

# Uninformed Search

Stephen G. Ware

CSCI 4525 / 5525



THE UNIVERSITY *of*  
NEW ORLEANS

# Grid World

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

- **Initial State:** Square L
- **Actions:** Up, Right, Down, Left
- **Transition:** Move to new square
- **Goal:** Square G (for goal)
- **Cost:** 1 unit per action
- **Performance:** Minimize cost

# Search

1. Let  $V$  be the set of visited nodes, empty.
2. Let  $F$  be the frontier, initially containing only the initial state.
3. Loop:
4.   If  $F$  is empty, return failure.
5.   Choose a node  $n$  to remove from  $F$ .
6.   If  $n$  is a solution, return  $n$ .
7.   Add  $n$  to  $V$ .
8.   For every successor  $s$  of  $n$  not in  $V$  or  $F$ :
9.     Add  $s$  to  $F$ .

# Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

 = Visited

 = Frontier

V is visited, initially empty.

F is the frontier, with start.

Loop:

If F is empty, return failure.

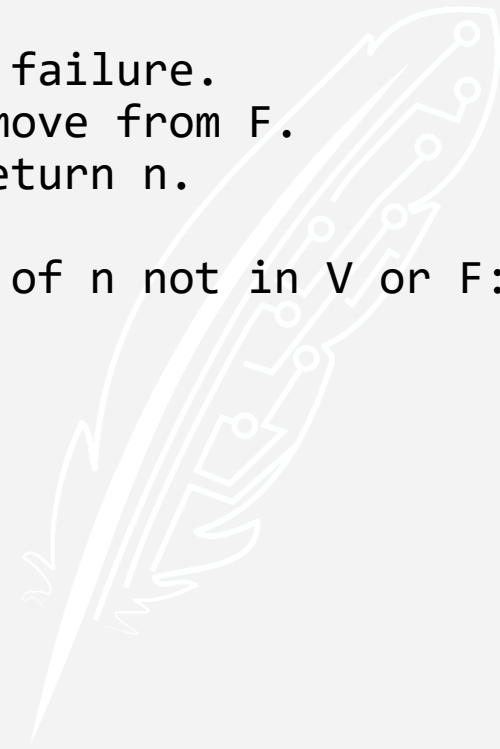
Choose a node n to remove from F.

If n is a solution, return n.

Add n to V.

For every successor s of n not in V or F:

Add s to F.



# Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

 = Visited

 = Frontier

V is visited, initially empty.

F is the frontier, with start.

Loop:

If F is empty, return failure.

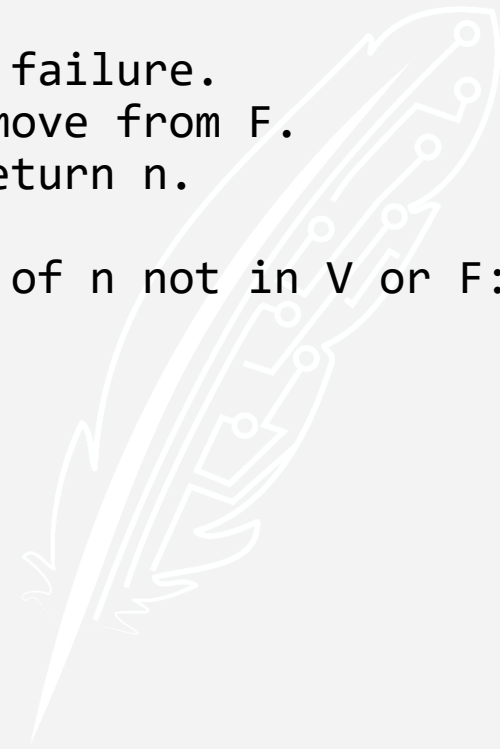
Choose a node n to remove from F.

If n is a solution, return n.

Add n to V.

For every successor s of n not in V or F:

Add s to F.



# Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

 = Visited

 = Frontier

**V is visited, initially empty.**

**F is the frontier, with start.**

**Loop:**

If F is empty, return failure.

Choose a node n to remove from F.

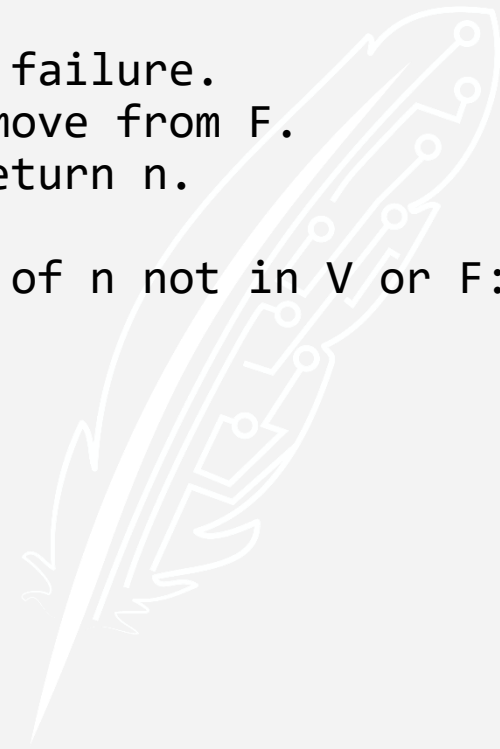
If n is a solution, return n.

Add n to V.

For every successor s of n not in V or F:

Add s to F.

**V = {}**



# Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

 = Visited

 = Frontier

V is visited, initially empty.  
F is the frontier, with start.

Loop:

If F is empty, return failure.

Choose a node n to remove from F.

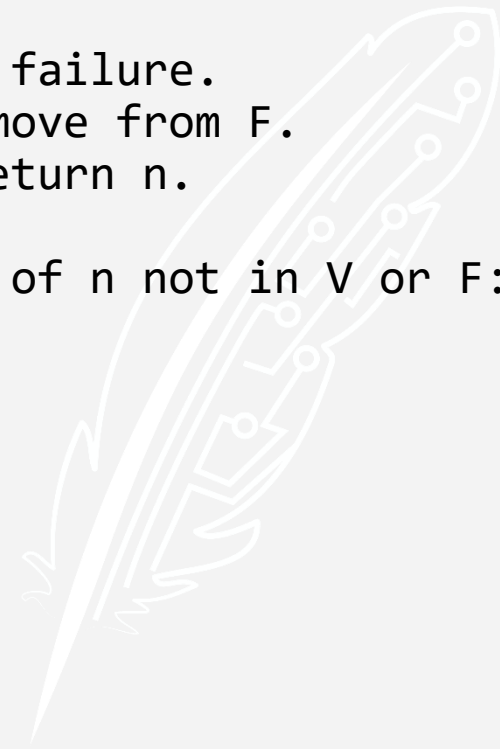
If n is a solution, return n.

Add n to V.

For every successor s of n not in V or F:

Add s to F.

V = {}



# Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

 = Visited

 = Frontier

V is visited, initially empty.  
F is the frontier, with start.

Loop:

If F is empty, return failure.

Choose a node n to remove from F.

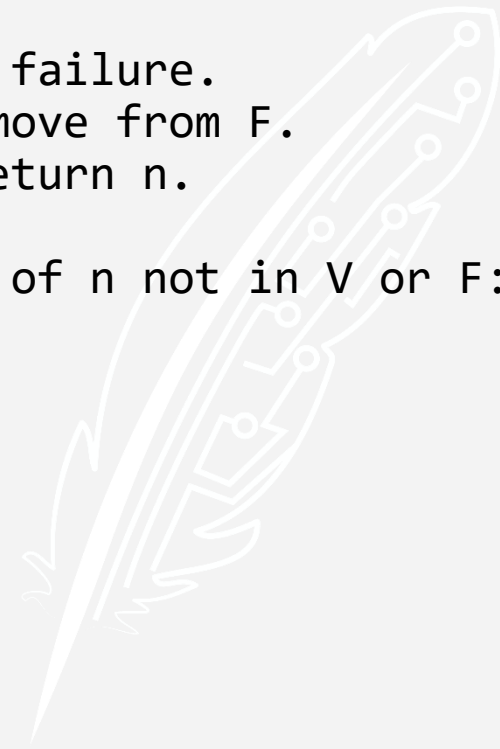
If n is a solution, return n.

Add n to V.

For every successor s of n not in V or F:

Add s to F.



V = {}  
F = {L}





# Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

 = Visited  
 = Frontier

V is visited, initially empty.

F is the frontier, with start.

Loop:

**If F is empty, return failure.**

Choose a node n to remove from F.

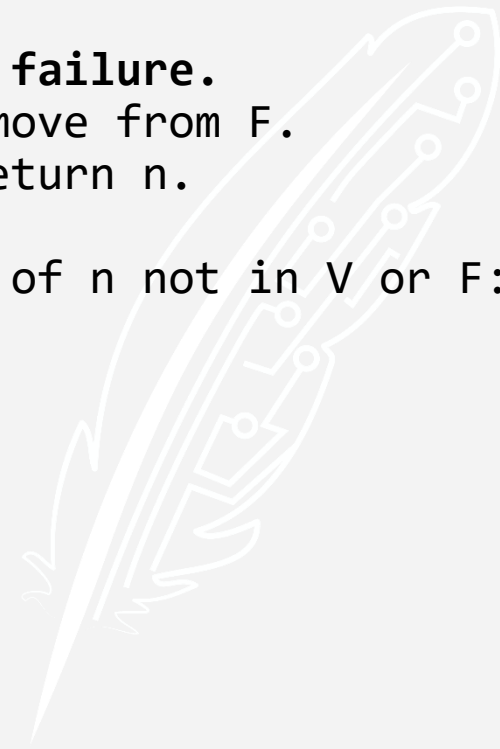
If n is a solution, return n.

Add n to V.

For every successor s of n not in V or F:

Add s to F.

V = {}  
F = {L}



# Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

 = Visited

 = Frontier

V is visited, initially empty.

F is the frontier, with start.

Loop:

If F is empty, return failure.

**Choose a node n to remove from F.**

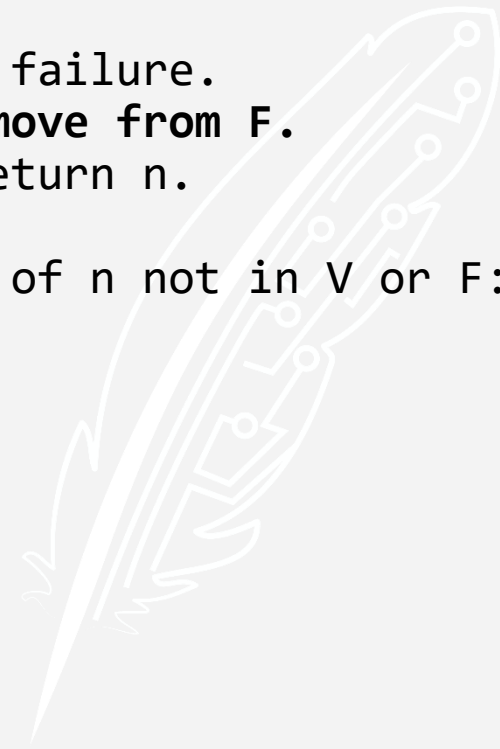
If n is a solution, return n.

Add n to V.

For every successor s of n not in V or F:

Add s to F.

V = {}  
F = {L}



# Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

 = Visited

 = Frontier

V is visited, initially empty.

F is the frontier, with start.

Loop:

If F is empty, return failure.

**Choose a node n to remove from F.**

If n is a solution, return n.

Add n to V.

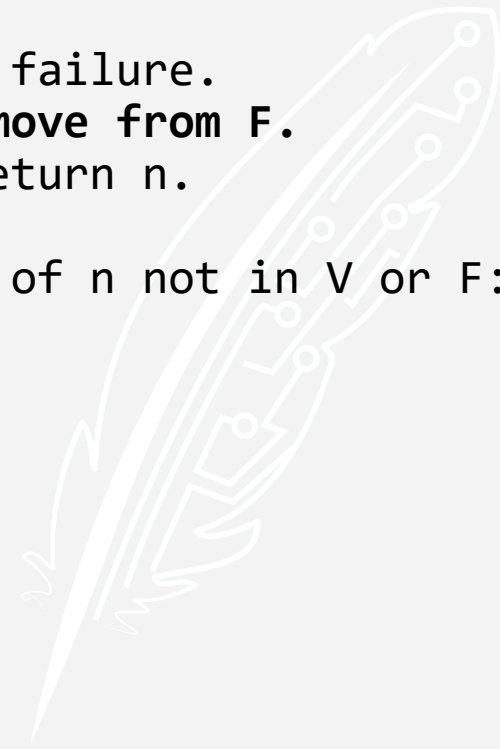
For every successor s of n not in V or F:

Add s to F.

V = {}



F = {}

n = L



# Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

 = Visited  
 = Frontier

V is visited, initially empty.

F is the frontier, with start.

Loop:

If F is empty, return failure.

Choose a node n to remove from F.

**If n is a solution, return n.**

Add n to V.

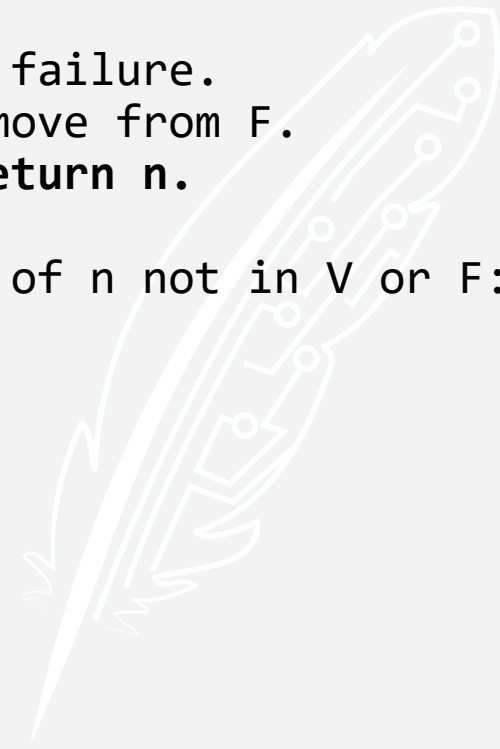
For every successor s of n not in V or F:

Add s to F.

V = {}

F = {}

n = L



# Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

 = Visited

 = Frontier

V is visited, initially empty.

F is the frontier, with start.

Loop:

If F is empty, return failure.

Choose a node n to remove from F.

If n is a solution, return n.

Add n to V.

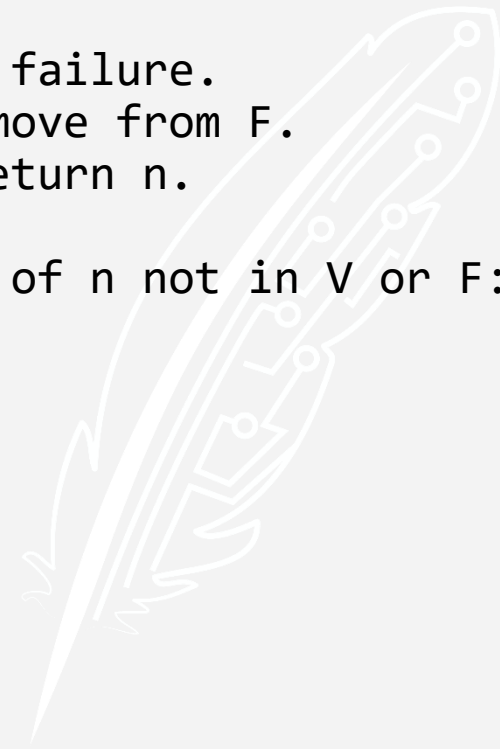
For every successor s of n not in V or F:

Add s to F.

V = {}

F = {}

n = L



# Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

 = Visited

 = Frontier

V is visited, initially empty.

F is the frontier, with start.

Loop:

If F is empty, return failure.

Choose a node n to remove from F.

If n is a solution, return n.

Add n to V.

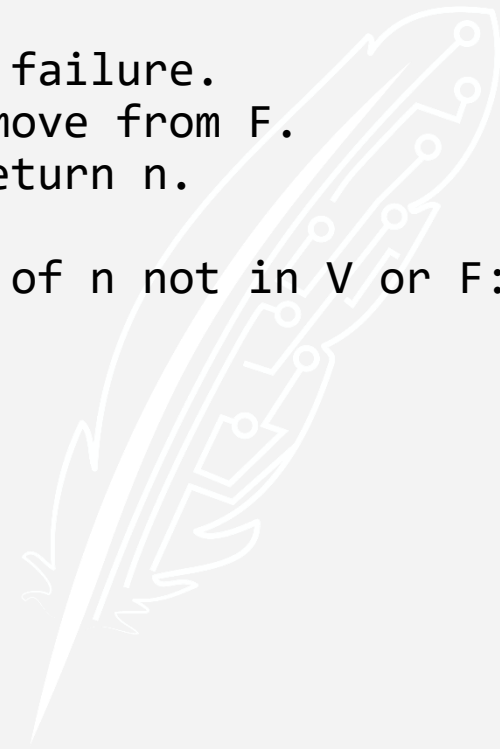
For every successor s of n not in V or F:

Add s to F.

V = {L}

F = {}

n = L



# Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

 = Visited

 = Frontier

V is visited, initially empty.

F is the frontier, with start.

Loop:

If F is empty, return failure.

Choose a node n to remove from F.

If n is a solution, return n.

Add n to V.

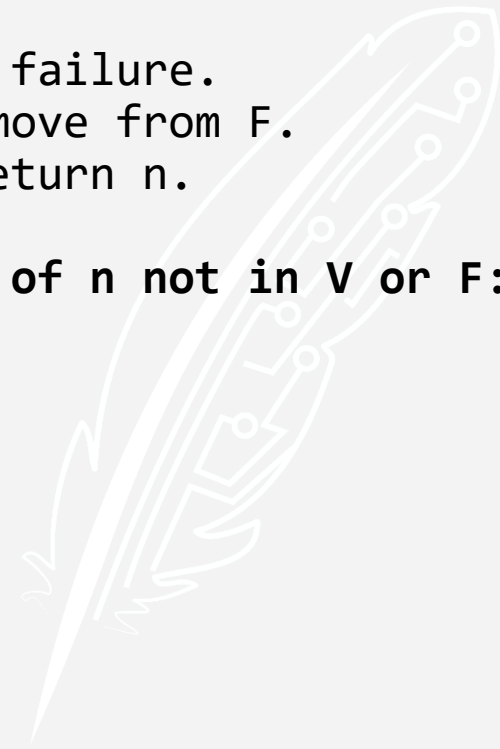
For every successor s of n not in V or F:

Add s to F.

V = {L}



F = {}

n = L



# Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

 = Visited  
 = Frontier

V is visited, initially empty.

F is the frontier, with start.

Loop:

If F is empty, return failure.

Choose a node n to remove from F.

If n is a solution, return n.

Add n to V.

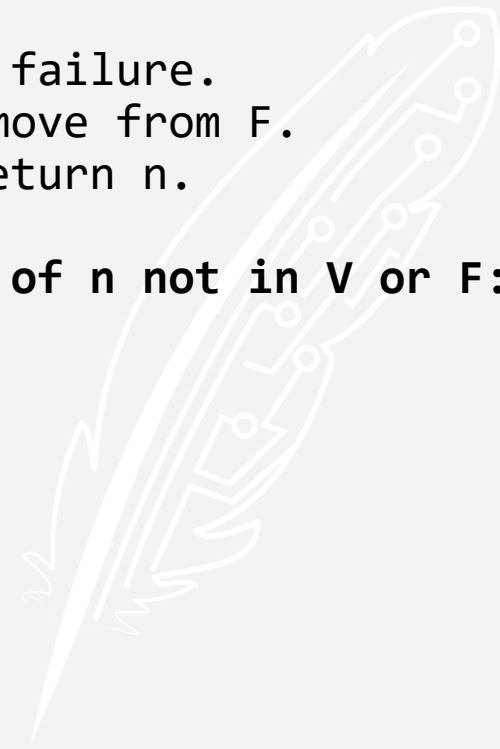
For every successor s of n not in V or F:

Add s to F.

V = {L}

F = {H}



n = L





# Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

 = Visited  
 = Frontier

V is visited, initially empty.

F is the frontier, with start.

Loop:

If F is empty, return failure.

Choose a node n to remove from F.

If n is a solution, return n.

Add n to V.

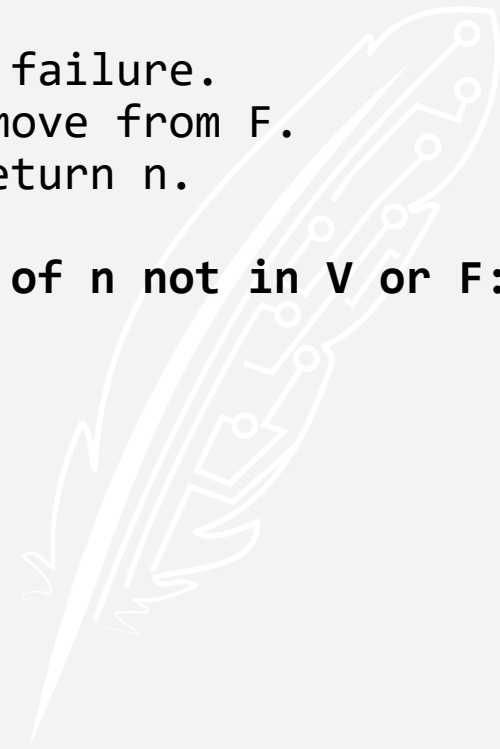
For every successor s of n not in V or F:

Add s to F.

V = {L}

F = {H, M}


n = L



# Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

 = Visited

 = Frontier

V is visited, initially empty.

F is the frontier, with start.

Loop:

If F is empty, return failure.

Choose a node n to remove from F.

If n is a solution, return n.

Add n to V.

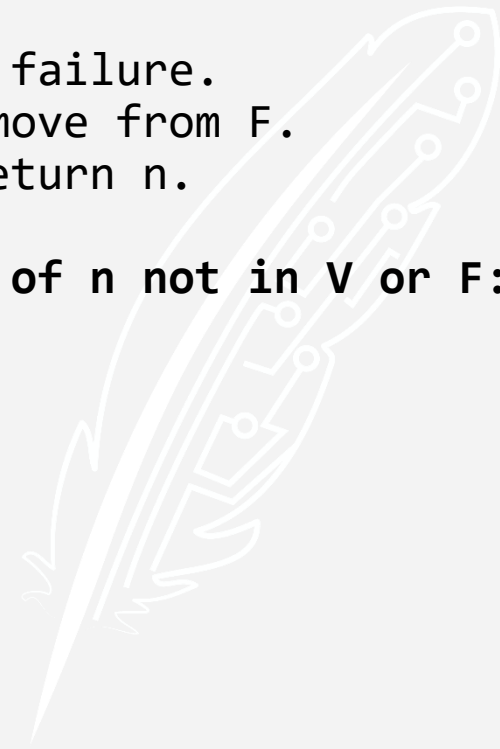
For every successor s of n not in V or F:

Add s to F.

V = {L}



F = {H, M, O}

n = L



# Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

 = Visited  
 = Frontier

V is visited, initially empty.

F is the frontier, with start.

Loop:

**If F is empty, return failure.**

Choose a node n to remove from F.

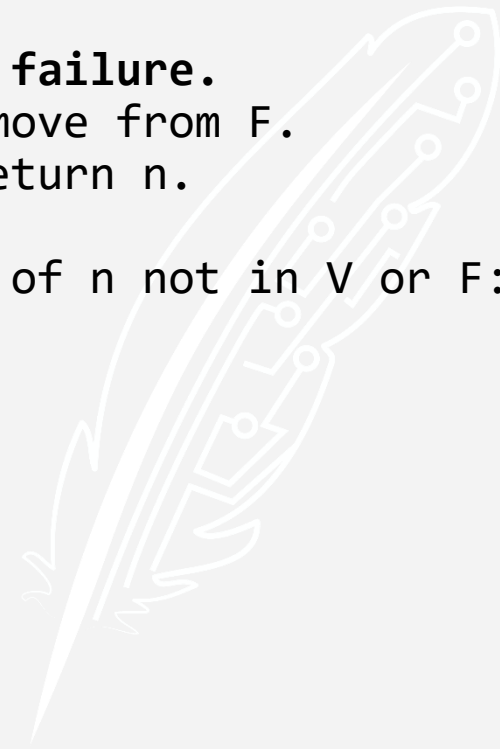
If n is a solution, return n.

Add n to V.

For every successor s of n not in V or F:



Add s to F.

V = {L}  
F = {H,M,O}  
n = L



# Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

 = Visited  
 = Frontier

V is visited, initially empty.

F is the frontier, with start.

Loop:

If F is empty, return failure.

**Choose a node n to remove from F.**

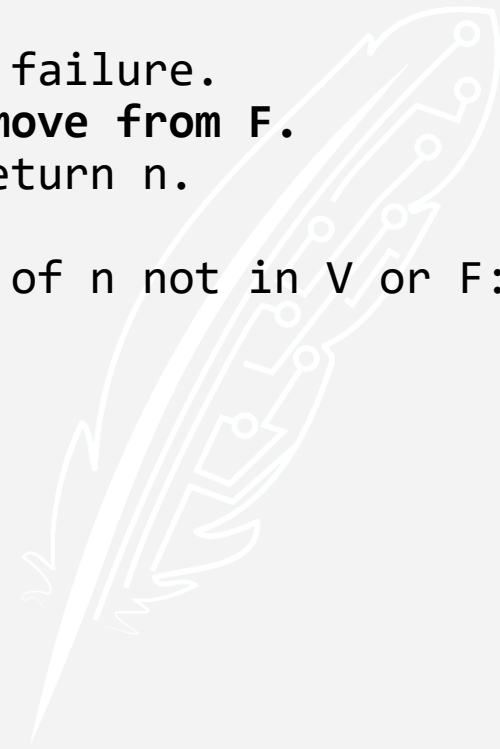
If n is a solution, return n.

Add n to V.

For every successor s of n not in V or F:



Add s to F.

V = {L}  
F = {H,M,O}  
n = L



# Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

 = Visited  
 = Frontier

V is visited, initially empty.

F is the frontier, with start.

Loop:

If F is empty, return failure.

**Choose a node n to remove from F.**

If n is a solution, return n.

Add n to V.

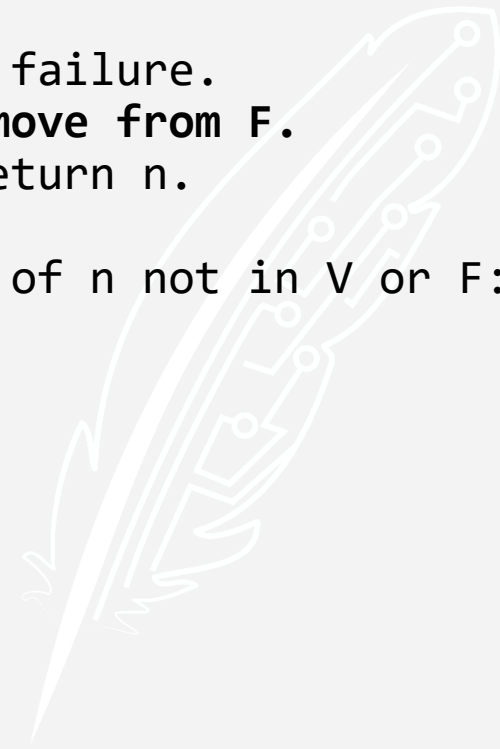
For every successor s of n not in V or F:

Add s to F.

V = {L}

F = {M, O}

n = H



# Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

 = Visited

 = Frontier

V is visited, initially empty.

F is the frontier, with start.

Loop:

If F is empty, return failure.

Choose a node n to remove from F.

If n is a **solution**, return n.

Add n to V.

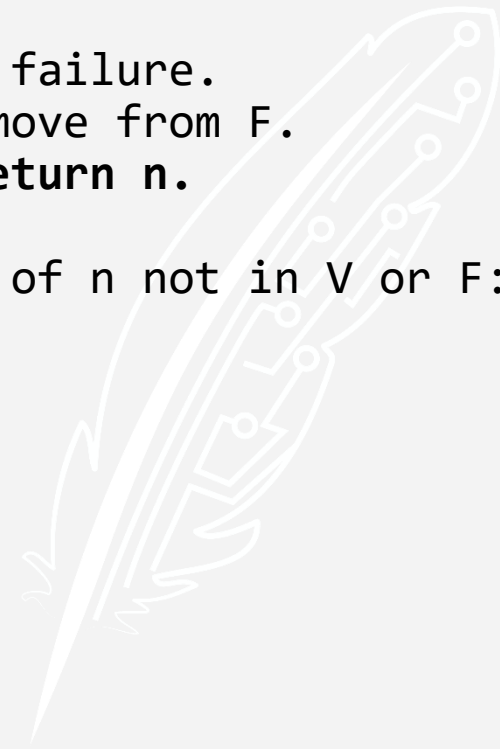
For every successor s of n not in V or F:

Add s to F.

V = {L}



F = {M, O}

n = H



# Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

 = Visited  
 = Frontier

V is visited, initially empty.

F is the frontier, with start.

Loop:

If F is empty, return failure.

Choose a node n to remove from F.

If n is a solution, return n.

Add n to V.

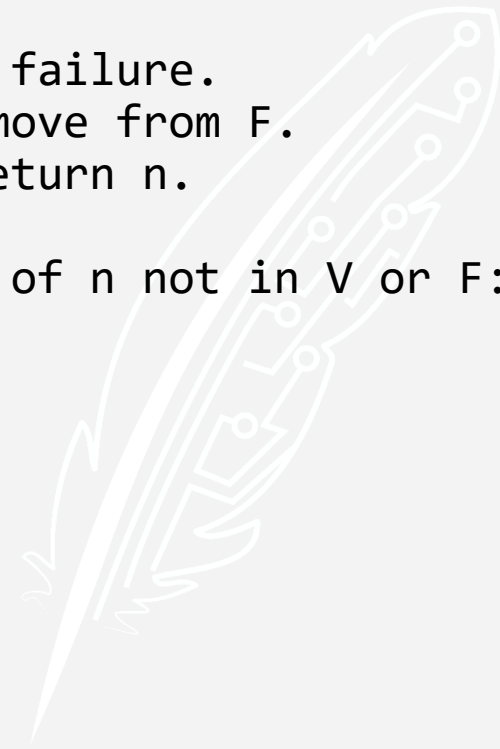
For every successor s of n not in V or F:

Add s to F.

V = {L}

F = {M, O}

n = H



# Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

 = Visited

 = Frontier

V is visited, initially empty.

F is the frontier, with start.

Loop:

If F is empty, return failure.

Choose a node n to remove from F.

If n is a solution, return n.

Add n to V.

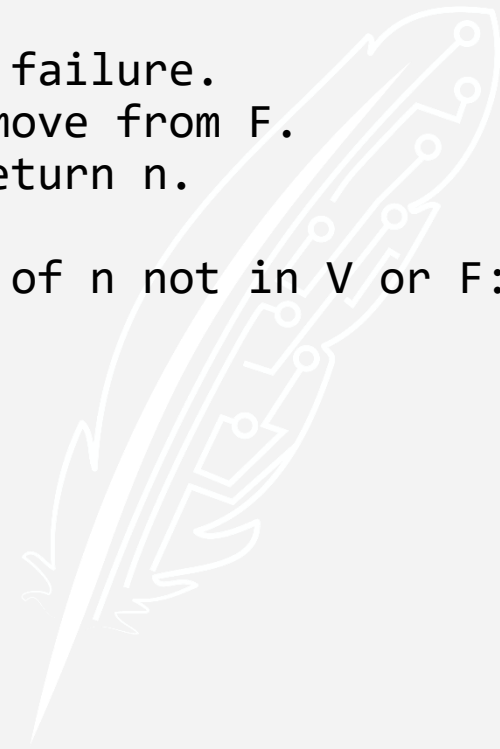
For every successor s of n not in V or F:

Add s to F.

V = {L,H}

F = {M,O}

n = H





# Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

 = Visited

 = Frontier

V is visited, initially empty.

F is the frontier, with start.

Loop:

If F is empty, return failure.

Choose a node n to remove from F.

If n is a solution, return n.

Add n to V.

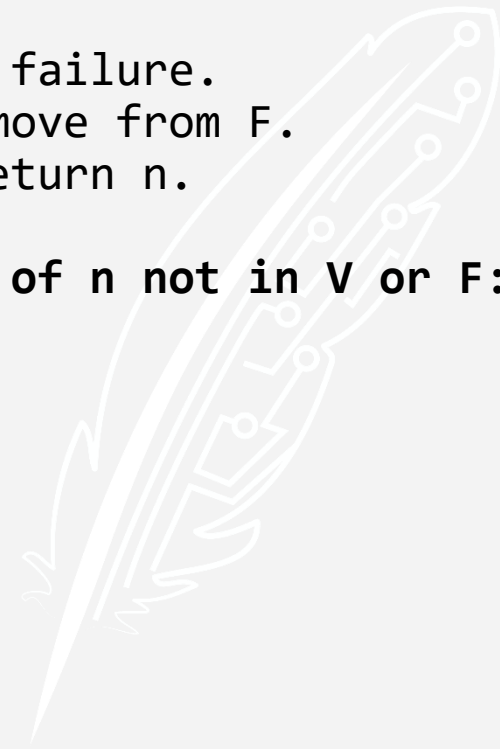
For every successor s of n not in V or F:

Add s to F.

V = {L,H}



F = {M,O}

n = H



# Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

 = Visited  
 = Frontier

V is visited, initially empty.

F is the frontier, with start.

Loop:

If F is empty, return failure.

Choose a node n to remove from F.

If n is a solution, return n.

Add n to V.

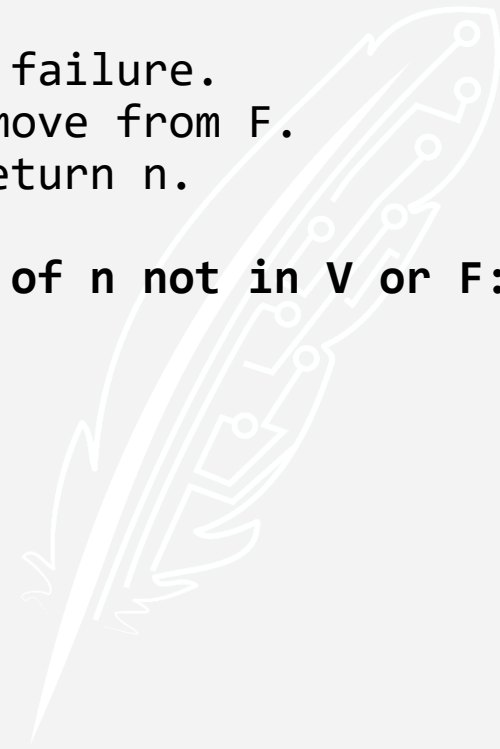
For every successor s of n not in V or F:

Add s to F.

V = {L,H}



F = {M,O,D}

n = H



# Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

 = Visited  
 = Frontier

V is visited, initially empty.

F is the frontier, with start.

Loop:

If F is empty, return failure.

Choose a node n to remove from F.

If n is a solution, return n.

Add n to V.

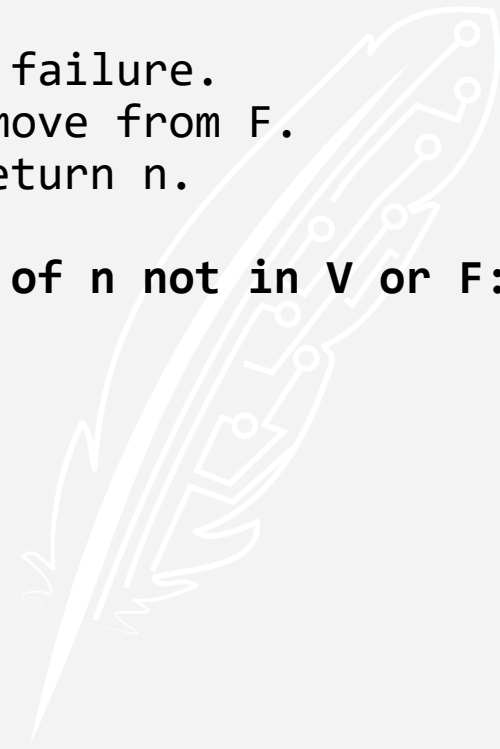
For every successor s of n not in V or F:

Add s to F.

V = {L,H}

F = {M,O,D,I}

n = H



# Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H

Frontier: M,O,D,I



# Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,D  
Frontier: M,O,I



# Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,D

Frontier: M,O,I,E,C



# Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,D,C  
Frontier: M,O,I,E



# Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,D,C  
Frontier: M,O,I,E,B





# Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,D,C,B  
Frontier: M,O,I,E



# Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,D,C,B

Frontier: M,O,I,E,G,A



# Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,D,C,B  
Frontier: M,O,I,E,A  
Goal found!



# Search

1. Let  $V$  be the set of visited nodes, empty.
2. Let  $F$  be the frontier, initially containing only the initial state.
3. Loop:
4.   If  $F$  is empty, return failure.
5.   Choose a node  $n$  to remove from  $F$ .
6.   If  $n$  is a solution, return  $n$ .
7.   Add  $n$  to  $V$ .
8.   For every successor  $s$  of  $n$  not in  $V$  or  $F$ :
9.     Add  $s$  to  $F$ .

# Search

1. Let  $V$  be the set of visited nodes, empty.
2. Let  $F$  be the frontier, initially containing only the initial state.
3. Loop:
4.   If  $F$  is empty, return failure.
5.   **Choose a node  $n$  to remove from  $F$ .**
6.   If  $n$  is a solution, return  $n$ .
7.   Add  $n$  to  $V$ .
8.   For every successor  $s$  of  $n$  not in  $V$  or  $F$ :
9.     Add  $s$  to  $F$ .

# Uninformed Search

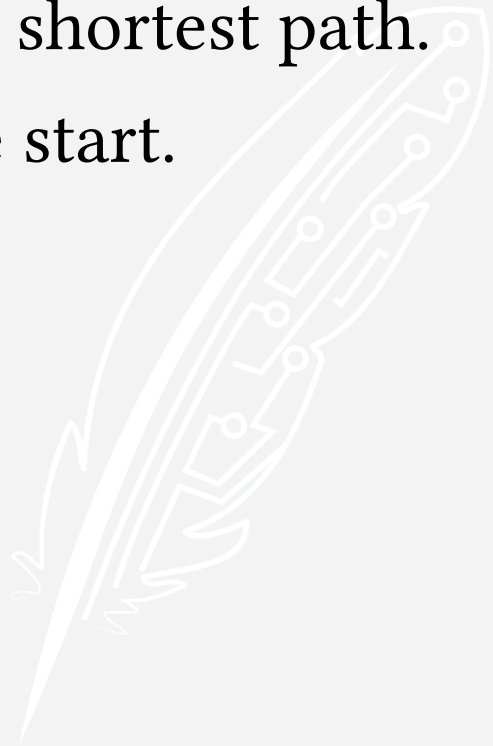
- Breadth First Search
- Depth First Search
- Uniform Cost Search
- Iterative Deepening Depth First Search
- Bidirectional Search



# Breadth First Search

When choosing a node from the frontier...

- always choose the node at the end of the shortest path.
- i.e. always choose the node closest to the start.
- i.e. always extend the shortest path.
- i.e. search as broadly as possible.



# Breadth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited:  
Frontier: L





# Breadth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L

Frontier: H,M,O



# Breadth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H

Frontier: M,O,D,I



# Breadth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,M

Frontier: O,D,I,P



# Breadth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,M,O  
Frontier: D,I,P,T



# Breadth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,M,O,D  
Frontier: I,P,T,E,C



# Breadth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,M,O,D,I  
Frontier: P,T,E,C



# Breadth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,M,O,D,I,P  
Frontier: T,E,C,U



# Breadth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,M,O,D,I,P,T  
Frontier: E,C,U,S





# Breadth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,M,O,D,I,P,T  
E

Frontier: C,U,S

# Breadth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,M,O,D,I,P,T  
E,C

Frontier: U,S,B

# Breadth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,M,O,D,I,P,T  
E,C,U

Frontier: S,B



# Breadth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,M,O,D,I,P,T  
E,C,U,S

Frontier: B,R



# Breadth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,M,O,D,I,P,T  
E,C,U,S,B

Frontier: R,G,A

# Breadth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,M,O,D,I,P,T  
E,C,U,S,B,R

Frontier: G,A



# Breadth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L, H, M, O, D, I, P, T  
E, C, U, S, B, R

Frontier: A  
Goal found!

# Breadth First Search

1. Let  $V$  be the set of visited nodes, empty.
2. Let  $F$  be the frontier, initially containing only the initial state.
3. Loop:
4.   If  $F$  is empty, return failure.
5.   **Choose a node  $n$  to remove from  $F$ .**
6.   If  $n$  is a solution, return  $n$ .
7.   Add  $n$  to  $V$ .
8.   For every successor  $s$  of  $n$  not in  $V$  or  $F$ :
9.     Add  $s$  to  $F$ .

**Use a first-in first-out (FIFO) queue for the frontier.**



# Breadth First Search

What can we say about the path discovered?

It is the shortest possible path (i.e. optimal).



# Complexity of BFS

Given a graph with  $n$  nodes and  $e$  edges,

- What is the time complexity?

$$O(n+e)$$

- What is the space complexity?

$$O(n)$$



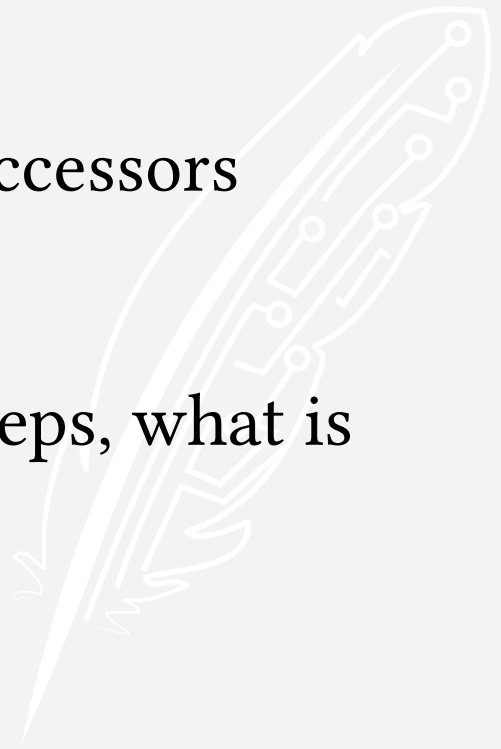
# Complexity of BFS

In practice, most search spaces are too large to explore fully, so we don't know  $n$  or  $e$ .

Instead, imagine that each node has  $b$  successors (i.e. a **branching factor** of  $b$ ).

If the shortest path to the solution is  $d$  steps, what is the time complexity of BFS?

$$O(b + b^2 + b^3 + \dots + b^d) = O(b^d)$$



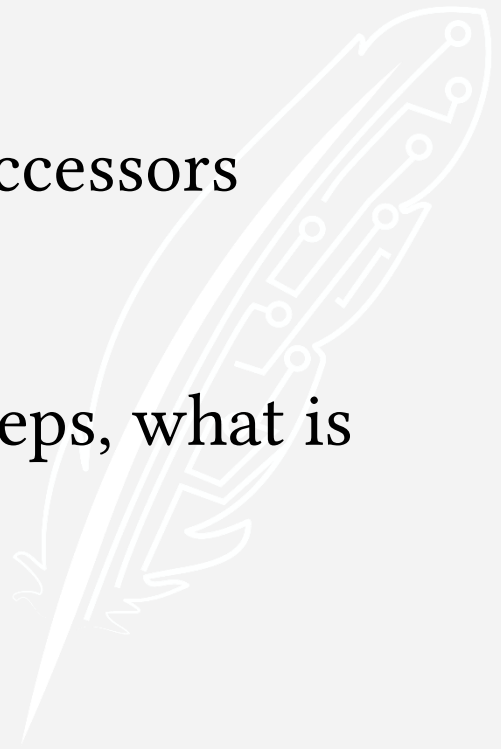
# Complexity of BFS

In practice, most search spaces are too large to explore fully, so we don't know  $n$  or  $e$ .

Instead, imagine that each node has  $b$  successors (i.e. a **branching factor** of  $b$ ).

If the shortest path to the solution is  $d$  steps, what is the space complexity of BFS?

$$O(b^{d-1} + b^d) = O(b^d)$$



# Depth First Search

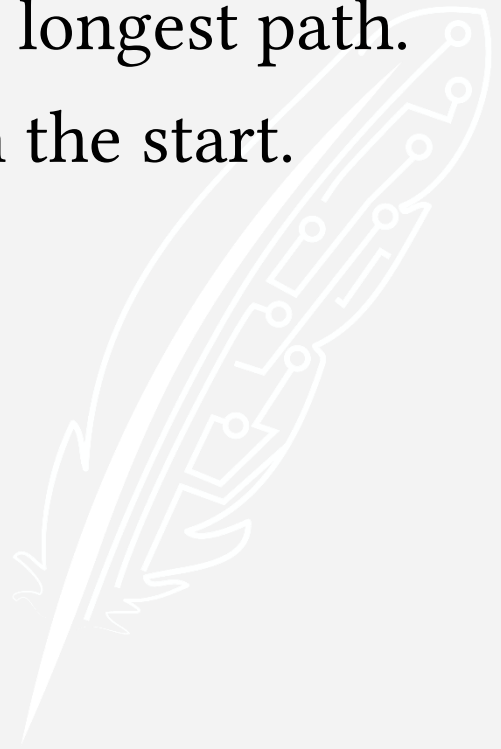
When choosing a node from the frontier...

always choose the node at the end of the longest path.

i.e. always choose the node farthest from the start.

i.e. always extend the longest path.

i.e. search as deeply as possible.



# Depth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited:  
Frontier: L



# Depth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L

Frontier: H,M,O



# Depth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H

Frontier: M,O,D,I





# Depth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,D

Frontier: M,O,I,E,C



# Depth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,D,E  
Frontier: M,O,I,C



# Depth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,D,E,I  
Frontier: M,O,C



# Depth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,D,E,I,M  
Frontier: O,C,P



# Depth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,D,E,I,M,P  
Frontier: O,C,U



# Depth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,D,E,I,M,P,U  
Frontier: O,C,T



# Depth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,D,E,I,M,P,U  
T

Frontier: O,C,S

# Depth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,D,E,I,M,P,U  
T,O

Frontier: C,S



# Depth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,D,E,I,M,P,U  
T,O

Frontier: C,S

Longest path (9) is a  
dead end!

# Depth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,D,E,I,M,P,U  
T,O

Frontier: C,S

Longest path (9) is a  
dead end!

Extend the next longest  
path (8).

# Depth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,D,E,I,M,P,U  
T,O,S

Frontier: C,R



# Depth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,D,E,I,M,P,U  
T,O,S,R

Frontier: C,Q



# Depth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,D,E,I,M,P,U  
T,O,S,R,Q

Frontier: C,N



# Depth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,D,E,I,M,P,U  
T,O,S,R,Q,N

Frontier: C,J

# Depth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,D,E,I,M,P,U  
T,O,S,R,Q,N,J

Frontier: C,F,K

# Depth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,D,E,I,M,P,U  
T,O,S,R,Q,N,J,F

Frontier: C,K,A,G



# Depth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,D,E,I,M,P,U  
T,O,S,R,Q,N,J,F  
A

Frontier: C,K,G,B

# Depth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,D,E,I,M,P,U  
T,O,S,R,Q,N,J,F  
A,B

Frontier: C,K,G

# Depth First Search

A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,D,E,I,M,P,U  
T,O,S,R,Q,N,J,F  
A,B,C

Frontier: K,G



# Depth First Search

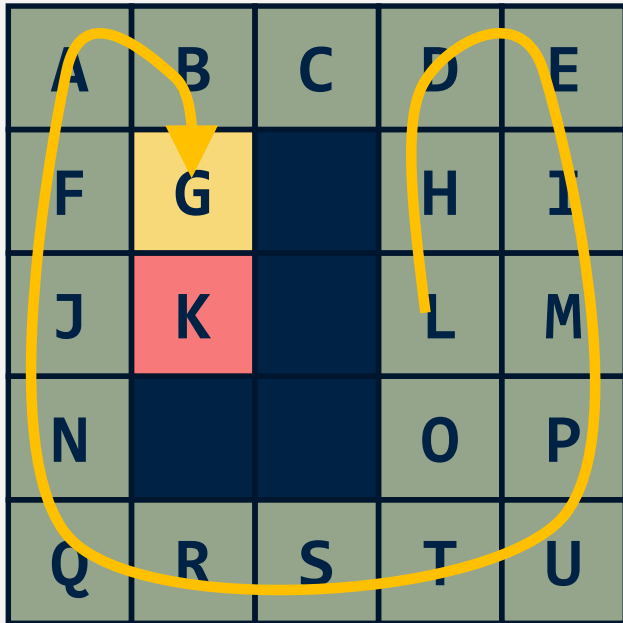
A	B	C	D	E
F	G		H	I
J	K		L	M
N			O	P
Q	R	S	T	U

Visited: L,H,D,E,I,M,P,U  
T,O,S,R,Q,N,J,F  
A,B,C

Frontier: K

Goal found!

# Depth First Search



Visited: L, H, D, E, I, M, P, U  
T, O, S, R, Q, N, J, F  
A, B, C

Frontier: K  
Goal found!

# Depth First Search

1. Let  $V$  be the set of visited nodes, empty.
2. Let  $F$  be the frontier, initially containing only the initial state.
3. Loop:
4.     If  $F$  is empty, return failure.
5.     **Choose a node  $n$  to remove from  $F$ .**
6.     If  $n$  is a solution, return  $n$ .
7.     Add  $n$  to  $V$ .
8.     For every successor  $s$  of  $n$  not in  $V$  or  $F$ :
9.         Add  $s$  to  $F$ .

**Use a first-in last-out (FILO) stack for the frontier.**

# Recursive Depth First Search

1. Let  $V$  be the set of visited nodes, empty.
2. Call  $\text{DFS}(V, \text{start})$ ;
3. Function  $\text{DFS}(V, n)$ :
4.     If  $n$  is a solution, return  $n$ .
5.     Add  $n$  to  $V$ .
6.     For every successor  $s$  of  $n$  not in  $V$ :
7.         Call  $\text{DFS}(V, s)$ .
8.     If no solution was found, fail.

**Uses the operating system's stack for the frontier.**

# Depth First Search

- If the search space is infinite, when will DFS fail?

It may run forever!

- Does DFS return the shortest path like BFS?

No!

- So why use DFS at all?





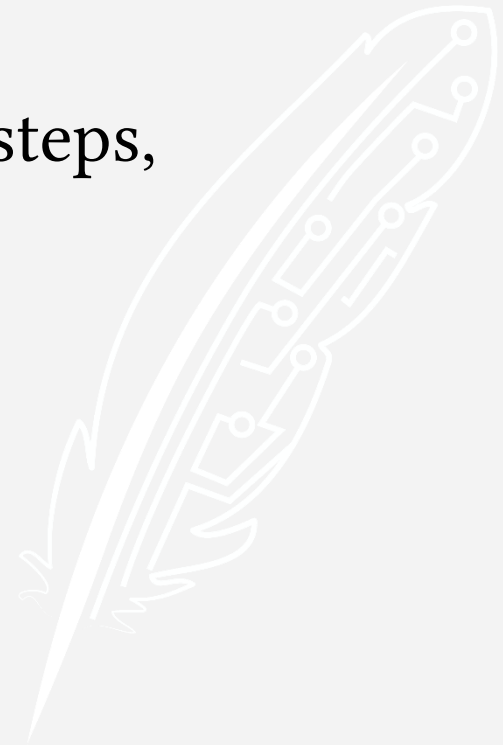
# Complexity of DFS

Given a state space in which...

- each node has  $b$  successors,
- the shortest path to the solution has  $d$  steps,
- the longest path in the space is  $m$  long

What is the time complexity of DFS?

$$O(b^m)$$



# Complexity of DFS

Given a state space in which...

- each node has  $b$  successors,
- the shortest path to the solution has  $d$  steps,
- the longest path in the space is  $m$  long
- the space graph can be treated like a tree  
(i.e. no repeats or we don't use  $V$  to track them)

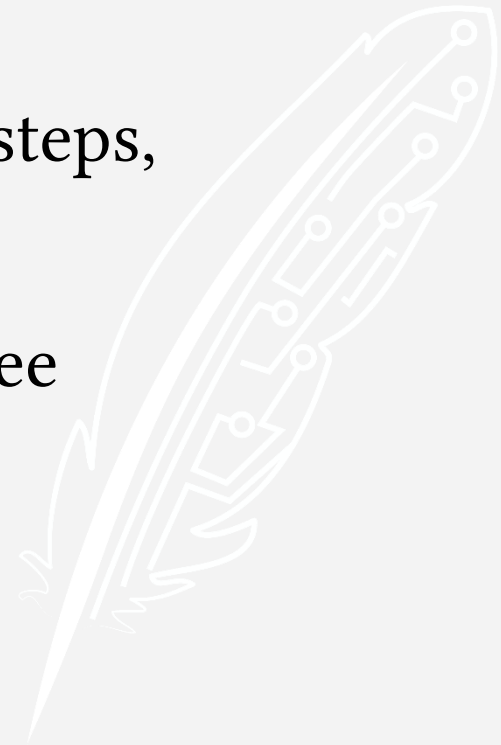
# Complexity of DFS

Given a state space in which...

- each node has  $b$  successors,
- the shortest path to the solution has  $d$  steps,
- the longest path in the space is  $m$  long
- the space graph can be treated like a tree

What is the space complexity of DFS?

$$O(bm)$$



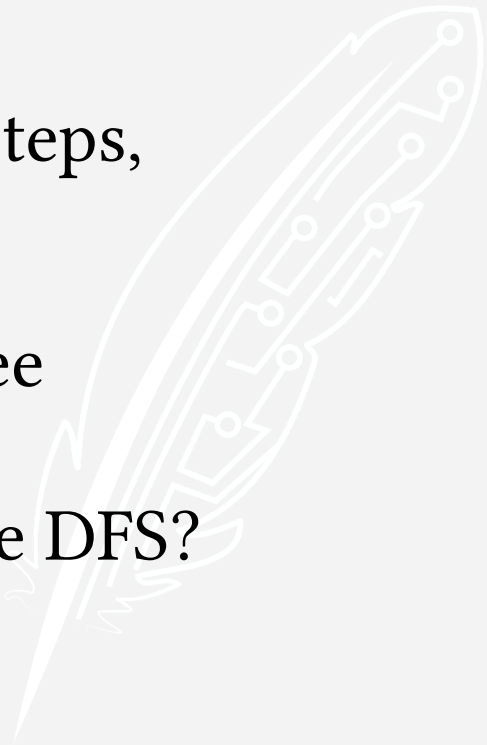
# Complexity of DFS

Given a state space in which...

- each node has  $b$  successors,
- the shortest path to the solution has  $d$  steps,
- the longest path in the space is  $m$  long
- the space graph can be treated like a tree

What is the space complexity of recursive DFS?

$$O(m)$$



# In Practice

Because of its low memory requirements, depth first search is often the starting place for many search algorithms.



# Uniform Cost Search

Used when not every step has the same cost (i.e. roads have different lengths).

A generalization of BFS, but instead of always expanding the shortest path (e.g. fewest roads), we always expand the lowest cost path (e.g. fewest miles).

Can't use a FIFO queue; now we need a priority queue.

# Iterative Deepening Search

Used when DFS is desirable, but the space may be infinite (or contain very long paths we want to avoid).

Modify DFS to never go deeper than a given limit  $d$ .

Start with  $d=1$ , then  $d=2$ , then  $d=3$ , etc. until a solution is found or  $d$  becomes infeasible.

DFS is a kind of iterative deepening where  $d=\infty$ .

# Iterative Deepening Search

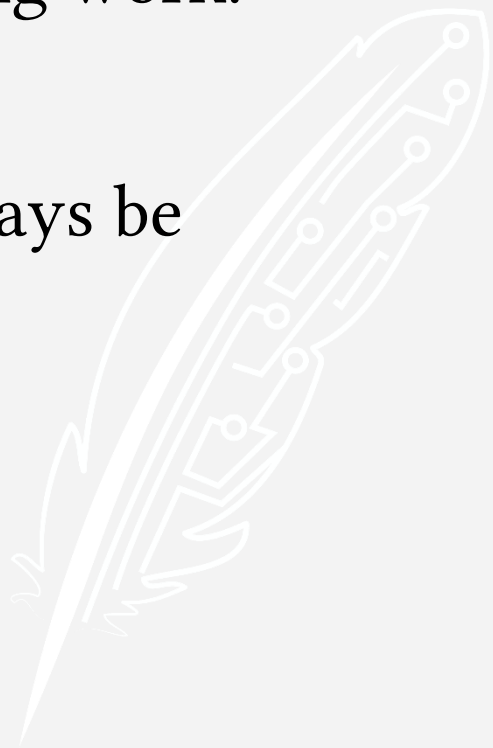
ID DFS means repeating DFS over and over with higher depth limits. This means repeating work. How does that affect its run time?

Not much, because the run time will always be dominated by the most recent search.

If the solution is at depth  $d$ :

DFS with depth limit  $d = O(b^d)$

IDDFS =  $O(b + b^2 + b^3 + \dots + b^d) = O(b^d)$





# Bidirectional Search

Search gets exponentially harder as it gets deeper.

Improvement: Run two searches simultaneously, one starting at the start and one starting at the goal. When their frontiers intersect, a path has been found.

Only works if we can move backwards.

Only works if we know which goal state is the end.