TreeNode<E> {
  private TreeNode<E>[] children;
  ...
  public TreeNode<E>() {

  }
}
```

String

0 1 2 9

null null null

alan

data[i] = 23;

data[i] = "alan"

~~data = new E[10];~~

data = (E[]) new Object[10];

workaround just for the runtime. static type of data[i] remains E

E is wrapped within the TreeNode class

~~children = (TN<E>) new Object[10];~~

children = (TN<E>) Array.newInstance(this.getClass(), 10);

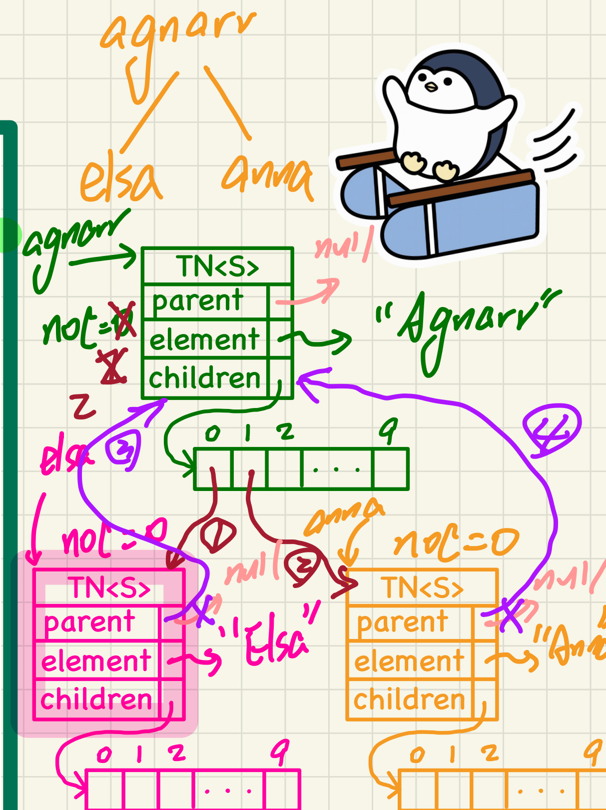
TreeNode<E>.getClass()

Tracing: Constructing a Tree

```java
@Test
public void test_general_trees_construction() {
  TreeNode<String> agnarr = new TreeNode<>("Agnarr");
  TreeNode<String> elsa = new TreeNode<>("Elsa");
  TreeNode<String> anna = new TreeNode<>("Anna");

① agnarr.addChild(elsa);
② agnarr.addChild(anna);
③ elsa.setParent(agnarr);
④ anna.setParent(agnarr);

  assertNull(agnarr.getParent());
  assertTrue(agnarr == elsa.getParent());
  assertTrue(agnarr == anna.getParent());
  assertTrue(agnarr.getChildren().length == 2);
  assertTrue(agnarr.getChildren()[0] == elsa);
  assertTrue(agnarr.getChildren()[1] == anna);
}
```



Aliasing ① elsa

② agnarr.getChildren()[0]

③ elsa.getParent().getChildren[0]

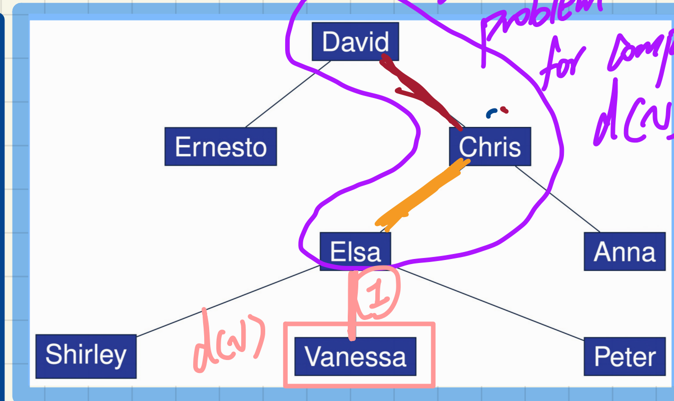
Tracing: Computing a Node's Depth

root of some subtree

```
public int depth(TreeNode<E> n) {
  if(n.getParent() == null) {
    return 0;
  }
  else {
    return 1 + depth(n.getParent());
  }
}
```

depth of root is 0

the depth of v's parent




```
@Test
public void test_general_trees_depths() {
  ... /* constructing a tree as shown above */
  TreeUtilities<String> u = new TreeUtilities<>();
  assertEquals(0, u.depth(david));
  assertEquals(1, u.depth(ernesto));
  assertEquals(1, u.depth(chris));
  assertEquals(2, u.depth(elsa));
  assertEquals(2, u.depth(anna));
  assertEquals(3, u.depth(shirley));
  assertEquals(3, u.depth(vanessa));
  assertEquals(3, u.depth(peter));
}
```

depth(vanessa)

→ .getP != null

= 1 + depth(Elsa)

→ .getP != null

= 1 + 1 + depth(Chris)

→ .getP != null

= 1 + 1 + 1 + $\boxed{\text{depth(David)}}$

0

= 3

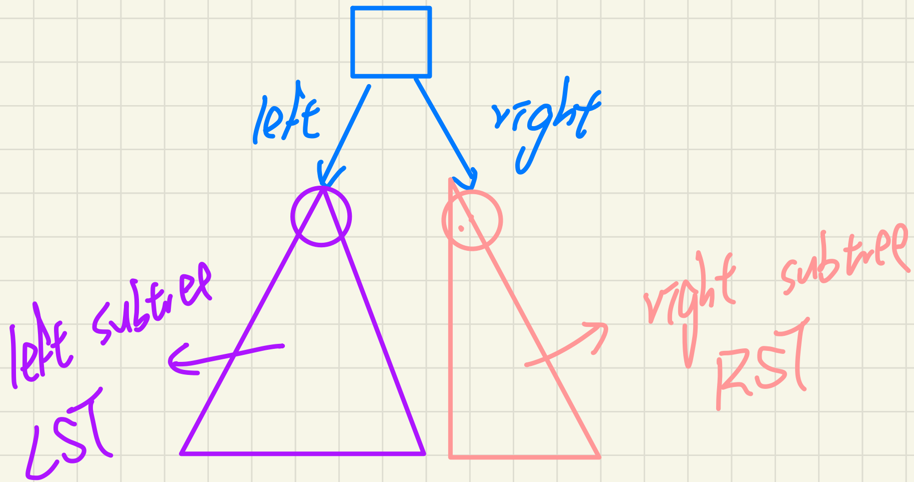
Binary Trees: Recursive Definition

- root
- size

Case 0 : Empty Tree

Case 1 : A singleton Tree

Case 2 : ≥ 2 nodes



Deriving the Sum of a Geometric Sequence

Initial Term: I

Common Factor: r

Number of Terms: k

$$\overset{2^0}{1} + \overset{2^1}{2} + \overset{2^2}{4} + 8 + 16 + \cdots + \overset{2^{10}}{1024}$$

$$1 \xrightarrow{*2} 2 \xrightarrow{*2} 4$$

$$\boxed{k = 11}$$

$$S_k = \overset{I\cdot r^0}{I} + I\cdot r + I\cdot r^2 + I\cdot r^3 + \cdots + I\cdot r^{k-1}$$

$$r\cdot S_k = I\cdot r + I\cdot r^2 + I\cdot r^3 + \cdots + I\cdot r^{k-1} + I\cdot r^k$$

$$r\cdot S_k - S_k = (r-1)\cdot S_k = I\cdot r^k - I = I\cdot(r^k - 1)$$?!?

$$S_k = \frac{I\cdot(r^k-1)}{r-1}$$

For BT: $r = 2$, $I = 1$

$S_k = r^k - 1$ → depth of "bottom" node

root