Continuing to Get to Know Java: *Stacks, Queues, and Searching Graphs*

Java Lecture 2

2-20-12

Paul Ruvolo

Outline for Today

- Continuing to get to know Java (review + Loops and Arrays)
- Our first Java Data Structures: Stacks and Queues
- Graph Searching with Stacks and Queues

Motivating Example:
Best Scrabble Word Revisited

Review from Last Time

- What is a Java Program?

  A Java program is a collection of classes

- What is a class?

  Classes define a **blueprint** for creating Objects (including the data each object contains and the operations that can be performed on it)

- What is an object?

  An object is particular ***instance*** of a class that is created at runtime by a java program

Data in Java

Two types of data in Java:

- Objects
- Primitives: int, byte, short, int, long, float, double, boolean, char

Review: A Sample Java Class

```
class Point extends Object
{
    private double x;
    private double y;

    public Point(double x_in, double y_in)
    {
        this.x = x_in;
        this.y = y_in;
    }

    public Point()
    {
        this.x = 0.0;
        this.y = 0.0;
    }

    public void nudgeBy(double delta_x, double delta_y)
    {
        this.x += delta_x;
        this.y += delta_y;
    }
    // ...
}
```

Point "is an" Object

Name of the Class

Data members

Inaccessible to the outside world

Constructors

Accessible to the outside world

Method (an operation on a Point)

Review: Static Methods in Java

```
class Point extends Object
```

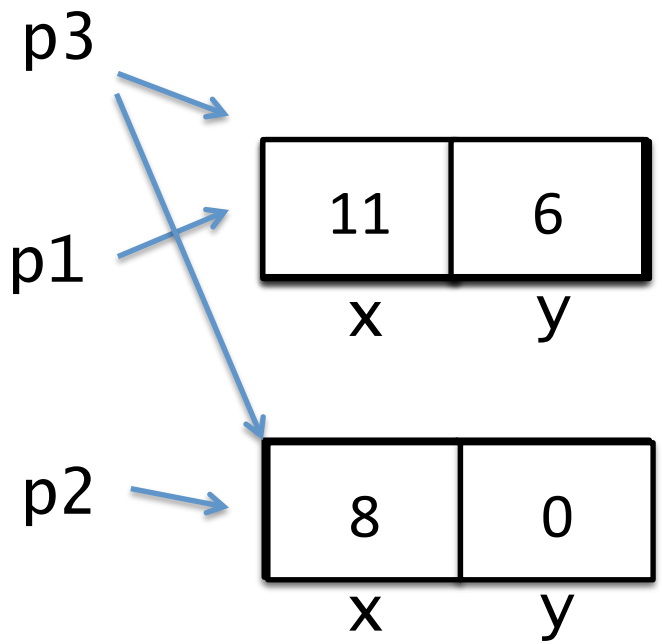
```
Point.java:106: non-static variable this cannot be
referenced from a static context
    System.out.println(this.x + " " + this.y);
                       ^
Point.java:106: non-static variable this cannot be
referenced from a static context
    System.out.println(this.x + " " + this.y);
                                      ^
2 errors
```

```
}
```

A static method is not invoked on a particular instance of an Object. As such it does not have access to the data members declared in the class.

Review: Objects and References

```
Point p1 = new Point(5,2);
Point p2 = new Point(5,2);
Point p3 = p1;

p1.nudgeBy(5,1);
p2.nudgeBy(3,-2);
p3.nudgeBy(5,3);

p3 = p2;
p1.nudgeBy(-4,0);
```



null

- **Null** is a special reference value where the reference points to nothing
- Common uses:
  - When declaring a reference that you want to initialize later
  - As a return value from a method in case of an error

null

```
public static void main(String[] args)
{
        String s = null;

        System.out.println(s.length());
}
```

Exceptions are Java's way of indicating errors at runtime (more on this in future lectures)

```
Output:
Exception in thread "main"
java.lang.NullPointerException
   at testnull.main(testnull.java:7)
```

```
public String getLastNameFromStudentID(Sting id)
{
        if (!list.contains(id))
        {
                return null;
        }
        //...
}
```

Loops in Java

- **While loop:** test condition, if met execute a code block

```
int n = 1;
while (n < 10)
  {
        System.out.println(n);
        n = n + 5;
  }
```

Output

1

6

N = 6

Loops in Java

- **For loop:** three components: initializer, condition, and update

```
for (int n = 1; n < 10; n = n + 5)
{
        System.out.println(n);
}
```

- Think of it as a structured while loop

```
int n = 1;
while (n < 10)
{
        System.out.println(n);
        n = n + 5;
}
```

Java Arrays

```
int[] myarray = new int[10];
for (int i = 0; i < 10; i++)
{
        myarray[i] = i;
}
System.out.println(Arrays.toString(myarray));
double[] mynums = {3.1415,2.7182, 1.6180};
System.out.println(Arrays.toString(mynums));
```

Arrays are objects!

Built-in Code for Converting an Array to a String

Specify values to put in the array

Output

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[3.1415, 2.7182, 1.618]
```

More With Arrays

```
String[] myArray = new String[5];

for (int i = 0; i < myArray.length; i++)
{
        myArray[i] = "testvalue " + i;
}
System.out.print("[ ");
for (int i = 0; i < myArray.length; i++)
{
        System.out.print(myArray[i] + " ");
}
System.out.println("]");
```

Arrays can store other objects

Access number of elements

Index starting from 0

Output

```
[ testvalue 0 testvalue 1 testvalue 2 testvalue 3 testvalue 4 ]
```

Any idea what this the output of this program would be?

```
public class test {
        public static void main(String[] args)
        {
                String[] a = new String[10];
                System.out.println(a[0]);
        }
}
```

```
Output:

null
```

Outline for Today

- Continuing to get to know Java (review + Loops and Arrays)
- **Our first Java Data Structures: Stacks and Queues**
- Graph Searching with Stacks and Queues

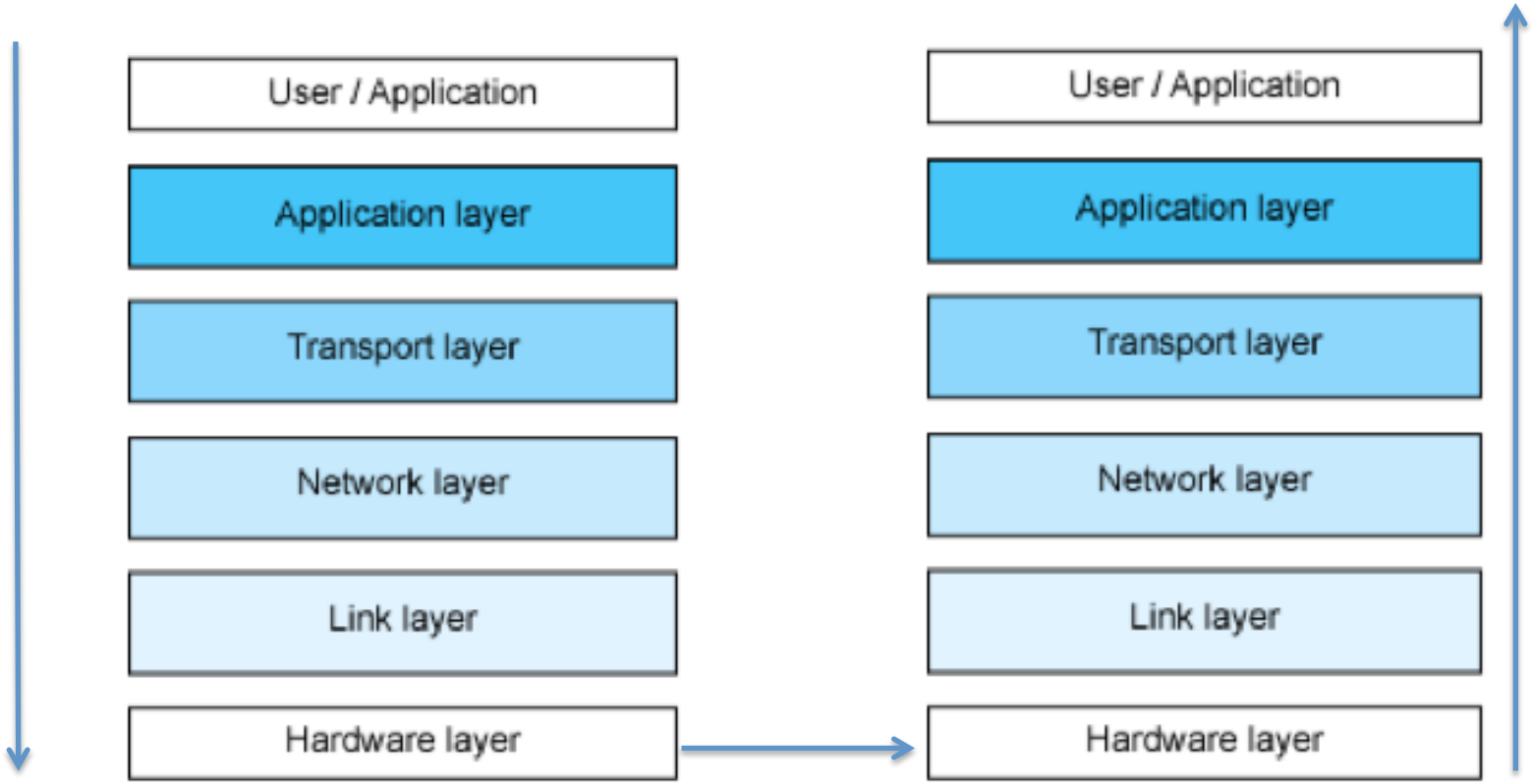
Abstract Data Types in Java

- **Theme of this course:** separating interface and implementation is a great idea
- Today we will see some mechanisms in Java to support this idea

Example of Abstraction in Computing: Networking

Joe sends an e-mail to Marissa

Marissa receives e-mail

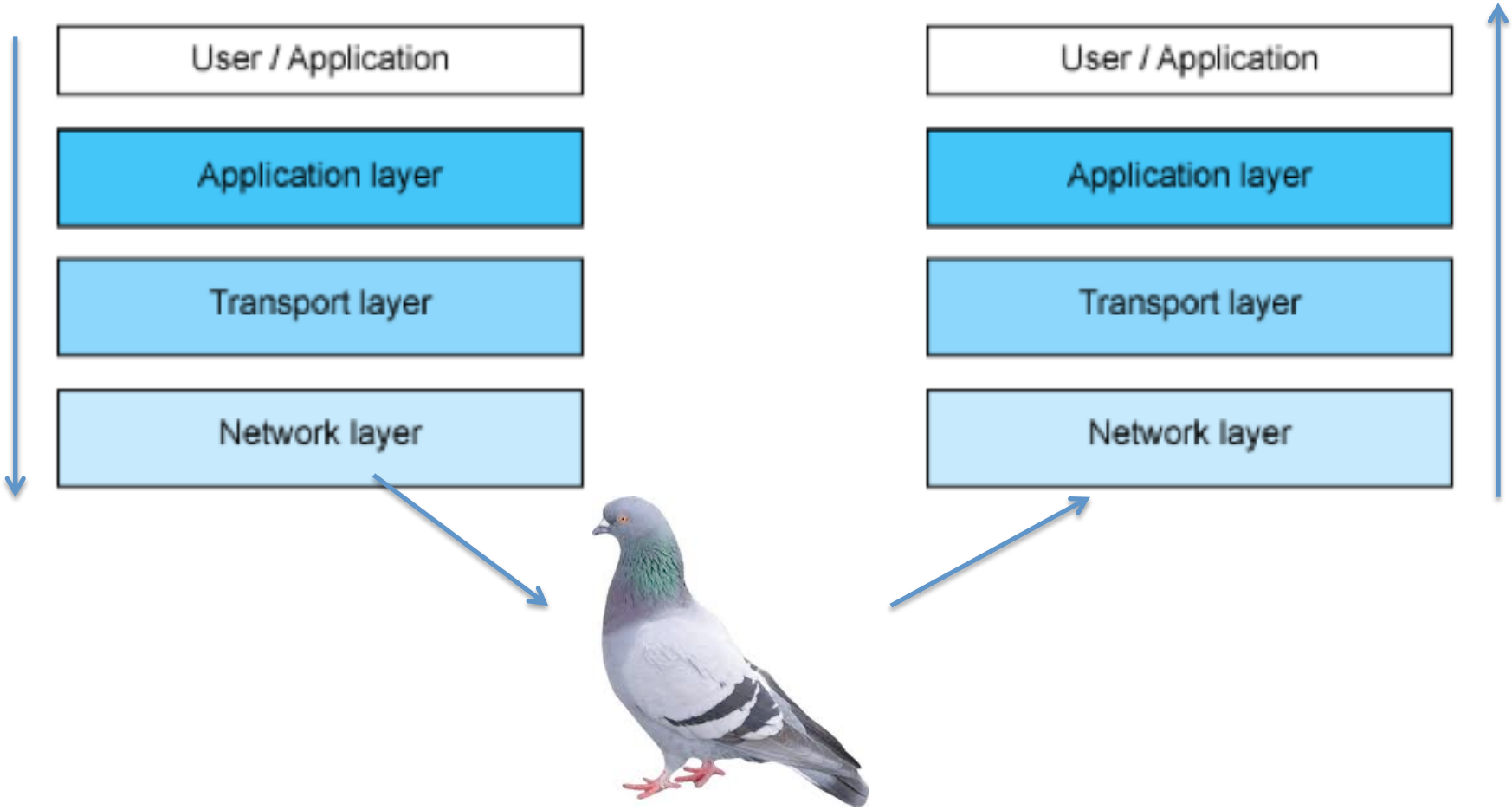


Fiber Optics / Coaxial Cable / Wireless / Satellite, etc.

An Alternate Proposal

Joe sends an e-mail to Marissa

Marissa receives e-mail



Abstraction Gone Bad? IP Over Pigeon

"Avian carriers can provide high delay, low throughput, and low altitude service. The connection topology is limited to a single point-to-point path for each carrier, used with standard carriers, but many carriers can be used without significant interference with each other, outside of early spring. This is because of the 3D ether space available to the carriers, in contrast to the 1D ether used by IEEE802.3. The carriers have an intrinsic collision avoidance system, which increases availability. Unlike some network technologies, such as packet radio, communication is not limited to line-of-sight distance. Connection oriented service is available in line-of-sight distance. Connection oriented service is available in some cities, usually based upon a central hub topology."

IETF Request for Comments: 1149

Benefits of Abstraction in Computer Programming

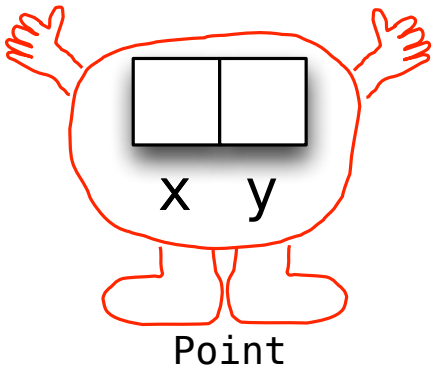
- **Managing Complexity:** focus on one part of the problem at a time
- **Division of labor:** allocate tasks in a large project
- **Helps us write shareable and maintainable code**

Java's Support for Abstraction

Encapsul the ways i can be used w e non- without br that was w

Not so fast!
test.java:6: y has private access in Point

Now to access the "y" coordinate directly!

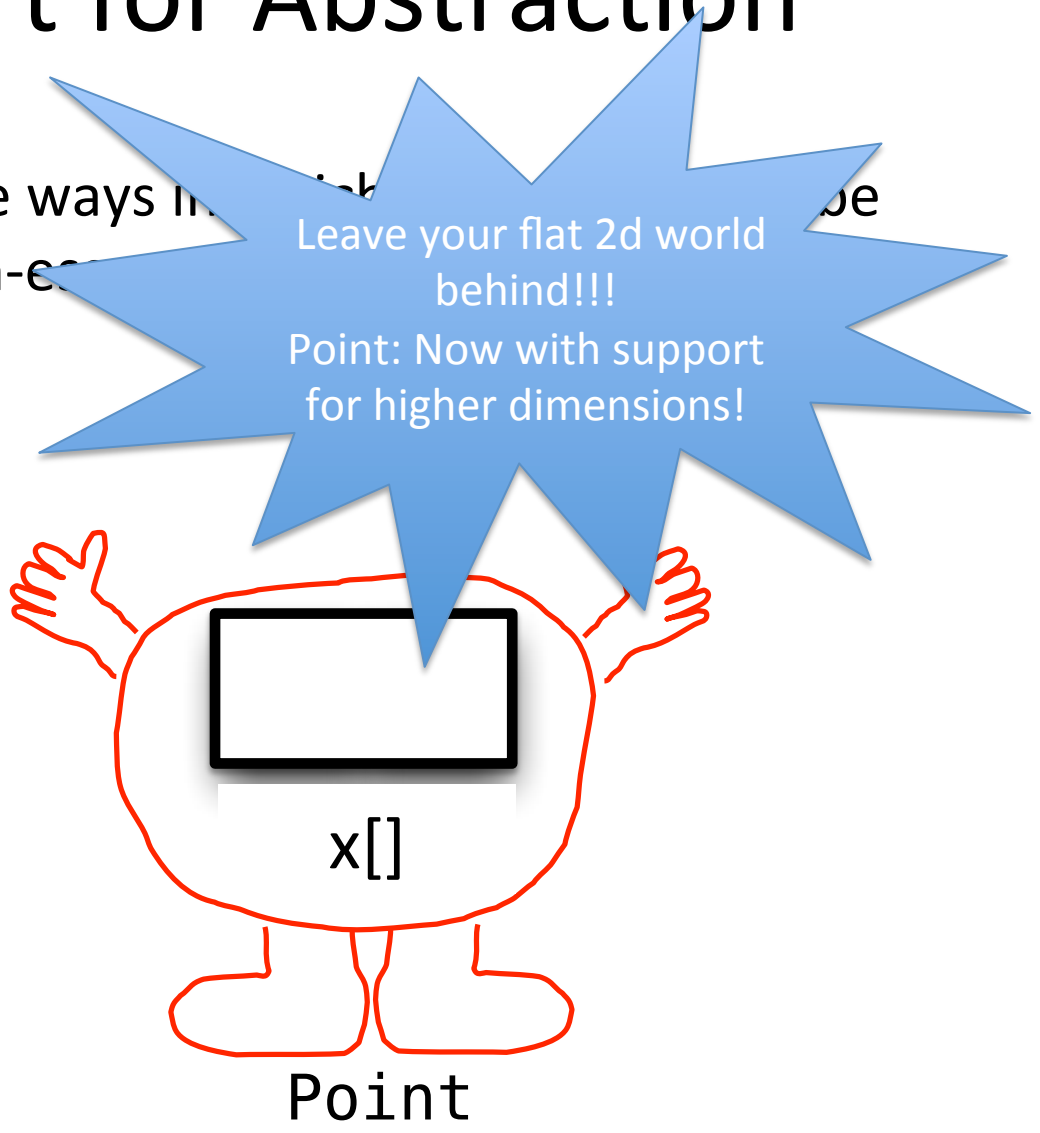


Java's Support for Abstraction

Encapsulation: restricting the ways in which [illegible] be used allows us to change non-essential details of existing code

```
public void setY(double y_in)
{
      y = y_in;
}
```

```
public void setY(double y_in)
{
      x[1] = y_in;
}
```



Our First Java Data Structure: Stack

- **What is a stack?** A data structure that allows data to be inserted and removed
- The only item that can be removed is the most recently inserted item (LIFO: Last in First Out)
- **Examples in C.S.:** parsing, recursion, artificial intelligence

Metaphor

Defining the Methods of a Stack

Let's create a Stack that holds Strings

```
public class StringStack
{
        ...
        public StringStack() { ... }
        public String pop() { ... }
        public String peek() { ... }
        public void push(String data) { ... }
        public boolean isEmpty() { ... }
        public String toString() { ... }
}
```

Constructor

Remove Item from Stack

View Item at the Top of the Stack

Place item on the Stack

Check if the Stack is Empty

Convert Stack to a String

Sample Code Using a Stack

```
public static void main(String[] args)
{
    StringStack st = new StringStack();

    st.push("1");
    st.push("2");
    st.push("3");
    st.push("4");

    System.out.println(st.pop());
    System.out.println(st.pop());

    st.push("5");
    System.out.println(st.pop());
}
```

st (top)

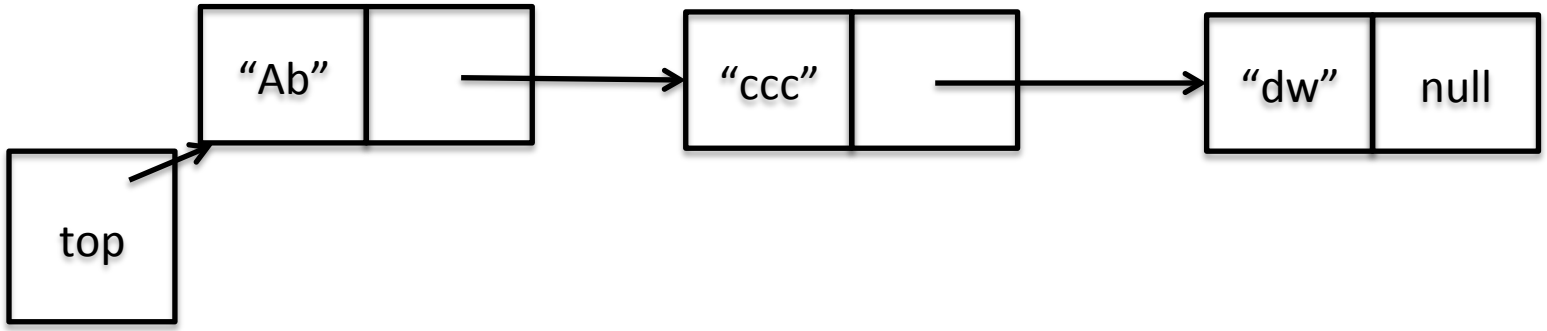
```
Output:
4
3
5
```

Implementing a Stack: What Underlying Data Structure Should We Use?

- One option: array

  Problem: arrays have bounded size

- Another option: linked list



- **Pros:** can grow to an unbounded size, removing and inserting from the start of the list is very efficient
- However, this is **not the only choice!!!** Remember Stack specifies an interface not an implementation.

Linked List Implementation of a Stack

```
public class StringStack extends Object
{
        private class LinkedListCell
        {
                private String data;
                private LinkedListCell next;

                public LinkedListCell(String data_in,
                                      LinkedListCell next_in)
                {
                        this.data = data_in;
                        this.next = next_in;
                }
        }

        private LinkedListCell top;

        public StringStack()
        {
                this.top = null;
        }
```

Inner class Helps in Implementing LinkedList

Indicates an empty stack

Methods of the Stack: pop

Removes top Element of the Stack

```
public String pop()
{
        if (this.isEmpty())
        {
                return null;
        }


}
```

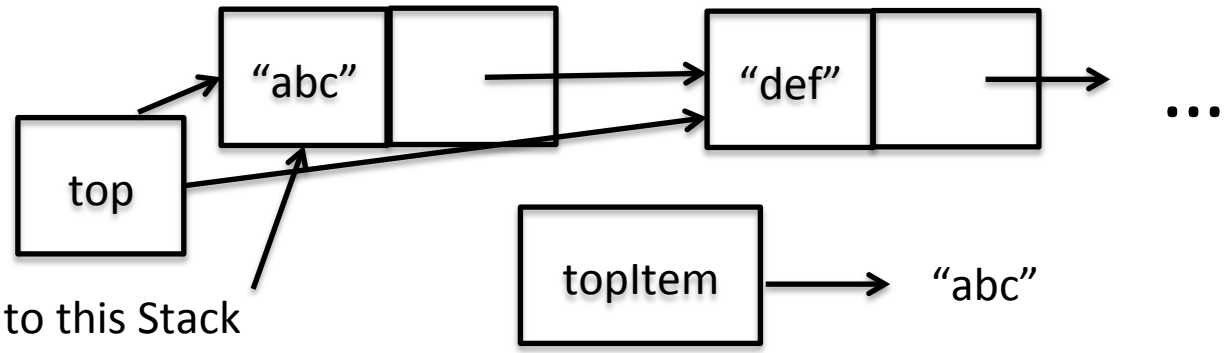
- ✓ StringStack()
- pop
- peek
- push
- isEmpty
- toString

Pop at Runtime

Removes top Element of the Stack

```
public String pop()
{
        if (this.isEmpty())
        {
                return null;
        }
        String topItem = this.top.data;
        this.top = this.top.next;
        return topItem;

}
```

✓ StringStack()
pop
peek
push
isEmpty
toString



What happens to this Stack Cell?

Methods of the Stack: peek

Returns top element of stack if it exists otherwise returns null

```
public String peek()
{
        if (this.isEmpty())
        {
                return null;
        }
        return this.top.data;
}
```

✓ StringStack()
✓pop
peek
push
isEmpty
toString

Methods of the Stack: push

Adds a String to the top of the stack

✓ StringStack()
✓ pop
✓ peek
push
isEmpty
toString

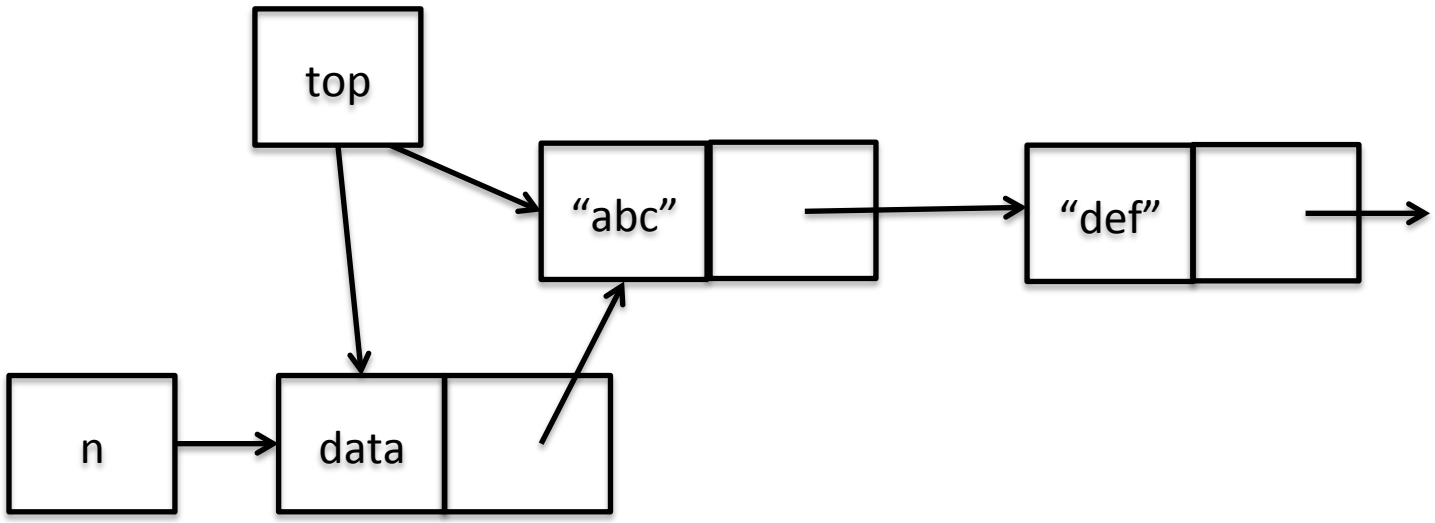
```
public void push(String data)
{
        LinkedListCell n = new LinkedListCell(data,this.top);

}
```

Push at Runtime

Adds a String to the top of the stack

✓ StringStack()
✓ pop
✓ peek
push
isEmpty
toString

```
public void push(String data)
{
    LinkedListCell n = new LinkedListCell(data,this.top);
    this.top = n;
}
```



Methods of the Stack: isEmpty

Returns true if and only if the stack is empty

```
public boolean isEmpty()
{
        return this.top == null;
}
```

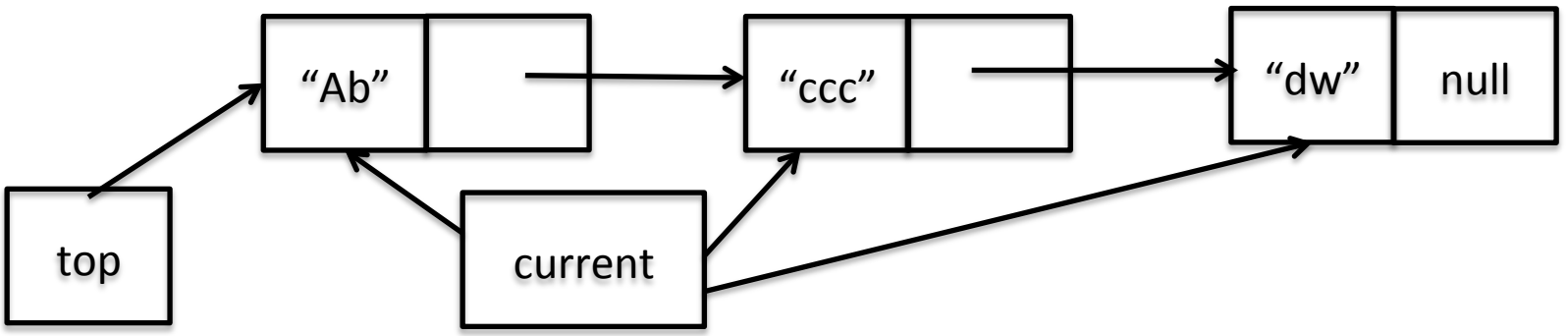
✓ StringStack()
✓ pop
✓ peek
✓ push
isEmpty
toString

Methods of the Stack: toString

Convert Stack to a String

```
public String toString()
{
        String stringValue = "[ \n";
        LinkedListCell current = this.top;
        while (current != null)
        {
                stringValue += " " + current.data + "\n";
                current = current.next;
        }
        stringValue  += "]";
        return stringValue;
}
```

✓ StringStack()
✓ pop
✓ peek
✓ push
✓ isEmpty
toString



Complexity of Methods for a Linked List Based Stack

- pop
- peek
- push
- isEmpty
- toString

Remember, we use **O()** to represent how the running time of each method depends on **"n"** (in this case "n" represents the number of elements currently on the stack)

Example: O() Pop

```
public String pop()
{
        if (this.isEmpty())
        {
                return null;
        }
        String topItem = this.top.data;
        this.top = this.top.next;
        return topItem;

}
```

The number of operations does not depend on the amount of data in the stack

Complexity of Methods for a Linked List Based Stack

- pop
- peek
- push
- isEmpty

} O(1)

- toString Ω(n)

Remember, we use O() to represent the running time of each method as a function of the amount of data in the stack

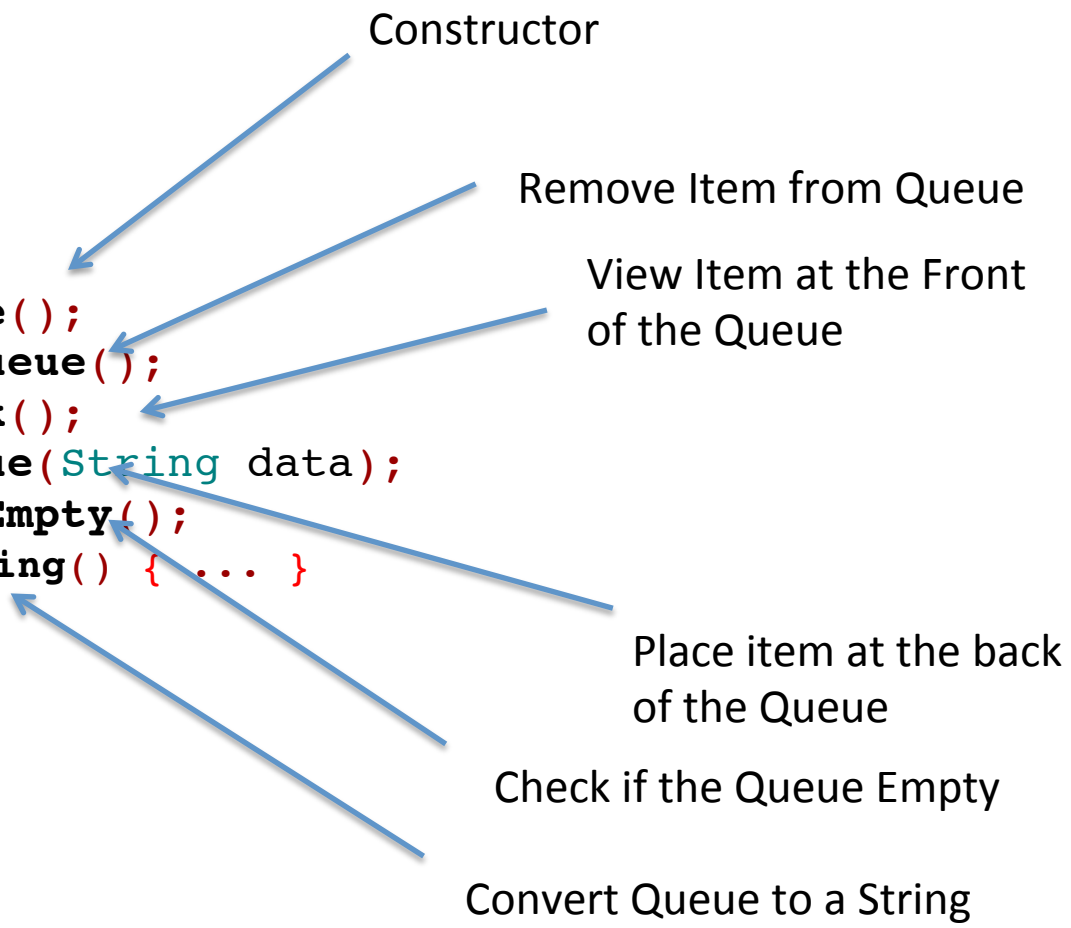
Queues

- **What is a queue?** A data structure that allows data to be inserted and removed
- The only item that can be removed is the least recently inserted item (FIFO: First in First Out)
- **Sample uses in C.S.:** network communication and process scheduling

Metaphor

What Methods Does a Queue Provide?

```
public class StringQueue
{
        ...
        public StringQueue();
        public String dequeue();
        public String peek();
        public void enqueue(String data);
        public boolean isEmpty();
        public String toString() { ... }
}
```



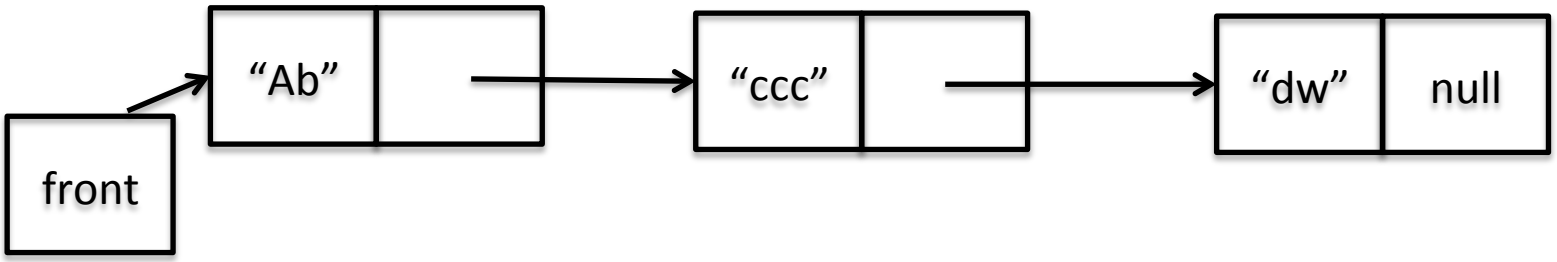
Sample Code Using a Queue

```
public static void main(String[] args)
{
        StringQueue q = new StringQueue();

        q.enqueue("1");
        q.enqueue("2");
        q.enqueue("3");
        q.enqueue("4");

        System.out.println(q.dequeue());
        System.out.println(q.dequeue());

        q.enqueue("5");
        System.out.println(q.dequeue());
}
```

Output:

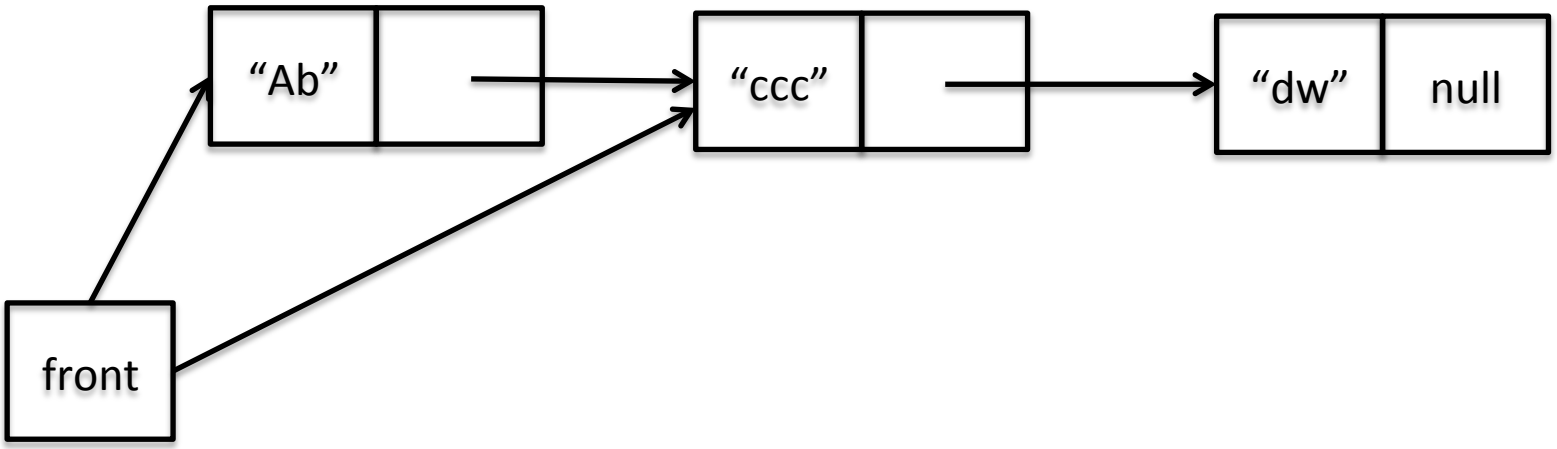
```
1
2
3
```

Implementing Queues with Linked Lists



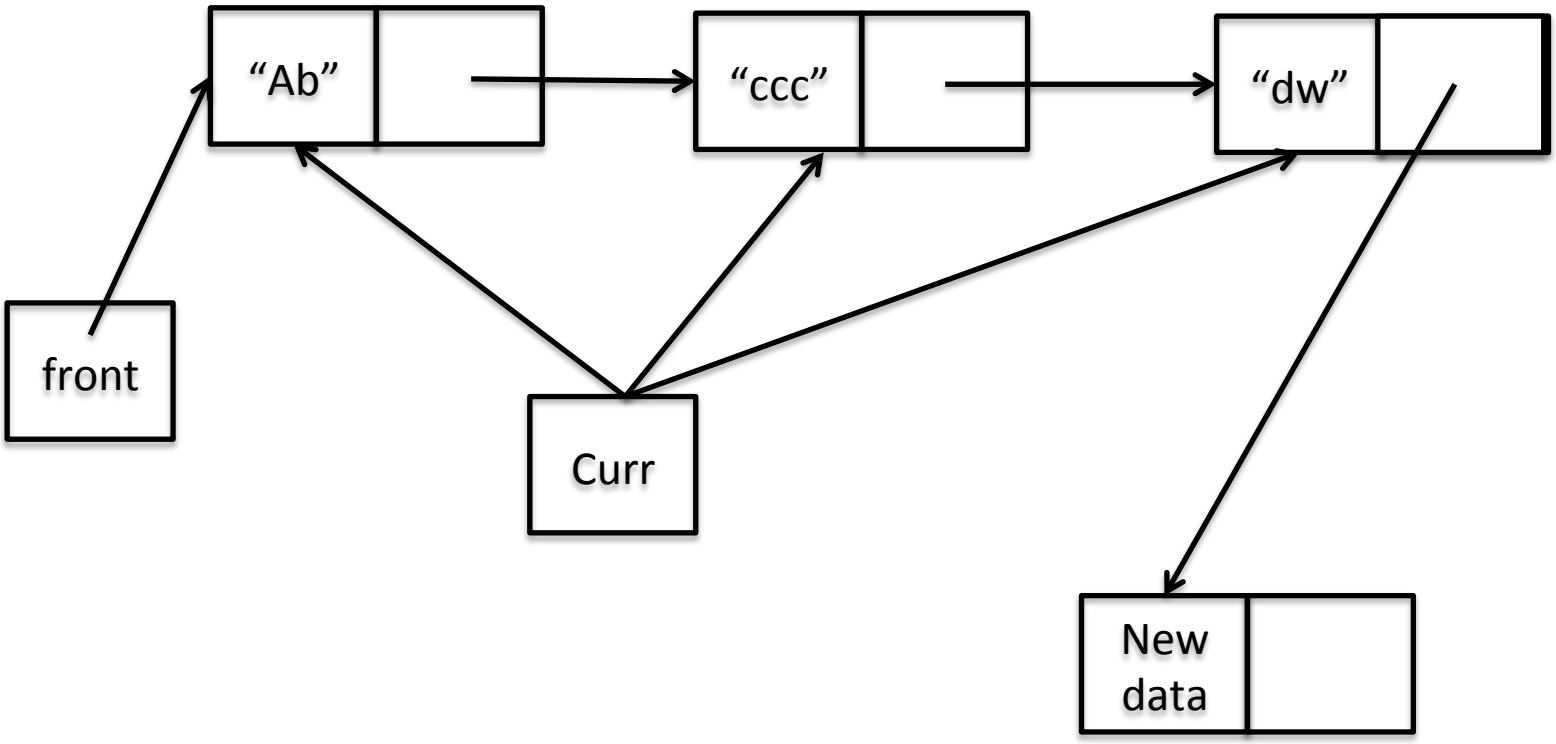
- Operations needed on the linked list
  - **dequeue:** remove from front
  - **enqueue:** add to back

Implementing Queues with Linked Lists: dequeue



Time Complexity: O(1)

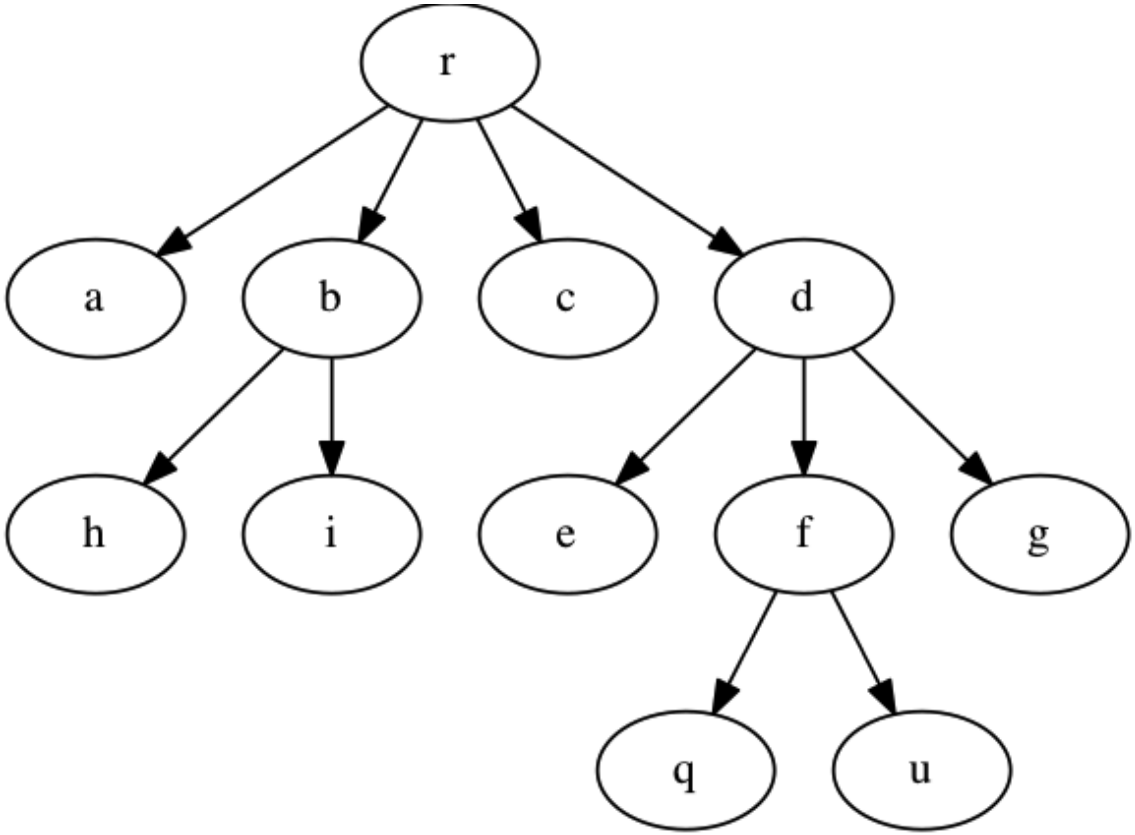
Implementing Queues with Linked Lists: enqueue



Time Complexity: O(n)

Graph Searching Revisited

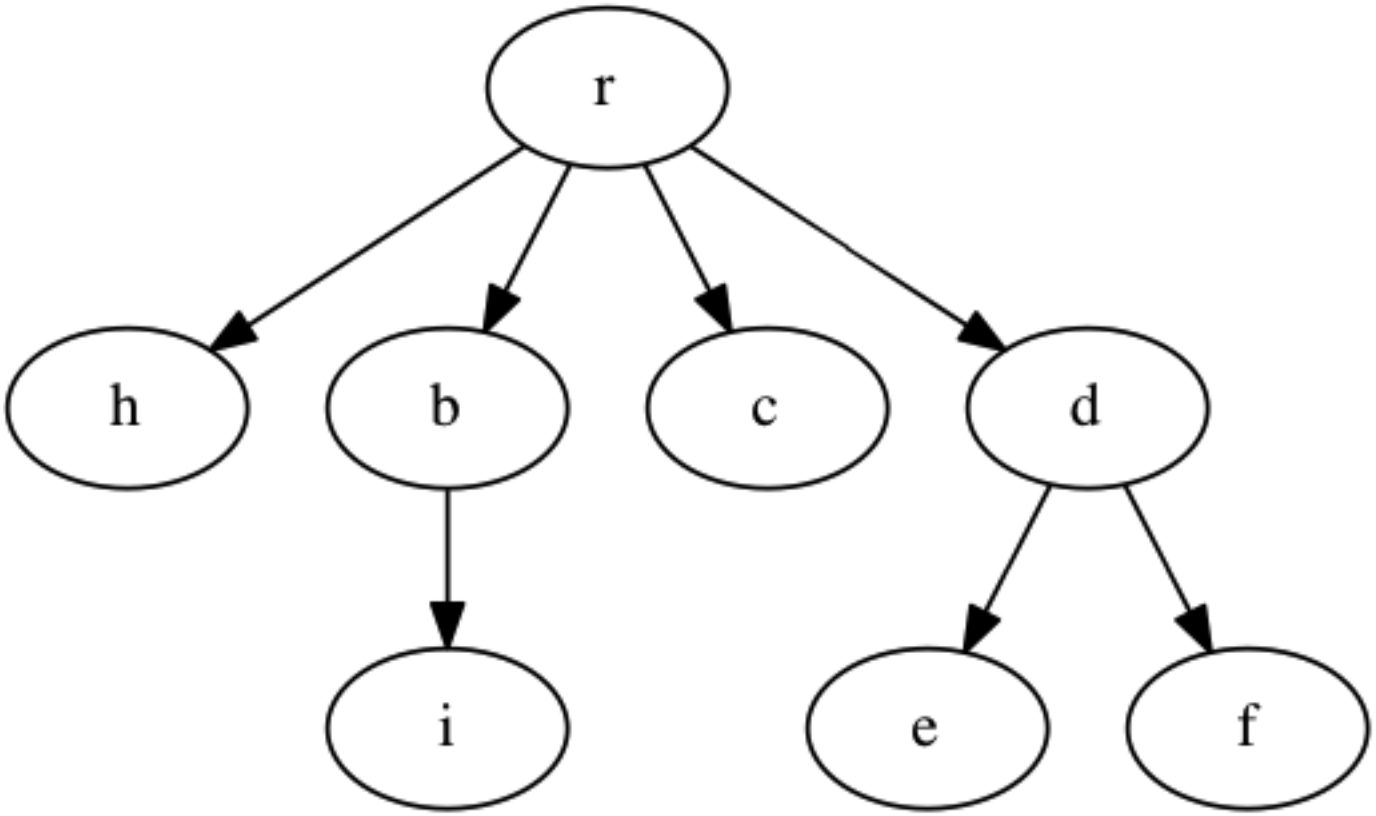
Recall: graphs consist of nodes (circles) and edges (connections)



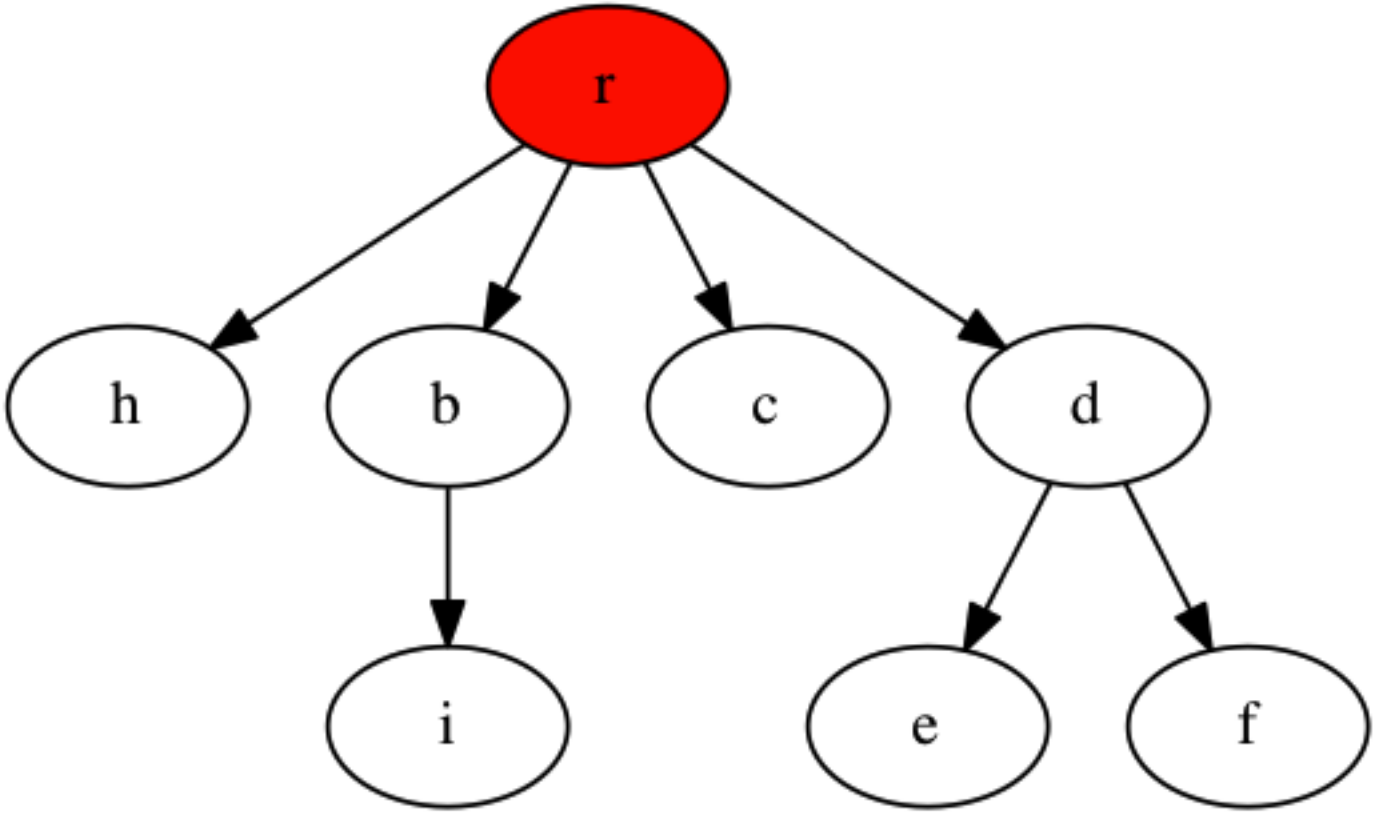
Searching a Graph

- **Objective:** Given a starting node find a target node
- Algorithms for solving the problem
  - Breadth First Search (BFS)
  - Depth First Search (DFS)

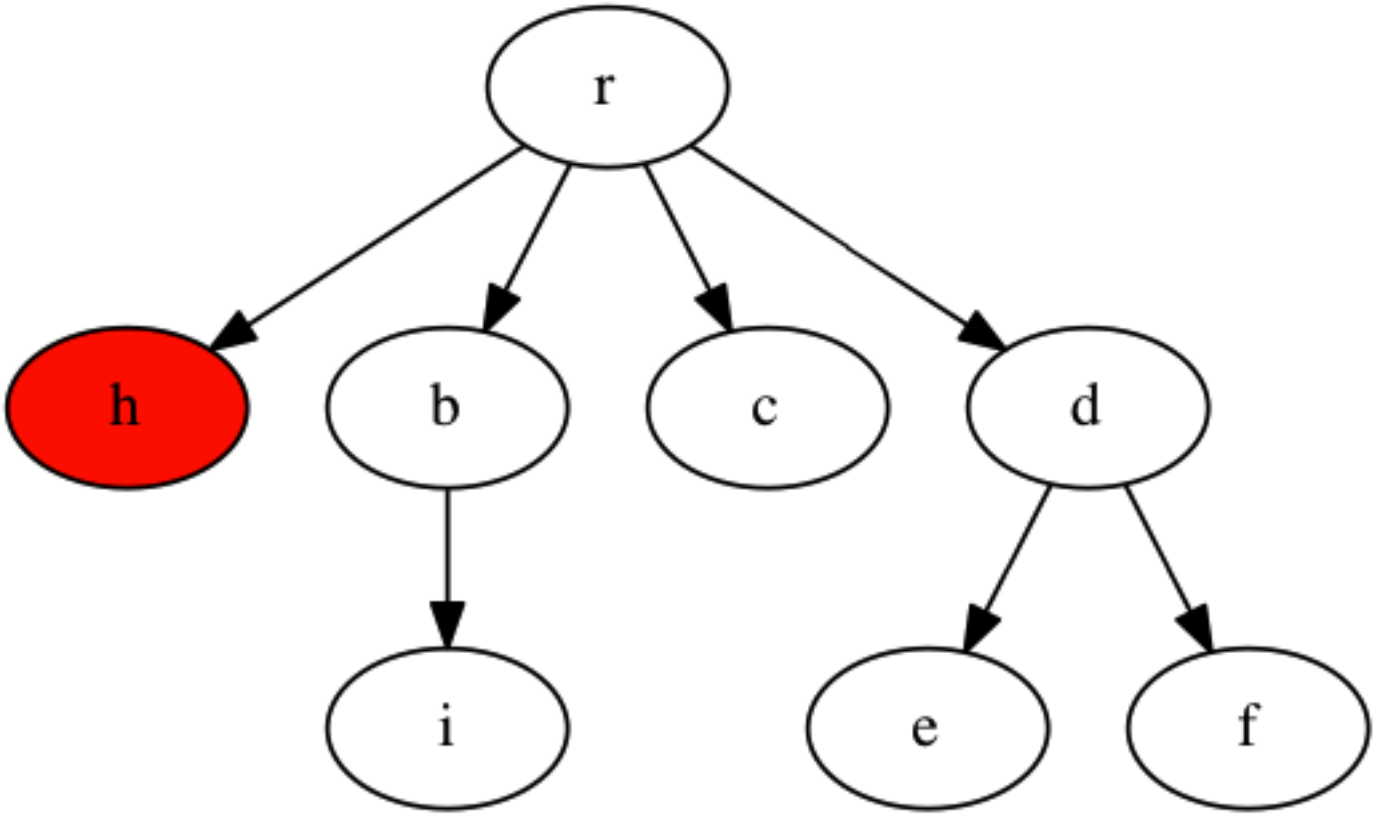
Example: Searching for c



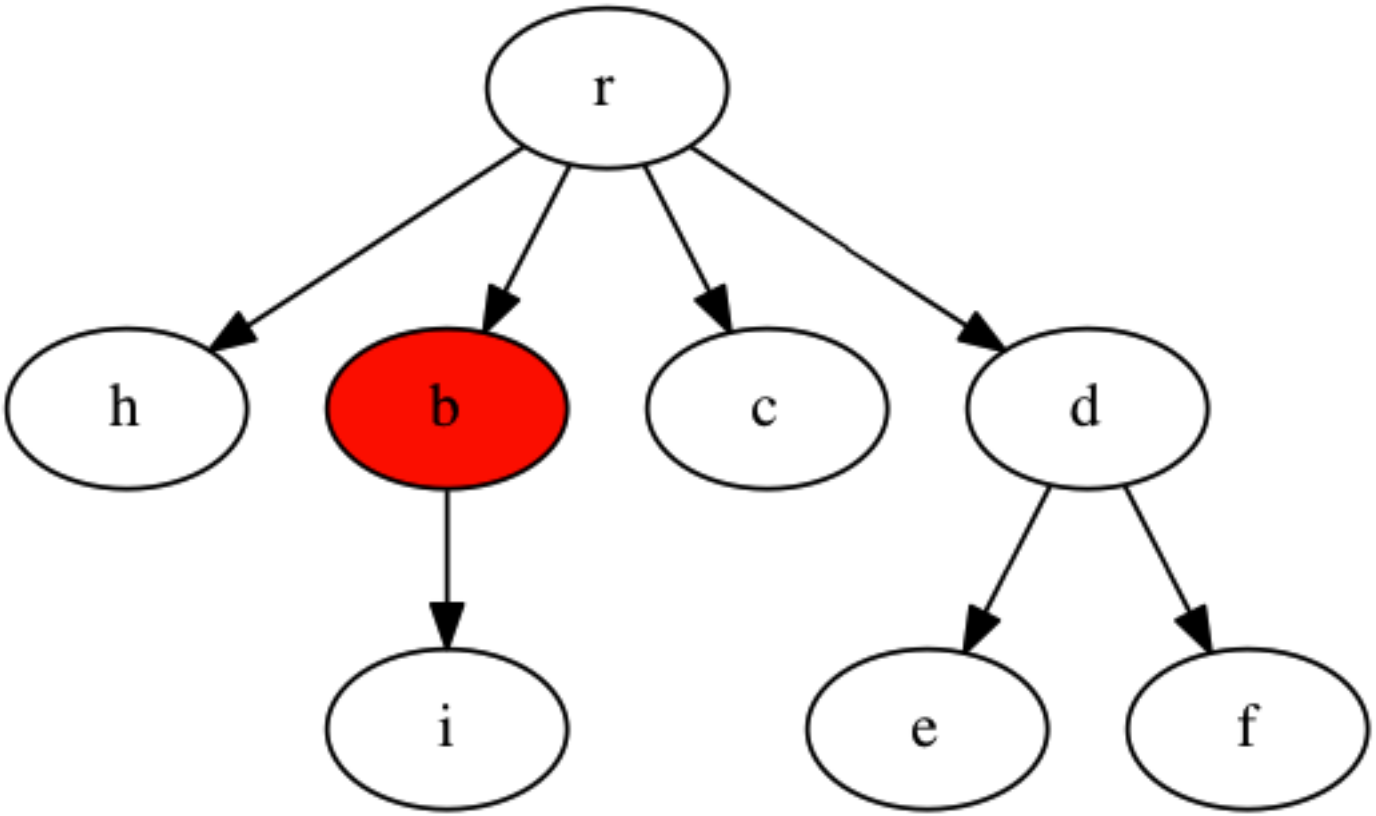
Example: Searching for c (BFS)



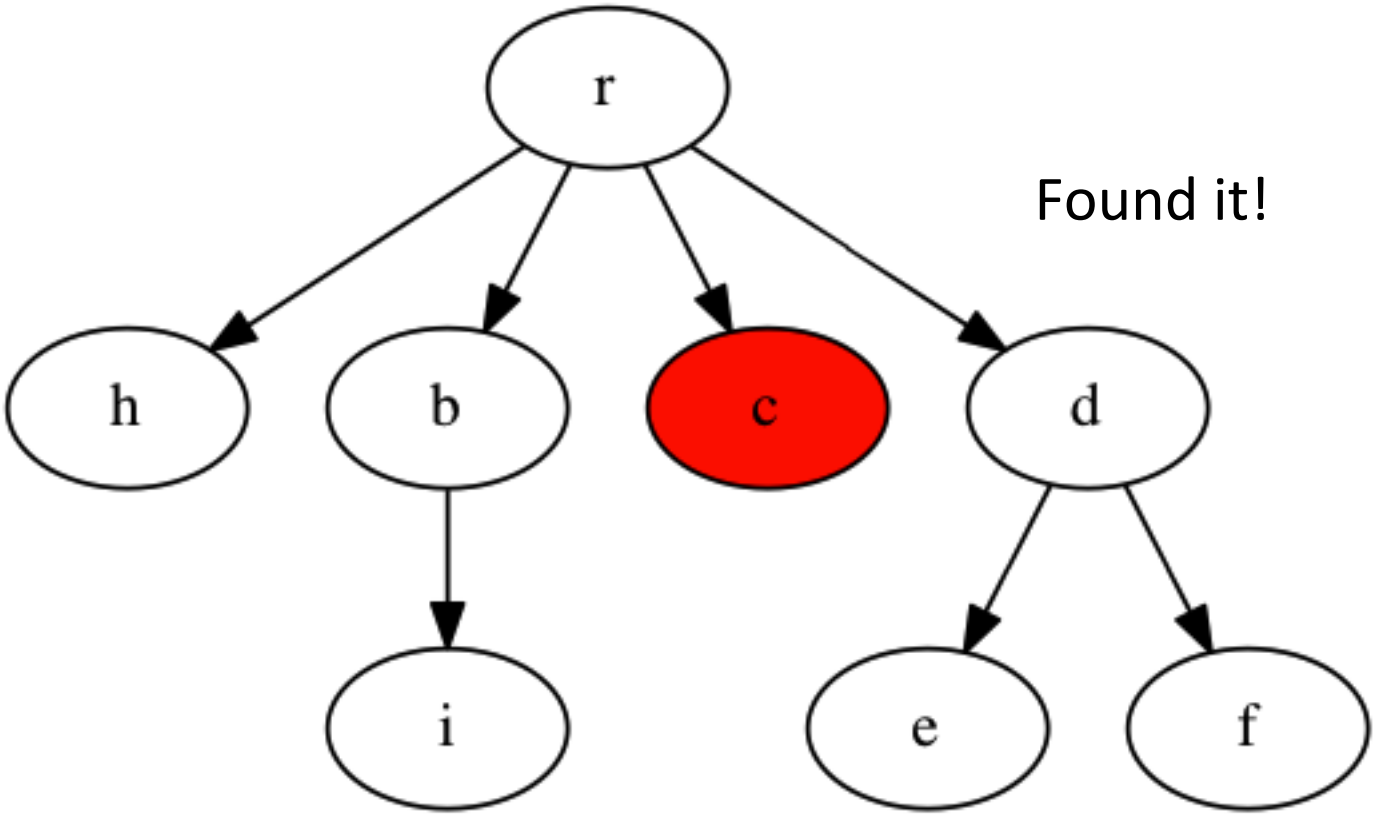
Example: Searching for c (BFS)



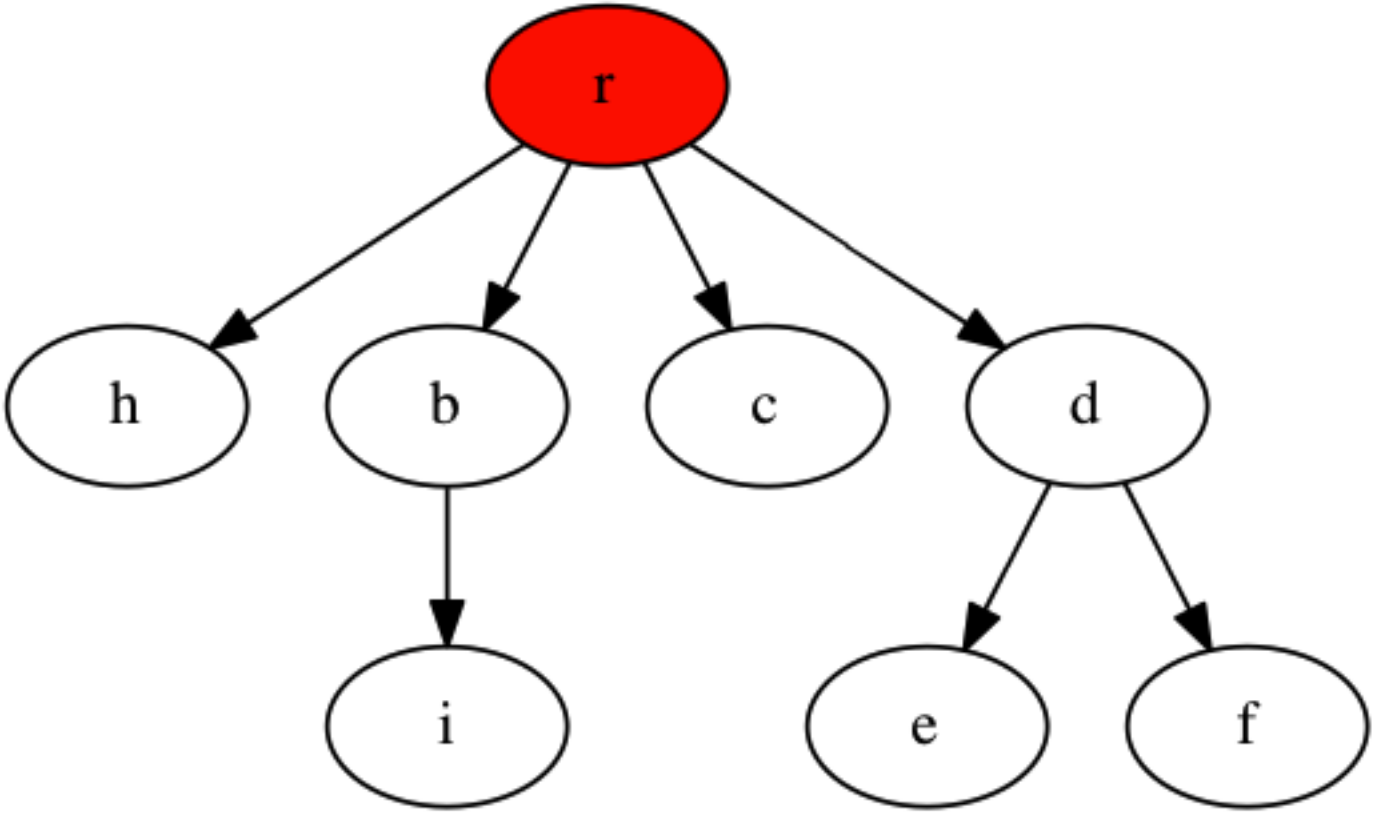
Example: Searching for c (BFS)



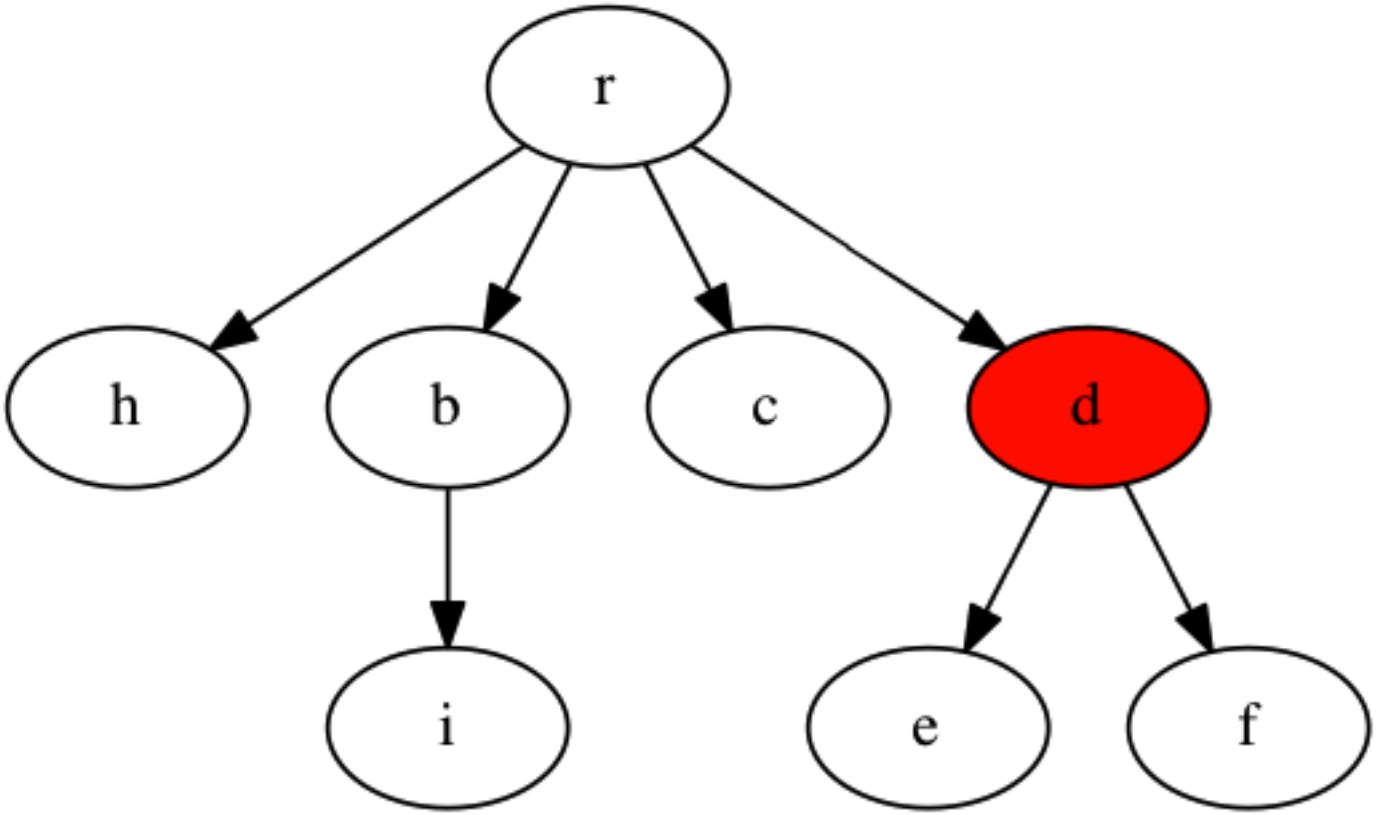
Example: Searching for c (BFS)



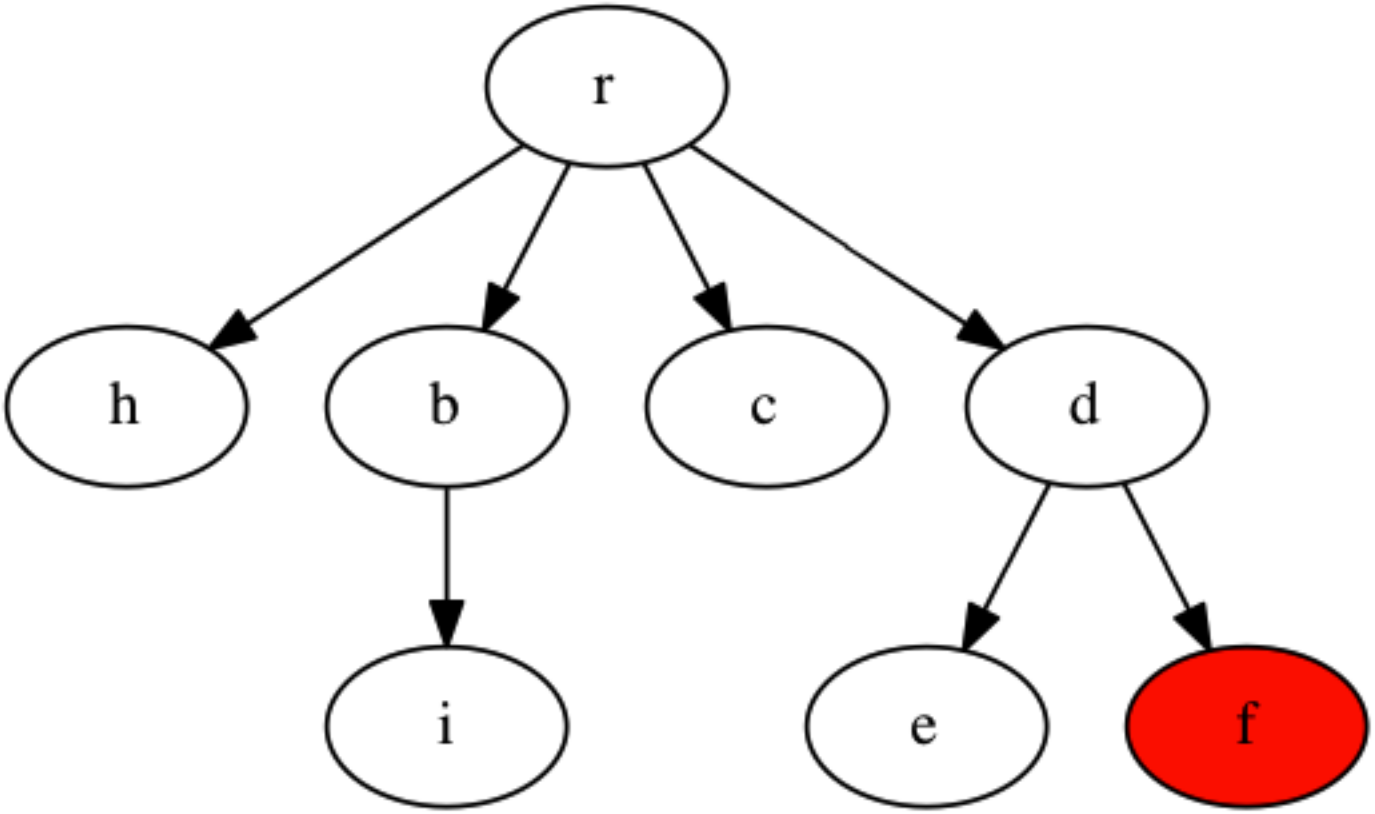
Example: Searching for c (DFS)



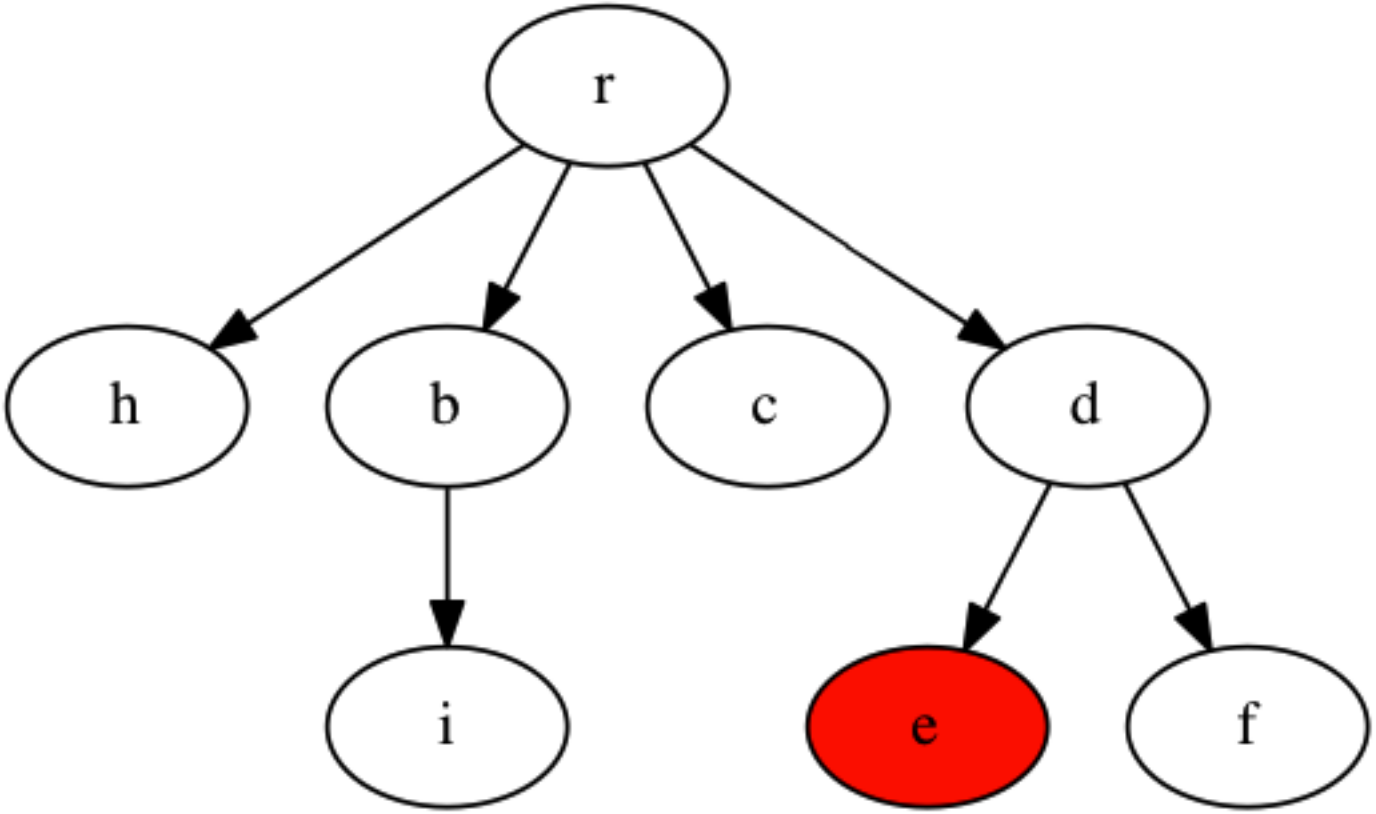
Example: Searching for c (DFS)



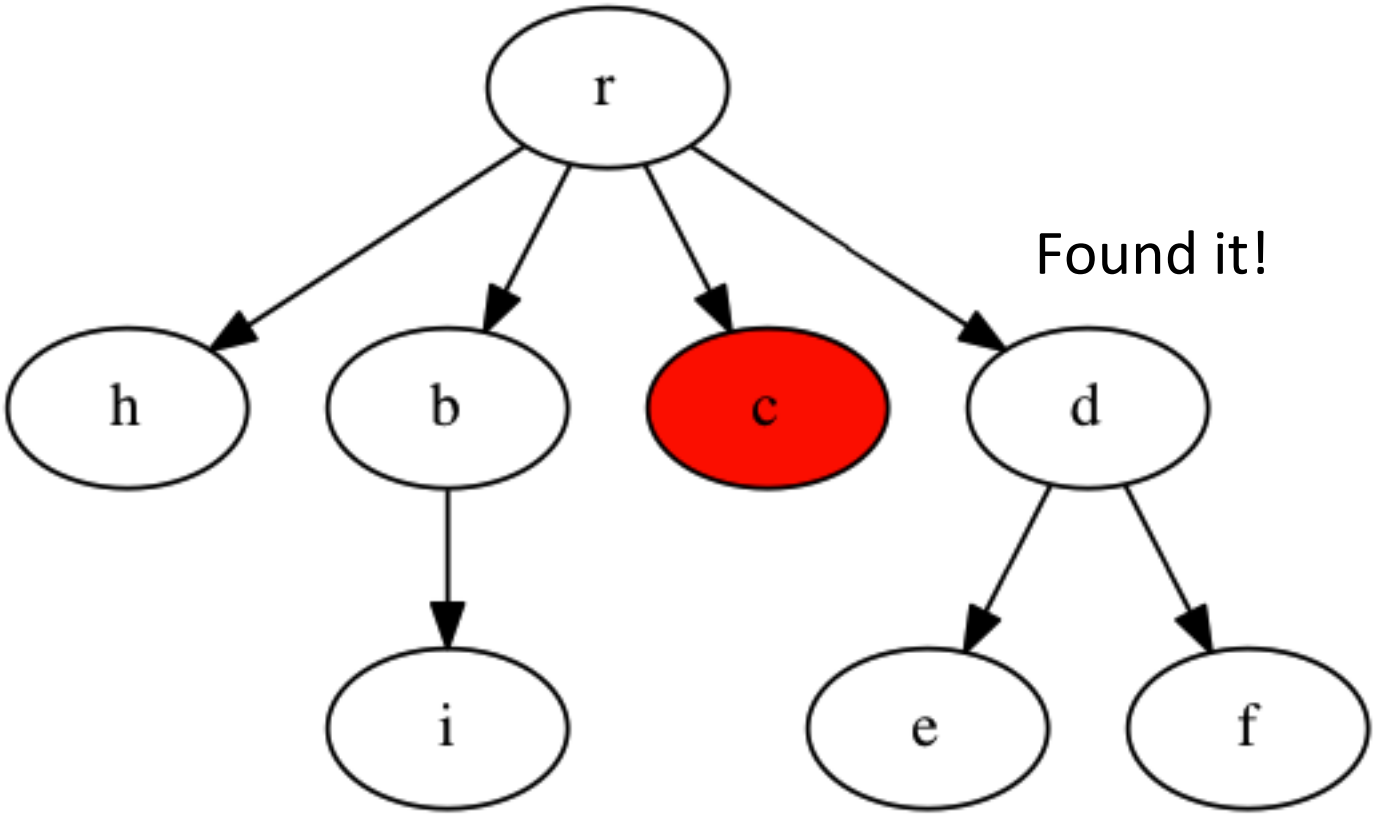
Example: Searching for c (DFS)



Example: Searching for c (DFS)



Example: Searching for c (DFS)



Let's Think About How We Might Implement These Algorithms

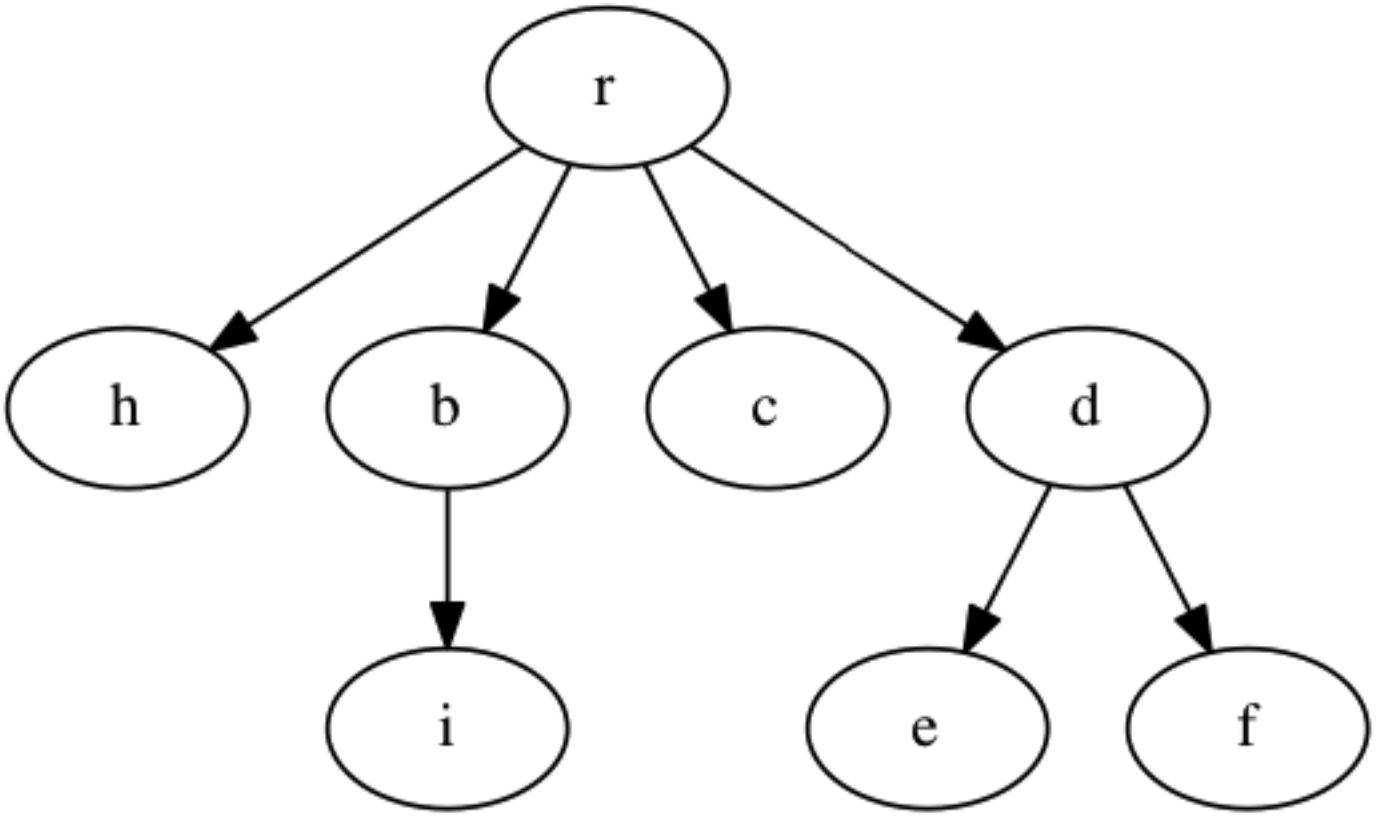
```
while areMoreNodesToSearch
    visit next node
    remove node from list of nodes to search
    add neighbors of current node to the list of nodes to search
```

We need to use a data structure to store the collection of nodes left to search

Which Data Structure is Appropriate for Which Algorithm?

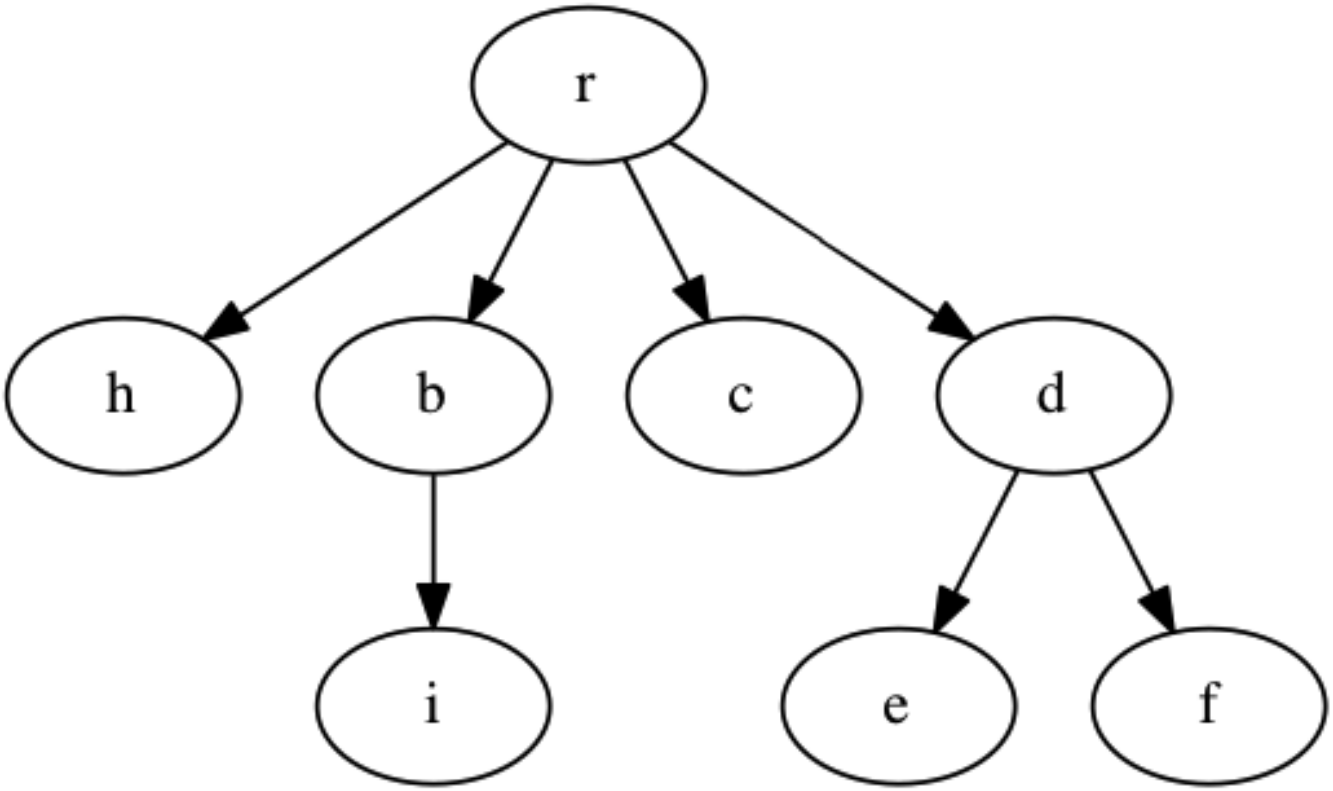
- BFS FIFO
  - At current node we have to wait for other nodes at current node's level to be visited before visiting the children of the current node
- DFS LIFO
  - Immediately visit the children of the current node

Example: Searching for c (BFS)

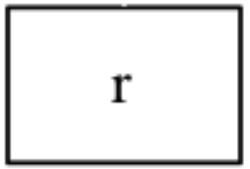


Queues to the Rescue!

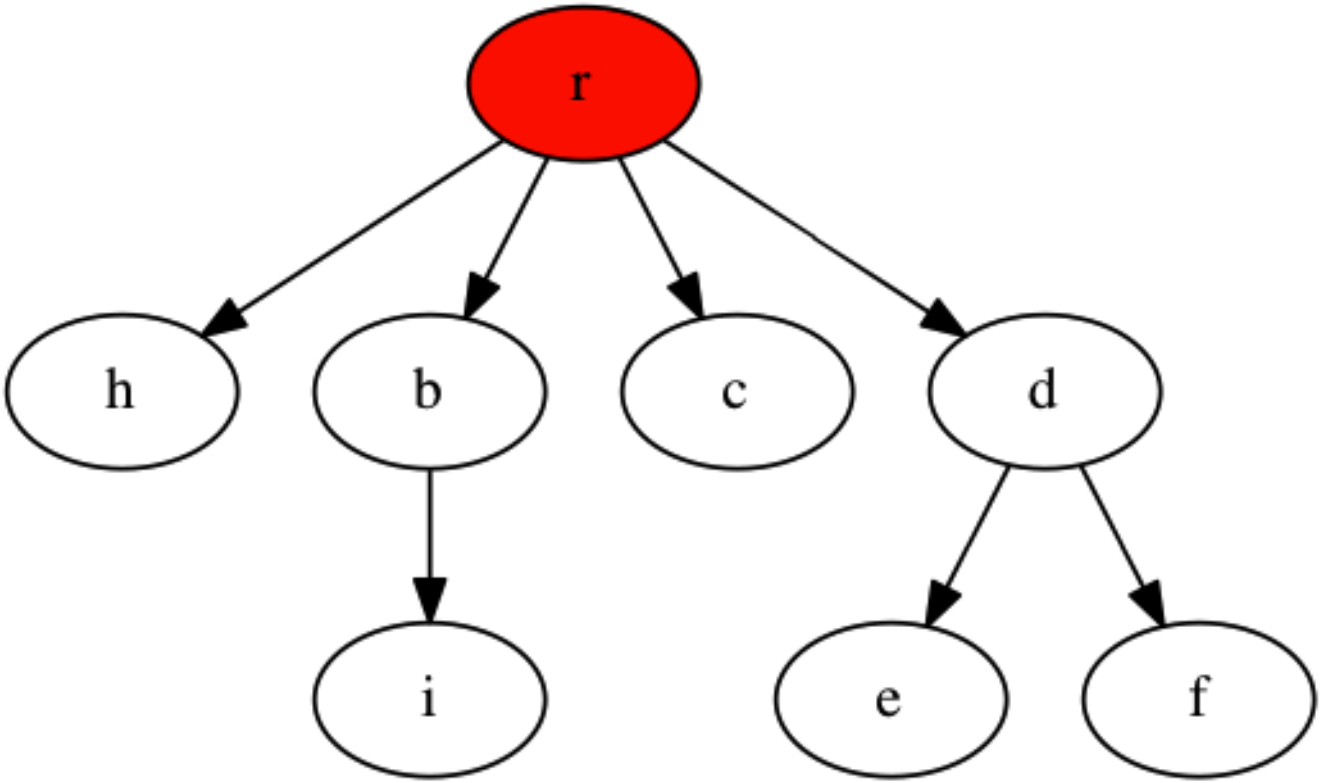
Example: Searching for c (BFS)



Queue (front)

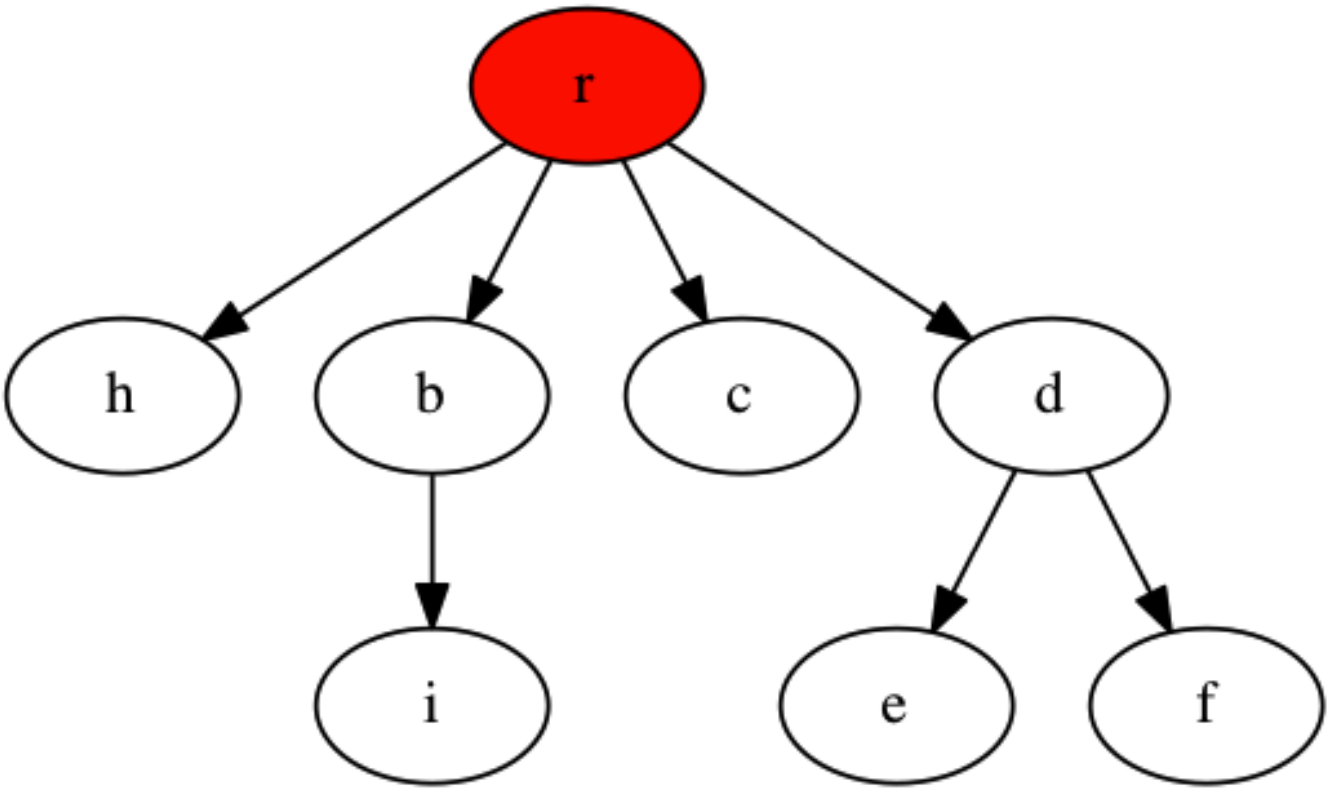


Example: Searching for c (BFS)

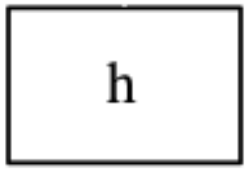


Queue (front)

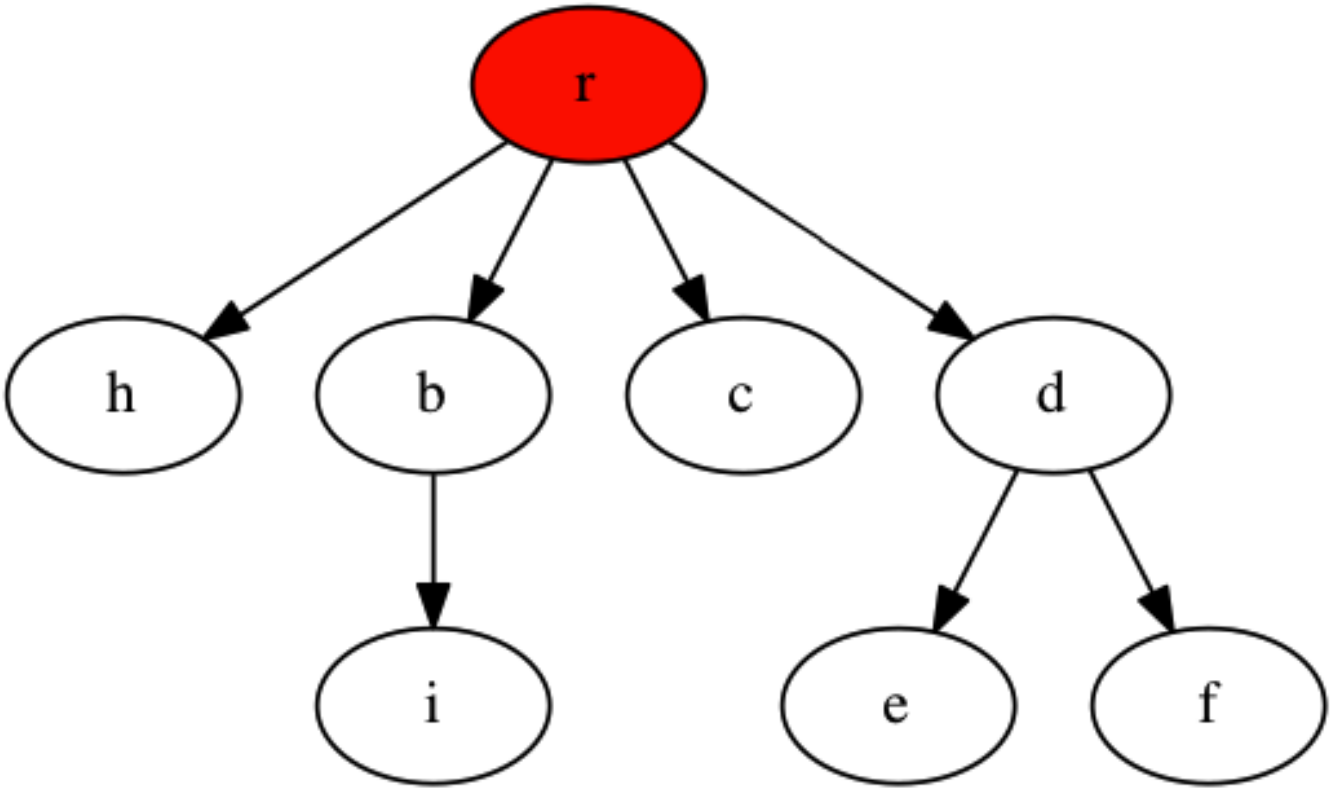
Example: Searching for c (BFS)



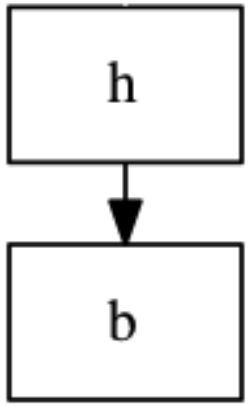
Queue (front)



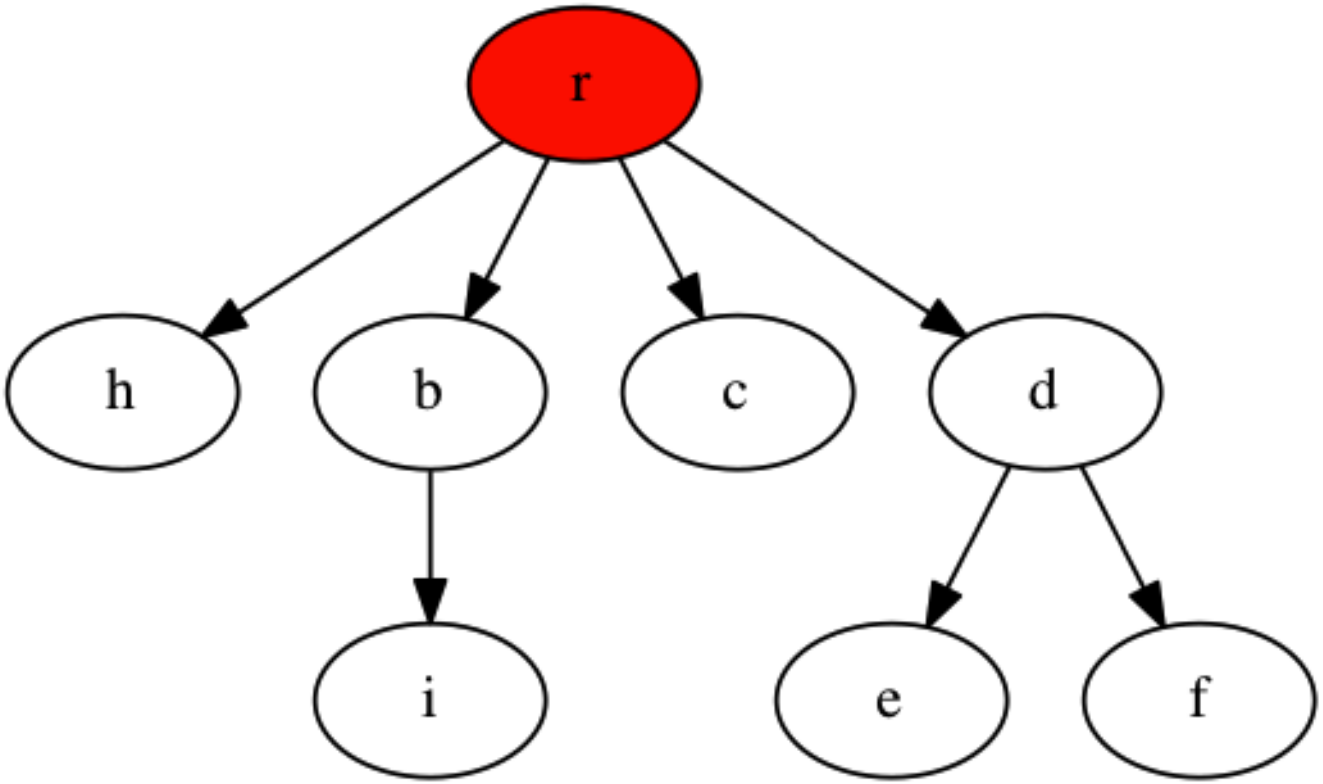
Example: Searching for c (BFS)



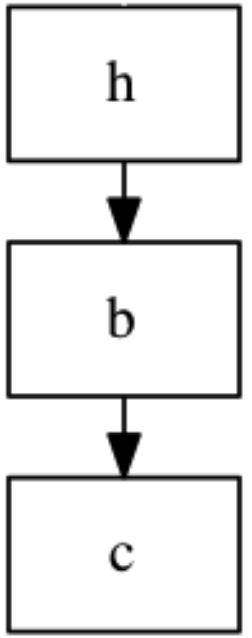
Queue (front)



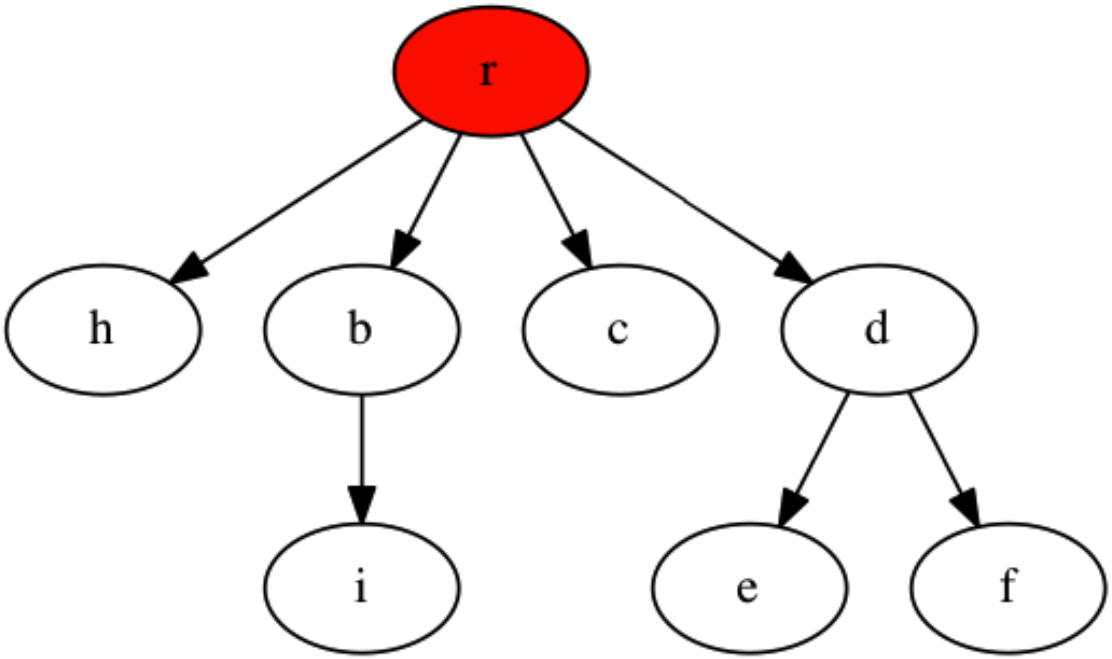
Example: Searching for c (BFS)

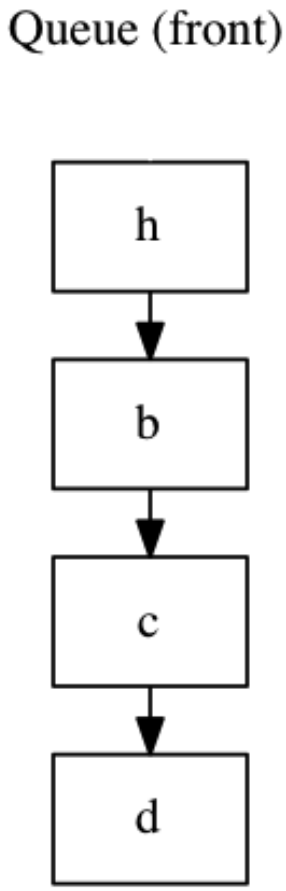


Queue (front)

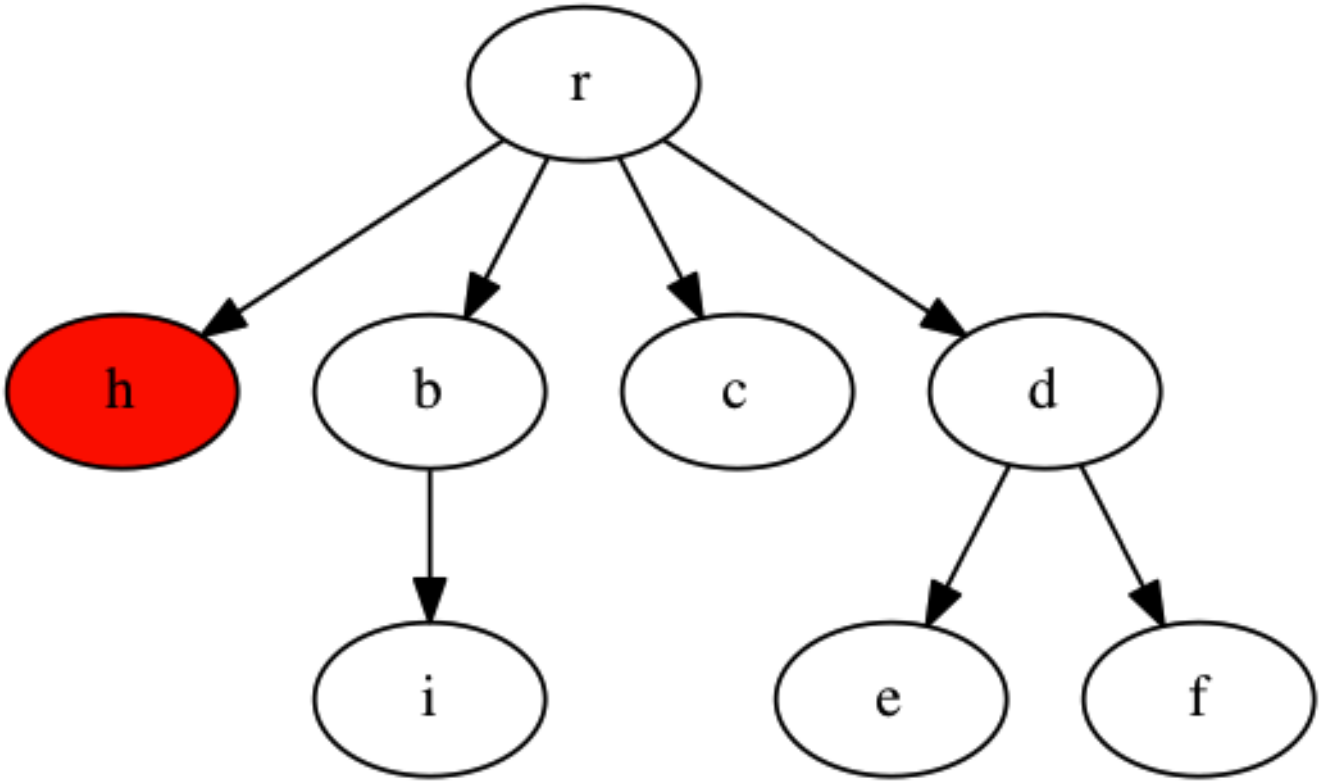


Example: Searching for c (BFS)

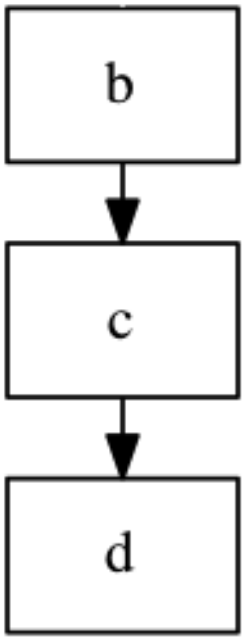




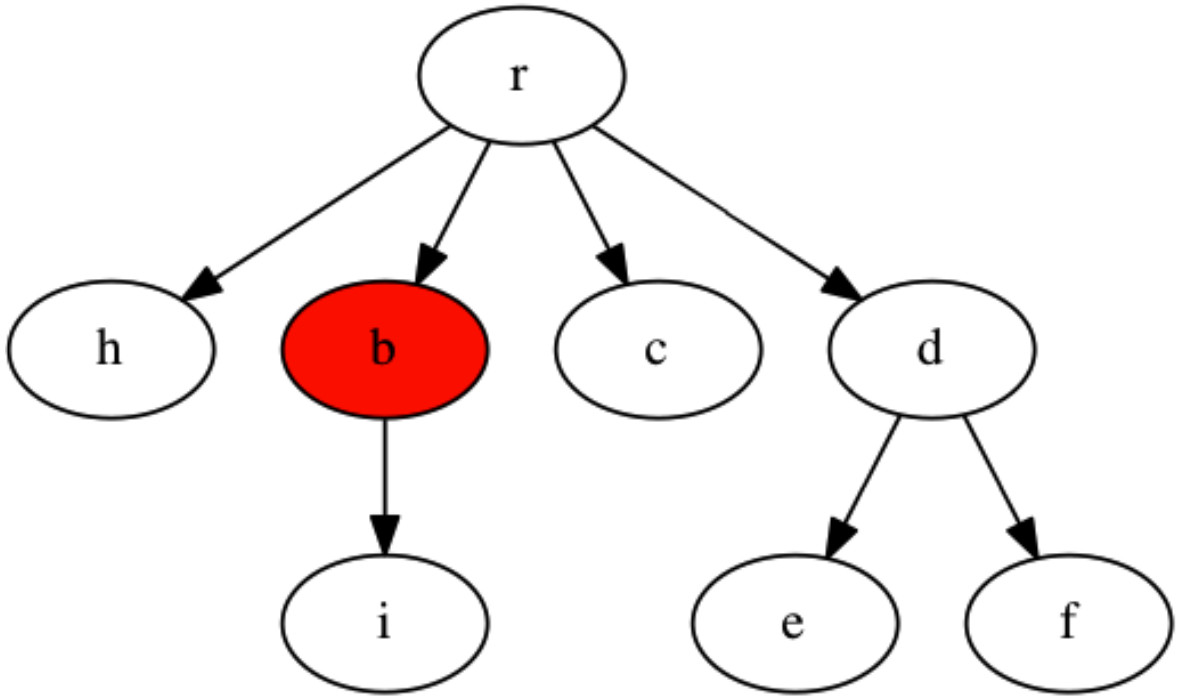
Example: Searching for c (BFS)

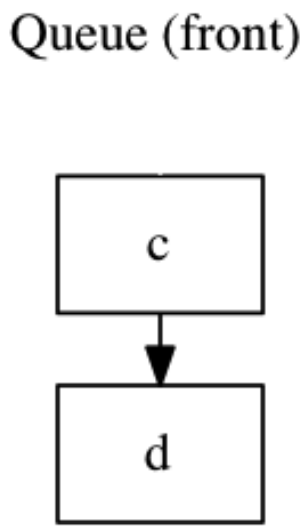


Queue (front)

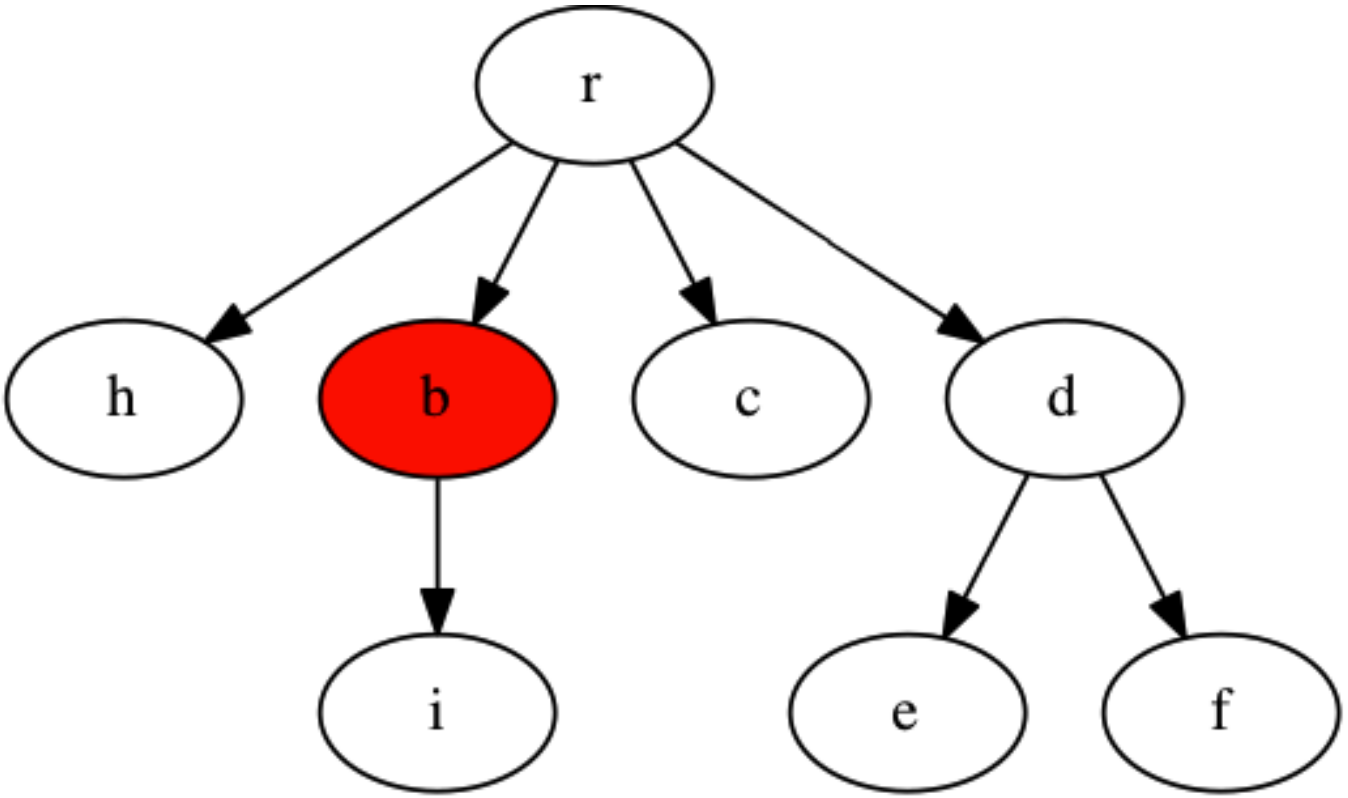


Example: Searching for c (BFS)

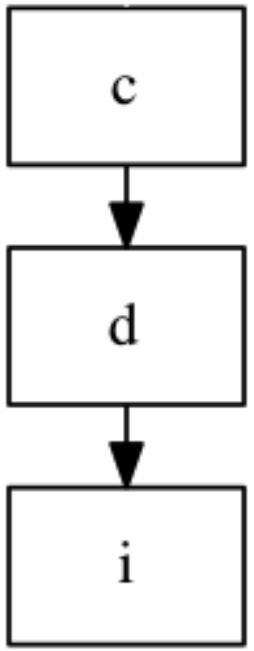




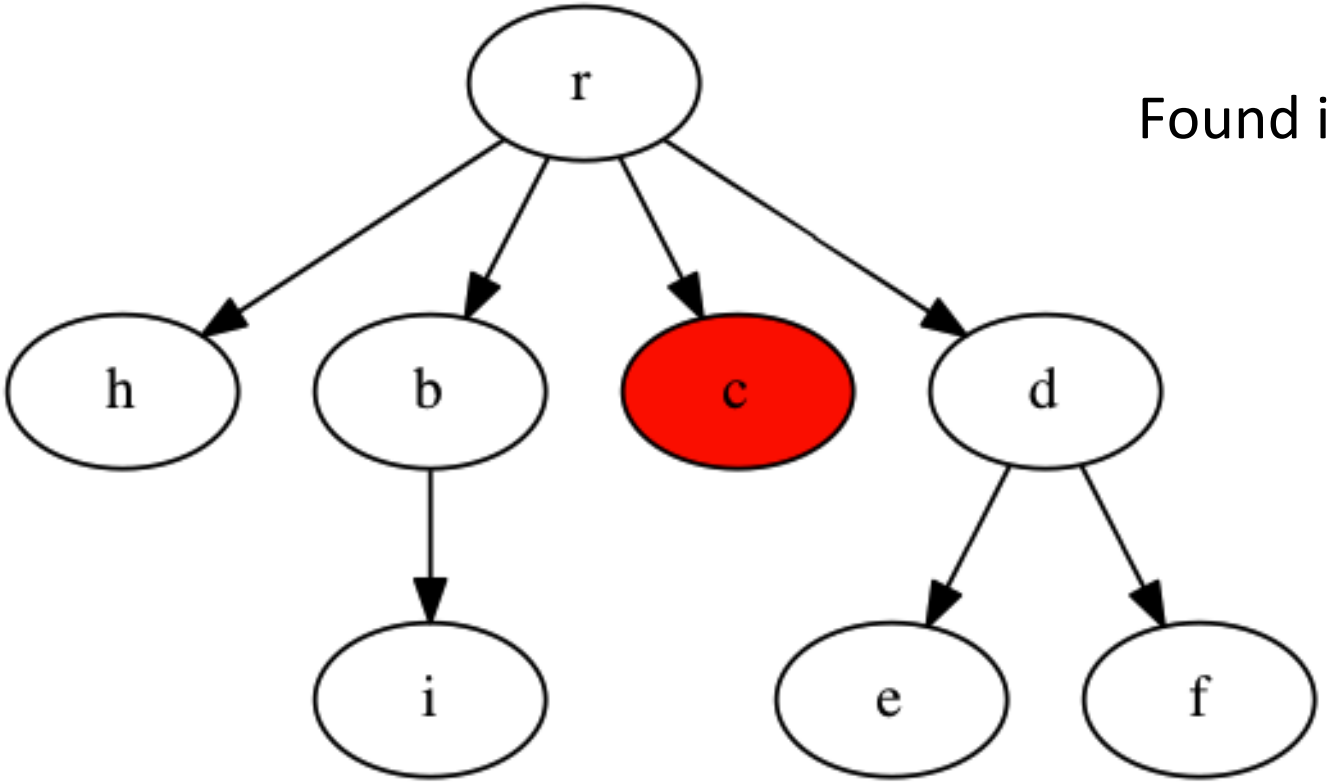
Example: Searching for c (BFS)



Queue (front)

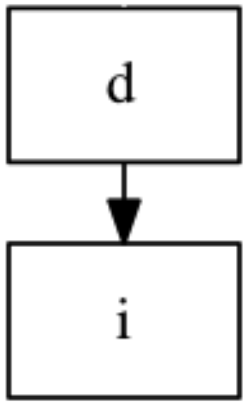


Example: Searching for c (BFS)

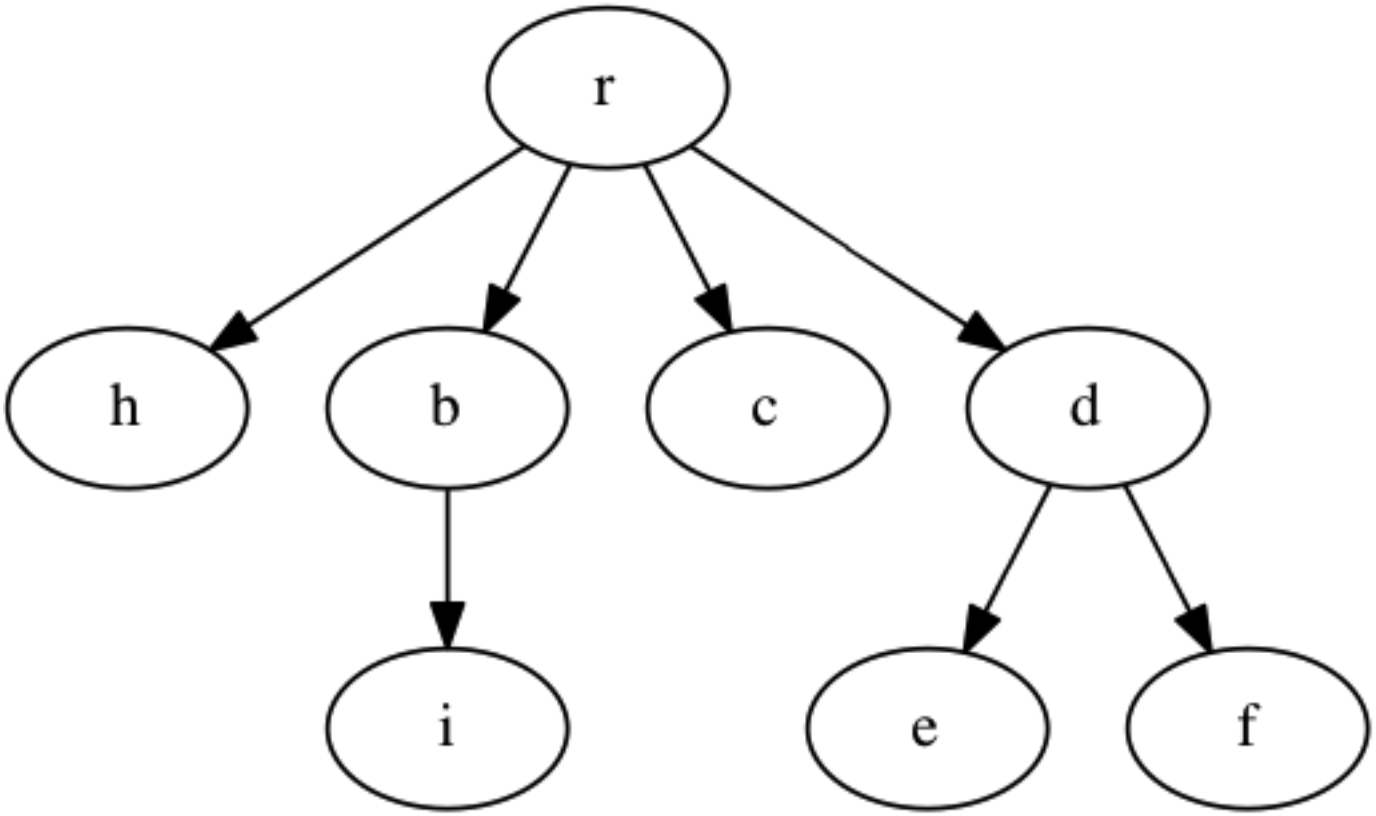


Found it!

Queue (front)

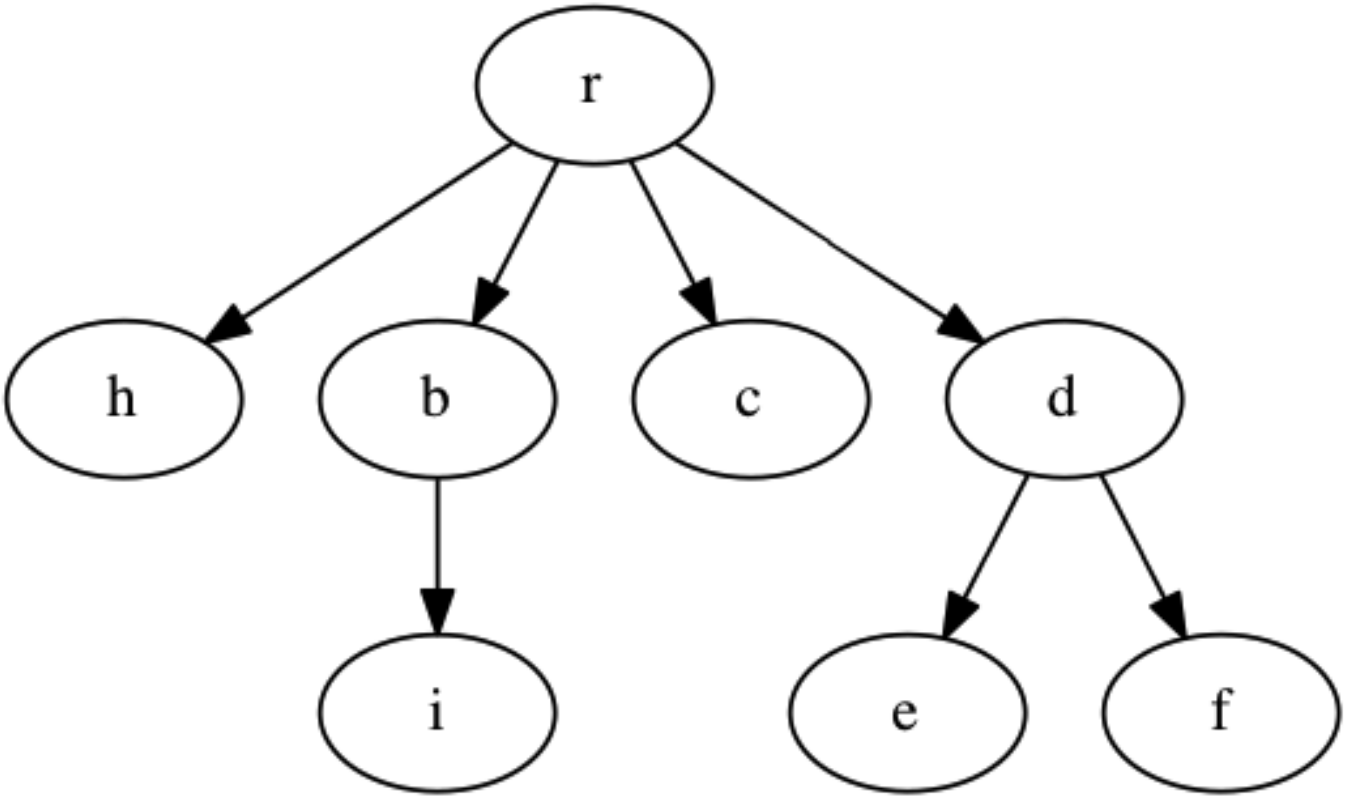


Example: Searching for c (DFS)

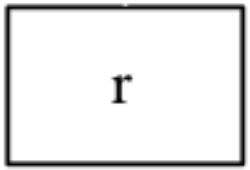


Stacks to the Rescue!

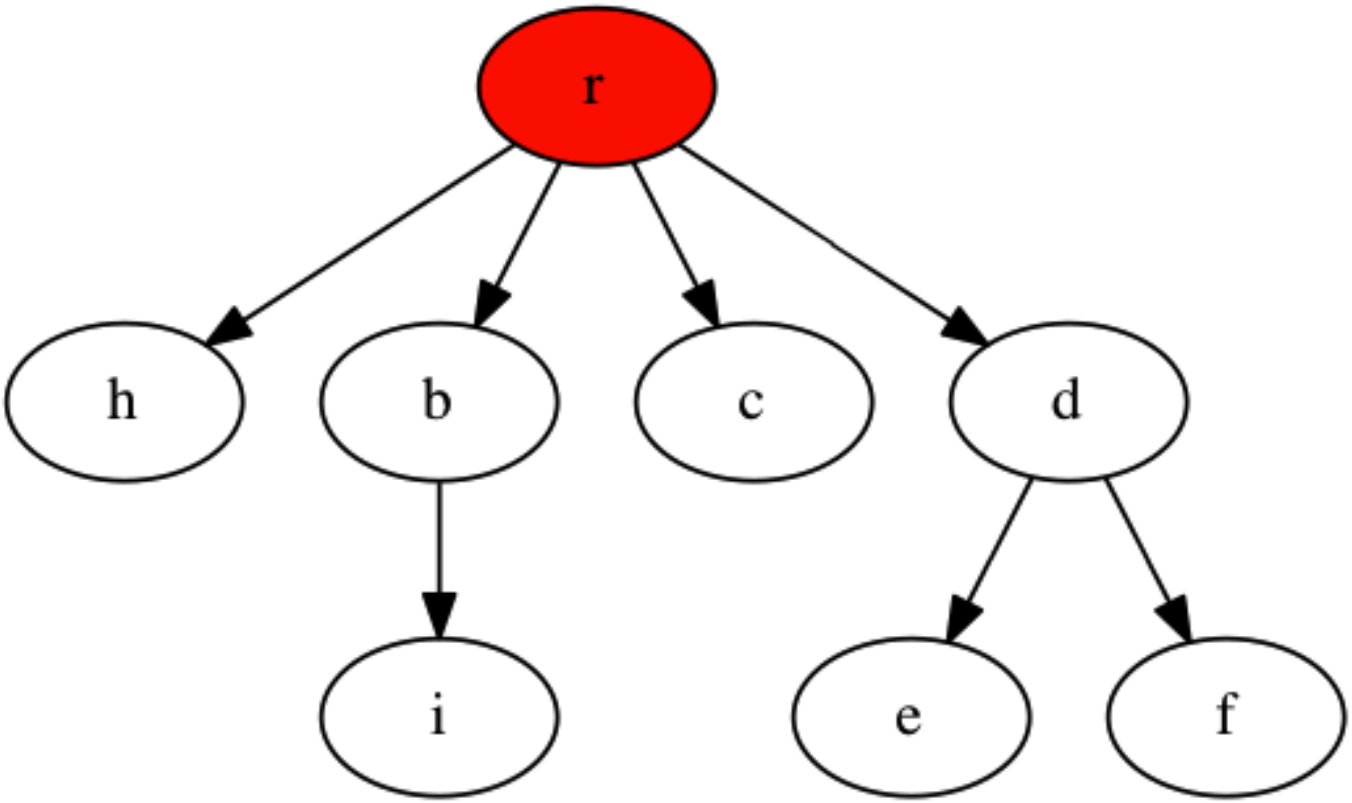
Example: Searching for c (DFS)



Stack (top)

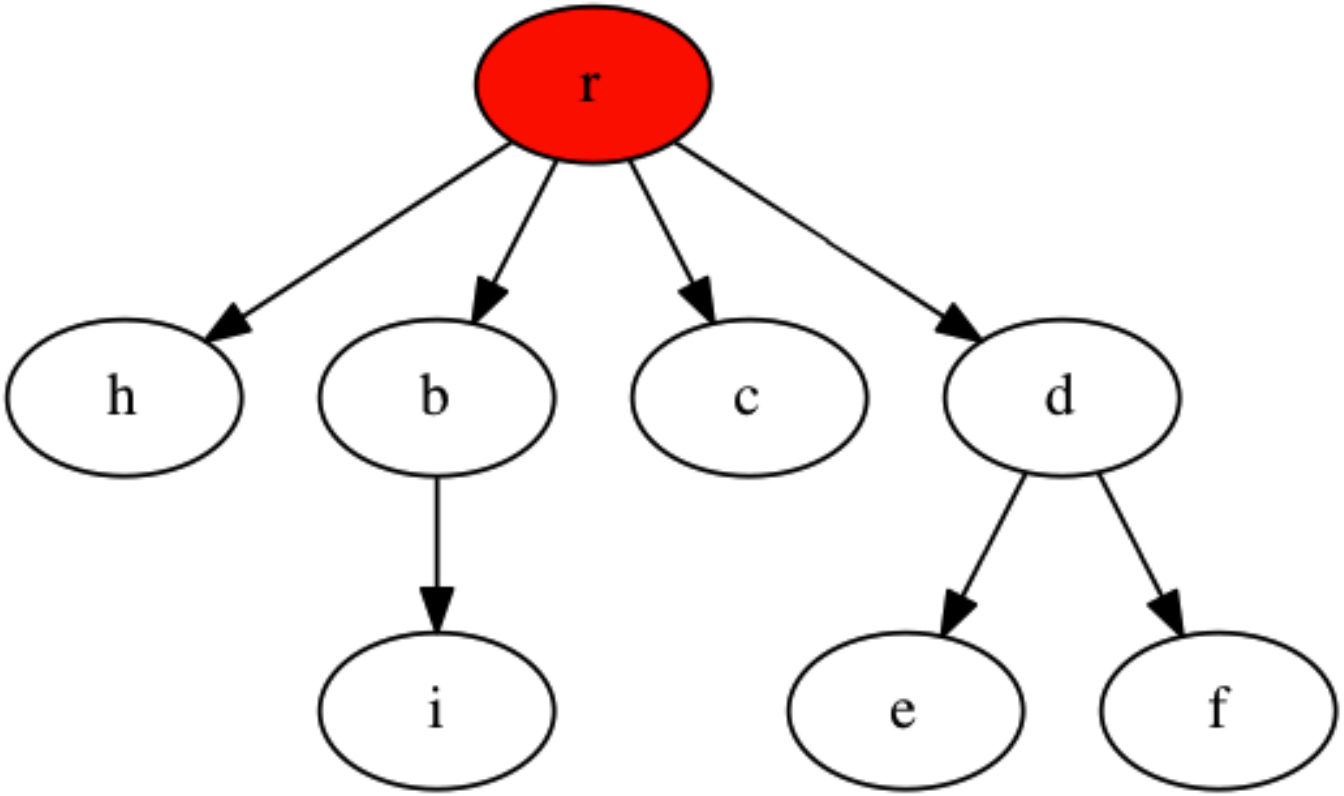


Example: Searching for c (DFS)

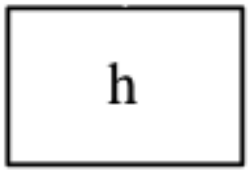


Stack (top)

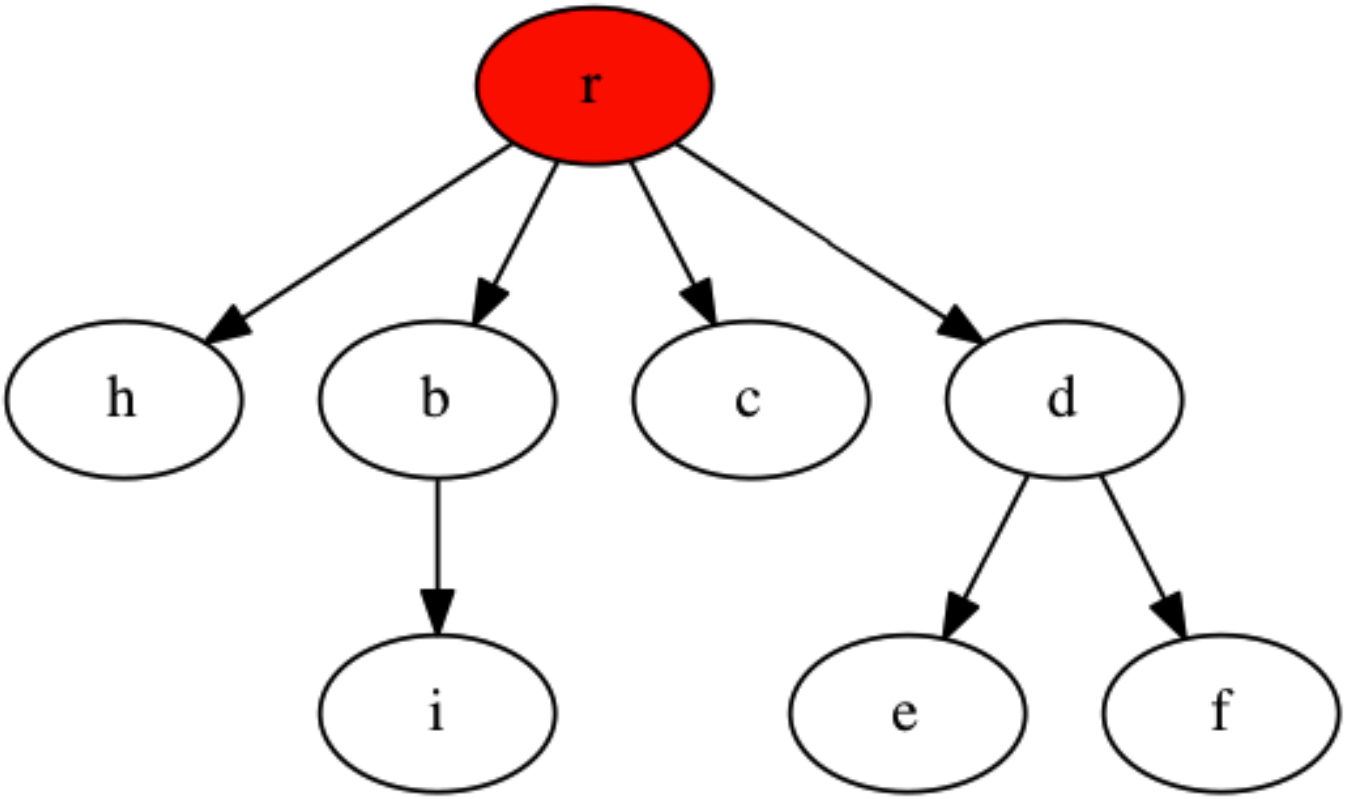
Example: Searching for c (DFS)

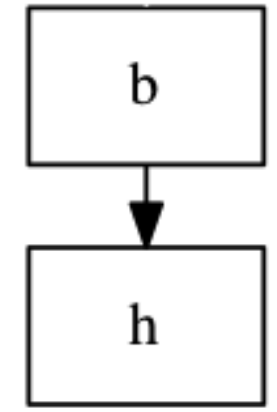


Stack (top)

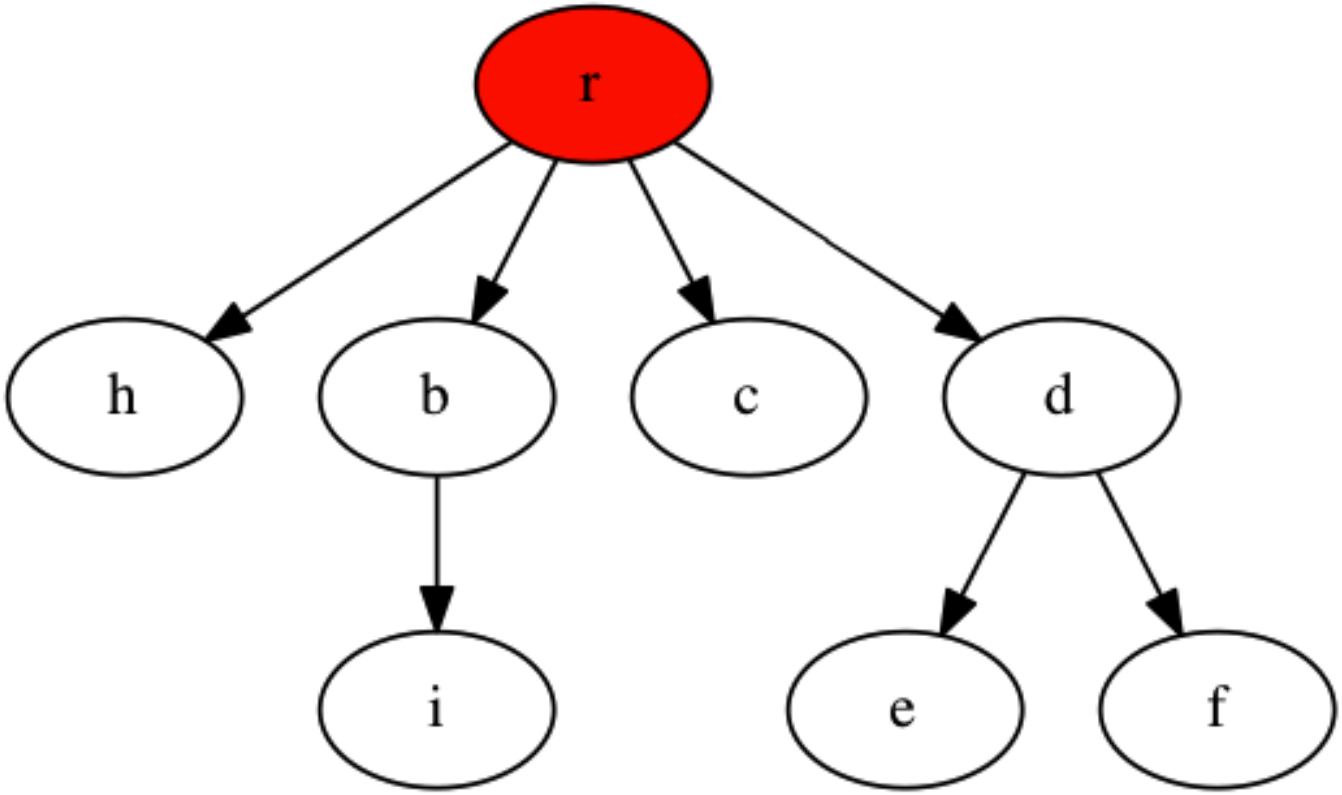


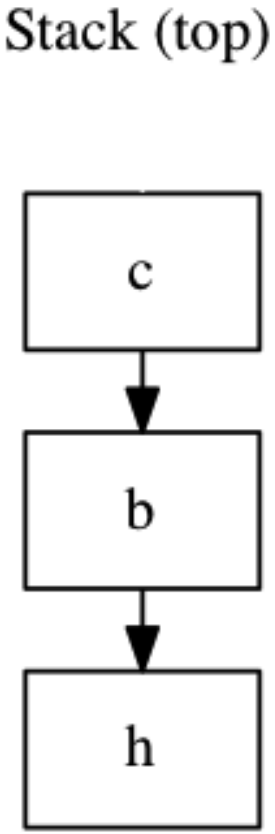
Example: Searching for c (DFS)



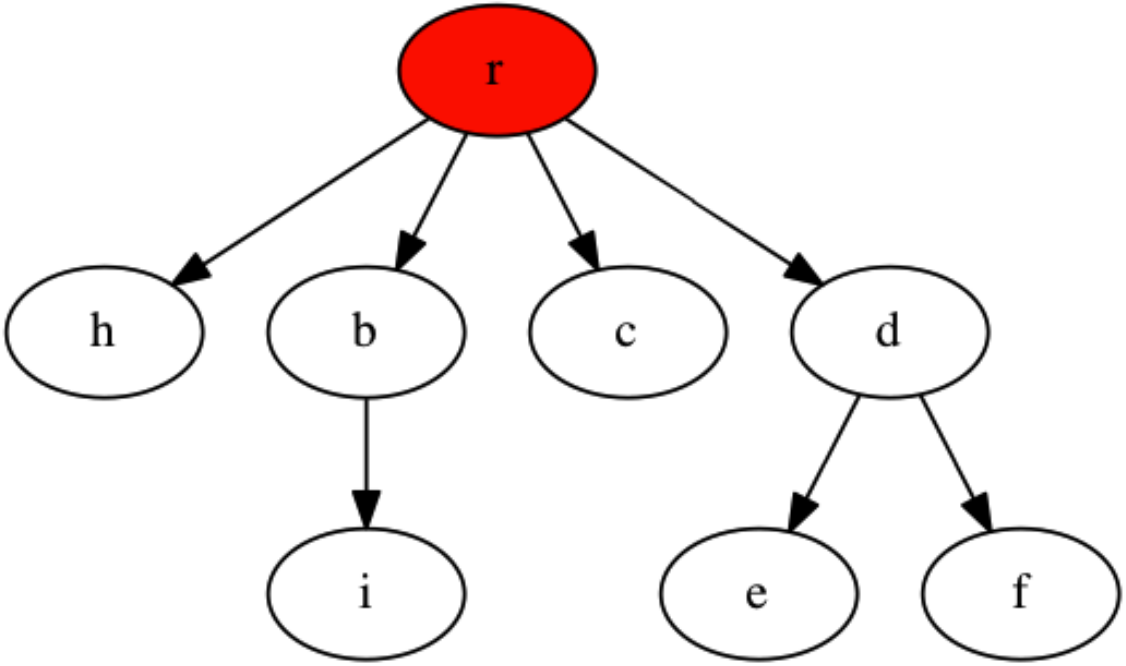


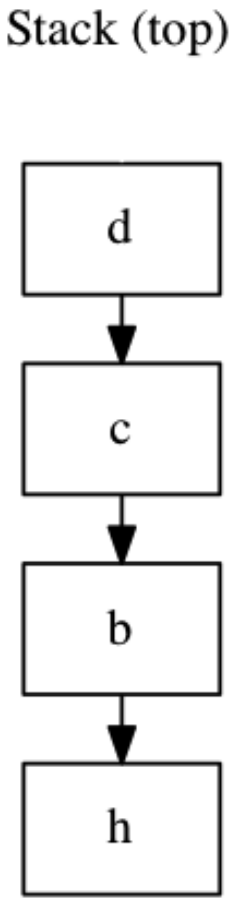
Example: Searching for c (DFS)



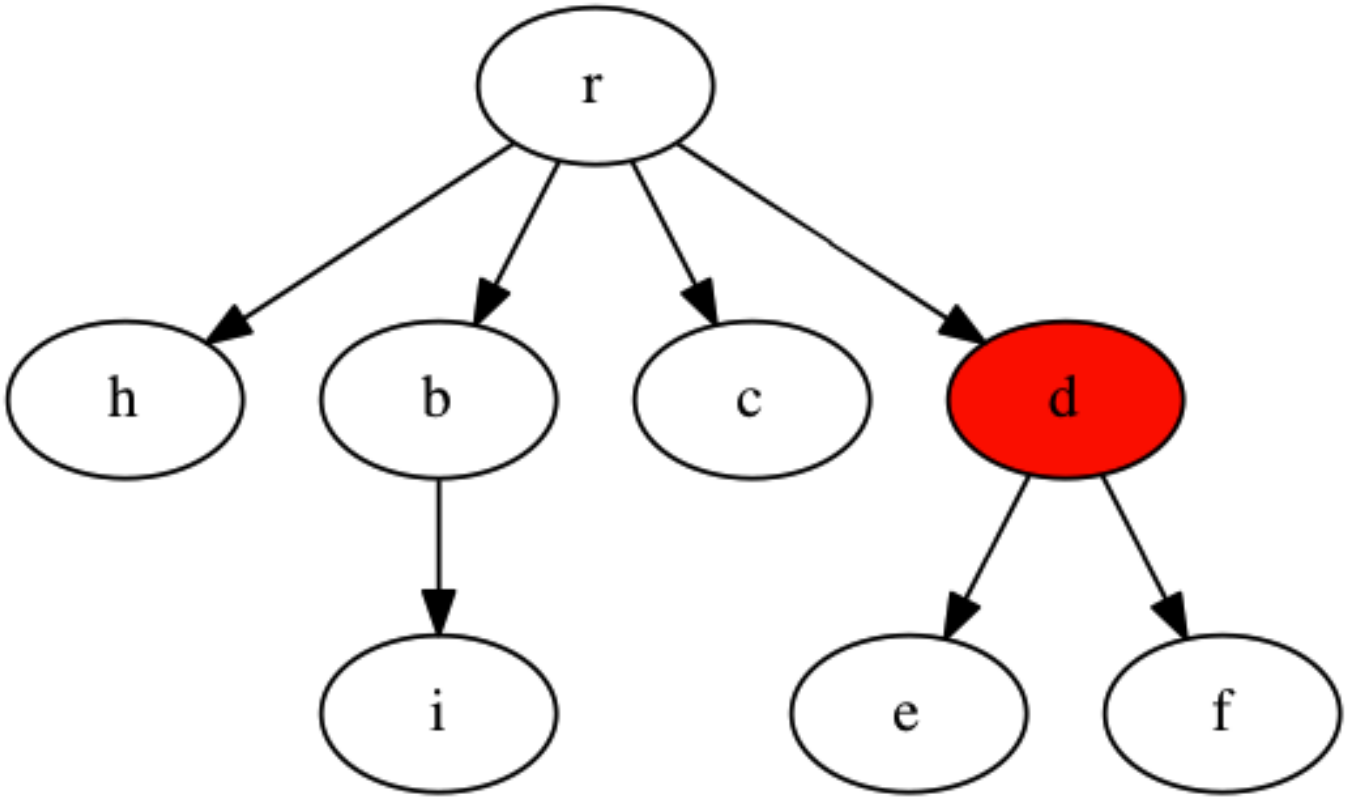


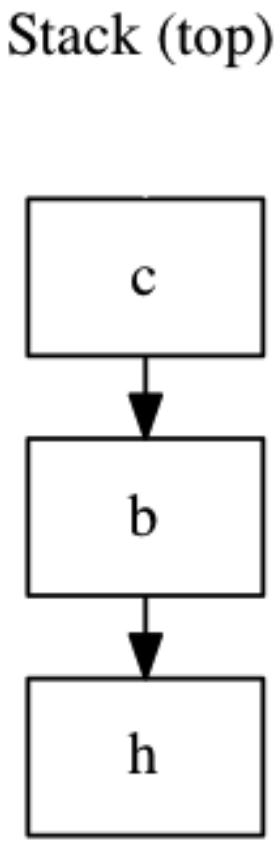
Example: Searching for c (DFS)



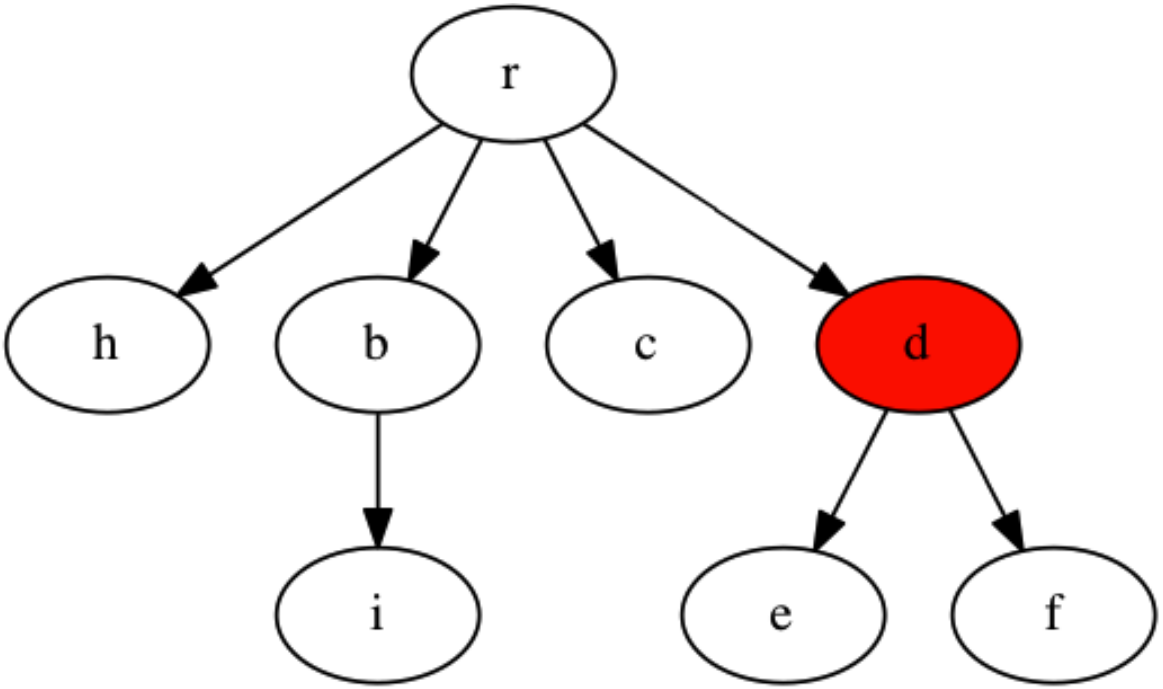


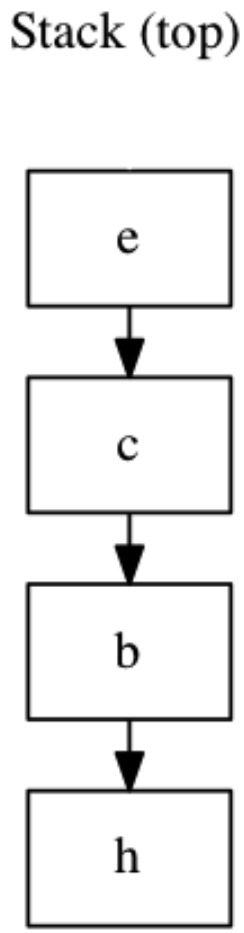
Example: Searching for c (DFS)



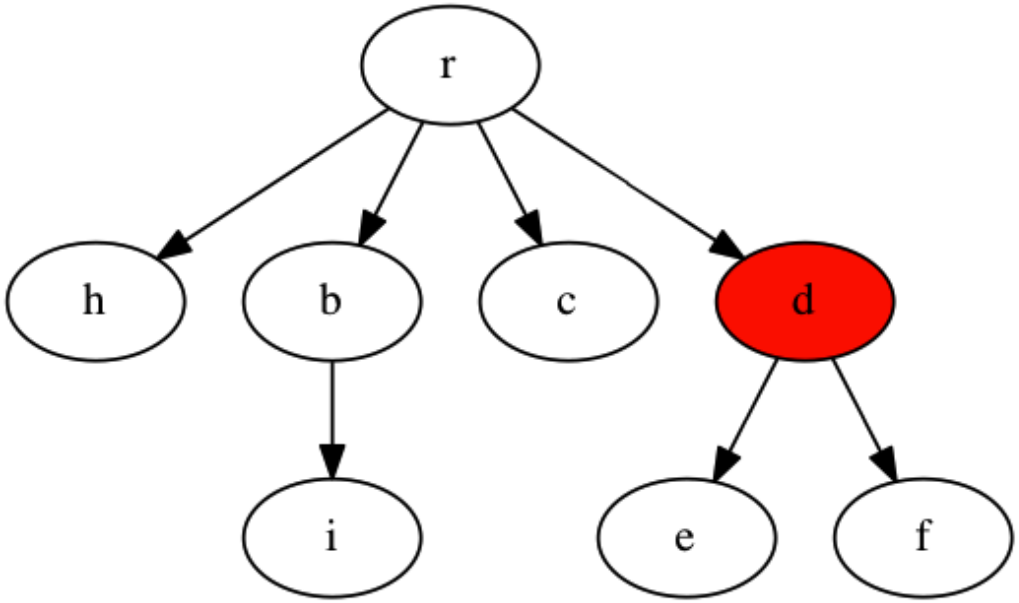


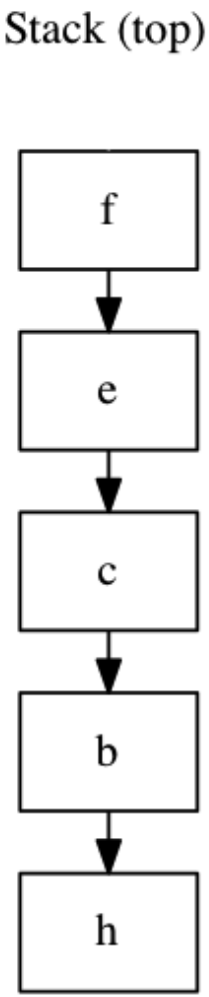
Example: Searching for c (DFS)



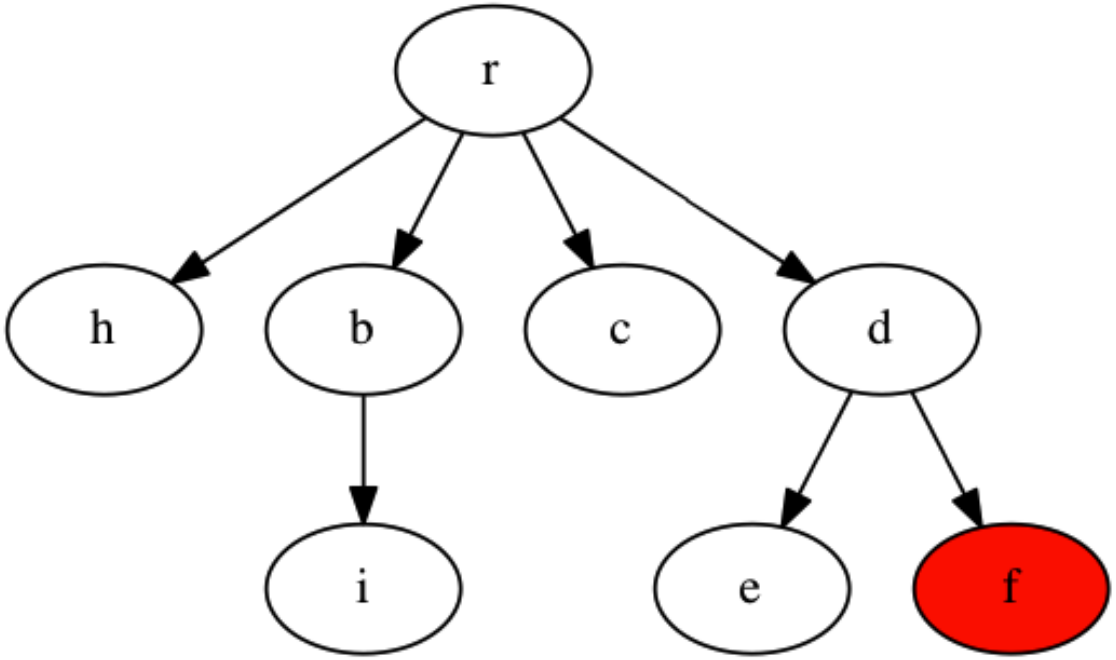


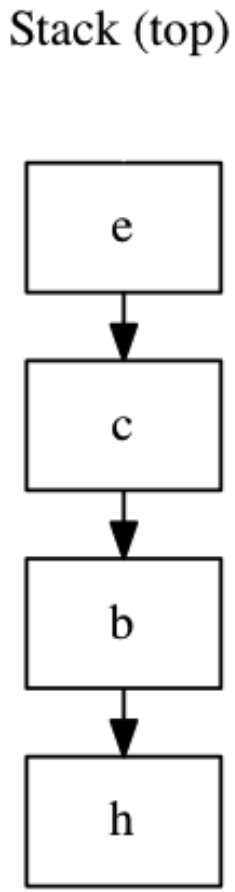
Example: Searching for c (DFS)



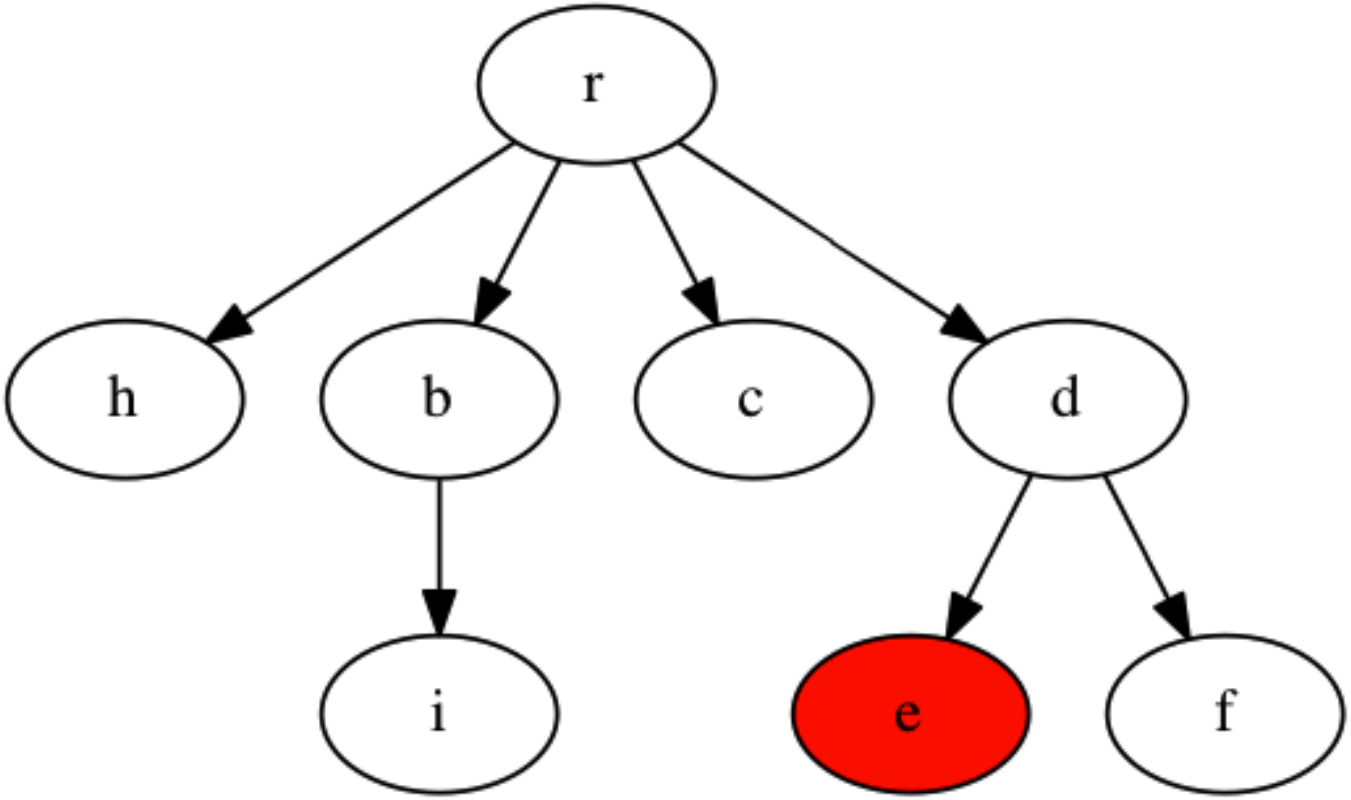


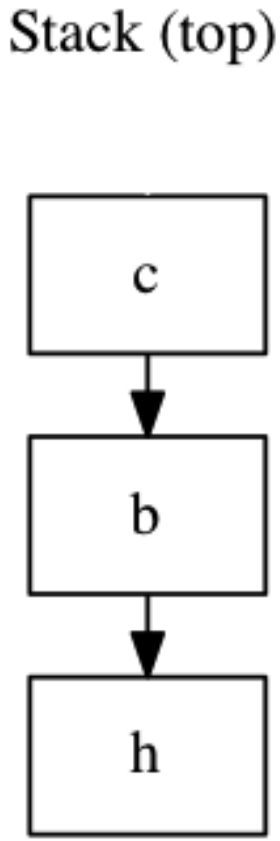
Example: Searching for c (DFS)



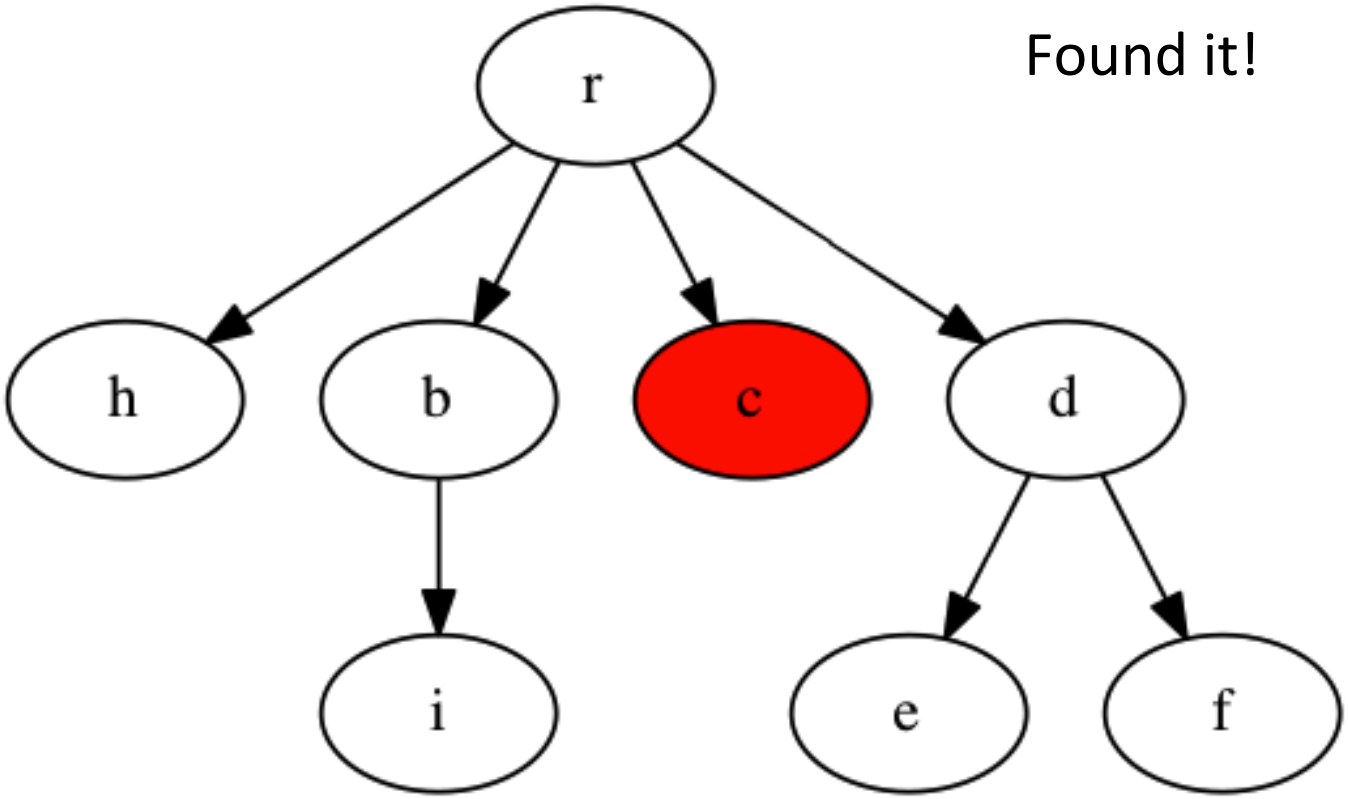


Example: Searching for c (DFS)



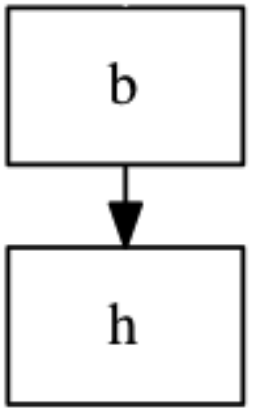


Example: Searching for c (DFS)



Found it!

Stack (top)

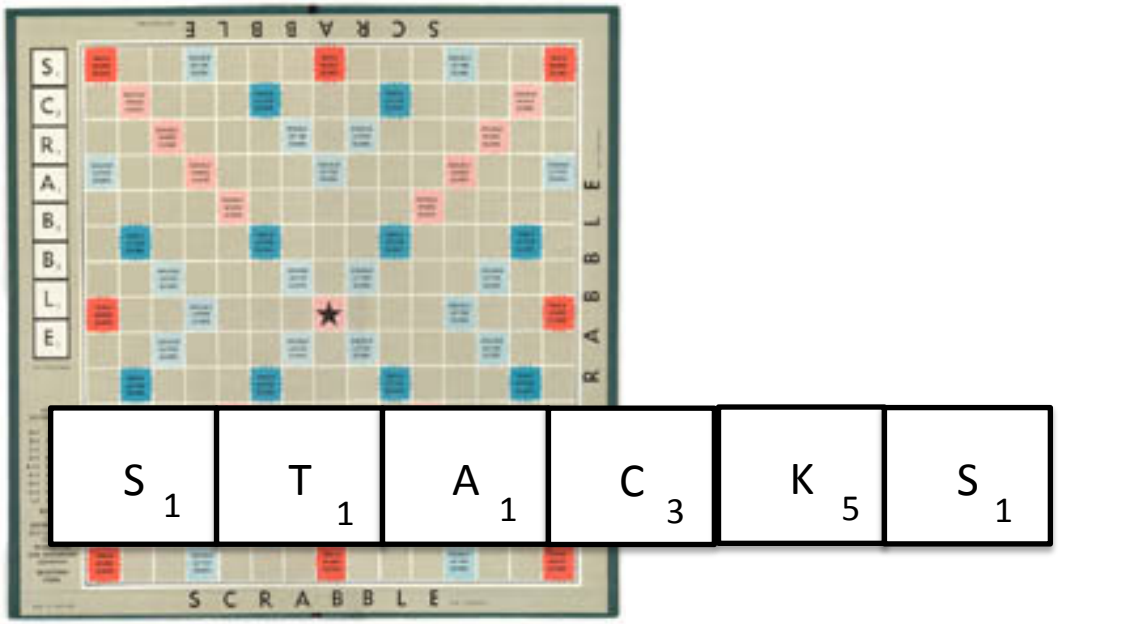


Live Demo (time permitting)

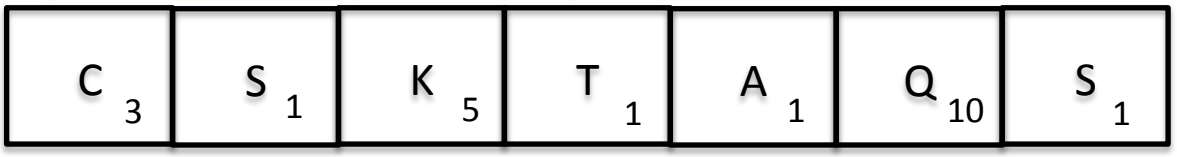
Example Application: Highest Point Scrabble Word Revisited

Searching graphs is a fundamental problem in computer science.
Homework: finding shortest path through a maze
Here is another example…

Scrabble Board:



Rack:



Highest Point Scrabble Word (Assignment 1)

- **Given:** a rack of tiles and a dictionary of legal scrabble words
- **Return:** the highest point scrabble word that can be made from the rack of tiles and its point value

Example

Input: computersciencerocks

Output: crockpots 19

Scrabble Dictionary Class Provided

```
public class ScrabbleDictionary
{
        ...
        public ScrabbleDictionary() { ... }

        public int getPointValue(String word) { ... }
        public boolean canMakeWord(String word,
                                   String rack) { ... }
        public String[] getAllWords();
        ...
}
```

We can consider these functions to be a black box (don't be afraid of abstraction!)

Optimal Scrabble Word Attempt 1

```
public class ScrabbleAttempt1
{
        public static void main(String[] args)
        {
                ScrabbleDictionary dict = new ScrabbleDictionary();
                String[] allWords = dict.getAllWords();
                String rack = "paulruvolo";
                String bestWord = "";
                int maxPointValue = 0;









        }

}
```

```
Output:
vapour - 11
```

Total Words tested: 84,009

Can Graphs Help Cut Down on the Number of Words We Have to Test?

Question for the class: In what ways is the method of exhaustive search inefficient?

Let's take a Reduced Dictionary of Words

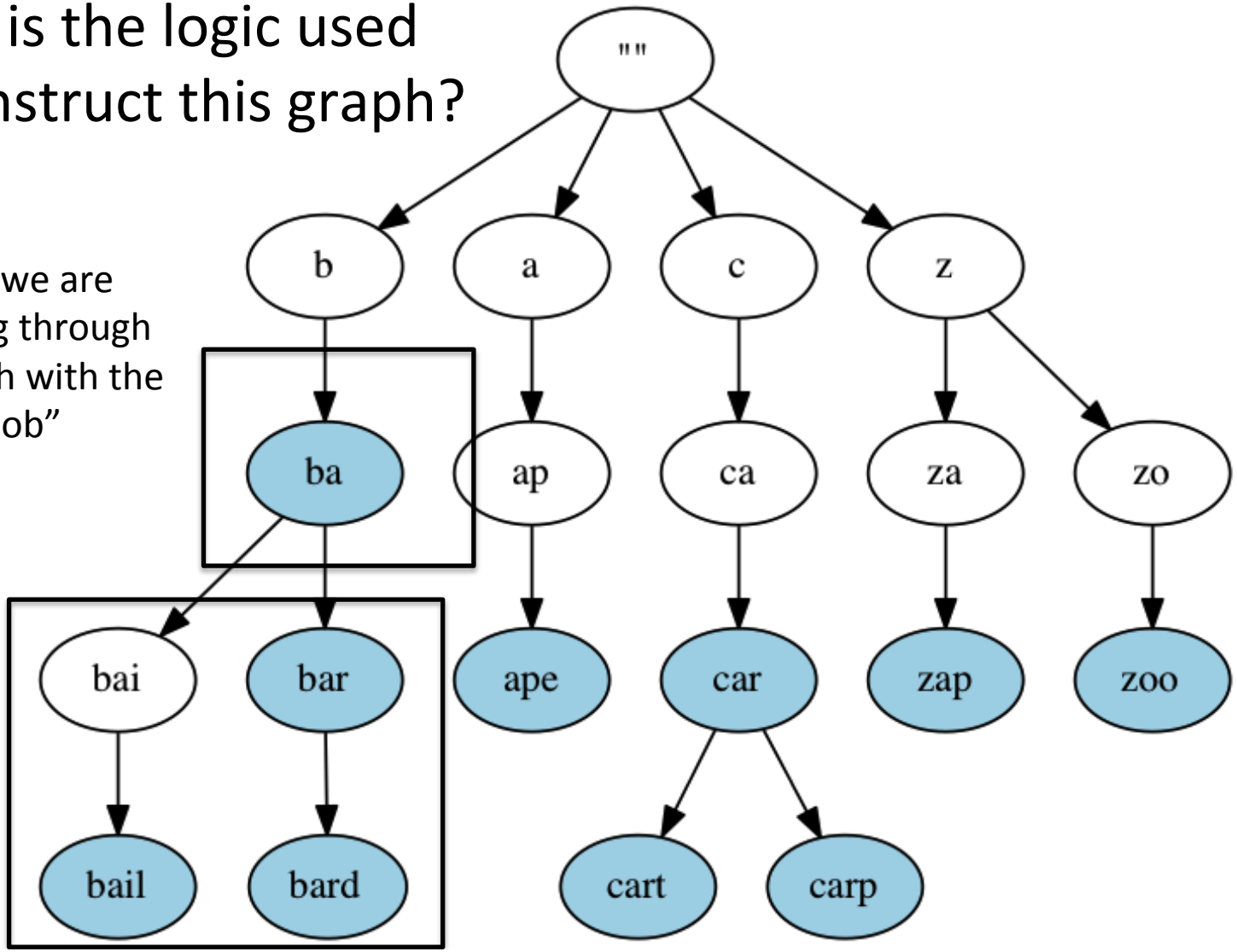
- ba
- bail
- ape
- bar
- car
- bard
- cart
- carp
- zap
- zoo

Let's represent these as a graph

Words as a Graph

What is the logic used to construct this graph?

Suppose we are searching through this graph with the rack “czoob”



Word List:

ba
bail
ape
bar
car
bard
cart
carp
zap
zoo

How Could We Search Such a Graph?

New methods for ScrabbleDictionary

- **getRoot:** returns the word at the root of the graph of valid prefixes
- **getChildren:** returns an array of the names of each childe node of the input node
- **isWord:** return true if the specified String is a valid Scrabble word

Searching a Graph Using DFS

```
ScrabbleDictionary dict = new ScrabbleDictionary();
String rack = "paulruvolo";
String bestWord = "";
StringStack toSearch = new StringStack();
int maxPointValue = 0;

toSearch.push(dict.getRoot());
```

…

Continued

```
while (!toSearch.isEmpty())
{

}
```

Output:
vapour 11
total searched 1275

Would Breadth First Search be Any More Efficient?