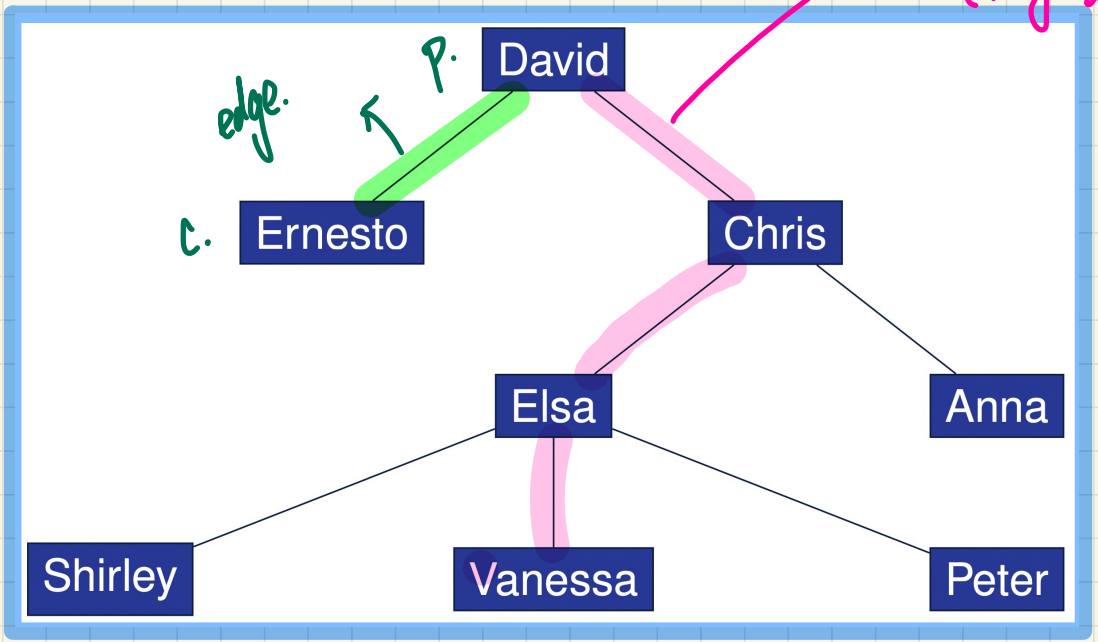
Lecture 17 - Monday, March 13

Announcements

- **ProgTest1** results to be released by Friday, March 17
- **Makeup Lecture** for WrittenTest1, ProgTest1
  + Expected to complete by: March 20

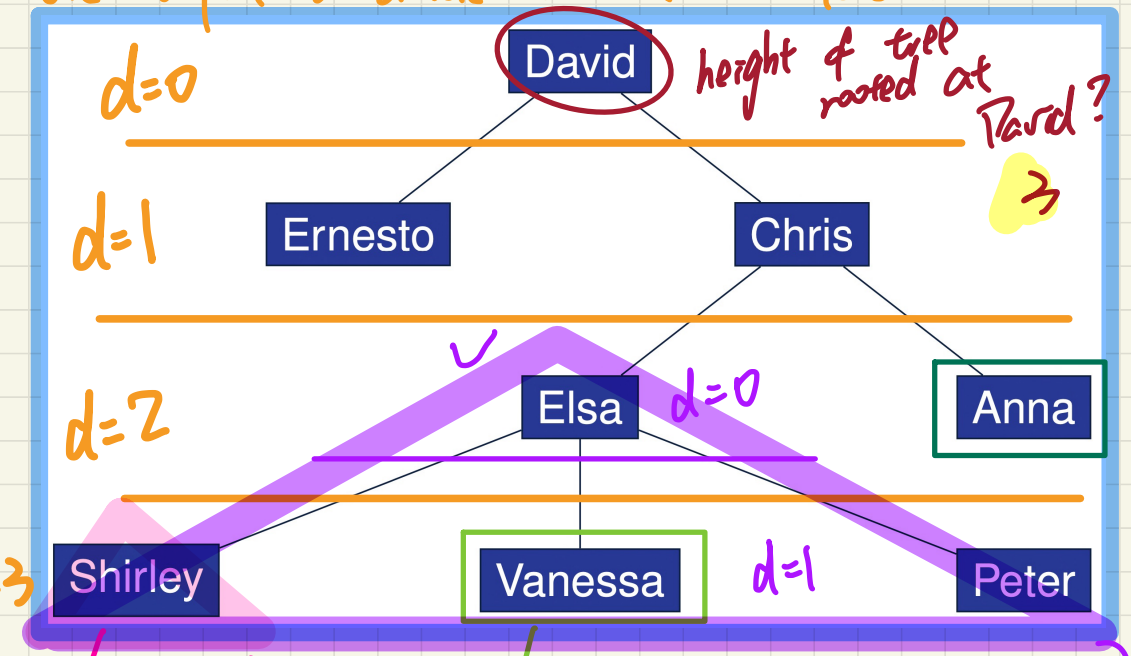
General Trees: Terminology (4)



a path (in general, as short as 1 single edge, as long as the height of tree)

- edge
- path
- depth
- height

Use "depth" to divide tree node into levels.



depth of a node

height of a (sub)tree

height of tree rooted at David?

3

d=0

d=2

d=1

height of subtree rooted at Shirley.

0

d = 3

① # of edges to the root

② # of ancestors - 1

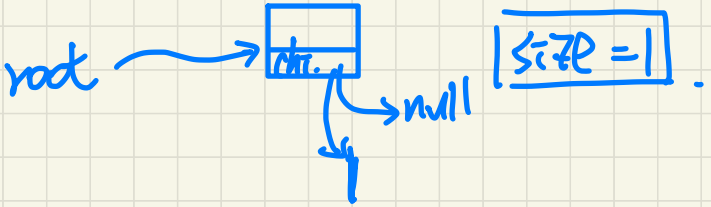
height of subtree rooted at Elsa?

1

General Trees: Recursive Definition

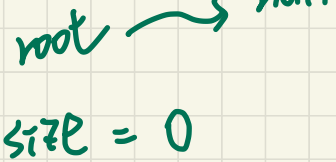
- root
- size

Case 1: A singleton tree

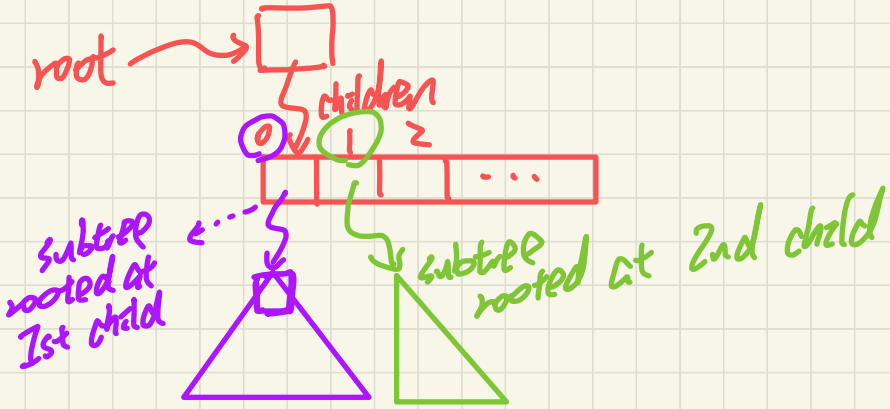


Case 0: Empty tree

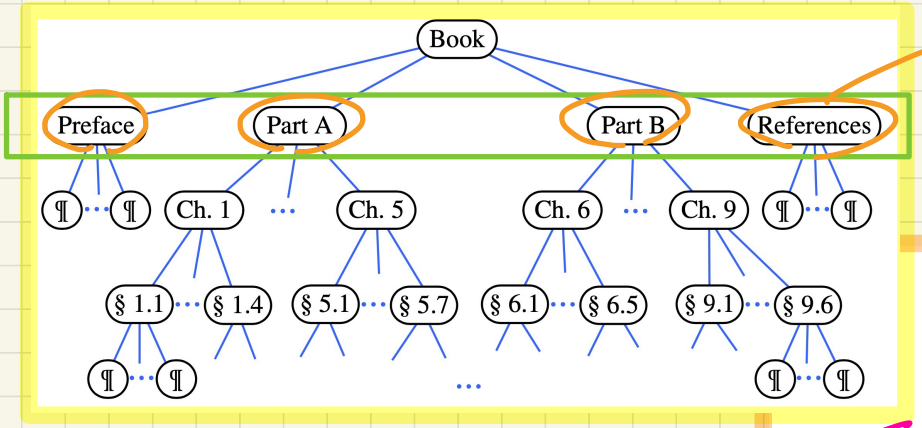
ϕ



Case 2: > 1 nodes

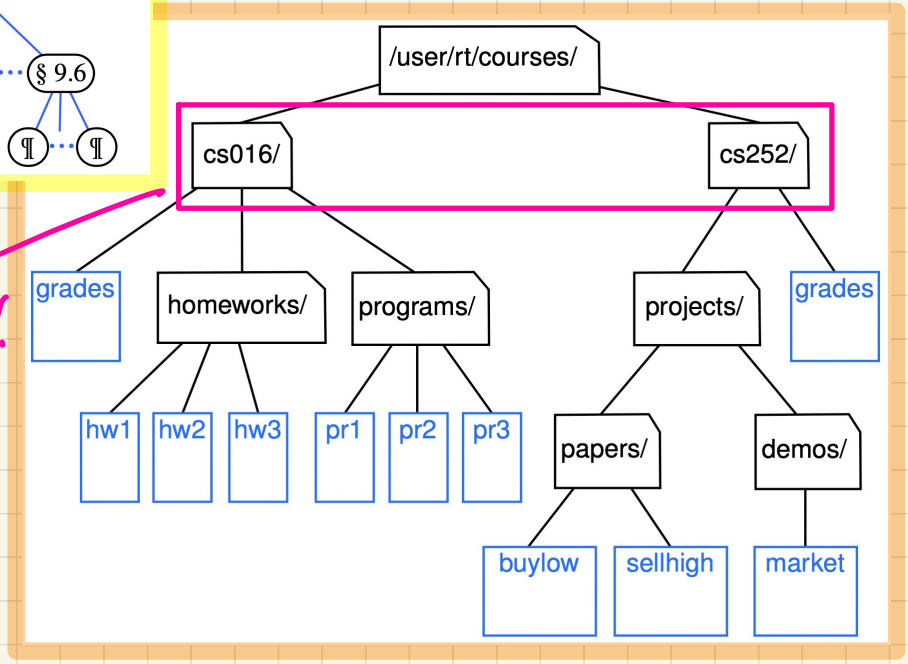


General Trees: Ordered vs. Unordered Trees



d=1

there's a linear order among children of the same level.



unordered!

red
cd
tree

Lecture

General Trees ADT

Implementing a Generic Tree in Java

Generic, General Tree Nodes

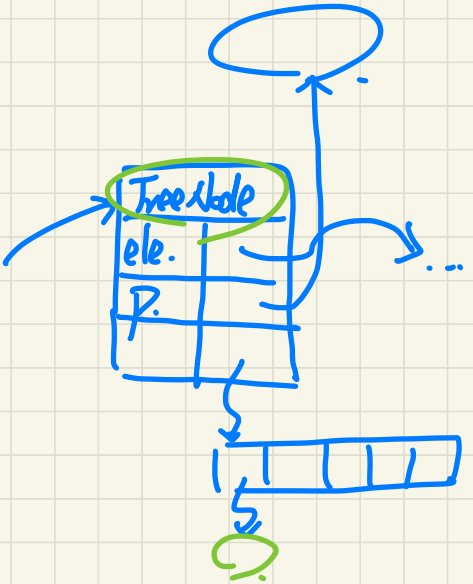
```
public class TreeNode<E> {
  private E element; /* data object */
  private TreeNode<E> parent; /* unique parent node */
  private TreeNode<E>[] children; /* list of child nodes */

  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) { ... }
}
```

# of child nodes



Compare:

+ prev ref.
+ next ref.

in a DLN.

Instantiating Generic Structures

```
class ArrayStack<E> {
    private E[] data;
    public ArrayStack<E>() {
        [          ]
        |
        v
        data = (E[]) new Object[..];
    }
```

```
class TreeNode<E> {
    private TreeNode<E>[] c;

    public TreeNode<E>() {
        c = (TreeNode<E>[])
              X
                  new Object[..];
        c = (TreeNode<E>[])
            Array.newInstance(this.getClass(),
                              ...);
```

E is wrapped within TN

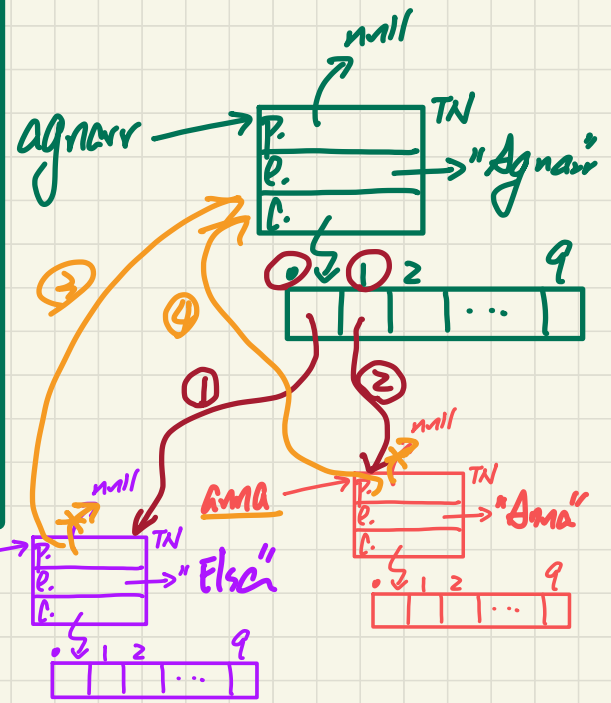
alt. TreeNode<E>.getClass()

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");

  agnarr.addChild(elsa);
  agnarr.addChild(anna);
  elsa.setParent(agnarr);
  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 agnarr.getChildren[0]

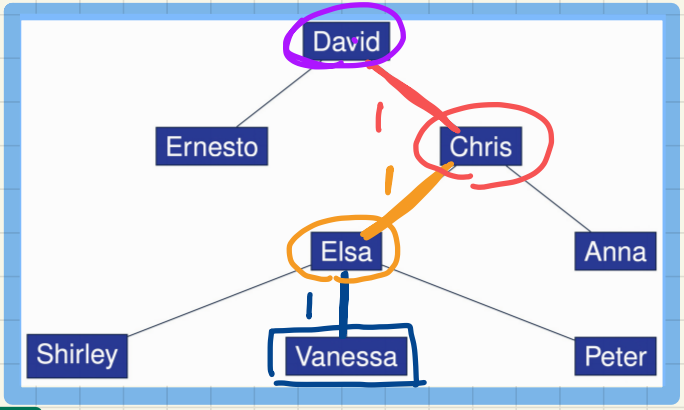
② elsa ③ elsa.getParent().getChildren[0]

Tracing: Computing a Node's Depth

arbitrary node in tree.

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

n is the root




```
@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)

= 1 + depth(Elsa)

↳ strictly smaller problem ∴ closer to root!

= 1 + 1 + depth(Chris)

= 1 + 1 + 1 + depth(David)

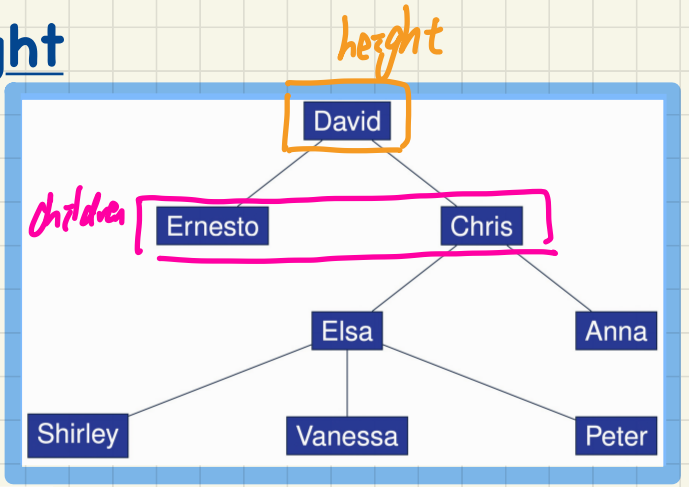
depth(David) → 0 (base case)

= 3.

Tracing: Computing a Tree's Height

```
public int height(TreeNode<E> n) {
  TreeNode<E>[] children = n.getChildren();
  if(children.length == 0) { return 0; }
  else {
    int max = 0;
    for(int i = 0; i < children.length; i ++) {
      int h = 1 + height(children[i]);
      max = h > max ? h : max;
    }
    return max;
  }
}
```

recursive call of height



height(chris)

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