SEARCHING AND TREES

- COMP1927 Computing 16x1
- Sedgewick Chapters 5, 12
SEARCHING

- Storing and searching sorted data:
- Dilemma: Inserting into a sorted sequence
  - Finding the insertion point on an array – $O(\log n)$ but then we have to move everything along to create room for the new item
  - Finding insertion point on a linked list $O(n)$ but then we can add the item in constant time.
- Can we get the best of both worlds?
TREE TERMINOLOGY

- Trees are branched data structures consisting of nodes and links (edges), with no cycles.
- Each node contains a data value.
- Each node has links to $\leq k$ other nodes ($k=2$ below).

![Diagram of a tree structure with nodes labeled 1, 2, 3, 4, 6, 8, and 9. The node 4 is the root, 2 and 8 are internal nodes, and 1, 3, 6, and 9 are leaves. Connections illustrate parent-child relationships.]
TREES AND SUBTREES

- Trees can be viewed as a set of nested structures: each node has $k$ possibly empty subtrees.
USES OF TREES

- Trees are used in many contexts, e.g. representing hierarchical data structures (e.g. expressions)
- Efficient searching (e.g. sets, symbol tables, ...)

![Search Tree](image1.png)

![Expression Tree](image2.png)
SPECIAL PROPERTIES OF SOME TREES

- **M-ary tree**: each internal node has exactly M children
- **Ordered tree**: constraints on the data/keys in the nodes
- **Balanced tree**: a tree with a minimal height for a given number of nodes
- **Degenerated tree**: a tree with the maximal height for a given number of nodes
Binary Trees

For much of this course, we focus on binary trees \((k=2)\) Binary trees can be defined recursively, as follows:

- A *binary tree* is either
  - empty (contains no nodes)
  - consists of a node, with two subtrees
    - each node contains a value
    - the left and right subtrees are *binary trees*
...TREE TERMINOLGY

○ Node level or depth = path length from root to node
  • Depth of the root is 0
  • Depth of a node is one higher than the level of its parent

○ We call the length of the longest path from the root to a node the **height** of a tree
**Binary Trees: Properties**

- A binary tree with $n$ nodes has a height of
  - at most
    - $n-1$ (if degenerate)
  - at least
    - $\text{floor}(\log_2(n))$ (if balanced)

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>6</th>
<th>9</th>
<th>13</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\log_2 10$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\log_2 100$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\log_2 1000$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\log_2 10000$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\log_2 100000$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

These properties are important to estimate the runtime complexity of tree algorithms!
EXAMPLES OF BINARY SEARCH TREES:

- Shape of tree is determined by the order of insertion
EXERCISE: INSERTION INTO BSTs

- For each of the sequences below start from an initially empty binary search tree
  - show the tree resulting from inserting the values in the order given
  - What is the height of each tree?
- (a) 4 2 6 5 1 7 3
- (b) 5 3 6 2 4 7 1
- (c) 1 2 3 4 5 6 7
A binary tree is a generalisation of a linked list:

- nodes are a structure with two links to nodes
- empty trees are NULL links

```c
typedef struct treenode *Treelink;

struct treenode {
    int data;
    Treelink left, right;
};
```
SEARCHING IN BSTs

- Recursive version
  // Returns non-zero if item is found, 
  // zero otherwise
  int search(TreeLink n, Item i) {
    int result;
    if (n == NULL) {
      result = 0;
    } else if (i < n->data) {
      result = search(n->left, i);
    } else if (i > n->data) {
      result = search(n->right, i);
    } else { // you found the item
      result = 1;
    }
    return result;
  }

* Exercise: Try writing an iterative version
INSERTION INTO A BST

- Cases for inserting value \( V \) into tree \( T \):
  - \( T \) is empty, make new node with \( V \) as root of new tree
  - root node contains \( V \), tree unchanged (no dupes)
  - \( V < \) value in root, insert into left subtree (recursive)
  - \( V > \) value in root, insert into right subtree (recursive)

- Non-recursive insertion of \( V \) into tree \( T \):
  - search to location where \( V \) belongs, keeping parent
  - make new node and attach to parent
  - whether to attach L or R depends on last move
**Binary Trees: Traversal**

- For trees, several well-defined visiting orders exist:
  - Depth first traversals
    - preorder (NLR) ... visit root, then left subtree, then right subtree
    - inorder (LNR) ... visit left subtree, then root, then right subtree
    - postorder (LRN) ... visit left subtree, then right subtree, then root
  - Breadth-first traversal or level-order ... visit root, then all its children, then all their children
EXAMPLE OF TRAVERSALS ON A BINARY TREE

- Pre-Order: 4 2 1 3 8 6 9
- In-Order: 1 2 3 4 6 8 9
- Post-Order: 1 3 2 6 9 8 4
- Level-Order: 4 2 8 1 3 6 8
DELETION FROM BSTS

- Insertion into a binary search tree is easy:
  - find location in tree where node to be added
  - create node and link to parent

- Deletion from a binary search tree is harder:
  - find the node to be deleted and its parent
  - unlink node from parent and delete
  - replace node in tree by ... ???
DELETION FROM BSTS...

- Easy option ... don't delete; just mark node as deleted
  - future searches simply ignore marked nodes
- If we want to delete, three cases to consider ...
  - zero subtrees ... unlink node from parent
  - one subtree ... replace node by child
  - two subtrees ... two children; one link in parent
DELETION FROM BSTS

- Case 1: value to be deleted is a leaf (zero subtrees)

delete k ...

```
       m
  g       t
 /     \   /   /
b      k  p   v
   \     /     /   v
    \   /     /     /  v
     \ /     /     /   v
      /       /     /    v
     /         /     /
    /          /     
   /            /     
  /              /     
 /                /     
/                  /     
```

DELETION FROM BSTS

- Case 1: value to be deleted is a leaf (zero subtrees)

deleted k ...
DELETION FROM BSTS

- Case 2: value to be deleted has one subtree

delete p ...
DELETION FROM BSTS

- Case 2: value to be deleted has one subtree

deleted p ...
DELETION FROM BSTS

- Case 3a: value to be deleted has two subtrees
- Replace deleted node by its immediate successor
  - The smallest (leftmost) node in the right subtree

delete m ...
DELETION FROM BSTS

- Case 3a: value to be deleted has two subtrees

deleted m (v2) ...
Binary Search Tree Properties

- Cost for searching/deleting:
  - Worst case: key is not in BST – search the height of the tree
    - Balanced trees – $O(\log_2 n)$
    - Degenerate trees – $O(n)$

- Cost for insertion:
  - Always traverse the height of the tree
    - Balanced trees – $O(\log_2 n)$
    - Degenerate trees – $O(n)$