Week 6: Monday

EECS 281

MAXIM ALEKSA

maximal.io


Apple Inc. developer.apple.com/wwdc/
WWDC17
Overview
Schedule
Consultations
Scholarships
Attending
More
Technology alone is not enough.
Technology must intersect with the liberal arts
and the humanities, to create new ideas and
experiences that push society forward. This
summer we bring together thousands of brilliant
minds representing many diverse perspectives,
passions, and talents to help us change the world.
Welcome to
WWDC17
San Jose, CA, June 5–9


"Hopefully this will cheer you up after getting back your midterm results" my IA
MICHIGAN
THE UNIVERSITY OF MICHIGAN
1817
CHAT

Equal-Sum Pairs

```
void findEqualSumPairs(const vector<int> &v) {
    unordered_map<int, pair<int, int>> sumsToPairs;
    for (int i = 0; i < v.size(); ++i) {
        for (int j = i + 1; j < v.size(); ++j) {
            int sum = v[i] + v[j];
            if (sumsToPairs.find(sum) == sumsToPairs.end()) {
                sumsToPairs[sum] = make_pair(v[i], v[j]);
            } else {
                pair<int, int> firstPair = sumsToPairs[sum];
                cout << "{" << firstPair.first << ", "
                     << firstPair.second << " } and {"
                     << v[i] << ", " << v[j] << "} << endl;
                return;
            }
        }
    }
    cout << "No pairs found" << endl;
}
```

Agenda

Trees

Binary Search Trees

AVL Trees

Tries

Trees

Trees

Tree is an ADT

Hierarchical tree structure

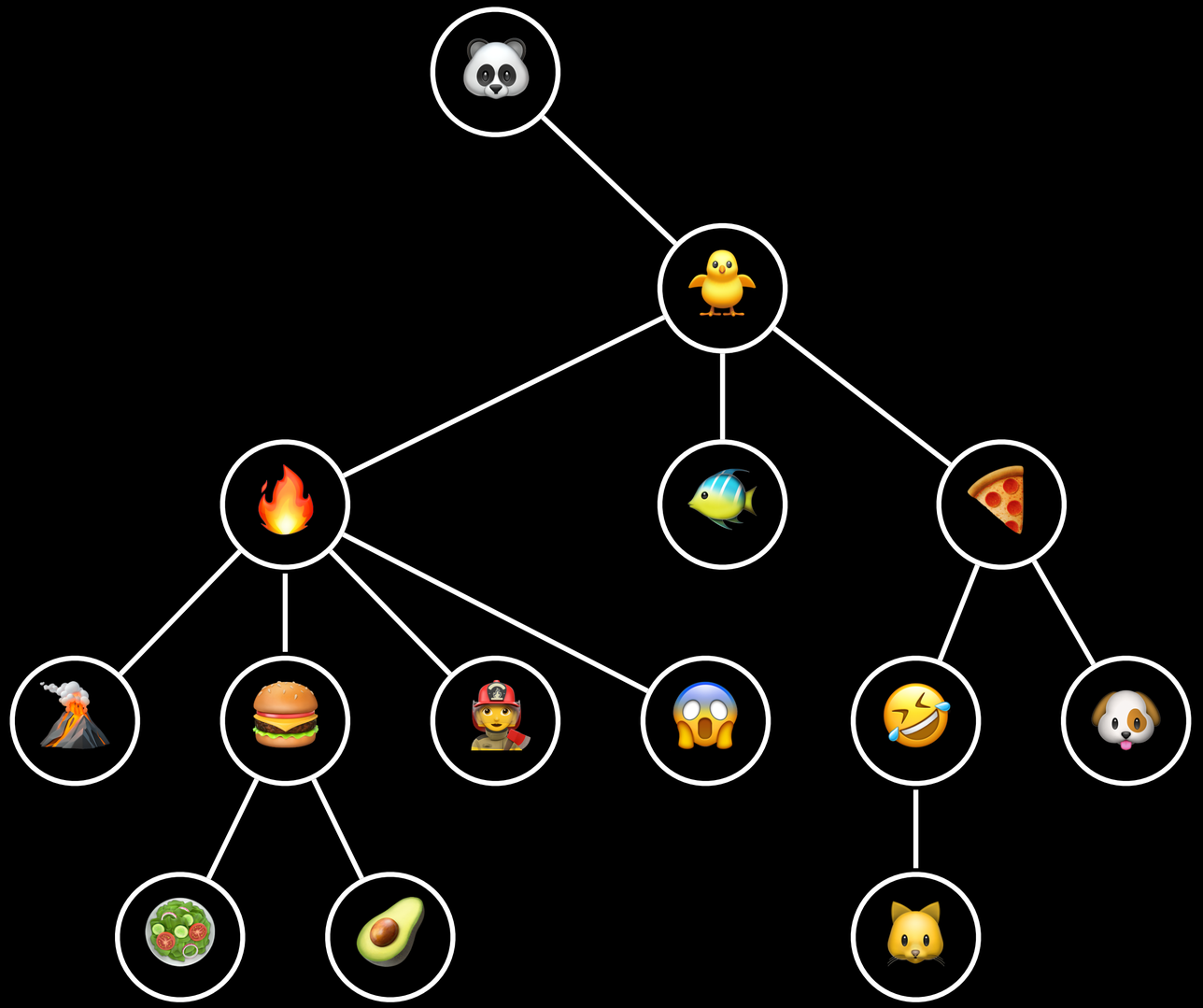
Root value and subtrees of children

Tree as a data structure

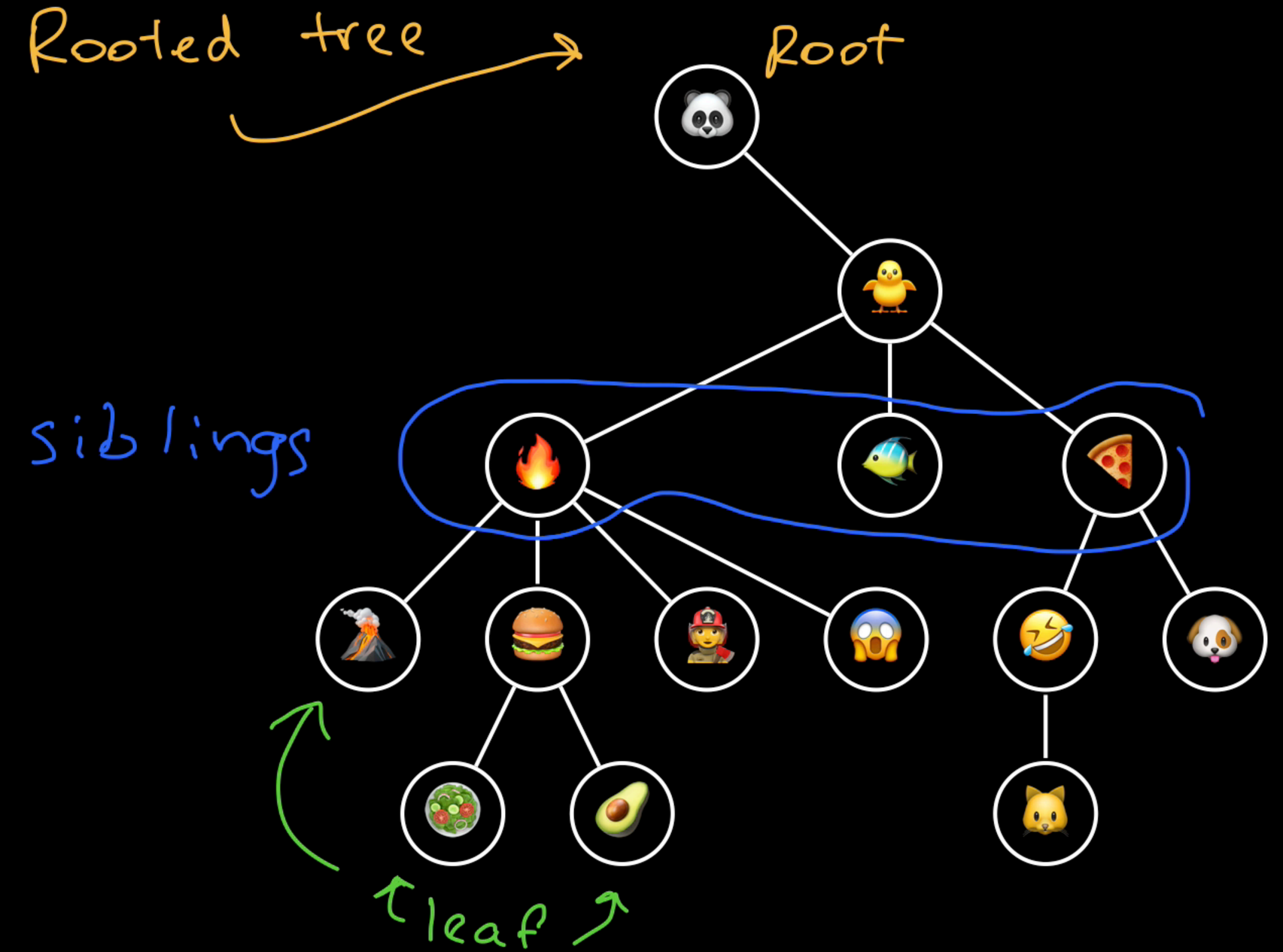
Graph
(a set of nodes and edges)

1 path between any 2 nodes

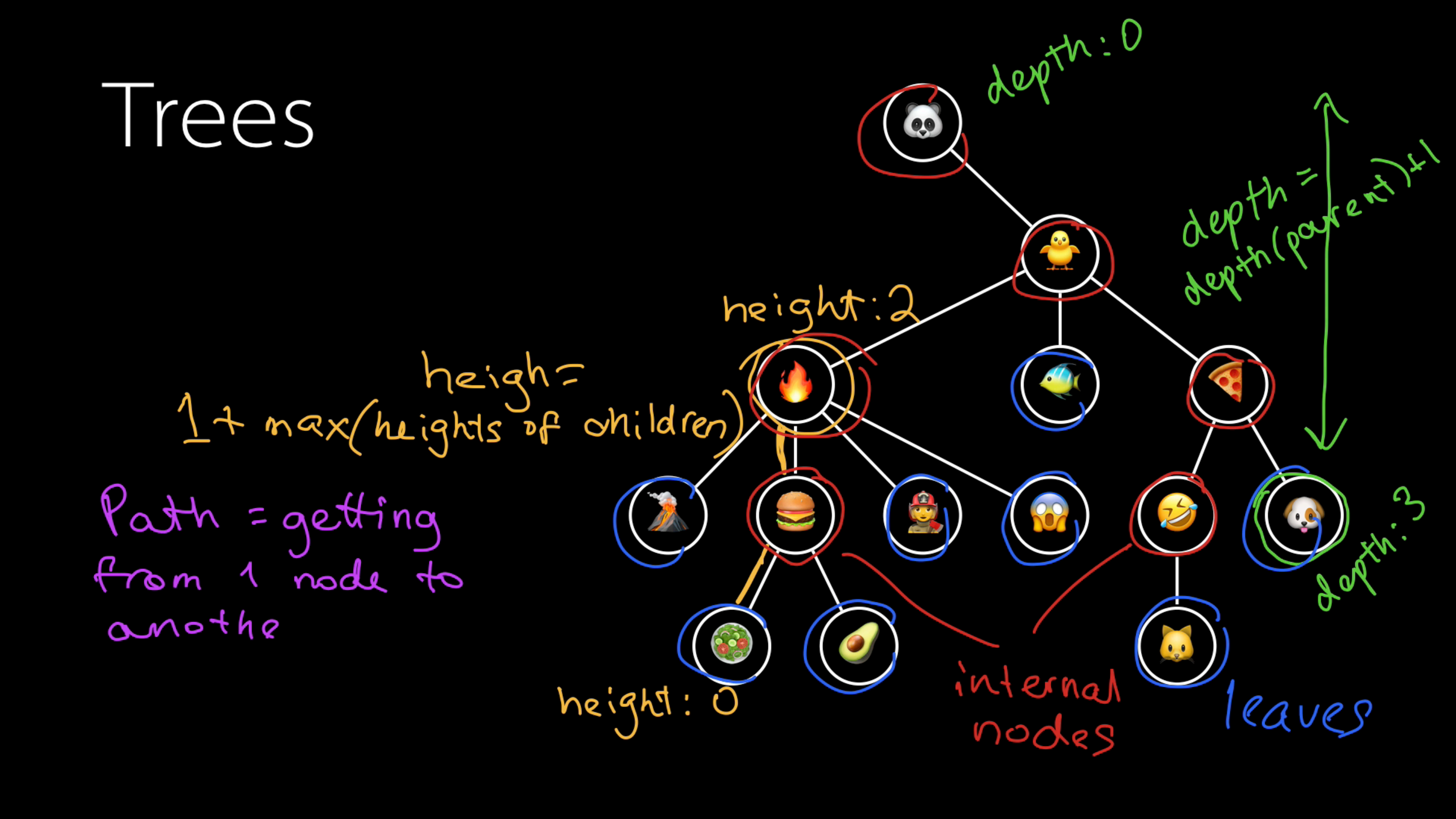
Trees



Trees



Trees

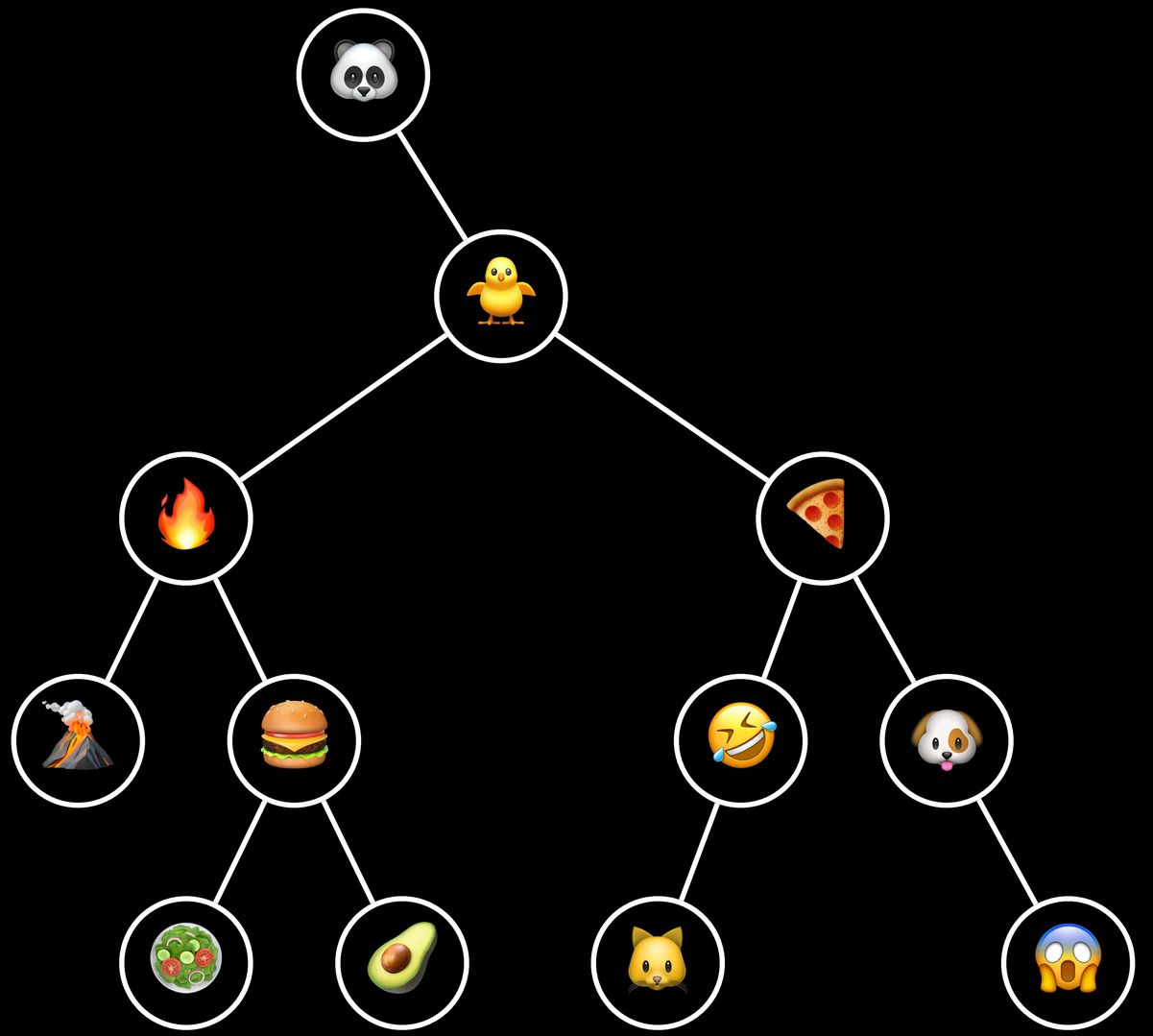

depth: 0
depth = depth(parent) + 1
height: 2
height =
1 + max(heights of children)
Path = getting from 1 node to anothe
depth: 3
height: 0
internal nodes
leaves

Binary tree

A rooted tree

No node has more than two children

Every child is either
a left child or a right child
of its parent



Binary Tree

```
template <typename T>
class BinaryTreeNode {
    T item;
    BinaryTreeNode *parent;
    BinaryTreeNode *left;
    BinaryTreeNode *right;
};

template <typename T>
class BinaryTree {
    BinaryTreeNode<T> *root;
    int size;
};
```

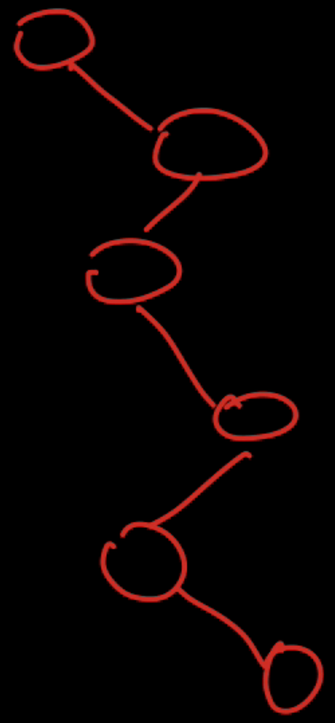
True or false

The height of a binary tree with N elements is log N.

True or false

maximal.io/section room EECS 183

The height of a binary tree with N elements is log N.



True or false

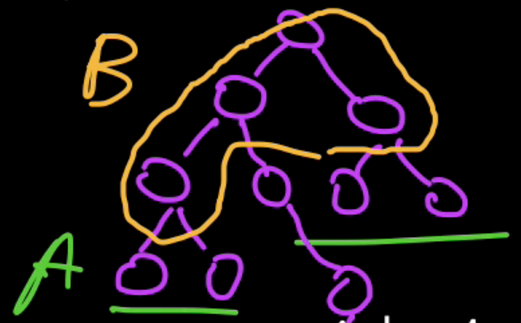
For any non-empty binary tree with A leaves and B nodes of degree 2, $A = B + 1$.

For any non-empty binary tree with A nodes and B nodes of degree 2, $A = B + 1$.

True or false

For any non-empty binary tree with A leaves and B nodes of degree 2, $A = B + 1$.

2 children



A : 5
B : 4

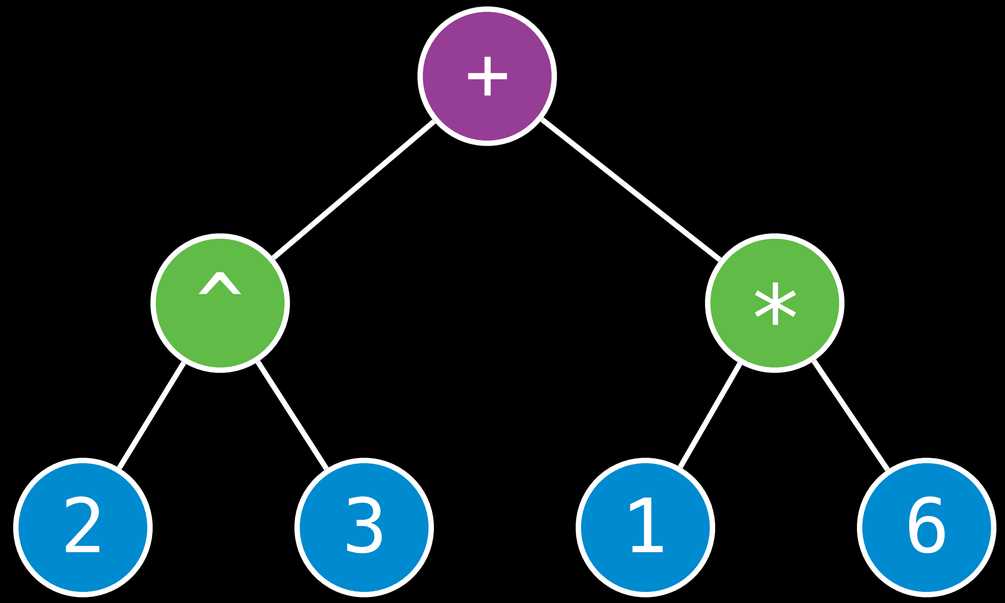
True

For any non-empty binary tree with A nodes and B nodes of degree 2, $A = B + 1$.

F

Tree traversals

Algorithm to visit each node
in a tree once



Pre-order traversal

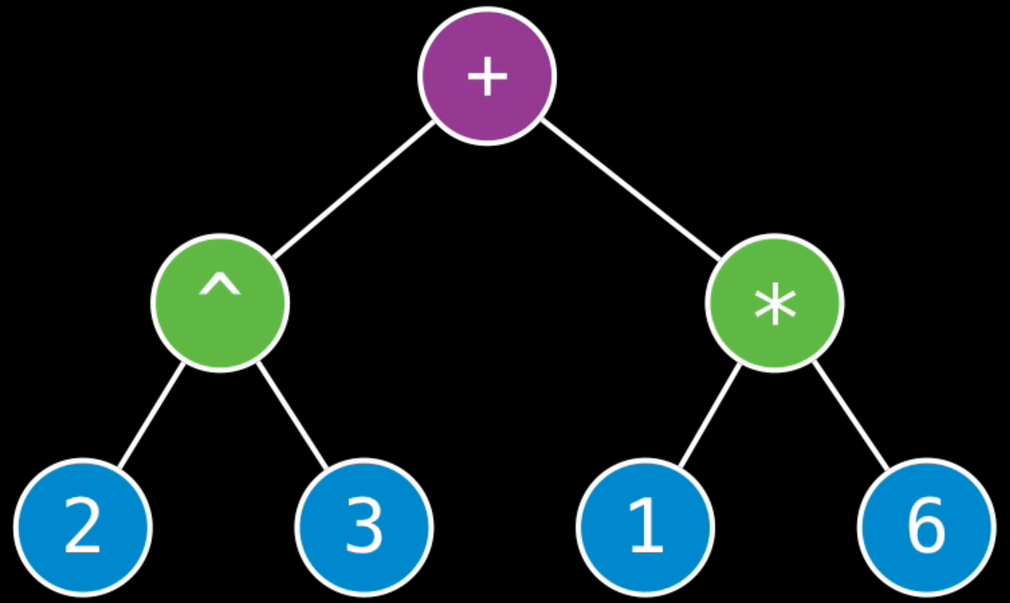
Post-order traversal

In-order traversal

Level-order traversal

Tree traversals

Algorithm to visit each node
in a tree once



Pre-order traversal Node, go left, go right

+ ^ 2 3 * 1 6 prefix

Post-order traversal go left, go right, node

2 3 ^ 1 6 * + postfix

Binary Tree only

In-order traversal go left, node, go right

2 ^ 3 + 1 * 6 infix

Level-order traversal

go level by level (nodes with same depth)

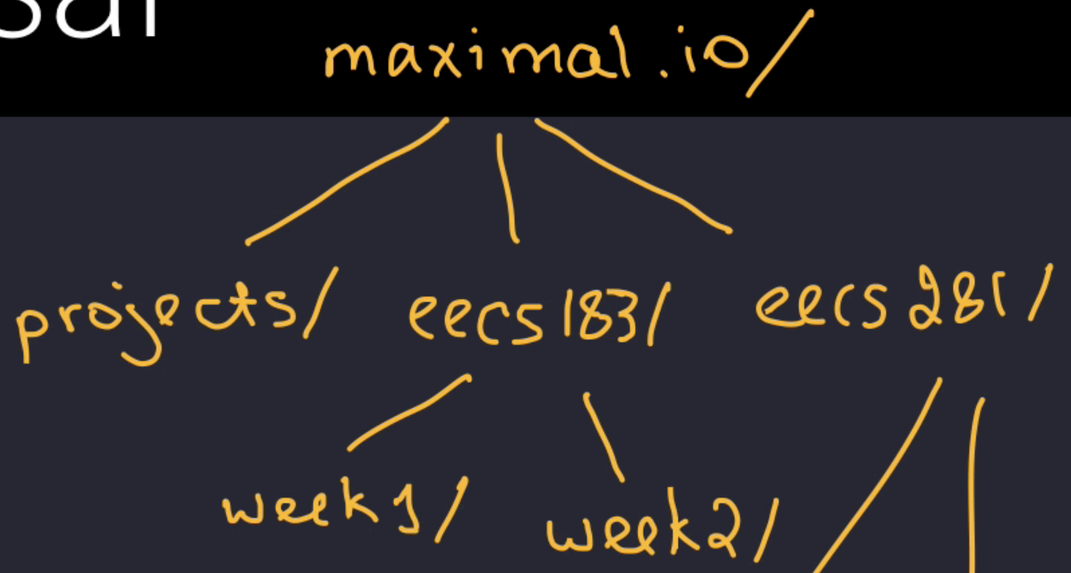
+ ^ * 2 3 1 6

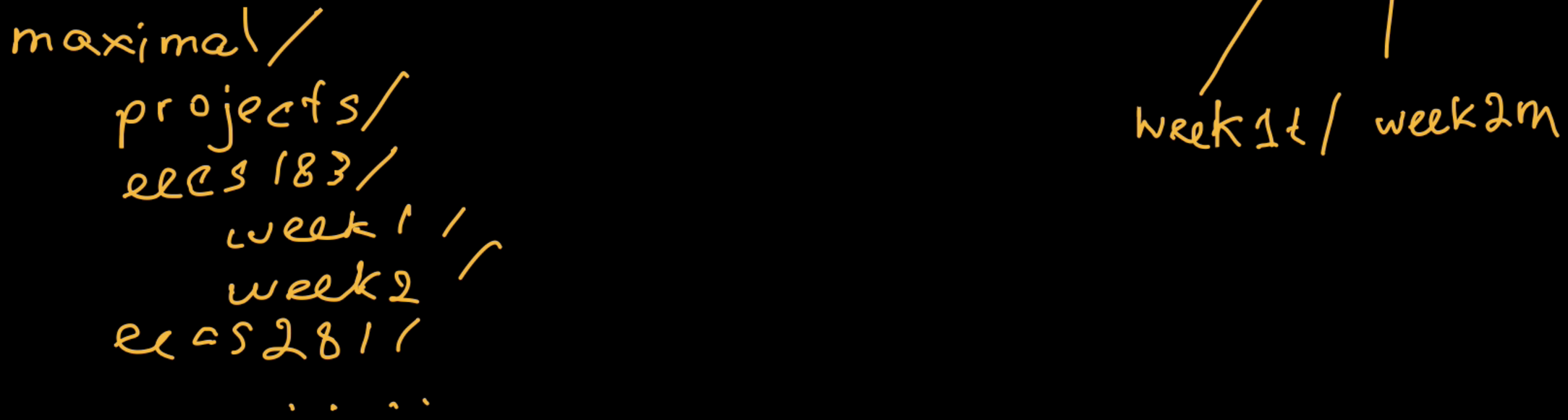
Pre-order traversal

```
preOrder(node):
    if node ≠ nil:
        visit(node.key)
        preOrder(node.left)
        preOrder(node.right)
```

Pre-order traversal

```
preOrder(node):
    if node ≠ nil:
        visit(node.key)
        preOrder(node.left)
        preOrder(node.right)
```





Post-order traversal

```
postOrder(node):
    if node ≠ nil:
        postOrder(node.left)
        postOrder(node.right)
        visit(node.key)
```

In-order traversal

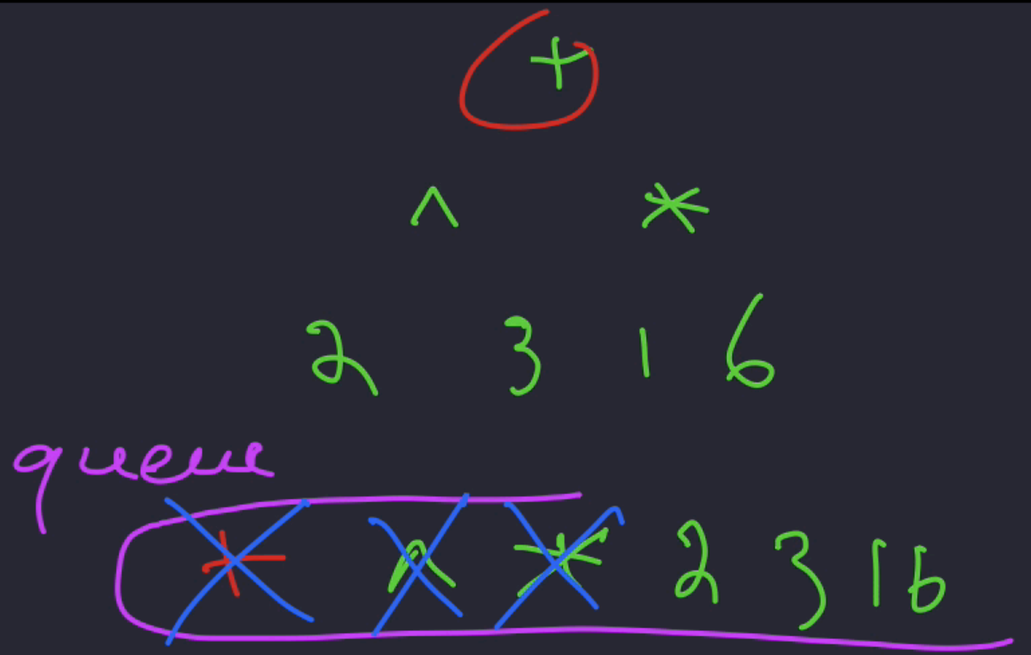
```
inOrder(node):
    if node ≠ nil:
        inOrder(node.left)
        visit(node.key)
        inOrder(node.right)
```

Level-order traversal

```
levelOrder(node):
    q = empty queue
    q.enqueue(node)
    while q is not empty:
        node = q.dequeue()
        visit(node)
        if node.left ≠ nil:
            q.enqueue(node.left)
        if node.right ≠ nil:
            q.enqueue(node.right)
```

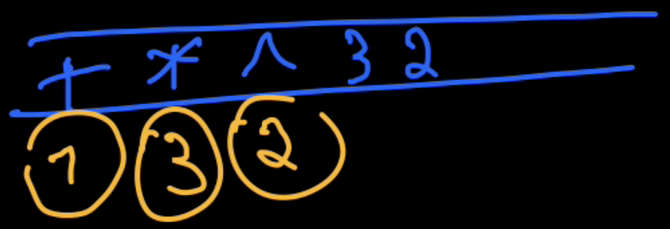
Level-order traversal

```
levelOrder(node):
    q = empty queue
    q.enqueue(node)
    while q is not empty:
        node = q.dequeue()
        visit(node)
        if node.left ≠ nil:
            q.enqueue(node.left)
        if node.right ≠ nil:
            q.enqueue(node.right)
```



if : ① use stack

② push children right → left



→ pre-order

Tree traversals

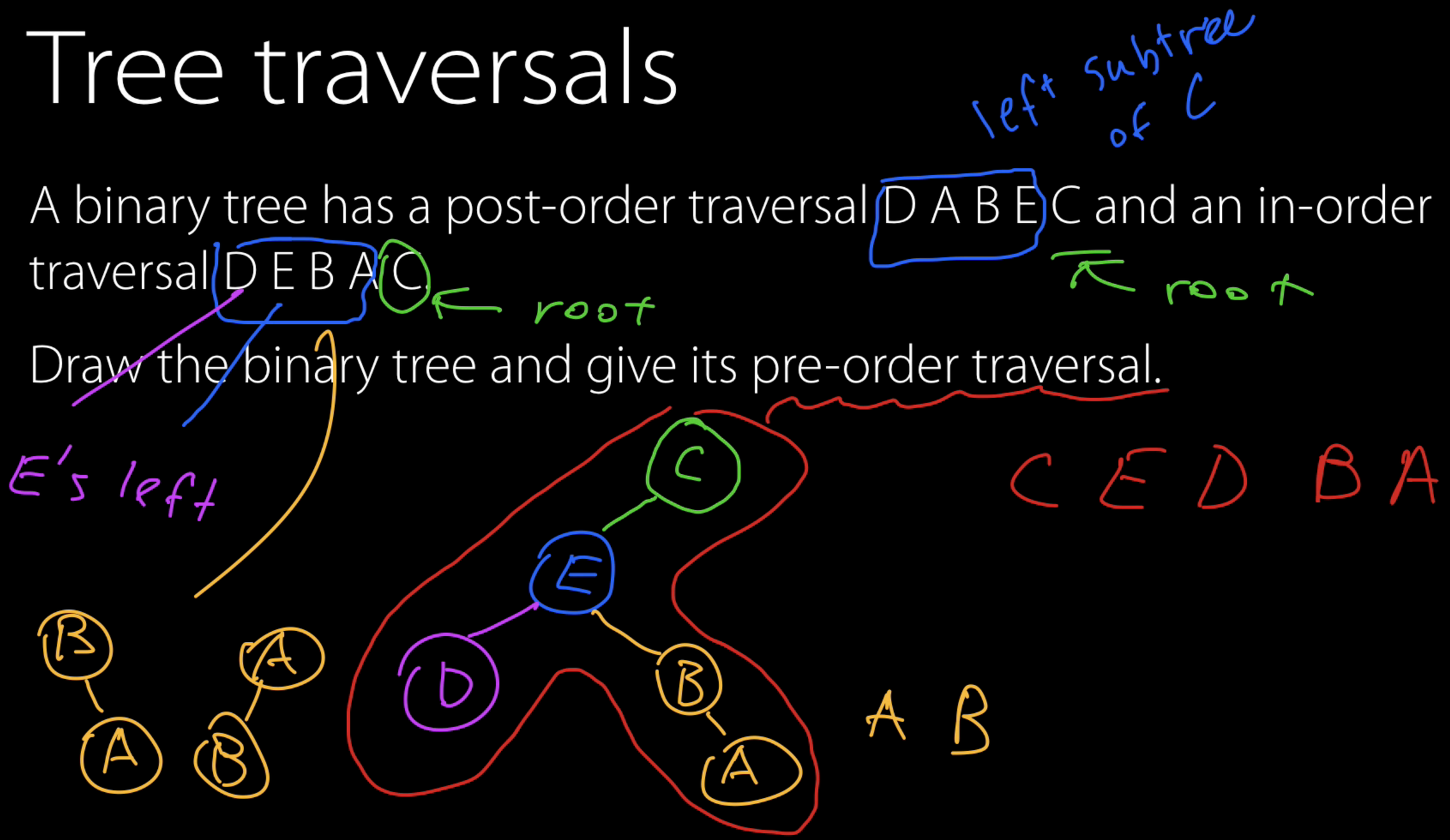
A binary tree has a post-order traversal D A B E C and an in-order traversal D E B A C.

Draw the binary tree and give its pre-order traversal.

Tree traversals

A binary tree has a post-order traversal D A B E C and an in-order traversal D E B A C

Draw the binary tree and give its pre-order traversal.



Ordered dictionary ADT

Insert

Find

Remove

Minimum

Maximum

"AAA"	"Anaa Airport"
...	
"DTW"	"Detroit"
"LHR"	"London Heathrow"
"SFO"	"San Francisco"
"YYZ"	"Toronto Pearson"
...	
"ZZZ"	"Aarhus Sea Airport"

Ordered dictionary ADT

Insert

Find

Remove

Minimum

Maximum

successor

predecessor

"AAA"	"Anaa Airport"
...	
"DTW"	"Detroit"
"LHR"	"London Heathrow"
"SFO"	"San Francisco"
"YYZ"	"Toronto Pearson"
...	
"ZZZ"	"Aarhus Sea Airport"

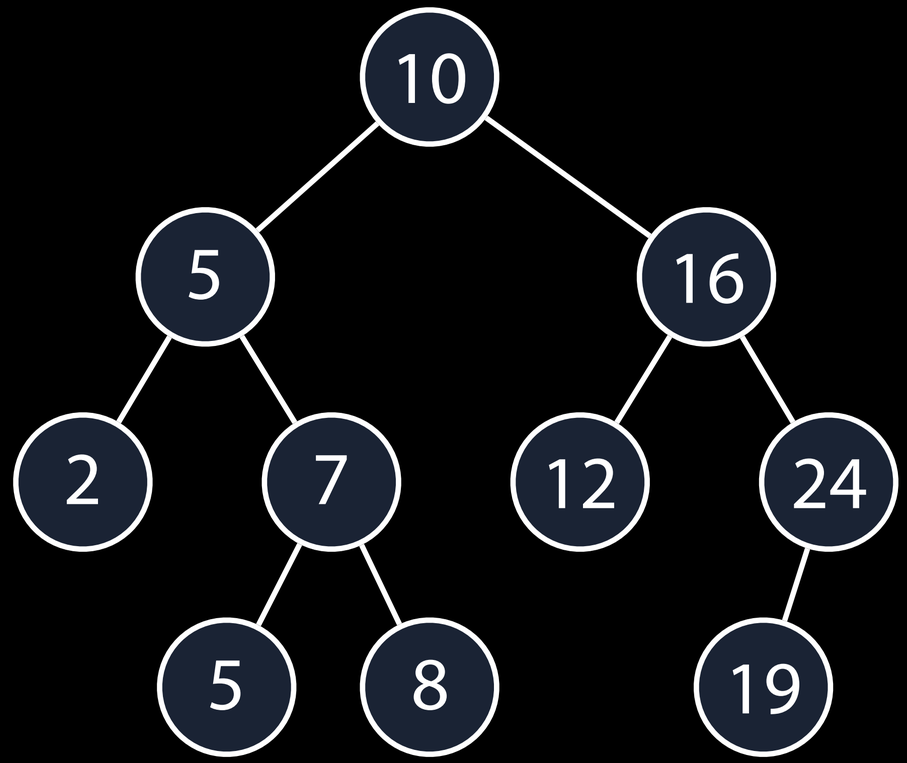
Binary search tree (BST)

Binary tree

Sorted

Binary-search-tree property

In-order traversal visits the nodes in sorted order.



Binary search tree (BST)

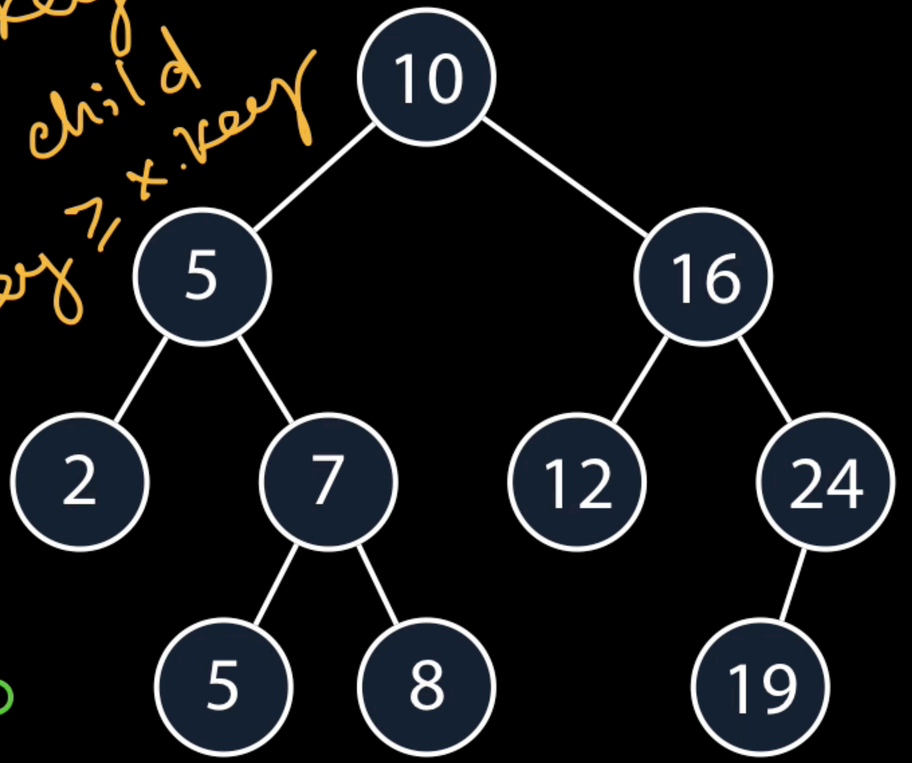
Binary tree

Sorted

Binary-search-tree property

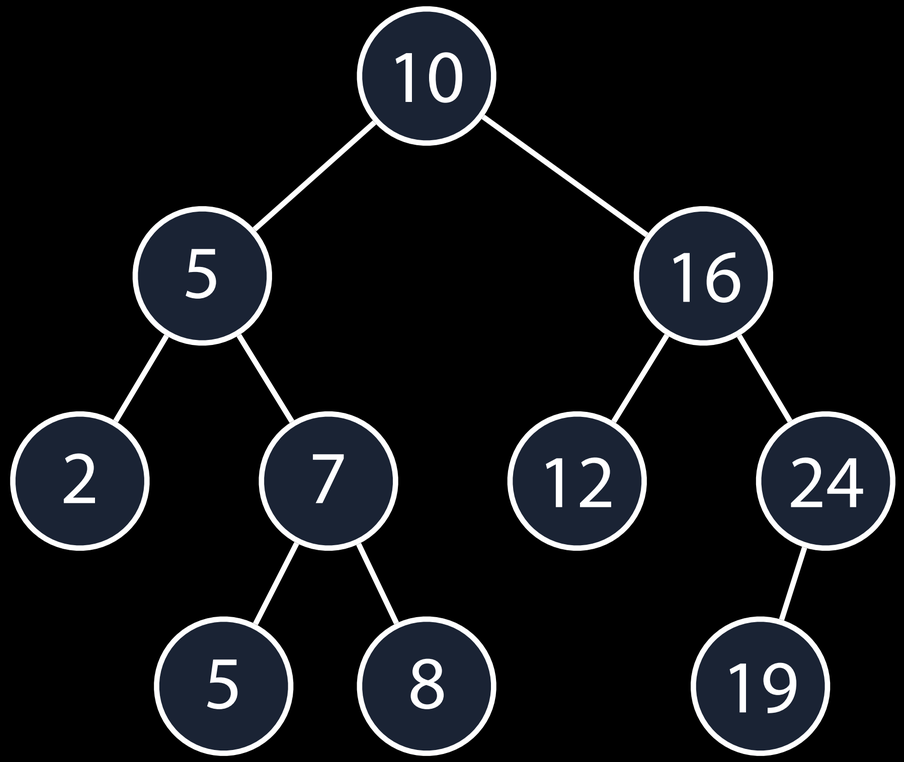
In-order traversal visits the nodes in sorted order.

node x
y is left child
if $y.key \leq x.key$
y is right child
if $y.key \geq x.key$



2 5 5 7 8 10
12 16 19 24

BST find



BST find

7

① check node if ==

↙

return

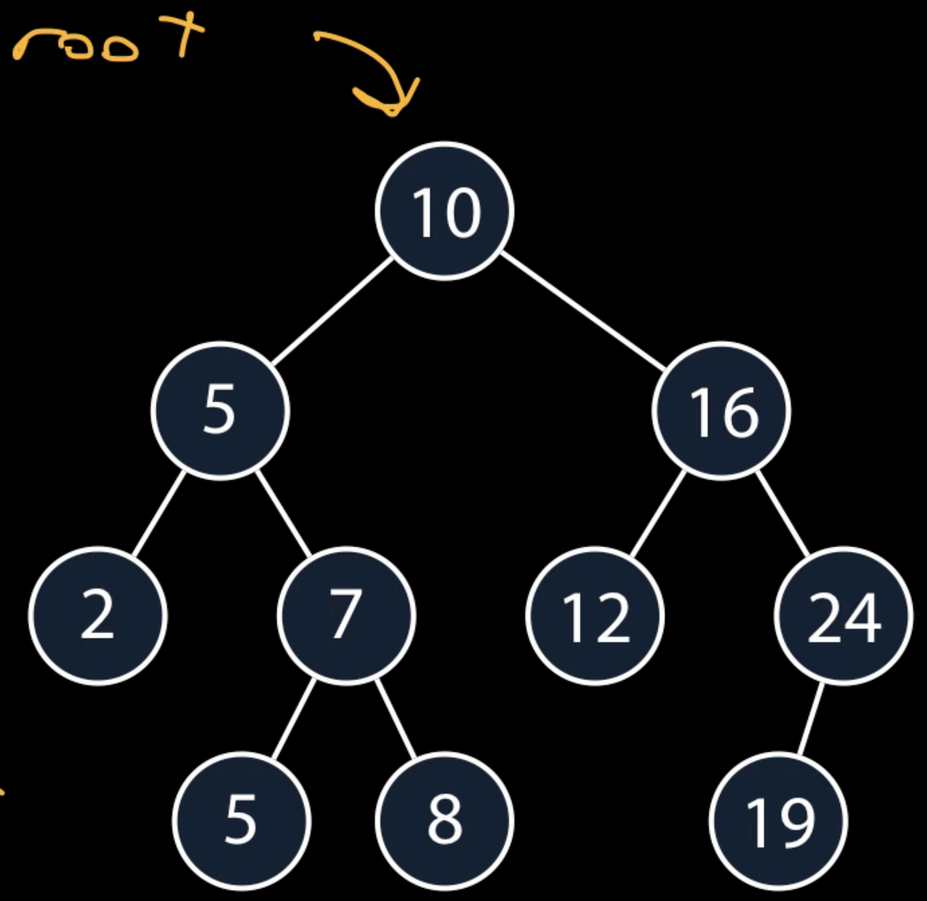
↘ N

node = node.left

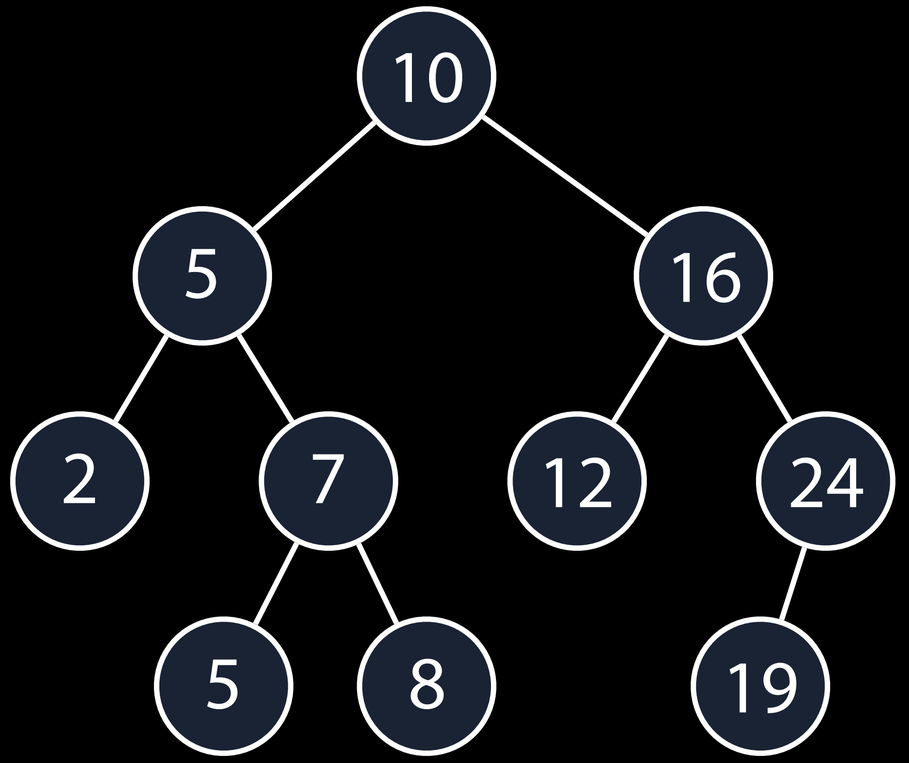
if <

node = node.right

if >



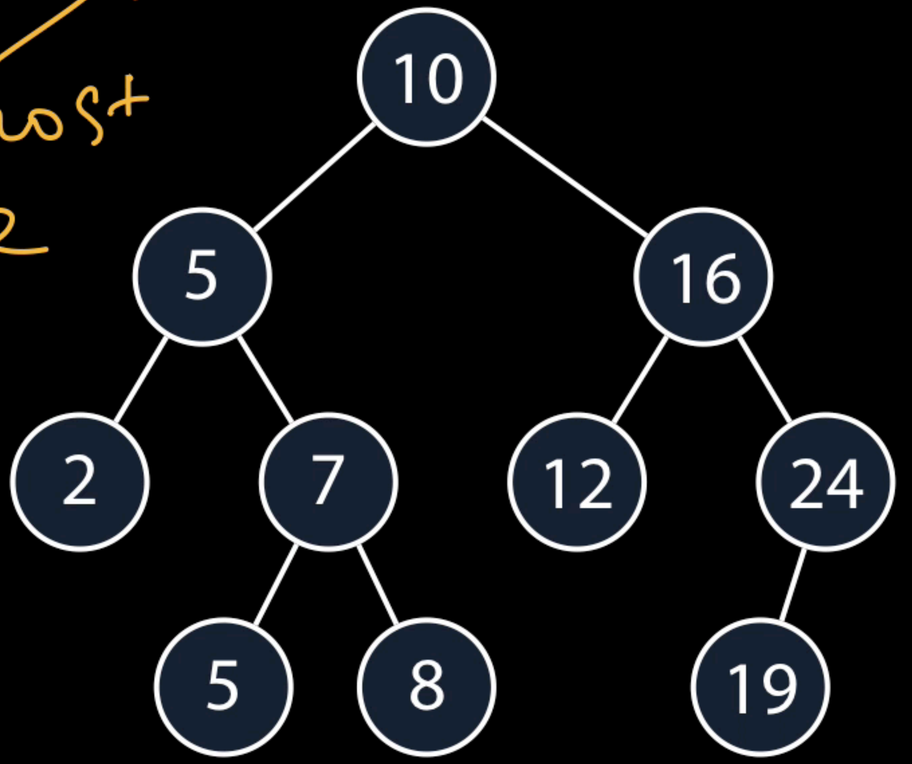
BST minimum and maximum



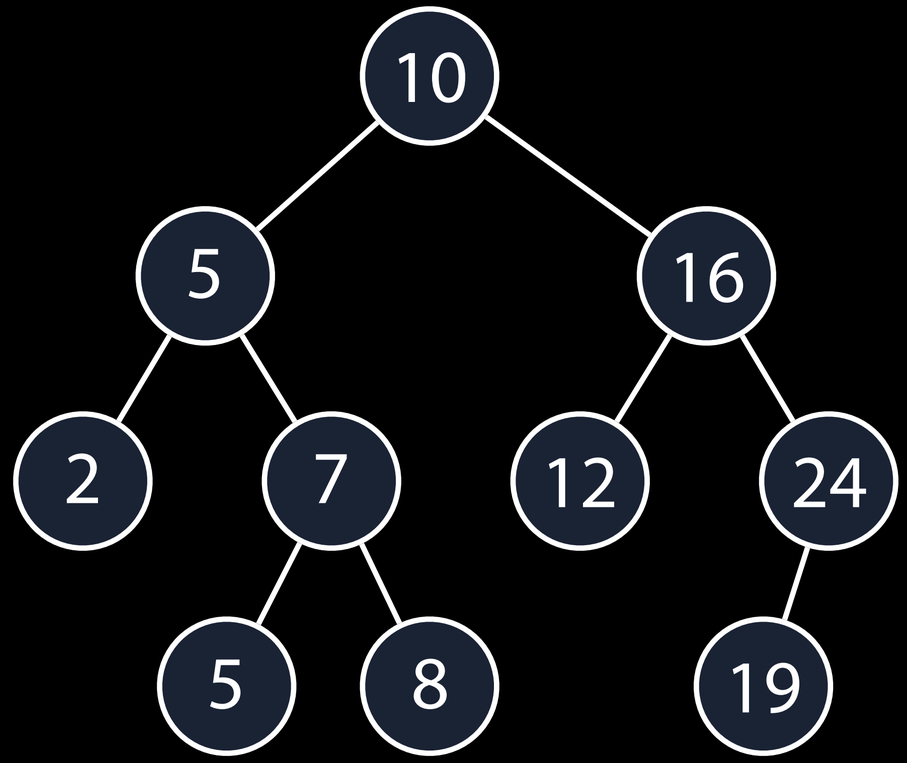
BST minimum and maximum

leftmost node

rightmost node



BST insert

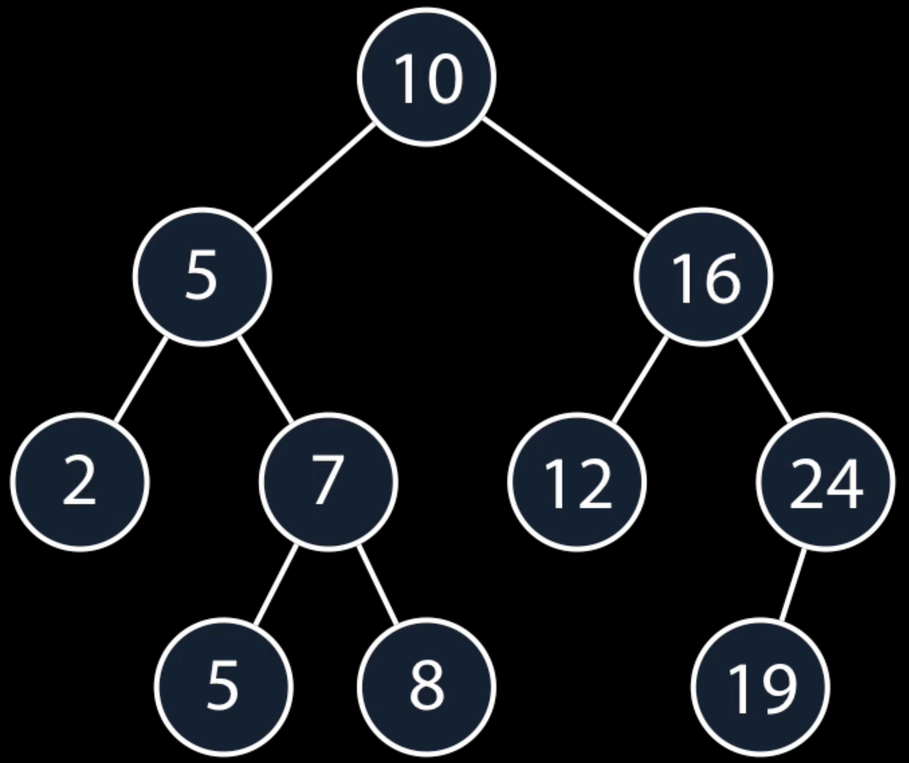


BST insert

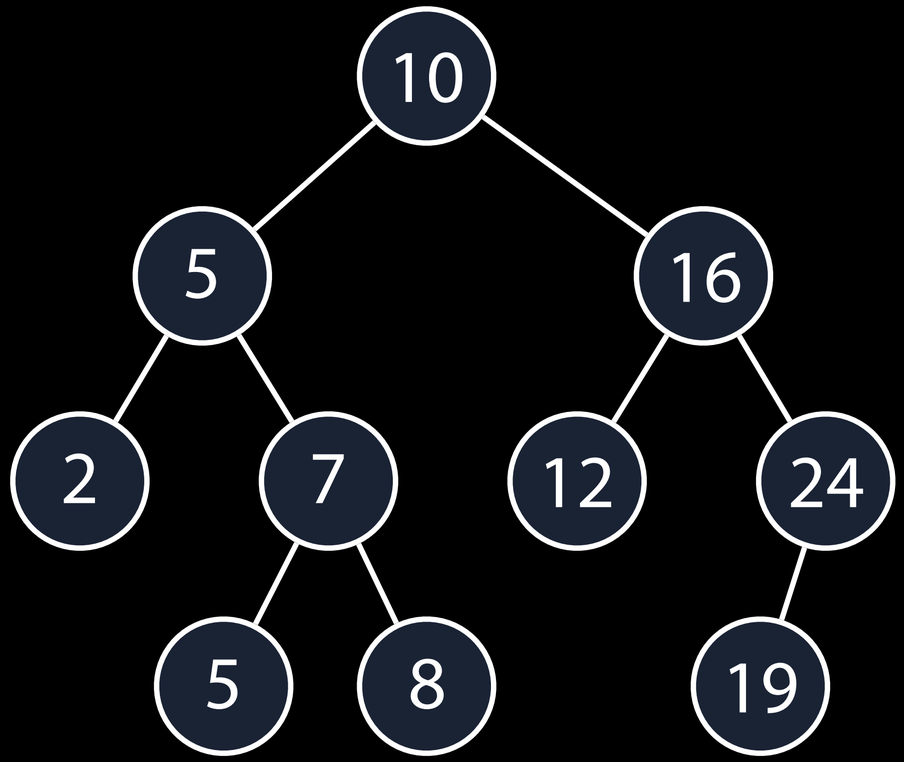
Same as find

When we reach nullptr

replace with new node



BST remove



BST remove

① same as find

② - if nullptr → done

- if found :

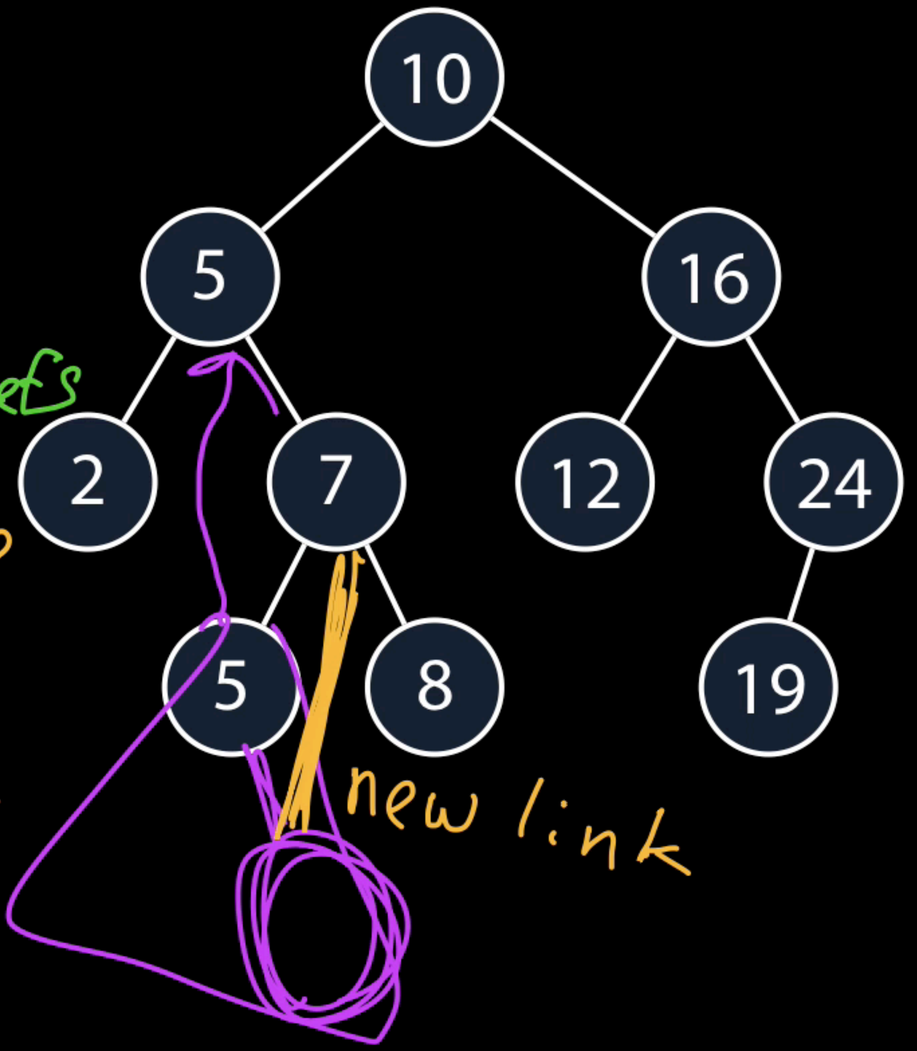
0) no children ✓ delete refs

1) 1 child : move child up

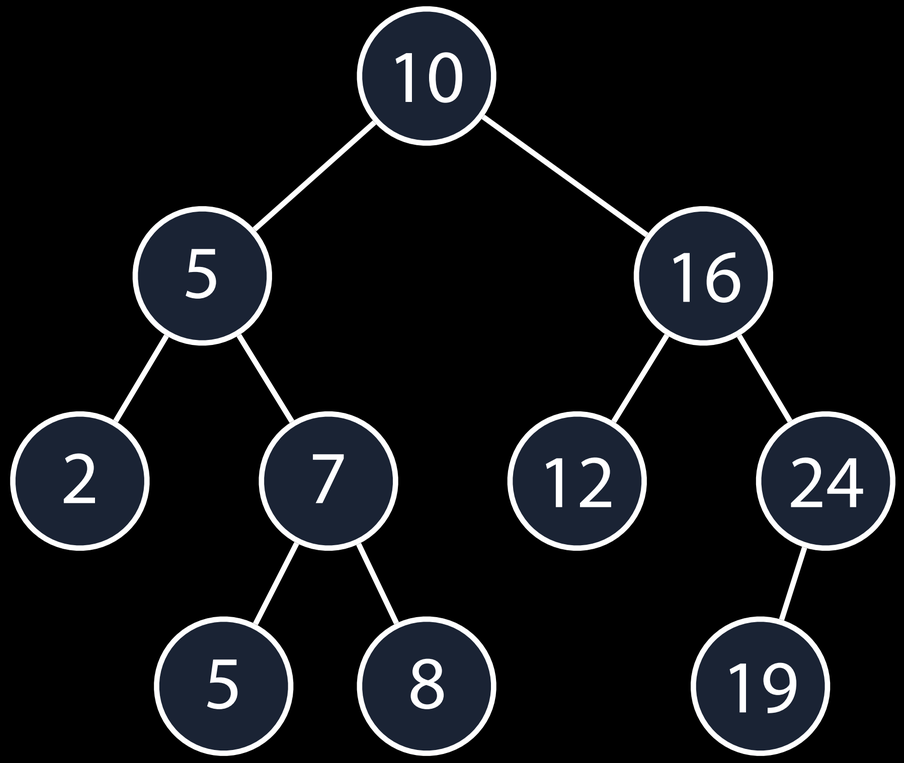
delete 24

2) 2 children: replace with successor

delete 5



BST successor and predecessor



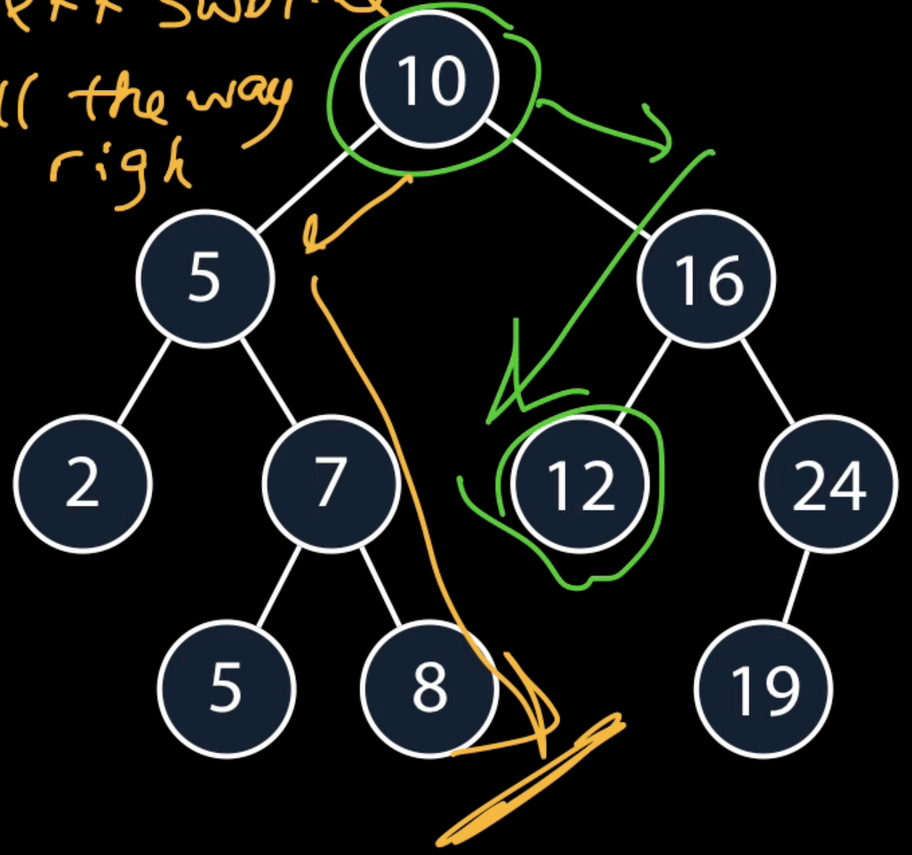
BST successor and predecessor

right subtree

all the way left

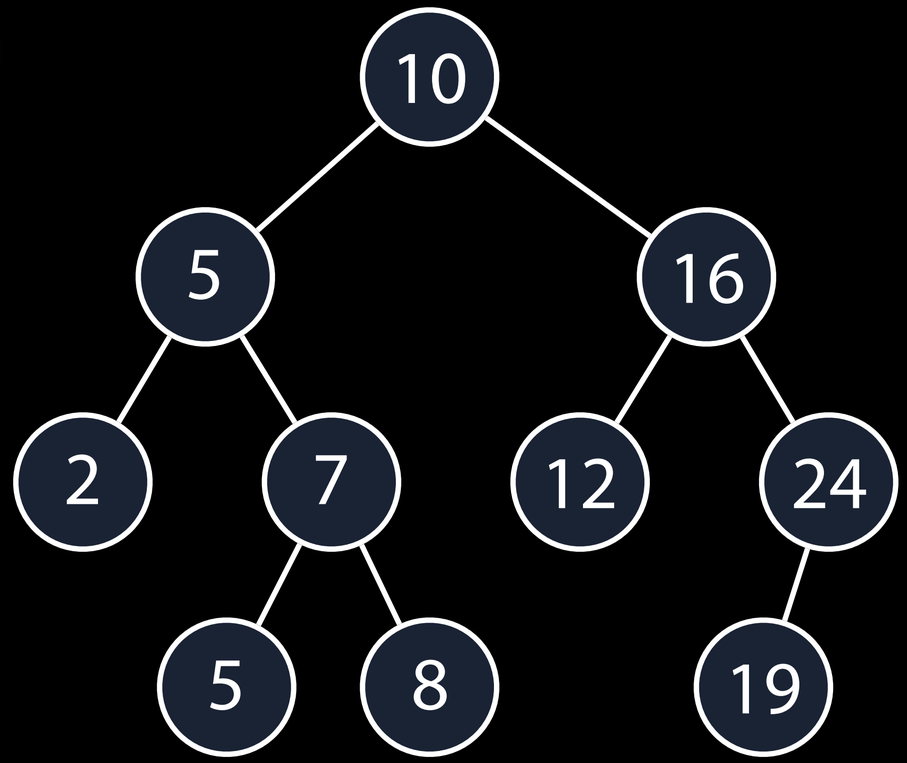
left subtree

all the way right



BST complexity

Complexity of find, insert, remove: $O(h)$



BST complexity

Complexity of find, insert, remove: $O(h)$

height

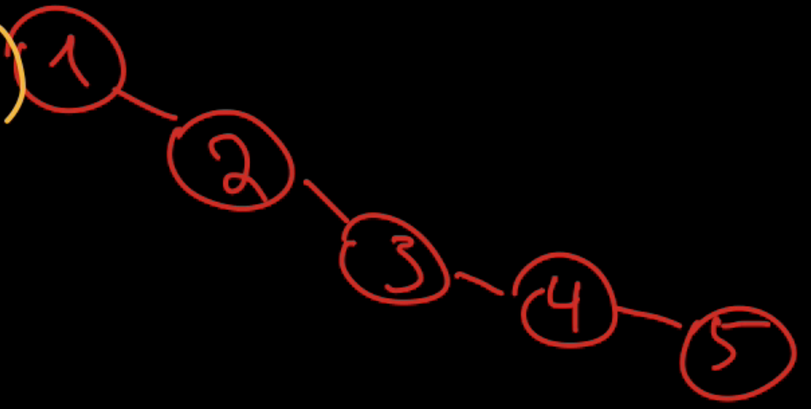
Best case:

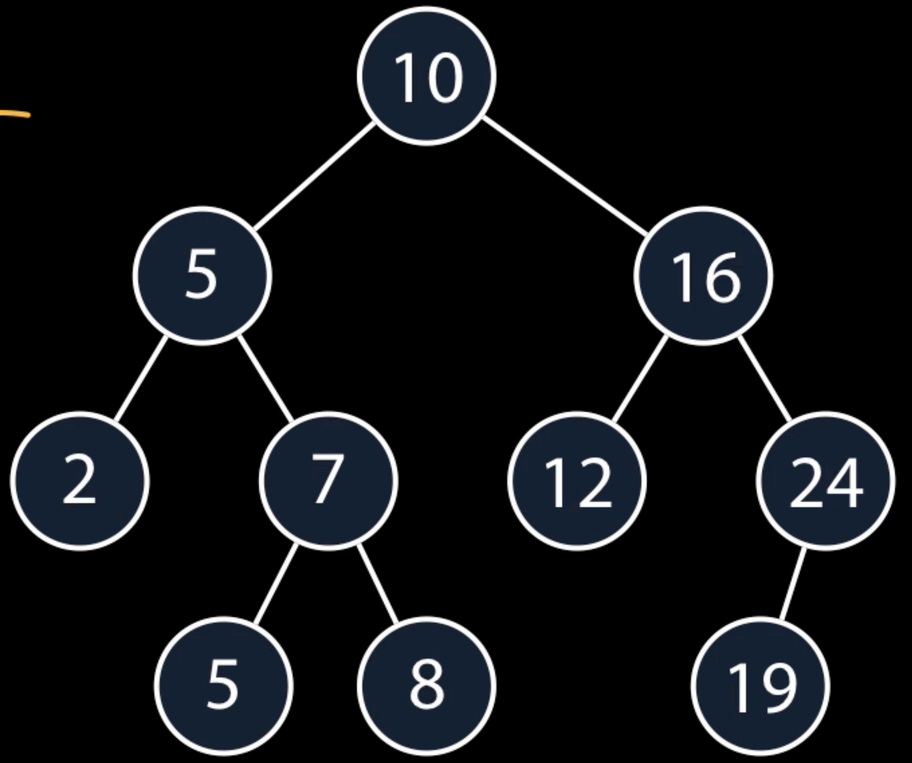
perfectly balanced tree

height: $\Theta(\log n)$

Worst case: 1 2 3 4 5

height: $\Theta(n)$





Balanced BSTs

Perfectly balanced binary tree with height h

Number of nodes $n = 2^{h+1} - 1$

No node has depth $d > \log_2 n$

All the leaf nodes are at the same level

Time complexity of find, insert, remove: $O(\log n)$

Balanced BSTs

Perfectly balanced binary tree with height h

Number of nodes $n = 2^{h+1} - 1$

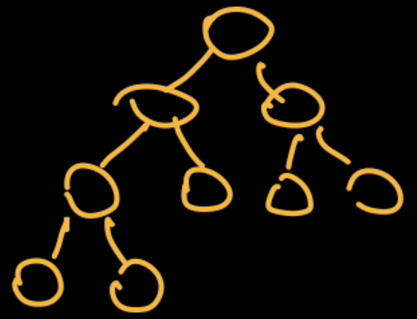
height $\Theta(\log_2 n)$

No node has depth $d > \log_2 n$

All the leaf nodes are at the same level

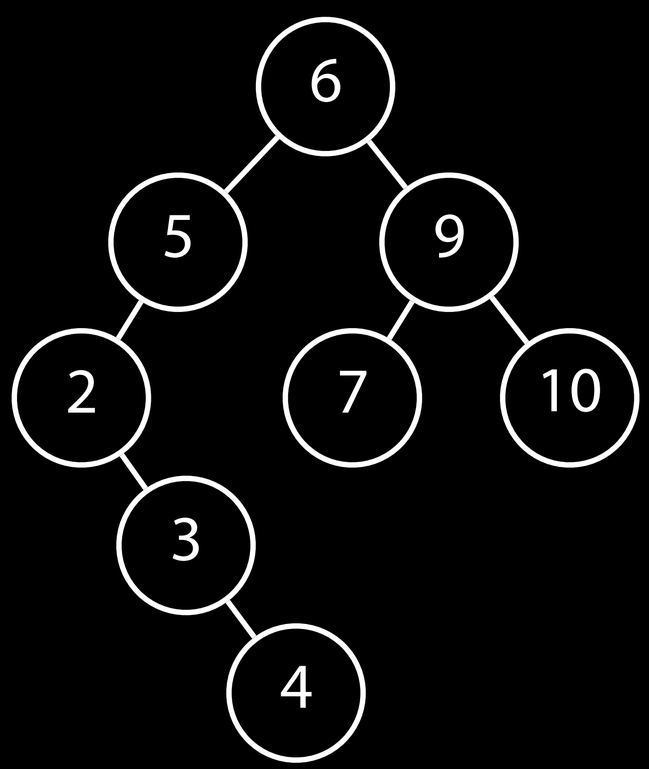
not always true or different by 1

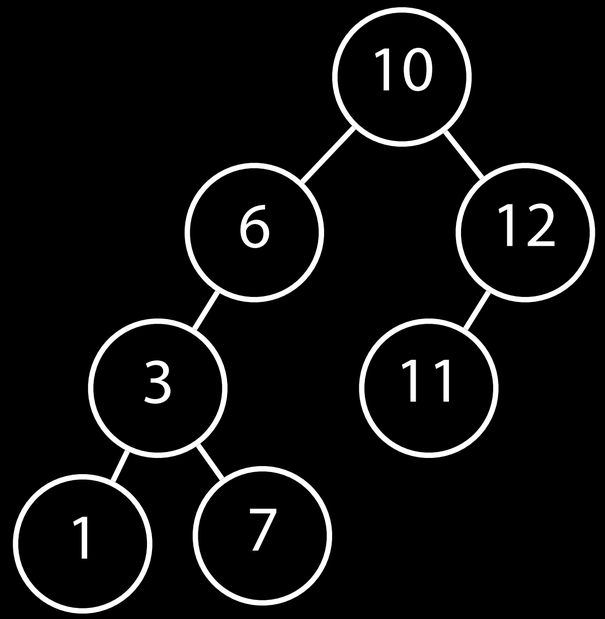
Time complexity of find, insert, remove: $O(\log n)$

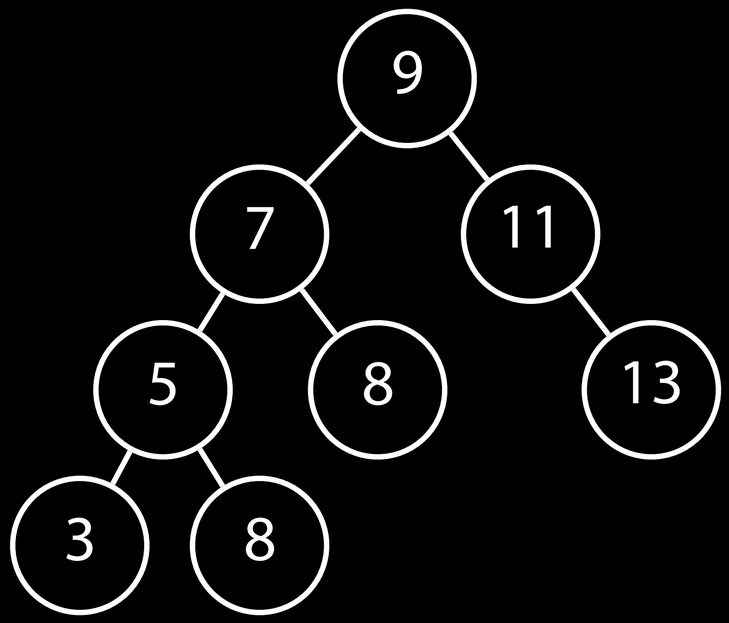


Balanced BSTs

For each following binary tree, determine if it is a BST, and whether it has minimum-BST- height (the height of the tree is the same as the height of the optimal binary search tree containing the given elements).

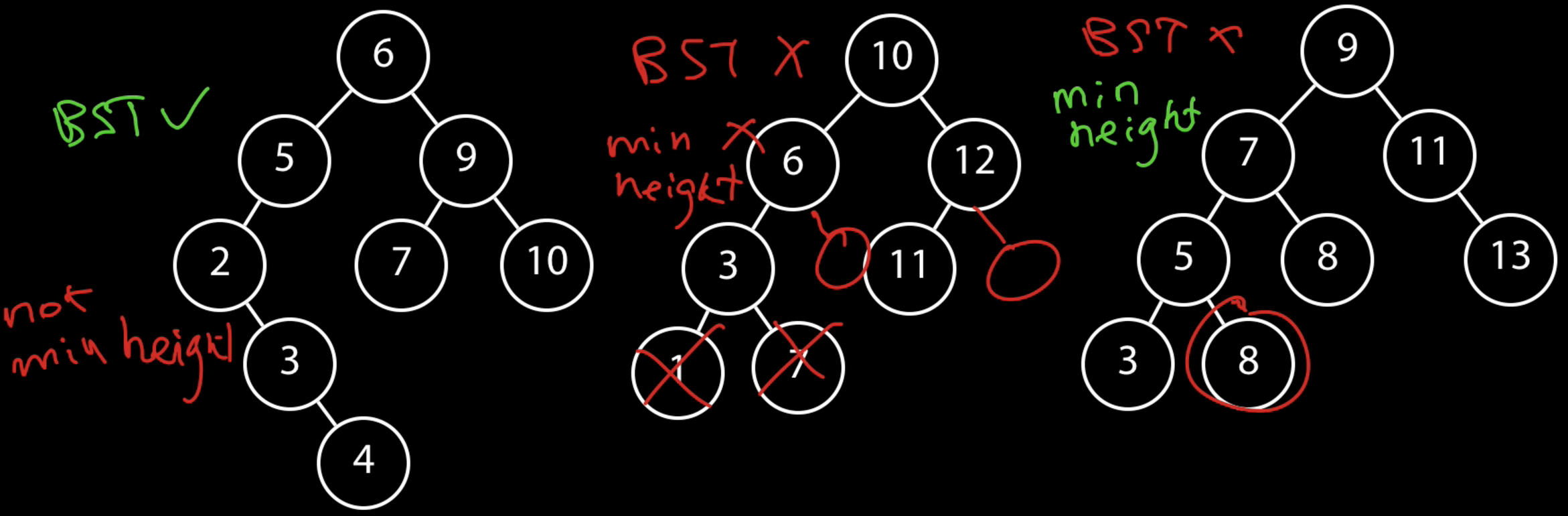






Balanced BSTs

For each following binary tree, determine if it is a BST, and whether it has minimum-BST- height (the height of the tree is the same as the height of the optimal binary search tree containing the given elements).



Balanced BSTs

Can we determine whether or not a BST is minimum-BST-height without having to check the values of each node? If so, how? If not, why not?

Let `h` be the height of the tree an `n` be the number of nodes.

Check that `h = floor(log(n))`.

Balanced BSTs

AVL Trees

B-Trees/2-3-4 Trees

BB[α] Trees

Red-black Trees

Splay-Trees

Skip Lists

Scapegoat Trees

Treaps

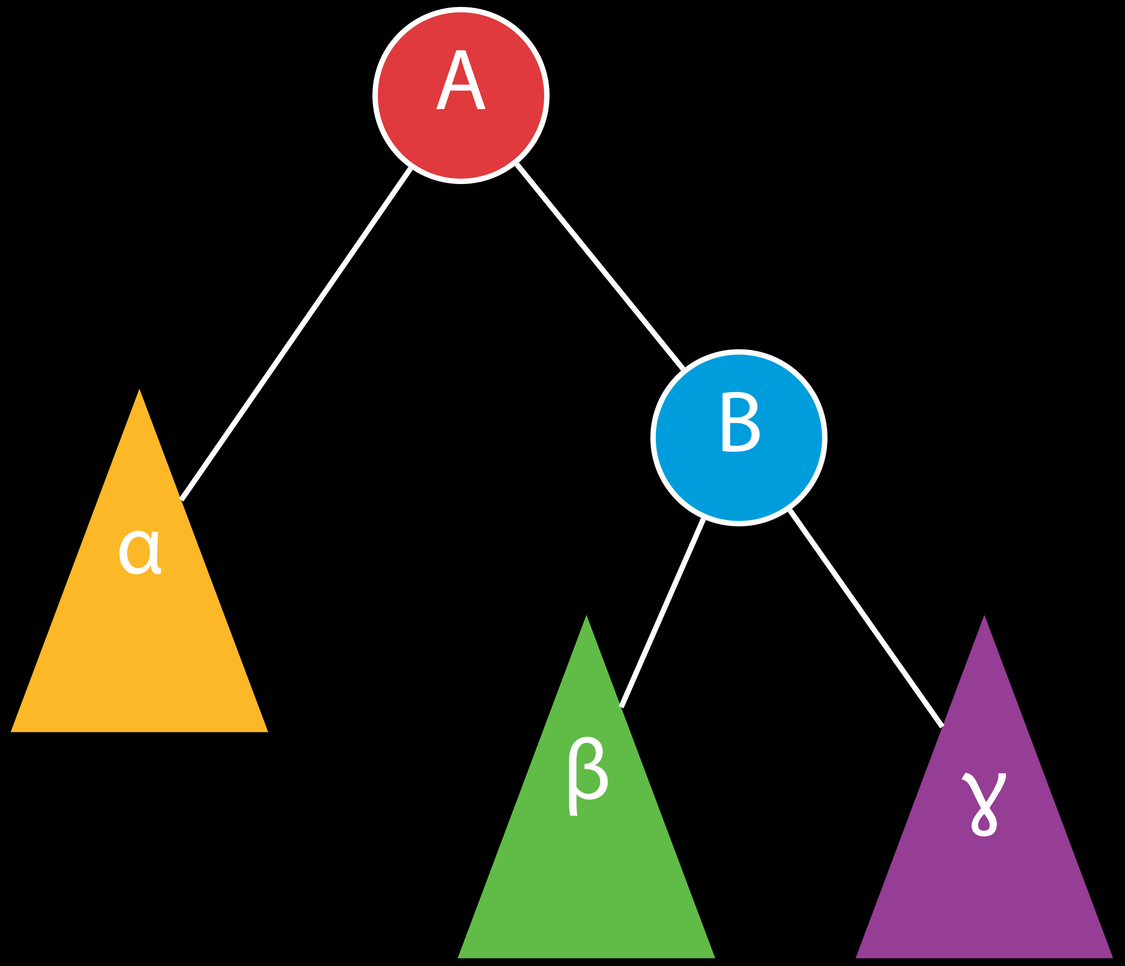
Definitions

Height of a node: max(height of left child, height of right child). Height of a leaf is 0.

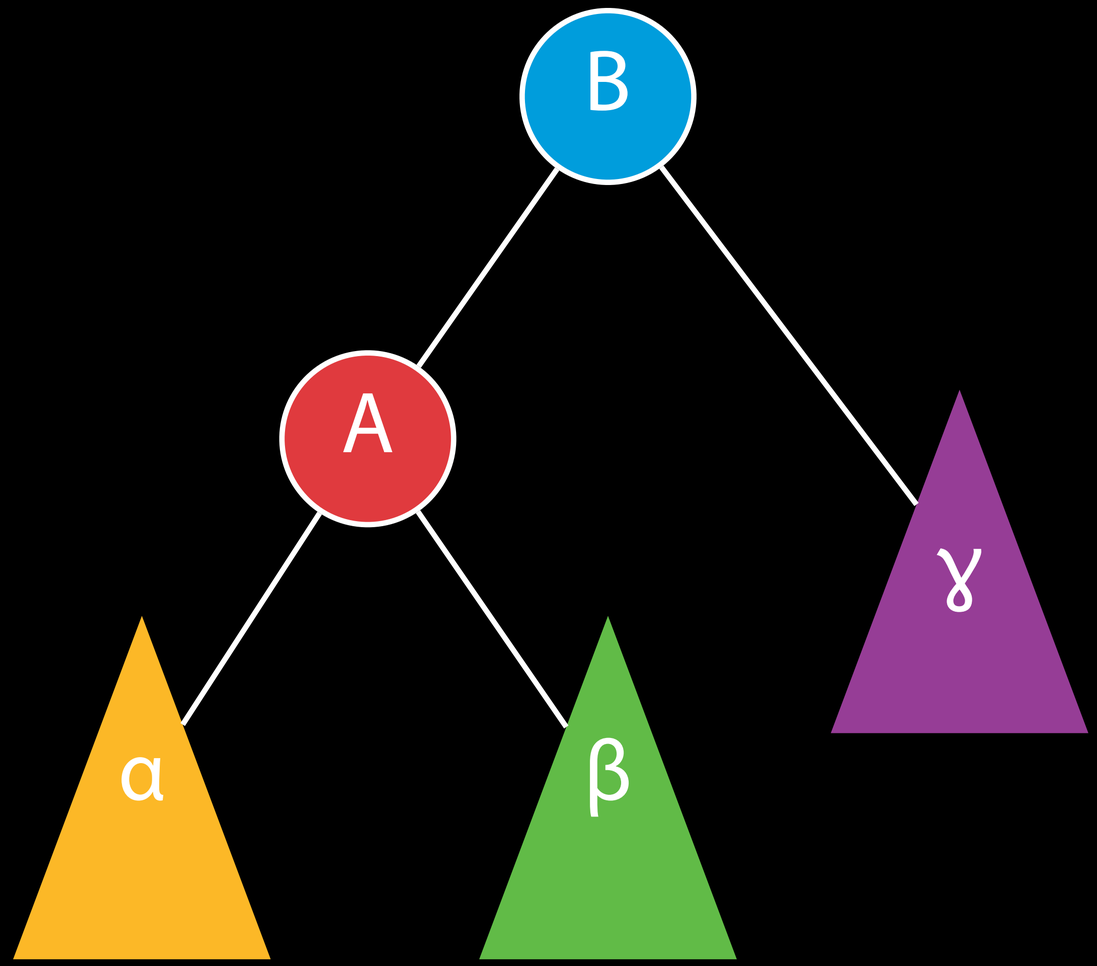
In-order predecessor: go to left subtree and find the right-most node.

In-order successor: go to right subtree and find the left-most node.

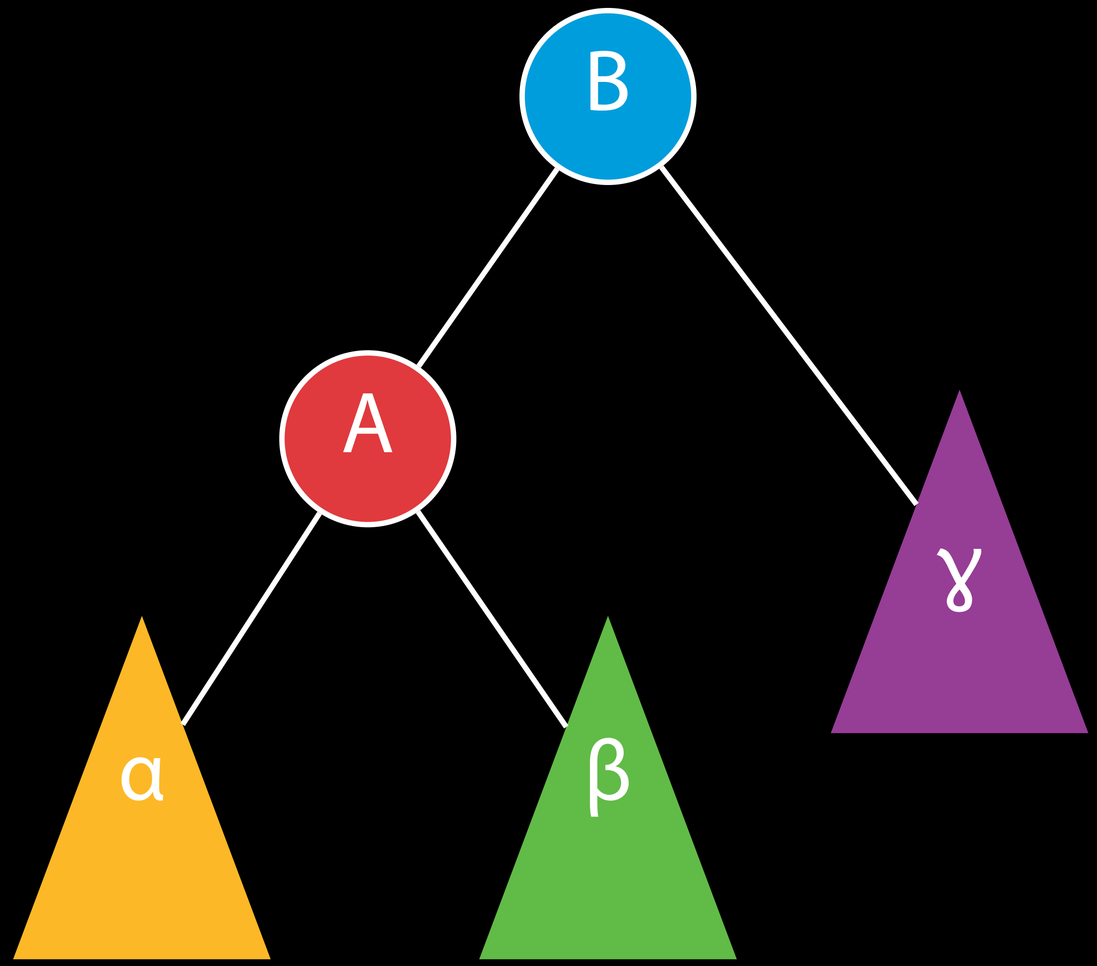
Left Rotation on A



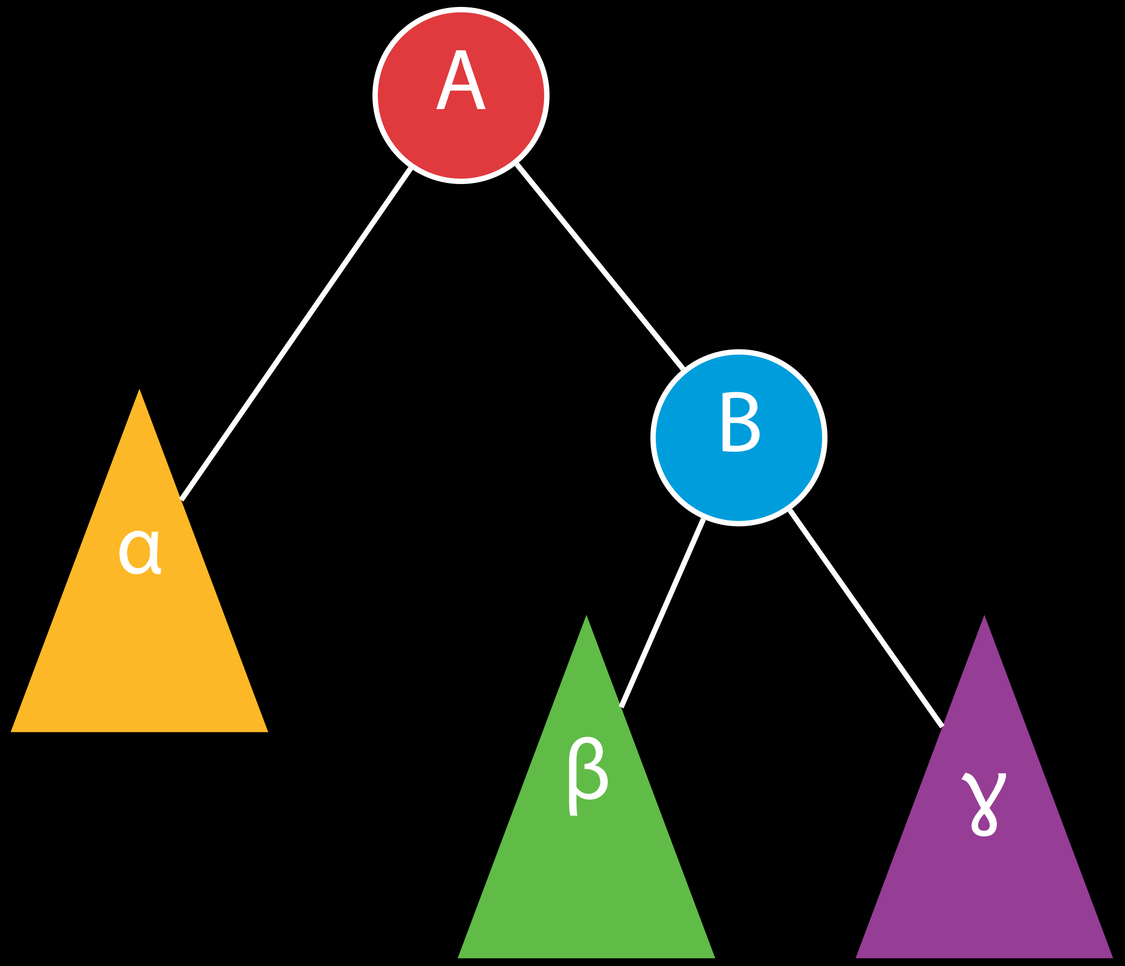
Left Rotation on A



Right Rotation on B

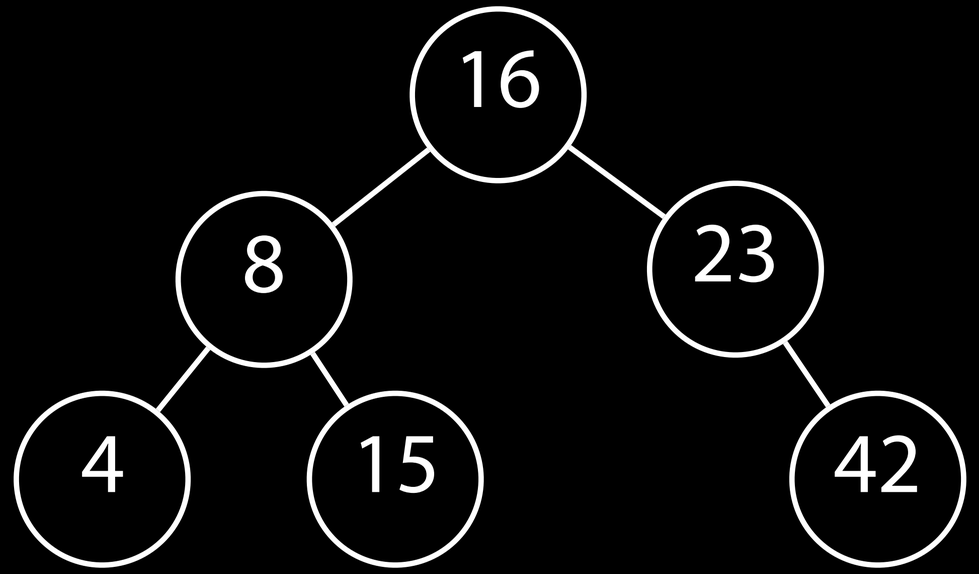


Right Rotation on B



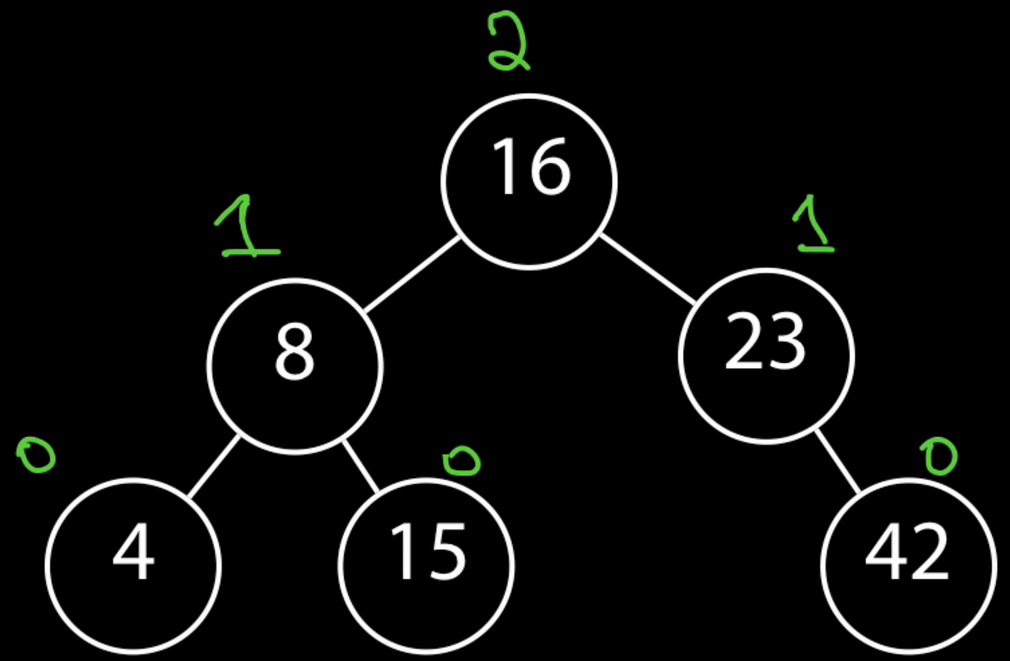
AVL Trees

Binary search tree. For any node, heights of the two child subtrees differ by at most 1.

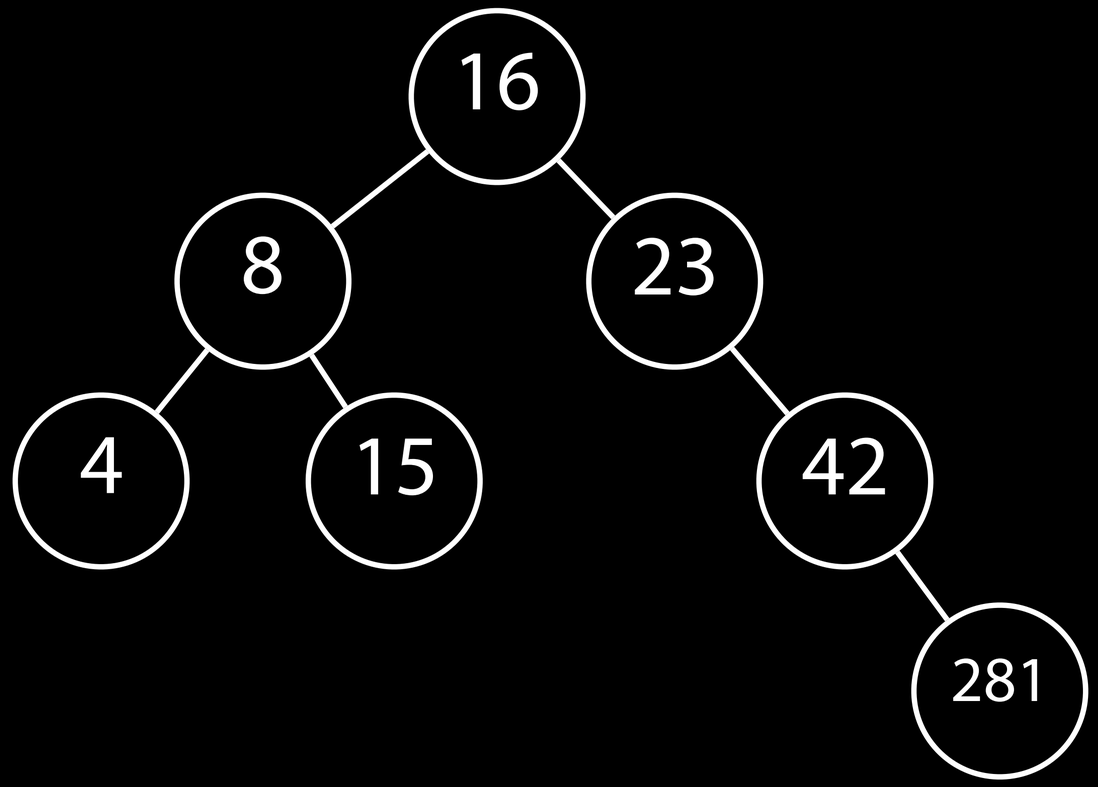


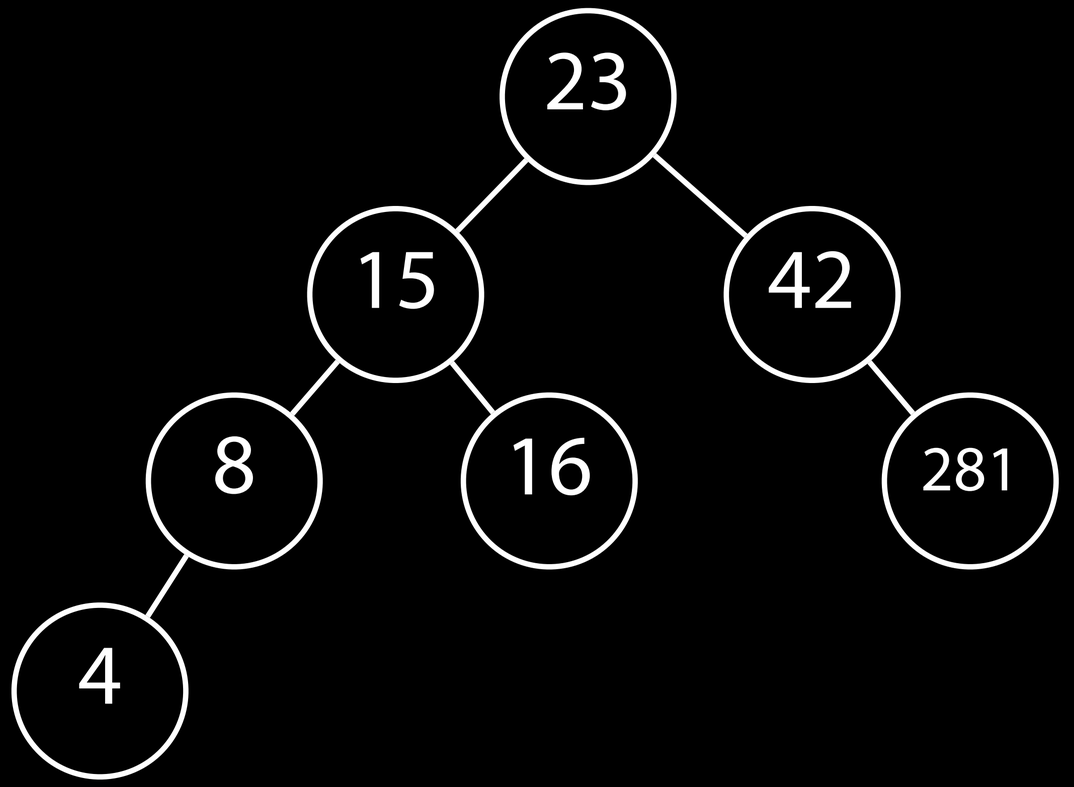
AVL Trees

Binary search tree. For any node, heights of the two child subtrees differ by at most 1.

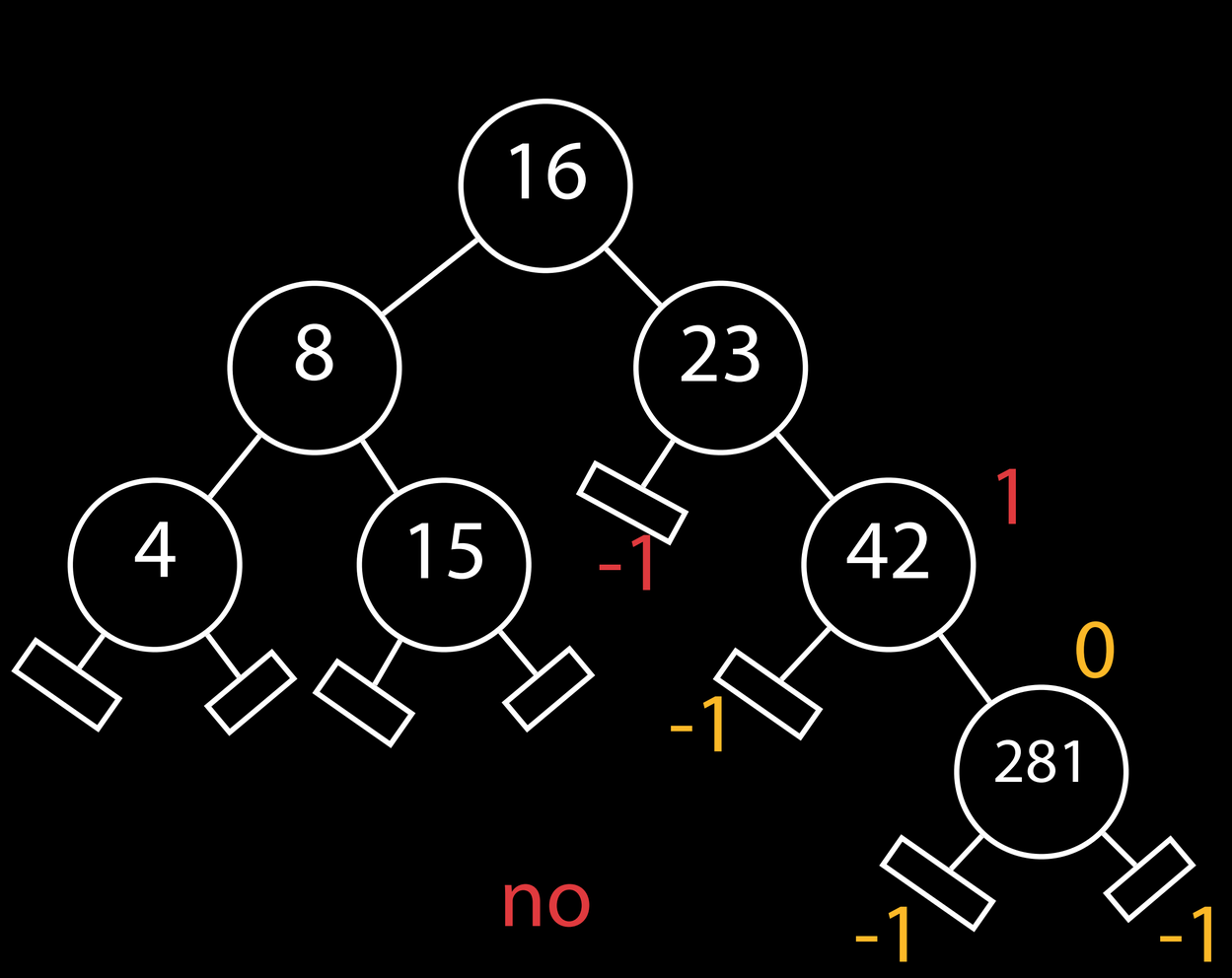


AVL Trees?

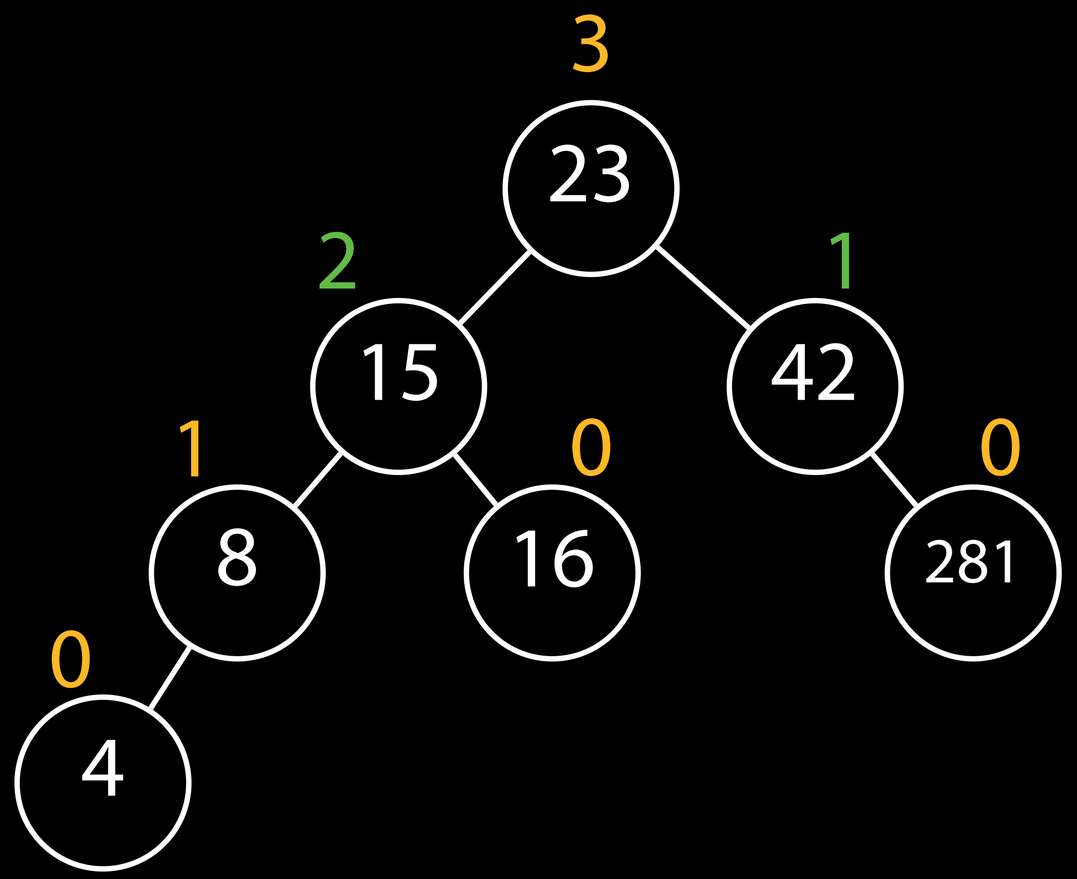




AVL Trees?



no



yes

AVL insert

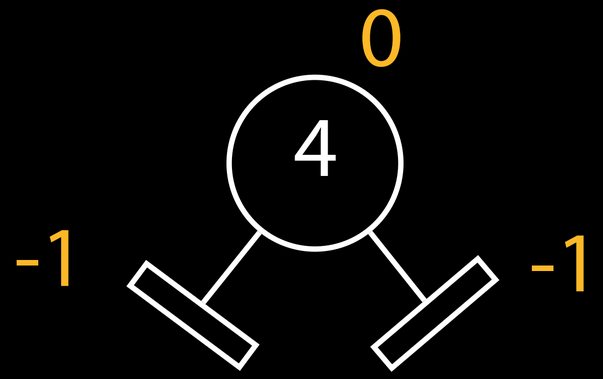
Insert ~~4, 8, 15, 16, 23 and 42~~ 4, 8, 15, 23, 16 and 42.

Insertion

Insert 4, 8, 15, 23, 16 and 42.

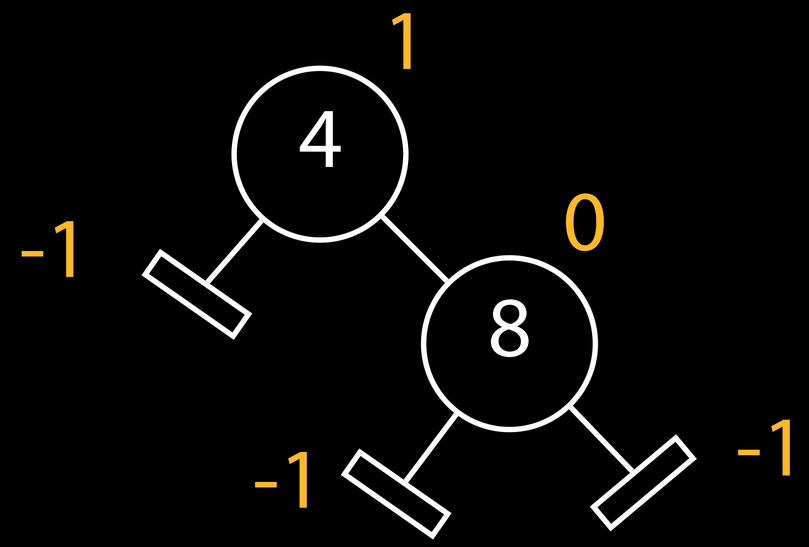
Insertion

Insert 4, 8, 15, 23, 16 and 42.



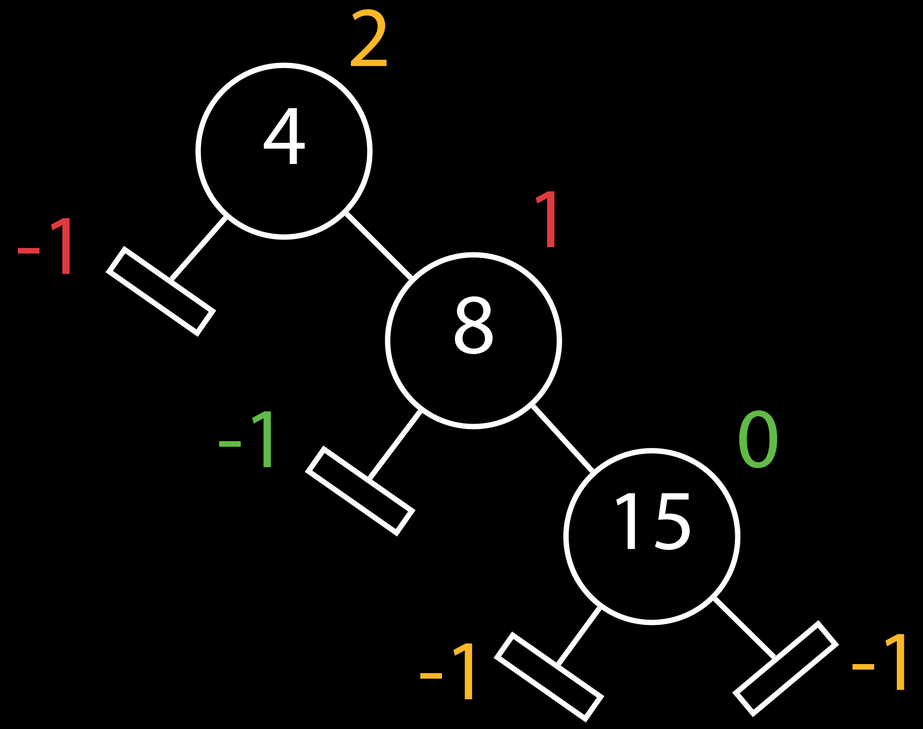
Insertion

Insert 4, 8, 15, 23, 16 and 42.



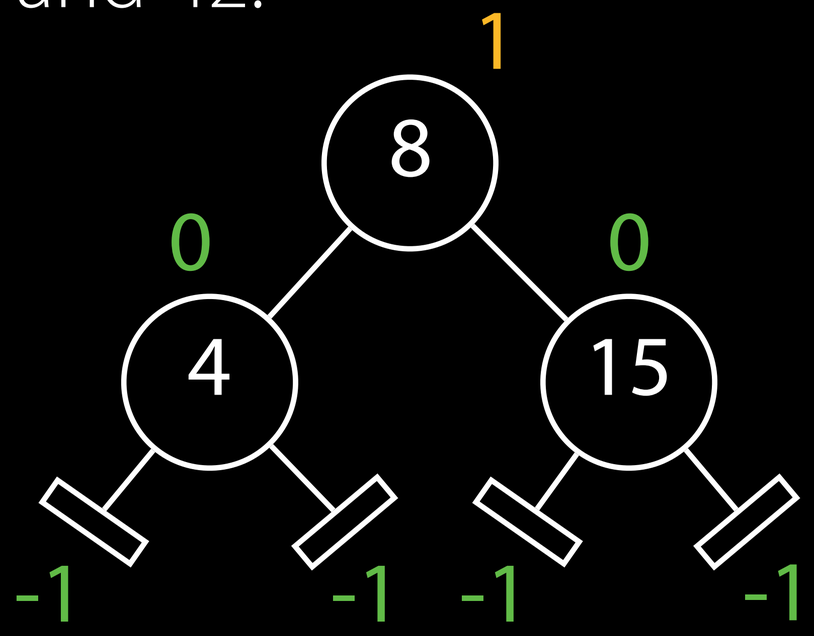
Insertion

Insert 4, 8, 15, 23, 16 and 42.



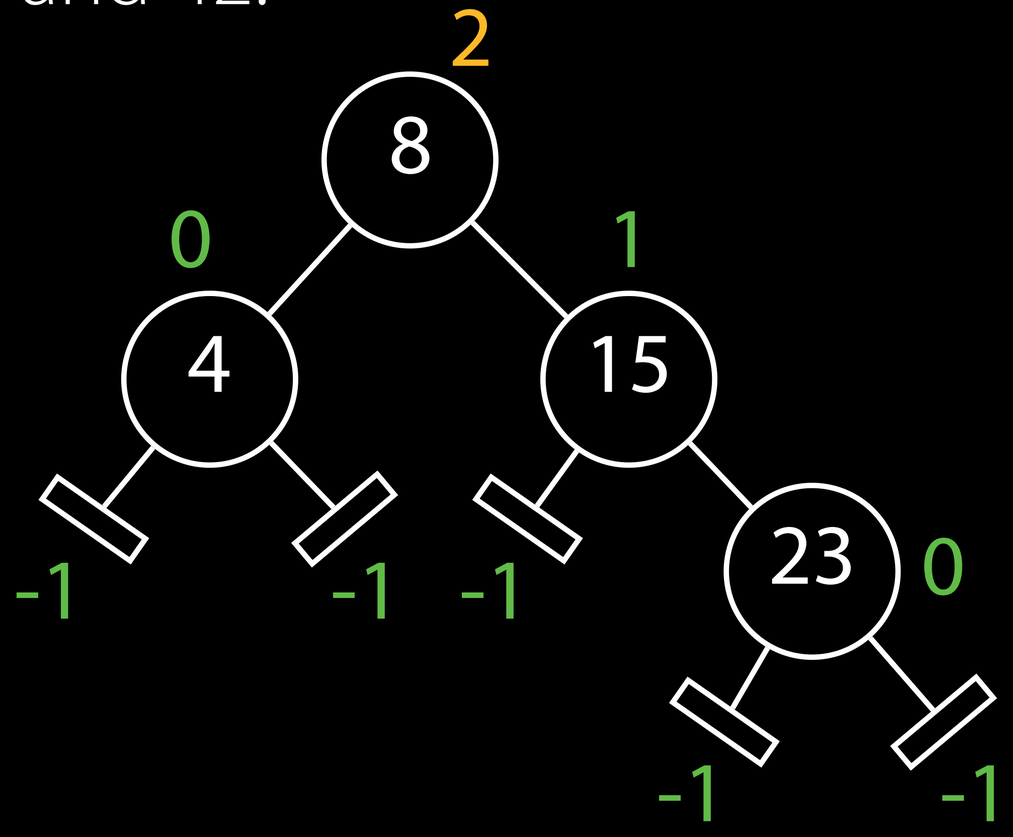
Insertion

Insert 4, 8, 15, 23, 16 and 42.



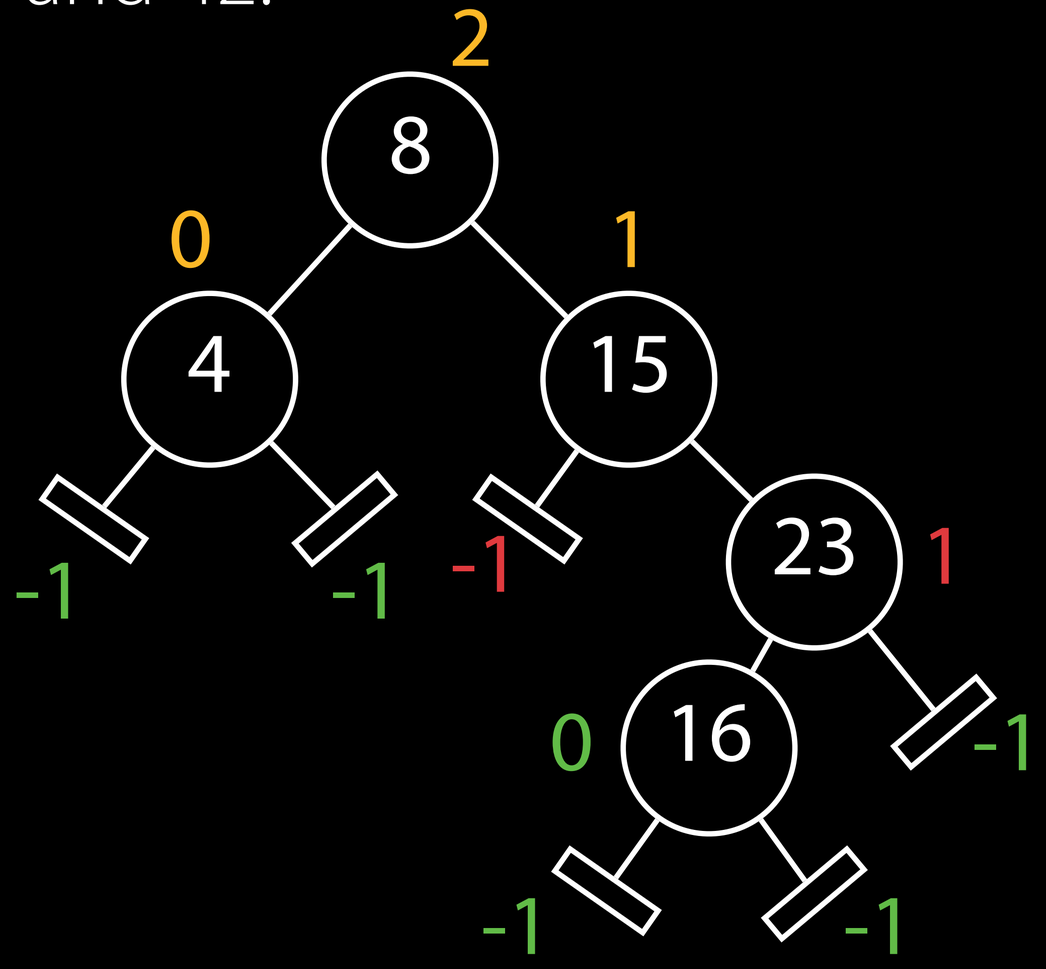
Insertion

Insert 4, 8, 15, 23, 16 and 42.



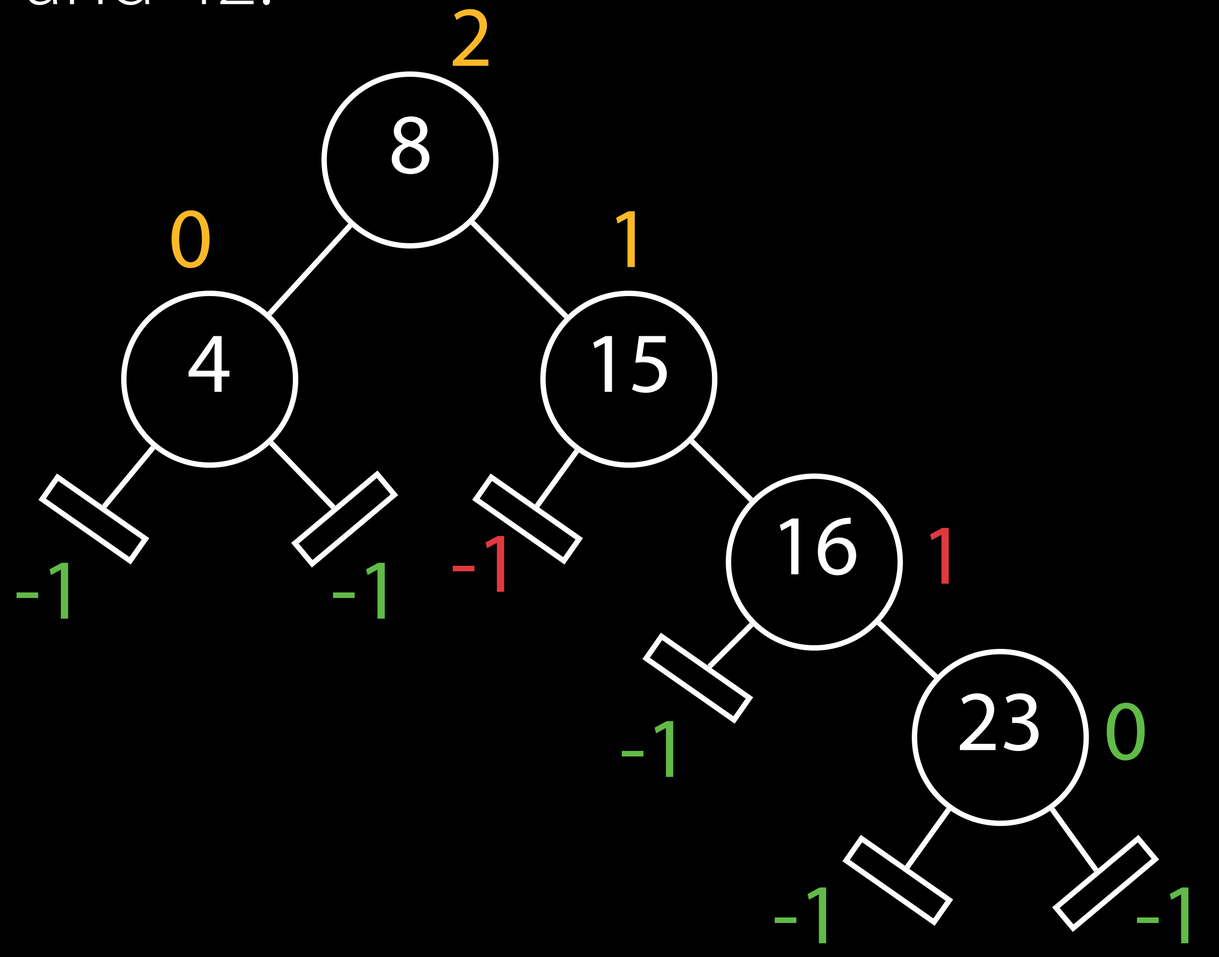
Insertion

Insert 4, 8, 15, 23, 16 and 42.



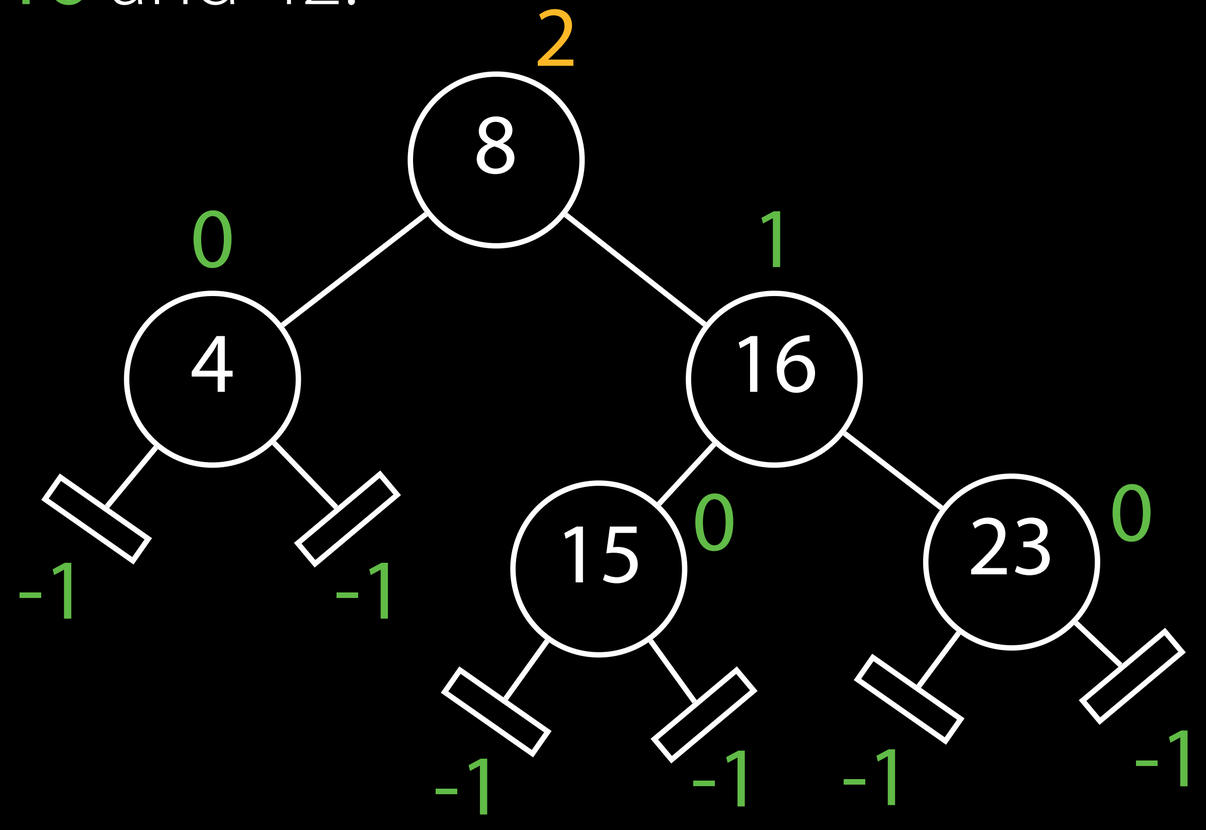
Insertion

Insert 4, 8, 15, 23, 16 and 42.



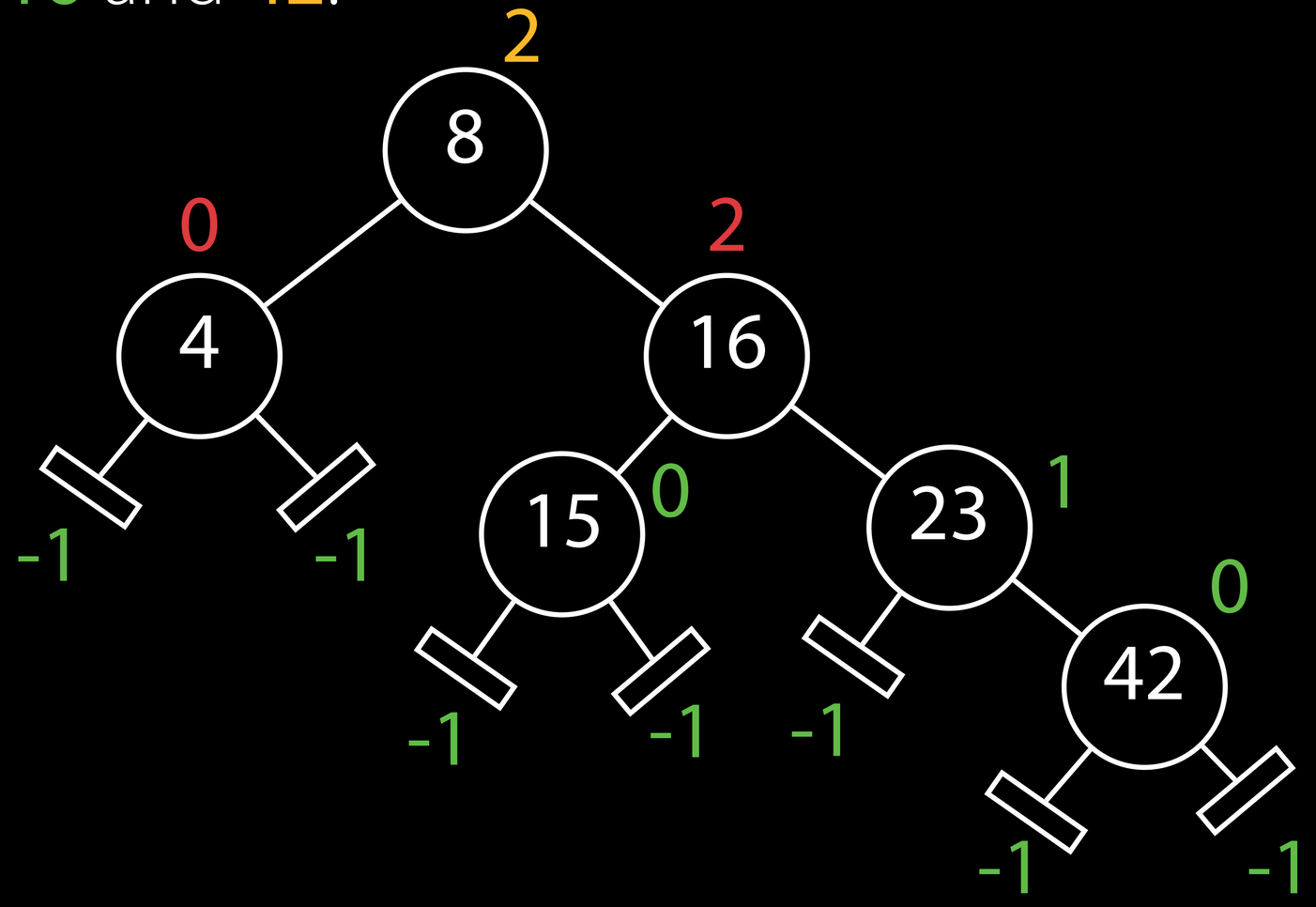
Insertion

Insert 4, 8, 15, 23, 16 and 42.



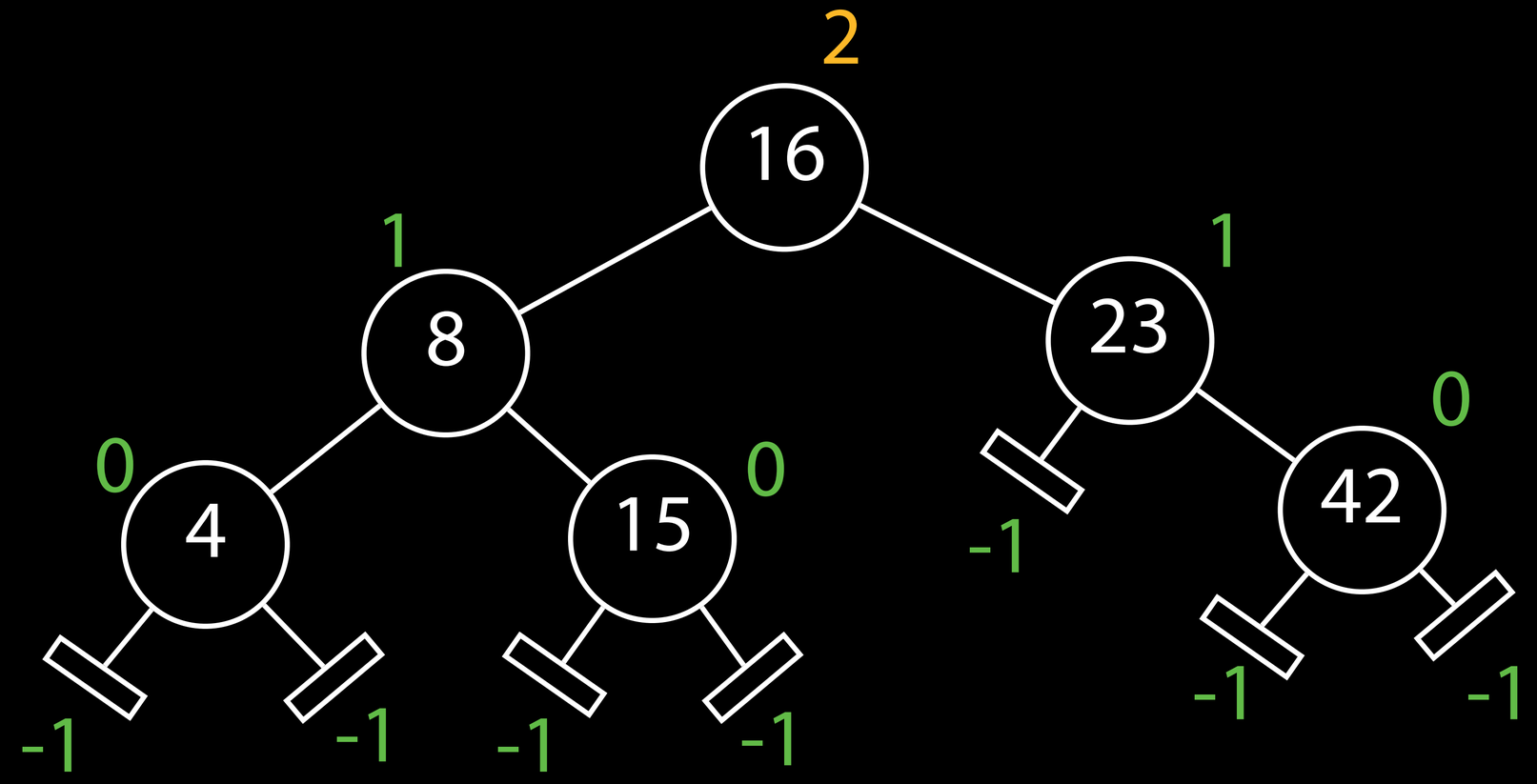
Insertion

Insert 4, 8, 15, 23, 16 and 42.



Insertion

Insert 4, 8, 15, 23, 16 and 42.



AVL tree performance

Searching: $O(\log N)$

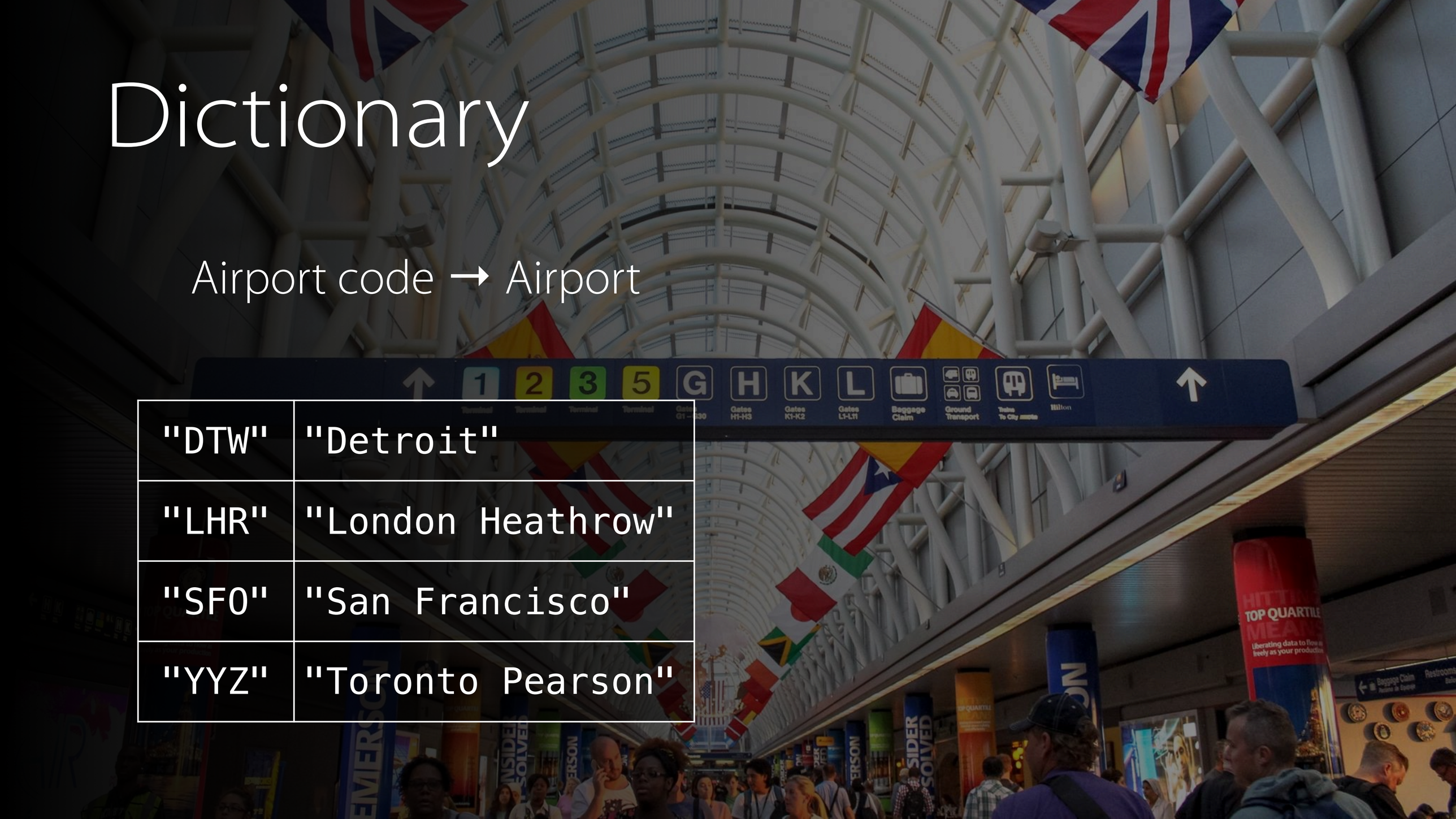
Insertion: $O(\log N)$

Deletion: $O(\log N)$

The path from the root to the deepest leaf in an AVL tree is at most $\sim 1.44 \log(N + 2)$

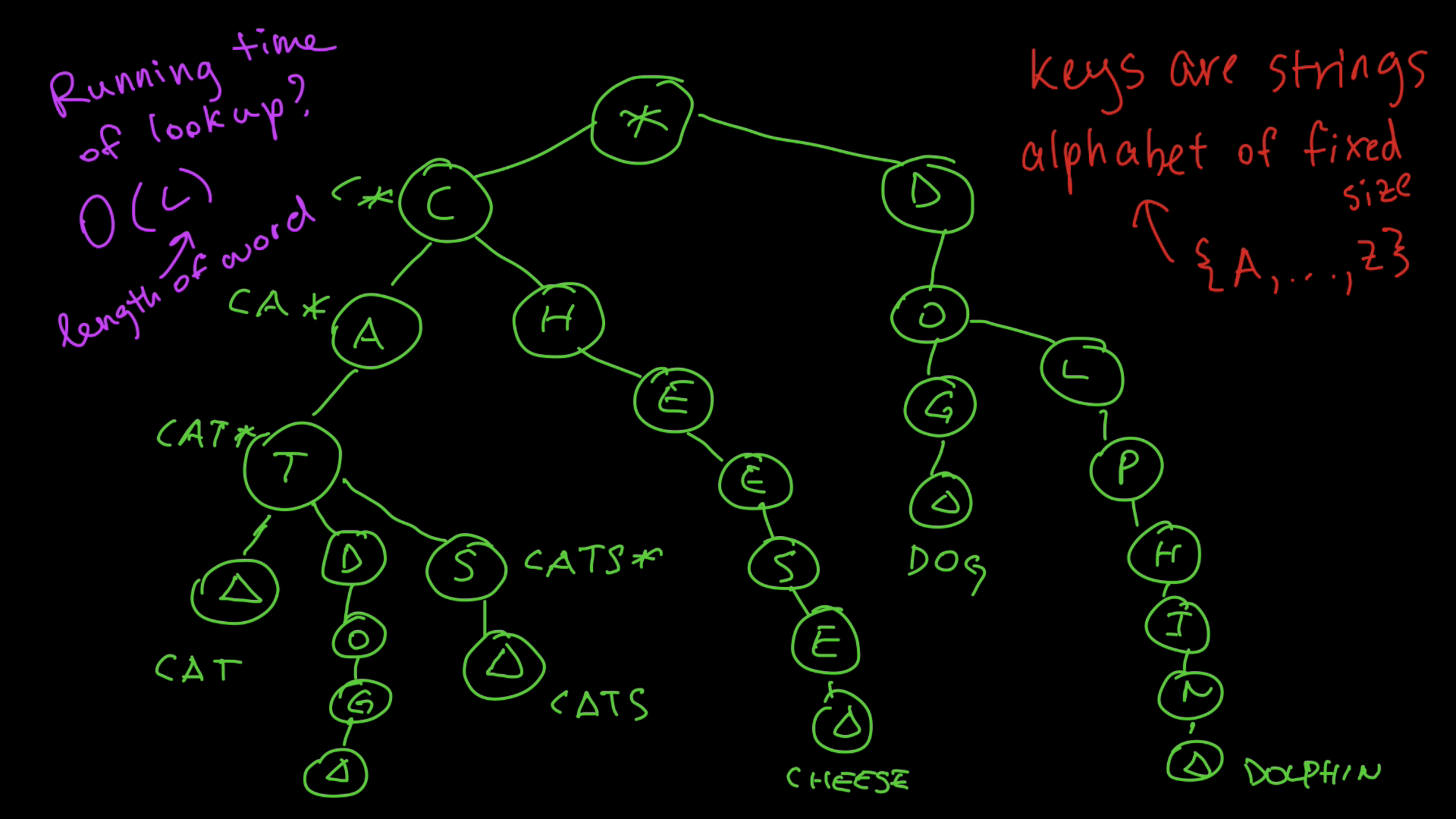
Dictionary

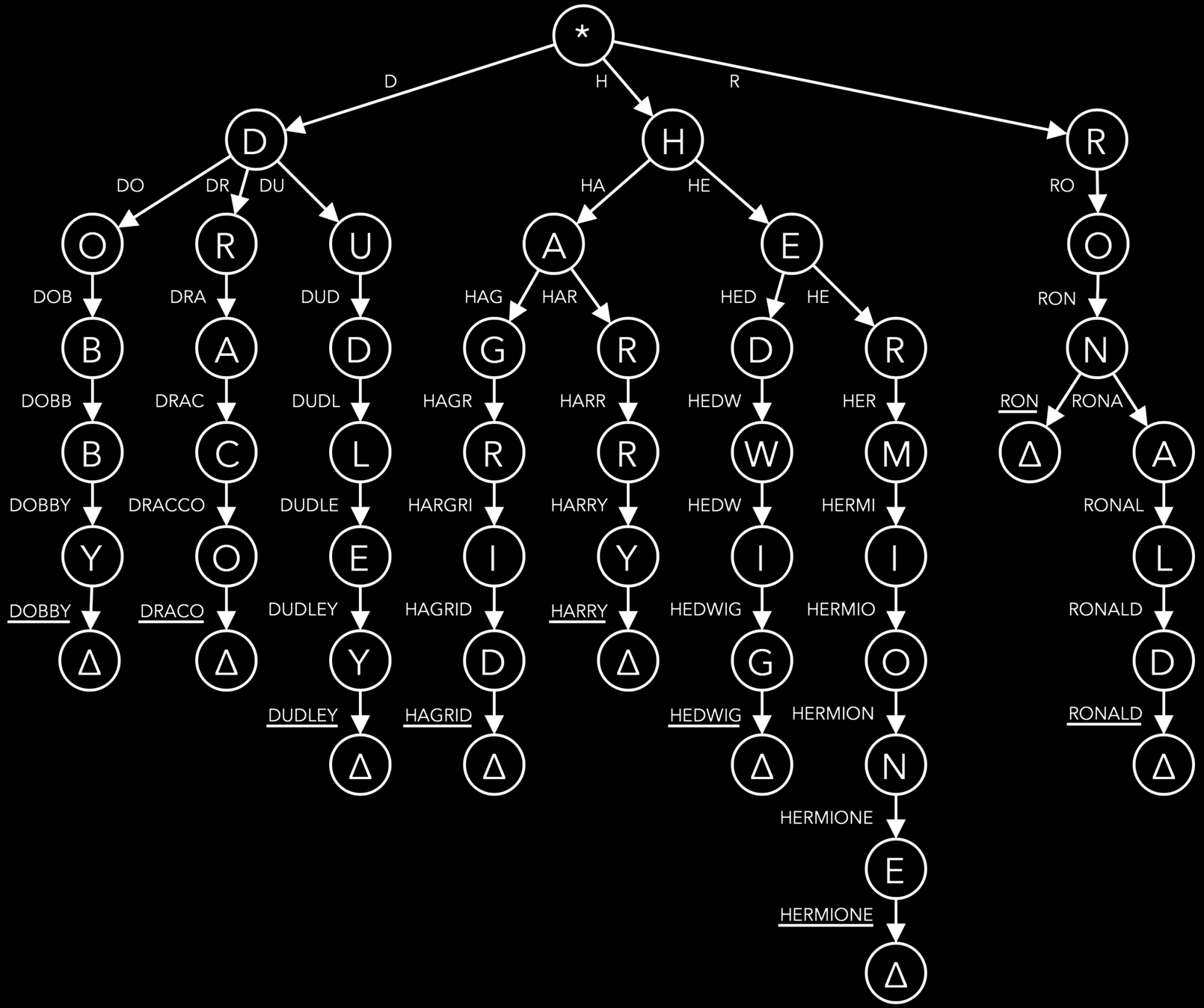
Airport code → Airport

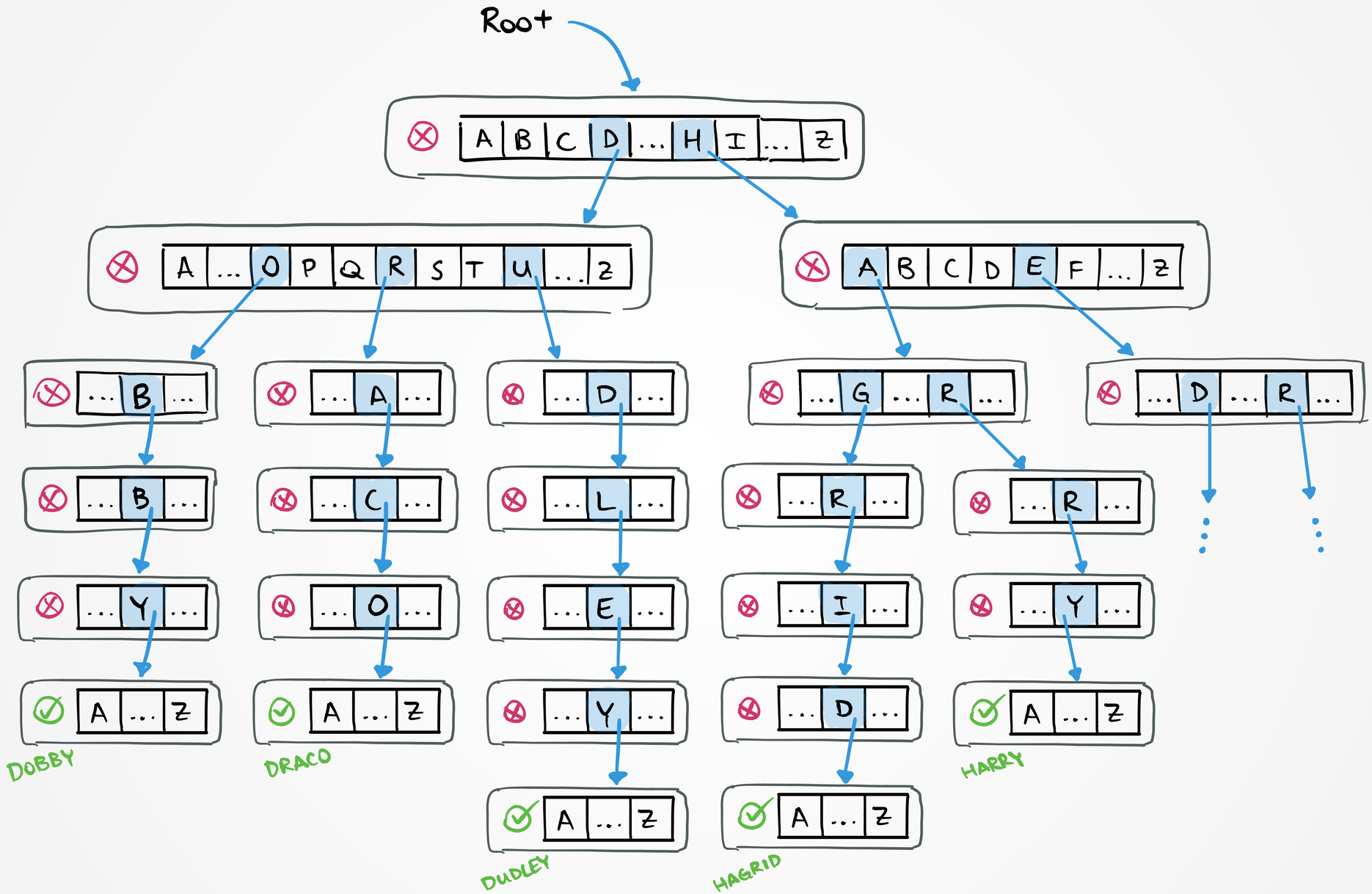


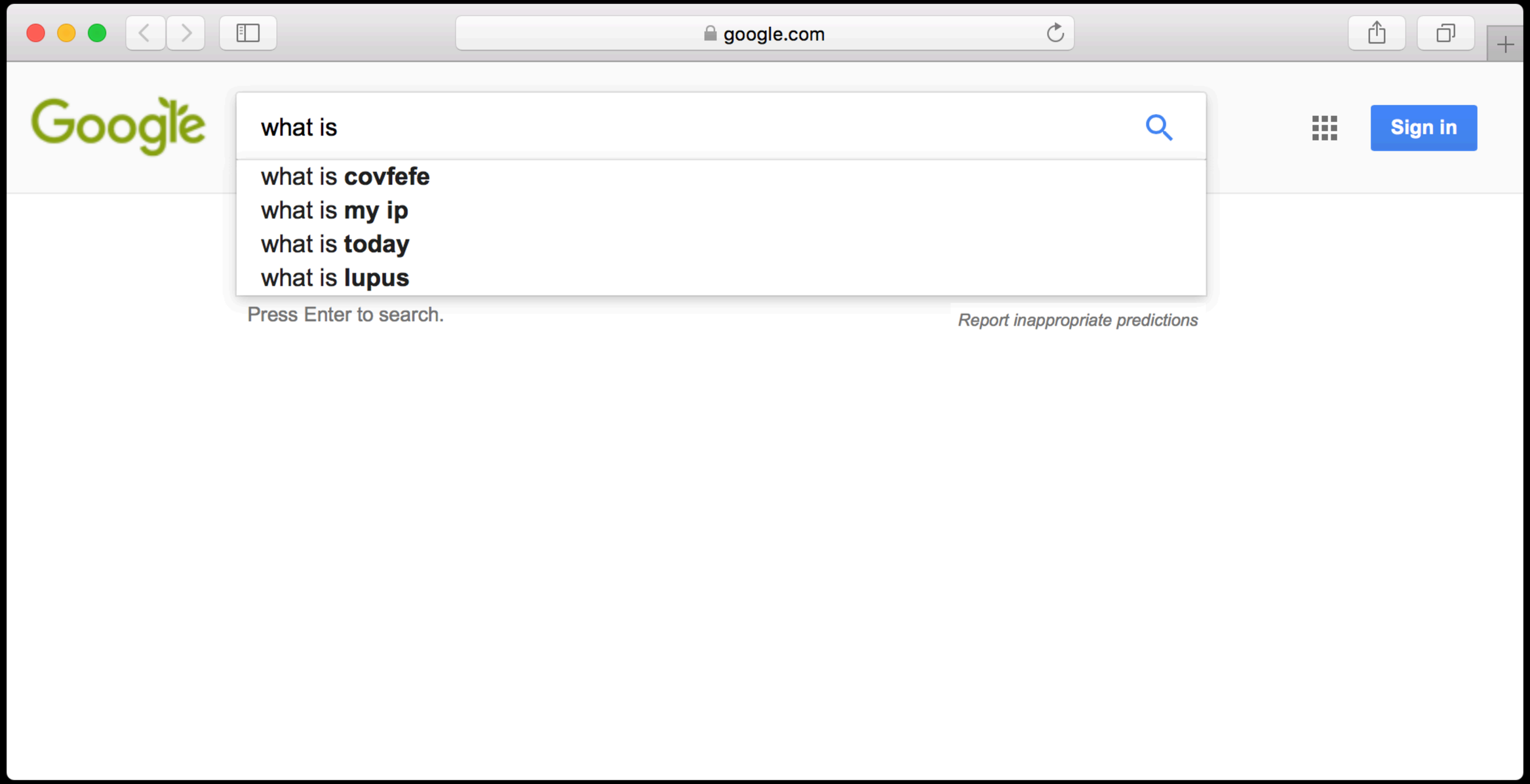
`"DTW"`	`"Detroit"`
`"LHR"`	`"London Heathrow"`
`"SFO"`	`"San Francisco"`
`"YYZ"`	`"Toronto Pearson"`

Trie


Running time of lookup?
O(L)
length of word
keys are strings
alphabet of fixed size
{A, ..., Z}
*
C*
CA*
CAT*
CATS*
DOG
DOLPHIN
CAT
CATS
CHEESE


*
D
H
R
D
H
R
DO
DR
DU
HA
HE
RO
O
R
U
A
E
O
DOB
DRA
DUD
HAG
HAR
HED
HE
RON
B
A
D
G
R
D
R
N
DOBB
DRAC
DUDL
HAGR
HARR
HEDW
HER
RON
RONA
B
C
L
R
R
W
M
Δ
A
DOBBY
DRACCO
DUDLE
HARGRI
HARRY
HEDW
HERMI
RONAL
Y
O
E
I
Y
I
I
L
DOBBY
DRACO
DUDLEY
HAGRID
HARRY
HEDWIG
HERMIO
RONALD
Δ
Δ
Y
D
Δ
G
O
D
DUDLEY
HAGRID
HEDWIG
HERMION
RONALD
Δ
Δ
Δ
N
Δ
HERMIONE
E
HERMIONE
Δ

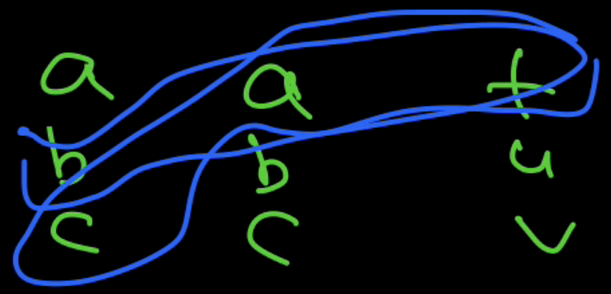

Root
A B C D ... H I ... Z
A ... O P Q R S T U ... Z
A B C D E F ... Z
... B ...
... A ...
... D ...
... G ... R ...
... D ... R ...
... B ...
... C ...
... L ...
... R ...
... R ...
... Y ...
... O ...
... E ...
... I ...
... Y ...
A ... Z
A ... Z
... Y ...
... D ...
A ... Z
A ... Z
DOBBY
DRACO
DUDLEY
HAGRID
HARRY


google.com
Google
what is
what is covfefe
what is my ip
what is today
what is lupus
Sign in
Press Enter to search.
Report inappropriate predictions

T9 Texting

CAT

2 2 8



Predictive Text Error Leads to Fatal Stabbing

ANDY CHALK | 10 FEBRUARY 2011 9:36 PM

149

A U.K. man has been convicted of manslaughter for stabbing his friend to death after a predictive text error ballooned into an argument and ultimately a vicious knife attack.

33-year-old Neil Brook and his neighbor, 27-year-old Josef Witkowski, had known each other for about six months before getting into a beef over a text message misunderstanding. Brook told police he'd sent Witkowski a text message asking "What are you on about mutter?", "mutter" being "a local colloquialism for a person who behaves in an antisocial or vulgar manner." But thanks to the predictive text on his phone, "mutter" was corrected to "nutter," a slang term meaning "deranged."

http://www.escapistmagazine.com/news/view/107702-Predictive-Text-Error-Leads-to-Fatal-Stabbing

Lab 7 assignment

Finish implementing a hash table

cantTellYa

conSealed

confidential

donutTry2Guess

hashCoded

noComment

shhhh

topSecret

underWraps

wow

Lab 8 assignment

Implement a trie

Tree Diameter

Q&A