CSC263 Tutorial #3

BSTs, but with duplicates

January 27, 2023

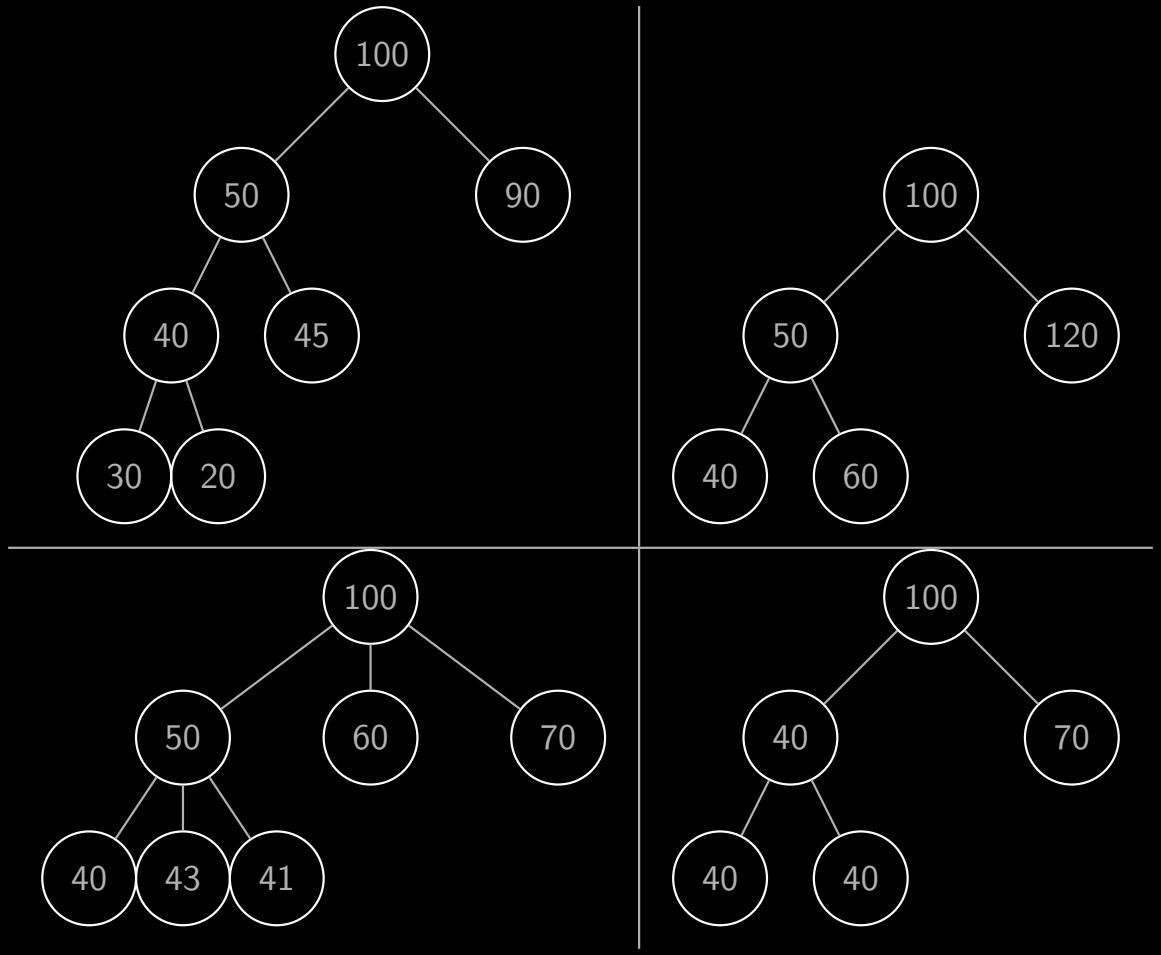
Things that will be covered in this tutorial

Things that will be covered in this tutorial

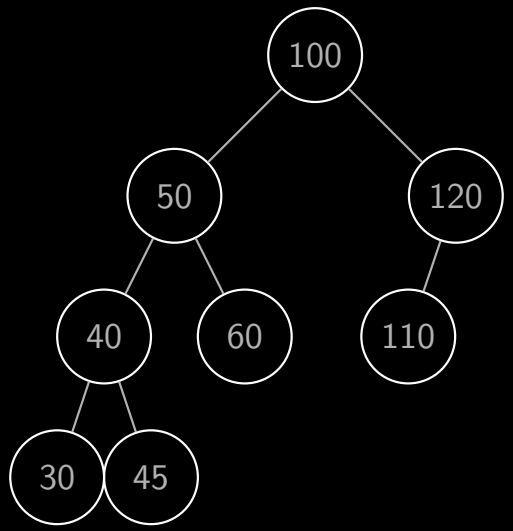
- ⋆ How do BST operations work?
- ⋆ What changes do I need to make to a BST to allow duplicate elements?
- ⋆ How bad does it get if there are many duplicate elements in a BST?

Binary Search Trees

Question: Which of the following is a binary search tree?



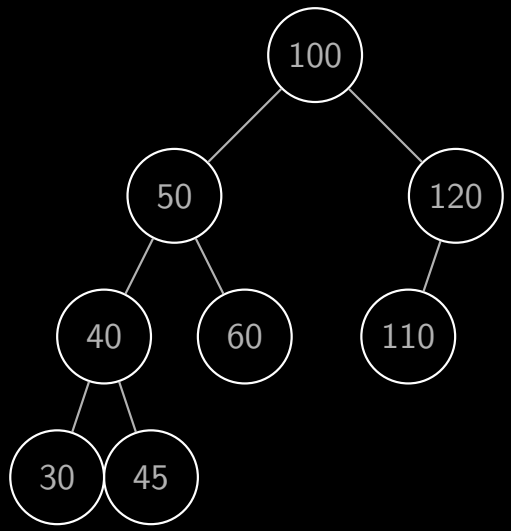
Binary Search Trees



Task: Perform the following operations on the above BST, in order:

- ⋆ Successor(100).
- ⋆ Successor(45).
- ⋆ Successor(120).

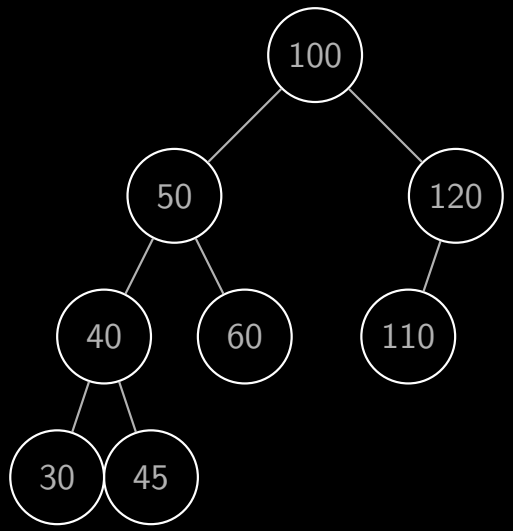
Binary Search Trees



Task: Perform the following operations on the above BST, in order:

- ⋆ Insert(130).
- ⋆ Insert(105).
- ⋆ Delete(60).
- ⋆ Delete(50).
- ⋆ Delete(100).

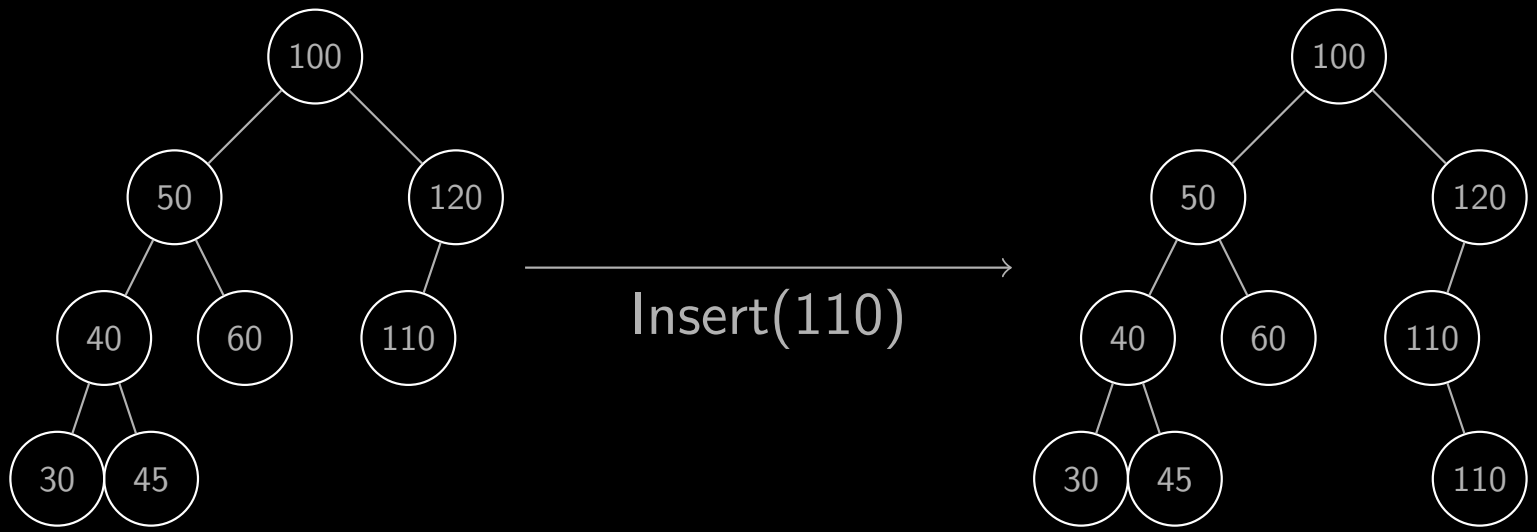
Binary Search Trees



Task: Perform the following operations on the above BST, in order:

- ⋆ Insert(130).
- ⋆ Insert(105).
- ⋆ Delete(60).
- ⋆ Delete(50).
- ⋆ Delete(100).

BST Insert... but with duplicates



Note: For now, equal keys are treated the same as larger keys.

BST Insert... but with duplicates

```
Insert(D, key, value):
    if D is empty:
        D.root.key = key
        D.root.value = value
    else if D.root.key >= key:
        Insert(D.left, key, value)
    else:
        Insert(D.right, key, value)
```

BST Insert... but with duplicates

```
Insert(D, key, value):
    if D is empty:
        D.root.key = key
        D.root.value = value
    else if D.root.key >= key:
        Insert(D.left, key, value)
    else:
        Insert(D.right, key, value)
```

Tutorial Question: Modify the above algorithm to handle inserting duplicate values (i.e. values that already exist in the BST).
Hint: You only need to modify one line!

BST Insert... but with duplicates

```
Insert(D, key, value):
    if D is empty:
        D.root.key = key
        D.root.value = value
    else if D.root.key > key:
        Insert(D.left, key, value)
    else:
        Insert(D.right, key, value)
```

BST Insert... but with duplicates

```
Insert(D, key, value):
    if D is empty:
        D.root.key = key
        D.root.value = value
    else if D.root.key > key:
        Insert(D.left, key, value)
    else:
        Insert(D.right, key, value)
```

Note that in Insert(), everything but the recursive call is constant-time.

BST Insert... but with duplicates

```
Insert(D, key, value):
    if D is empty:
        D.root.key = key
        D.root.value = value
    else if D.root.key > key:
        Insert(D.left, key, value)
    else:
        Insert(D.right, key, value)
```

Note that in Insert(), everything but the recursive call is constant-time.
Question:

BST Insert... but with duplicates

```
Insert(D, key, value):
    if D is empty:
        D.root.key = key
        D.root.value = value
    else if D.root.key > key:
        Insert(D.left, key, value)
    else:
        Insert(D.right, key, value)
```

Note that in Insert(), everything but the recursive call is constant-time.

Question:

- ⋆ Consider inserting 4 identical elements $1, 1, 1, 1$ into an empty BST. How many Insert() calls (including recursive calls) are needed?

BST Insert... but with duplicates

```
Insert(D, key, value):
    if D is empty:
        D.root.key = key
        D.root.value = value
    else if D.root.key > key:
        Insert(D.left, key, value)
    else:
        Insert(D.right, key, value)
```

Note that in Insert(), everything but the recursive call is constant-time.

Question:

- ⋆ Consider inserting 4 identical elements $1, 1, 1, 1$ into an empty BST. How many Insert() calls (including recursive calls) are needed?
- ⋆ **Tutorial Question**: Consider inserting n identical elements $1, 1, \ldots, 1$ into an empty BST. How many Insert() calls (including recursive calls) are needed?

Improving BST insert

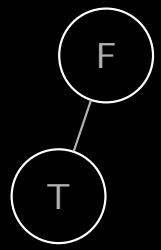
Each node will now have a `goLeft` boolean attribute (default True).

Improving BST insert

Each node will now have a `goLeft` boolean attribute (default True).

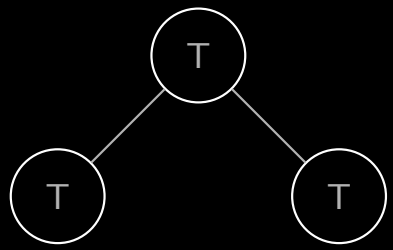
Improving BST insert

Each node will now have a `goLeft` boolean attribute (default True).



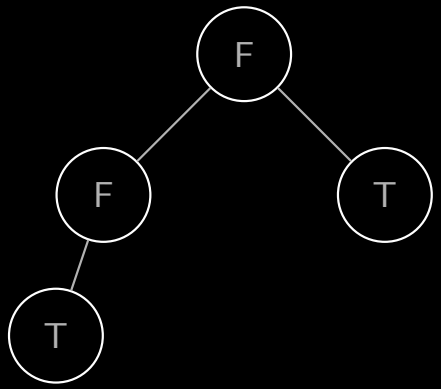
Improving BST insert

Each node will now have a `goLeft` boolean attribute (default True).



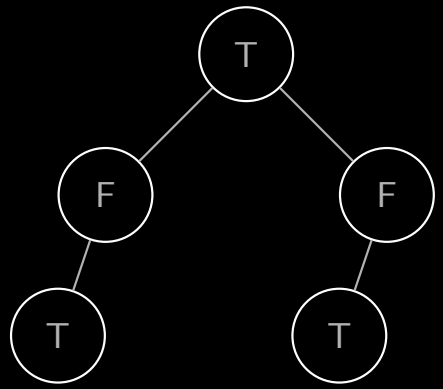
Improving BST insert

Each node will now have a `goLeft` boolean attribute (default True).



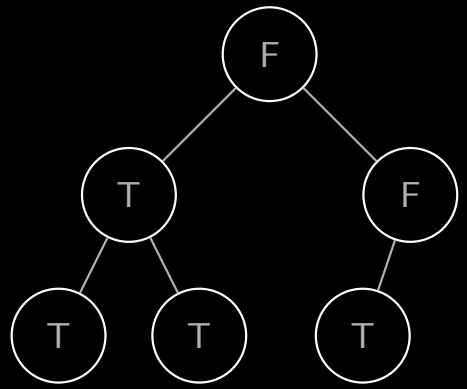
Improving BST insert

Each node will now have a `goLeft` boolean attribute (default True).



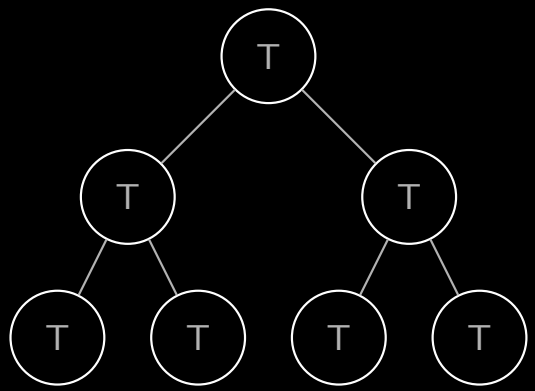
Improving BST insert

Each node will now have a `goLeft` boolean attribute (default True).



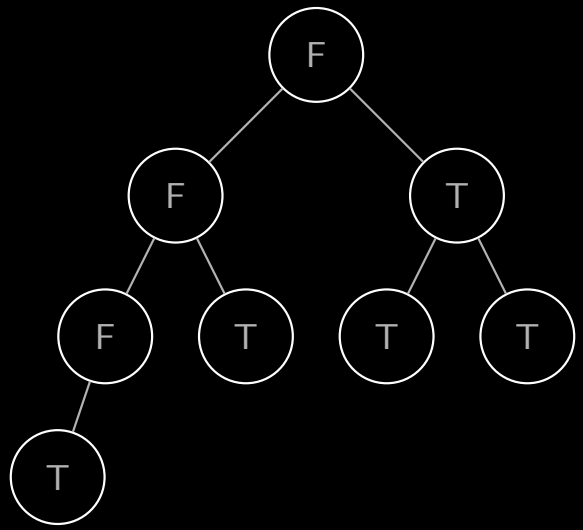
Improving BST insert

Each node will now have a `goLeft` boolean attribute (default True).



Improving BST insert

Each node will now have a `goLeft` boolean attribute (default True).



Improving BST insert

```
Insert(D, key, value):
    if D is empty:
        D.root.key = key
        D.root.value = value
    else if D.root.key > key:
        Insert(D.left, key, value)
    else:
        Insert(D.right, key, value)
```

Tutorial Question: Modify the BST insert algorithm to handle `goLeft`.

Improving BST insert

Consider inserting n identical elements into an empty BST with `goLeft` implemented.

Improving BST insert

Consider inserting n identical elements into an empty BST with `goLeft` implemented.

Question: Approximately how long does the ith insertion take?

Improving BST insert

Consider inserting n identical elements into an empty BST with `goLeft` implemented.

Question: Approximately how long does the ith insertion take?
Answer: $\log_2(i)$.

Improving BST insert

Consider inserting n identical elements into an empty BST with `goLeft` implemented.

Question: Approximately how long does the ith insertion take?
Answer: $\log_2(i)$.

Tutorial Question: Approximately how long does it take to insert all n identical elements? Can you bound it with a Big-Θ expression?
Hint:

- $\log(x) < \log(y)$ whenever $x < y$.
- $\log(\frac{x}{y}) < \log(x) - \log(y)$.

Improving BST insert, part 2

Instead of `goLeft`, we will randomly choose to insert duplicates into the left or right subtree.

Improving BST insert, part 2

Instead of `goLeft`, we will randomly choose to insert duplicates into the left or right subtree.

```
Insert(D, key, value):
    if D is empty:
        D.root.key = key
        D.root.value = value
    else if D.root.key > key:
        Insert(D.left, key, value)
    else:
        Insert(D.right, key, value)
```

Tutorial Question: Modify the above to randomly insert identical keys, with a `random(0, 1)` call. What's the worst case of inserting n identical elements here?
Bonus: What's the *average* cost of inserting n identical elements?

Improving BST insert, part 3

Open your quiz, and go to question 3!