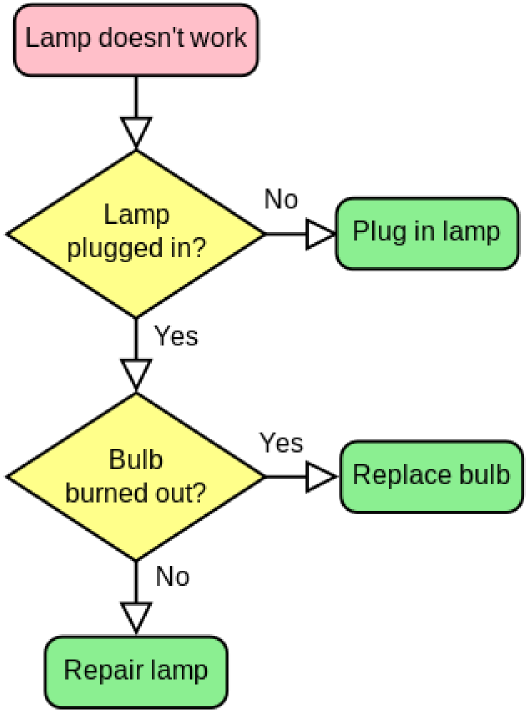
```java
import java.util.Random;

public class Puzzle5 {

    private static Random rnd = new Random();

    public static void main(String[] args) {
        StringBuffer word = null;
        switch(rnd.nextInt(2)) {
            case 1: word = new StringBuffer('R');
            case 2: word = new StringBuffer('T');
            default: word = new StringBuffer('J');
        }
        word.append("est");
        System.out.println(String.format("It's time for a %s.", word));
    }

}
```

```java
public class Puzzle6 {

    public static void main(String[] args) {
        System.out.println(getResult());
    }

    private static boolean getResult() {
        try {
            return true;
        } finally {
            return false;
        }
    }

}
```

Ice Breaker Exercise: EIL5 “Programming”

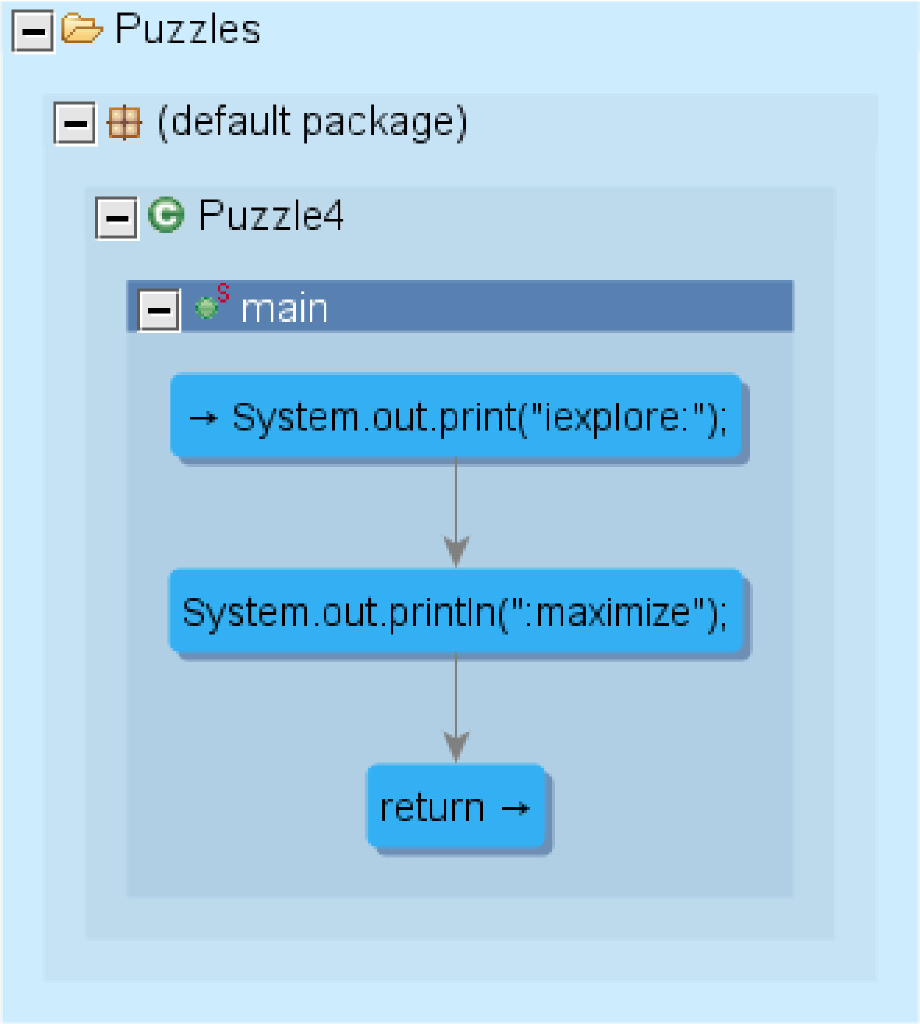
- Explain It Like I’m Five (EIL5): How do computer programs work?
- Can your explanation intuitively address:
  - Complexity of software
  - Programming bugs
  - Security issues

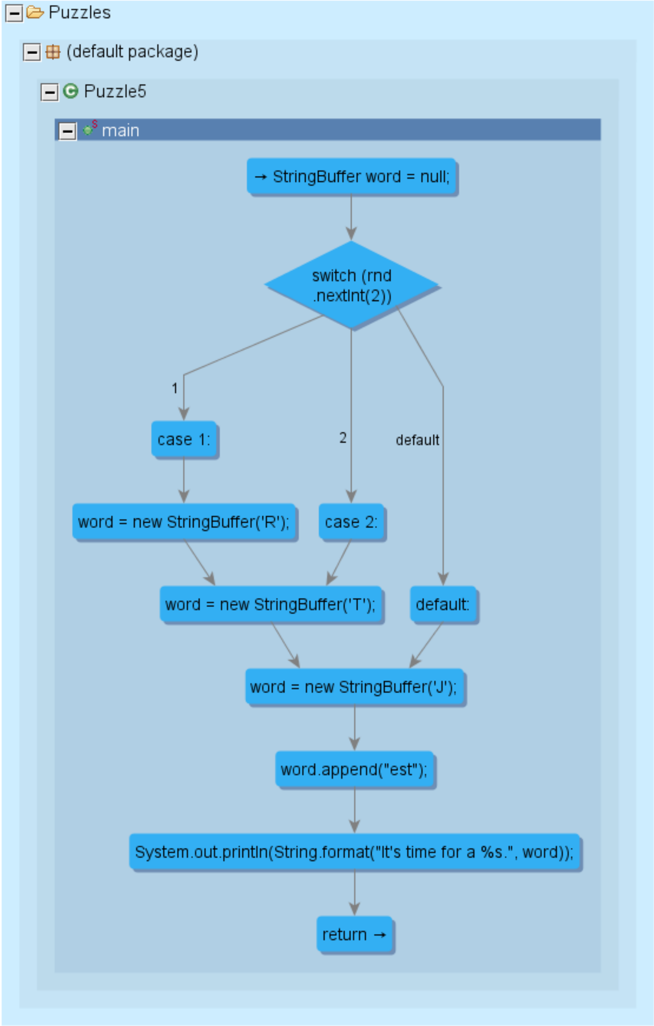


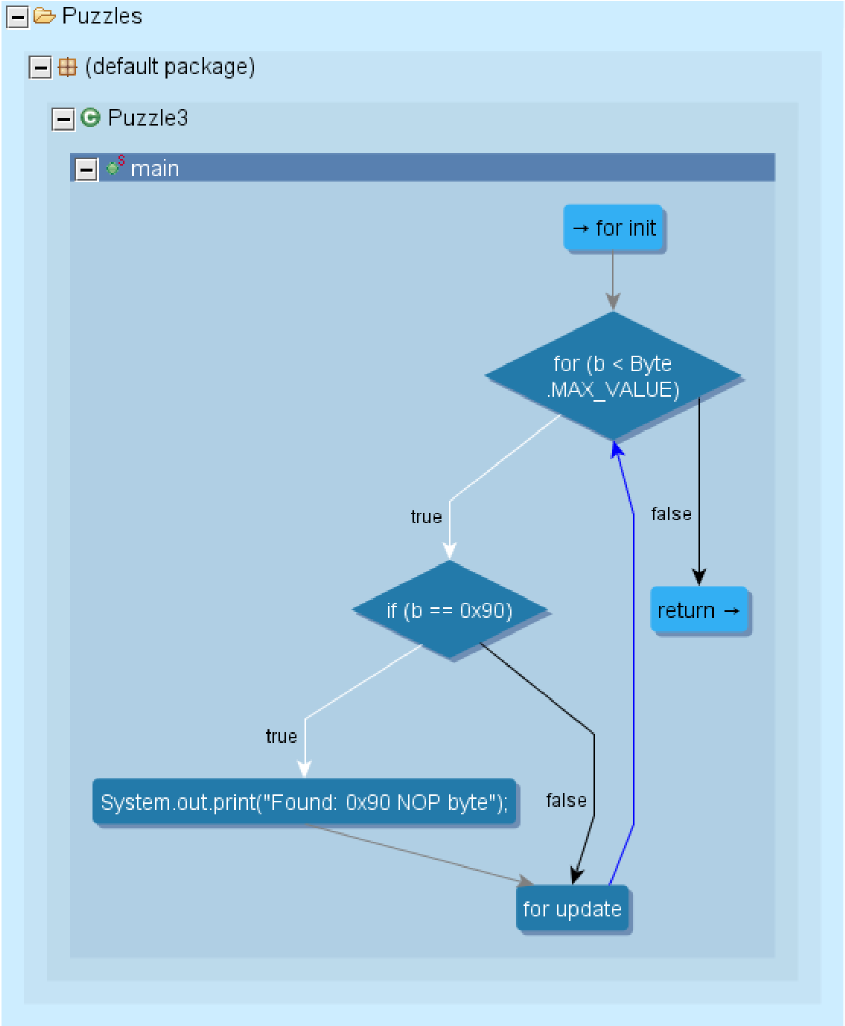
Control Flow Graph (CFG)

- A control flow graph (CFG) is a graph representation that captures the paths that might be traversed through a program during its execution, (i.e. the orderings that the program’s statements may be executed in at runtime).
- Reading: *Frances E. Allen. 1970. Control flow analysis. In Proceedings of a symposium on Compiler optimization. ACM, New York, NY, USA, 1-19.*

Control Flow Graph (CFG)



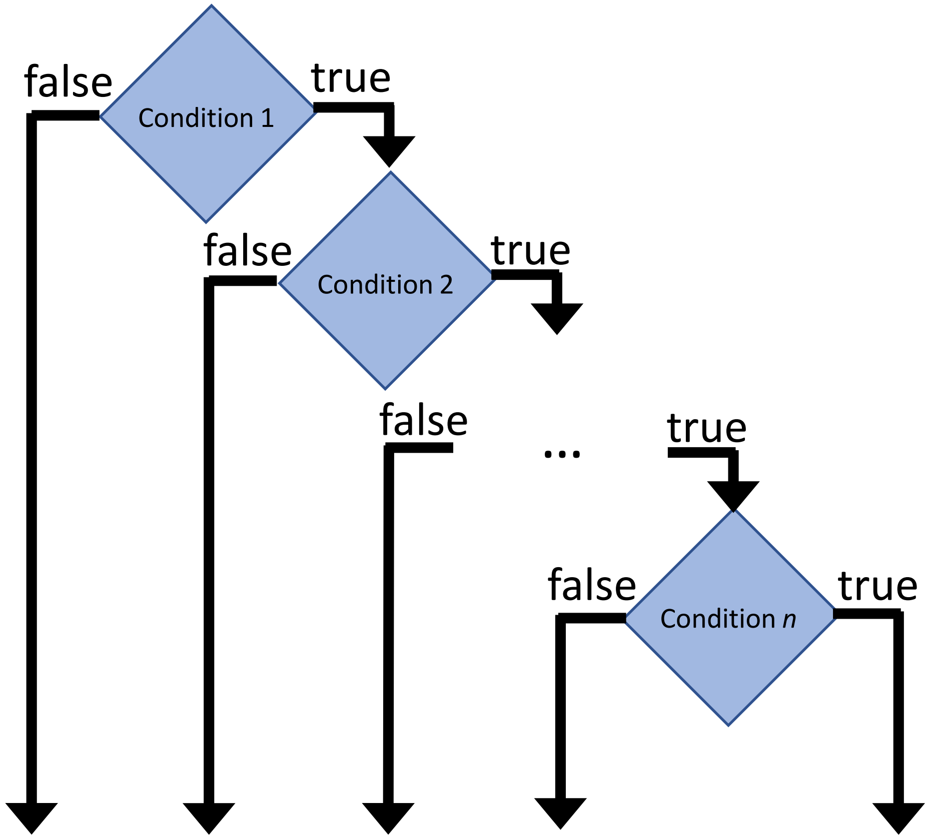




Counting Program Paths

- How many paths are there for *n* nested branches?

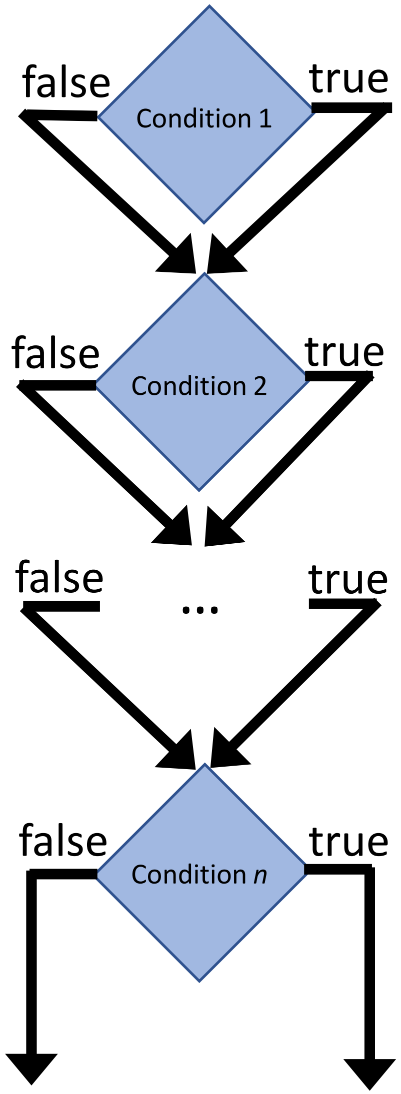
```
if(condition_1){
    if(condition_2){
        if(condition_3){
            …
            if(condition_n){
                // conditions 1 through n
                // must all be true to reach here
            }
        }
    }
}
```



Counting Program Paths

- How many paths are there for *n* non-nested branches?

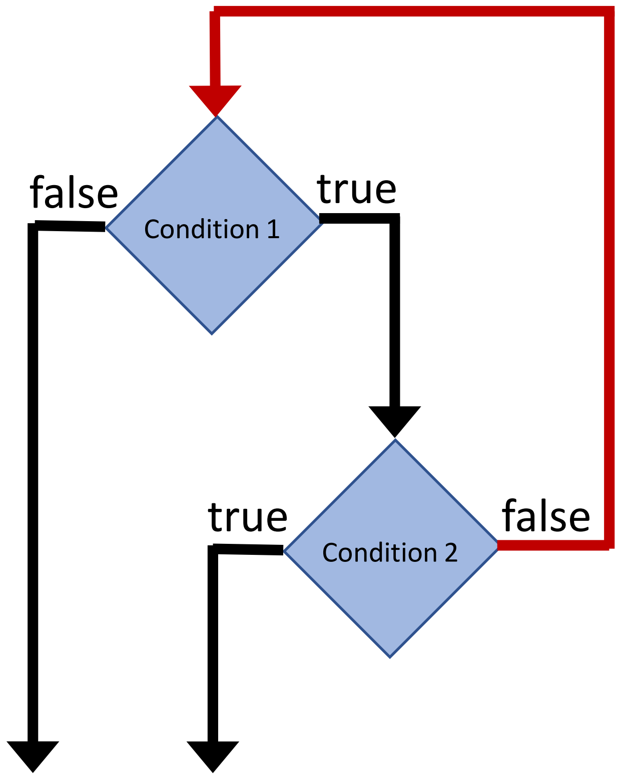
```
if(condition_1){
   // code block 1
}
if(condition_2){
   // code block 2
}
if(condition_3){
   // code block 3
}
…
if(condition_n){
   // code block n
}
```



Considering Loops

- Programs may have loops
  - How many paths does this program have?
  - Can we say if this program halts?

```
while(condition_1){
    if(condition_2){
        break;
    }
}
```



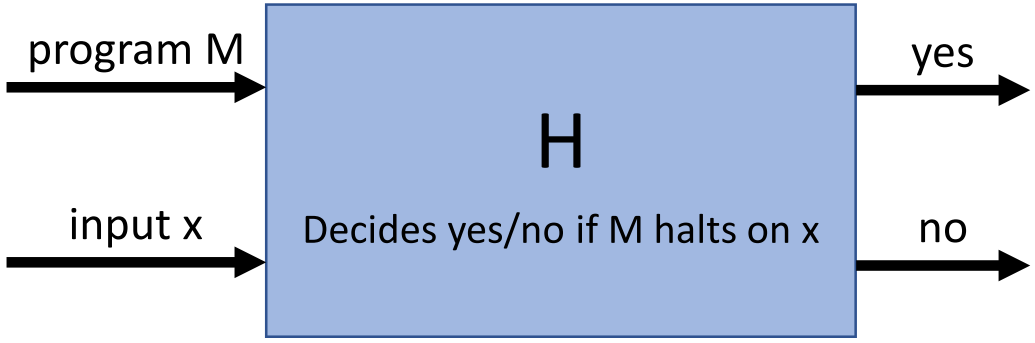
The Halting Problem

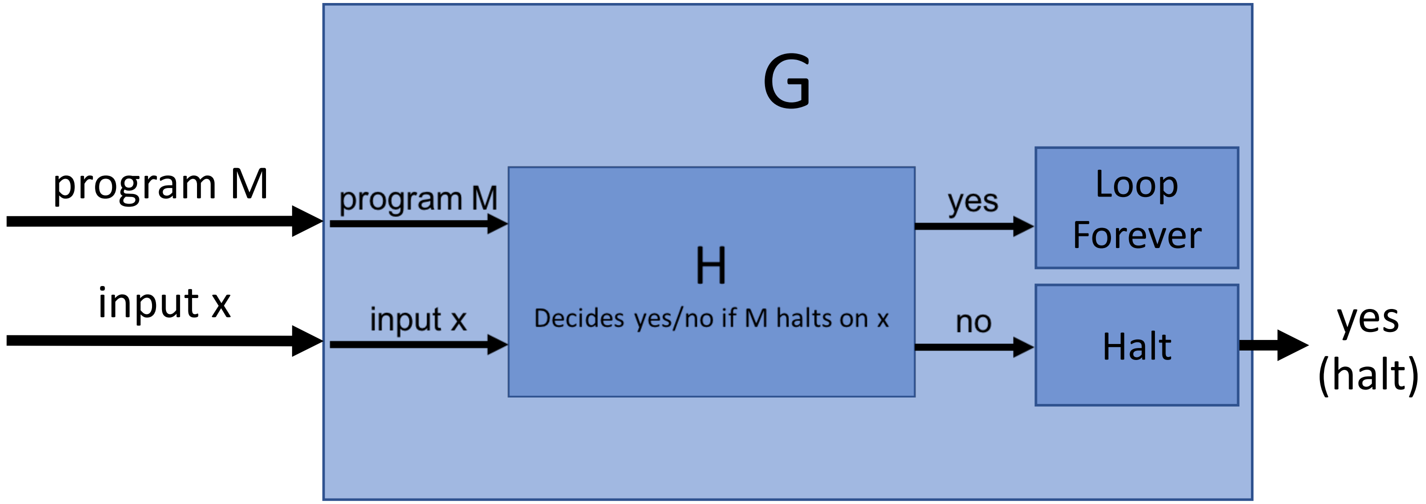
Suppose, we could construct:
H(M, x) := if *M* halts on *x* then return true else return false

Then we could construct:
G(M, x) := if G(M, *x)* is false then return true else loop forever

But if we then pass *G* to itself, that is G(G,G), we get a contradiction between what *G* does and what *H* says that *G* does. If *H* says that *G* halts, then *G* does not halt. If *H* says that *G* does not halt, then it does halt.

H cannot exist.





Group Formation

- Count off 1 to 4

Eclipse Plugin Development + Atlas

- Setting Up Eclipse with Atlas
  - Install: Atlas (all plugins), Plugin Development Tools
- Debug As → Eclipse Application
- Xinu Project (included with Atlas)
- Atlas Shell (add plugin project to dependencies)

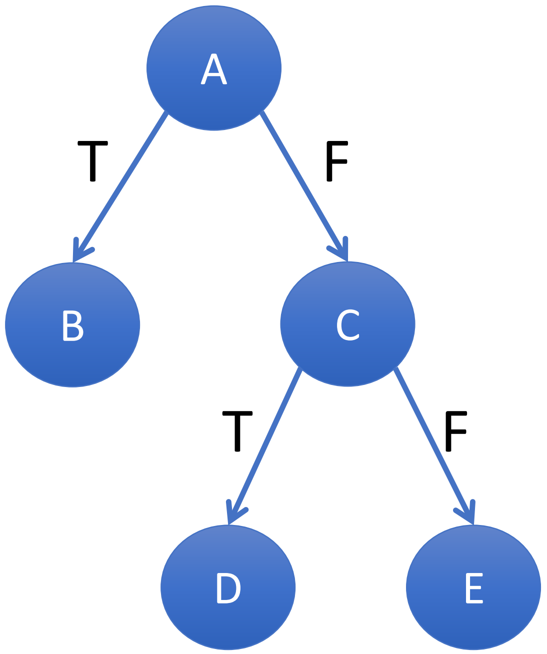
DFS Enumeration Strategy

Push A (root) on stack.

Stack: [A]

History:

Paths:



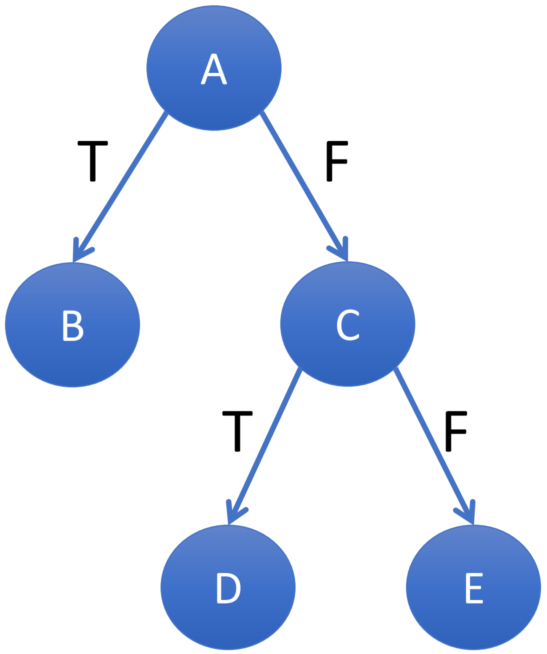
DFS Enumeration Strategy

Pop A onto history. A is not a leaf so push C,B.

Stack: [C, B]

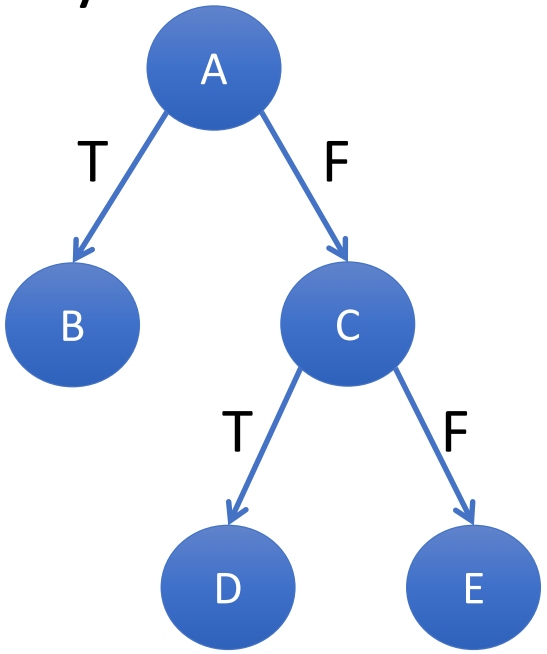
History: [A]

Paths:



DFS Enumeration Strategy

Pop B onto history. B is a leaf so save path and trim history.



Stack: [C]

History: [A, ~~B~~]

Paths: [A, B]

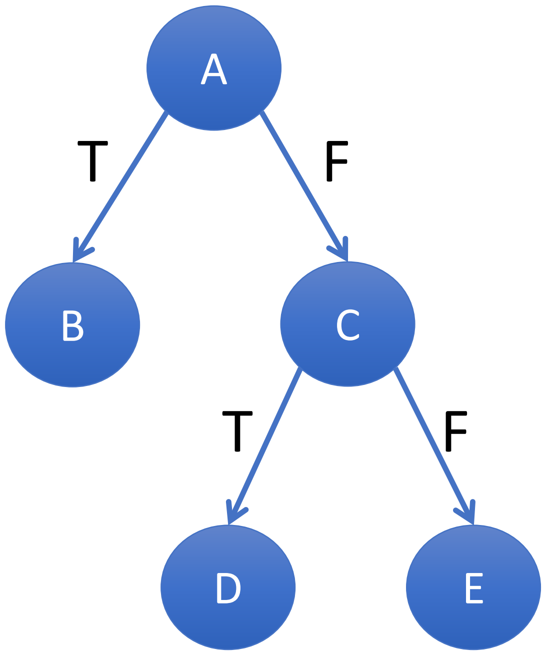
DFS Enumeration Strategy

Pop C onto history. C is not a leaf so push E,D.

Stack: [E, D]

History: [A, C]

Paths: [A, B]



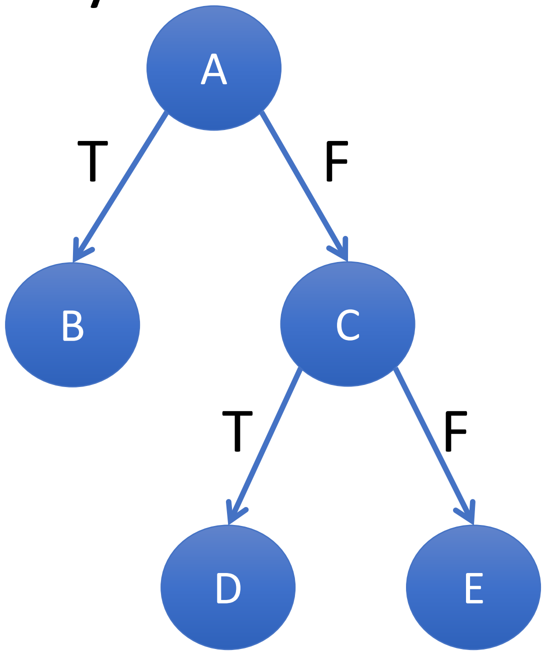
DFS Enumeration Strategy

Pop D onto history. D is a leaf so save path and trim history.

Stack: [E]

History: [A, C, ~~D~~]

Paths: [A, B], [A, C, D]



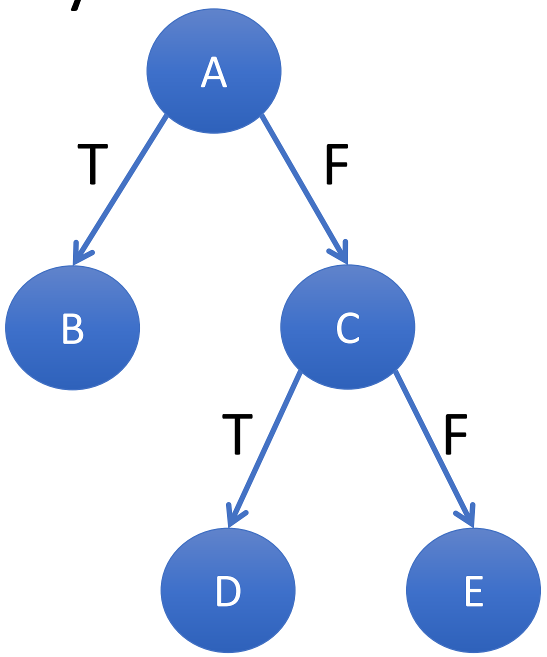
DFS Enumeration Strategy

Pop E onto history. E is a leaf so save path and trim history.

Stack: []

History: [A, C, ~~E~~]

Paths: [A, B], [A, C, D], [A, C, E]



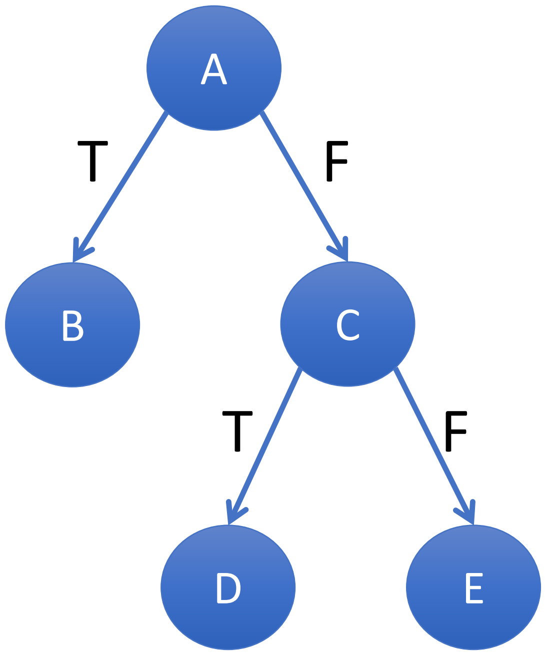
DFS Enumeration Strategy

Stack is empty. Paths are enumerated.

Stack: []

History: [A, C]

Paths: [A, B], [A, C, D], [A, C, E]



Problem 2 (remaining class time)

- Discuss path counting strategy
  - Why do we need a DAG? What’s the implications of using a DAG?
  - What is the complexity of the algorithm with respect to the DAG?
- Explore support code
  - CFGPrinter.java Example
- Group Coding for problem 2