Data Structures

Let's solve a maze!

Before we start: A few words about outlines

Outlines

- Get thoughts out *before* writing code
  - Can revise after writing code
  - Makes code more readable
  - Does *not* replace function headers or comments
- Pseudocode versus paragraphs
  - Pseudocode makes it easier to translate to code
  - Either case, should be able to understand the algorithm from reading it
- Pseudocode
  - Should be the “English” version of your code—copying code is as useful as reading the code itself.
  - Helper functions called should be explained

Quick aside: Why you should care

- *"This is just CS2 busywork."*
  - Data structures are fundamental to CS; you'll have to know them to take future CS classes.
- *"The libraries are already written, so I don't have to know how they work."*
  - What if a library isn't available?
  - What if you need special functionality?
- *"I only need to know one data structure, and I can apply it to every situation."*
  - That's the whole point of this week's assignment...
  - Using the right data structure can improve performance

There are so many out there!

This week, we'll be working with stacks and queues, but there are lots more...

Linked lists

Sets

Hash tables

Graphs

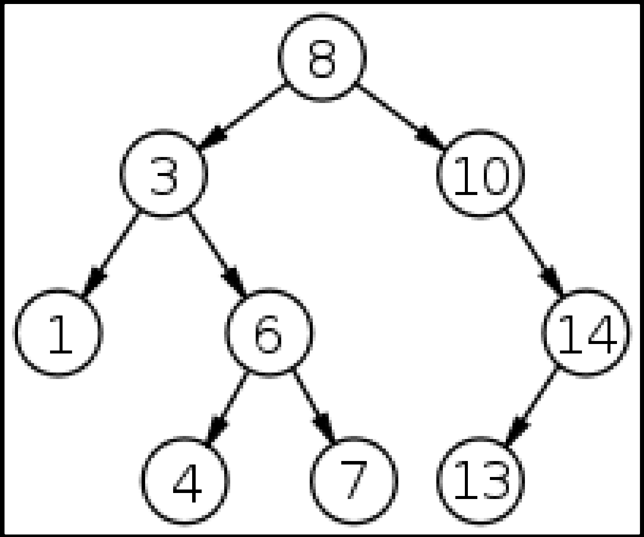
Trees

Arrays

New topic: Binary search trees as a simple data structure

Search trees (and why)

To find an element in a linked list, we must visit consecutive list elements until we find what we want. Not true for search trees!



BST: Average O(log n)
LL: Average O(n)

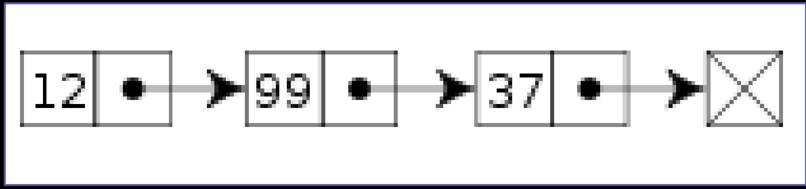
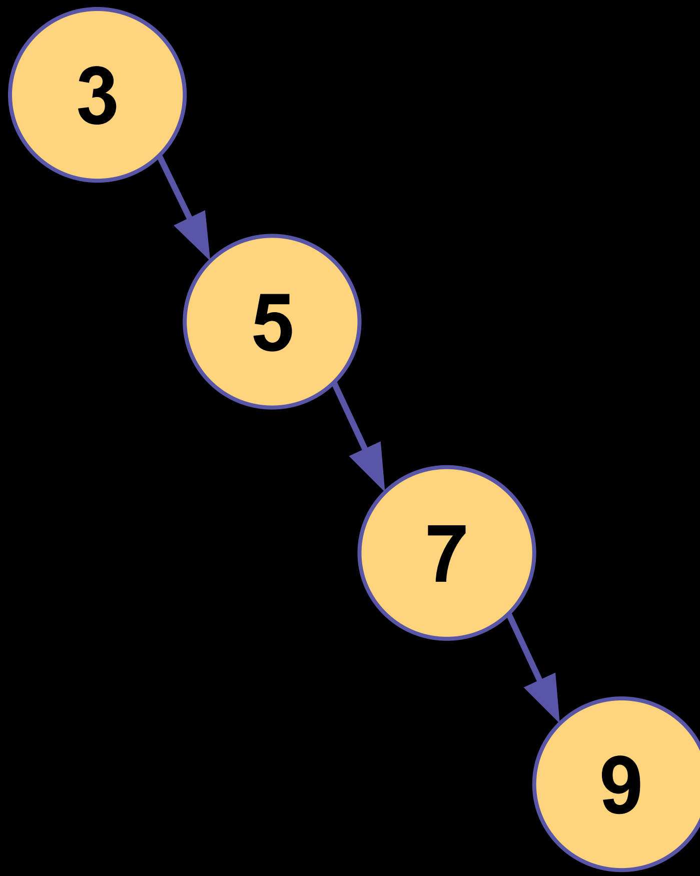


Image credit: Wikipedia

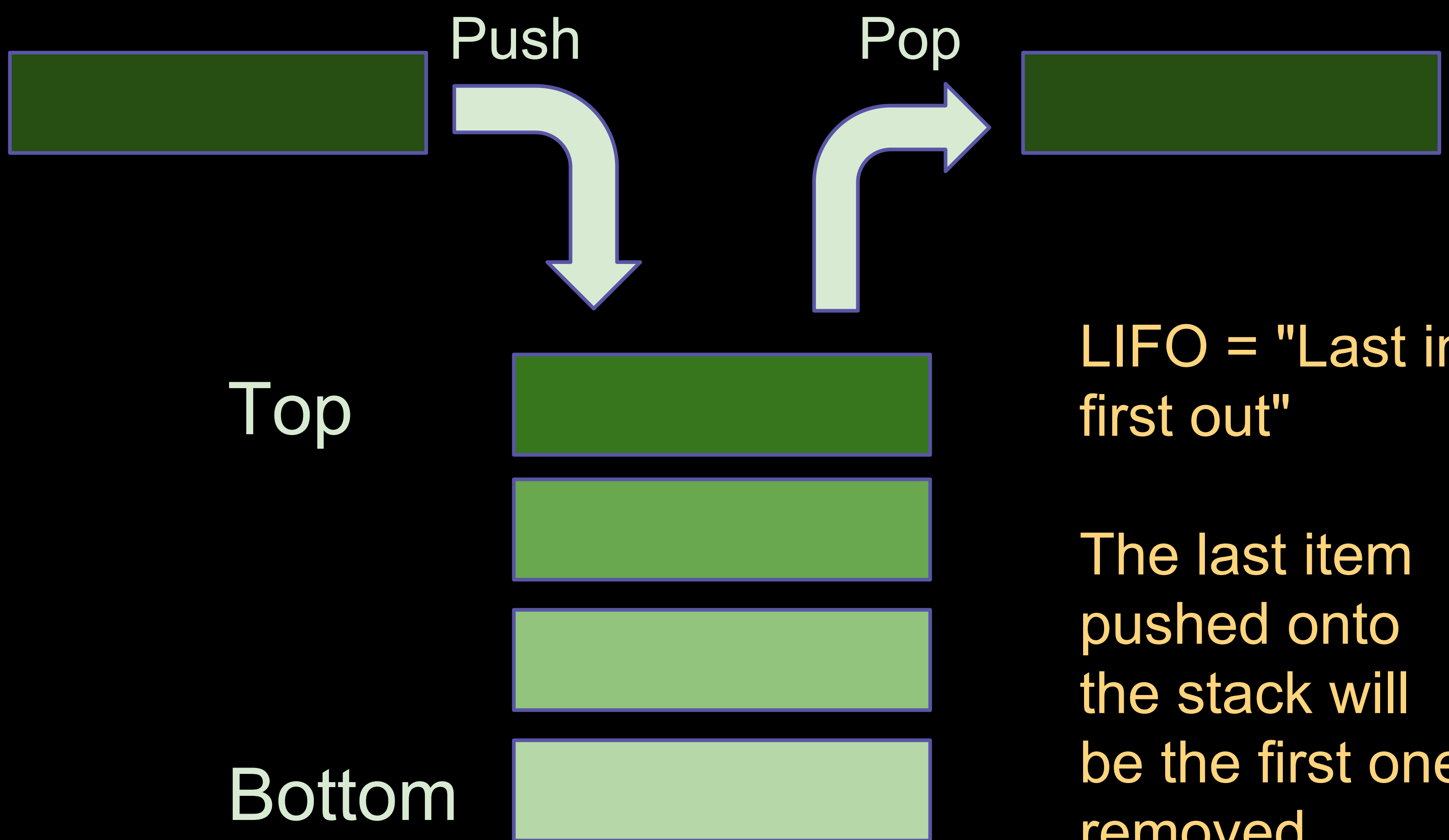
Search trees (and why)

Balance is important! If the BST is poorly constructed, it effectively becomes a linked list, and our average search time increases.



New topics: Stacks and queues; search algorithms

Basics of the stack



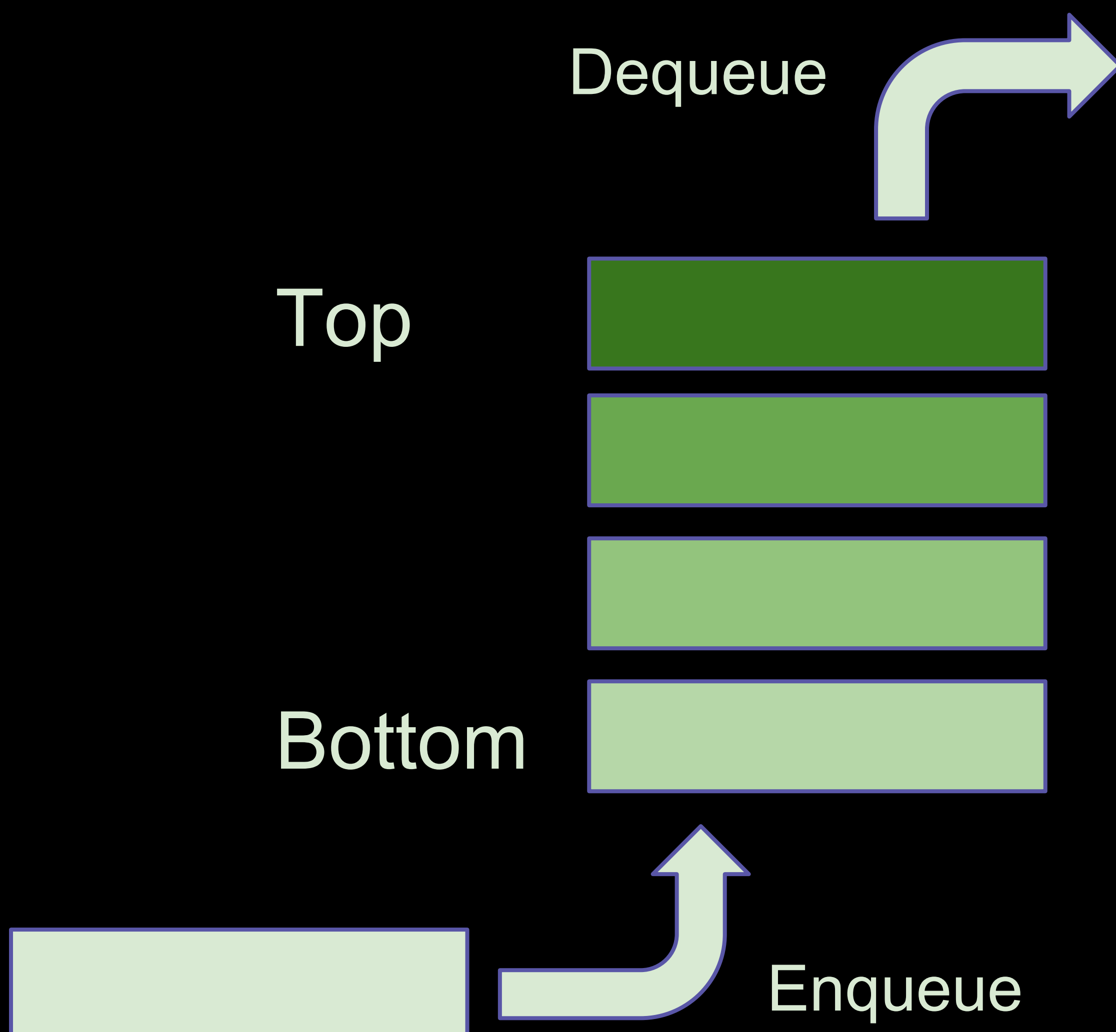
LIFO = "Last in first out"

The last item pushed onto the stack will be the first one removed.

Stacks: Implementation details

- What happens if you pop from an empty stack?
- How do you handle adding the first item?
- How do you know when the stack is empty?
- One way to manage a stack:
  - Keep a pointer to the top of the stack
  - Each item stores a pointer to the item below it
- Remember LIFO: Add and remove items from the top of the stack.

Basics of the queue



FIFO = "First in first out"

The first item pushed onto the stack will be the first one removed.

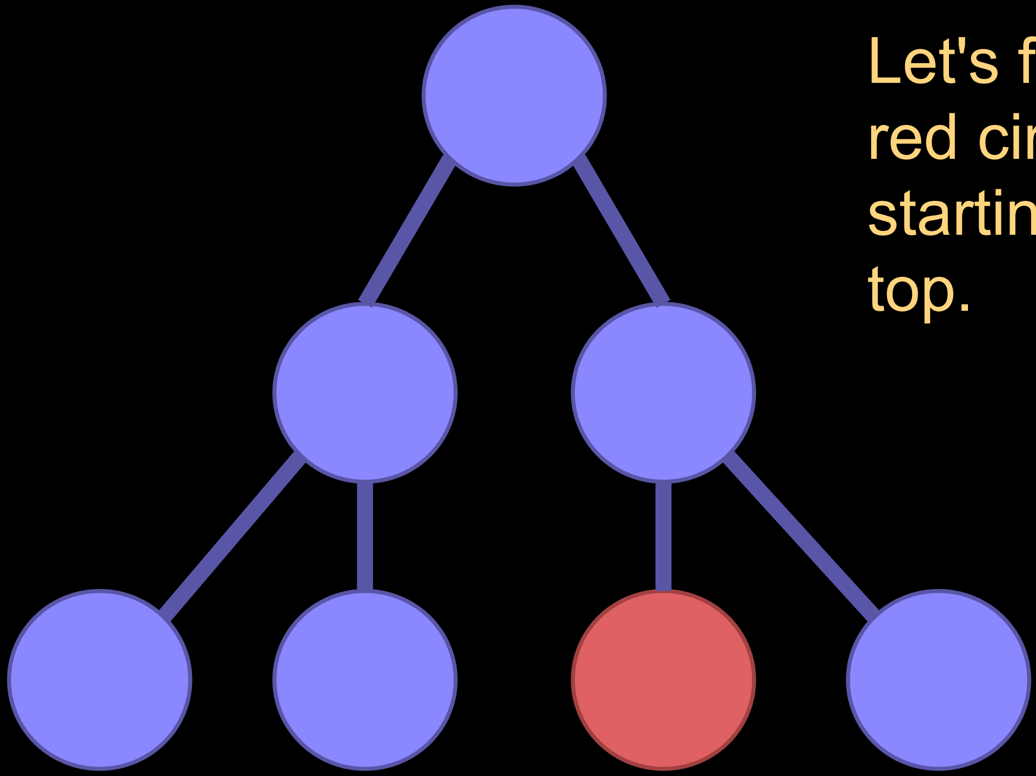
Queues: Implementation details

- What happens if you dequeue from an empty queue?
- How do you handle adding the first item?
- How do you know when the queue is empty?
- One way to manage a queue:
  - Keep a pointer to the front and rear of the queue
  - Each queue item stores a pointer to the next item in the queue
- Remember FIFO: Add new items to the rear, and remove items from the front.

Introducing the depth-first search (DFS)

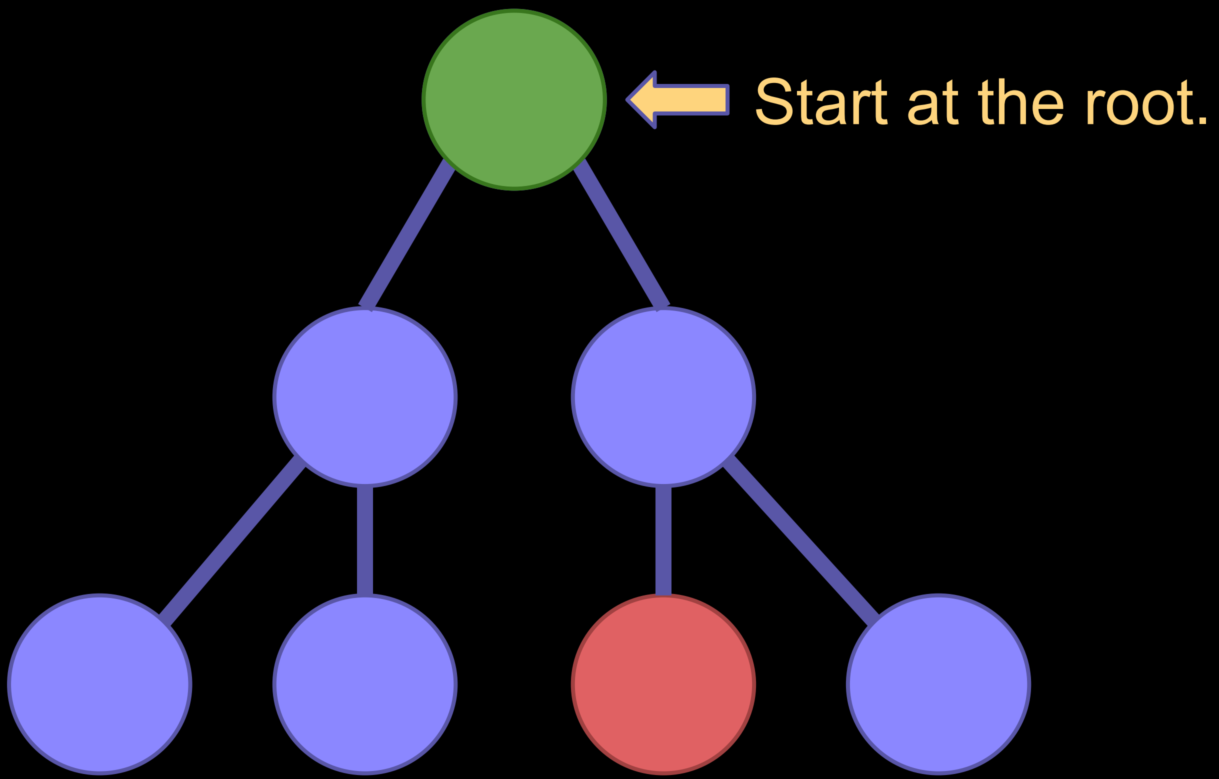
- DFS searches as far as possible along a branch before backtracking.
- From any node, we move to the first unvisited child node until we can go no further (i.e. the node we have found has no children).
- Then backtrack until we hit a node with unvisited children and repeat the process.

Solve something easy with DFS



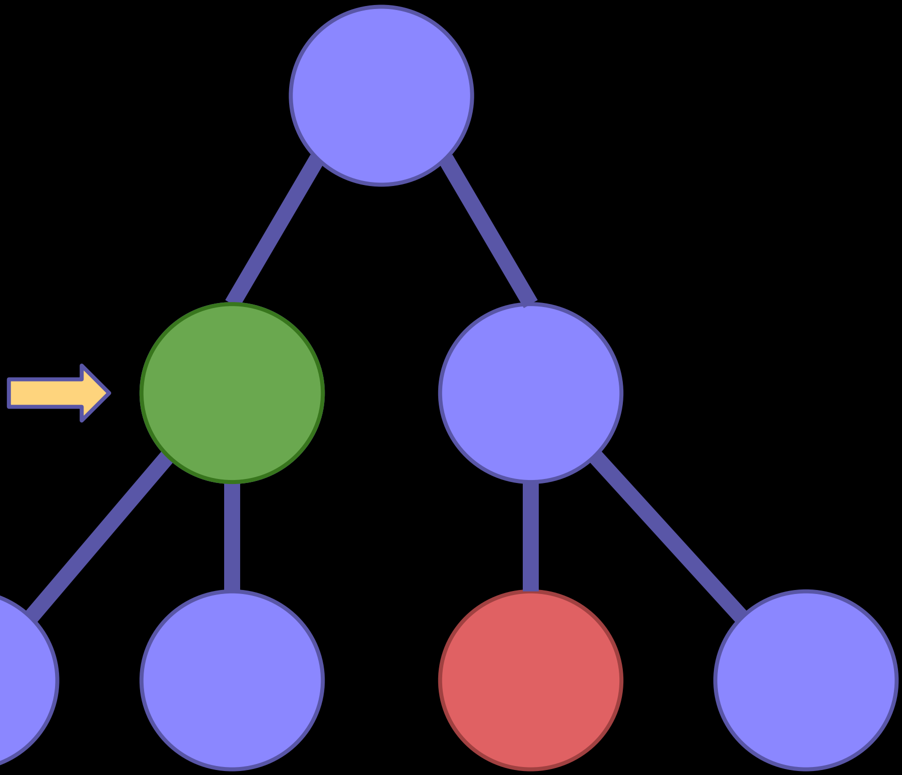
Let's find the red circle, starting at the top.

Solve something easy with DFS



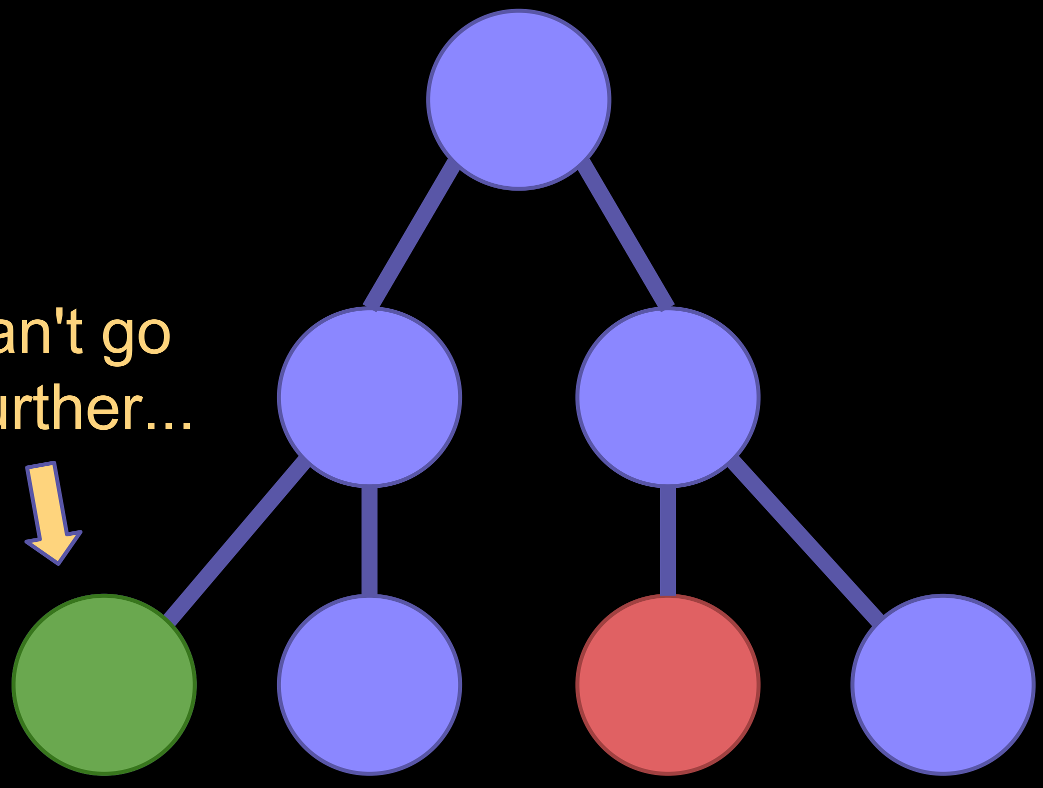
Solve something easy with DFS

Move to the first child.



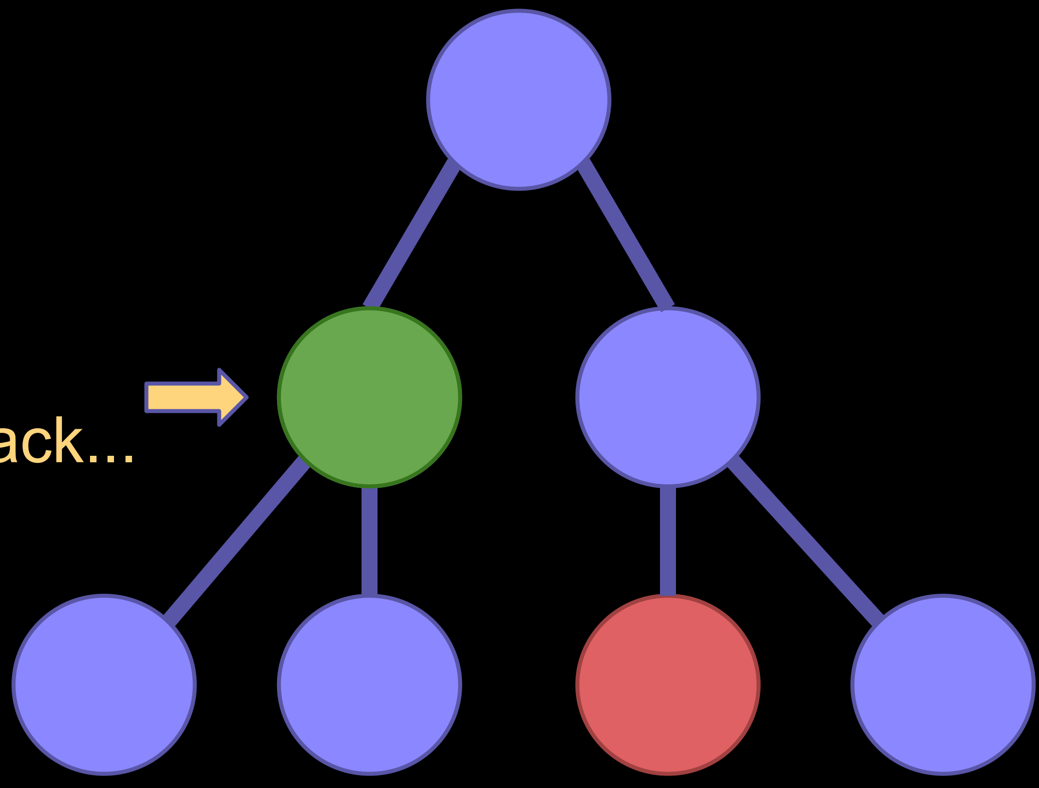
Solve something easy with DFS

We can't go
any further...

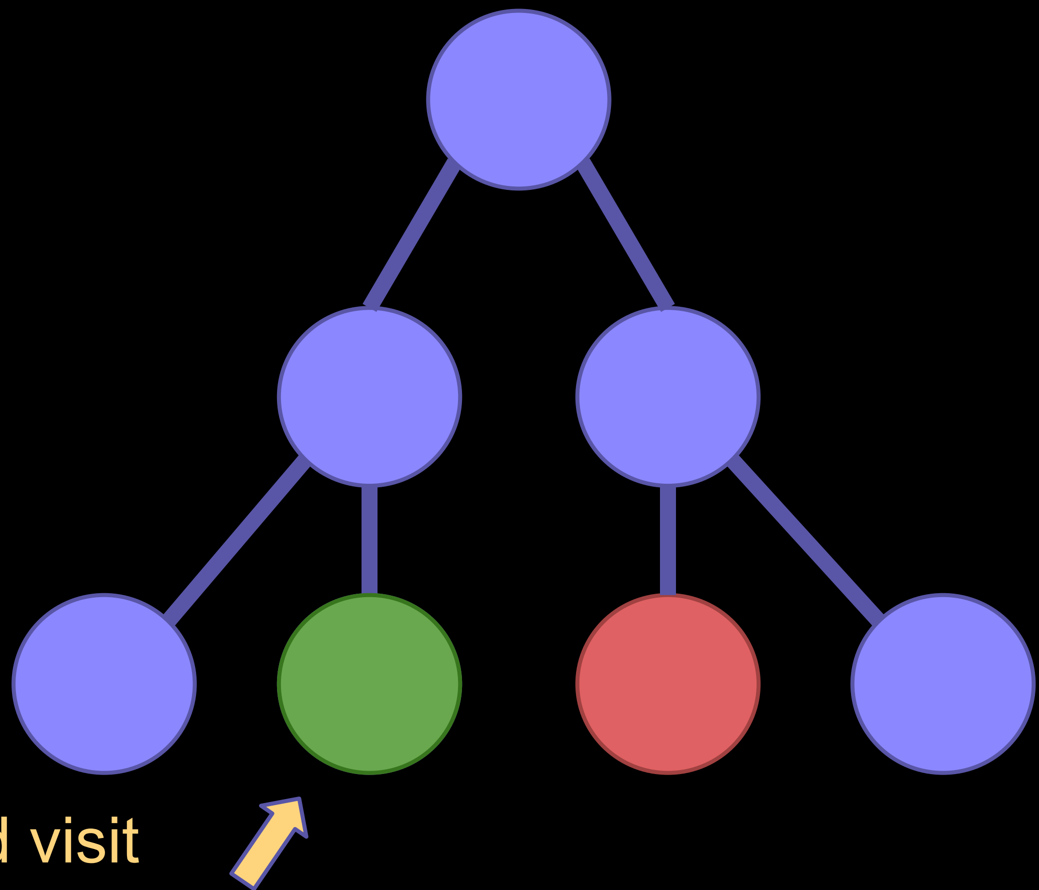


Solve something easy with DFS

... so
backtrack...



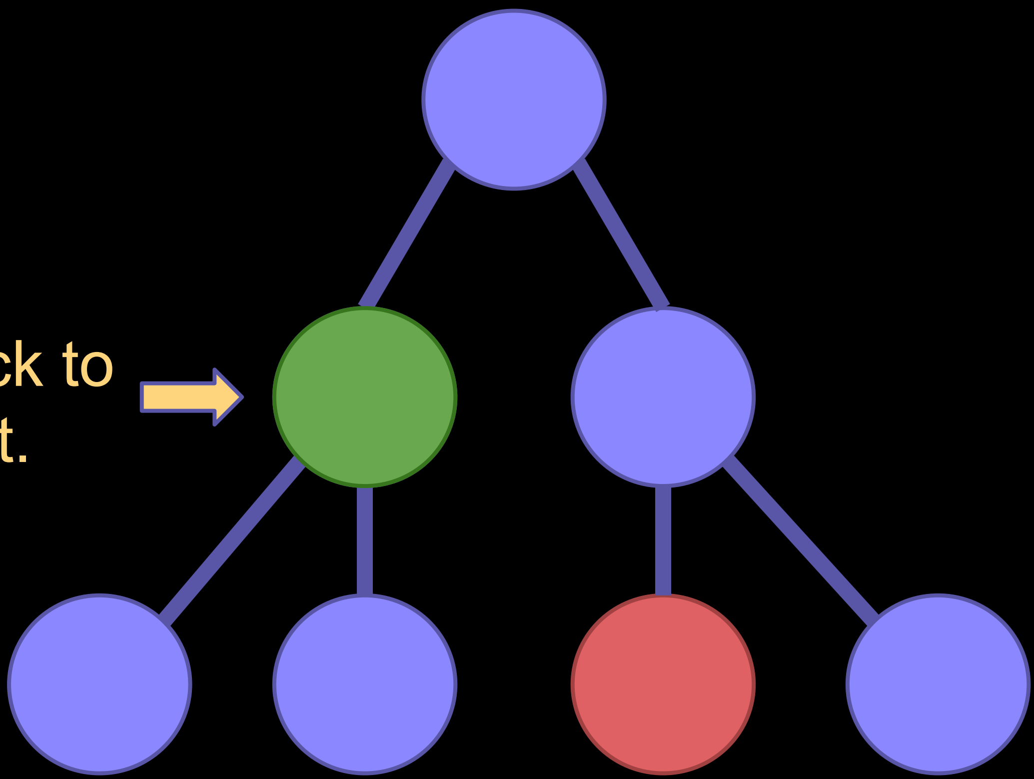
Solve something easy with DFS



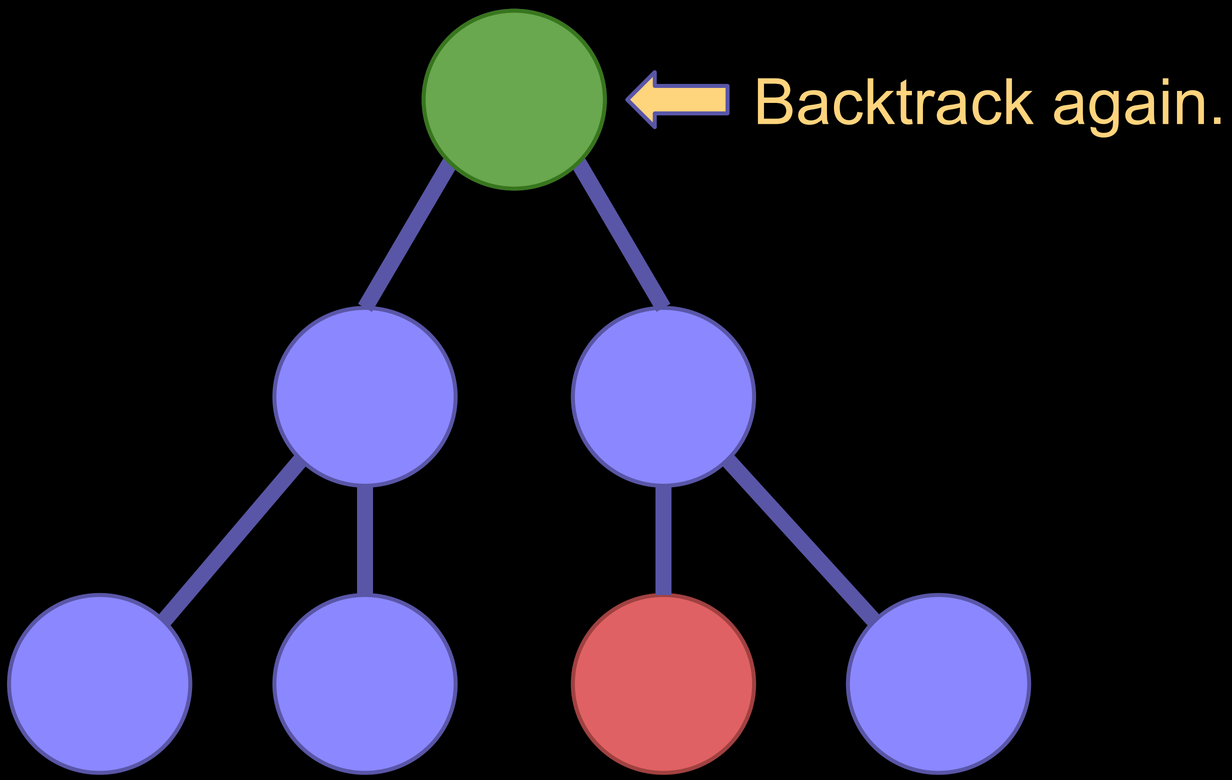
... and visit
the next child.

Solve something easy with DFS

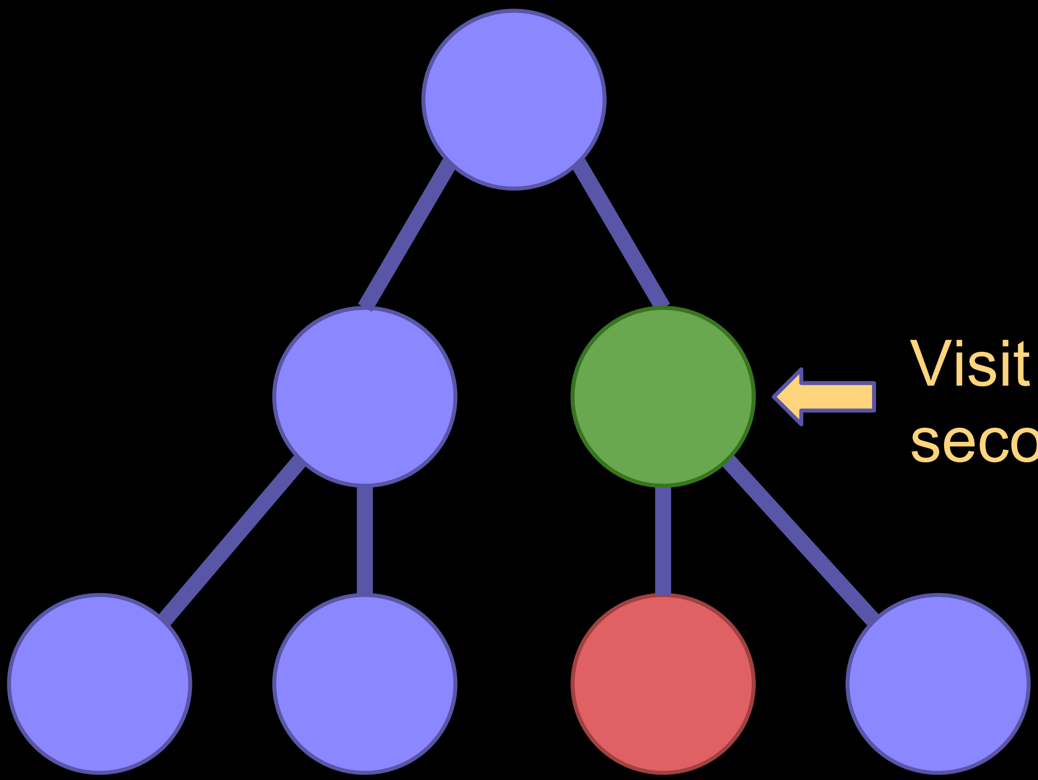
We're back to
the parent.



Solve something easy with DFS

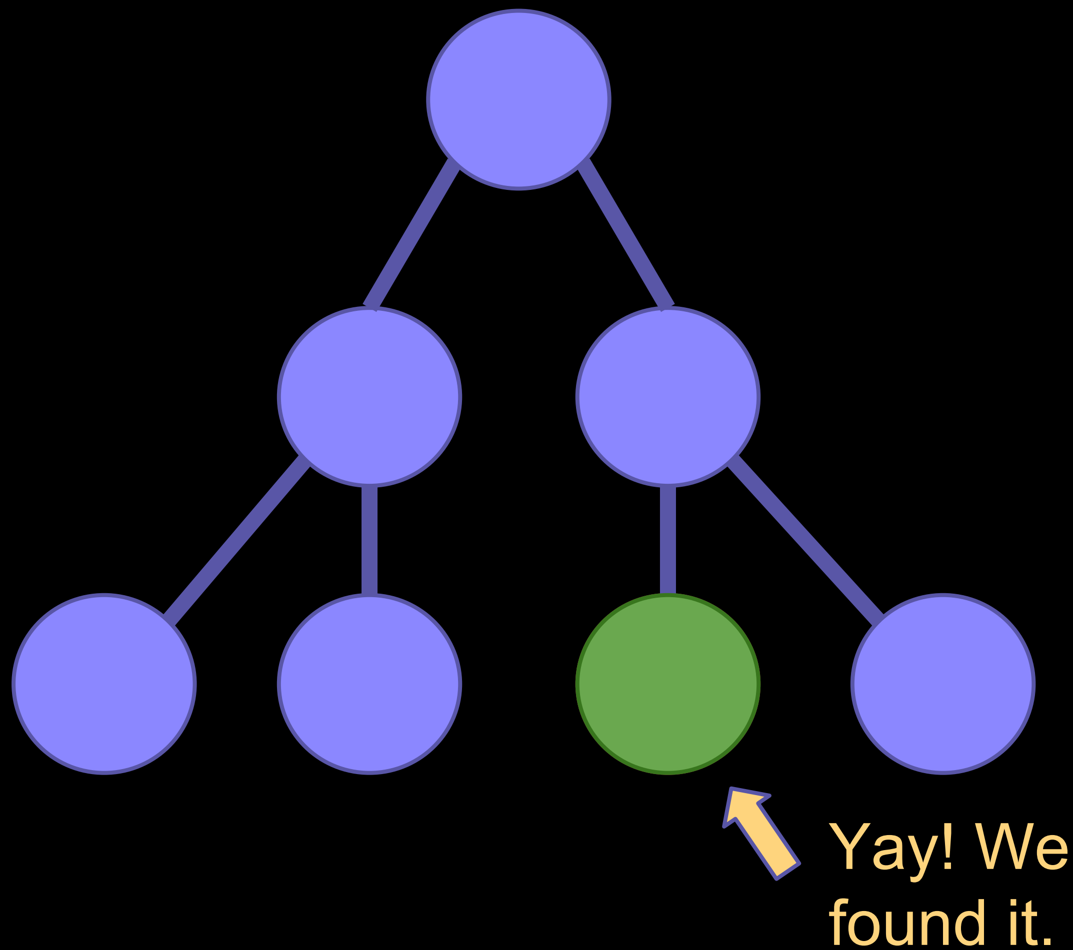


Solve something easy with DFS



Visit the
second child.

Solve something easy with DFS

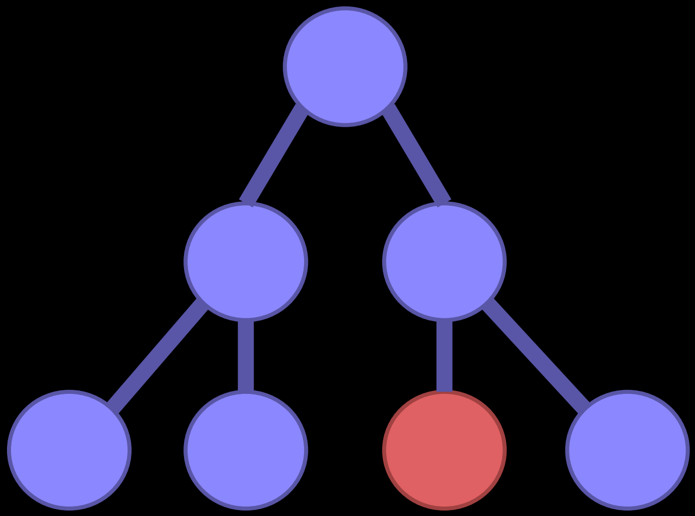


What does DFS have to do with data structures?

Some DFS pseudocode with a stack...

```
Push the root node

while stack is not empty:
    Mark the current
    node as visited

    if current node is end:
        Stop the search
    else:
        Push the next unvisited
        child node onto the
stack
```

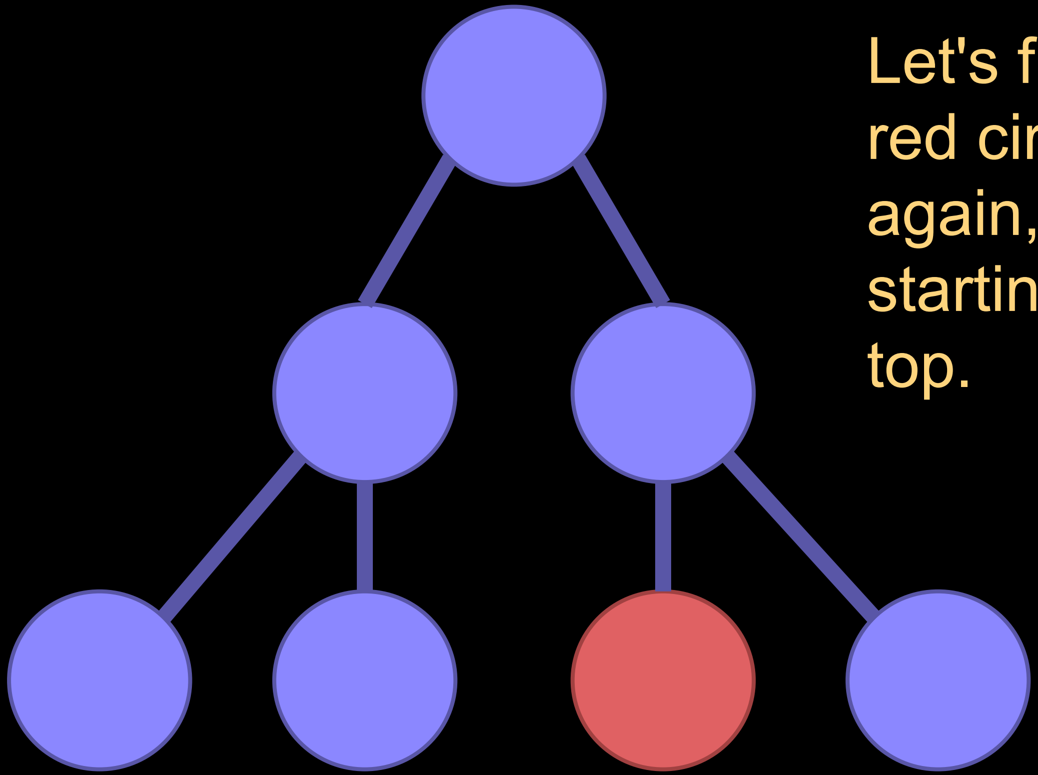
Backtracking is as simple as popping items from the stack.



Introducing the breadth-first search (BFS)

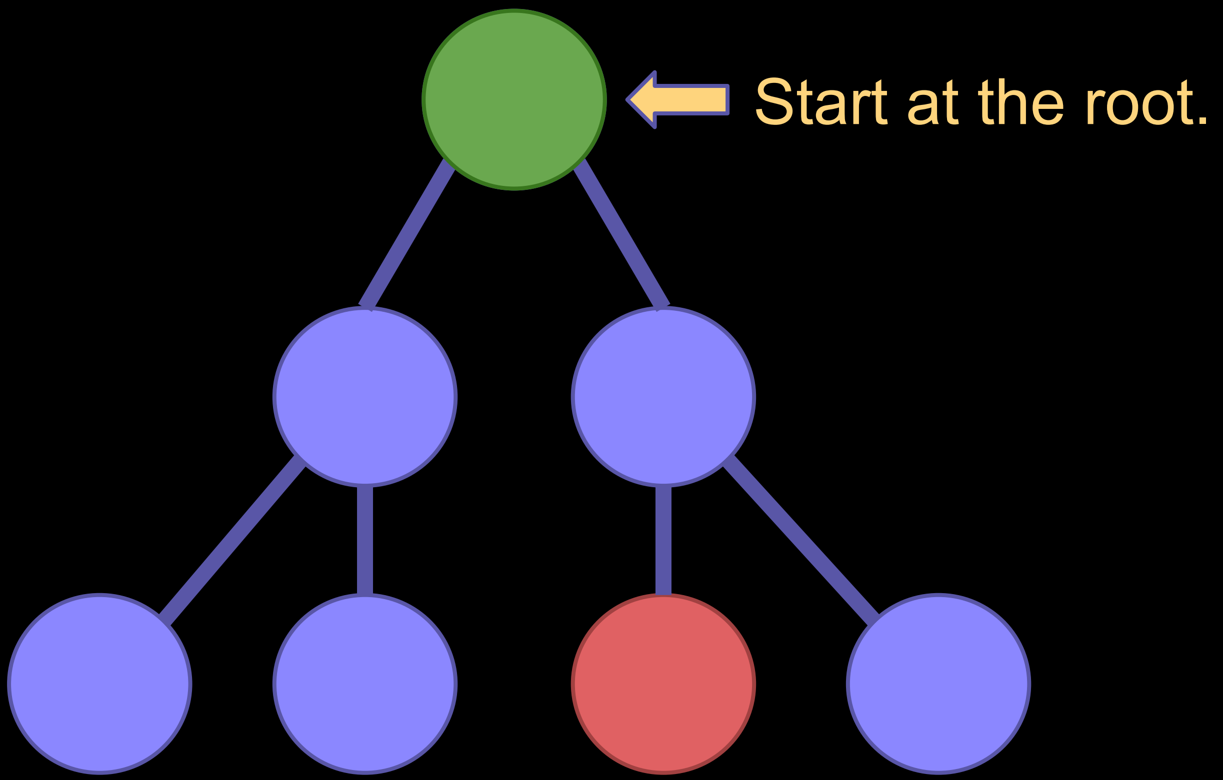
- BFS searches each level of nodes in turn (root, root's children, root's children's children...).
- Though it can certainly reach the end of a branch, it never backtracks.

Solve something easy with BFS



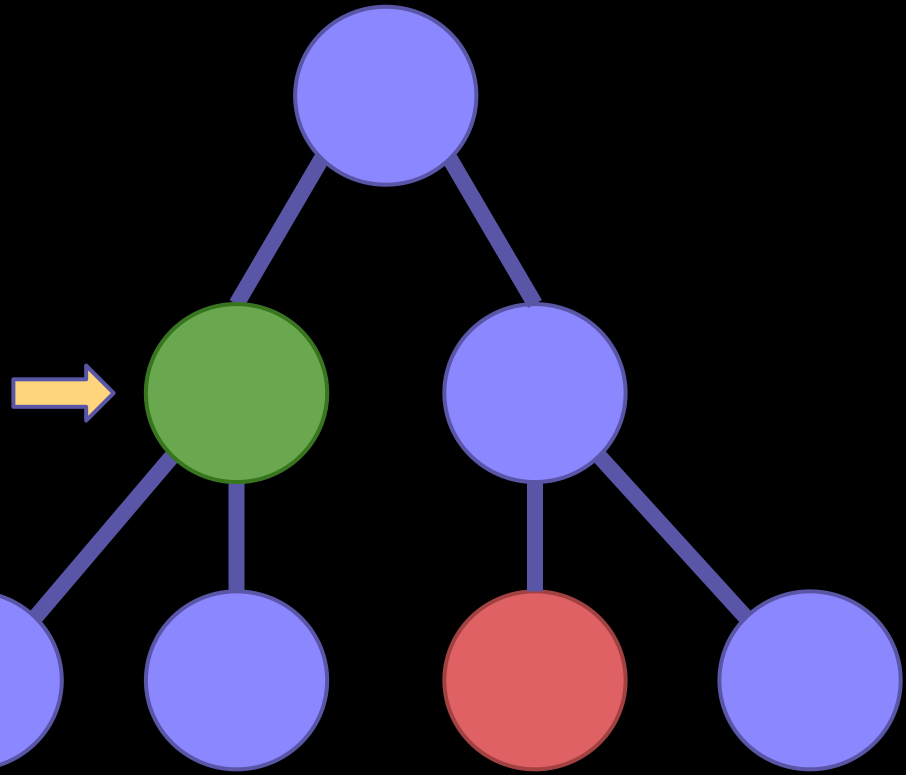
Let's find the red circle again, still starting at the top.

Solve something easy with BFS

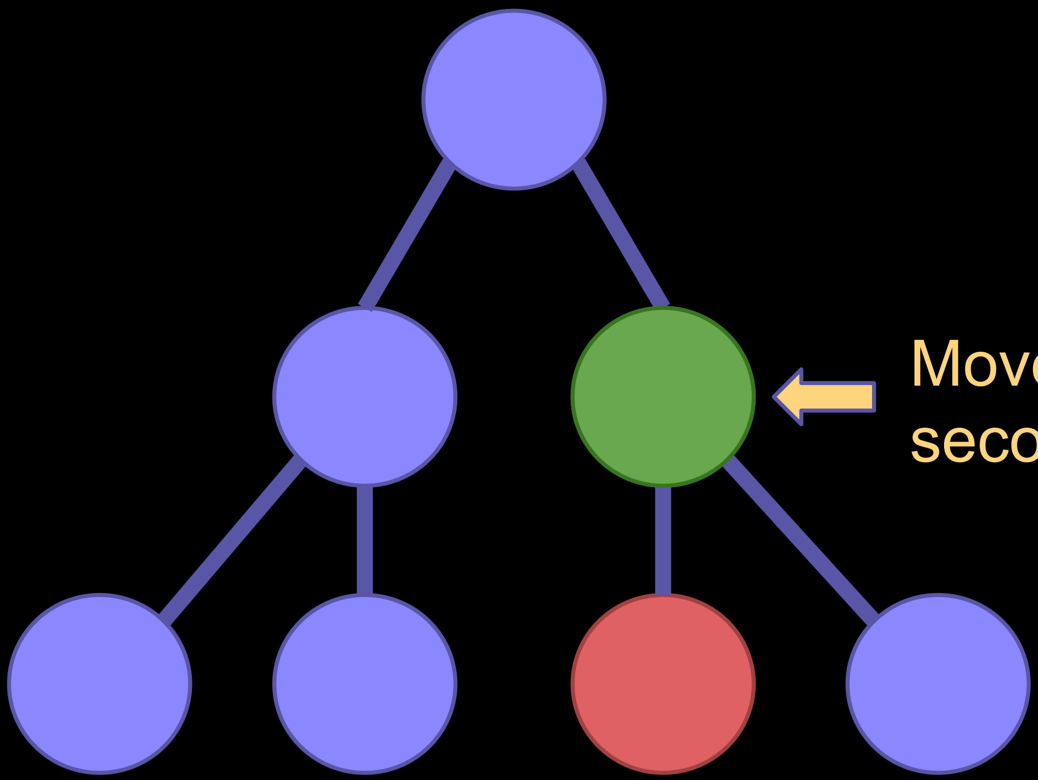


Solve something easy with BFS

Move to the first child.



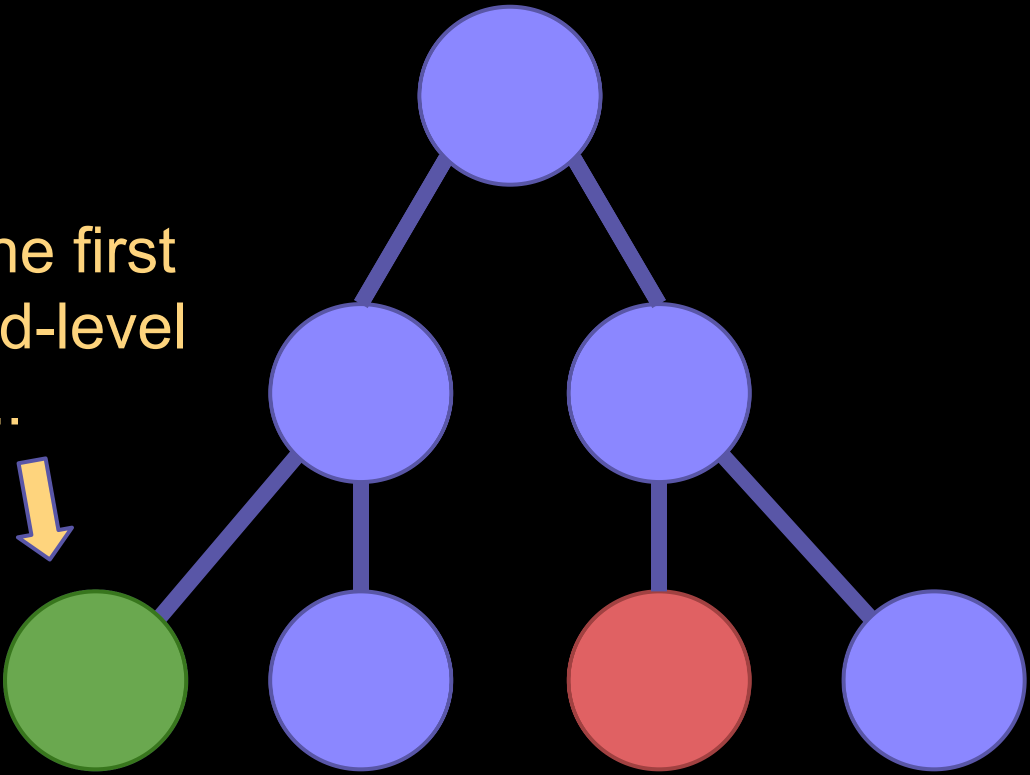
Solve something easy with BFS



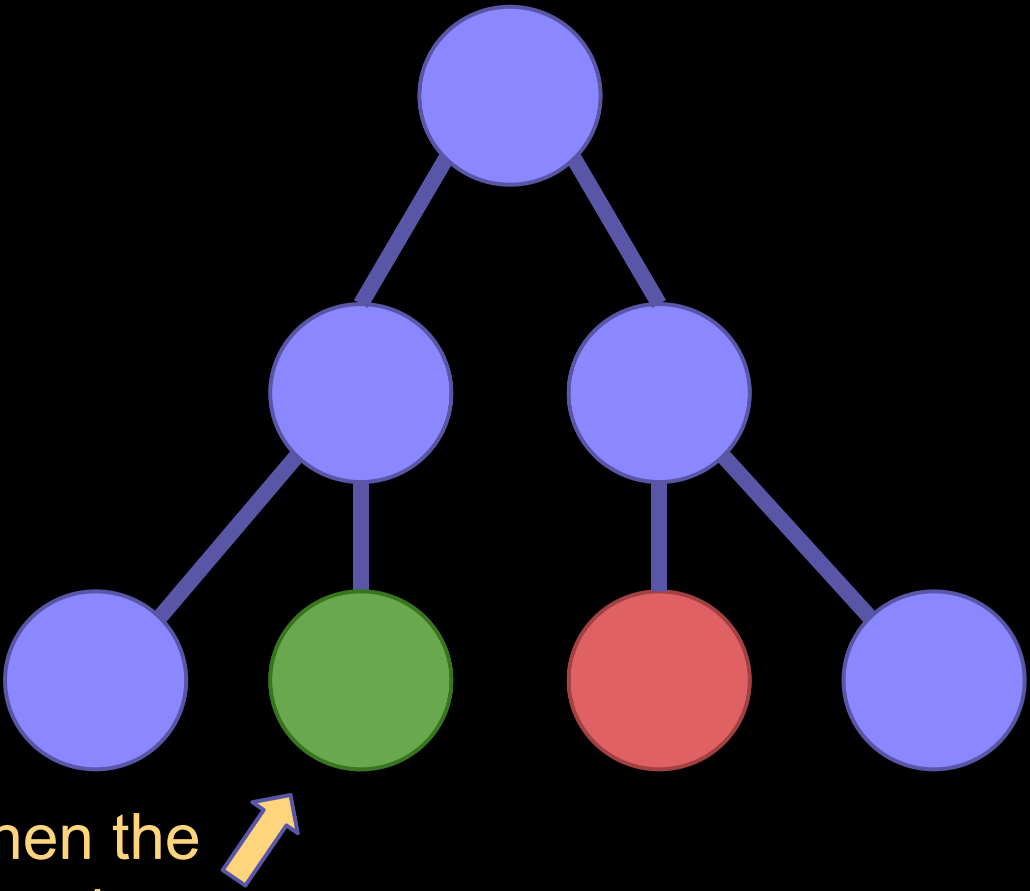
Move to the second child.

Solve something easy with BFS

Visit the first second-level node...

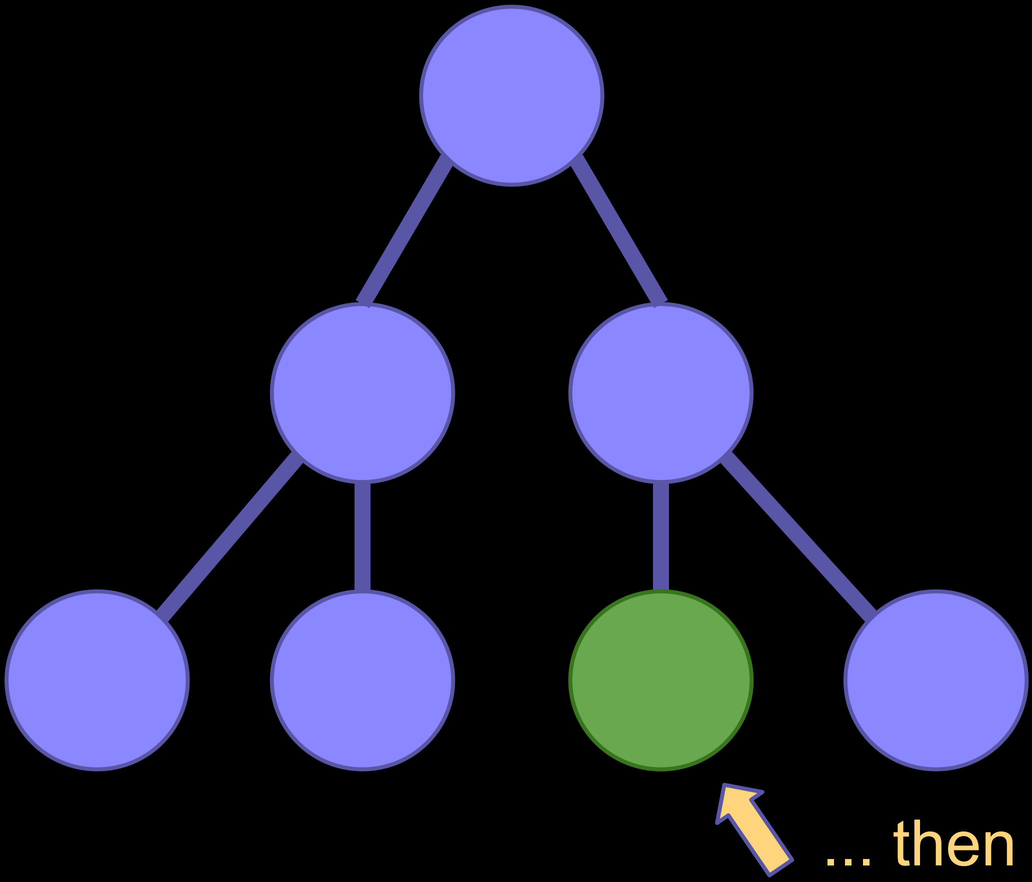


Solve something easy with BFS



... then the
second...

Solve something easy with BFS

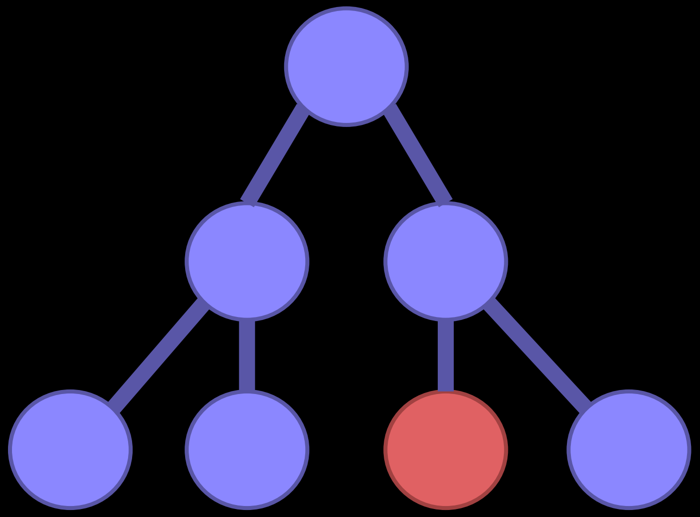


... then the
third. Found it!

What does BFS have to do with data structures?

Some BFS pseudocode with a queue...

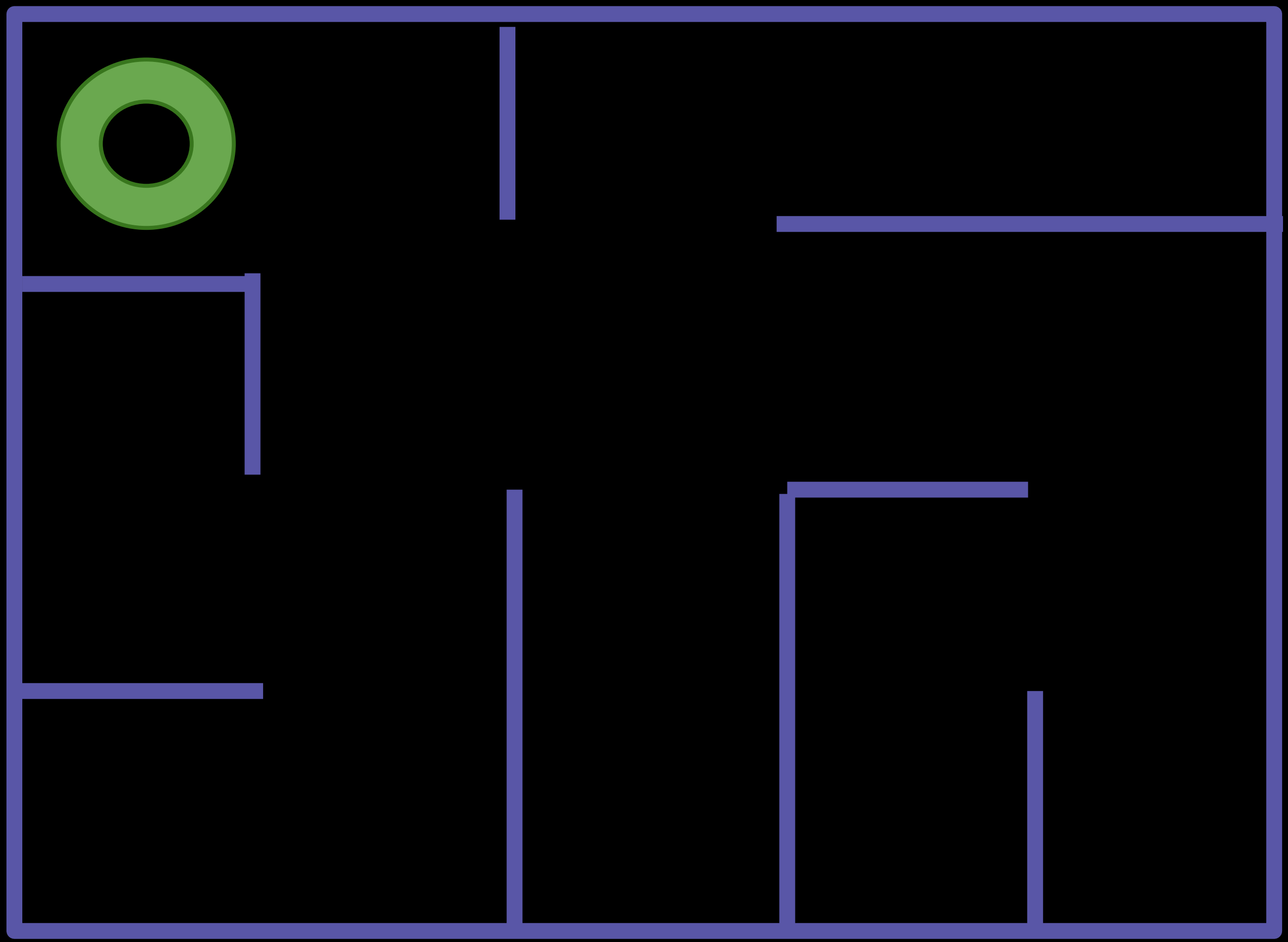
```
Enqueue the first node

while queue is not empty:
    Mark the current node as
    visited

    if current node is end:
        Stop the search
    else:
        Enqueue all unvisited
        child nodes
```



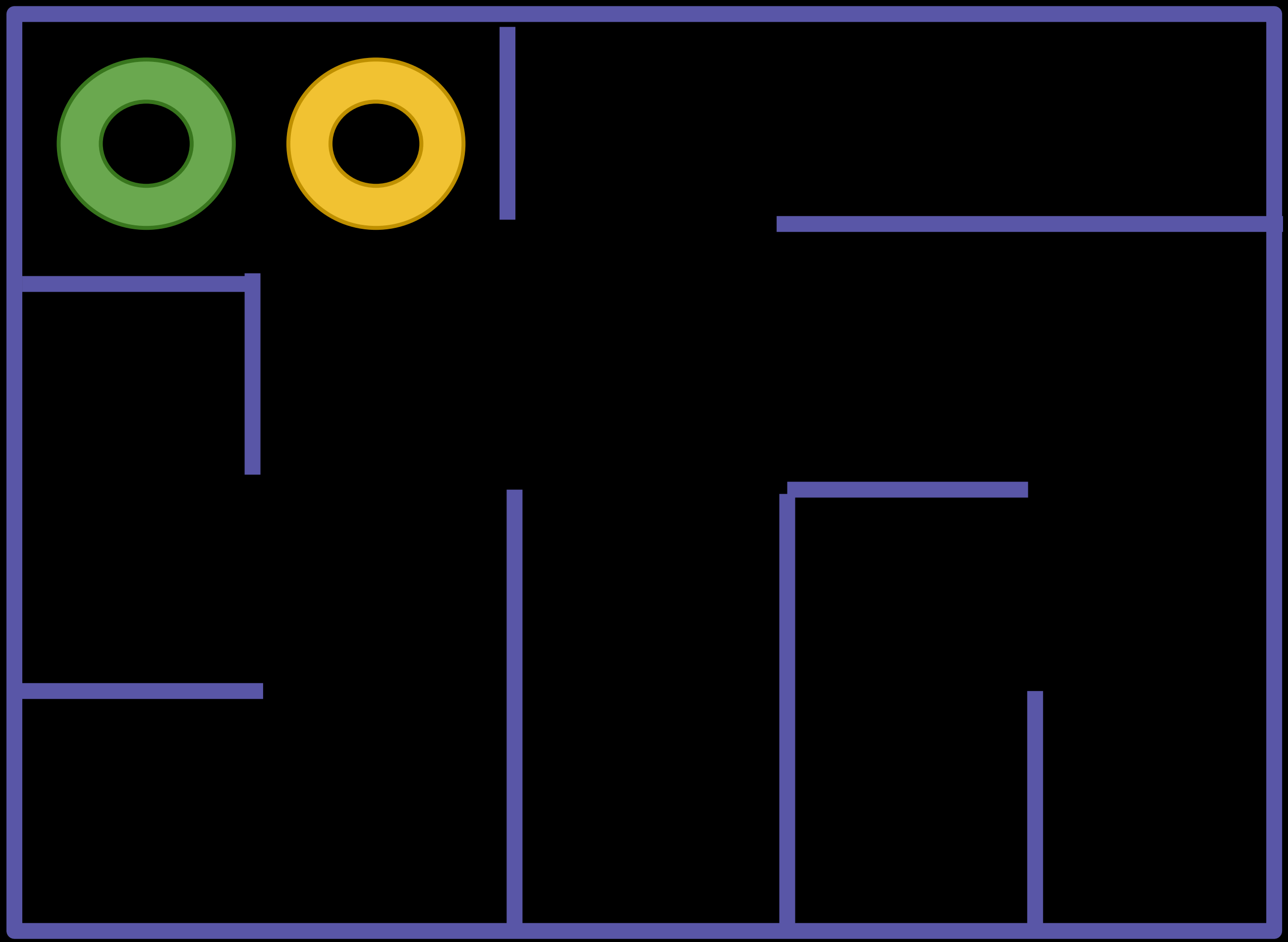
From trees to mazes

- Trees aren't much different than mazes.
- A node of the tree corresponds to a square of the maze.
- The children of a node correspond to the squares you can move to from the current square.
- You have to keep track of where you've been!

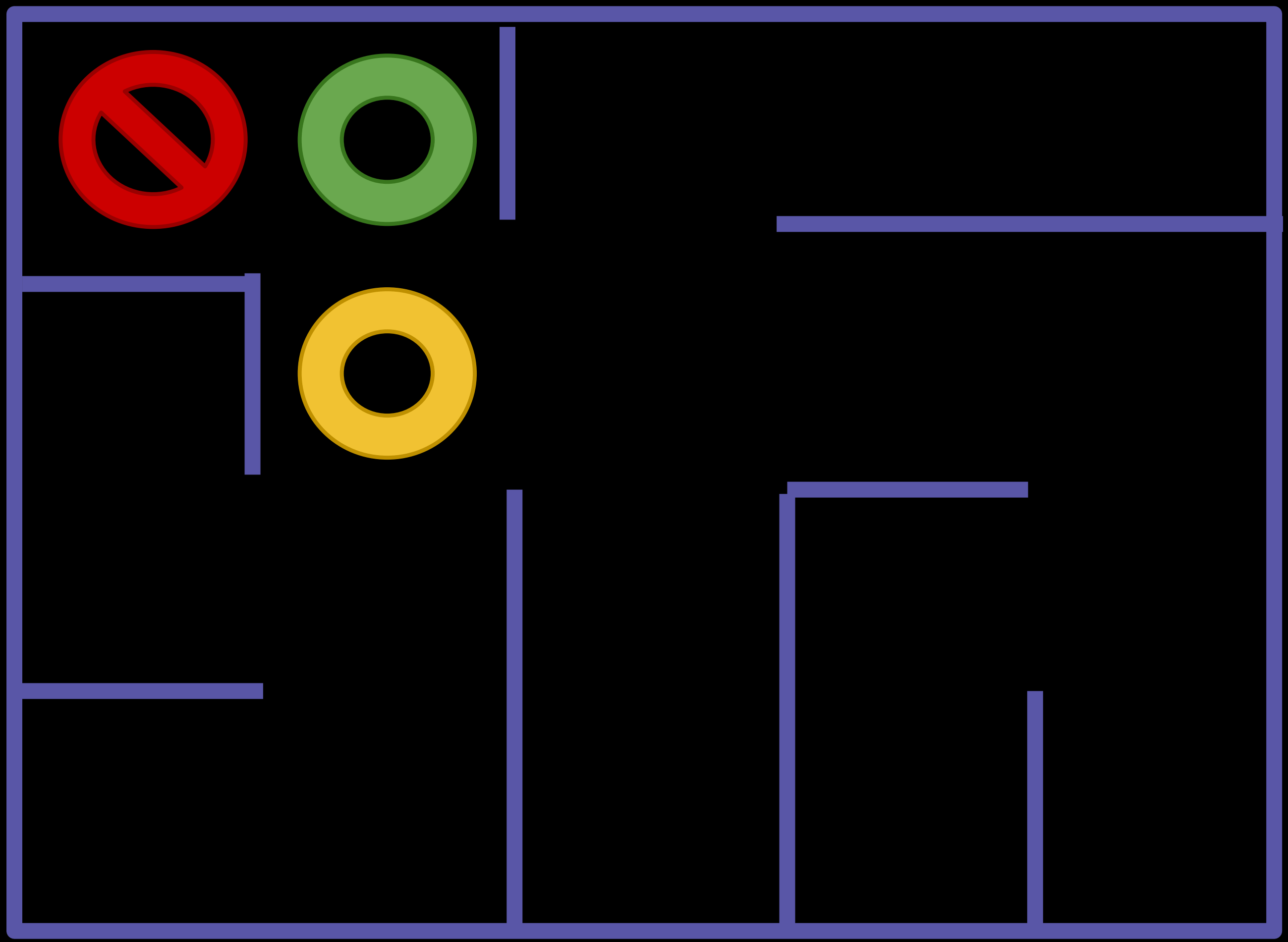
But what does this have to do with mazes???



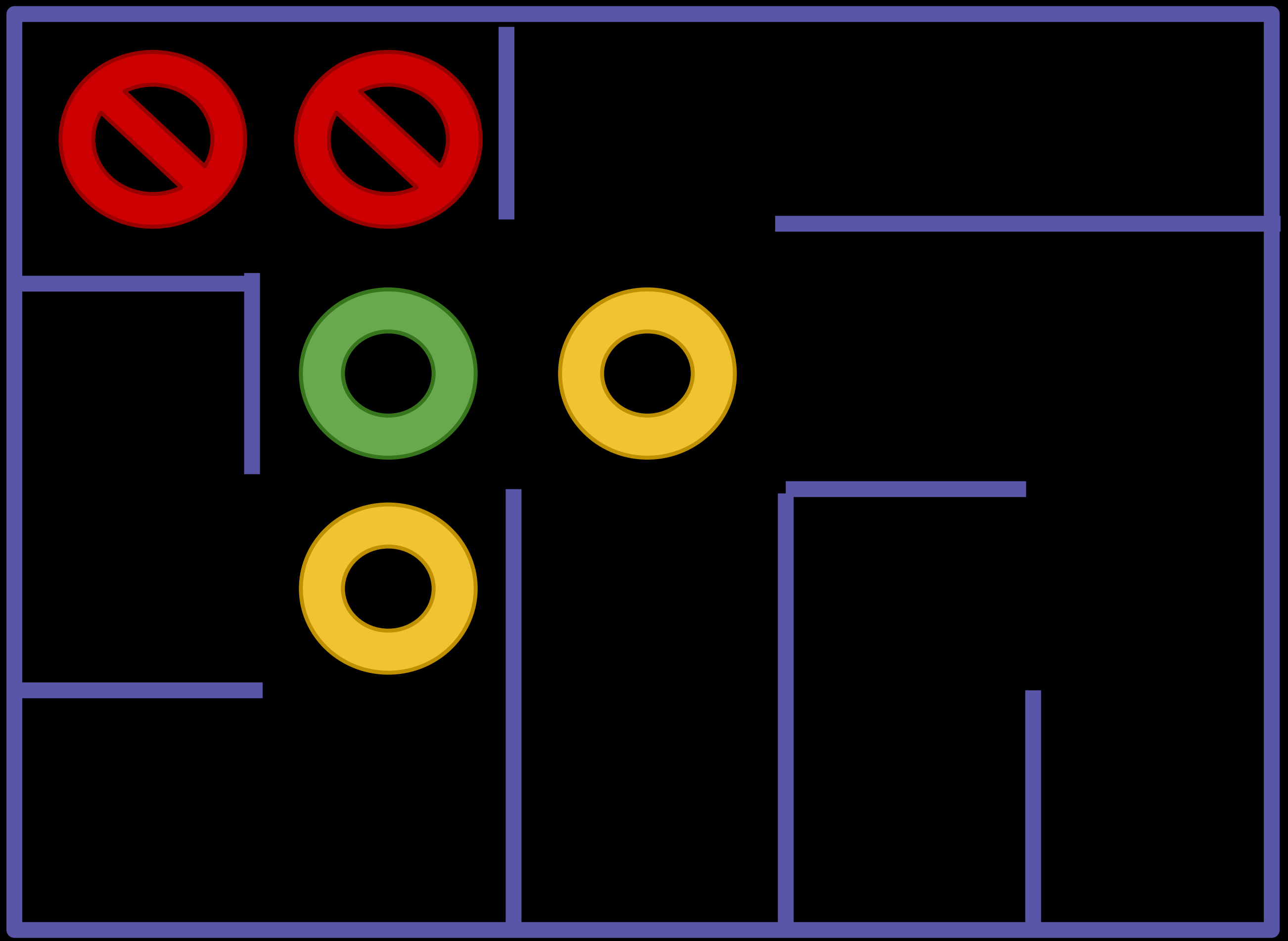
A maze can be viewed as a tree whose node is your starting point in the maze and every possible adjacent move from a node is a child.



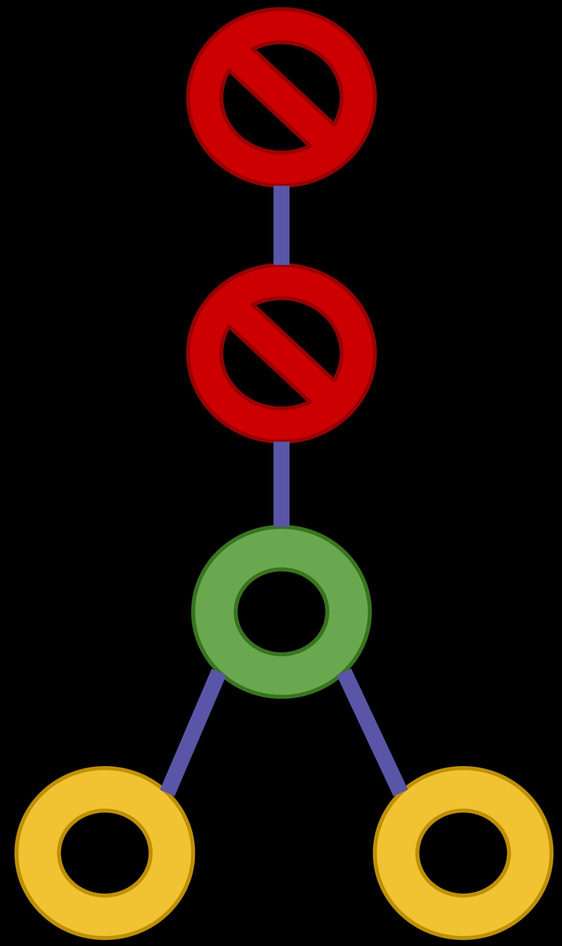
A maze can be viewed as a tree whose node is your starting point in the maze and every possible adjacent move from a node is a child.



A maze can be viewed as a tree whose node is your starting point in the maze and every possible adjacent move from a node is a child.



A maze can be viewed as a tree whose node is your starting point in the maze and every possible adjacent move from a node is a child.



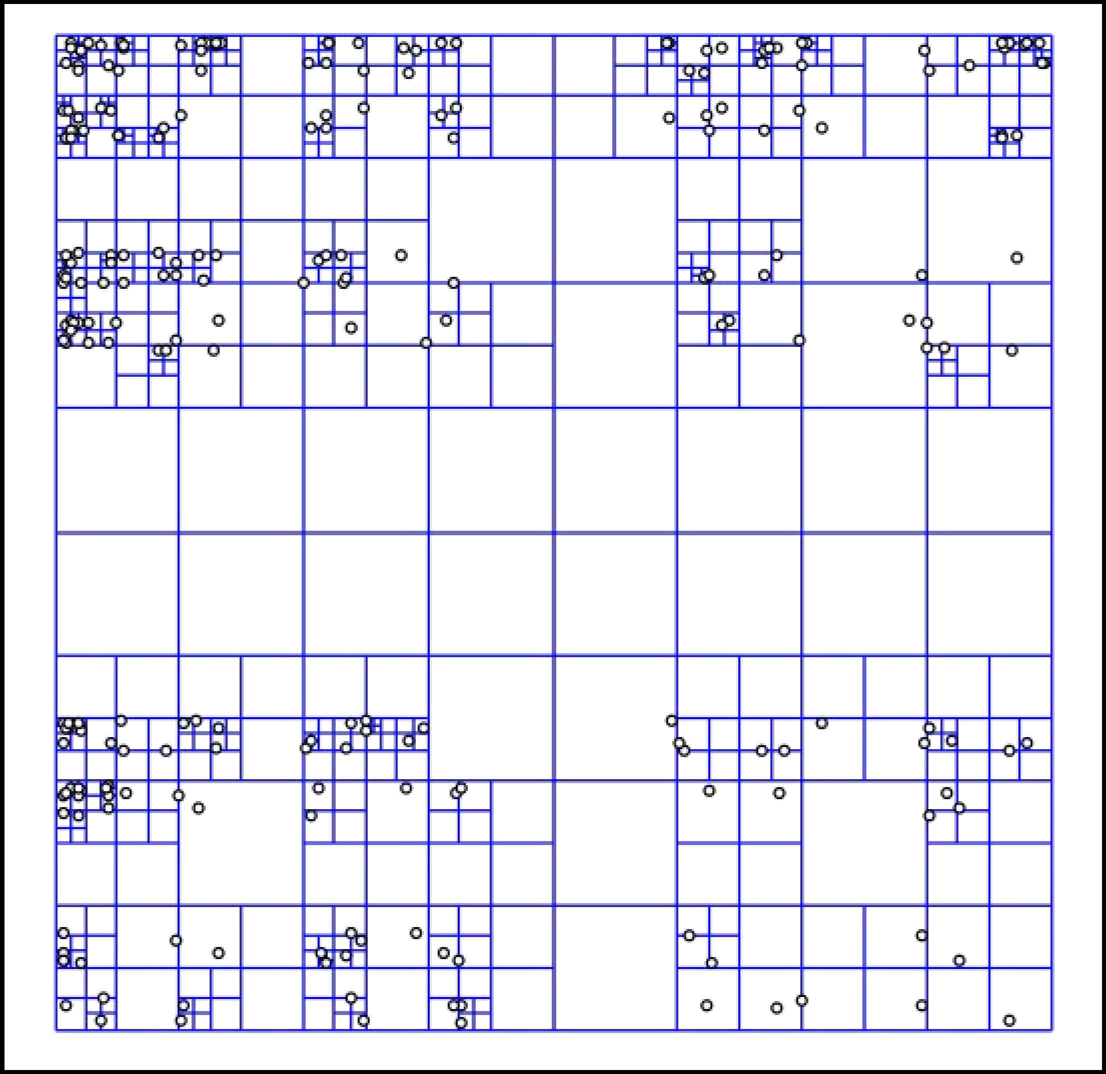
The tree equivalent to the maze might look something like this; hopefully, you can see that mazes and trees are basically the same thing, so you can search a maze with BFS and DFS.

Notes

- Read the supplied code before starting
- In DepthFirstSolver::get_path(), do not use push()
- The stack and queue themselves should be linked lists, not std::vector's
- Comment your code
- Don't delete files that running "make clean" wouldn't delete

Advanced topic: Quadtrees (or how to organize spatial points)

Quadtrees



No rectangle (region) contains more than one point!

Image credit: Wikipedia

Good Luck