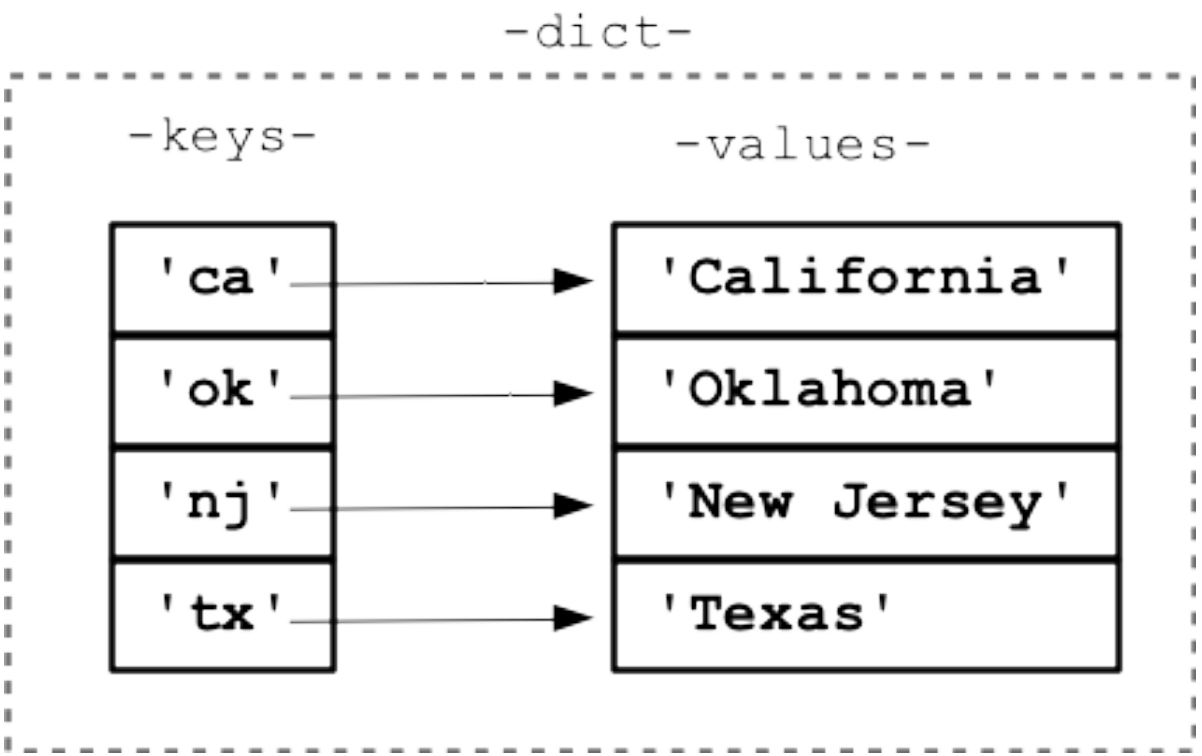
Day 4

Dictionaries

Another common data structure is a **dictionary**.

A dictionary associates **keys** with **values**.

Dictionaries



Creating a Dictionary

```
animals = {"big": "elephant", "small": "mouse"}
big_animal = animals["big"]
print(big_animal + " is big")
```

Reading From a Dictionary

```
animals = {"big": "elephant", "small": "mouse"}

list(animals)          # ["big", "small"]
"big" in animals       # True
"medium" in animals # False
animals["medium"]      # KeyError

for key, value in animals.items():
  print(key + " is " + value)
```

Adding to a Dictionary

```
animals = {"big": "elephant", "small": "mouse"}

"medium" in animals   # False
animals["medium"] = "gorilla"
"medium" in animals   # True
```

Deleting from a Dictionary

```
animals = {"big": "elephant", "small": "mouse"}

del animals["big"]
"big" in animals   # False
animals["big"]     # KeyError
```

Dictionaries

```
sounds = {"bird": "chirp", "cat": "meow", "cow": "moo"}

def get_longest_sound():
  longest_sound = ""
  for animal, sound in sounds.items():
    if len(sound) > len(longest_sound):
      longest_sound = sound
  return longest_sound

longest_sound(sounds)   # ?
```

Dictionaries

```
def get_ok_coffee(ratings):
  ok_coffee_places = {}
  for place, rating in ratings.items():
    if 3 < rating < 8:
      ok_coffee_places[place] = rating
  return ok_coffee_places

get_ok_coffee({
  "Starbucks": 5,
  "McDonald's": 1,
  "Dunkin' Donuts": 4,
  "Bongo Java": 8,
})    # What does this return?
```

Dictionaries inside Dictionaries

Values of dictionaries can be anything.

For representing complex data, it's helpful to have dictionaries be values.

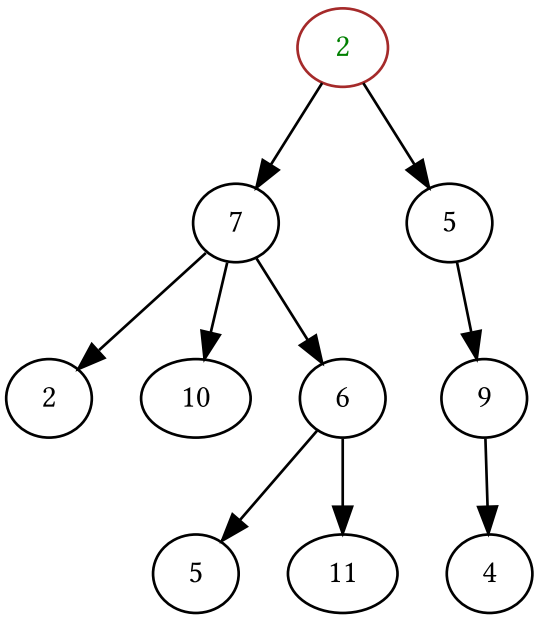
```
movies = {
   "best_comedy": {
      "title": "Baywatch",
      "rating": 4.9,
    },
    "best_romance": {
      "title": "Love Actually",
      "rating": 4.7,
    },
}
best_comedy_title = movies["best_comedy"]["title"]
```

Advanced Topic: Trees

The following topic is something that would be introduced in the last half of an introductory college Computer Science course.

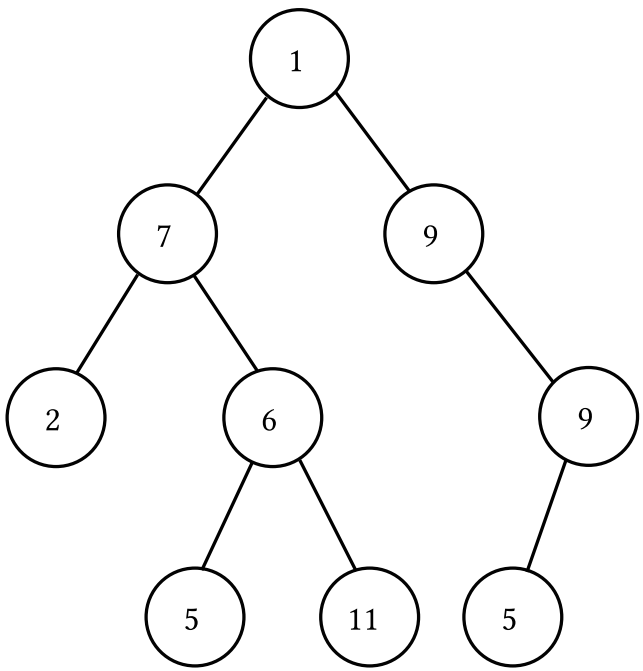
Creating Your Own Data Structure: Tree

A **tree** is a **data structure** that represents a hierarchy using **nodes** where each node has one or more **children**.



Binary Search Tree

A **binary search tree** is a kind of tree where each node has at most two children, named **left** and **right**.



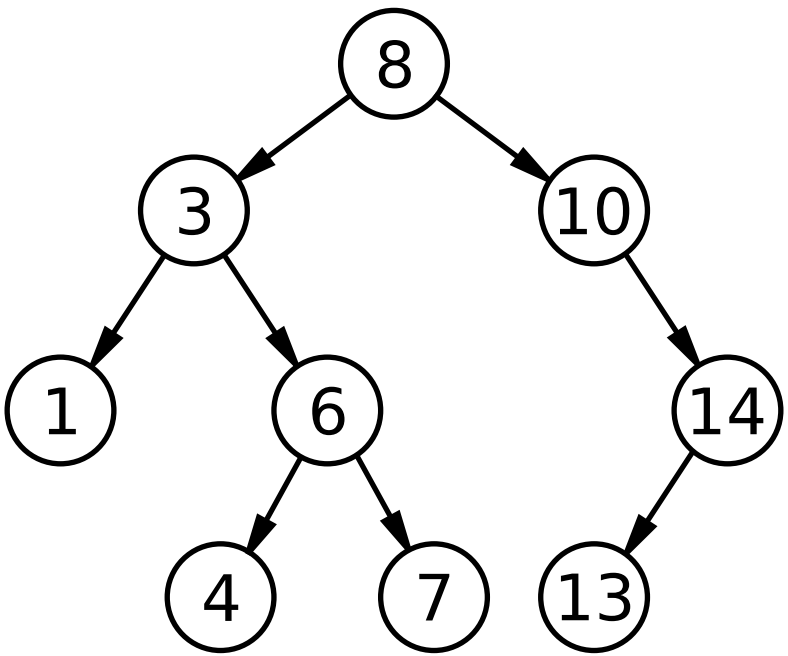
Binary Search Tree

Binary trees can be used to efficiently store numeric data.

It works like this:

- Each node contains a number called `key`
- The `key` of all children to the left is less than the `key` of the node
- The `key` of all children to the right is greater than the key of the node

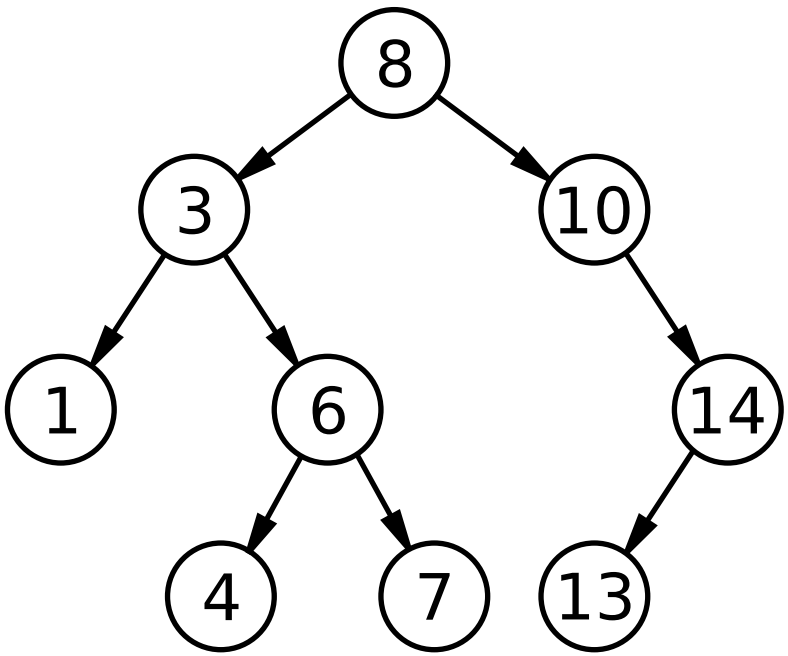
Binary Search Tree



```
tree = {
  "key": 7,
  "left": {
    "key": 3,
  },
  "right": {
    "key": 12,
    "left": {
      "key": 9,
    },
    "right": {
      "key": 15
    }
  }
}
```

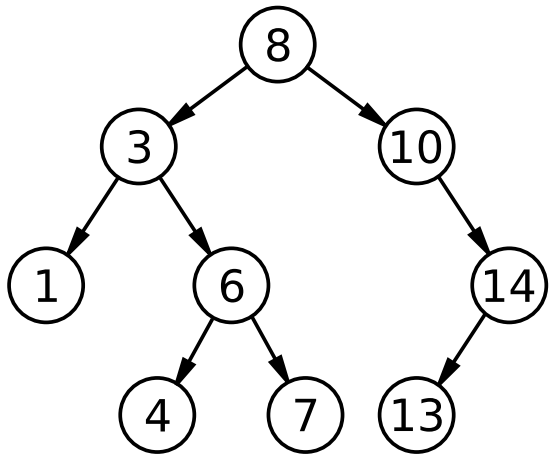
Search Algorithm

Given a tree and a number n, what is an algorithm to search for that number in the tree?



Search Algorithm (Iterative)

1. Start at the root and assign it to `node`
2. If `n` equals `key`, return true
3. If `n` is less than `key`:
    1. If there is no left child, return false
    2. Assign the left child to `node`, go to 2
4. If `n` is greater than `key`:
    1. If there is no right child, return false
    2. Assign right child to `node`, go to 2



```
def search(root, n):
  node = root
  while True:
    key = node["key"]
    if n == key:
      return True
    if n < key:
       if "left" in node:
         node = node["left"]
       else:
         return False
    else:
       if "right" in node:
         node = node["right"]
       else:
         return False
```

Break

Lab 4: Tightrope

https://tinyurl.com/wilson-pi-day-4