Data Structures

Trees and Tries

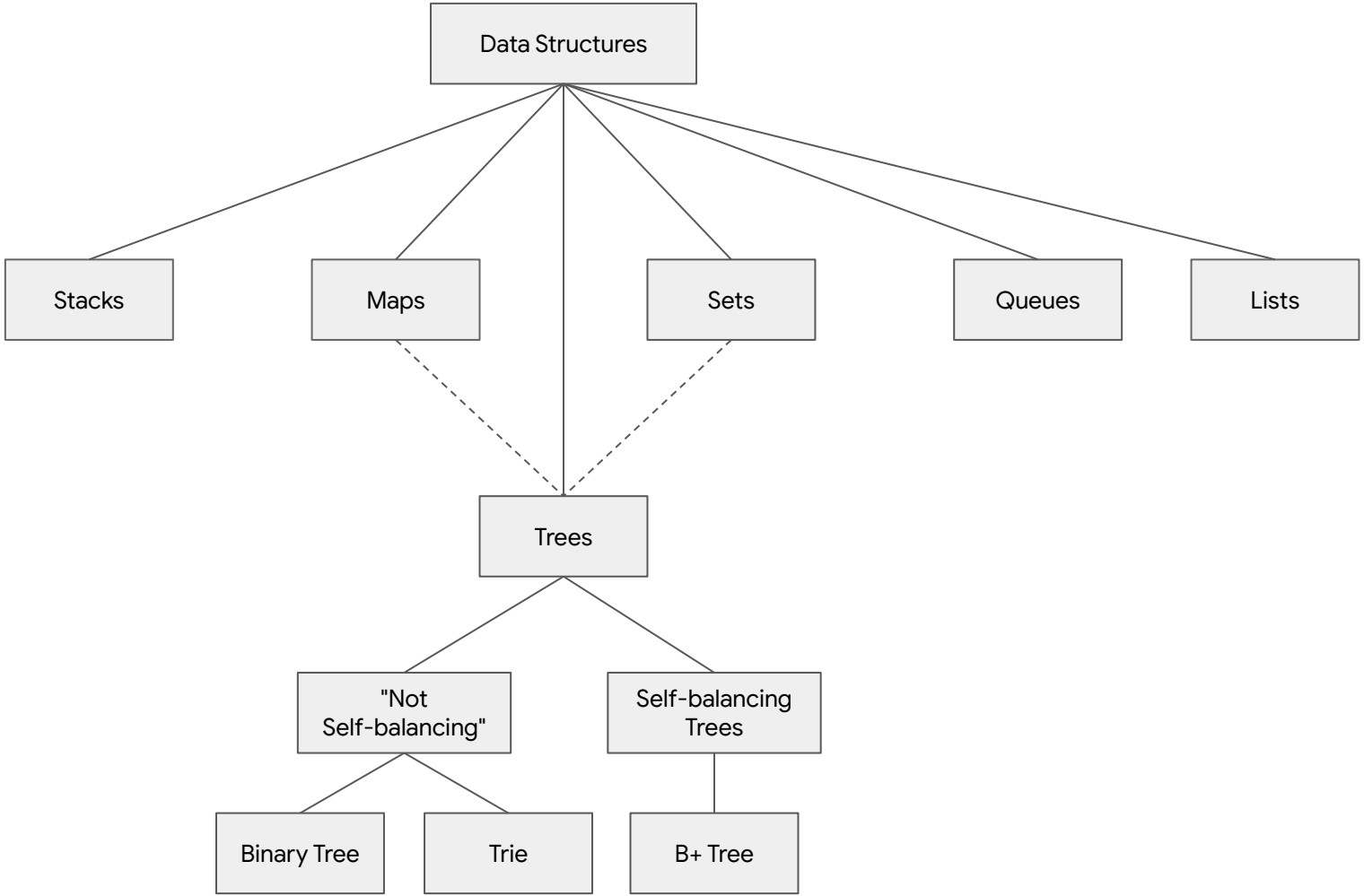
Objectives

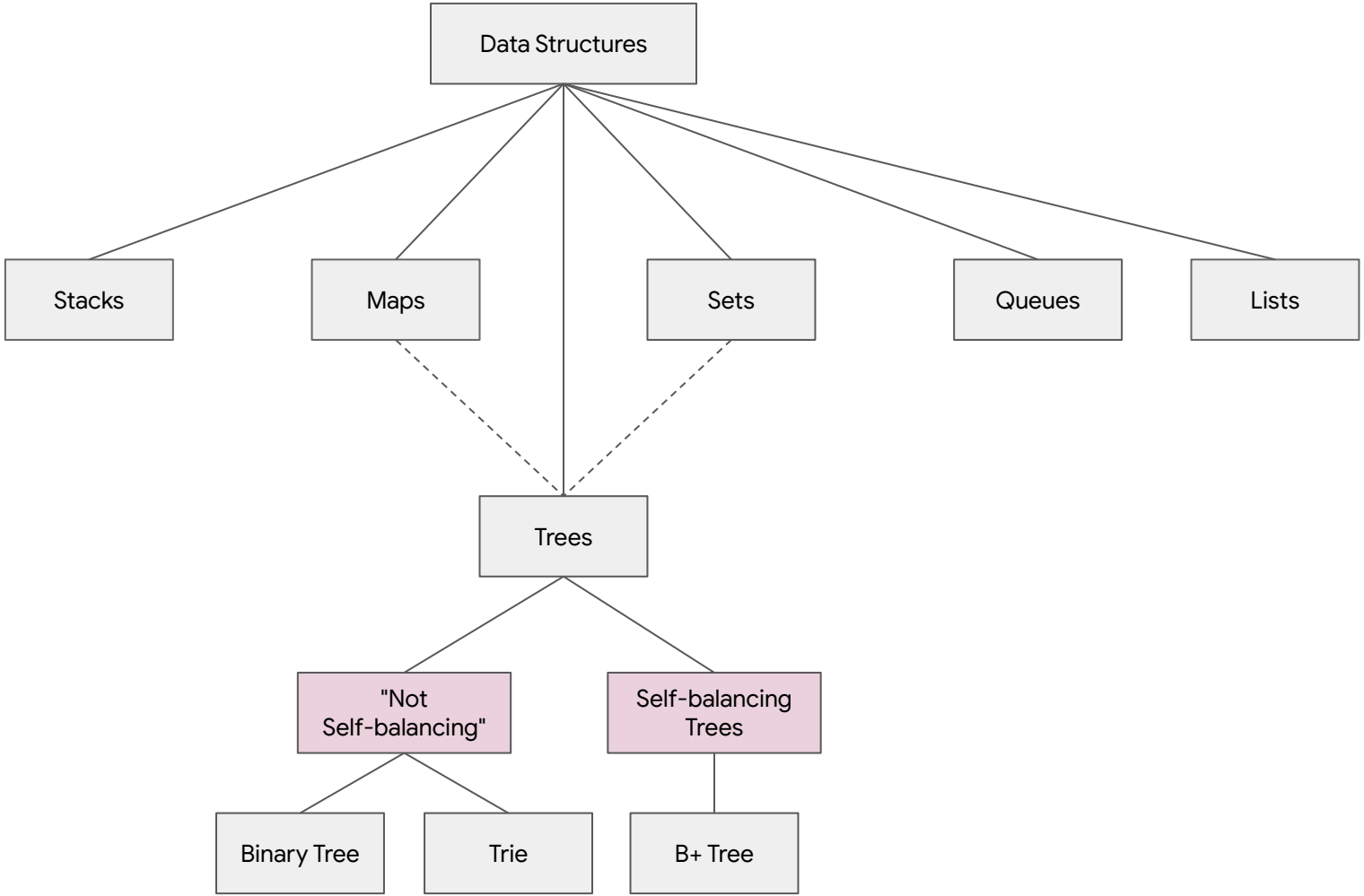
Primary Objectives

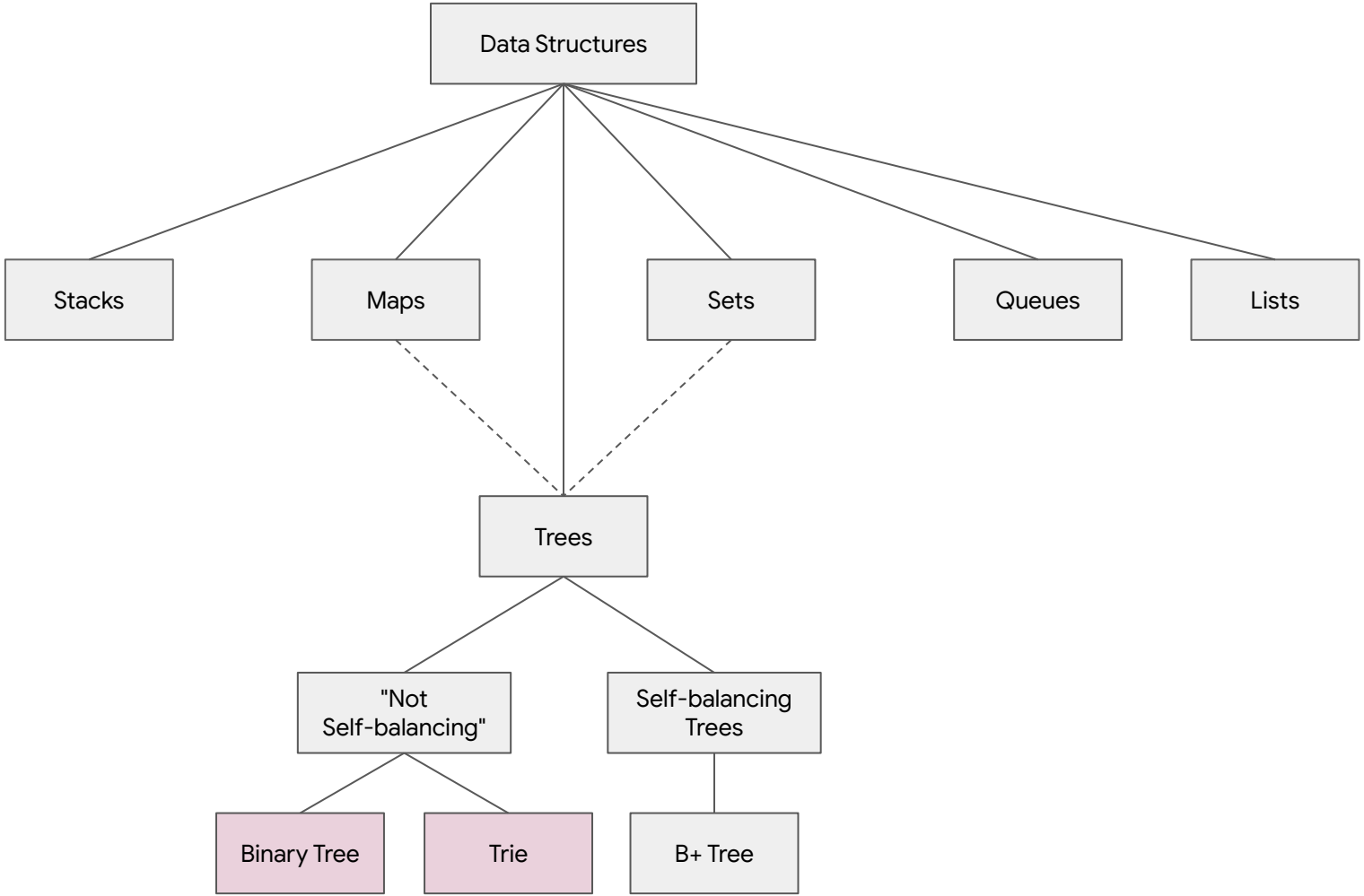
- What is a **tree** and when could I use it?
- What is a **trie** and when could I use it?

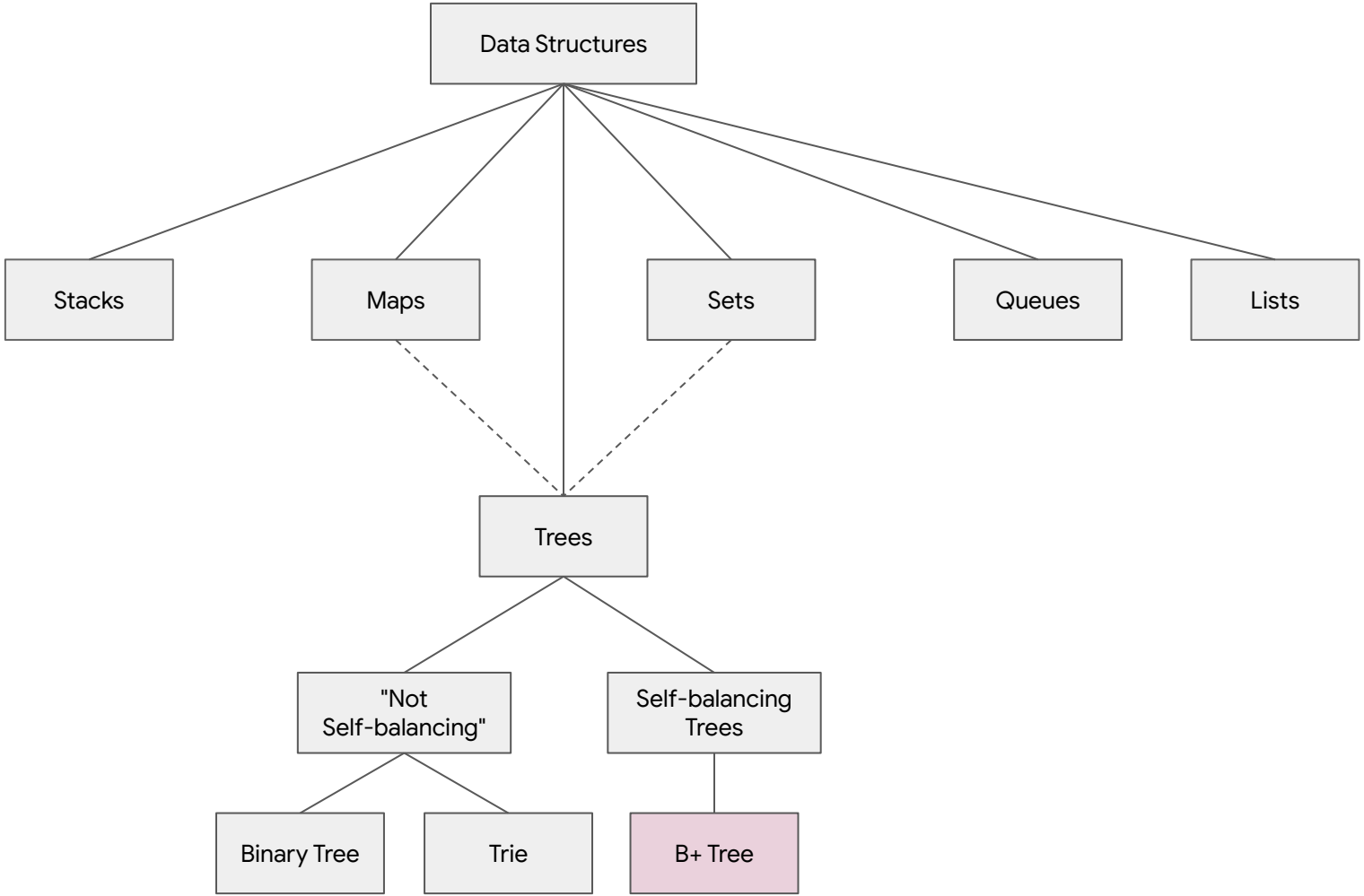
Secondary Objectives

- What trade-offs am I making when I use a **tree**?
- What trade-offs am I making when I use a **trie**?


Data Structures
Stacks
Maps
Sets
Queues
Lists
Trees
"Not Self-balancing"
Self-balancing Trees
Binary Tree
Trie
B+ Tree


Data Structures
Stacks
Maps
Sets
Queues
Lists
Trees
"Not Self-balancing"
Self-balancing Trees
Binary Tree
Trie
B+ Tree

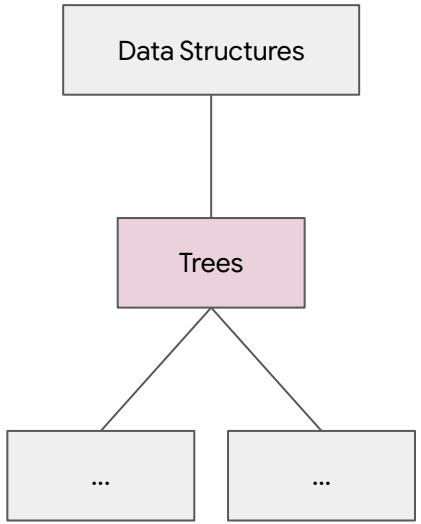

Data Structures
Stacks
Maps
Sets
Queues
Lists
Trees
"Not Self-balancing"
Self-balancing Trees
Binary Tree
Trie
B+ Tree


Data Structures
Stacks
Maps
Sets
Queues
Lists
Trees
"Not Self-balancing"
Self-balancing Trees
Binary Tree
Trie
B+ Tree

Trees

Characteristics

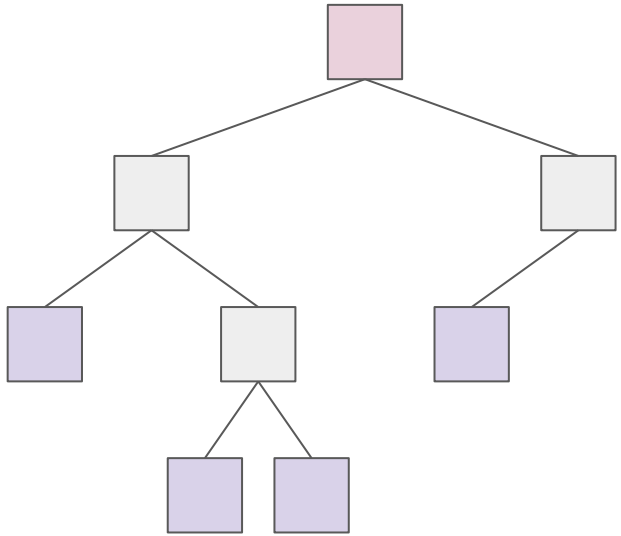
- Ordered Data
- Linked-structure
- Single starting point, many ending points
- Linear flow



Binary Tree

Characteristics

- Every node has 0, 1, or 2 children.
- Self-balancing and non-self-balancing version.



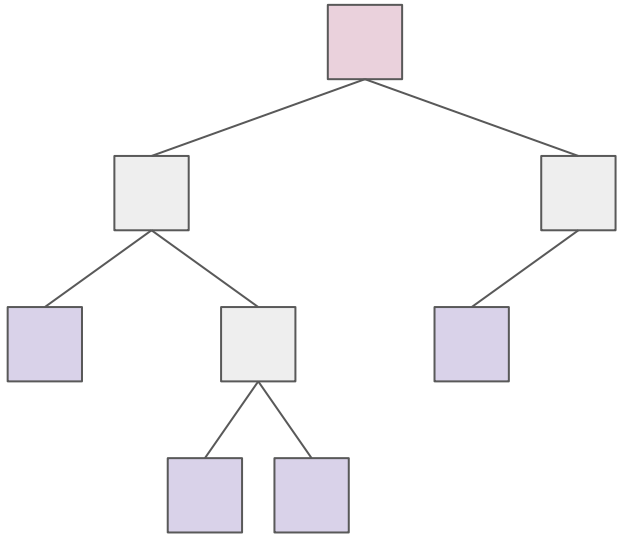
N-ary Tree

Characteristics

- Every node has 0, 1, 2, ... N children.
- Self-balancing and non-self-balancing version.

Note

- A binary tree is a N-ary Tree where N = 2.



Classifications

Full

Complete

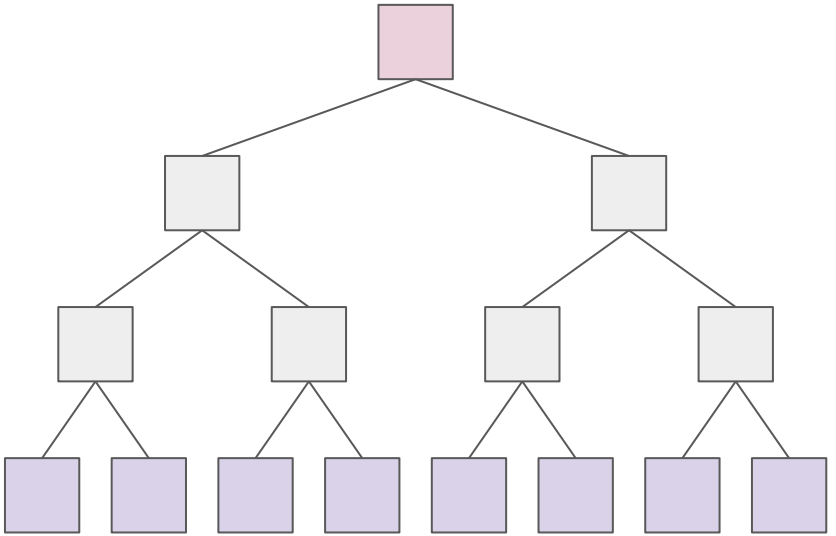
Balanced

Classifications

Full

Complete

Balanced

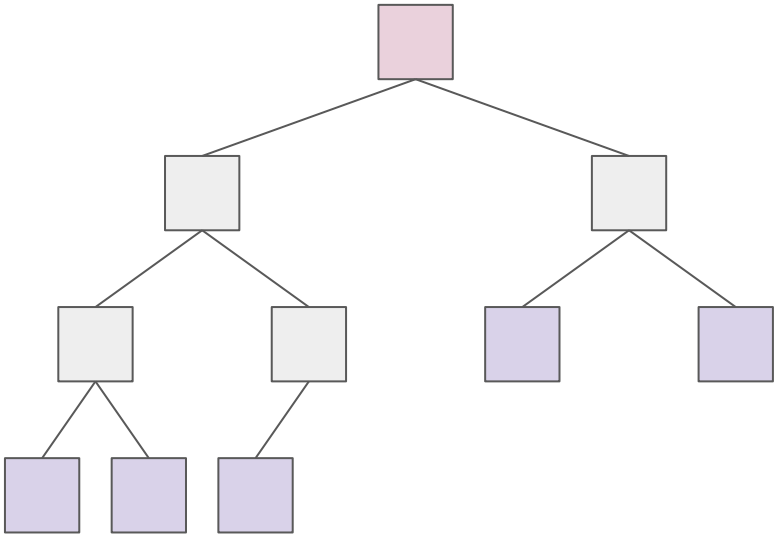


Classifications

Full

Complete

Balanced

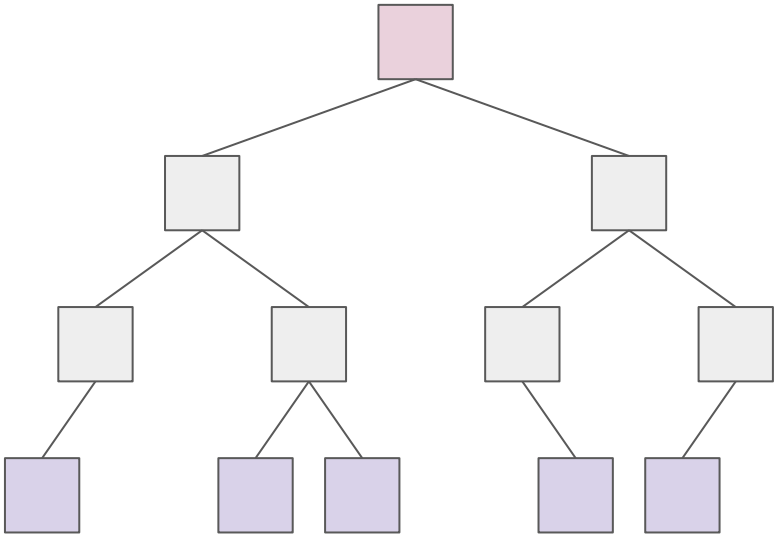


Classifications

Full

Complete

Balanced

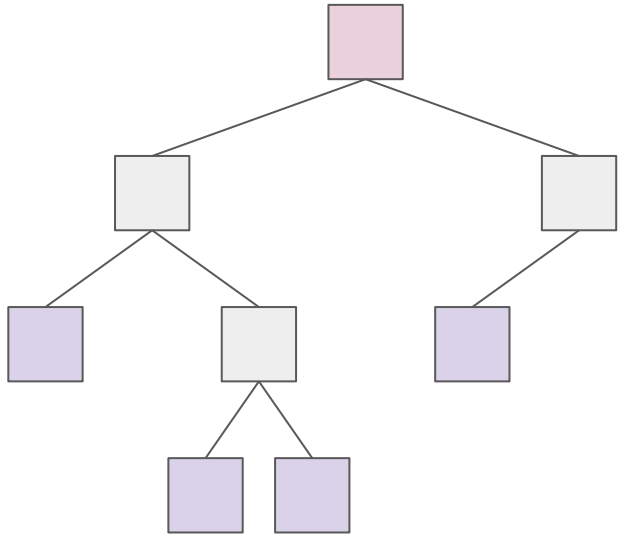


Classifications

Full

Complete

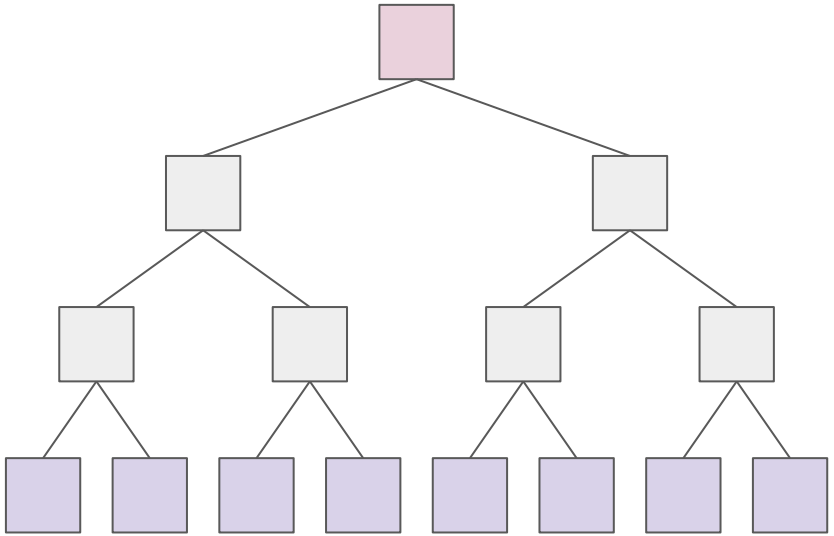
Balanced



Traversal

Traversal

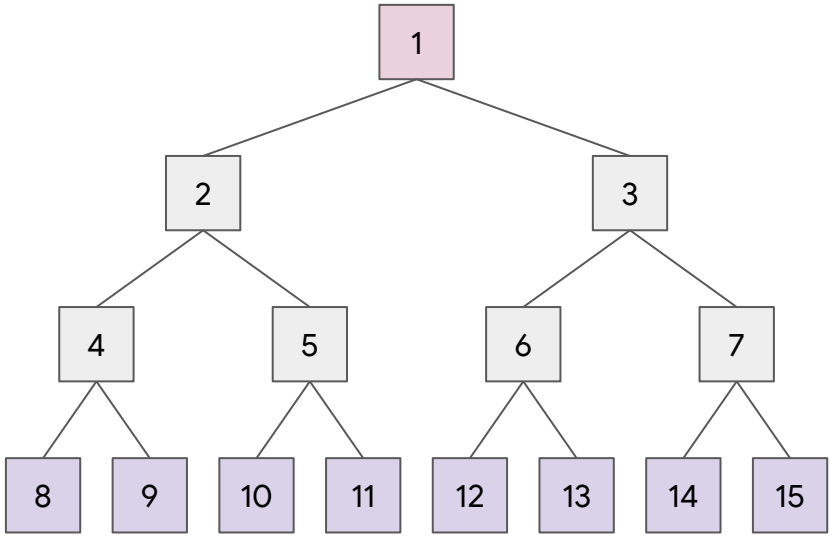
Defines how we move through a tree.



Traversal

Breadth-First

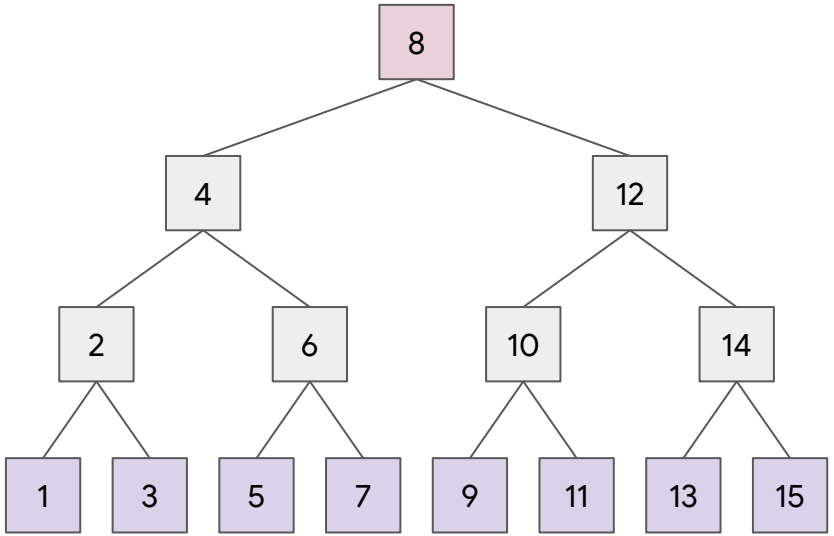
Visit nodes in a tree from "left to right".



Traversal

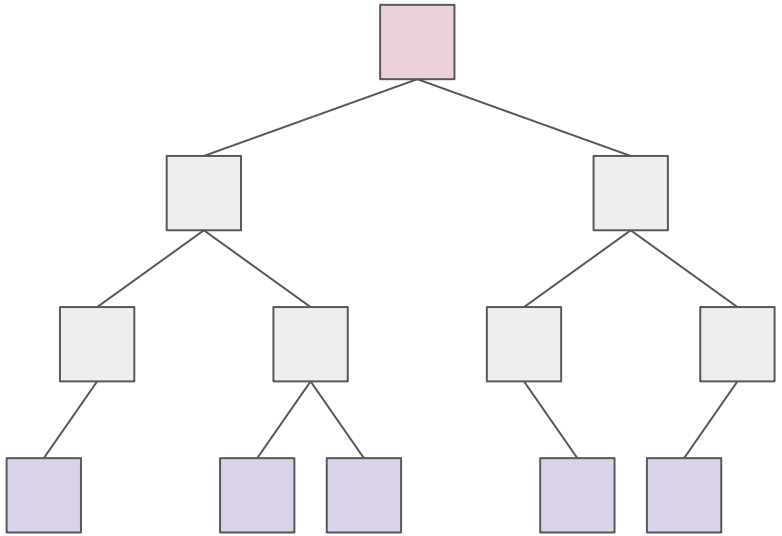
Depth-First

Visit nodes recursively following the pattern "left", "middle", then "right".

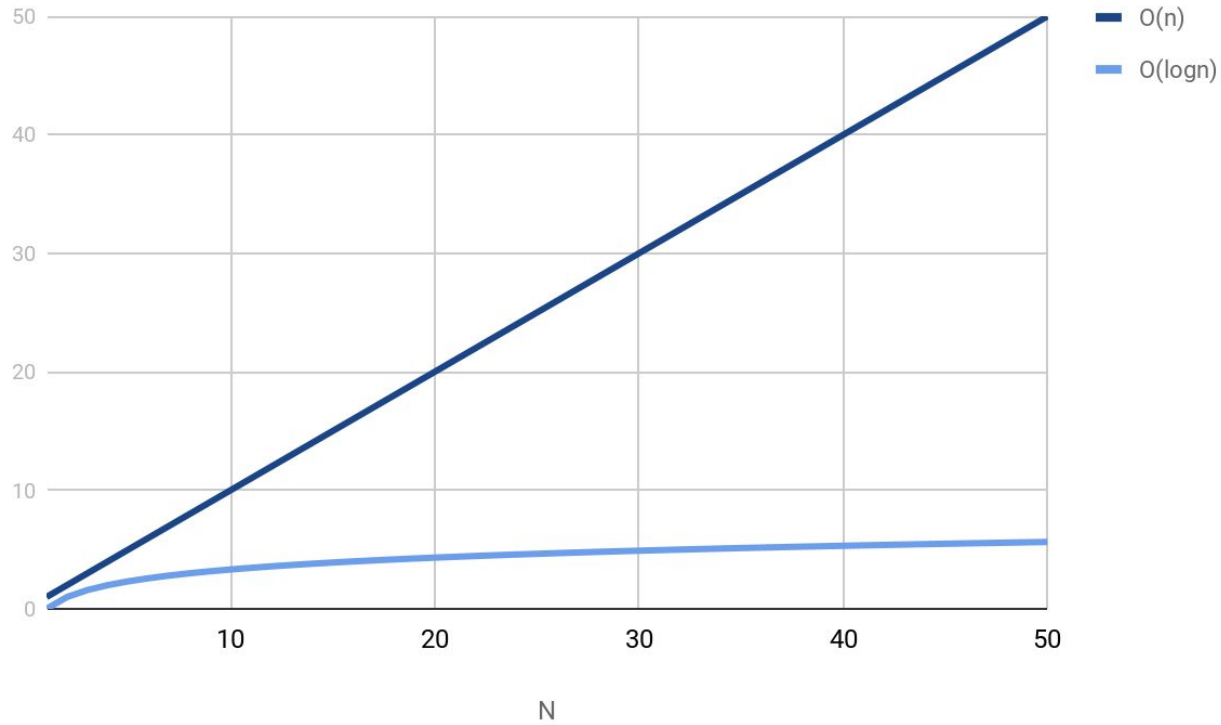


Balancing

Why keep a tree balanced?



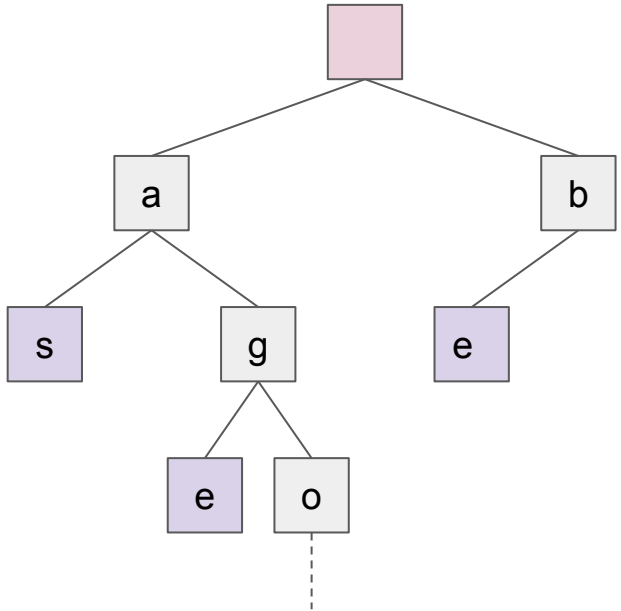
Balancing



Tries

Characteristics

- Every node in a path represents one item in a series.
- Series that start the same, share paths in the tree.



Tries

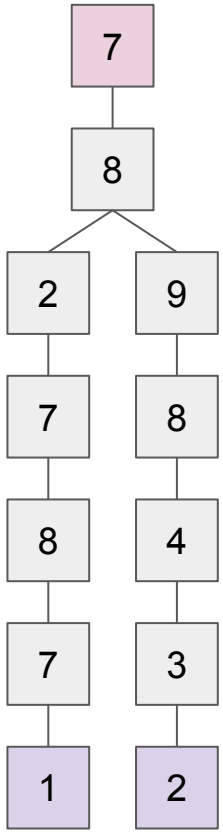
Suppose you needed a trie to store phone numbers.

Tries : Add

```
Trie t = new Trie();

t.add("782-7871");
```

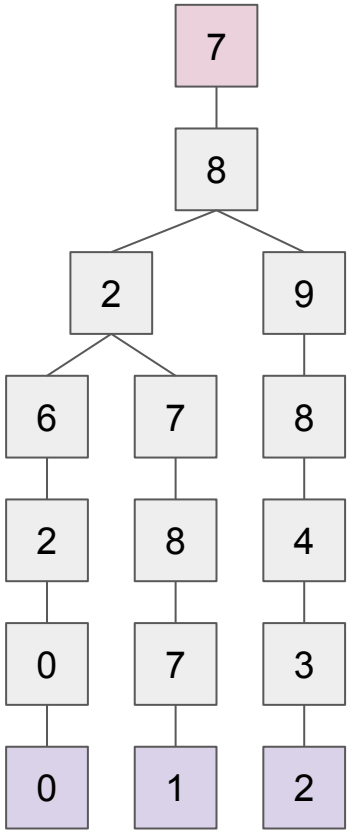
Tries : Add

```
Trie t = new Trie();

t.add("782-7871");
t.add("789-8432");
```



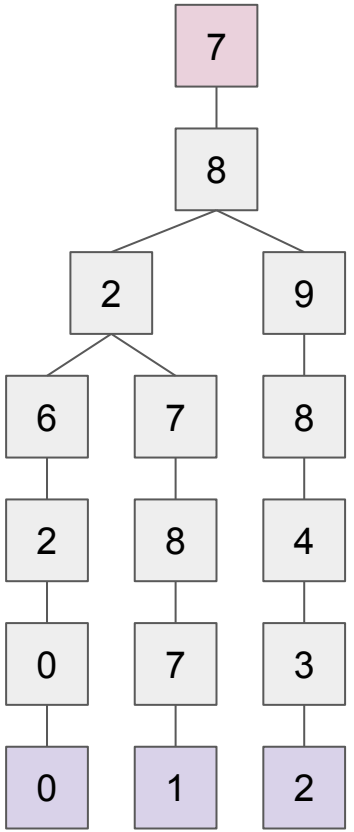
Tries : Add

```
Trie t = new Trie();

t.add("782-7871");
t.add("789-8432");
t.add("782-6200");
```



Tries : Contains

```
Trie t = new Trie();

t.add("782-7871");
t.add("789-8432");
t.add("782-6200");

t.contains("782-6200"); // true
t.contains("782-6210"); // false
```



Reflection

Primary Objectives

- What is a **tree** and when could I use it?
- What is a **trie** and when could I use it?

Secondary Objectives

- What trade-offs am I making when I use a **tree**?
- What trade-offs am I making when I use a **trie**?

Extra Curricular : B+ Tree

Goal

- To see how combining elements of different data structures together create more powerful data structures.

Elements

- Borrows from **Binary Trees**
- Borrows from **Listed Lists**
- Borrows from **Array Lists**

Extra Curricular : B+ Tree