CSE 250
Data Structures

Dr. Eric Mikida
epmikida@buffalo.edu
208 Capen Hall

Lec 18: Mazes

Announcements

- WA3 due Sunday @ 11:59PM
  - See Piazza post @261 for more resources/tips

Recap

Stacks: Last In First Out (LIFO)

- Push (put item on top of the stack)
- Pop (take item off top of stack)
- Peek (peek at top of stack)

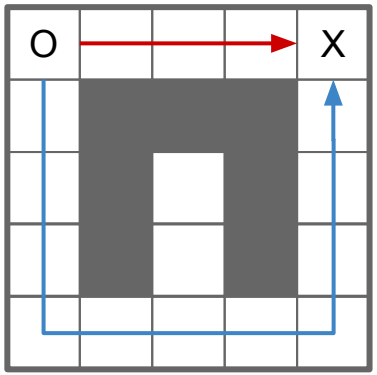
Queues: First in First Out (FIFO)

- Enqueue (put item on the end of the queue)
- Dequeue (take item off the front of the queue)
- Peek (peek at the front of the queue)

Mazes

O is the start, **X** is the objective

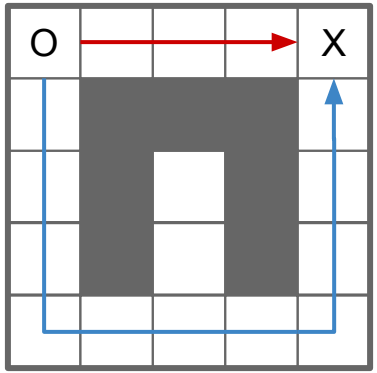
- There may be multiple paths
- Generally, we want the shortest



Approach 1: Take the first available route in one direction

- Right, Down, Left, or Up
- Down, Right, Up, or Left

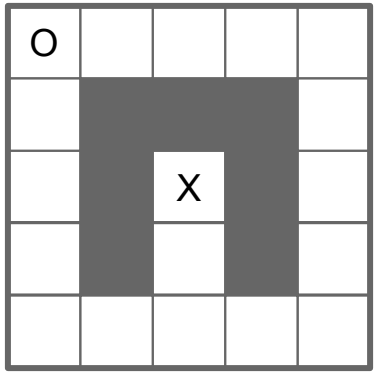
Mazes



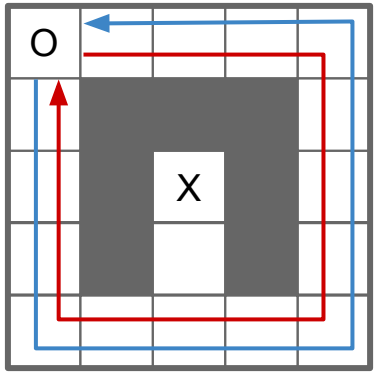
How do you know which one is best?

Is there anything wrong with this algorithm?

Mazes



Mazes



Priority order doesn't guarantee exploring the entire maze

Formalizing Maze-Solving

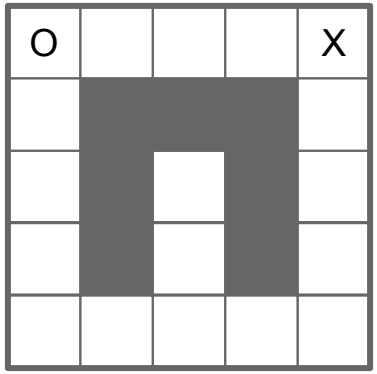
Inputs:

- The map: an *n* x *m* grid of squares which are either filled or empty
- The **O** is at position *start*
- The **X** is at position *dest*

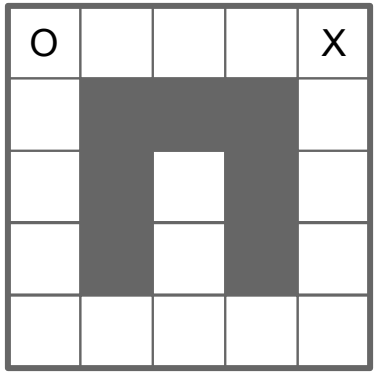
Goal: Compute `steps(start, dest)`, the minimum number of steps from start to end.

How do we define the `steps` *function?*

Mazes

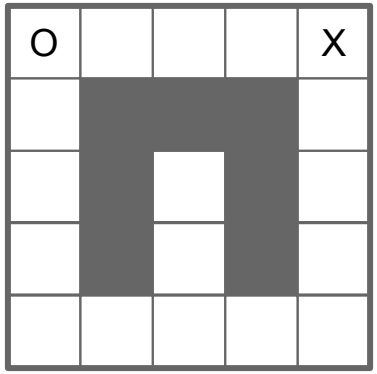


Mazes



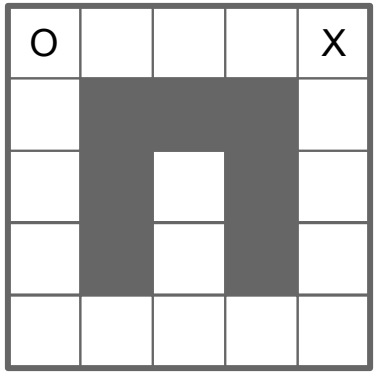
How many steps are required if we are already on the X?

Mazes



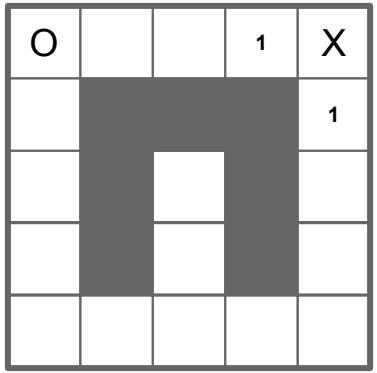
How many steps are required if we are already on the X*?* **0**

Mazes



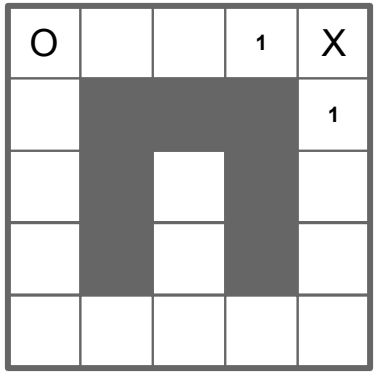
How many steps are required if we are right next to X?

Mazes



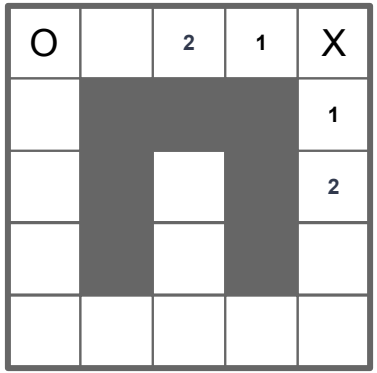
How many steps are required if we are right next to X*?* **1**

Mazes

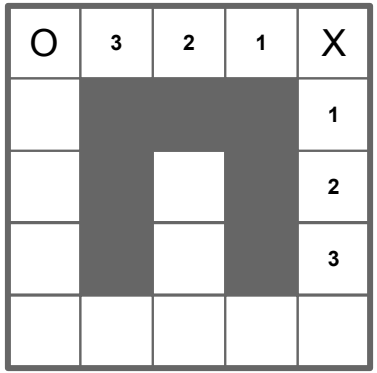


And the squares next to those?

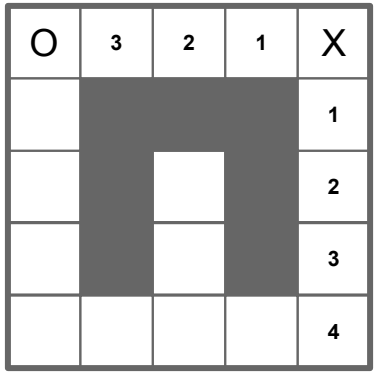
Mazes



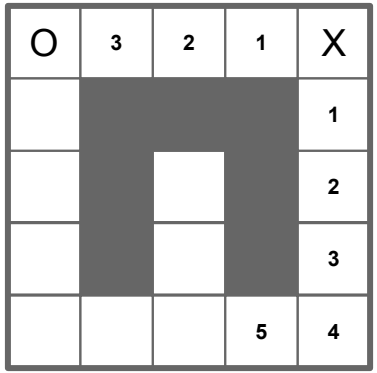
Mazes



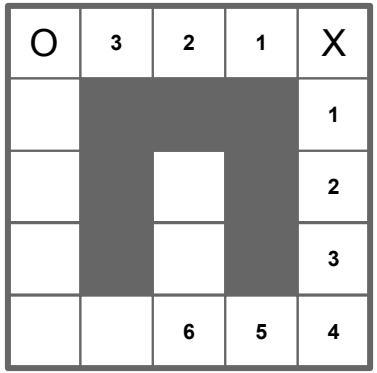
Mazes



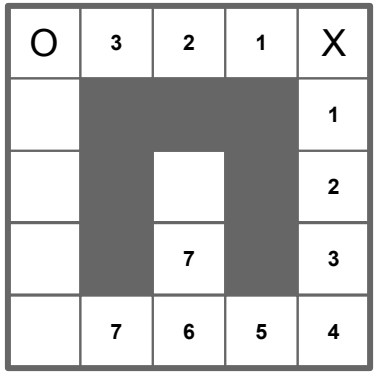
Mazes



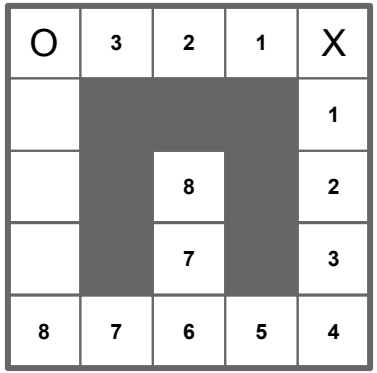
Mazes



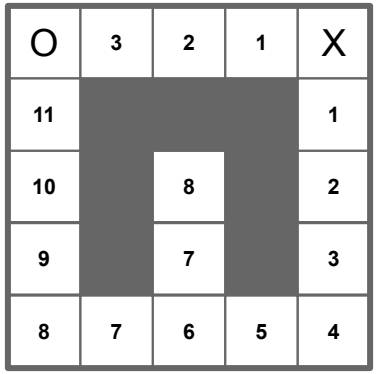
Mazes



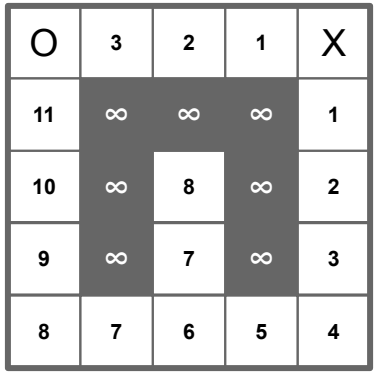
Mazes



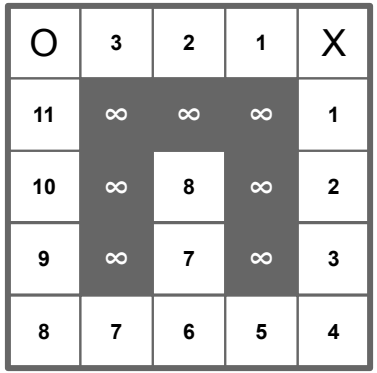
Mazes



Mazes



Mazes



So what is the number of steps from O to X?

Mazes

O	3	2	1	X
11	∞	∞	∞	1
10	∞	8	∞	2
9	∞	7	∞	3
8	7	6	5	4

So what is the number of steps from O *to* X*?*

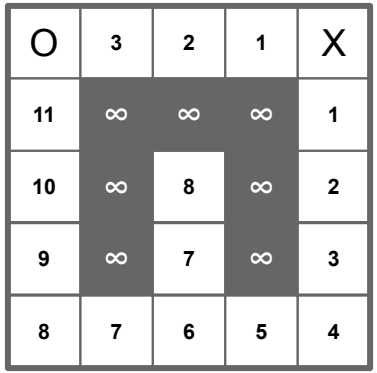
4 (min of neighbors + 1)

Mazes

O	3	2	1	X
11	∞	∞	∞	1
10	∞	8	∞	2
9	∞	7	∞	3
8	7	6	5	4

Does this solution remind you of anything?

Mazes



Does this solution remind you of anything?

Recursion!

Mazes

$$steps(\texttt{pos}, \texttt{dest}) = \begin{cases} 0 & \textbf{if } \texttt{pos} = \texttt{dest} \\ \infty & \textbf{if } is\_filled(\texttt{pos}) \\ 1 + min\_adjacent(\texttt{pos}, \texttt{dest}) & \textbf{otherwise} \end{cases}$$

where...

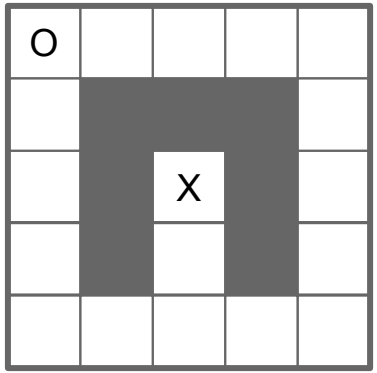
$$min\_adjacent(\texttt{pos}, \texttt{dest}) = \min \begin{cases} steps(moveRight(\texttt{pos}), \texttt{dest}) \\ steps(moveDown(\texttt{pos}), \texttt{dest}) \\ steps(moveLeft(\texttt{pos}), \texttt{dest}) \\ steps(moveUp(\texttt{pos}), \texttt{dest}) \end{cases}$$

Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_filled(pos) then return ∞
  else return 1 + min of
    steps(moveRight(pos), dest)
    steps(moveDown(pos), dest)
    steps(moveLeft(pos), dest)
    steps(moveUp(pos), dest)
```

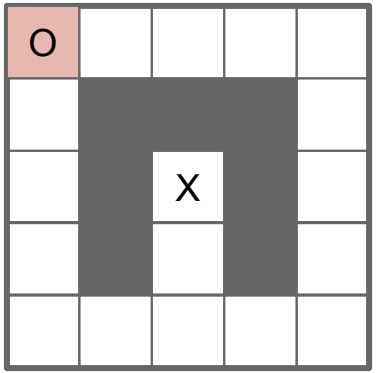
Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_filled(pos) then return ∞
  else return 1 + min of
    steps(moveRight(pos), dest)
    steps(moveDown(pos), dest)
    steps(moveLeft(pos), dest)
    steps(moveUp(pos), dest)
```



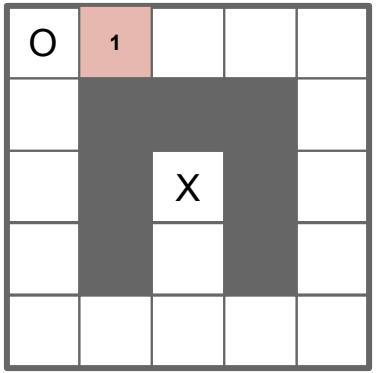
Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_filled(pos) then return ∞
  else return 1 + min of
    steps(moveRight(pos), dest)
    steps(moveDown(pos), dest)
    steps(moveLeft(pos), dest)
    steps(moveUp(pos), dest)
```



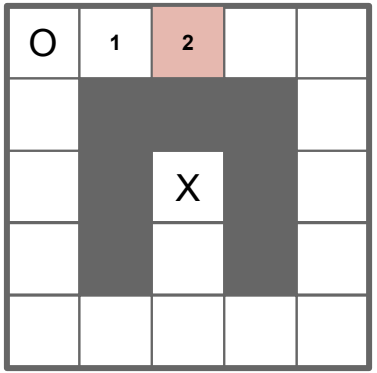
Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_filled(pos) then return ∞
  else return 1 + min of
    steps(moveRight(pos), dest)
    steps(moveDown(pos), dest)
    steps(moveLeft(pos), dest)
    steps(moveUp(pos), dest)
```



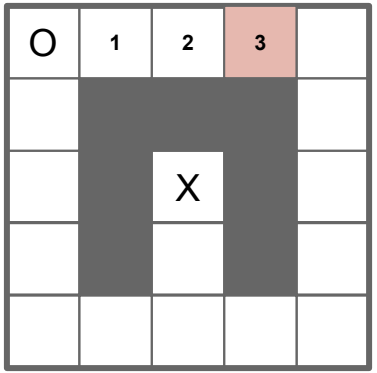
Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_filled(pos) then return ∞
  else return 1 + min of
    steps(moveRight(pos), dest)
    steps(moveDown(pos), dest)
    steps(moveLeft(pos), dest)
    steps(moveUp(pos), dest)
```



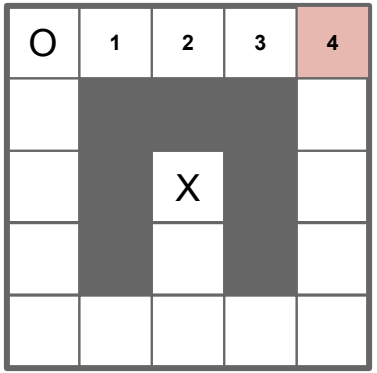
Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_filled(pos) then return ∞
  else return 1 + min of
    steps(moveRight(pos), dest)
    steps(moveDown(pos), dest)
    steps(moveLeft(pos), dest)
    steps(moveUp(pos), dest)
```



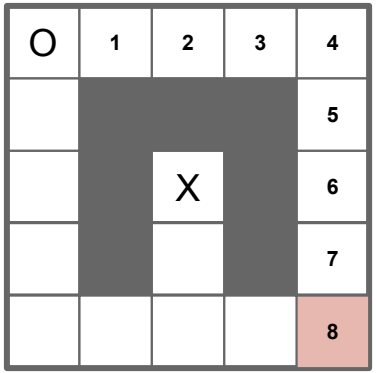
Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_filled(pos) then return ∞
  else return 1 + min of
    steps(moveRight(pos), dest)
    steps(moveDown(pos), dest)
    steps(moveLeft(pos), dest)
    steps(moveUp(pos), dest)
```



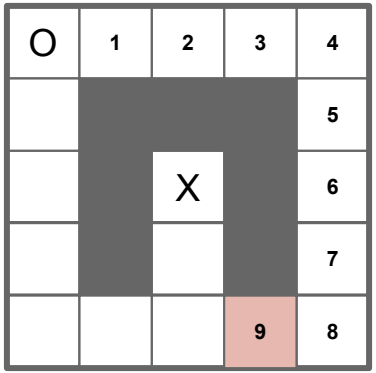
Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_filled(pos) then return ∞
  else return 1 + min of
    steps(moveRight(pos), dest)
    steps(moveDown(pos), dest)
    steps(moveLeft(pos), dest)
    steps(moveUp(pos), dest)
```



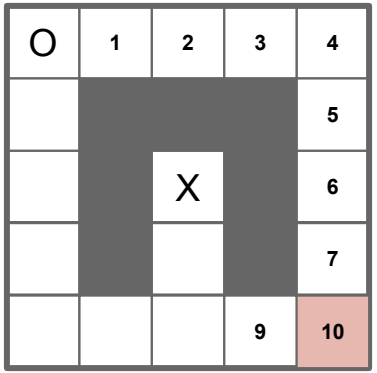
Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_filled(pos) then return ∞
  else return 1 + min of
    steps(moveRight(pos), dest)
    steps(moveDown(pos), dest)
    steps(moveLeft(pos), dest)
    steps(moveUp(pos), dest)
```



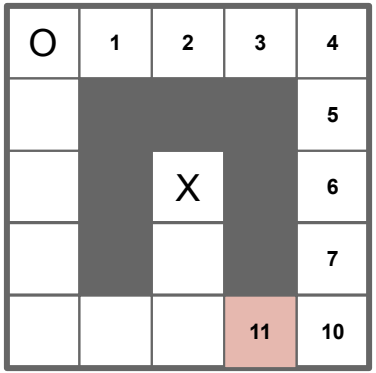
Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_filled(pos) then return ∞
  else return 1 + min of
    steps(moveRight(pos), dest)
    steps(moveDown(pos), dest)
    steps(moveLeft(pos), dest)
    steps(moveUp(pos), dest)
```



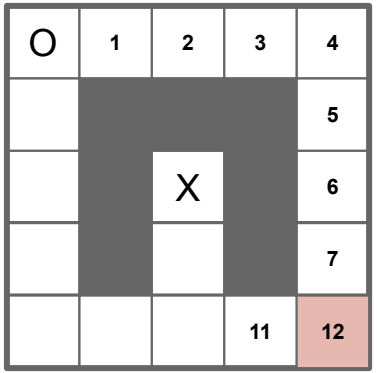
Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_filled(pos) then return ∞
  else return 1 + min of
    steps(moveRight(pos), dest)
    steps(moveDown(pos), dest)
    steps(moveLeft(pos), dest)
    steps(moveUp(pos), dest)
```



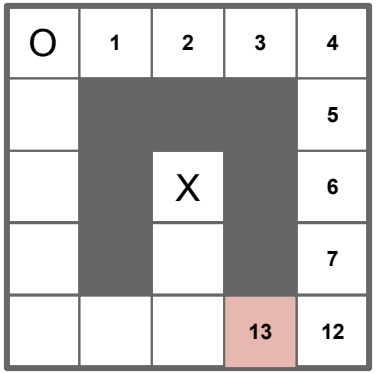
Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_filled(pos) then return ∞
  else return 1 + min of
    steps(moveRight(pos), dest)
    steps(moveDown(pos), dest)
    steps(moveLeft(pos), dest)
    steps(moveUp(pos), dest)
```



Mazes

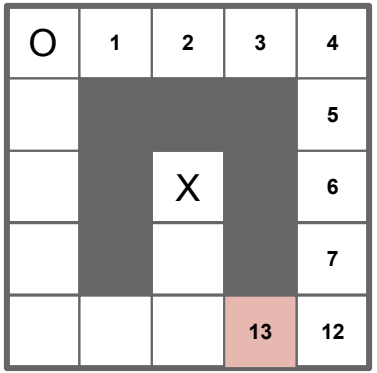
```
steps(pos, dest):
  if pos == dest then return 0
  elif is_filled(pos) then return ∞
  else return 1 + min of
    steps(moveRight(pos), dest)
    steps(moveDown(pos), dest)
    steps(moveLeft(pos), dest)
    steps(moveUp(pos), dest)
```



Problem: Infinite loop!

Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_filled(pos) then return ∞
  else return 1 + min of
    steps(moveRight(pos), dest)
    steps(moveDown(pos), dest)
    steps(moveLeft(pos), dest)
    steps(moveUp(pos), dest)
```



Problem: Infinite loop!

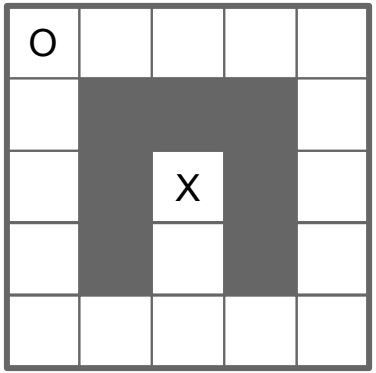
Insight: A path with a loop in it can't be shorter than one without the loop

Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_visited(pos) then return ∞
  elif is_filled(pos) then return ∞
  else
    Mark pos as visited
    return 1 + min of all 4 steps
```

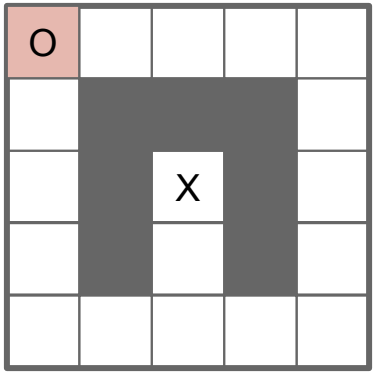
Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_visited(pos) then return ∞
  elif is_filled(pos) then return ∞
  else
    Mark pos as visited
    return 1 + min of all 4 steps
```



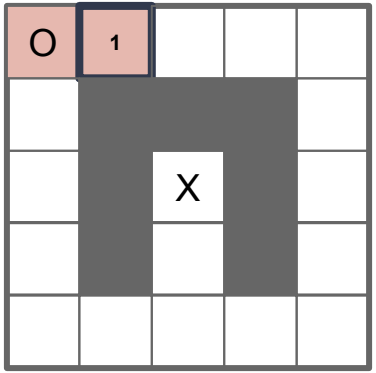
Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_visited(pos) then return ∞
  elif is_filled(pos) then return ∞
  else
    Mark pos as visited
    return 1 + min of all 4 steps
```



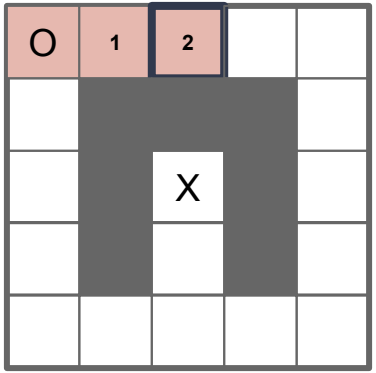
Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_visited(pos) then return ∞
  elif is_filled(pos) then return ∞
  else
    Mark pos as visited
    return 1 + min of all 4 steps
```



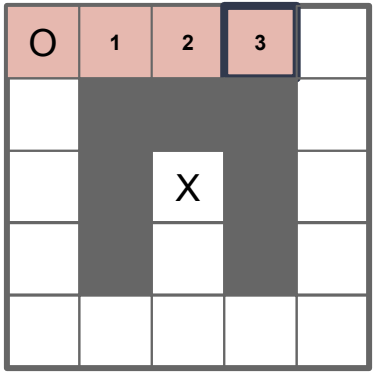
Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_visited(pos) then return ∞
  elif is_filled(pos) then return ∞
  else
    Mark pos as visited
    return 1 + min of all 4 steps
```



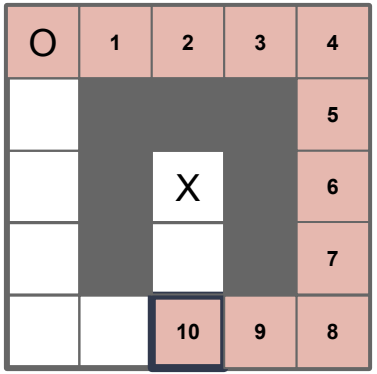
Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_visited(pos) then return ∞
  elif is_filled(pos) then return ∞
  else
    Mark pos as visited
    return 1 + min of all 4 steps
```



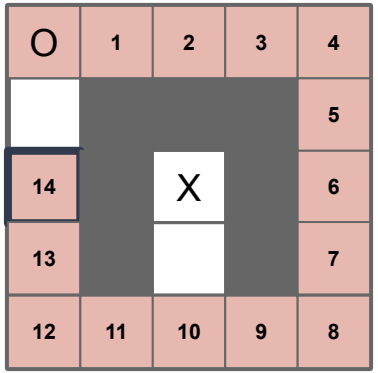
Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_visited(pos) then return ∞
  elif is_filled(pos) then return ∞
  else
    Mark pos as visited
    return 1 + min of all 4 steps
```



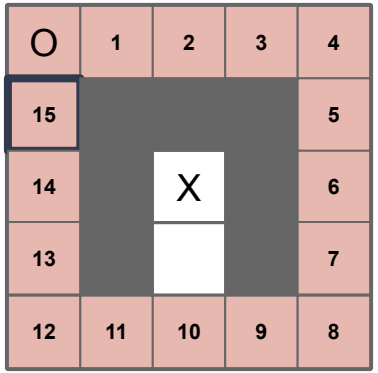
Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_visited(pos) then return ∞
  elif is_filled(pos) then return ∞
  else
    Mark pos as visited
    return 1 + min of all 4 steps
```



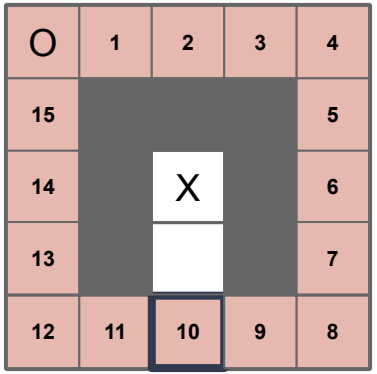
Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_visited(pos) then return ∞
  elif is_filled(pos) then return ∞
  else
    Mark pos as visited
    return 1 + min of all 4 steps
```



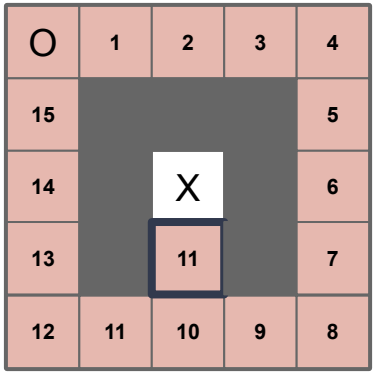
Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_visited(pos) then return ∞
  elif is_filled(pos) then return ∞
  else
    Mark pos as visited
    return 1 + min of all 4 steps
```



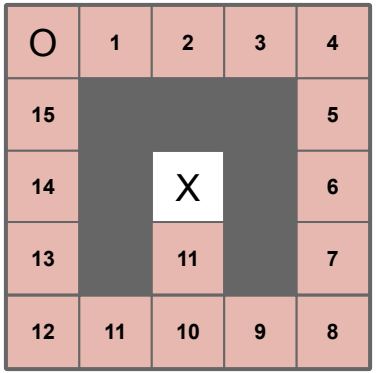
Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_visited(pos) then return ∞
  elif is_filled(pos) then return ∞
  else
    Mark pos as visited
    return 1 + min of all 4 steps
```



Mazes

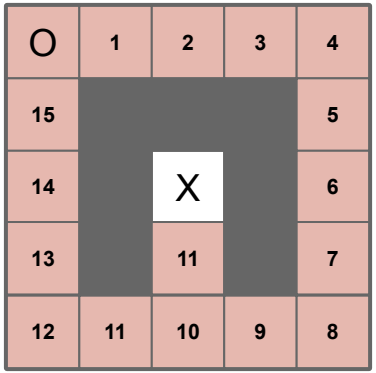
```
steps(pos, dest):
  if pos == dest then return 0
  elif is_visited(pos) then return ∞
  elif is_filled(pos) then return ∞
  else
    Mark pos as visited
    return 1 + min of all 4 steps
```



Problem: The first time you visit a node may be from a longer path!

Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_visited(pos) then return ∞
  elif is_filled(pos) then return ∞
  else
    Mark pos as visited
    return 1 + min of all 4 steps
```



Problem: The first time you visit a node may be from a longer path!

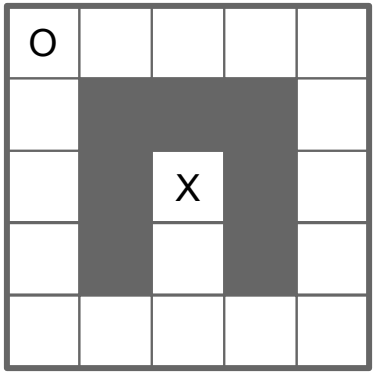
Insight: Unmark nodes as you leave them

Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_visited(pos) then return ∞
  elif is_filled(pos) then return ∞
  else
    Mark pos as visited
    min = 1 + min of all 4 steps
    Mark pos as unvisited
    return min
```

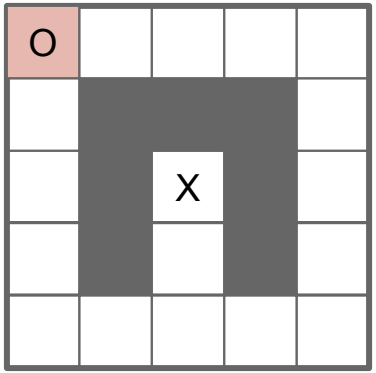
Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_visited(pos) then return ∞
  elif is_filled(pos) then return ∞
  else
    Mark pos as visited
    min = 1 + min of all 4 steps
    Mark pos as unvisited
    return min
```



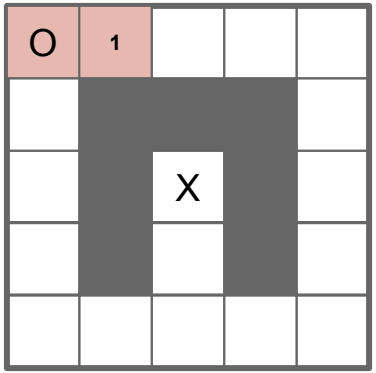
Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_visited(pos) then return ∞
  elif is_filled(pos) then return ∞
  else
    Mark pos as visited
    min = 1 + min of all 4 steps
    Mark pos as unvisited
    return min
```



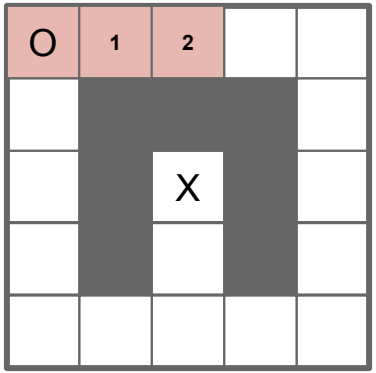
Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_visited(pos) then return ∞
  elif is_filled(pos) then return ∞
  else
    Mark pos as visited
    min = 1 + min of all 4 steps
    Mark pos as unvisited
    return min
```



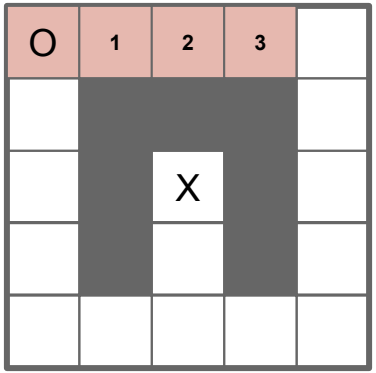
Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_visited(pos) then return ∞
  elif is_filled(pos) then return ∞
  else
    Mark pos as visited
    min = 1 + min of all 4 steps
    Mark pos as unvisited
    return min
```



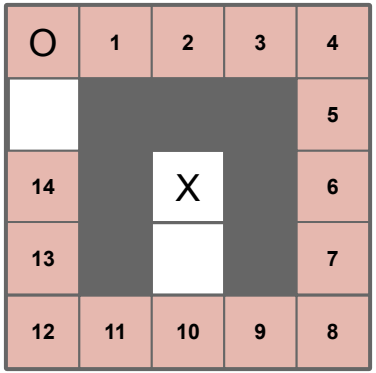
Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_visited(pos) then return ∞
  elif is_filled(pos) then return ∞
  else
    Mark pos as visited
    min = 1 + min of all 4 steps
    Mark pos as unvisited
    return min
```



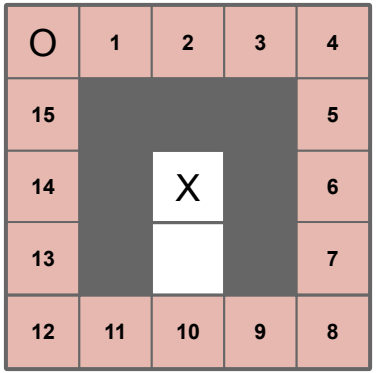
Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_visited(pos) then return ∞
  elif is_filled(pos) then return ∞
  else
    Mark pos as visited
    min = 1 + min of all 4 steps
    Mark pos as unvisited
    return min
```



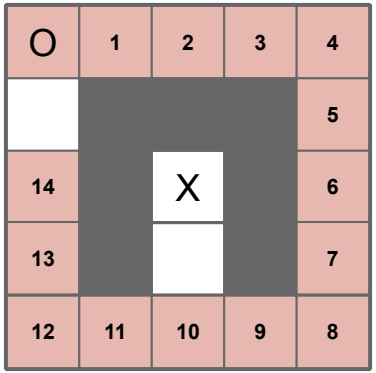
Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_visited(pos) then return ∞
  elif is_filled(pos) then return ∞
  else
    Mark pos as visited
    min = 1 + min of all 4 steps
    Mark pos as unvisited
    return min
```



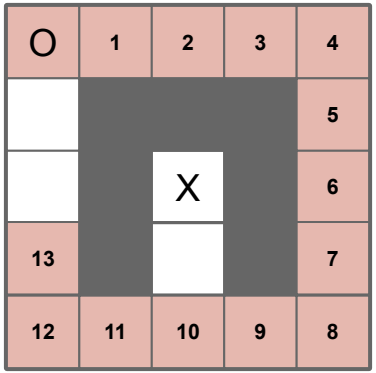
Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_visited(pos) then return ∞
  elif is_filled(pos) then return ∞
  else
    Mark pos as visited
    min = 1 + min of all 4 steps
    Mark pos as unvisited
    return min
```



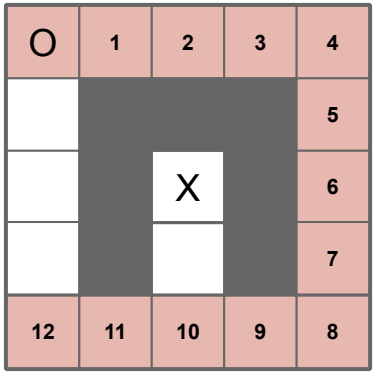
Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_visited(pos) then return ∞
  elif is_filled(pos) then return ∞
  else
    Mark pos as visited
    min = 1 + min of all 4 steps
    Mark pos as unvisited
    return min
```



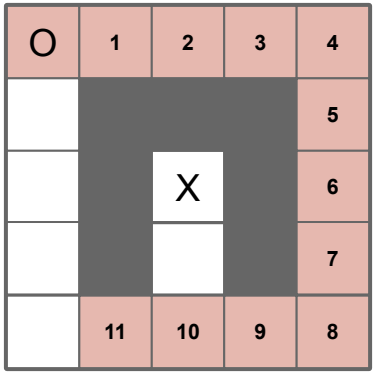
Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_visited(pos) then return ∞
  elif is_filled(pos) then return ∞
  else
    Mark pos as visited
    min = 1 + min of all 4 steps
    Mark pos as unvisited
    return min
```



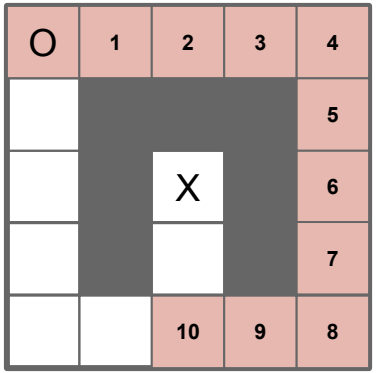
Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_visited(pos) then return ∞
  elif is_filled(pos) then return ∞
  else
    Mark pos as visited
    min = 1 + min of all 4 steps
    Mark pos as unvisited
    return min
```



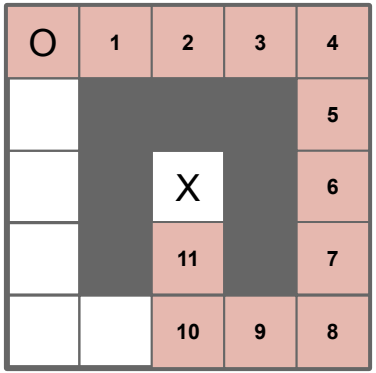
Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_visited(pos) then return ∞
  elif is_filled(pos) then return ∞
  else
    Mark pos as visited
    min = 1 + min of all 4 steps
    Mark pos as unvisited
    return min
```



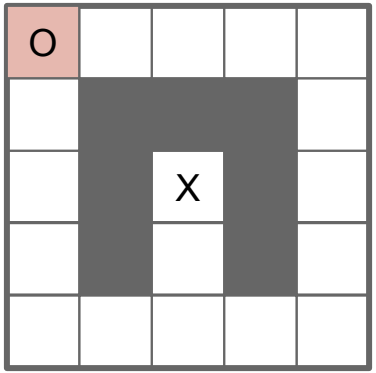
Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_visited(pos) then return ∞
  elif is_filled(pos) then return ∞
  else
    Mark pos as visited
    min = 1 + min of all 4 steps
    Mark pos as unvisited
    return min
```



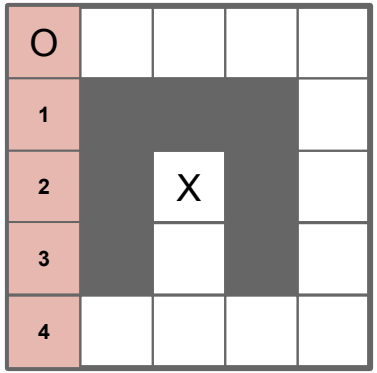
Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_visited(pos) then return ∞
  elif is_filled(pos) then return ∞
  else
    Mark pos as visited
    min = 1 + min of all 4 steps
    Mark pos as unvisited
    return min
```



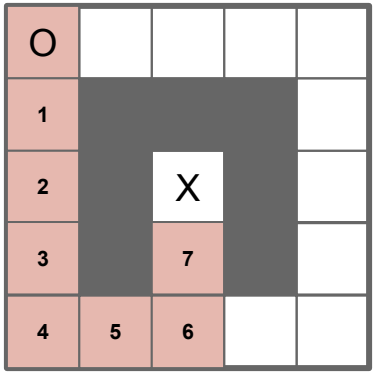
Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_visited(pos) then return ∞
  elif is_filled(pos) then return ∞
  else
    Mark pos as visited
    min = 1 + min of all 4 steps
    Mark pos as unvisited
    return min
```



Mazes

```
steps(pos, dest):
  if pos == dest then return 0
  elif is_visited(pos) then return ∞
  elif is_filled(pos) then return ∞
  else
    Mark pos as visited
    min = 1 + min of all 4 steps
    Mark pos as unvisited
    return min
```



Formalizing Maze-Solving

Inputs:

- The map: an *n* x *m* grid of squares which are either filled or empty
- The **O** is at position *start*
- The **X** is at position *dest*

Goal: Compute `steps(start, dest)`, the minimum number of steps from start to end.

Formalizing Maze-Solving

Inputs:

- The map: an *n* x *m* grid of squares which are either filled or empty
- The **O** is at position *start*
- The **X** is at position *dest*

Goal: Compute `steps(start, dest)`, the minimum number of steps from start to end. ✓

Formalizing Maze-Solving

Inputs:

- The map: an *n* x *m* grid of squares which are either filled or empty
- The **O** is at position *start*
- The **X** is at position *dest*

Goal: Compute `steps(start, dest)`, the minimum number of steps from start to end. ✓

What path did we take?

Mazes

Idea: Keep track of the nodes marked visited…that's our path!

Mazes: Now with…some data structure?

```
steps(pos, dest, visited):
  if pos == dest then return visited.copy()
  elif pos ∈ visited then return no_path
  elif is_filled(pos) then return no_path
  else
    visited.append(pos)
    bestPath = 1 + min of all 4 steps
    visited.removeLast()
    return bestPath
```

Mazes: Now with…some data structure?

```
steps(pos, dest, visited):
  if pos == dest then return visited.copy()
  elif pos ∈ visited then return no_path
  elif is_filled(pos) then return no_path
  else
    visited.append(pos)
    bestPath = 1 + min of all 4 steps
    visited.removeLast()
    return bestPath
```

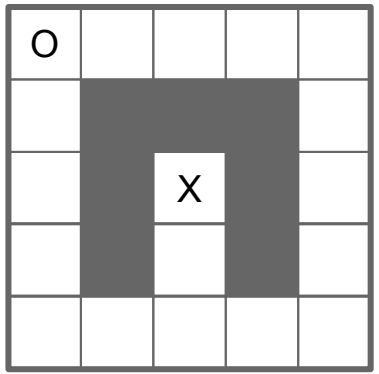
What could this data structure be??

Mazes: Now with…Stacks!

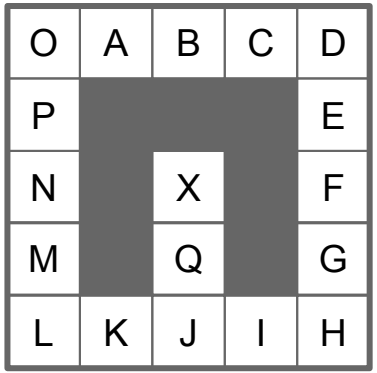
```
steps(pos, dest, visited):
  if pos == dest then return visited.copy()
  elif pos ∈ visited then return no_path
  elif is_filled(pos) then return no_path
  else
    visited.push(pos)
    bestPath = 1 + min of all 4 steps
    visited.pop()
    return bestPath
```

A stack!

Tracing an Example Search



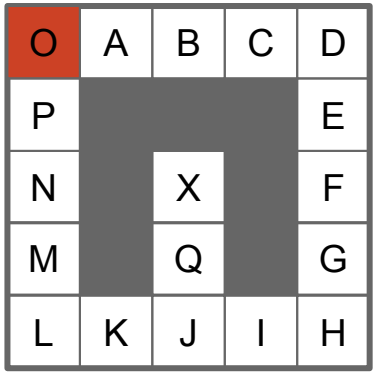
Tracing an Example Search



Call Stack

`visited`

Tracing an Example Search



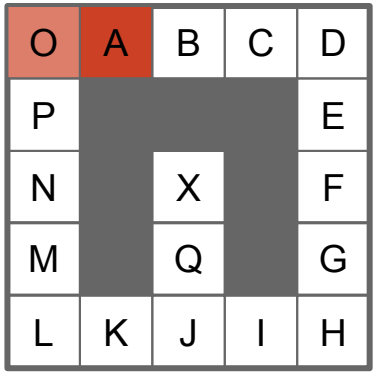

```
steps(O,X):
  steps(moveRight,X)
  steps(moveLeft,X)
  steps(moveUp,X)
  steps(moveDown,X)
```

Call Stack

O

visited

Tracing an Example Search



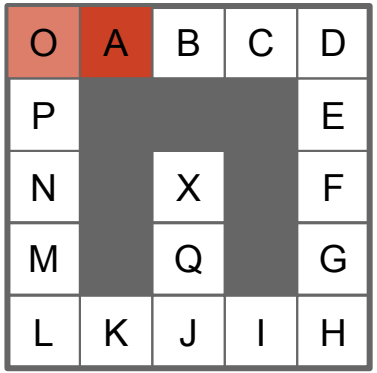

```
steps(A,X):
  steps(moveRight,X)
  steps(moveLeft,X)
  steps(moveUp,X)
  steps(moveDown,X)
steps(O,X):
  steps(moveRight,X)
  steps(moveLeft,X)
  steps(moveUp,X)
  steps(moveDown,X)
```

Call Stack

A
O

visited

Tracing an Example Search




```
steps(A,X):
  steps(moveRight,X)
  steps(moveLeft,X)
  steps(moveUp,X)
  steps(moveDown,X)
```

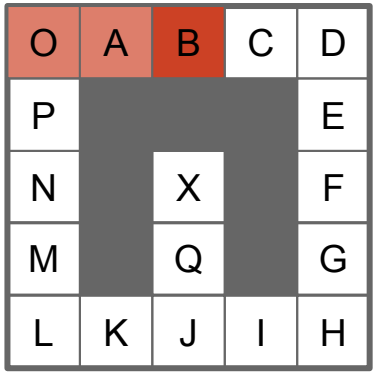
```
steps(O,X)
```

Call Stack

A
O

visited

Tracing an Example Search



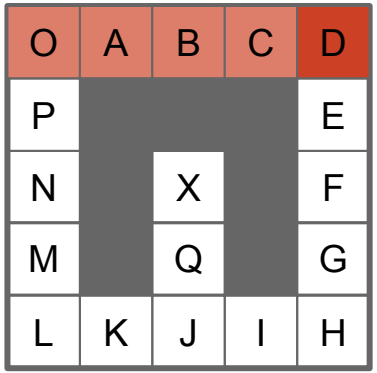

```
steps(B,X):
  steps(moveRight,X)
  steps(moveLeft,X)
  steps(moveUp,X)
  steps(moveDown,X)
steps(A,X):
  steps(moveRight,X)
  steps(moveLeft,X)
  steps(moveUp,X)
  steps(moveDown,X)
steps(O,X)
```

Call Stack

B
A
O

visited

Tracing an Example Search



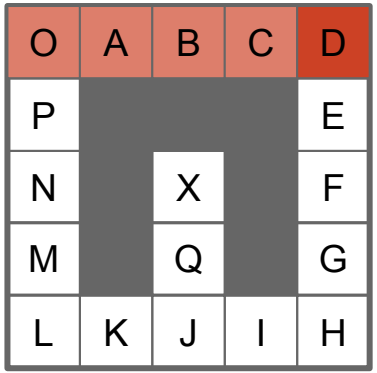

```
steps(D,X):
  steps(moveRight,X)
  steps(moveLeft,X)
  steps(moveUp,X)
  steps(moveDown,X)
steps(C,X)
steps(B,X)
steps(A,X)
steps(O,X)
```

Call Stack

D
C
B
A
O

visited

Tracing an Example Search




```
steps(D,X):
  steps(moveRight,X)
  steps(moveLeft,X)
  steps(moveUp,X)
  steps(moveDown,X)
```

```
steps(C,X)
```

```
steps(B,X)
```

```
steps(A,X)
```

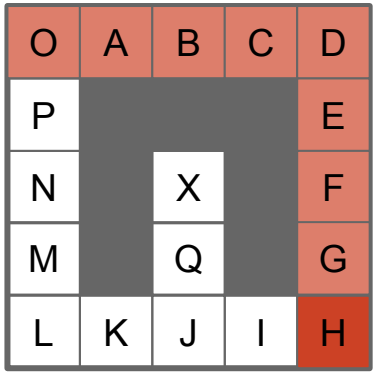
```
steps(O,X)
```

Call Stack

D
C
B
A
O

`visited`

Tracing an Example Search



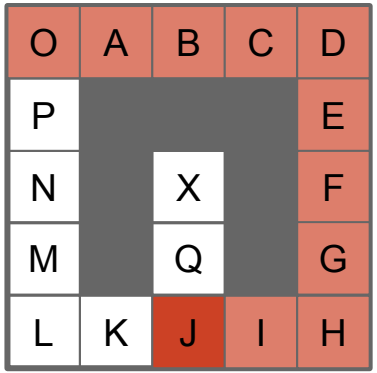
steps(H,X)
steps(G,X)
steps(F,X)
steps(E,X)
steps(D,X)
steps(C,X)
steps(B,X)
steps(A,X)
steps(O,X)

Call Stack

H
G
F
E
D
C
B
A
O

visited

Tracing an Example Search



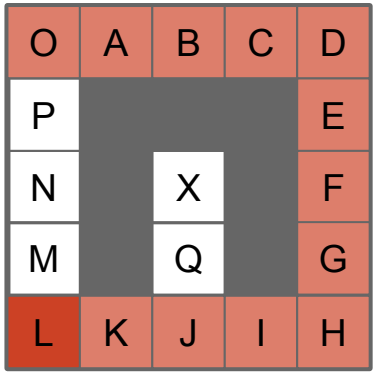

```
steps(J,X):
  steps(moveRight,X)
  steps(moveLeft,X)
  steps(moveUp,X)
  steps(moveDown,X)
steps(I,X)
steps(H,X)
...
steps(O,X)
```

Call Stack

J
I
H
…
O

visited

Tracing an Example Search



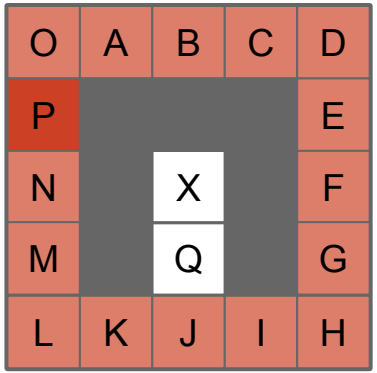
steps(L,X)
steps(K,X)
steps(J,X)
steps(I,X)
steps(H,X)
…
steps(O,X)

Call Stack

L
K
J
I
H
…
O

visited

Tracing an Example Search



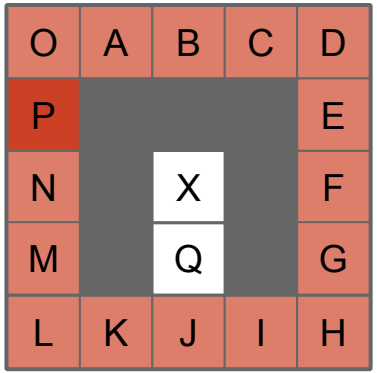

```
steps(P,X):
  steps(moveRight,X)
  steps(moveLeft,X)
  steps(moveUp,X)
  steps(moveDown,X)
steps(N,X)
steps(M,X)
steps(L,X)
...
steps(O,X)
```

Call Stack

P
N
M
L
…
O

visited

Tracing an Example Search




```
steps(P,X):
  steps(moveRight,X)
  steps(moveLeft,X)
  steps(moveUp,X)
  steps(moveDown,X)
steps(N,X)
steps(M,X)
steps(L,X)
...
steps(O,X)
```

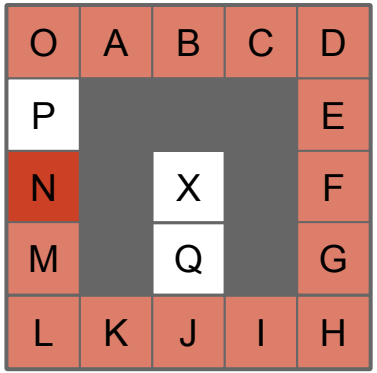
Call Stack

All 4 return no_path, so min is also no_path

P
N
M
L
…
O

visited

Tracing an Example Search



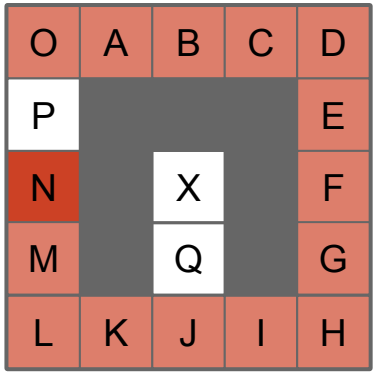

```
steps(N,X):
  steps(moveRight,X)
  steps(moveLeft,X)
  steps(moveUp,X)
  steps(moveDown,X)
steps(M,X)
steps(L,X)
...
steps(O,X)
```

Call Stack

N
M
L
…
O

visited

Tracing an Example Search



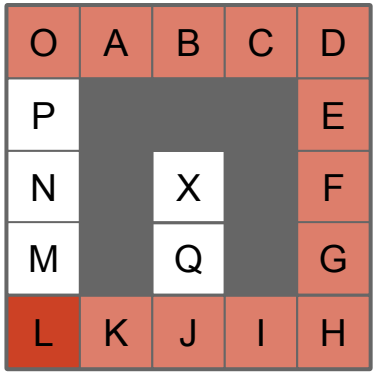

```
steps(N,X):
  steps(moveRight,X)
  steps(moveLeft,X)
  steps(moveUp,X)
  steps(moveDown,X)
steps(M,X)
steps(L,X)
...
steps(O,X)
```

Call Stack

N
M
L
…
O

visited

Tracing an Example Search



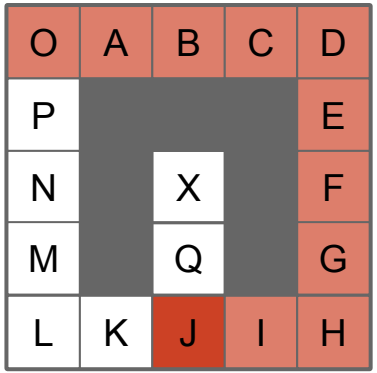
steps(L,X)
…
steps(O,X)

Call Stack

L
…
O

visited

Tracing an Example Search



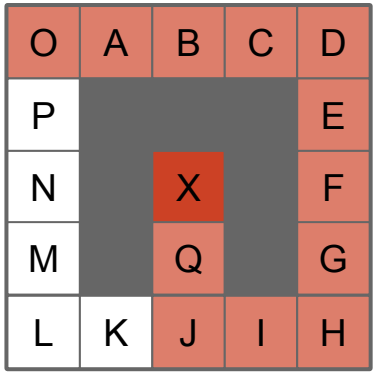

```
steps(J,X):
  steps(moveRight,X)
  steps(moveLeft,X)
  steps(moveUp,X)
  steps(moveDown,X)
steps(I,X)
steps(H,X)
...
steps(O,X)
```

Call Stack

J
I
H
…
O

visited

Tracing an Example Search



steps(X,X) return visited.copy!
steps(Q,X)
steps(J,X)
steps(I,X)
steps(H,X)
…
steps(O,X)

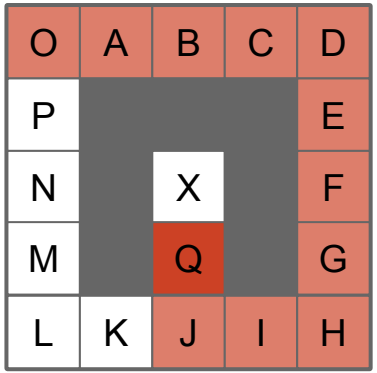
Call Stack

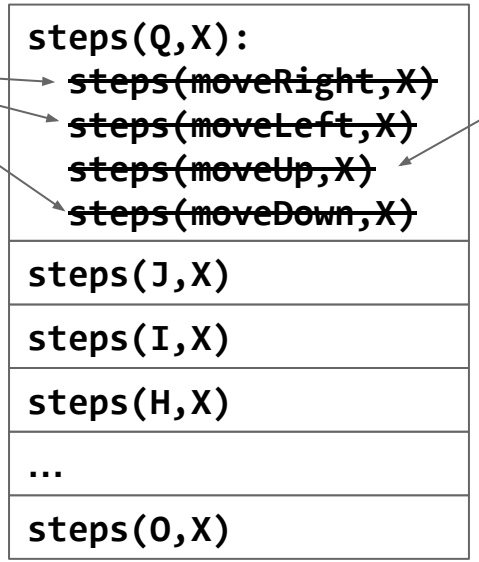
X
Q
J
I
H
…
O

visited

Tracing an Example Search

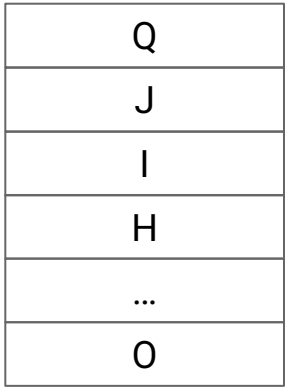
returned no_path





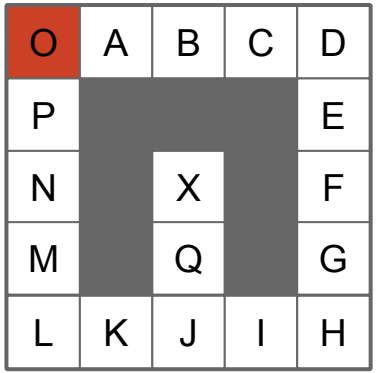
Call Stack

returned OABCDEFGHIJQX



visited

Tracing an Example Search



returned OABCDEFGHIJQX

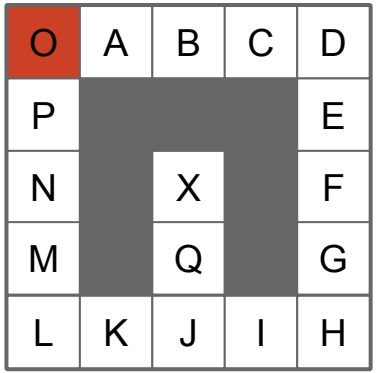
```
steps(O,X):
  steps(moveRight,X)
  steps(moveLeft,X)
  steps(moveUp,X)
  steps(moveDown,X)
```

Call Stack

O

visited

Tracing an Example Search




```
steps(O,X):
  steps(moveRight,X)
  steps(moveLeft,X)
  steps(moveUp,X)
  steps(moveDown,X)
```

returned OABCDEFGHIJQX

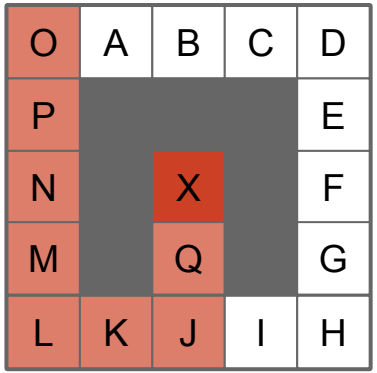
returned no_path

Call Stack

O

visited

Tracing an Example Search



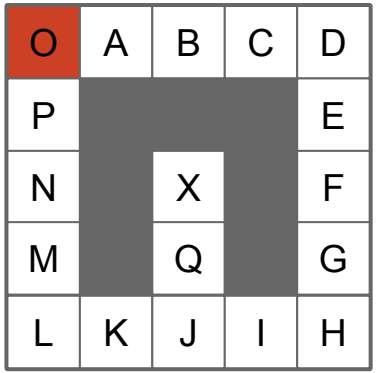

```
steps(X,X)
return visited.copy!
steps(Q,X)
steps(J,X)
steps(K,X)
steps(L,X)
...
steps(O,X)
```

Call Stack

X
Q
J
K
L
…
O

`visited`

Tracing an Example Search




```
steps(O,X):
  steps(moveRight,X)
  steps(moveLeft,X)
  steps(moveUp,X)
  steps(moveDown,X)
```

returned OABCDEFGHIJQX

returned no_path

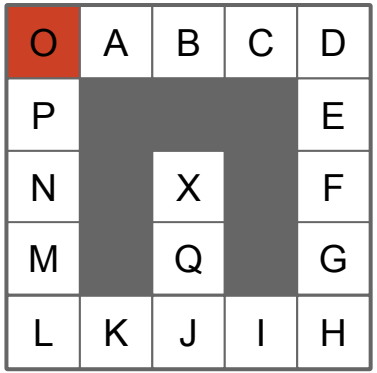
returned OPNMLKJQX

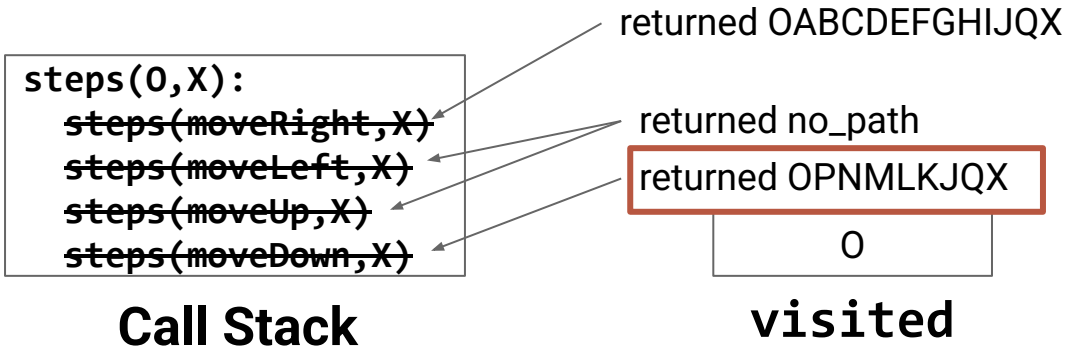
Call Stack

O

visited

Tracing an Example Search





Queues?

Thought Experiment: Can we do something similar with queues?

Queues?

Thought Experiment: Can we do something similar with queues?

Hold that thought!