k-ary Trees

- In a **k-ary tree**, every node has between 0 and $k$ children.
- In a **full (proper)** k-ary tree, every node has exactly 0 or $k$ children.
- In a **complete** k-ary tree, every level is entirely filled, except possibly the deepest, where all nodes are as far left as possible.
- In a **perfect** k-ary tree, every leaf has the same depth and the tree is full.
How to implement binary trees?

Node:
- data
- parent
- left child
- right child

Collections/Dictionaries as arrays

<table>
<thead>
<tr>
<th>What?</th>
<th>Sequential Search (unordered sequence)</th>
<th>Binary Search (ordered sequence)</th>
</tr>
</thead>
<tbody>
<tr>
<td>search</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>insert</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>delete</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>min/max</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>floor/ceiling</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>rank</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
</tr>
</tbody>
</table>

Binary Search Trees

- A BST is a **binary tree**
- A BST has **symmetric order**
  - each node $x$ in a BST has a key $\text{key}(x)$
  - for all nodes $y$ in the left subtree of $x$, $\text{key}(y) < \text{key}(x)$ **
  - for all nodes $y$ in the right subtree of $x$, $\text{key}(y) > \text{key}(x)$ **

(**) assume that the keys of a BST are pairwise distinct
class BSTNode {
private:
  int data;
  BSTNode *left;
  BSTNode *right;
public:
  BSTNode(int d);
  ~BSTNode();
friend class BSTree;
};

class BSTree {
private:
  unsigned int size;
  BSTNode *root;
  void destroy(BSTNode *p);
public:
  BSTree();
  ~BSTree();
  void insert(int d);
  void remove(int d);
  BSTNode *search(int d);
};

Search

· Start at root node

· If the search key matches the current node’s key then found

· If search key is greater than current node’s key
  · search recursively on right child

· If search key is less than current node’s key
  · key search recursively on left child

· Stop recursion when current node is NULL (not found)
### Search: Iterative Algorithm

- Perform a Search operation
- If **found**, no need to insert (may increase counter)
- If **not found**, insert node where Search stopped

---

### Search: Recursive Algorithm

---

### Insert

- Perform a Search operation
- If **found**, no need to insert (may increase counter)
- If **not found**, insert node where Search stopped

---

**Diagram:**

![Binary Search Tree](image)
Insert: Iterative Algorithm

Insert: Recursive Algorithm