Number guessing game

I'm thinking of a number between 1 and n
You are trying to guess the answer
For each guess, I'll tell you “correct”, “higher” or “lower”
Describe an algorithm that minimizes the number of guesses

Binary Search Trees

BST – A binary tree where a parent’s value is greater than all values in the left subtree and less than or equal to all the values in the right subtree

\[ \text{leftTree}(i) < i \leq \text{rightTree}(i) \]

and the left and right children are also binary search trees

Why not?

\[ \text{leftTree}(i) \leq i \leq \text{rightTree}(i) \]

Ambiguous about where elements that are equal would reside
Example

Can be implemented with references or an array

What else can we conclude?

$leftTree(i) < i \leq rightTree(i)$

The smallest element is the leftmost element

The largest element is the rightmost element

Another example: the solo tree

Another example: the twig
Operations
- Search(T, k) – Does value k exist in tree T
- Insert(T, k) – Insert value k into tree T
- Delete(T, x) – Delete node x from tree T
- Minimum(T) – What is the smallest value in the tree?
- Maximum(T) – What is the largest value in the tree?
- Successor(T, x) – What is the next element in sorted order after x
- Predecessor(T, x) – What is the previous element in sorted order of x
- Median(T) – return the median of the values in tree T

Search
How do we find an element?

BSTSearch(x, k)
1. if x = null or k = x
2. return x
3. elseif k < x
4. return BSTSearch(LEFT(x), k)
5. else
6. return BSTSearch(RIGHT(x), k)

Finding an element
Search(T, 9)

Finding an element
Search(T, 9)
Finding an element

Search(T, 9)

Finding an element

Search(T, 9)

Finding an element

Search(T, 9)

Finding an element

Search(T, 9)
Is BSTSearch correct?

```
BSTSearch(x, k)
1  if x = null or k = x
2     return x
3  else if k < x
4     return BSTSearch(LEFT(x), k)
5  else
6     return BSTSearch(RIGHT(x), k)
```

leftTree(i) < i ≤ rightTree(i)

Running time of BSTSearch

- Worst case?
  - $\Theta$ (height of the tree)

- Average case?
  - $O$ (height of the tree)

- Best case?
  - $O(1)$

Height of the tree

- Worst case height?
  - n-1
  - “the twig”

- Best case height?
  - $\lceil \log_2 n \rceil$
  - complete (or near complete) binary tree

- Average case height?
  - Depends on two things:
    - the data
    - how we build the tree!

Insertion

```
BSTInsert(T, x)
1  if Root(T) = null
2     Root(T) = x
3  else
4    y = Root(T)
5    while y ≠ null
6      prev = y
7      if x < y
8        y = LEFT(y)
9      else
10     y = RIGHT(y)
11    Parent(x) = prev
12    if x < prev
13      Left(prev) = x
14    else
15      Right(prev) = x
```
Insertion

BSTINSERT(T, z)
1 if Root(T) = null
2 Root(T) ← z
else
4 y ← Root(T)
5 while y ≠ null
6 prev ← y
7 if x < y
8 y ← LEFT(y)
9 else
10 y ← RIGHT(y)
11 Parent(y) ← prev
12 if x < prev
13 LEFT(prev) ← x
14 else
15 RIGHT(prev) ← x

Similar to search

InertBSTSearch(x, y)
1 while x ≠ null and y ≠ x
2 if x < y
3 y ← LEFT(y)
4 else
5 y ← RIGHT(y)
6 return y

Find the correct
location in the tree

Insertion

BSTINSERT(T, z)
1 if Root(T) = null
2 Root(T) ← z
else
4 y ← Root(T)
5 while y ≠ null
6 prev ← y
7 if x < y
8 y ← LEFT(y)
9 else
10 y ← RIGHT(y)
11 Parent(y) ← prev
12 if x < prev
13 LEFT(prev) ← x
14 else
15 RIGHT(prev) ← x

Insertion

BSTINSERT(T, z)
1 if Root(T) = null
2 Root(T) ← z
else
4 y ← Root(T)
5 while y ≠ null
6 prev ← y
7 if x < y
8 y ← LEFT(y)
9 else
10 y ← RIGHT(y)
11 Parent(y) ← prev
12 if x < prev
13 LEFT(prev) ← x
14 else
15 RIGHT(prev) ← x

Insertion

BSTINSERT(T, z)
1 if Root(T) = null
2 Root(T) ← z
else
4 y ← Root(T)
5 while y ≠ null
6 prev ← y
7 if x < y
8 y ← LEFT(y)
9 else
10 y ← RIGHT(y)
11 Parent(y) ← prev
12 if x < prev
13 LEFT(prev) ← x
14 else
15 RIGHT(prev) ← x
Correctness?

BSTInsert(T, x)
1 If Root(T) = null
2 Root(T) ← x
3 else
4 y ← Root(T)
5 while y ≠ null
6 prev ← y
7 if x < y
8 y ← left(y)
9 else
10 y ← right(y)
11 Parent(z) ← prev
12 if z < prev
13 Left(prev) ← x
14 else
15 Right(prev) ← x

Correctness

BSTInsert(T, x)
1 If Root(T) = null
2 Root(T) ← x
3 else
4 y ← Root(T)
5 while y ≠ null
6 prev ← y
7 if x < y
8 y ← left(y)
9 else
10 y ← right(y)
11 Parent(z) ← prev
12 if z < prev
13 Left(prev) ← x
14 else
15 Right(prev) ← x

What happens if it is a duplicate?

Inserting duplicate

Insert(T, 14)

leftTree(i) < i ≤ rightTree(i)

Inserting duplicate

Insert(T, 14)

leftTree(i) < i ≤ rightTree(i)
Running time

```
BSTINSERT(T, x)
1 if Root(T) = null
2 Root(T) ← x
3 else
4 y ← Root(T)
5 while y ≠ null
6 prev ← y
7 if x < y
8 y ← LEFT(y)
9 else
10 y ← RIGHT(y)
11 Parent(y) ← prev
12 if x < prev
13 LEFT(prev) ← x
14 else
15 RIGHT(prev) ← x
```

Running time

```
BSTINSERT(T, x)
1 if Root(T) = null
2 Root(T) ← x
3 else
4 y ← Root(T)
5 while y ≠ null
6 prev ← y
7 if x < y
8 y ← LEFT(y)
9 else
10 y ← RIGHT(y)
11 Parent(y) ← prev
12 if x < prev
13 LEFT(prev) ← x
14 else
15 RIGHT(prev) ← x
```

Running time

Insert(T, 15)

Height of the tree

Worst case: “the twig” – When will this happen?
Height of the tree

Best case: “complete” – When will this happen?

Average case for random data?

Randomly inserting data into a BST generates a tree on average that is $O(\log n)$

Visiting all nodes

In sorted order
Visiting all nodes

In sorted order

5, 8

12

8

5, 9, 20

14

Visiting all nodes

In sorted order

5, 8, 9

12

8

5, 9

14

20

Visiting all nodes

In sorted order

5, 8, 9, 12

12

8

5, 9

14

20

Visiting all nodes

What's happening?

5, 8, 9, 12

12

8

5, 9

14

20
Visiting all nodes

In sorted order 5, 8, 9, 12, 14

Visiting all nodes

In sorted order 5, 8, 9, 12, 14, 20

Visiting all nodes in order

\[
\text{INORDER\textsc{TreeWalk}(z)}
\]

1. \text{if } z \neq \text{null}
2. \text{INORDER\textsc{TreeWalk}(LEFT(z))}
3. \text{print } x
4. \text{INORDER\textsc{TreeWalk}(RIGHT(z))}

Visiting all nodes in order

\[
\text{INORDER\textsc{TreeWalk}(z)}
\]

1. \text{if } z \neq \text{null}
2. \text{INORDER\textsc{TreeWalk}(LEFT(z))}
3. \text{print } x
4. \text{INORDER\textsc{TreeWalk}(RIGHT(z))}

any operation
Is it correct?

\[
\text{INORDERTreeWalk}(x) \\
1 \quad \text{if } x \neq \text{null} \\
2 \quad \text{INORDERTreeWalk(LEFT}(x)\text{)} \\
3 \quad \text{print } x \\
4 \quad \text{INORDERTreeWalk(RIGHT}(x)\text{)}
\]

Does it print out all of the nodes in sorted order?

\[
leftTree(i) < i \leq rightTree(i)
\]

What about?

\[
\text{TreeWalk}(x) \\
1 \quad \text{if } x \neq \text{null} \\
2 \quad \text{print } x \\
3 \quad \text{TreeWalk(LEFT}(x)\text{)} \\
4 \quad \text{TreeWalk(RIGHT}(x)\text{)}
\]

Preorder traversal

12, 8, 5, 9, 14, 20

How is this useful?

Tree copying: insert in to new tree in preorder

prefix notation: \((2+3)^4 \cdot 4 \Rightarrow \ast + 2 3 4\)
Postorder traversal

5, 9, 8, 20, 14, 12

How is this useful?

postfix notation: \((2 + 3) \times 4\)
\[ \Rightarrow 4 \ 3 \ 2 \ + \ * \]

Min/Max

Running time of min/max?

\[ O(\text{height of the tree}) \]

Successor and predecessor

Predecessor(12)? 9
Successor and predecessor

Predecessor in general?
- largest node of all those smaller than this node
- rightmost element of the left subtree

Successor

Successor in general?
- smallest node of all those larger than this node
- leftmost element of the right subtree

What if the node doesn't have a right subtree?
- smallest node of all those larger than this node
- leftmost element of the right subtree
What if the node doesn’t have a right subtree?

The successor is the node that has x as a predecessor.

Successor

successor is the node that has x as a predecessor

Successor

successor is the node that has x as a predecessor

Successor

successor is the node that has x as a predecessor
Successor

successor is the node that has x as a predecessor

keep going up until we're no longer a right child

Successor

Successor

Successor

Successor

Successor

Successor

Successor

Successor

successor is the node that has x as a predecessor

if we have a right subtree, return the smallest of the right subtree

find the node that x is the predecessor of

keep going up until we're no longer a right child
Successor running time

\[ O(\text{height of the tree}) \]

**Successor**

```python
SUCCESSION(z)
1. if RIGHT(z) ≠ null
2. \quad return BST_MIN(RIGHT(z))
3. else
4. \quad y ← PARENT(z)
5. \quad while y ≠ null and z ← RIGHT(y)
6. \quad x ← y
7. return y
```

Deletion

Three cases!

Deletion: case 1

No children
Just delete the node
Deletion: case 2

One child

Splice out the node

Delete: case 2

One child

Splice out the node

Deletion: case 3

Two children

Replace x with its successor

Deletion: case 3

Two children

Replace x with its successor
Deletion: case 3

Two children

Will we always have a successor?

Why successor?
- Larger than the left subtree
- Less than or equal to right subtree

Height of the tree

Most of the operations take time $O(\text{height of the tree})$

We said trees built from random data have height $O(\log n)$, which is asymptotically tight

Two problems:
- We can’t always insure random data
- What happens when we delete nodes and insert others after building a tree?

Balanced trees

Make sure that the trees remain balanced!
- Red-black trees
- AVL trees
- 2-3-4 trees
- ...

B-trees

Red-black trees: BST (plus some)

1. every node is either red or black
2. root is black
3. leaves (NIL) are black
4. if a node is red, both children are black
5. for every node, all paths from the node to descendant leaves contain the same number of black nodes.

Red-black trees: BST (plus some)

1. every node is either red or black
2. root is black
3. leaves (NIL) are black
4. if a node is red, both children are black
5. for every node, all paths from the node to descendant leaves contain the same number of black nodes.

$h(x)$: height of node $x$: number of edges in longest path from $x$ to a leaf

82

Red-black trees: BST (plus some)

What is the height of the root node?

83

Red-black trees: BST (plus some)

$h(x)$: height of node $x$: number of edges in longest path from $x$ to a leaf

4

84

Red-black trees: BST (plus some)

1. every node is either red or black
2. root is black
3. leaves (NIL) are black
4. if a node is red, both children are black
5. for every node, all paths from the node to descendant leaves contain the same number of black nodes.

$bh(x)$: black height of node $x$: number of black nodes on a path from $x$ to leaf (not including $x$)

Why don't we say "path with the most..."?

85
Red-black trees: BST (plus some)

1. every node is either red or black
2. root is black
3. leaves (NIL) are black
4. if a node is red, both children are black
5. for every node, all paths from the node to descendant leaves contain the same number of black nodes.

Why don’t we say “path with the most…”?

Claim 1: For every node \( x \), \( bh(x) \geq h(x)/2 \)

Proof?
Bounding the height

Claim 1: For every node $x$, $bh(x) \geq h(x)/2$

1. Every node is either red or black
2. Root is black
3. Leaves (NIL) are black
4. If a node is red, both children are black
5. For every node, all paths from the node to descendant leaves contain the same number of black nodes.

$h(x)$: height of node $x$; number of edges in longest path from $x$ to a leaf.

$bh(x)$: black height of node $x$; number of black nodes on a path from $x$ to a leaf (not including $x$).

Minimum black nodes on path: $bh(x) \geq \frac{h(x)}{2}$. $bh$ does not include $x$, i.e., the root in this case.

Bounding the height

Claim 2: The subtree rooted at any node $x$ contains at least $2^{bh(x)} - 1$ internal (non-leaf) nodes.

Proof?

Base case:
Claim 2: The subtree rooted at any node $x$ contains at least $2^{bh(x)} - 1$ internal (non-leaf) nodes

**Base case: leaf ($h(x) = 0$)**
- $bh(x) = 0$
- $2^0 - 1 = 0$

**Inductive case:**
- $h(x) > 0$
- IH: $2^{bh(y)} - 1$ for all $y$ that are subtrees of $x$

$x$ is red: $bh(child(x)) = bh(x) - 1$
$x$ is black: $bh(child(x)) = bh(x)$ or $bh(x) - 1$

What is $bh(child(x))$ wrt $bh(x)$?

$bh(x)$: black height of node $x$; number of black nodes on a path from $x$ to leaf (not including $x$)

$bh(child(x))$: black height of child node of $x$; number of black nodes on a path from child node to leaf (not including child node and $x$)
Claim 2: The subtree rooted at any node $x$ contains at least $2^{bh(x)} - 1$ internal (non-leaf) nodes

Inductive case: $h(x) > 0$

IH: $2^{bh(y)} - 1$ for all $y$ that are subtrees of $x$

$bh(child(x)) \geq bh(x) - 1$

How many internal nodes are in this tree (at least)?

Claim 1: For every node $x$, $bh(x) \leq \frac{h(x)}{2}$

Claim 2: The subtree rooted at any node $x$ contains at least $2^{bh(x)} - 1$ internal (non-leaf) nodes

Inductive case: $h(x) > 0$

IH: $2^{bh(y)} - 1$ for all $y$ that are subtrees of $x$

$bh(child(x)) \geq bh(x) - 1$

$2^{bh(x)-1} - 1 + 2^{bh(x)-1} - 1 + 1 = 2^{bh(x)} - 1$

How does this help us?
Bounding the height

Claim 1: For every node \( x \), \( bh(x) \geq \frac{h(x)}{2} \)

Claim 2: The subtree rooted at any node \( x \) contains at least \( 2^{bh(x)} - 1 \) internal (non-leaf) nodes

\[
\begin{align*}
n &\geq 2^{bh(x)} - 1 \\
n &\geq 2^{h(x)/2} - 1 \\
n + 1 &\geq 2^{h(x)/2} \\
h(x) &\leq 2\log(n + 1)
\end{align*}
\]

What does this mean?

Bounding the height

- every node is either red or black
- root is black
- leaves (NIL) are black
- if a node is red, both children are black
- for every node, all paths from the node to descendant leaves contain the same number of black nodes.

Search
Insert
Delete
Maximum

If we can maintain these properties: height \( O(\log n) \)

Can it be done?

Can we maintain the red-black tree properties without making insertion and deletion more expensive?

A quick example

https://www.youtube.com/watch?v=vDHFF4wWyU

Can we maintain the red-black tree properties without making insertion and deletion more expensive?

https://en.wikipedia.org/wiki/Tree_rotation#/media/File:Tree_rotation.png