Binary Search Trees

- Definitions and terminologies
- Search and insert
- Traversal
- Ordered operations
- Delete
  - Delete minimum and maximum
  - Delete a given key
Hibbard deletion

To delete a node with key \( k \): search for node \( t \) containing key \( k \).

Case 0. [0 children]

```
 deleting C
```

```
node to delete
```
Hibbard deletion

To delete a node with key $k$: search for node $t$ containing key $k$.

**Case 0. [0 children]** Delete $t$ by setting parent link to null.
Hibbard deletion

To delete a node with key $k$: search for node $t$ containing key $k$.

Case 1. [1 child]
Hibbard deletion

To delete a node with key $k$: search for node $t$ containing key $k$.

Case 1. [1 child] Delete $t$ by replacing parent link.
Hibbard deletion

To delete a node with key $k$: search for node $t$ containing key $k$.

Case 2. [2 children]
Hibbard deletion

To delete a node with key \( k \): search for node \( t \) containing key \( k \).

**Case 2.** [2 children]

- Find successor \( x \) of \( t \).
- Delete the minimum in \( t \)'s right subtree.
- Put \( x \) in \( t \)'s spot.
public void delete(Key key) {
    root = delete(root, key);
}

private Node delete(Node x, Key key) {  
    if (x == null) return null;
    int cmp = key.compareTo(x.key);
    if (cmp < 0) x.left = delete(x.left, key);
    else if (cmp > 0) x.right = delete(x.right, key);
    else {
        if (x.right == null) return x.left;

        Node t = x;
        x = min(t.right);
        x.right = deleteMin(t.right);
        x.left = t.left;
    }
    x.N = size(x.left) + size(x.right) + 1;
    return x;
}
Hibbard deletion: analysis

Unsatisfactory solution. Not symmetric.

Surprising consequence. Trees not random (!) $\Rightarrow \sqrt{N}$ per op.
Longstanding open problem. Simple and efficient delete for BSTs.
### ST implementations: summary

<table>
<thead>
<tr>
<th>implementation</th>
<th>guarantee</th>
<th>average case</th>
<th>ordered iteration?</th>
<th>operations on keys</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>search</td>
<td>insert</td>
<td>delete</td>
<td>search hit</td>
</tr>
<tr>
<td>sequential search (linked list)</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N/2</td>
</tr>
<tr>
<td>binary search (ordered array)</td>
<td>lg N</td>
<td>N</td>
<td>N</td>
<td>lg N</td>
</tr>
<tr>
<td>BST</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>1.39 lg N</td>
</tr>
</tbody>
</table>

Other operations also become √N if deletions allowed.
Hw5

• BST tree class (to be implemented)
  – BST tree node class
    • key (String type for short name)
    • data (MovieInfo type)
    • left and right children
  – root to the tree
  – methods
    • Insert()
    • findExact()
    • findPrefix()

• IndexTester (provided)
  – Creates an empty BST tree
  – Reads the input movies or actors files
  – Builds a MovieInfo object for each row, insert it into the BST tree
  – Asks for user search string and search for the MovieInfo object