Uninformed Search

Stephen G. Ware

CS 463G

Narrative Intelligence LAB

University of Kentucky

Grid World

- **Initial State:** Square T
- **Actions:** Up, Right, Down, Left
- **Transition:** Move to new square
- **Goal:** Square C
- **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:
9.     Add s to F.
```

Search in Grid World

We are trying to find a path from `T` to `C`.

What is "a node?"

- Option 1: A node is path, e.g. `{T, P, O}`
  - The solution is already a path.
  - There are infinitely many paths.
- Option 2: A node is a single grid square, e.g. `P`
  - Need to reconstruct the path after finding a solution.
  - There are only n squares.

Search in Grid World

We are trying to find a path from `T` to `C`.

What is "a node?"

- Option 1: A node is path, e.g. `{T, P, O}`
  - The solution is already a path.
  - There are infinitely many paths.
- Option 2: A node is a single grid square, e.g. `P`
  - Need to reconstruct the path after finding a solution.
  - There are only n squares.
  - A square means "there is a path from `T` to this square."

Search

= 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:
    Add s to F.
```

Search


```
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:
    Add s to F.
```

= Visited = Frontier

Search


```
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:
    Add s to F.

V = {}
```

= Visited = Frontier

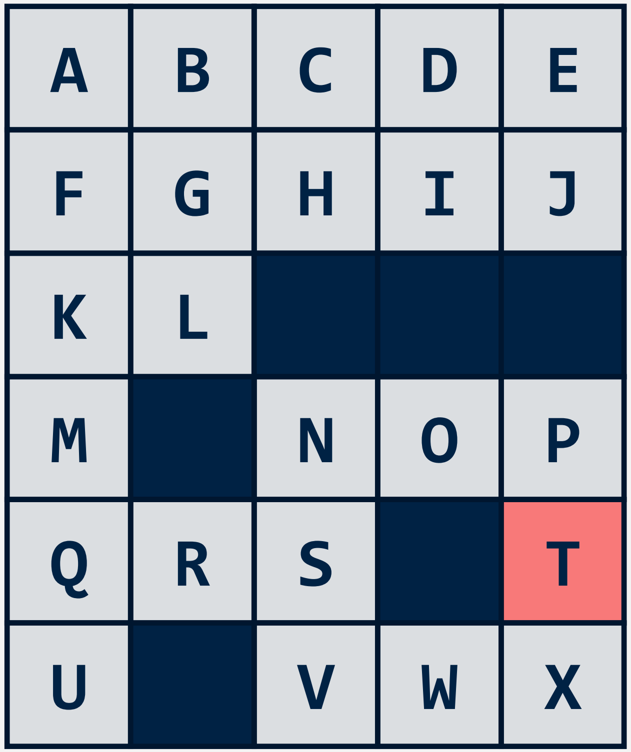
Search


```
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:
    Add s to F.

V = {}
```

= Visited = Frontier

Search



= 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:
    Add s to F.
```

V = {}
F = {T}

Search


```
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:
    Add s to F.
```

V = {}
F = {T}

= Visited = Frontier

Search


```
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:
    Add s to F.
```

V = {}
F = {T}

= Visited = Frontier

Search


```
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:
    Add s to F.
```

V = {}
F = {}
n = T

= Visited = Frontier

Search

■ = 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:
    Add s to F.
```

V = {}
F = {}
n = T

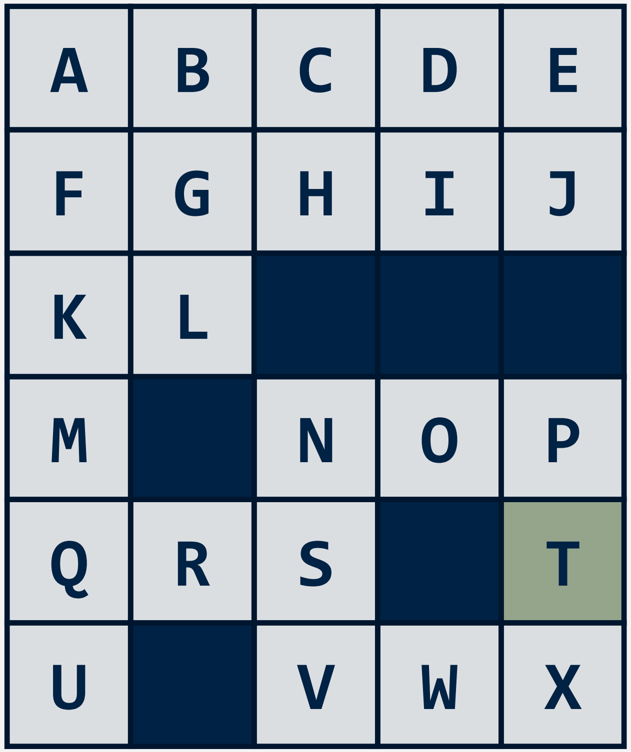
Search


```
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:
    Add s to F.
```

V = {}
F = {}
n = T

= Visited = Frontier

Search

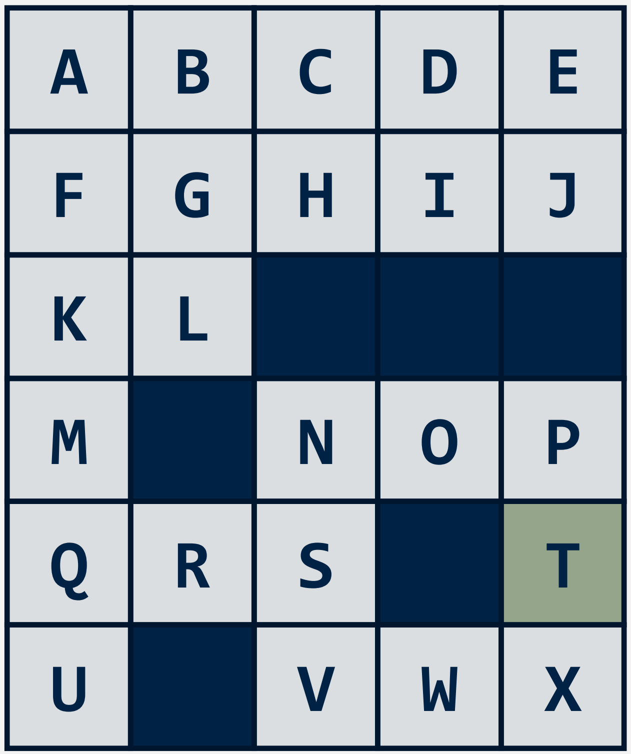



```
V is visited, initially empty.
F is the frontier, with start.
Loop:
  If F is empty, return failure.
  Choose a node n to remove from F.
  If n is a solution, return n.
  Add n to V.
  For every successor s of n not in V:
    Add s to F.
```

V = {T}
F = {}
n = T

= Visited = Frontier

Search

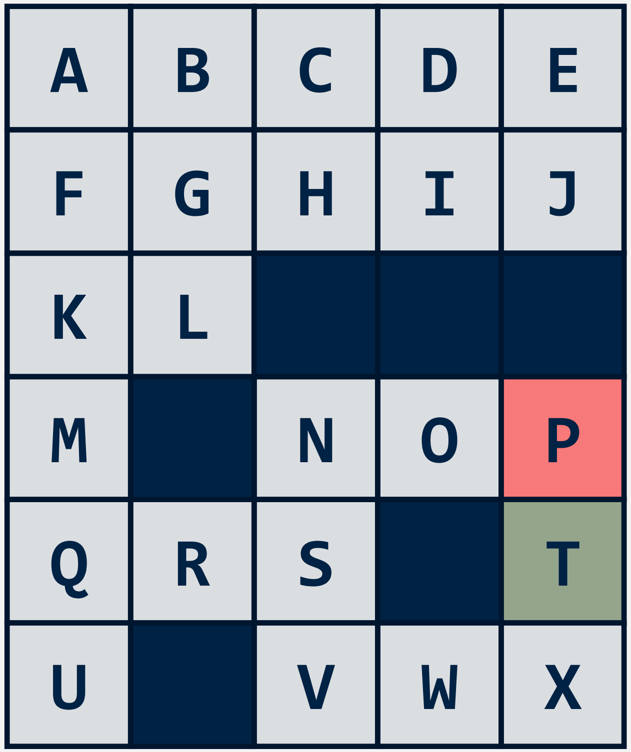



```
V is visited, initially empty.
F is the frontier, with start.
Loop:
  If F is empty, return failure.
  Choose a node n to remove from F.
  If n is a solution, return n.
  Add n to V.
  For every successor s of n not in V:
    Add s to F.
```

V = {T}
F = {}
n = T

= Visited = Frontier

Search



= 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:
    Add s to F.
```

V = {T}
F = {P}
n = T

Search

= 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:
    Add s to F.
```

V = {T}
F = {P,X}
n = T

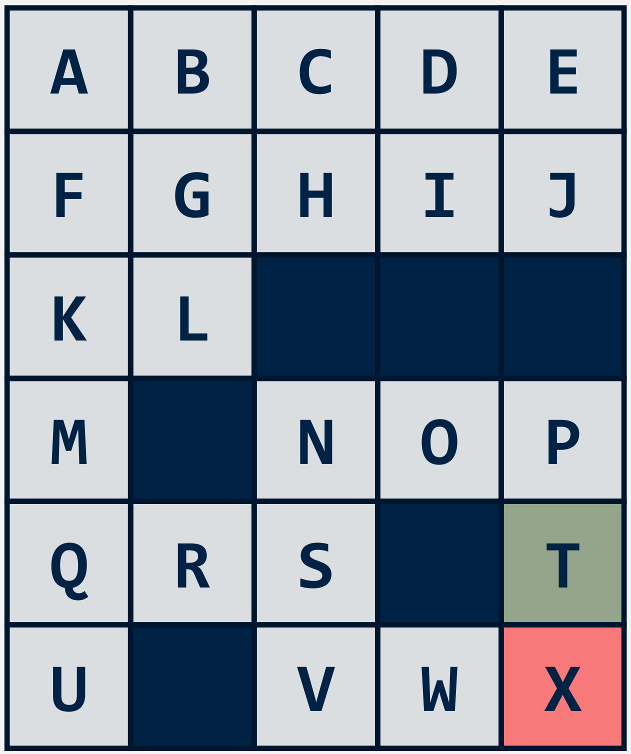
Search


```
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:
    Add s to F.
```

V = {T}
F = {P,X}

= Visited = Frontier

Search

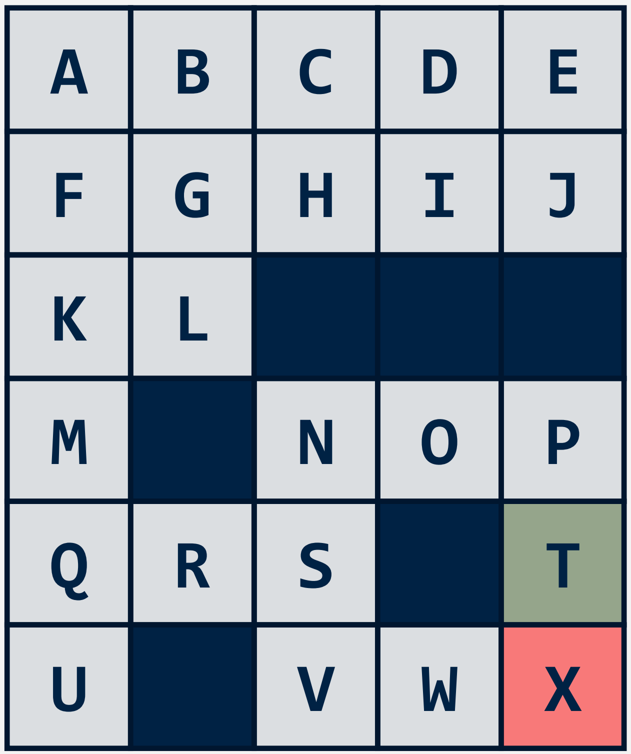



```
V is visited, initially empty.
F is the frontier, with start.
Loop:
  If F is empty, return failure.
  Choose a node n to remove from F.
  If n is a solution, return n.
  Add n to V.
  For every successor s of n not in V:
    Add s to F.
```

V = {T}
F = {X}
n = P

= Visited = Frontier

Search



= 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:
    Add s to F.
```

V = {T}
F = {X}
n = P

Search


```
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:
    Add s to F.
```

V = {T,P}
F = {X}
n = P

= Visited = Frontier

Search


```
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:
    Add s to F.
```

V = {T,P}
F = {X,O}
n = P

= Visited = Frontier

Search

Visited:
T,P

Frontier:
X,O

Search

Visited:
T,P,O

Frontier:
X,N

Search

Visited:
T,P,O,N

Frontier:
X,S

Search

Visited:
T,P,O,N,S

Frontier:
X,V,R

Search

Visited:
T,P,O,N,S,R

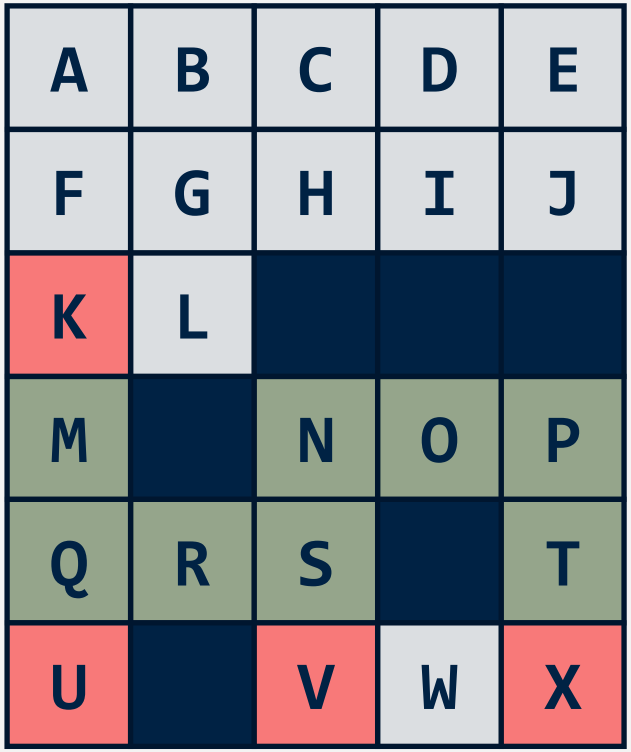
Frontier:
X,V,Q

Search

Visited:
T,P,O,N,S,R,Q

Frontier:
X,V,M,U

Search



Visited:
T,P,O,N,S,R,Q,M

Frontier:
X,V,U,K

Search

Visited:
T,P,O,N,S,R,Q,M,K

Frontier:
X,V,U,F,L

Search

Visited:
T,P,O,N,S,R,Q,M,K,L

Frontier:
X,V,U,F,G

Search

Visited:
T,P,O,N,S,R,Q,M,K,L,G

Frontier:
X,V,U,F,B,H

Search

Visited:
T,P,O,N,S,R,Q,M,K,L,G,B

Frontier:
X,V,U,F,H,A,C

Search

Visited:
T,P,O,N,S,R,Q,M,K,L,G,B

Frontier:
X,V,U,F,H,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:
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:
9.   Add s to F.

Uninformed Search

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

Visited:

Frontier:
T

Breadth First Search

Visited:
T

Frontier:
P,X

Breadth First Search

Visited:
T,P

Frontier:
X,O

Breadth First Search

Visited:
T,P,X

Frontier:
O,W

Breadth First Search

Visited:
T,P,X,O

Frontier:
W,N

Breadth First Search

Visited:
T,P,X,O,W

Frontier:
N,V

Breadth First Search

Visited:
T,P,X,O,W,N

Frontier:
V,S

Breadth First Search

Visited:
T,P,X,O,W,N,V

Frontier:
S

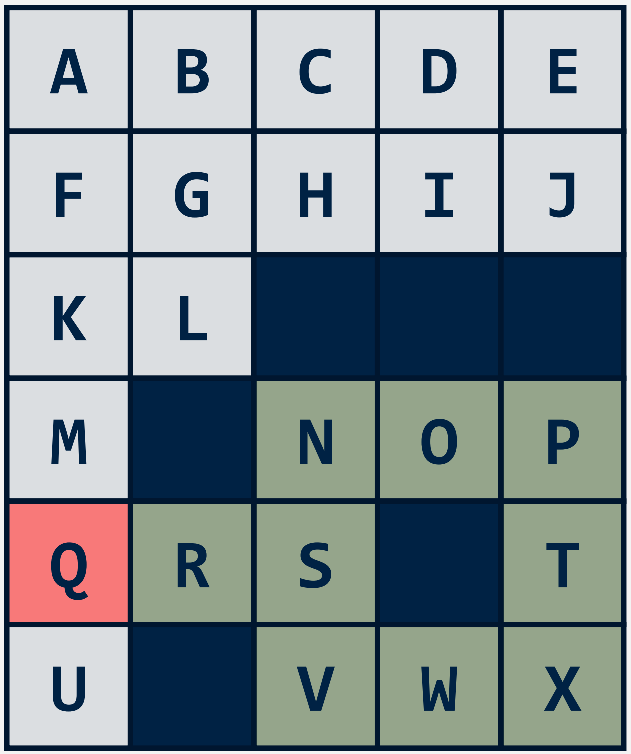
Notice we did not add a second copy of S.

Breadth First Search

Visited:
T,P,X,O,W,N,V,S

Frontier:
R

Breadth First Search



Visited:
T,P,X,O,W,N,V,S,R

Frontier:
Q

Breadth First Search

Visited:
T,P,X,O,W,N,V,S,R,Q

Frontier:
M,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		V	W	X

Visited:
T,P,X,O,W,N,V,S,R,Q,M

Frontier:
U,K

NIL

Breadth First Search

Visited:
T,P,X,O,W,N,V,S,R,Q,M,U

Frontier:
K

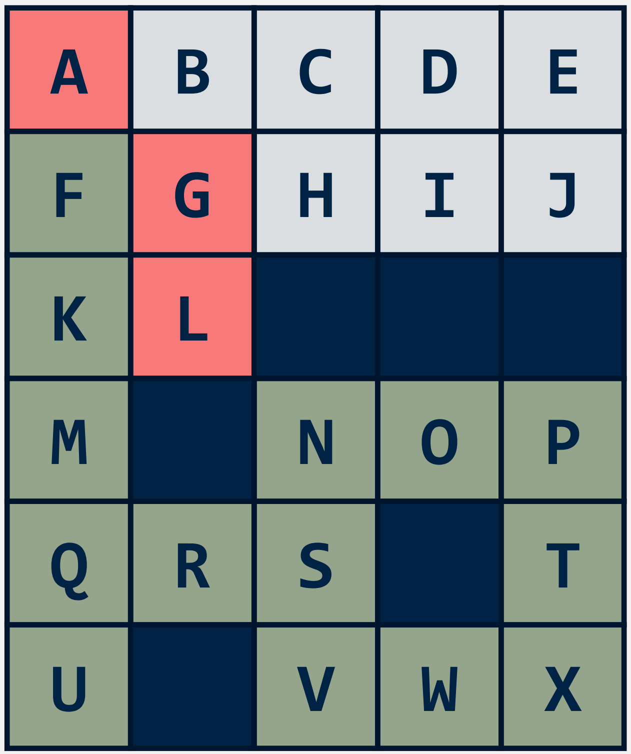
Breadth First Search

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

Visited:
T,P,X,O,W,N,V,S,R,Q,M,U,
K

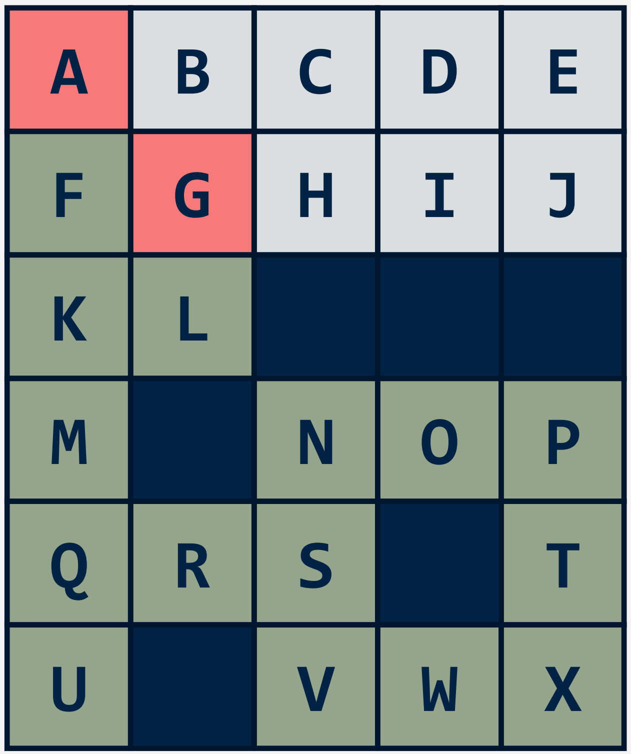
Frontier:
F,L

Breadth First Search



Visited:
T,P,X,O,W,N,V,S,R,Q,M,U,
K,F
Frontier:
L,A,G

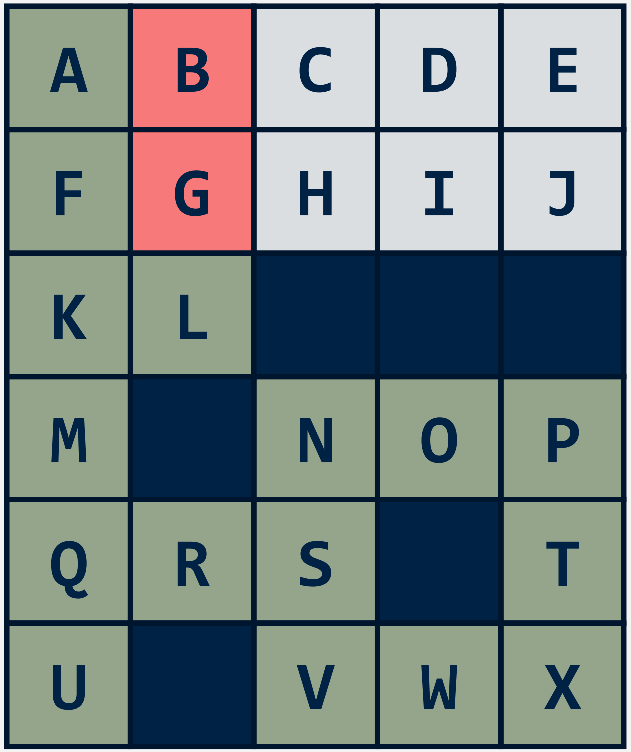
Breadth First Search



Visited:
T,P,X,O,W,N,V,S,R,Q,M,U,
K,F,L

Frontier:
A,G

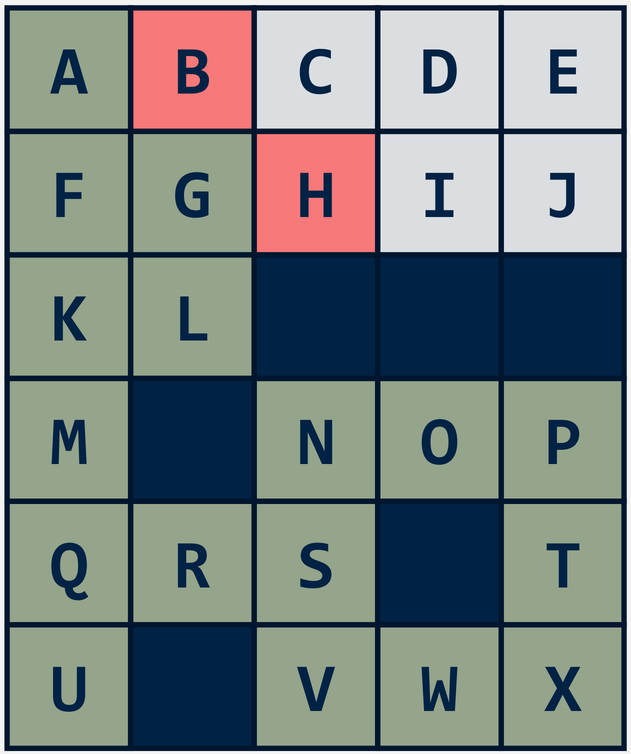
Breadth First Search



Visited:
T,P,X,O,W,N,V,S,R,Q,M,U,K,F,L,A

Frontier:
G,B

Breadth First Search



Visited:
T,P,X,O,W,N,V,S,R,Q,M,U,
K,F,L,A,G

Frontier:
B,H

Breadth First Search

Visited:
T,P,X,O,W,N,V,S,R,Q,M,U,
K,F,L,A,G,B

Frontier:
H,C

Breadth First Search

Visited:
T,P,X,O,W,N,V,S,R,Q,M,U,
K,F,L,A,G,B,H
Frontier:
C,I

Breadth First Search

Visited:
T,P,X,O,W,N,V,S,R,Q,M,U,
K,F,L,A,G,B,H
Frontier:
C,I

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:
9.     Add s to F.
```

Breadth First Search

```
1. Let V be the set of visited nodes, empty.
2. Let F be a queue, 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:
9.     Add s to F.
```

Breadth First Search

```
1. Let V be the set of visited nodes, empty.
2. Let F be a queue, 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:
9.     Add s to F unless s is already in F.
```

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?

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

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

Visited:

Frontier:
T

Depth First Search

Visited:
T

Frontier:
P,X

Depth First Search

Visited:
T,X

Frontier:
P,W

Depth First Search

Visited:
T,X,W

Frontier:
P,V

Depth First Search

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

Visited:
T,X,W,V

Frontier:
P,S

Depth First Search

Visited:
T,X,W,V,S

Frontier:
P,N,R

Depth First Search

Visited:
T,X,W,V,S,R

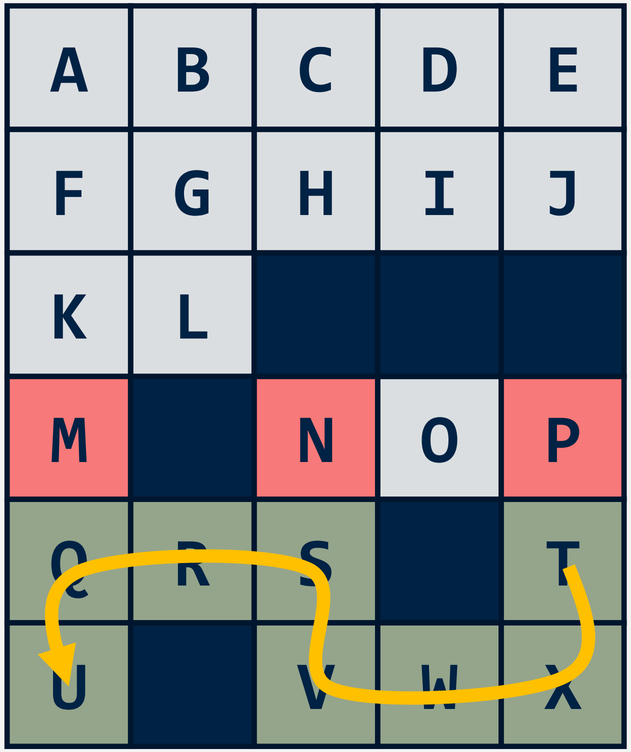
Frontier:
P,N,Q

Depth First Search

Visited:
T,X,W,V,S,R,Q

Frontier:
P,N,M,U

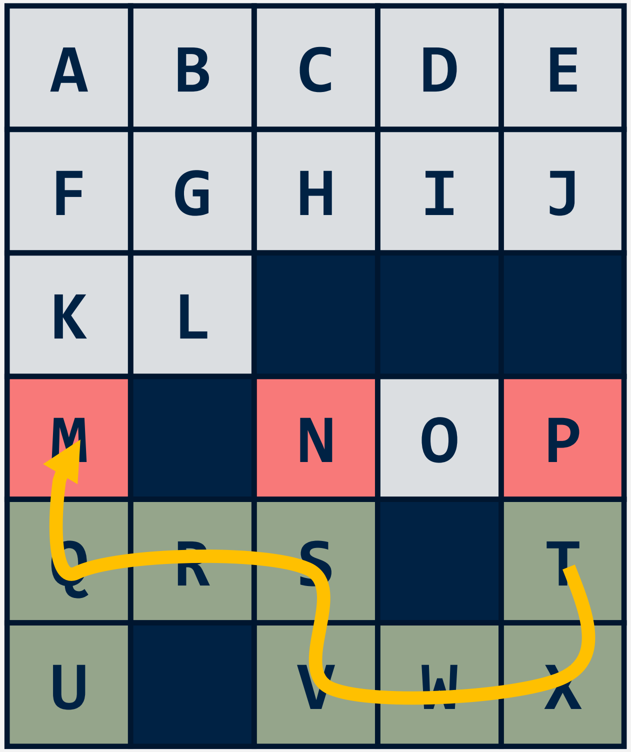
Depth First Search



Visited:
T,X,W,V,S,R,Q,U

Frontier:
P,N,M

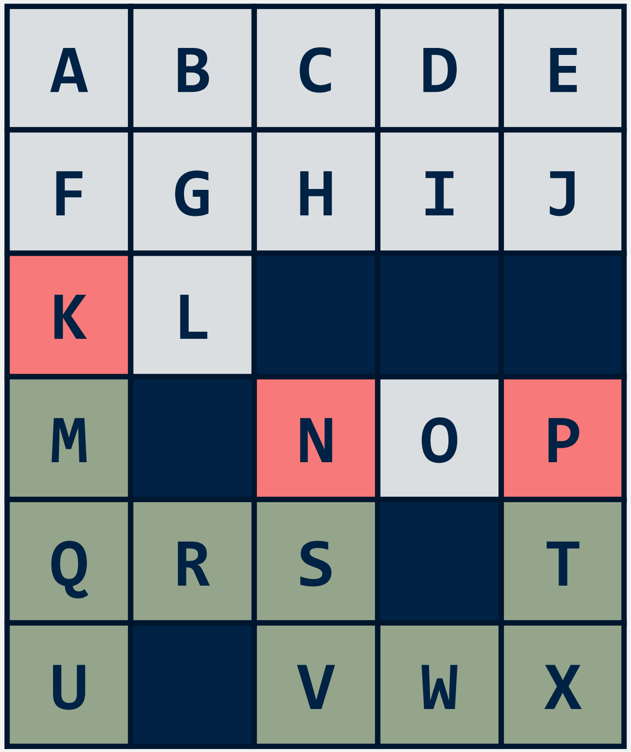
Depth First Search



Visited:
T,X,W,V,S,R,Q,U

Frontier:
P,N,M

Depth First Search



Visited:
T,X,W,V,S,R,Q,U,M

Frontier:
P,N,K

Depth First Search

Visited:
T,X,W,V,S,R,Q,U,M,K

Frontier:
P,N,F,L

Depth First Search

Visited:
T,X,W,V,S,R,Q,U,M,K,L

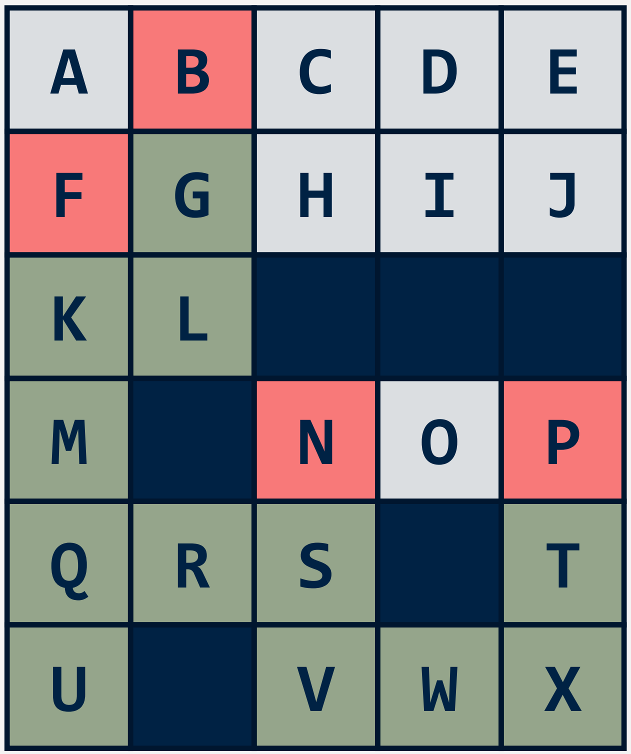
Frontier:
P,N,F,G

Depth First Search

Visited:
T,X,W,V,S,R,Q,U,M,K,L,G

Frontier:
P,N,F

Depth First Search



Visited:
T,X,W,V,S,R,Q,U,M,K,L,G

Frontier:
P,N,F,B

Depth First Search

Visited:
T,X,W,V,S,R,Q,U,M,K,L,G

Frontier:
P,N,F,B,H

We have found a
Longer path to F.

Depth First Search

Visited:
T,X,W,V,S,R,Q,U,M,K,L,G

Frontier:
P,N,B,H,F

Depth First Search

Visited:
T,X,W,V,S,R,Q,U,M,K,L,G,F

Frontier:
P,N,B,H,A

Depth First Search

Visited:
T,X,W,V,S,R,Q,U,M,K,L,G,F,A

Frontier:
P,N,H,B

Depth First Search

Visited:
T,X,W,V,S,R,Q,U,M,K,L,G,F,A,B
Frontier:
P,N,H,C

Depth First Search

Visited:
T,X,W,V,S,R,Q,U,M,K,L,G,F,A,B
Frontier:
P,N,H

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:
9.     Add s to F.
```

Depth First Search

```
1. Let V be the set of visited nodes, empty.
2. Let F be a stack, 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:
9.     Add s to F.
```

Depth First Search

```
1. Let V be the set of visited nodes, empty.
2. Let F be a stack, 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:
9.     Put s at the top of F.
```

Recursive Depth First Search

```
1. Let V be the set of visited nodes, empty.
2. Call DFS(V,start);
3. Function 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 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.

… and there's a way to address the time problem!

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 n.

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

DFS is a kind of iterative deepening where n start at ∞.

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?

If the solution is at depth 0: $O(1)$

If the solution is at depth 1: $O(1+b)$

If the solution is at depth 2: $O(1+b+b^2)$

If the solution is at depth 3: $O(1+b+b^2+b^3)$

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?

If the solution is at depth 0: $O(1)$

If the solution is at depth 1: $O(b)$

If the solution is at depth 2: $O(b^2)$

If the solution is at depth 3: $O(b^3)$

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?

If the solution is at depth 0: $O(b^0)$

If the solution is at depth 1: $O(b^1)$

If the solution is at depth 2: $O(b^2)$

If the solution is at depth 3: $O(b^3)$

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?

If the solution is at depth *d*: $O(b^d)$

Asymptotically, the same as breadth first search!

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).

Also known as Dijkstra's Algorithm when you run it for the whole graph, rather than a single destination.

Grid World

- **Initial State:** Square T
- **Actions:** Up, Right, Down, Left
- **Transition:** Move to new square
- **Goal:** Square C (for goal)
- **Cost:** 1 after ; 2 after
- **Performance:** Minimize cost

Uniform Cost Search

Visited:

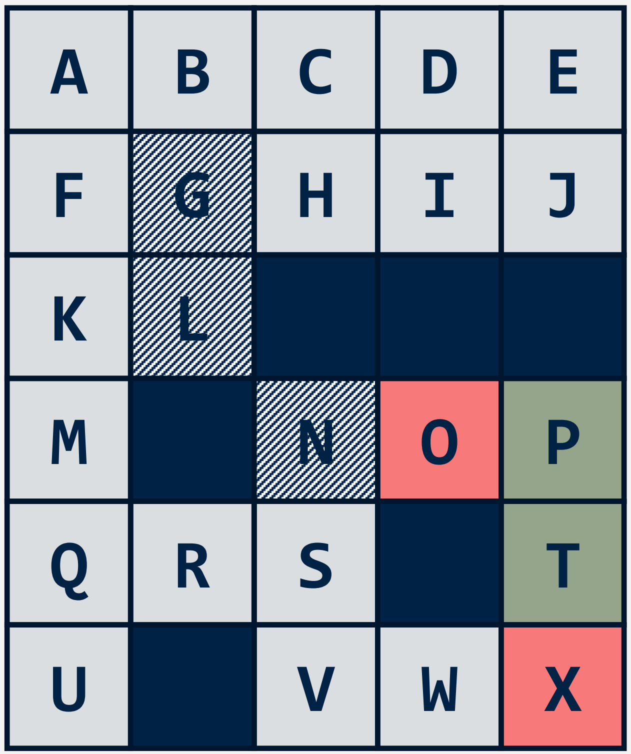
Frontier:
T=0

Uniform Cost Search

Visited:
T

Frontier:
P=1, X=1

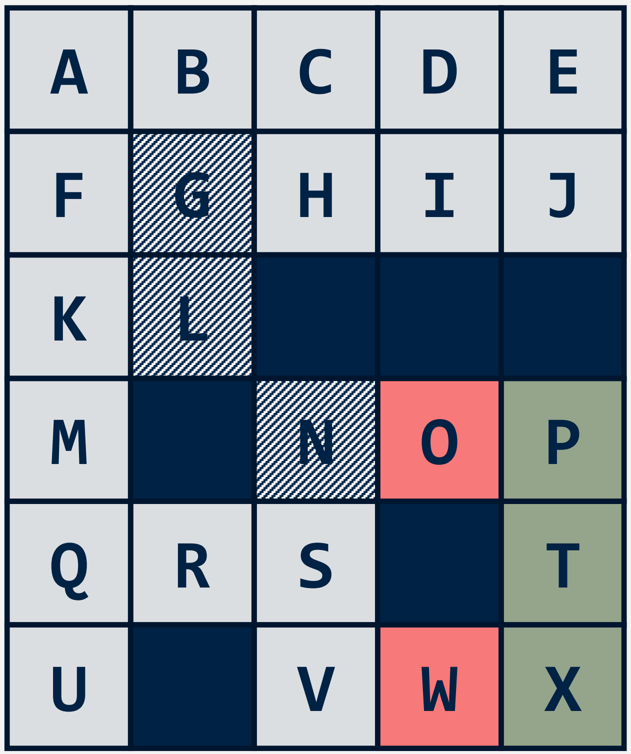
Uniform Cost Search



Visited:
T,P

Frontier:
X=1, O=2

Uniform Cost Search



Visited:
T,P,X

Frontier:
O=2, W=2

Uniform Cost Search

Visited:
T,P,X,O

Frontier:
W=2, N=3

Uniform Cost Search

Visited:
T,P,X,O,W

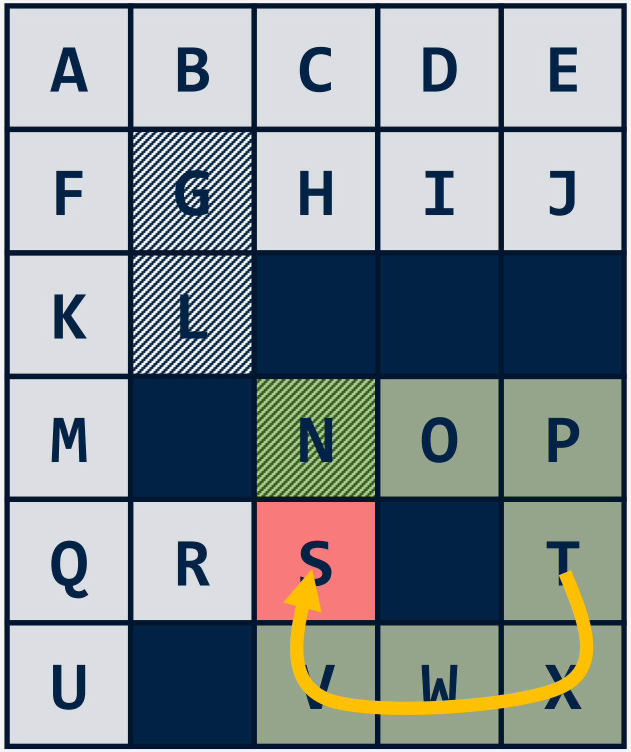
Frontier:
N=3, V=3

Uniform Cost Search

Visited:
T,P,X,O,W,N

Frontier:
V=3, S=5

Uniform Cost Search



Visited:
T,P,X,O,W,N,V

Frontier:
S=5

We found a lower cost path to S.

Uniform Cost Search

Visited:
T,P,X,O,W,N,V

Frontier:
S=4

We found a lower cost path to S.

Uniform Cost Search

Visited:
T,P,X,O,W,N,V,S

Frontier:
R=5

Uniform Cost Search

Visited:
T,P,X,O,W,N,V,S,R

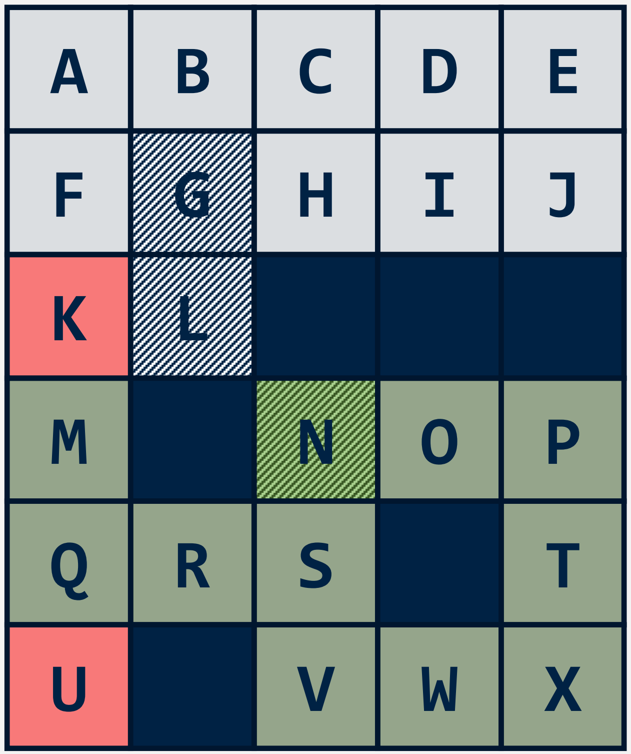
Frontier:
Q=6

Uniform Cost Search

Visited:
T,P,X,O,W,N,V,S,R,Q

Frontier:
M=7, U=7

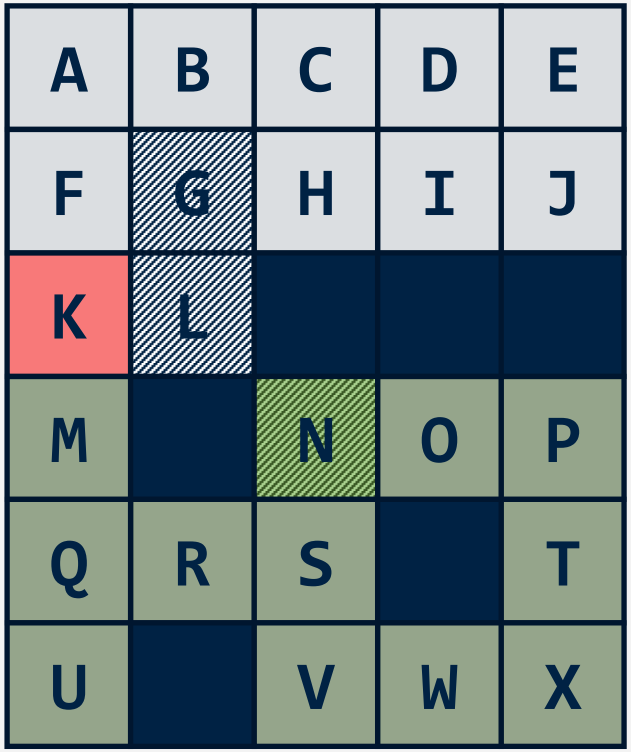
Uniform Cost Search



Visited:
T,P,X,O,W,N,V,S,R,Q,M

Frontier:
U=7, K=8

Uniform Cost Search



Visited:
T,P,X,O,W,N,V,S,R,Q,M,U

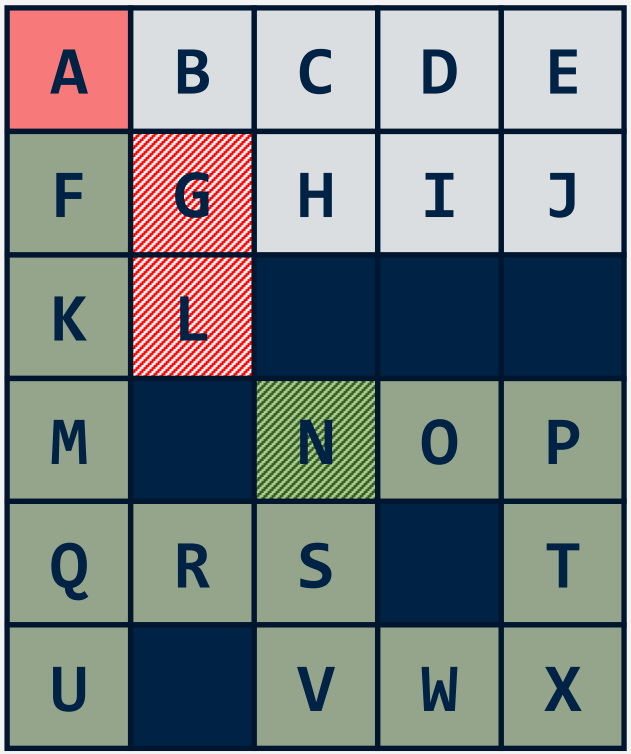
Frontier:
K=8

Uniform Cost Search

Visited:
T,P,X,O,W,N,V,S,R,Q,M,U,K

Frontier:
F=9, L=9

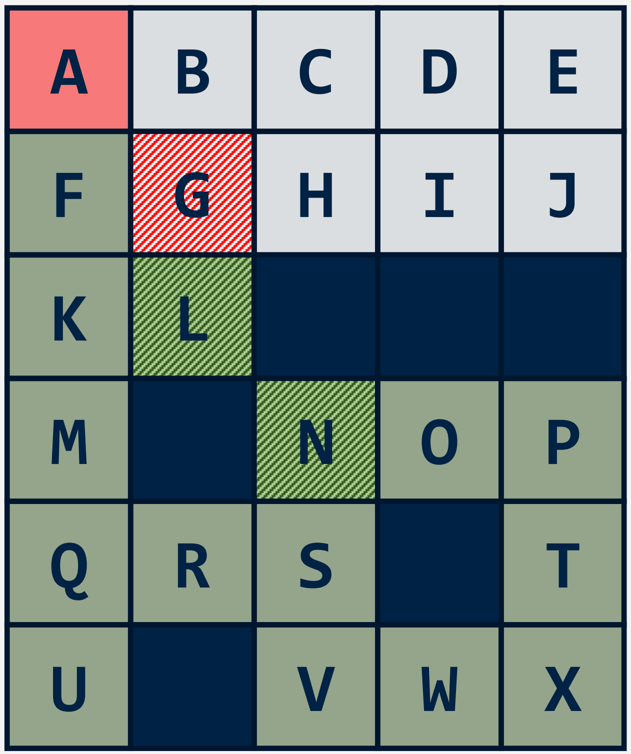
Uniform Cost Search



Visited:
T,P,X,O,W,N,V,S,R,Q,M,U,K,F

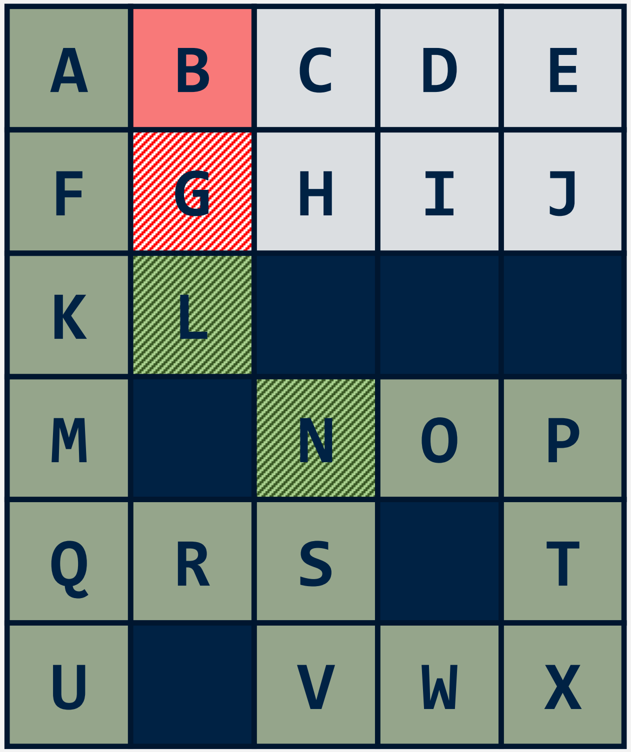
Frontier:
L=9, A=10, G=10

Uniform Cost Search



Visited:
T,P,X,O,W,N,V,S,R,Q,M,U,K,
F,L
Frontier:
A=10, G=10

Uniform Cost Search



Visited:
T,P,X,O,W,N,V,S,R,Q,M,U,K,
F,L,A
Frontier:
G=10, B=11

Uniform Cost Search

Visited:
T,P,X,O,W,N,V,S,R,Q,M,U,K,F,L,A,G

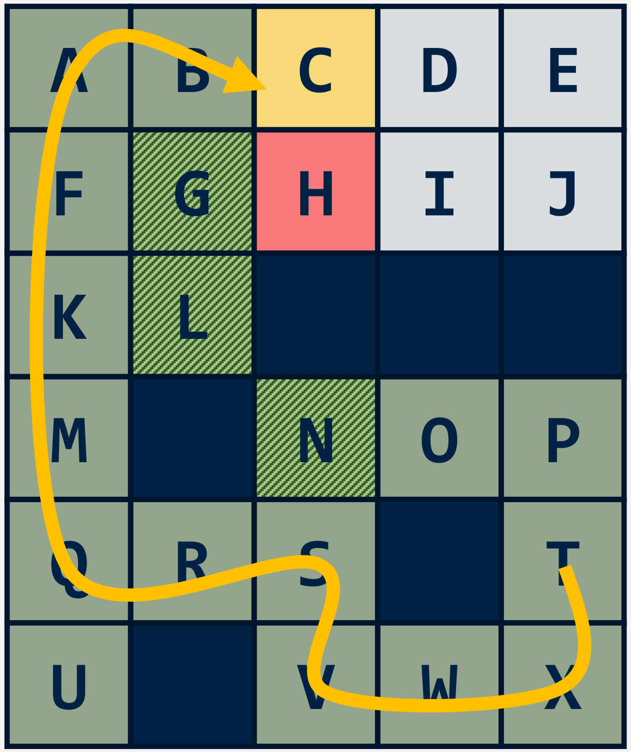
Frontier:
B=11, H=12

Uniform Cost Search

Visited:
T,P,X,O,W,N,V,S,R,Q,M,U,K,
F,L,A,G,B

Frontier:
H=12, C=12

Uniform Cost Search



Visited:
T,P,X,O,W,N,V,S,R,Q,M,U,K,
F,L,A,G,B

Frontier:
H=12

Goal found!

Uniform 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:
9.     Add s to F.
```

Uniform Cost Search

```
1. Let V be the set of visited nodes, empty.
2. Let F be a min priority queue, 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:
9.     Add s to F with its cost as the key.
       (or decrease key of s if it is in F)
```

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.