16.3 BST Definitions

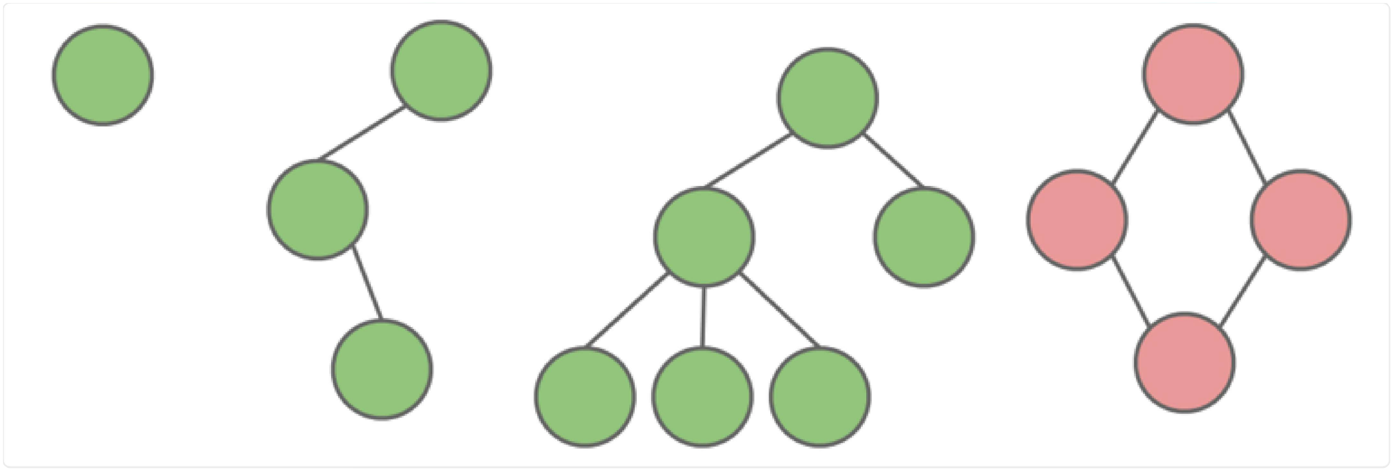
Trees are composed of nodes and edges that connect those nodes.

Constraint: there is only one path between any two nodes.

In some trees, we select a **root** node which is a node that has no parents.

A tree also has **leaves**, which are nodes with no children.

In the picture below, the green structures are valid trees, while the pink structure is not (since it has a cycle).

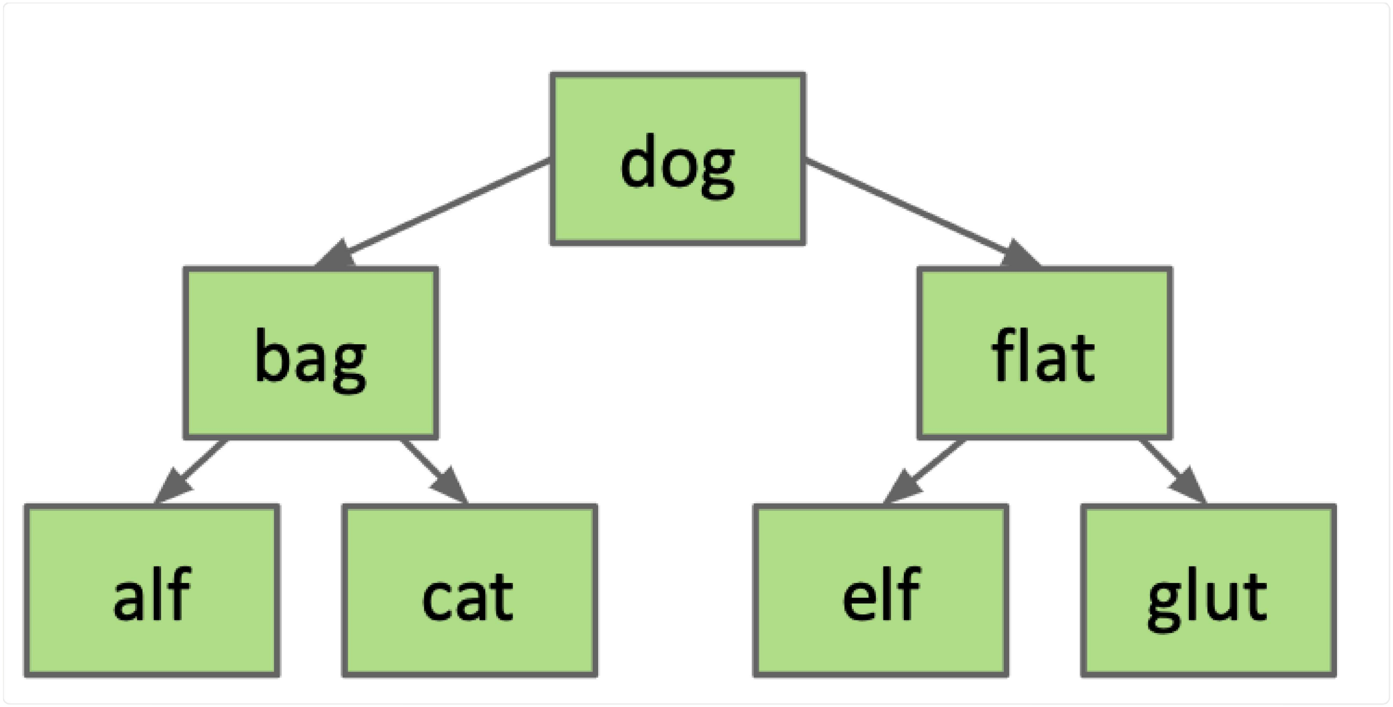


Examples of valid and invalid trees

Relating this to the original tree structure we came up with earlier, we can now introduce new constraints to the already existing constraints. This creates more specific types of trees, two examples being Binary Trees and Binary Search Trees.

- **Binary Trees**: in addition to the above requirements, also hold the binary property constraint. That is, each node has either 0, 1, or 2 children.
- **Binary Search Trees**: in addition to all of the above requirements, also hold the property that For every node X in the tree:
    - Every key in the left subtree is less than X's key.
    -

Every key in the right subtree is greater than X's key. **Remember this property!! We will reference it a lot throughout the duration of this module and 61B.



A valid BST: every key in the left subtree is smaller than its parent, and every key in the right subtree is larger.

Here is the BST module we'll be using for the rest of this chapter:

```
private class BST<Key> {
    private Key key;
    private BST left;
    private BST right;

    public BST(Key key, BST left, BST Right) {
        this.key = key;
        this.left = left;
        this.right = right;
    }

    public BST(Key key) {
        this.key = key;
    }
}
```

Previous
16.2 Binary Search Trees

Next

16.4 BST Operations