Guest lecture: Joshua Brody

Outline:

- Review Dictionary ADT
- Review Binary Search Trees
- BST implementation:
  getMinKey, update, get, insert, [remove]

Dictionary ADT

- maps keys ⇒ values
- all keys are unique
- Core Dictionary operations insert, get, remove
  must be fast

Tree terminology Recap

root

parent

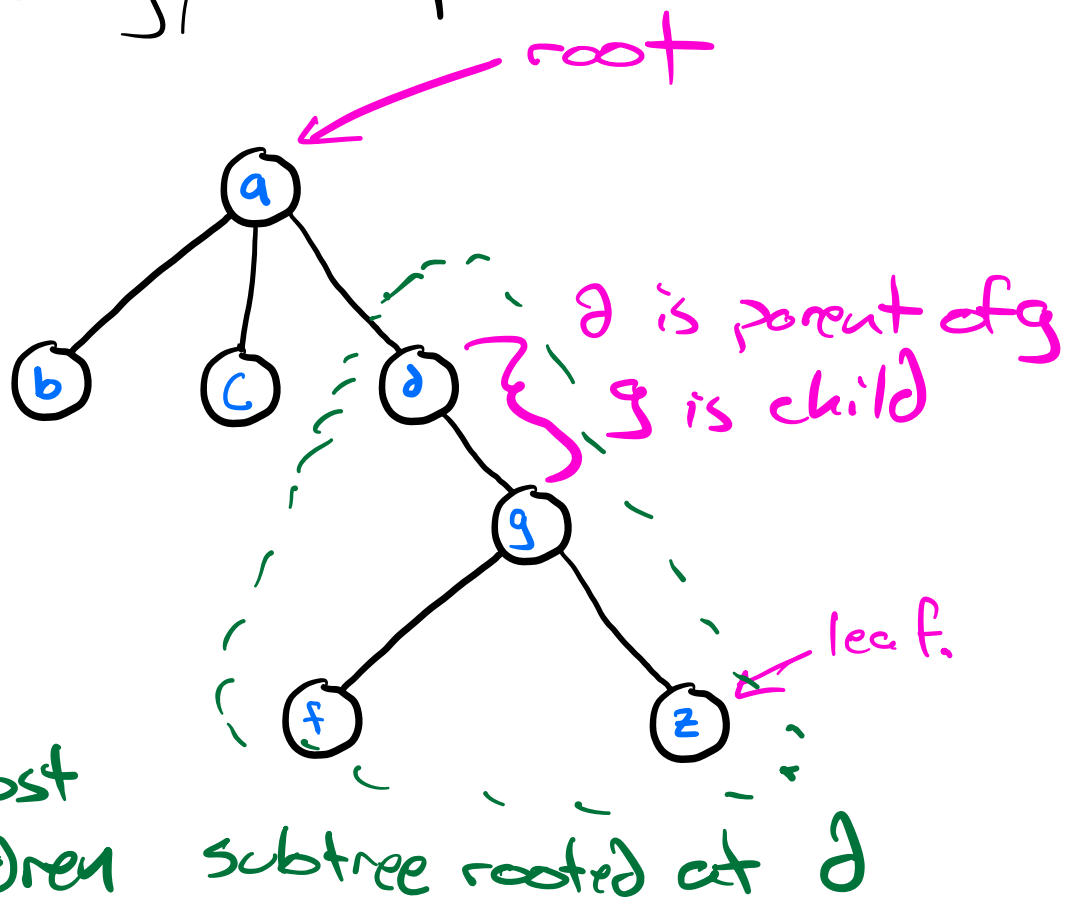
child

leaf

subtree

binary tree

↳ each node has at most two children



Binary Search Trees (BST)

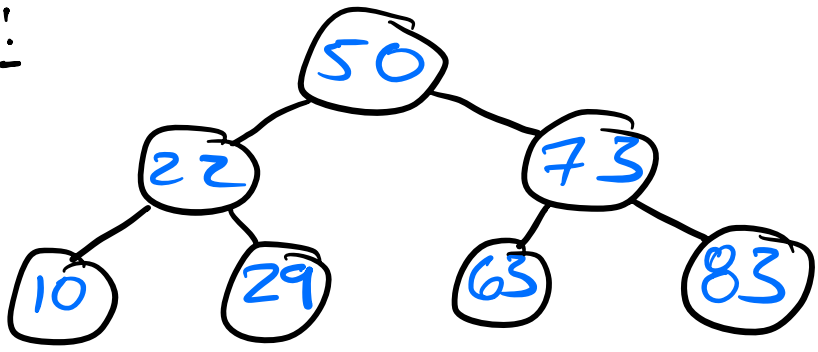
- A binary tree
- **Invariant:** the **BST property** applies to every node

BST Property:

For every node N in a BST:

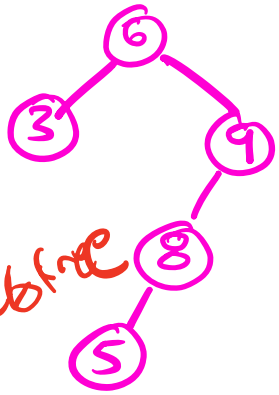
- every key in left subtree of N must be < N's key
- every key in right subtree of N must be > N's key

example:



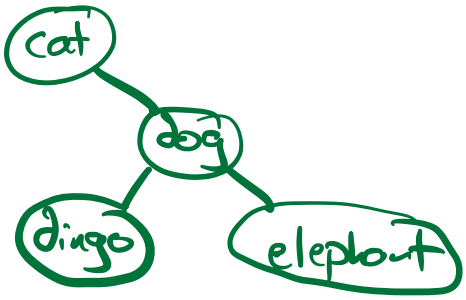
Exercise: which of the following are BSTs?

(a) ✗

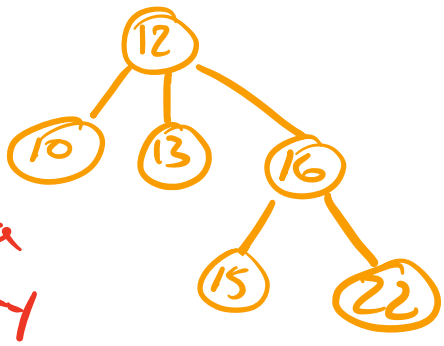


5<6 but on right subtree

(b) ✓

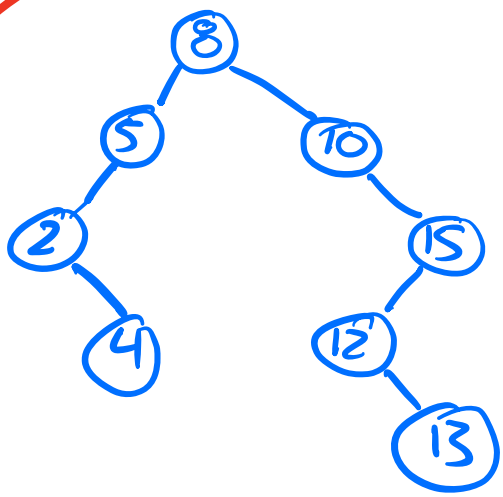


(c) ✗

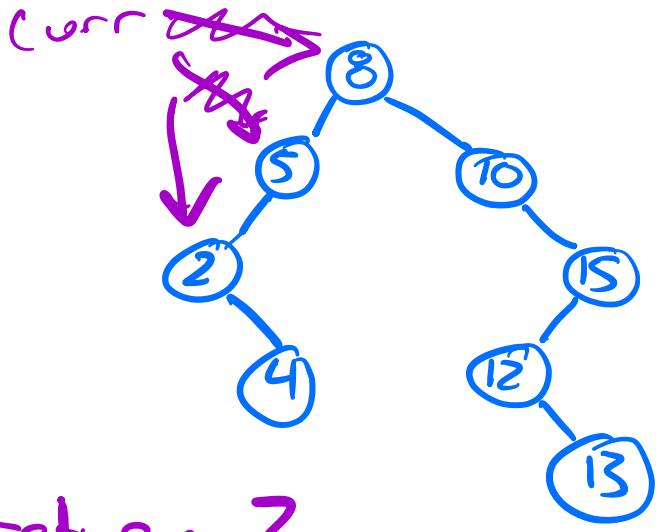


not a binary tree :(

(d) ✓



Exercise Develop pseudocode for getMinKey()



return 2.

```
getMinKey()
    Start at root
    while a left child exists
        go to left.
```

Q: Can min key ever happen after a right turn?

BST implementation templated on K, V

LinkedBSTNode<K,V>

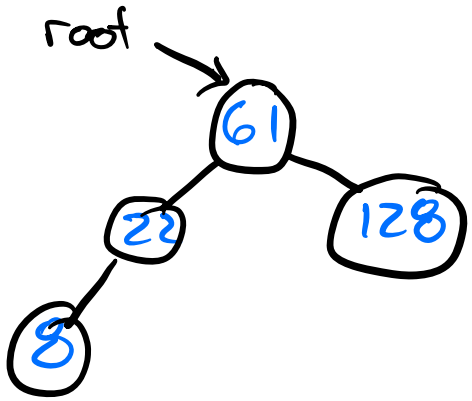
- K key
- V value
- LinkedBSTNode<K,V> *left
- LinkedBSTNode<K,V> *right

LinkedBST<K,V>

- LinkedBSTNode<K,V> *root
- int size

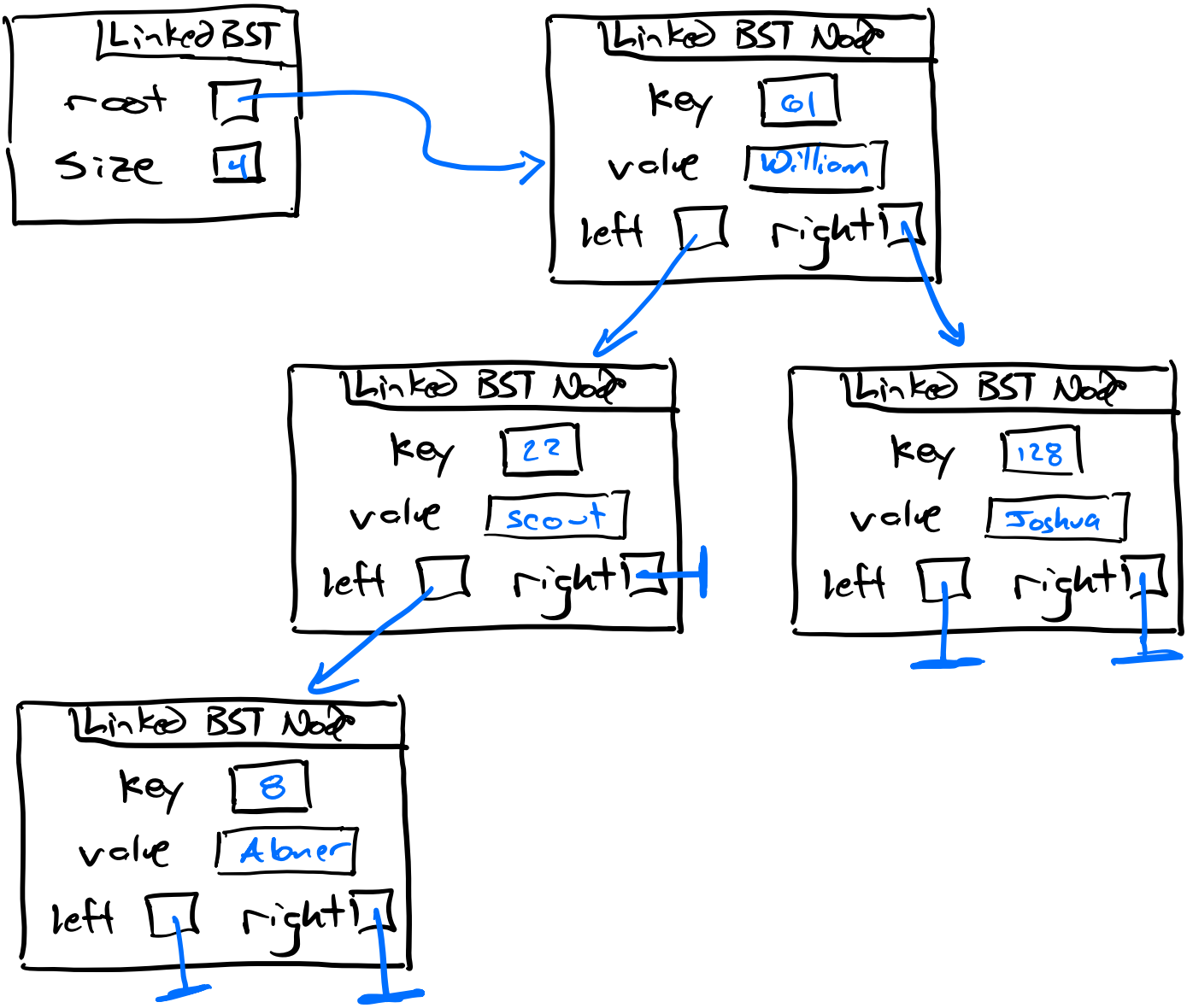
example:

61 → "William"
22 → "Scout"
8 → "Abner"
128 → "Joshua"



Note: shorthand, values not shown

Memory Diagram



Implementing BST methods

- BSTs are recursive structures, so it makes sense to implement methods using recursion
- Often w/ recursion we'll have one method initiate recursive process, and another helper method that does the work.

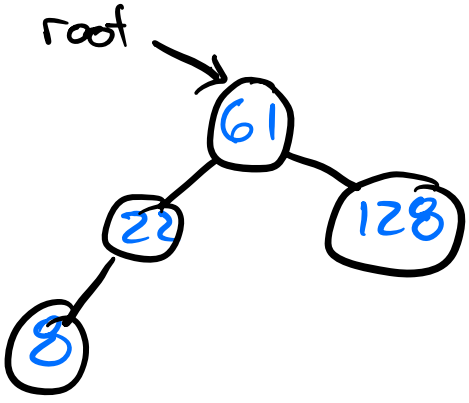
Let's work on pseudocode for implementing the get method:

```
V get(K key)
```

This searches through the BST for the given key and returns the associated value when key is found.

Note: if key not found, get(...) should throw an error.

Ex: get(24)



Pseudocode for get

```
// public method
V get(K key)
    return findInSubtree(root, key)

// private helper method
V findInSubtree(current, key)
    // base cases
    if current is nullptr
        throw error "key not found"
    if key is current's key
        return current value
    // recursive cases
    if key < current key
        return findInSubtree(current's left,
                             key)
    else // key > current key
        return findInSubtree(current's right,
                             key)
```

Exercise: give pseudocode for update(key, value)

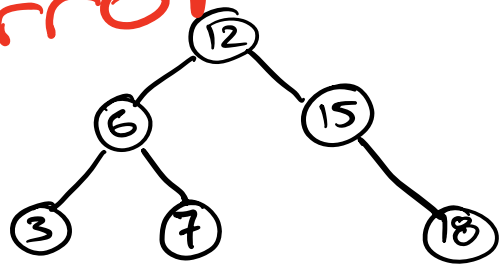
```
update(key, value)
    updateInSubtree(root, key, value)

updateInSubtree(current, key, value)
    if current is NULL
        throw error "key not found"
    if key == current key
        current value = value

    if key < current key
        updateInSubtree(current's
            left key, key, value)
    else // key > current key
        updateInSubtree(current-right
            key value)
```

Now let's focus on **insert**

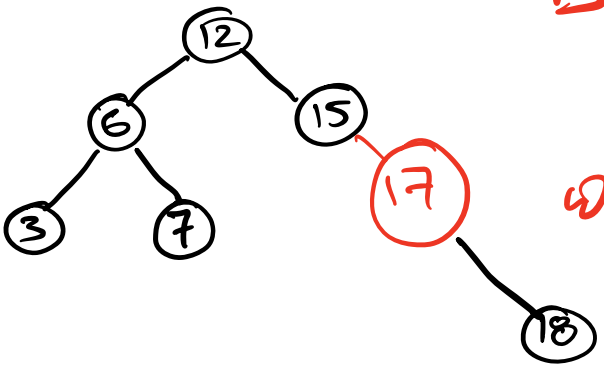
`void insert(K key, V value)`

insert(6)

throw error "duplicate key"

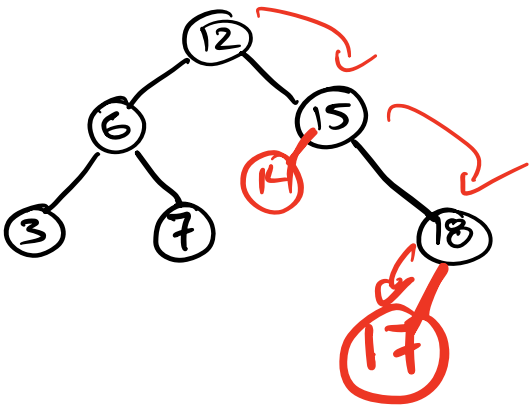


insert(17)

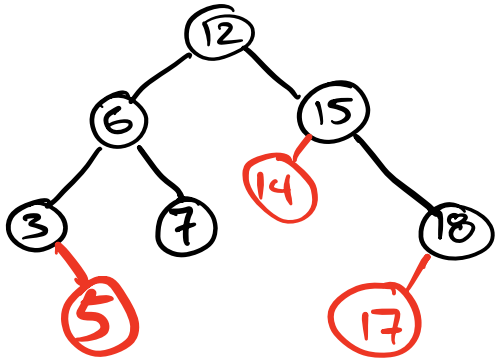


✓BST property maintained

Warning: need to check left subtree of 18 to make sure BST property holds



insert(5)



Pseudocode for insert

```
// Public method
void insert(key, value)
    // insert changes tree structure, so update root
    root = insertInSubtree(root, key, value)
    increment size

// private helper function
Node* insertInSubTree(current, key, value)
    // base cases
    if current is nullptr
        return new LinkedBSTNode(key, value)
    if key is current's key
        throw error "keys must be unique"
    // recursive cases
    if key < current's key
        current's left =
            insertInSubtree(current's left child,
                            key, value)
    else
        current's right=
            insertInSubtree(current's right child,
                            key, value)
    return current // always return ptr to root of subtree
```

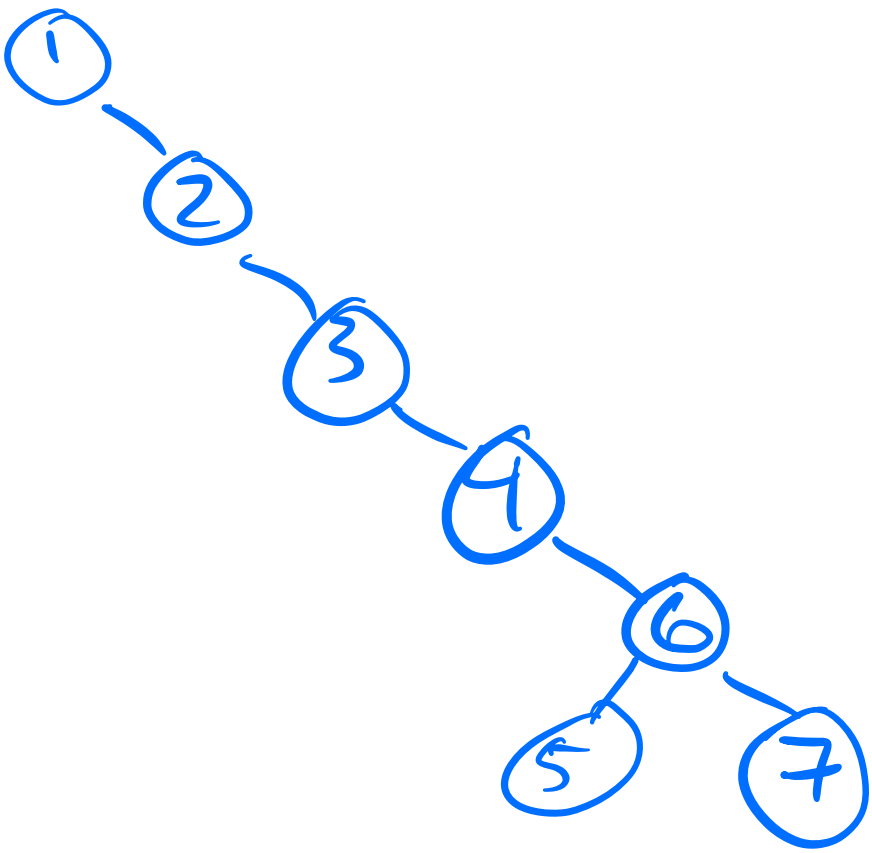
Conventions — for recursive helper methods, pass in ptr to root of current subtree
— return ptr to root of (modified) subtree

Note: must set child ptr to root of new subtree

Q: what is runtime of get? update? insert?

A: we want to exploit BST property to get binary search like $O(\log n)$ runtime.

Bad news: tree need not be balanced



runtime: $O(\text{height})$

Goodnews:

Possible to implement BSTs that guarantee height is $O(\log n)$