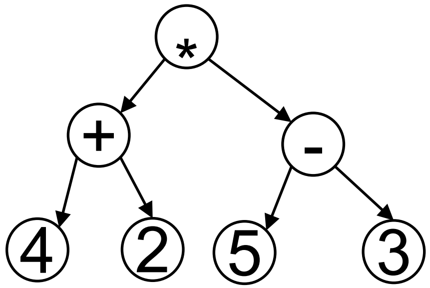
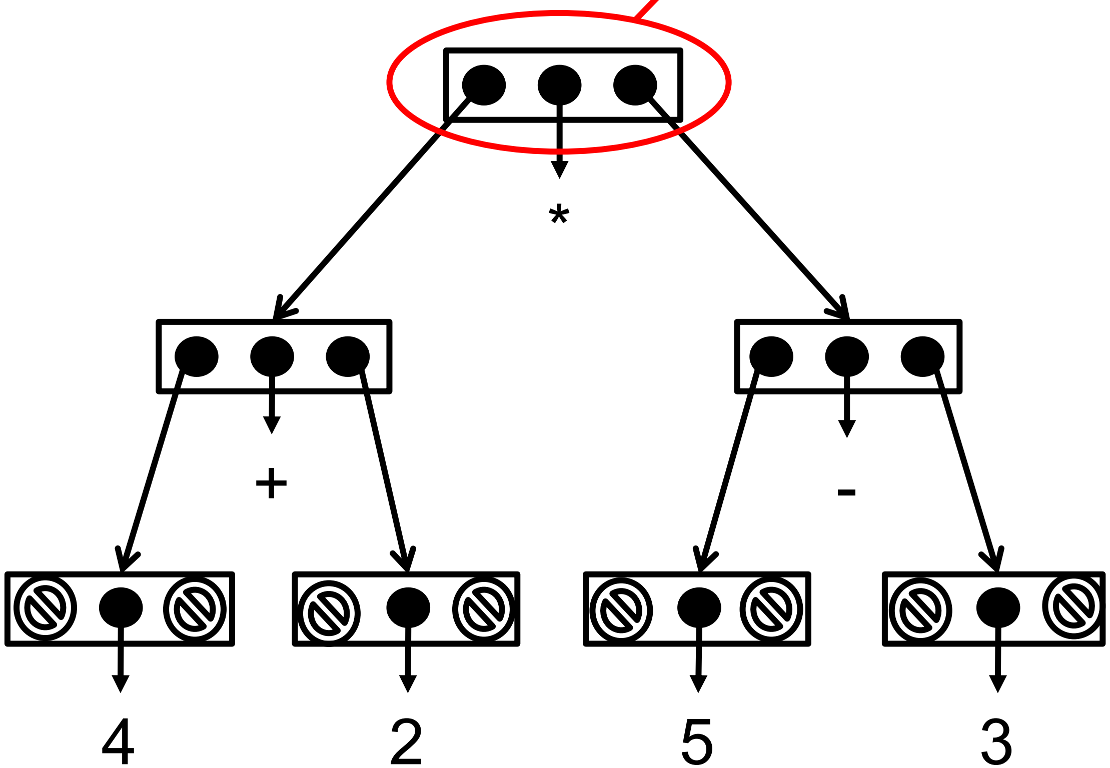
Binary Search Trees

Recap
Definition of binary search tree
Searching a binary search tree
Adding to a binary search tree

Reminder: BinaryTreeNode



Note: these are not lists; they are object instances of the BinaryTreeNode class



BinaryTreeNode
element leftchild rightchild

Let's look at searching again …

	Array-based	Linked list
Arbitrary access	O(1)	O(n)
Add at end	O(n) worst case O(1) on average	O(1)
Add in middle	O(n)	O(1) if given reference O(n) if given position
Delete from end	O(n) worst case O(1) on average	O(1) if given reference O(n) if given position
Delete from middle	O(n)	O(1) if given reference O(n) if given position
sorting	(see semester 2)	(see semester 2)
searching	Unordered: O(n) Ordered: O(log n)	Unordered: O(n) Ordered: O(n)

Bad!

Good!

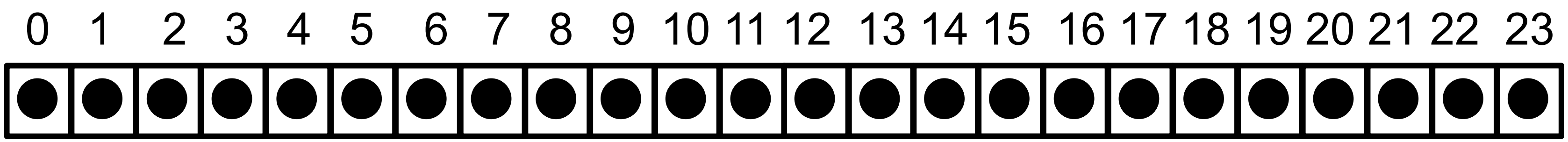
Good!

Bad!

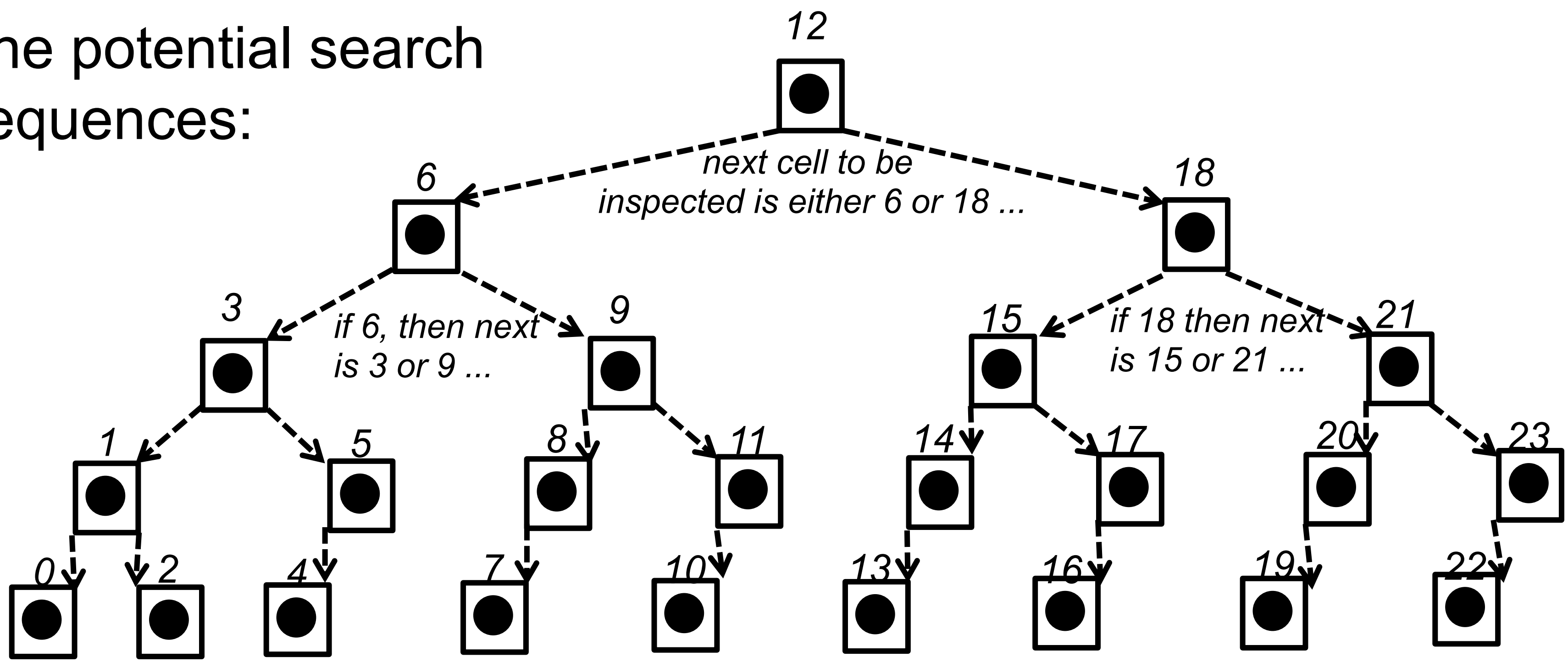
Can we build a linked structure to get the search complexity of array-based lists, but with faster add/delete?

Sequence of cell lookups in binary search

A sorted list:



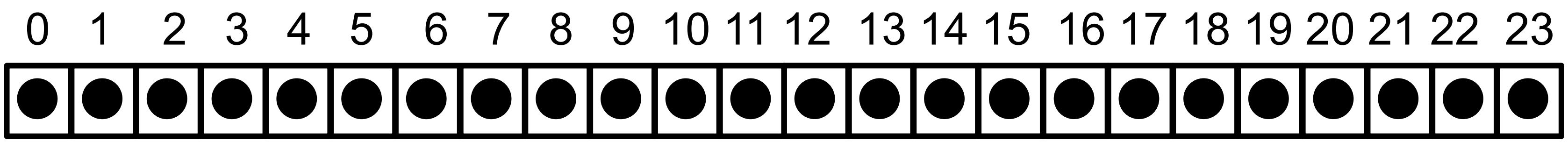
The potential search sequences:



So can we create a linked tree to replicate these possible search sequences (and then try to have efficient add/delete)?

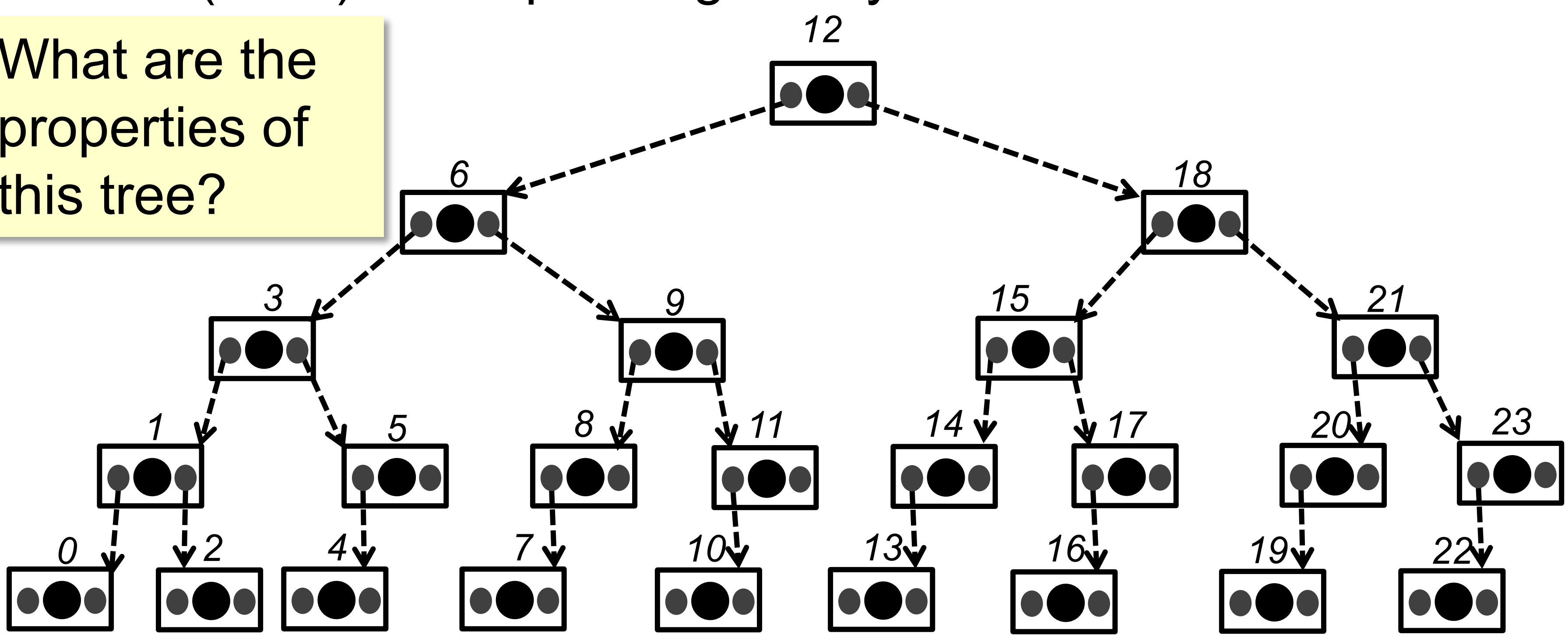
Linked structures for efficient searching?

A sorted list:



And its (ideal) corresponding binary search tree:

What are the properties of this tree?



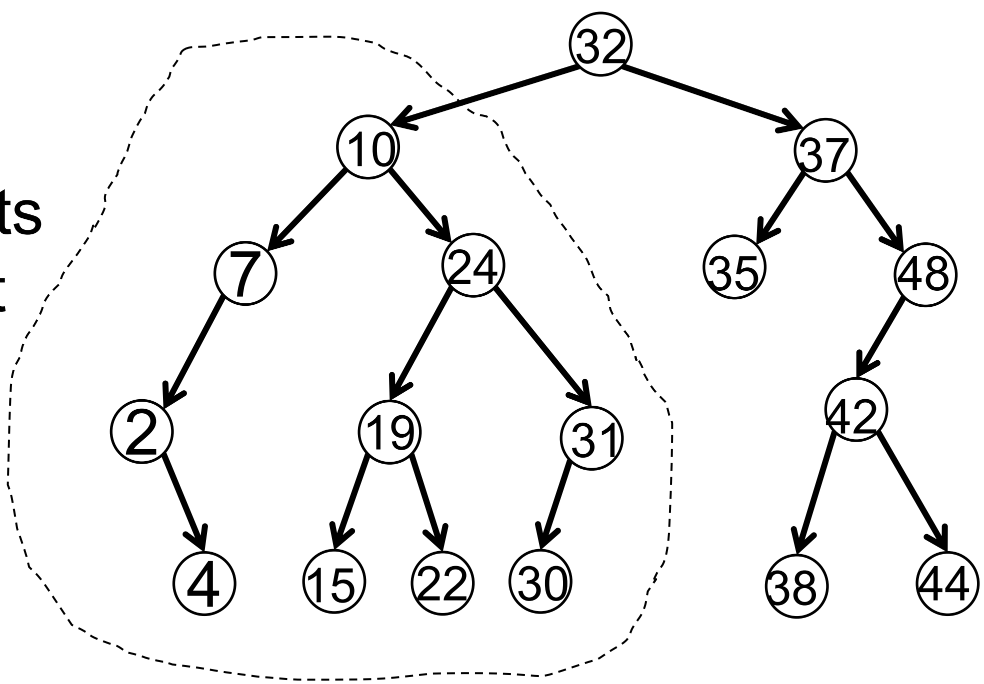
Binary Search Tree

A *binary search tree* is a binary tree representing an ordered sequence of elements, where:

all left descendants of a node have values less than the node's value

all right descendants of a node have values greater than the node's value

left descendants of the root node



Note! The arrows represent the tree structure, not the order of the elements in the sequence

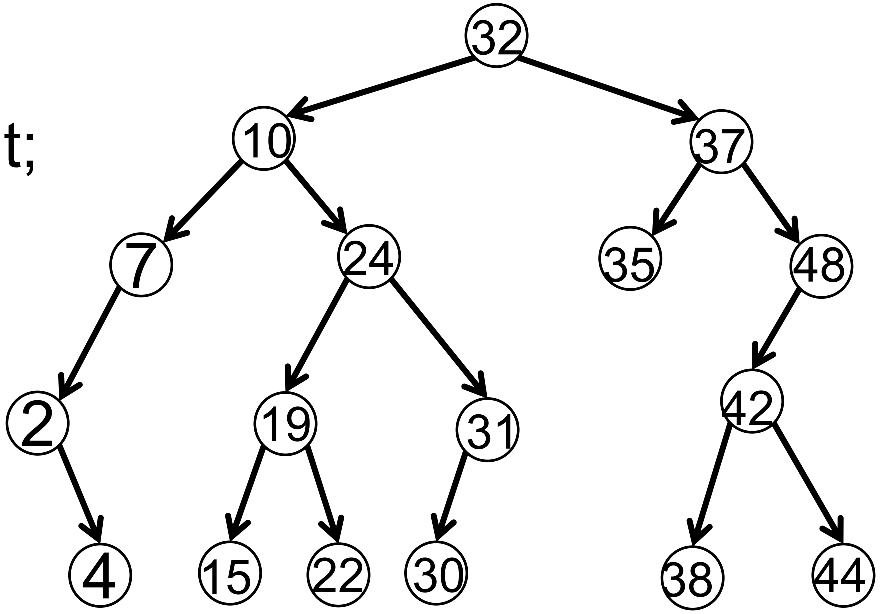
Exercise: how would you print the ordered sequence represented by this tree?

Searching a binary search tree

Input: a reference to the root node of the tree; a target to search for
Goal: if the target we are searching for is in the tree, return the node that has it as its element; else return None

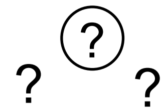
- all left descendants of a node have values less than the node's value
- all right descendants of a node have values greater than the node's value

From now on, we only have a BST; we do not have any list



Searching a binary search tree

Is 27 in this tree



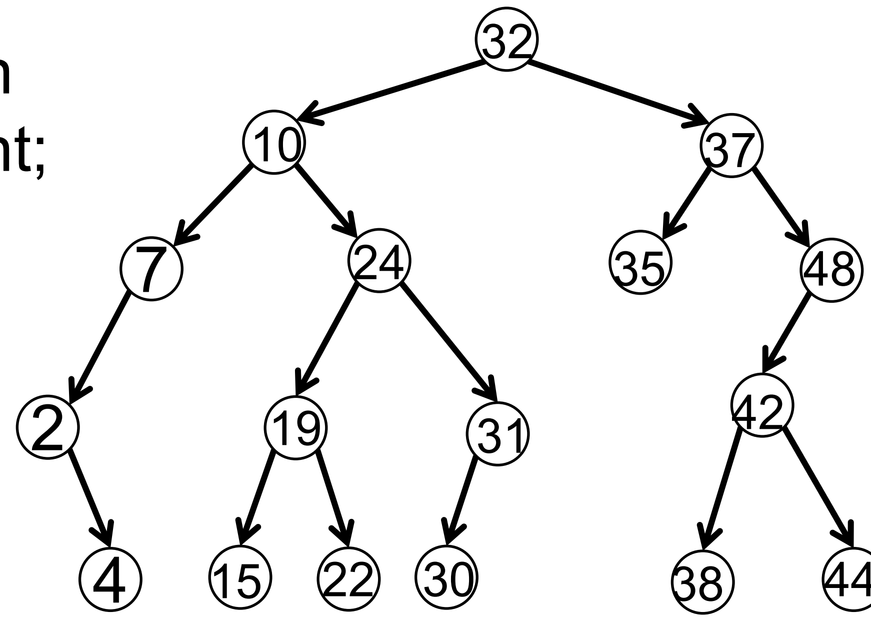
Searching a binary search tree

Input: a reference to the root node of the tree; a target to search for
Goal: if the target we are searching for is in the tree, return the node that has it as its element; else return None

- all left descendants of a node have values less than the node's value
- all right descendants of a node have values greater than the node's value

```
search(node, item):
   if node == None
      return None
   if node's element > item
      return search(leftchild, item)
   else if node's element < item
      return search(rightchild, item)
   else return node
```

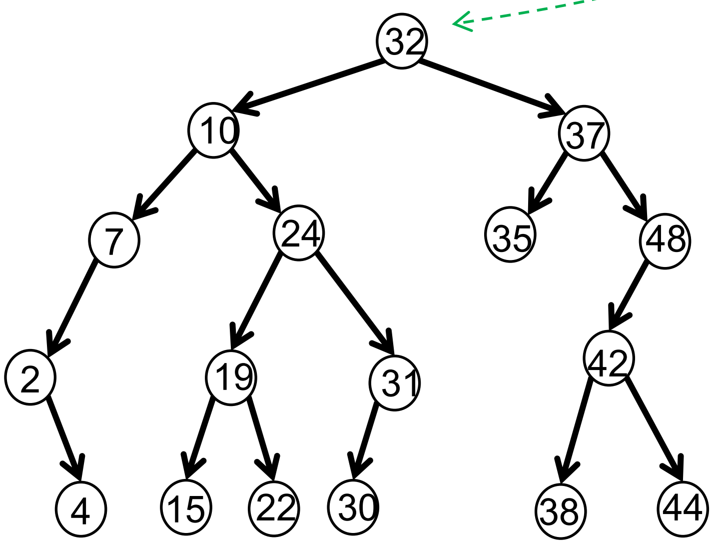
From now on, we only have a BST; we do not have any list



If h is the height of the root, then this is O(h)

Adding a node to a BST

We know the value we want to add: x
We know the location of this node:



Requirement: maintain the order property

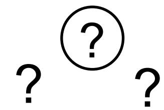
Aim: minimise the work

When asked to add, if we allow only one copy of each element, then

1) first we need to check that the element is not already there
2) then we need to add it

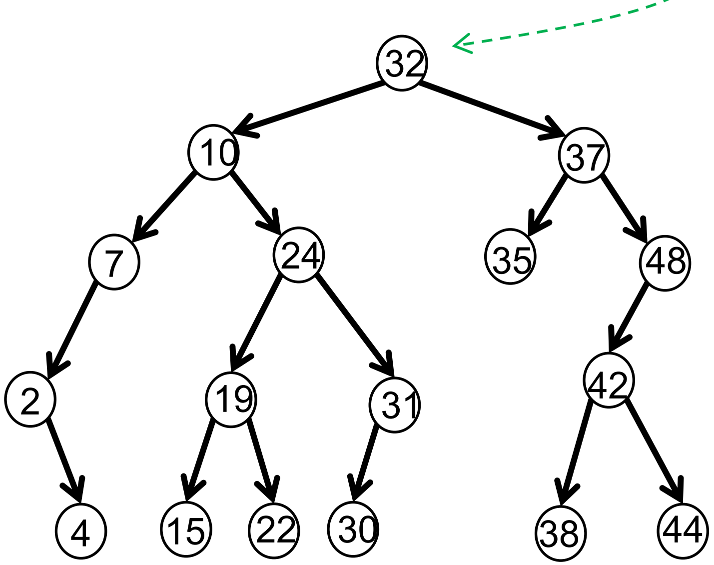
Searching a binary search tree

Add 27 to this tree



Adding a node to a BST (ii)

We know the value we want to add: x
We know the location of this node:



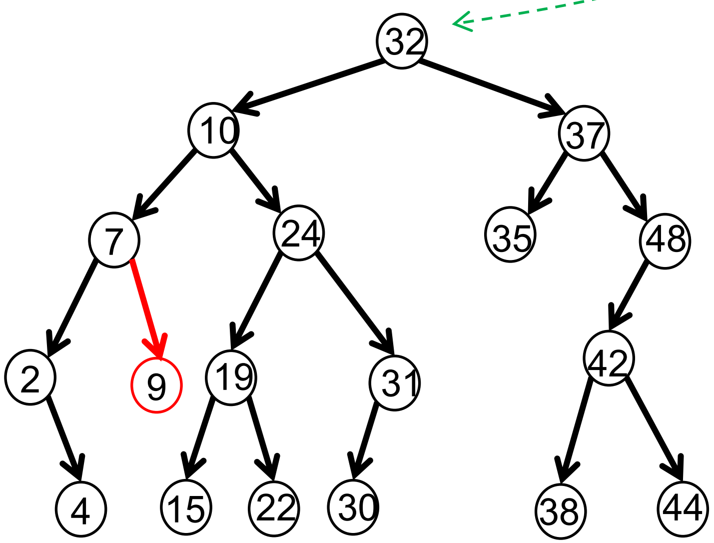
If a node is not in the tree, then the search ends when we reach a null value.

Solution: add the element there, and don't change anything else in the tree

Class exercise: add 9
Class exercise: add 43

Adding a node to a BST (ii)

We know the value we want to add: x
We know the location of this node:



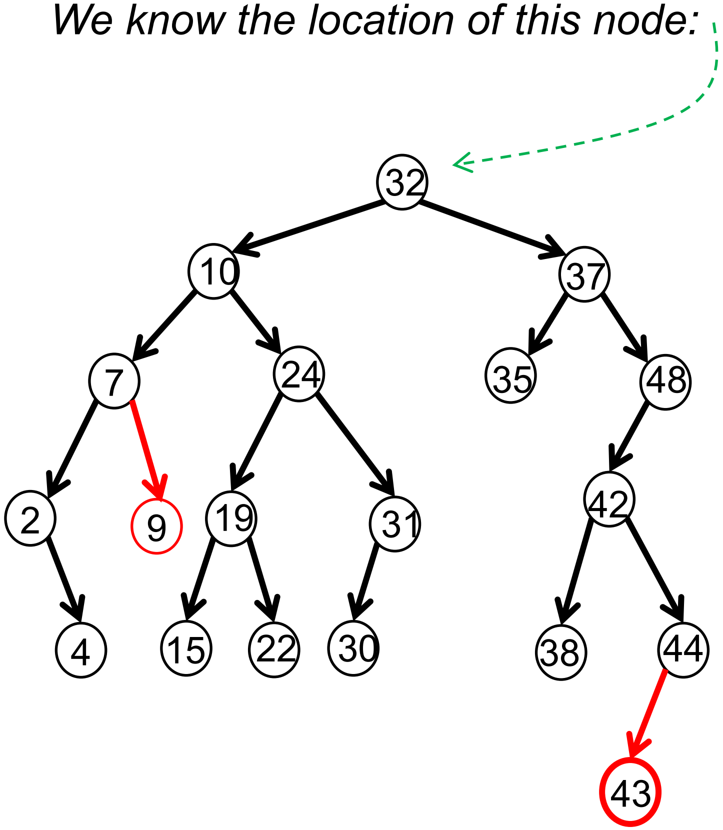
If a node is not in the tree, then the search ends when we reach a null value.

Solution: add the element there, and don't change anything else in the tree

Class exercise: add 9
Class exercise: add 43

Adding a node to a BST (ii)

We know the value we want to add: x
We know the location of this node:



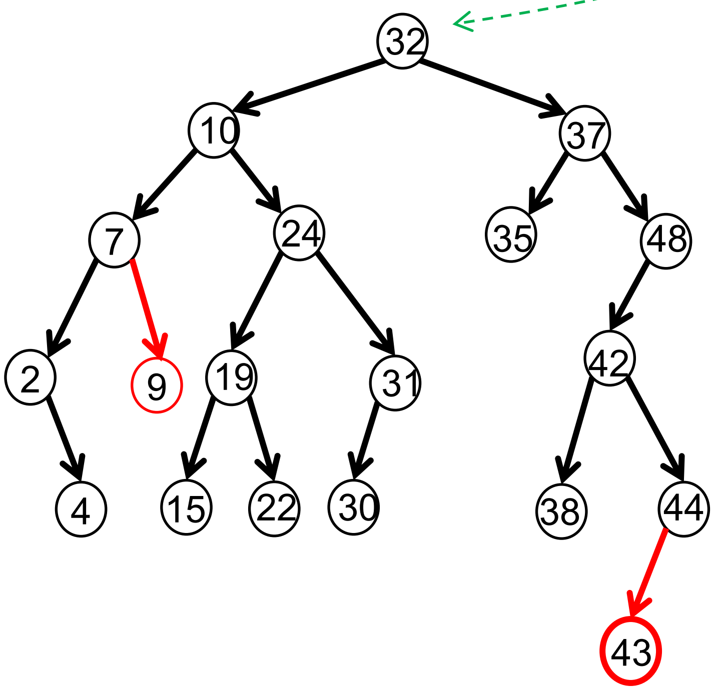
If a node is not in the tree, then the search ends when we reach a null value.

Solution: add the element there, and don't change anything else in the tree

Class exercise: add 9
Class exercise: add 43

Adding a node to a BST (iii)

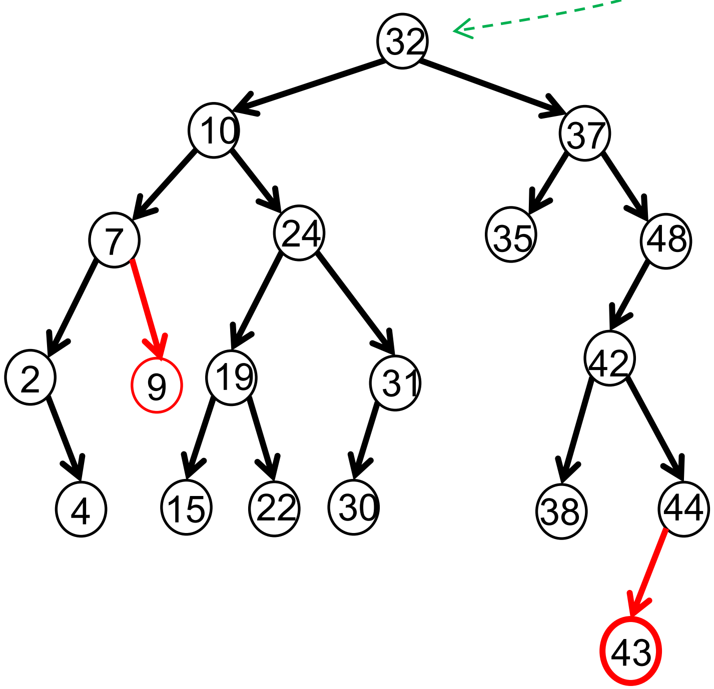
We know the value we want to add: x
We know the location of this node:




```
add(node, x):
   if x < current element
      if no left child
         add x as new left child
      else
         add(node.left, x)
   else if x > current element
      if no right child
         add x as new right child
      else
         add(node.right, x)
   else
      #do nothing - already there
```

Adding a node to a BST (iv)

We know the value we want to add: x
We know the location of this node:



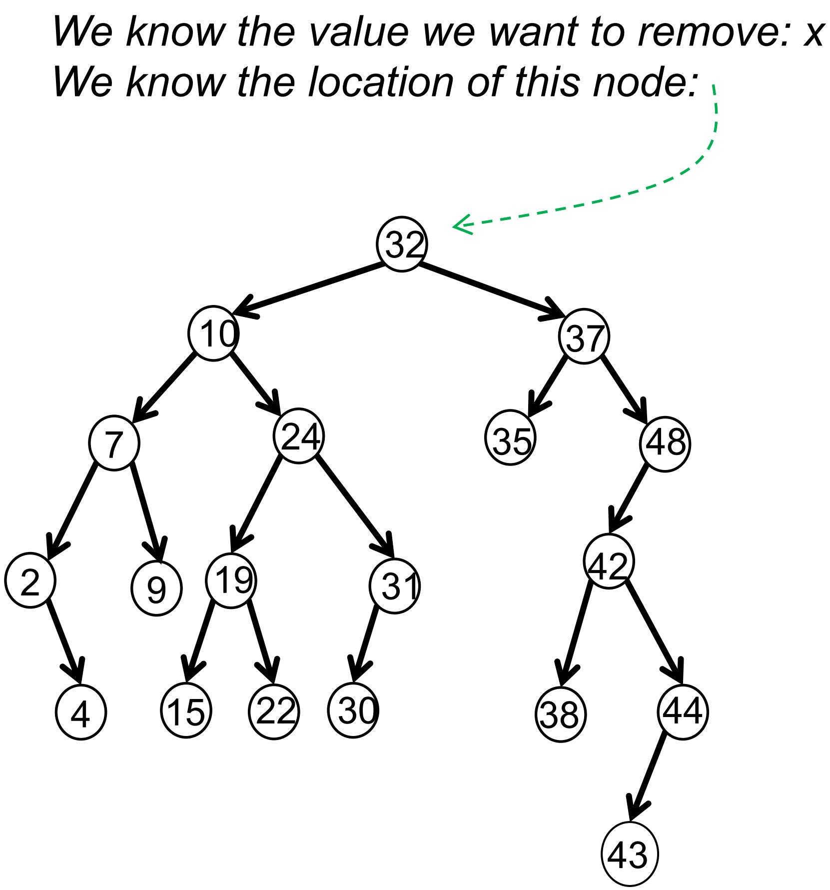
For a given BST, for each possible addition, if we are restricted to simply adding the new element in an empty place, then there is only one possible location in the tree.

Class exercise: what are the empty places in the tree on the left, and what values can they take?

Adding a node to a BST: complexity

To find the node or its natural position is O(height of tree)
Adding the node at that position is a constant number of operations.

Removing a node from a BST



Requirement: maintain the BST property

Aim: minimise the work

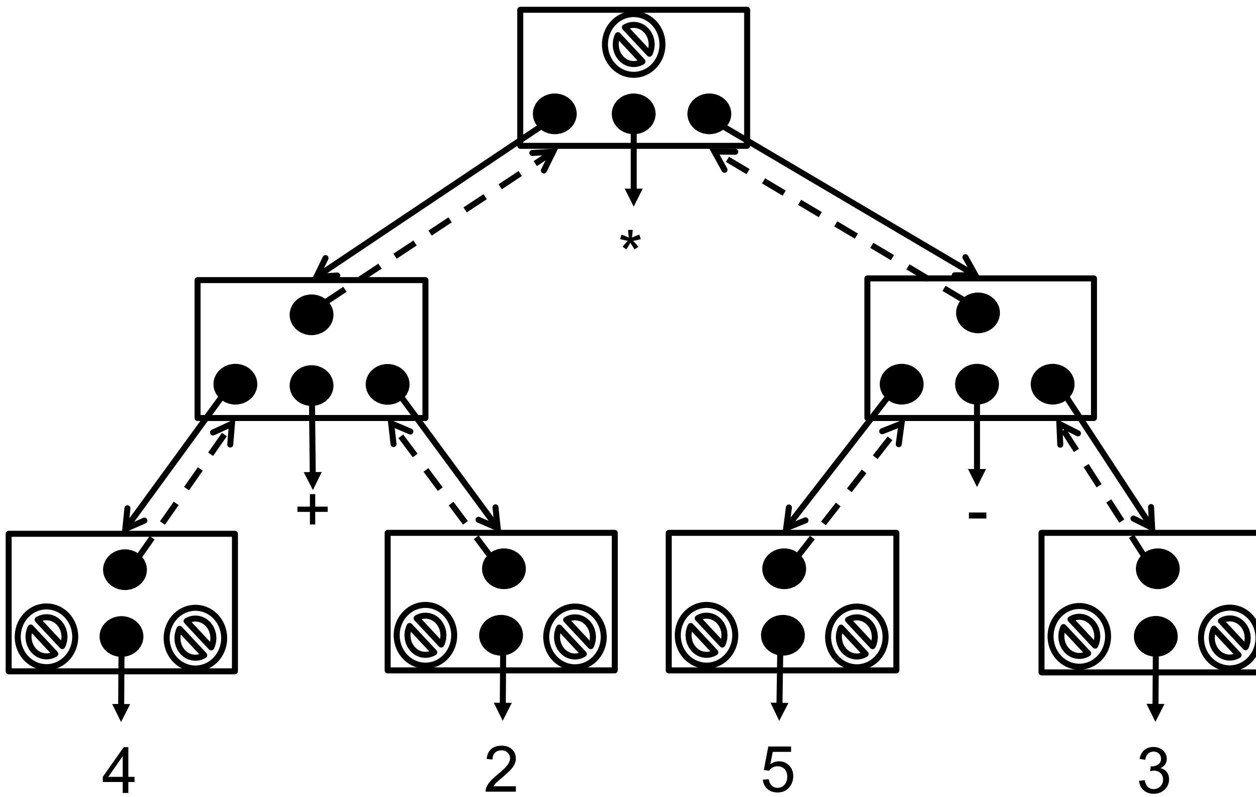
Start by finding the node in the tree (and if it is not there, stop)

Handle the removal by breaking it down into different cases

Change the representation so that each node also points to its parent.

BinaryTreeNode

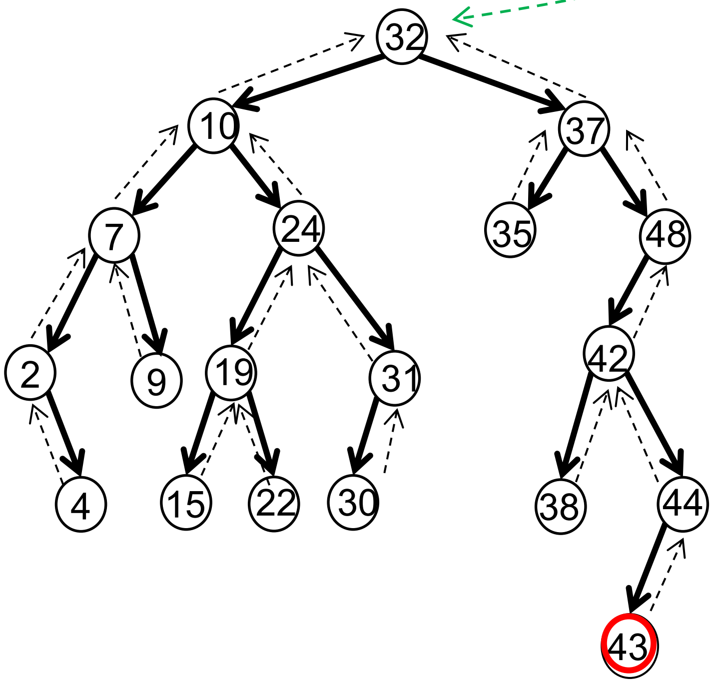
BinaryTreeNode2
element leftchild rightchild parent



This is now a *doubly-linked* tree ...

Removing a leaf node from a BST

We know the value we want to remove: x
We know the location of this node:

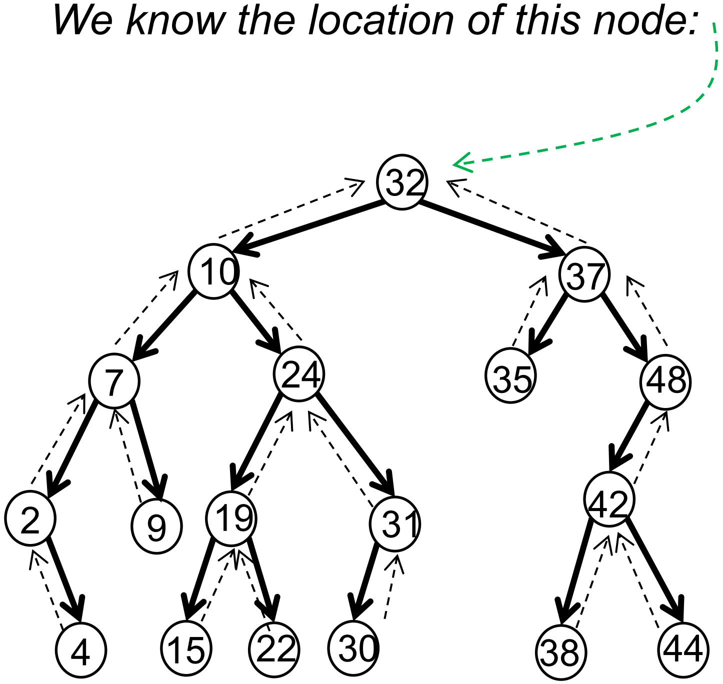


Example: remove 43

We have found the node
we want to remove:
and it is a *<u>leaf</u>*

Removing a leaf node from a BST

We know the value we want to remove: x
We know the location of this node:



Example: remove 43

Easy:
- update the parent's appropriate child reference
- set the node's parent to None
- remember the element
- set the node's element to None
- return the element

Removing a root & leaf node from a BST

We know the value we want to remove: x
We know the location of this node:

But be careful if the leaf node is also the root node ...

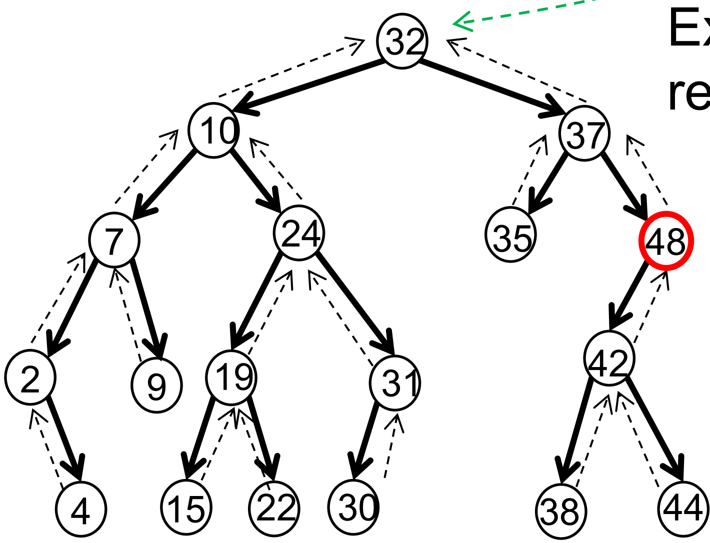
The root node has no parent.

Easy:

~~update the parent's child reference~~
set the node's parent to None
remember the element
set the node's element to None
return the element

Removing a semi-leaf node from a BST

We know the value we want to remove: x
We know the location of this node:



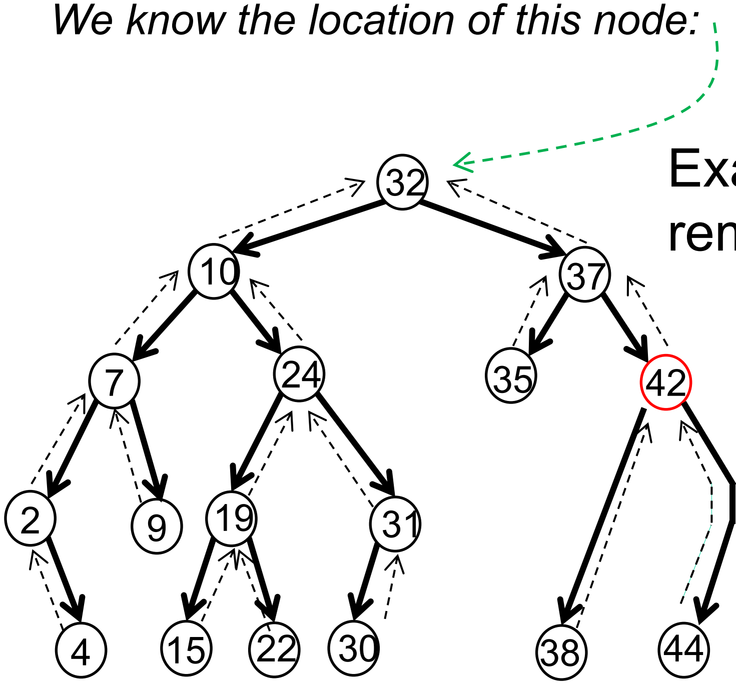
Example:
remove 48

A *semileaf* is a node with only one child

We have found the node we want to remove: and it is a semileaf

Removing a semi-leaf node from a BST

We know the value we want to remove: x
We know the location of this node:



Example:
remove 48

A *semileaf* is a node with only one child

Remember the semileaf's item
Copy the semileaf's child's item into the semileaf.
Remember the semileaf child node
Rearrange the references to bypass the child node
Wipe out the child
Return the original element

Order properties are maintained
- semileaf and all descendants were larger than semileaf's parent, so all descendants are still larger
- no other relationship has changed

Next Lecture

Full version of deleting from binary search trees