Graphs, DFS, BFS

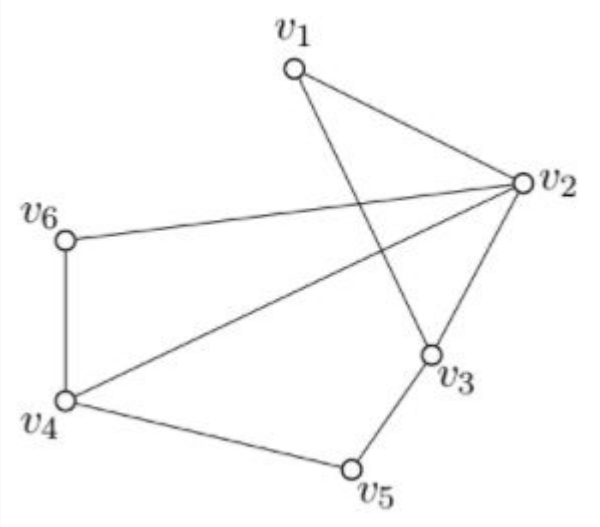
SOI Camp Ftan -- Advanced track

Jakub Závodný

12.2.2018

Graphs

Graphs



Graphs

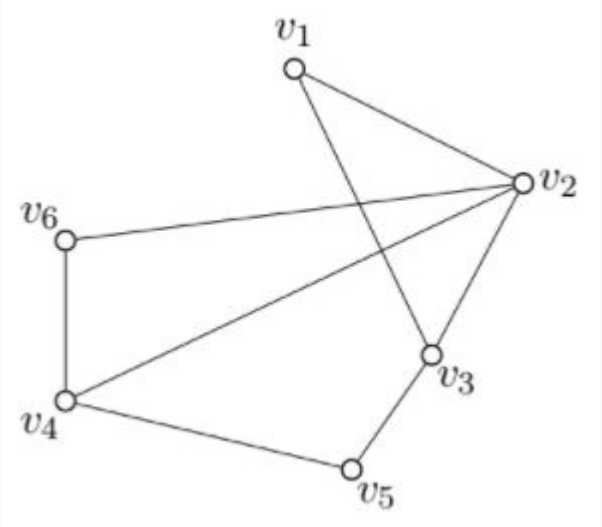
Graph = (V, E)

V = set of vertices

E = set of edges (pairs of vertices)

N = |V| = number of vertices

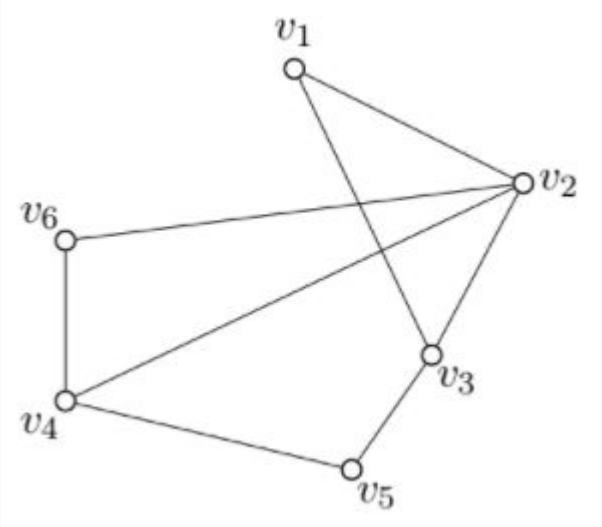
M = |E| = number of edges



Graph Representations

V = {1, 2, 3, 4, 5, 6}

E = {(1, 2), (1, 3), (2, 3), (2, 4), (2, 6),
(3, 5), (4, 5), (4, 6)}

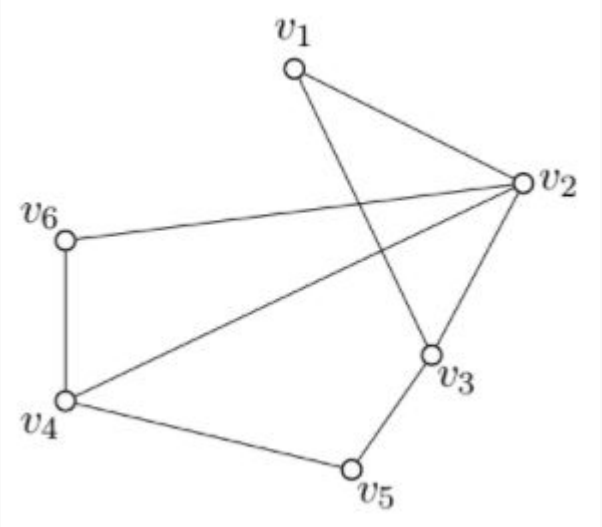


Graph Representations

~~V = {1, 2, 3, 4, 5, 6}~~

`n = 6`

E = {(1, 2), (1, 3), (2, 3), (2, 4), (2, 6),
(3, 5), (4, 5), (4, 6)}

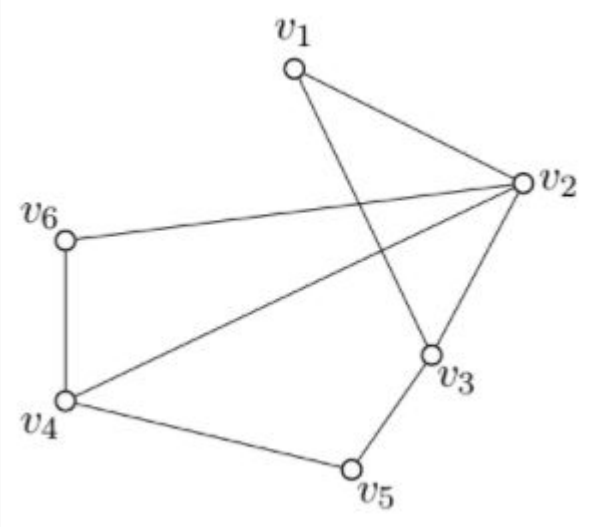


Graph Representations

~~V = {1, 2, 3, 4, 5, 6}~~

```
n = 6
name = {"Zurich", "Basel", "Zug", …
```

E = {(1, 2), (1, 3), (2, 3), (2, 4), (2, 6),
(3, 5), (4, 5), (4, 6)}

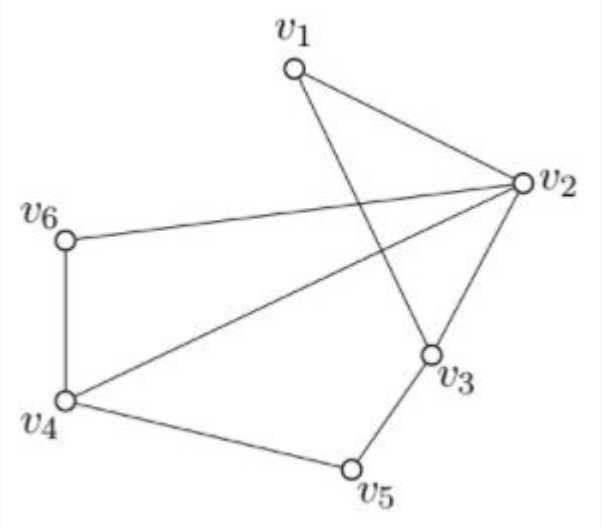


Graph Representations

~~V = {1, 2, 3, 4, 5, 6}~~

`n = 6`

E = {(1, 2), (1, 3), (2, 3), (2, 4), (2, 6),
(3, 5), (4, 5), (4, 6)}



Graph Representations

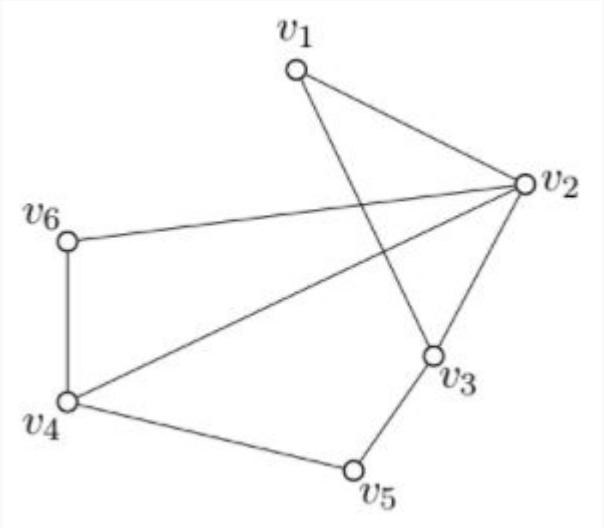
~~V = {1, 2, 3, 4, 5, 6}~~

```
n = 6
```

E = {(1, 2), (1, 3), (2, 3), (2, 4), (2, 6),
(3, 5), (4, 5), (4, 6)}

```
e = vector<pair<int, int>>
```

List of edges



Graph Representations

~~V = {1, 2, 3, 4, 5, 6}~~

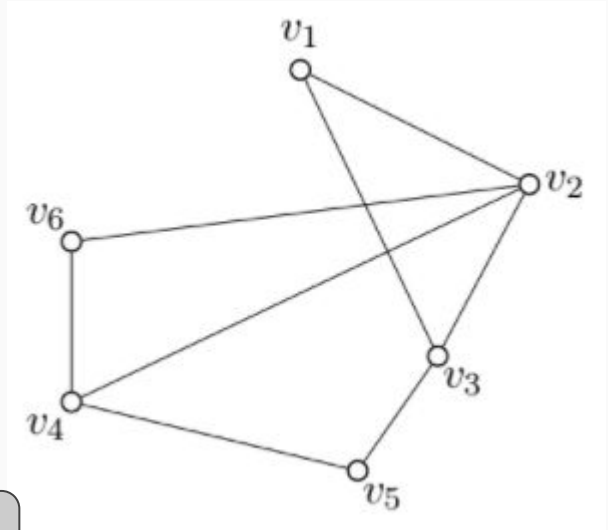
```
n = 6
```

E = {(1, 2), (1, 3), (2, 3), (2, 4), (2, 6),
(3, 5), (4, 5), (4, 6)}

A = {{2,3}, {1,3,4,6}, {1,2,5}, {2,5,6}, {3,4}, {2,6}}

```
a = vector<vector<int>>
```

List of lists of neighbours (“adjacency list”)

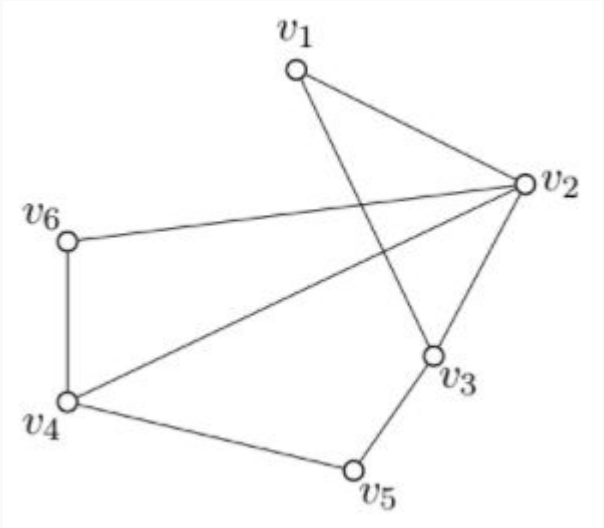


Graph Representations

E = {(1, 2), (1, 3), (2, 3), (2, 4), (2, 6),
(3, 5), (4, 5), (4, 6)}
A = {{2,3}, {1,3,4,6}, {1,2,5}, {2,5,6}, {3,4}, {2,6}}
A' = {{0,1,1,0,0,0},
{1,0,1,1,0,1},
{1,1,0,0,1,0}, ...}

```
a = vector<vector<bool>>
```

Adjacency matrix.



Graph Representations

List of edges:

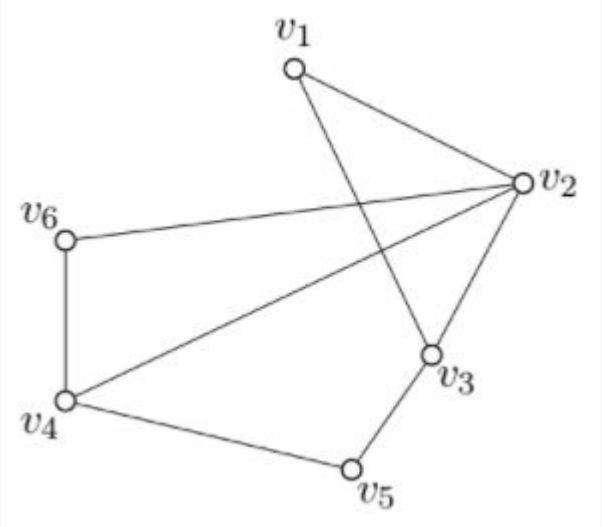
```
e = vector<pair<int, int>>
```

Adjacency lists:

```
a = vector<vector<int>>
```

Adjacency matrix:

```
a = vector<vector<bool>>
```



Graph Representations: Weighted Graphs

List of edges:

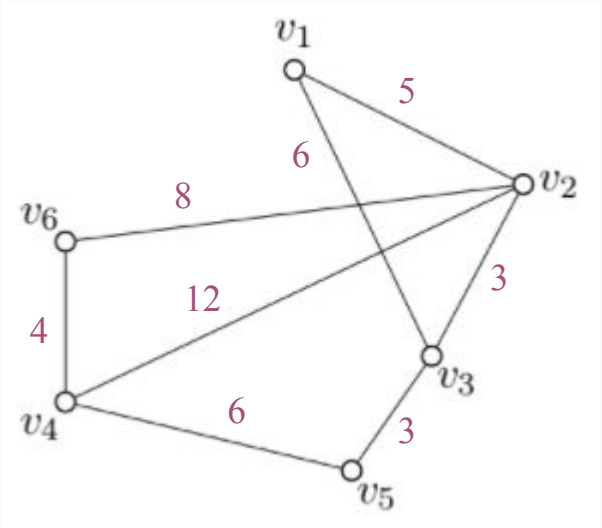
```
e = vector<pair<pair<int, int>, int>>
```

Adjacency lists:

```
a = vector<vector<pair<int, int>>>
```

Adjacency matrix:

```
a = vector<vector<int>>
```



Graph Representations: Weighted Graphs

List of edges:

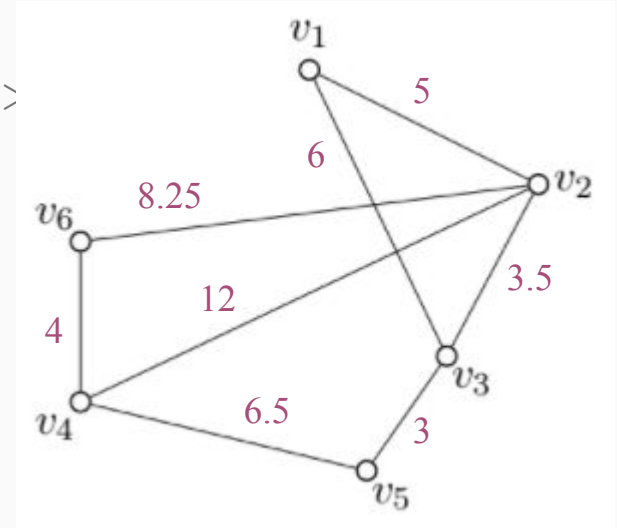
```
e = vector<pair<pair<int, int>, float>>
```

Adjacency lists:

```
a = vector<vector<pair<int, float>>>
```

Adjacency matrix:

```
a = vector<vector<float>>
```



Graph Representations

List of edges:

```
e = vector<pair<int, int>>
```

Size O(|E|)

Adjacency lists:

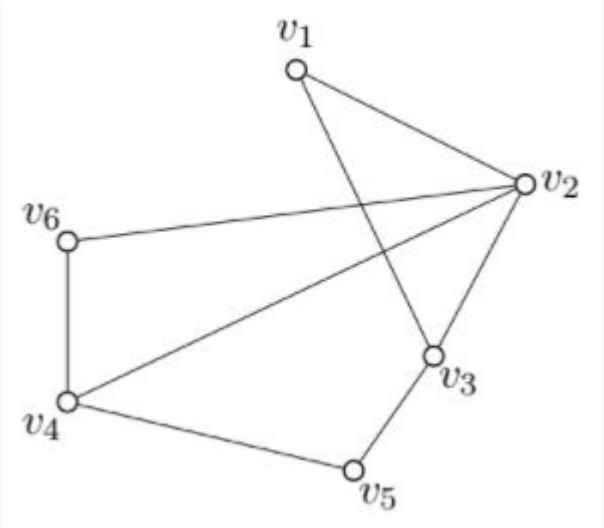
```
a = vector<vector<int>>
```

Size O(|V| + |E|)

Adjacency matrix:

```
a = vector<vector<bool>>
```

Size $O(|V|^2)$



Graph Representations

List of edges:

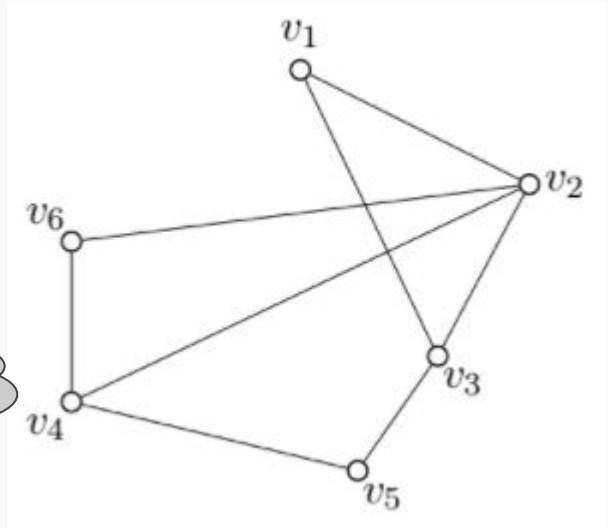
```
e = vector<pair<int, int>>
```

Adjacency lists:

```
a = vector<vector<int>>
```

Adjacency matrix:

```
a = vector<vector<bool>>
```



Graph Representations

List of edges:

```
e = vector<pair<int, int>>
```

Adjacency lists:

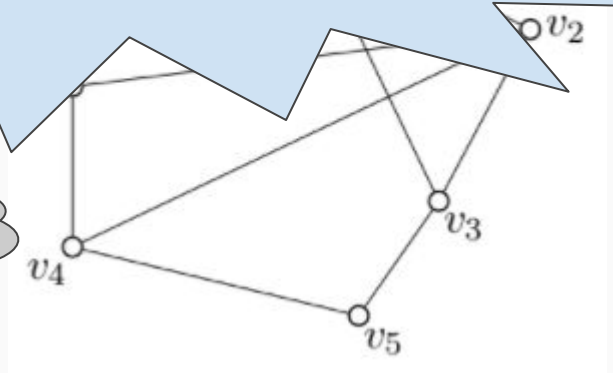
```
a = vector<vector<int>>
```

Adjacency matrix:

```
a = vector<vector<bool>>
```

Depends on the situation.

Which one to use?



Graph Representations

List of edges:

```
e = vector<pair<int, int>
```

Adjacency lists:

```
a = vector<vector<int>>
```

Adjacency matrix:

```
a = vector<vector<bool>>
```

Depends on the situation.

There are
many other variations.

Graph Representations

List of edges:

```
e = vector<pair<int
```

Adjacency lists:

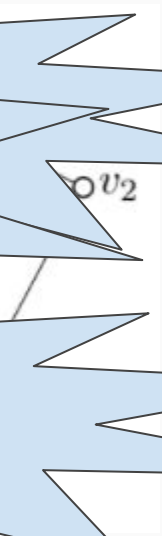
```
a = vector<vector<int
```

Adjacency matrix:

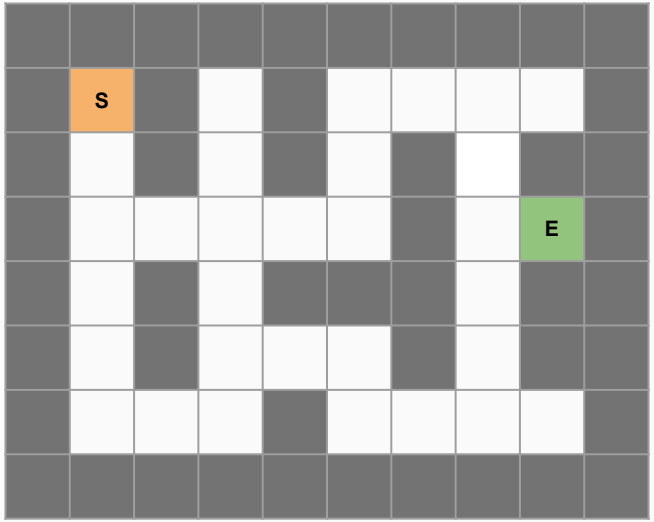
```
a = vector<vector<bool>>
```

Graphs are everywhere!

re are many other variations.



Graph Representations



Graph Representations

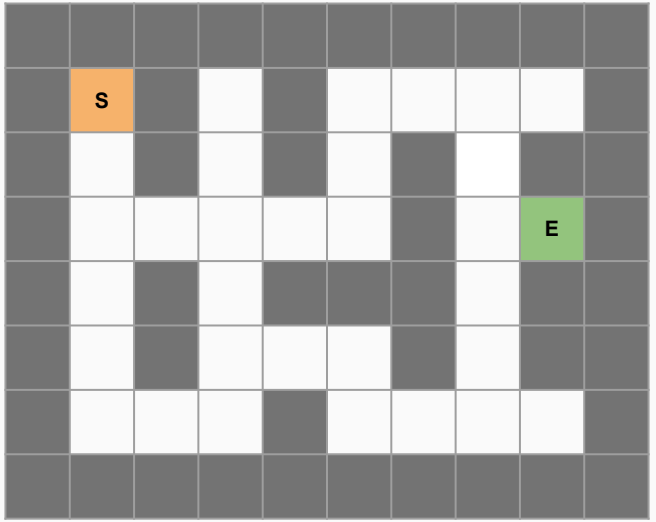
Maze:

Is an implicit graph.

```
m = vector<vector<char>>
```

Vertices are all fields (i,j) with `m[i][j] == '.'` (or `'S'` or `'E'`)

Neighbours of (i,j) are (i,j+1), (i,j-1), (i+1,j), (i-1,j) when they exist.



Graph Representations

Maze:

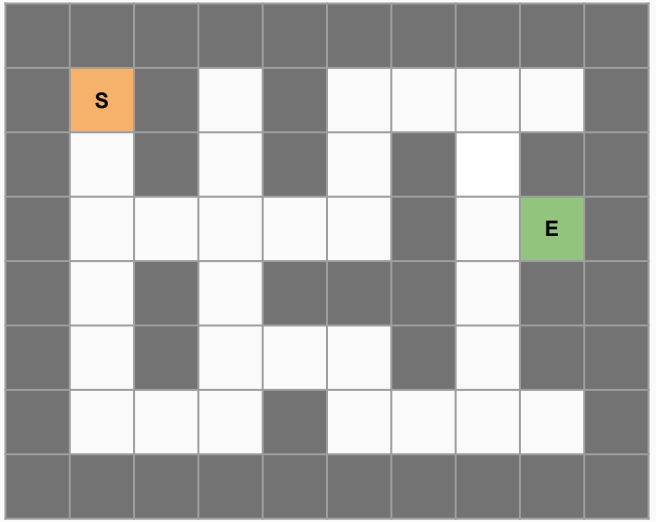
Is an implicit graph.

```
m = vector<vector<char>>
```

Vertices are all fields (i,j) with `m[i][j] == '.'` (or `'S'` or `'E'`)

Neighbours of (i,j) are (i,j+1), (i,j-1), (i+1,j), (i-1,j) when they exist.

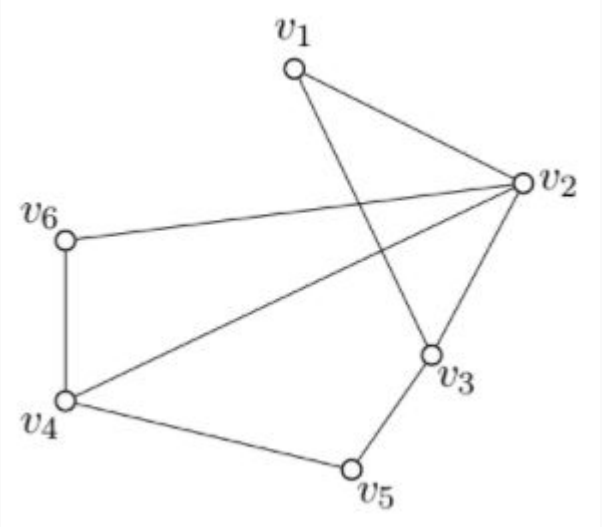
We do not need to “store” the edges!



Searching Graphs

Depth-first Search

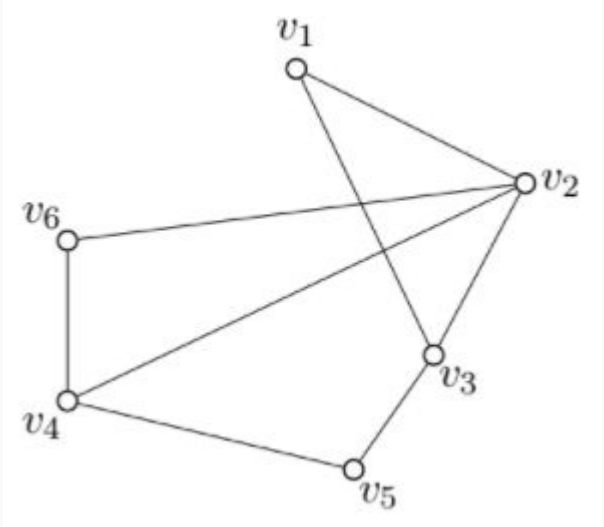
Breadth-first Search



Searching Graphs:

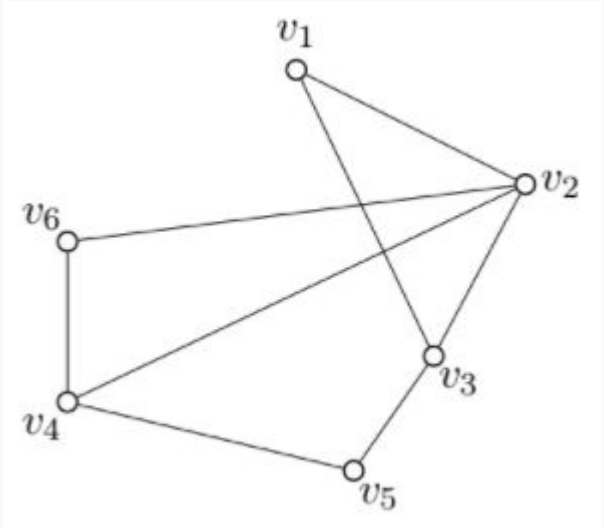
```
mark all vertices not visited
mark start as visited
vertices to look at = {start}

while we still have vertices to look at {
   take one of them
   for all its unvisited neighbors n {
      mark n visited
      put n into vertices to look at
   }
}
```



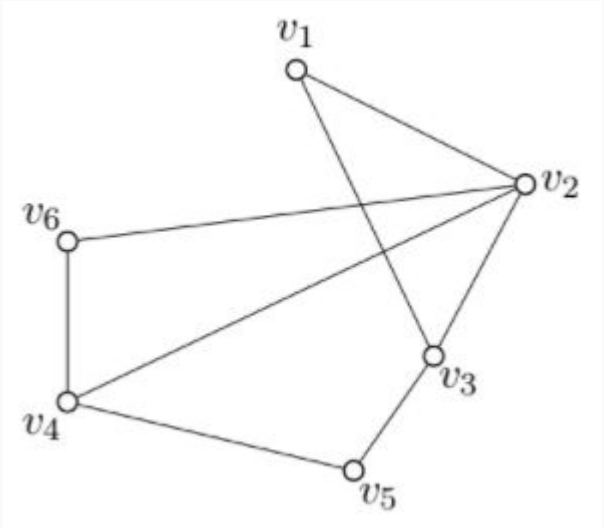
Searching Graphs: DFS

```
vector<int> stack;
visited[start] = true; stack.push_back(start);
while !stack.empty() {
  v = stack.pop()
  for (int n : a[v]) {
    if (!visited[n]) {
      visited[n] = true;
      stack.push_back(n);
    }
  }
}
```



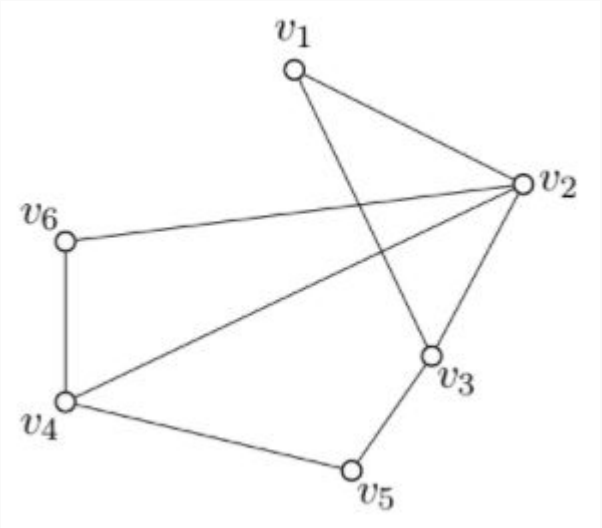
Searching Graphs: BFS

```
queue<int> q;
visited[start] = true; q.push_back(start);
while !q.empty() {
  v = q.pop_front()
  for (int n : a[v]) {
    if (!visited[n]) {
      visited[n] = true;
      q.push_back(n);
    }
  }
}
```



Searching Graphs: Dijkstra's Algorithm

```
priority_queue<float, int> q;
distance[start] = 0; q.push_back((0, start));
while !q.empty() {
  (d, v) = q.top()
  for (int n : a[v]) {
    if (there is a shortcut to n via v) {
      update distance[n]
      q.push_back(distance[n], n);
    }
  }
}
```

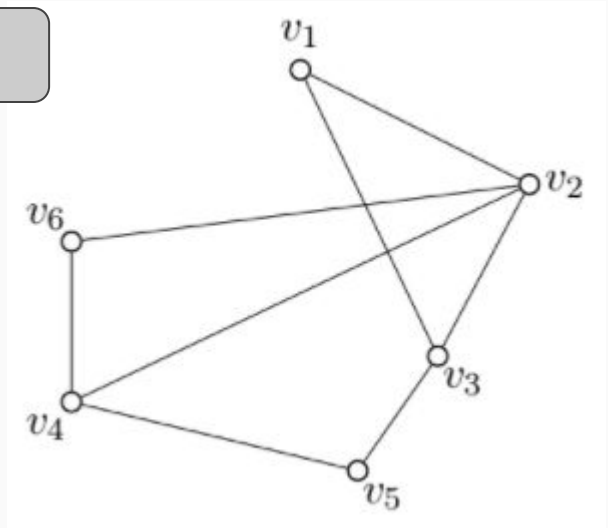


Searching Graphs:

```
mark all vertices not visited
mark start as visited
vertices to look at = {start}

while we still have vertices to look at {
   take one of them
   for all its unvisited neighbors n {
      mark n visited
      put n into vertices to look at
   }
}
```

```
vector<bool> visited;
```

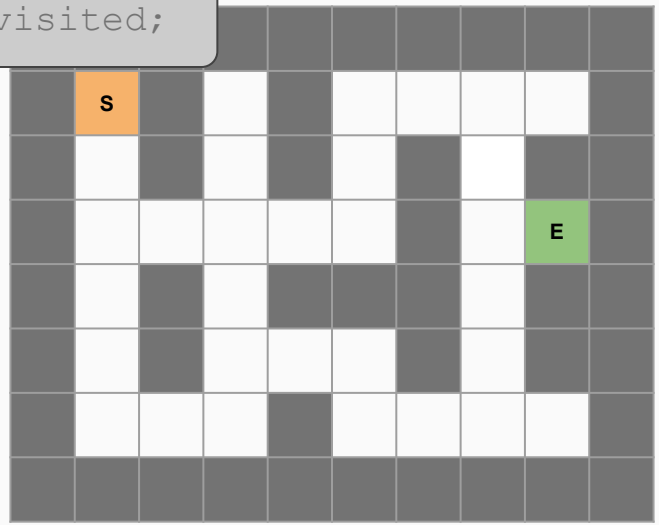


Searching Graphs:

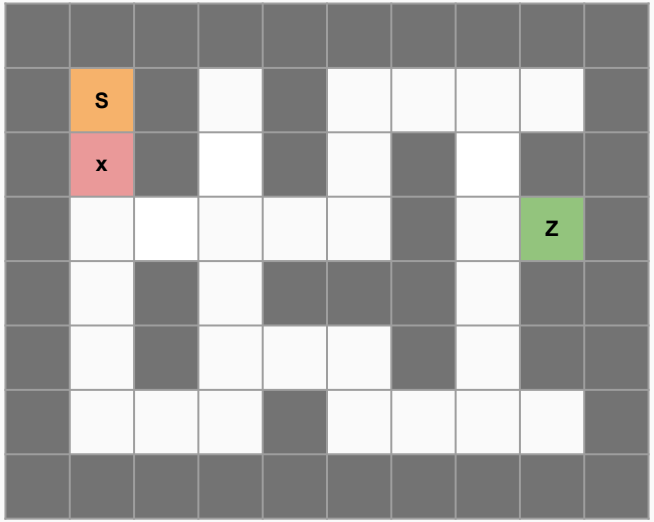
```
mark all vertices not visited
mark start as visited
vertices to look at = {start}

while we still have vertices to look at {
   take one of them
   for all its unvisited neighbors n {
      mark n visited
      put n into vertices to look at
   }
}
```

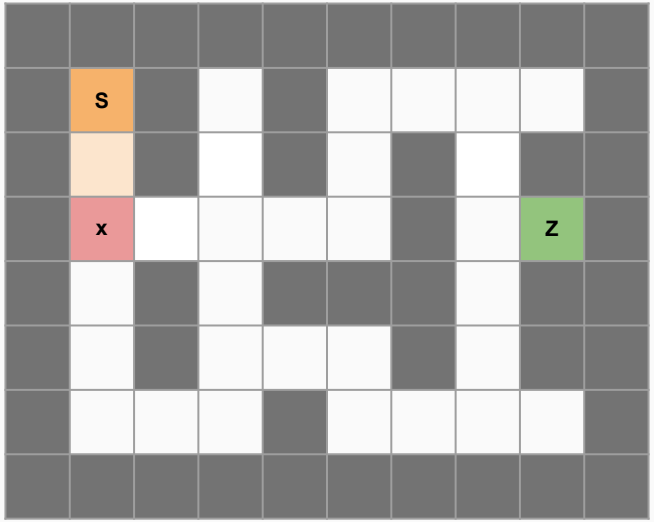
```
vector<vector<bool>> visited;
```



DFS in a Maze

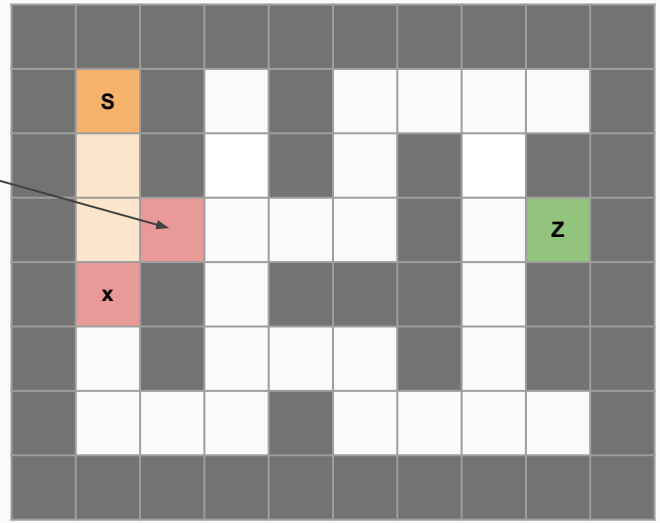


DFS in a Maze



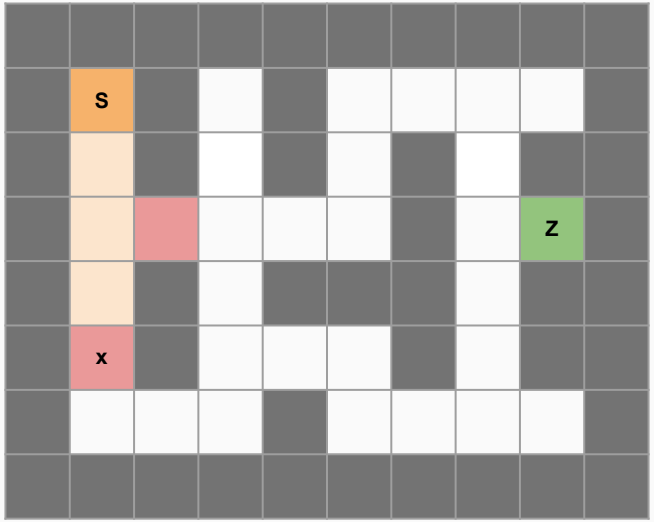
DFS in a Maze

```
stack = [
   (3,2)
]
```



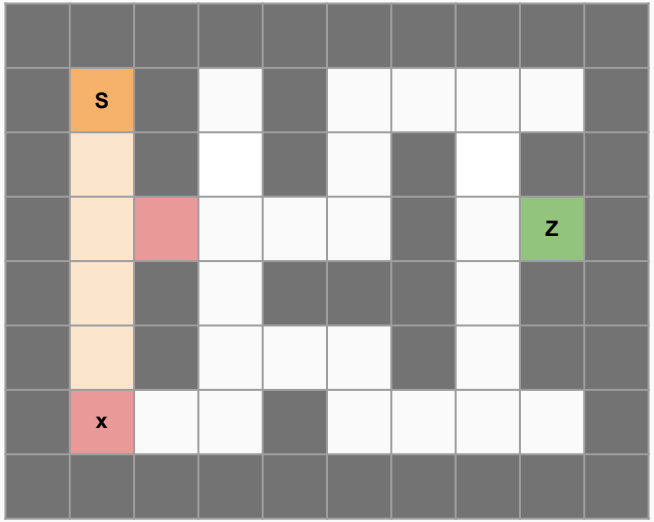
DFS in a Maze

```
stack = [
  (3,2)
]
```



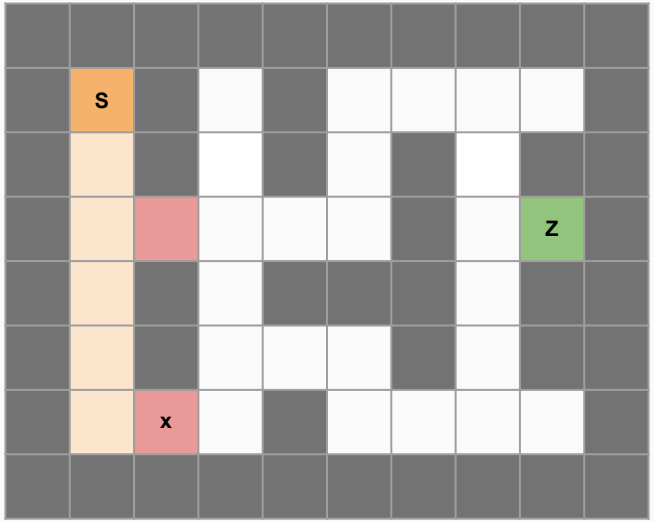
DFS in a Maze

```
stack = [
  (3,2)
]
```



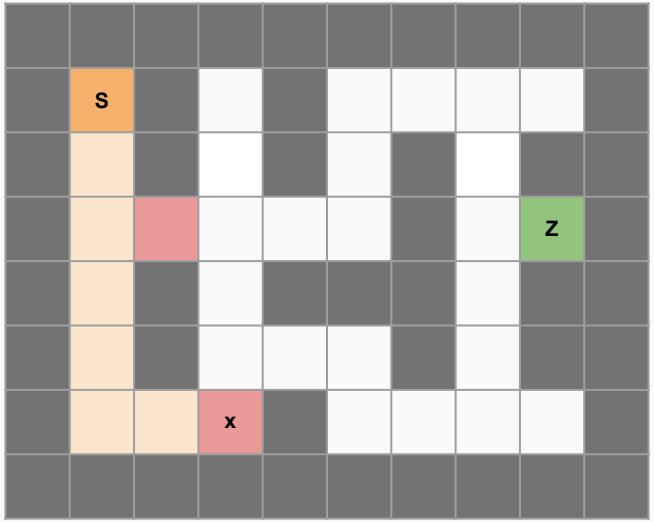
DFS in a Maze

```
stack = [
  (3,2)
]
```



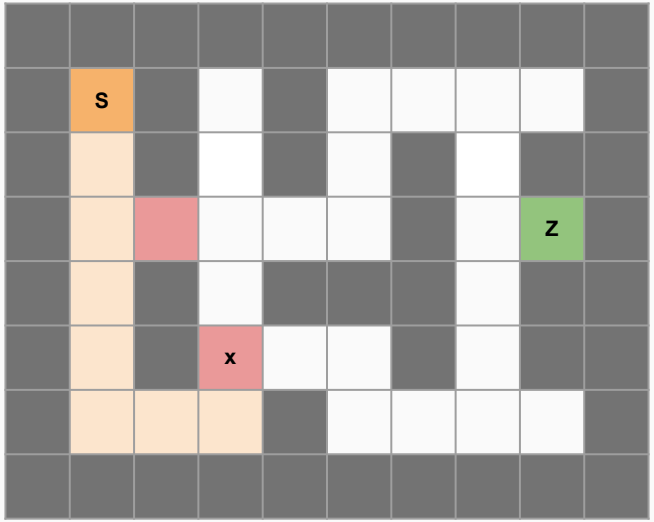
DFS in a Maze

```
stack = [
  (3,2)
]
```



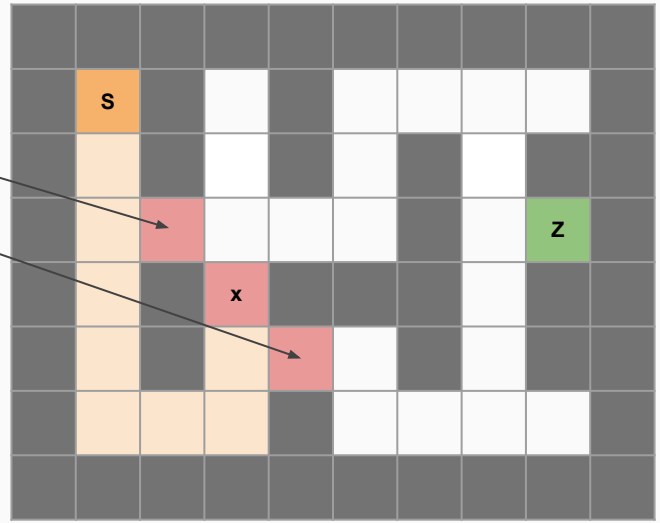
DFS in a Maze

```
stack = [
  (3,2)
]
```



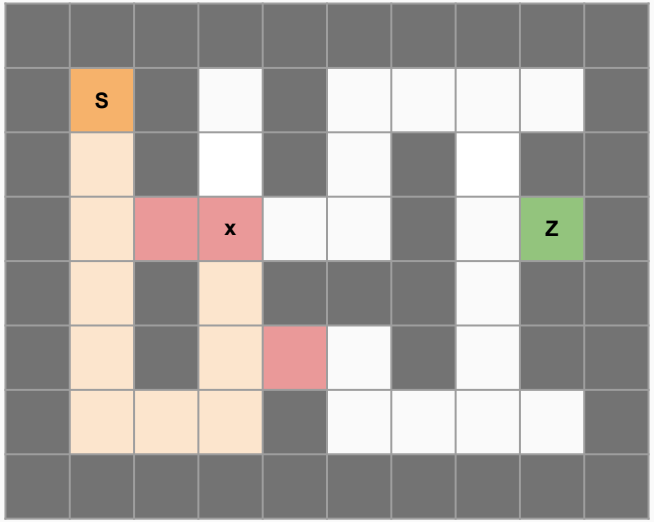
DFS in a Maze

```
stack = [
   (3,2),
   (5,4)
]
```



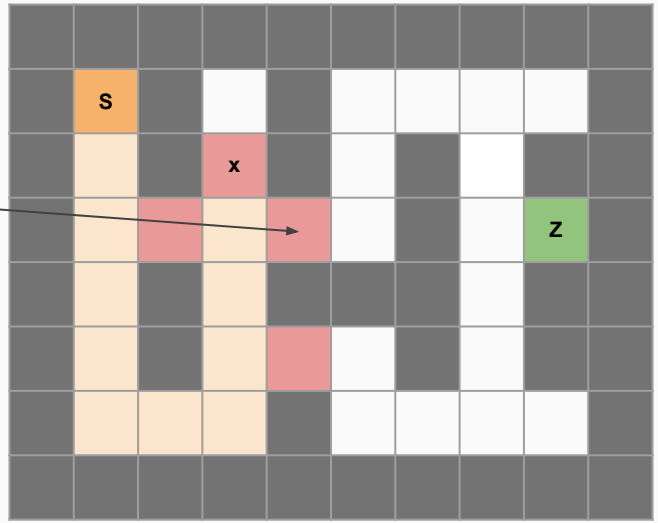
DFS in a Maze

```
stack = [
  (3,2),
  (5,4)
]
```



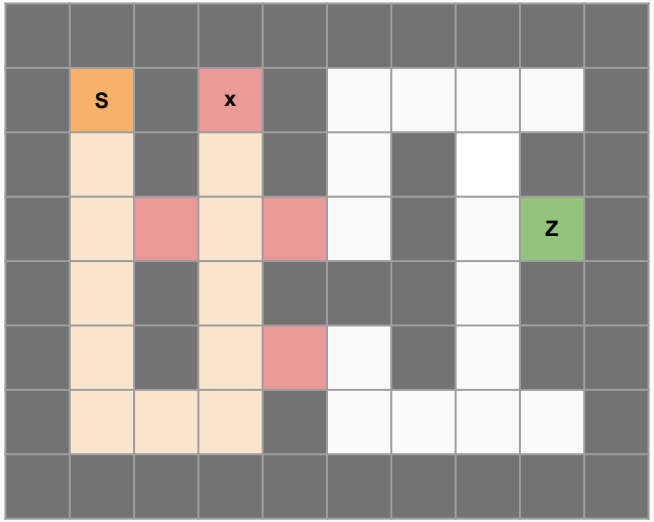
DFS in a Maze

```
stack = [
   (3,2),
   (5,4),
   (3,4)
]
```



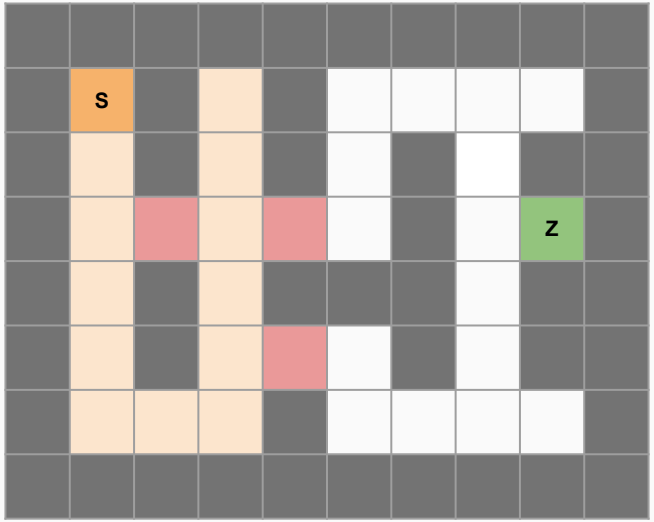
DFS in a Maze

```
stack = [
   (3,2),
   (5,4),
   (3,4)
]
```



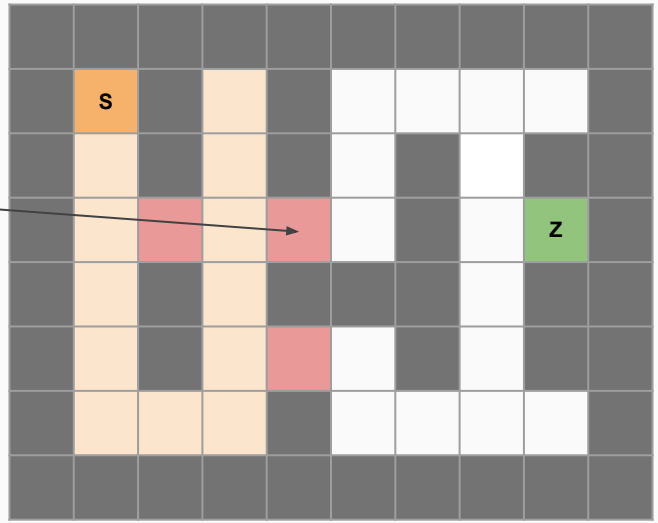
DFS in a Maze

```
stack = [
   (3,2),
   (5,4),
   (3,4)
]
```



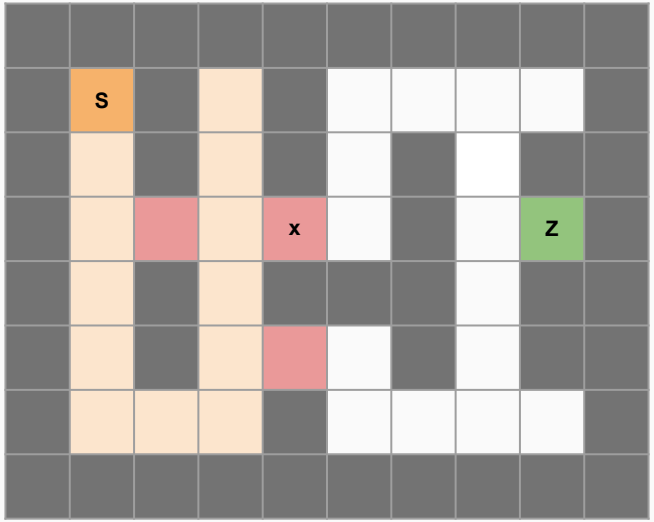
DFS in a Maze

```
stack = [
   (3,2),
   (5,4),
   (3,4)
]
```



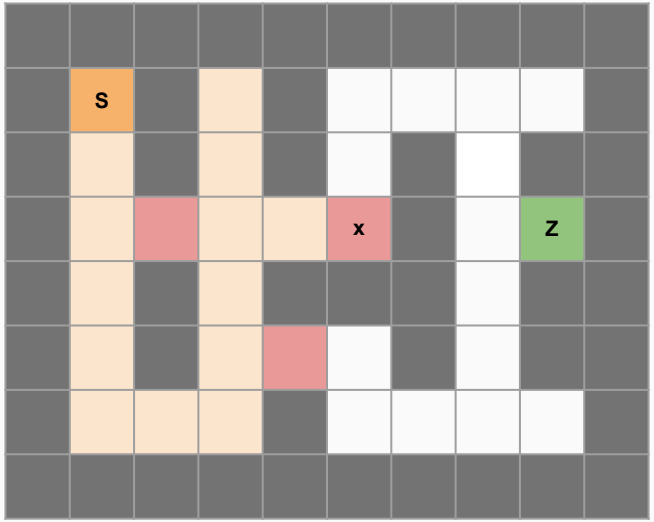
DFS in a Maze

```
stack = [
   (3,2),
   (5,4)
]
```



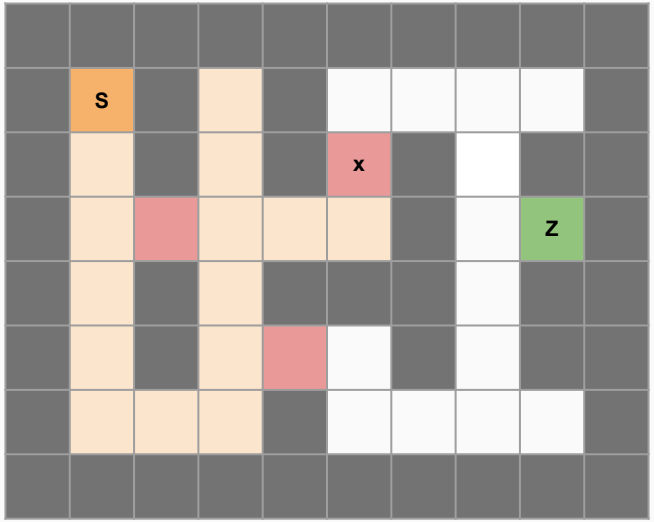
DFS in a Maze

```
stack = [
  (3,2),
  (5,4)
]
```



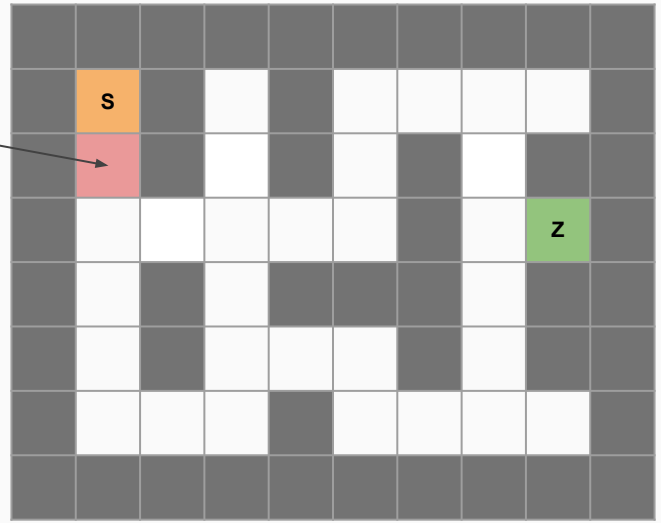
DFS in a Maze

```
stack = [
  (3,2),
  (5,4)
]
```



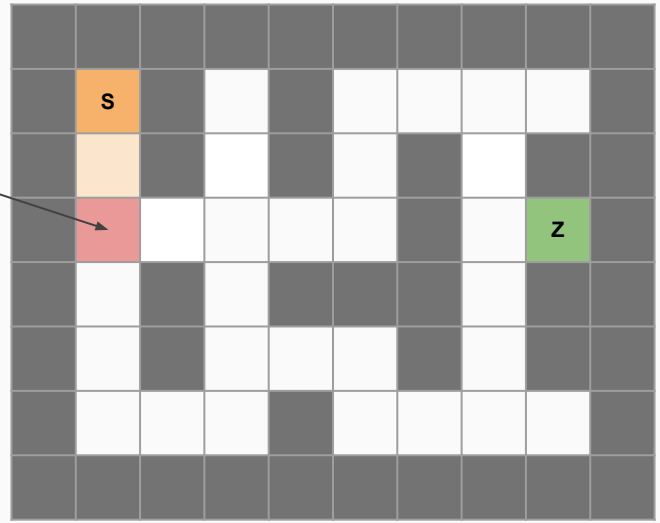
BFS in a Maze

```
queue = [
   (2,1)
]
```



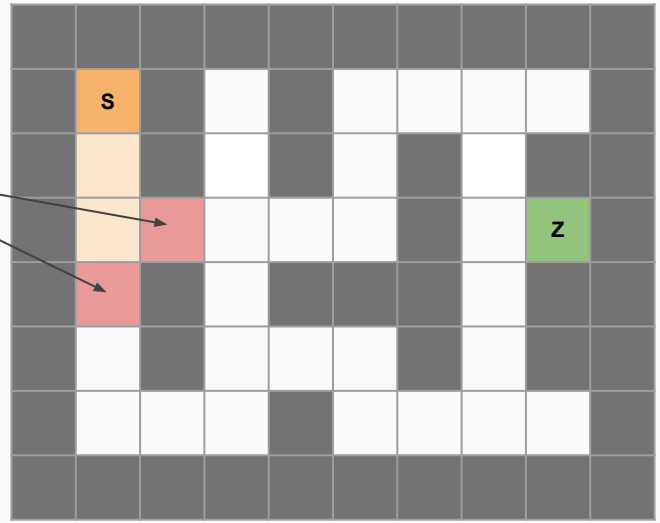
BFS in a Maze

```
queue = [
   (3,1)
]
```



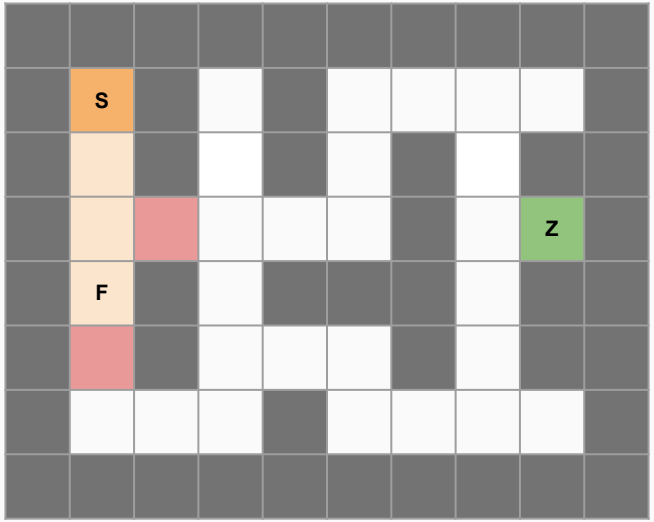
BFS in a Maze

```
queue = [
   (4,1),
   (3,2)
]
```



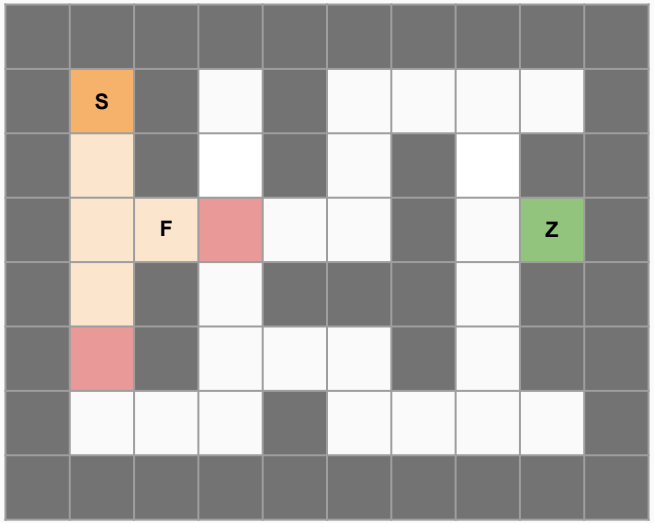
BFS in a Maze

```
queue = [
  (4,1),
  (3,2),
  (5,1)
]
```



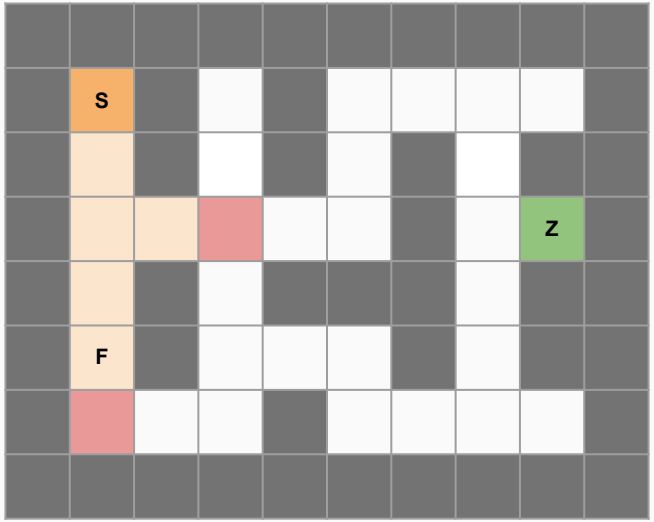
BFS in a Maze

```
queue = [
  (3,2),
  (5,1),
  (3,3)
]
```



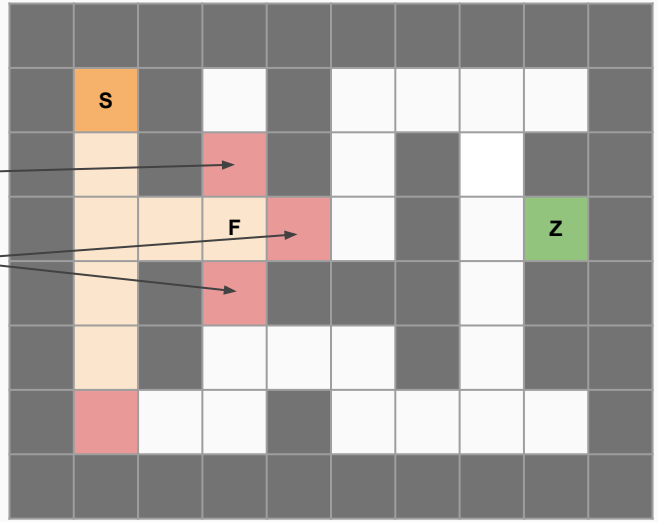
BFS in a Maze

```
queue = [
  (5,1),
  (3,3),
  (6,1)
]
```



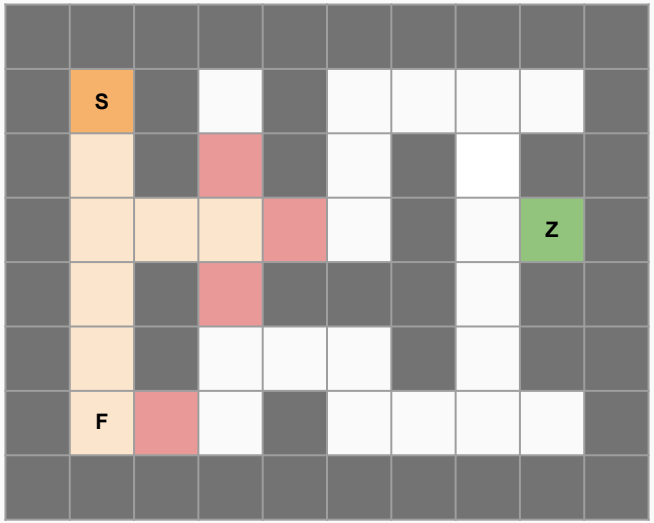
BFS in a Maze

```
queue = [
  (3,3),
   (6,1),
   (2,3),
   (4,3),
   (3,4),
]
```



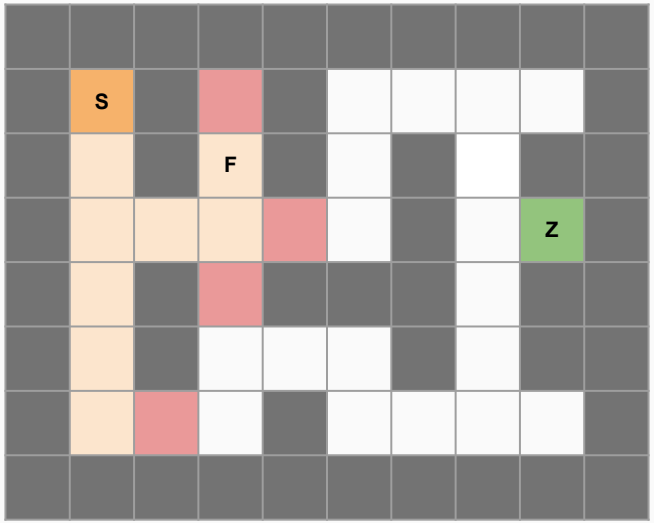
BFS in a Maze

```
queue = [
  (6,1),
  (2,3),
  (4,3),
  (3,4),
  (6,2),
]
```



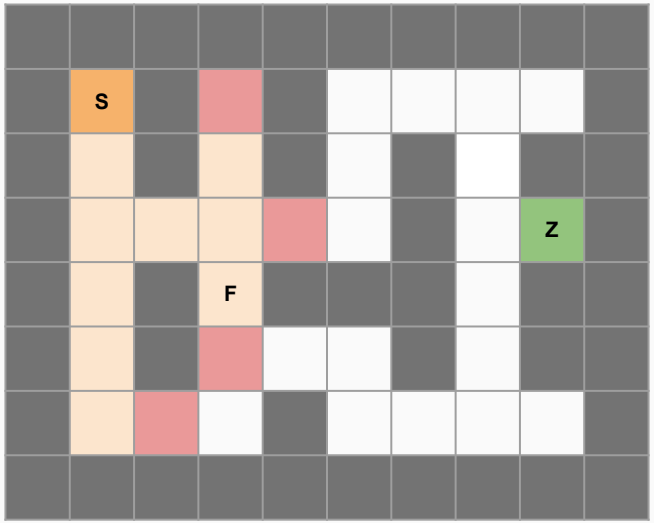
BFS in a Maze

```
queue = [
  (2,3),
   (4,3),
   (3,4),
   (6,2),
   (1,3)
]
```



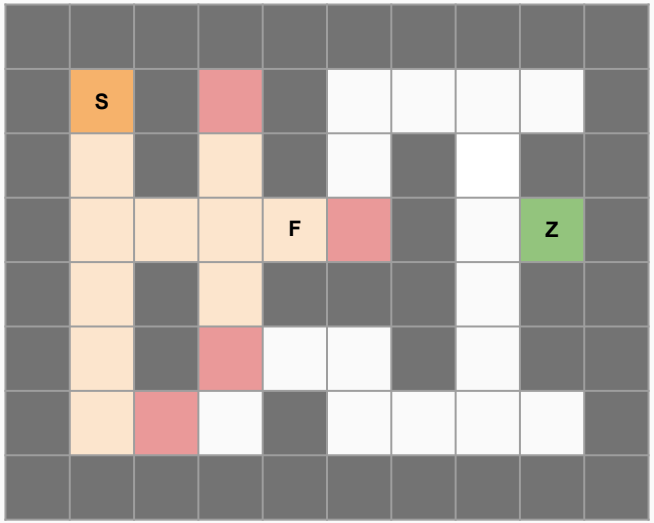
BFS in a Maze

```
queue = [
  (4,3),
   (3,4),
   (6,2),
   (1,3),
   (5,3)
]
```



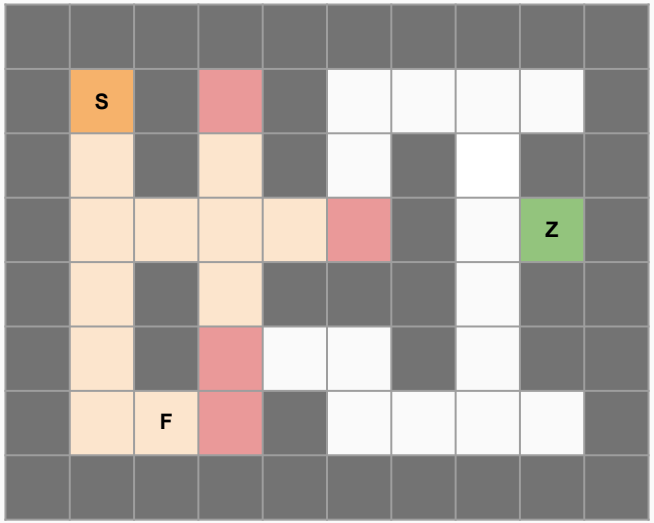
BFS in a Maze

```
queue = [
  (3,4),
   (6,2),
   (1,3),
   (5,3),
   (3,5)
]
```



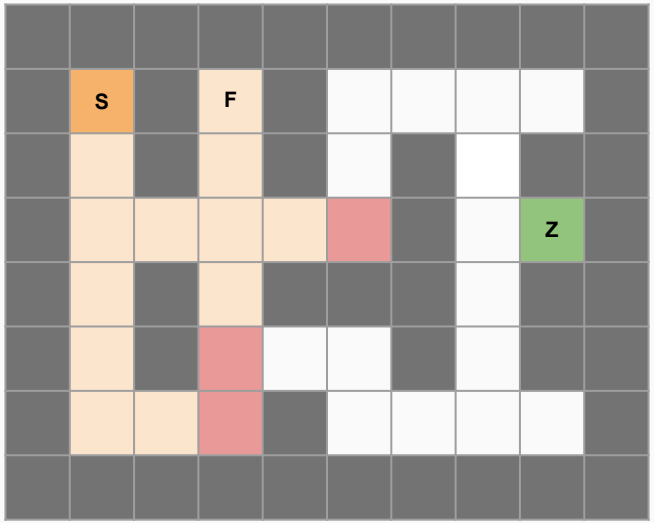
BFS in a Maze

```
queue = [
  (6,2),
  (1,3),
  (5,3),
  (3,5),
  (6,3)
]
```



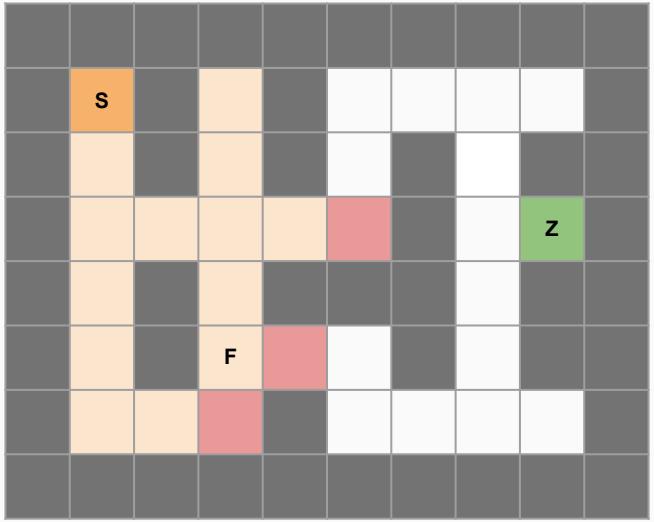
BFS in a Maze

```
queue = [
  (1,3),
  (5,3),
  (3,5),
  (6,3)
]
```



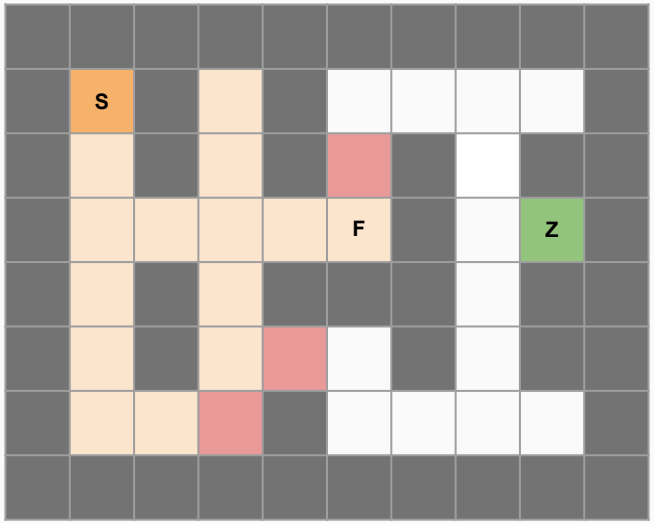
BFS in a Maze

```
queue = [
  (5,3),
   (3,5),
   (6,3),
   (5,4)
]
```



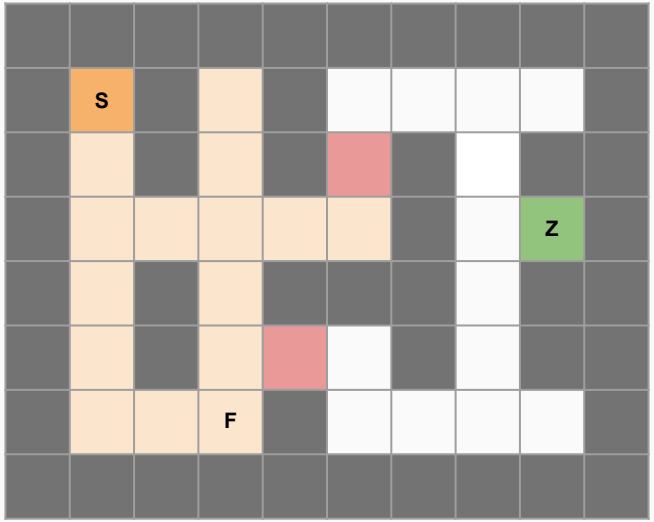
BFS in a Maze

```
queue = [
  (3,5),
  (6,3),
  (5,4),
  (2,5)
]
```



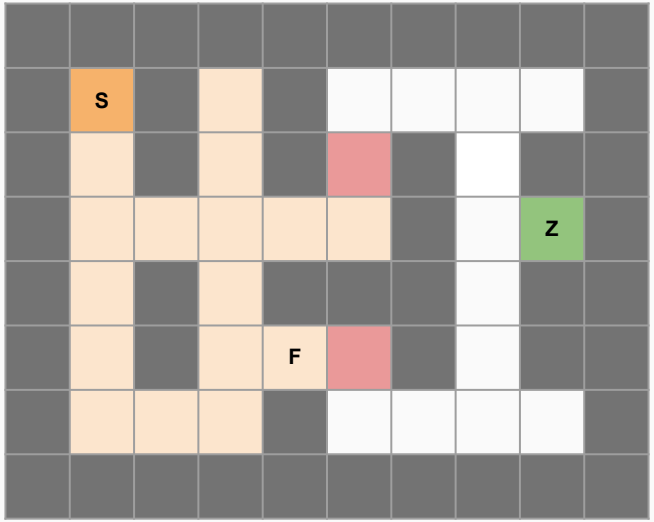
BFS in a Maze

```
queue = [
  (6,3),
  (5,4),
  (2,5)
]
```



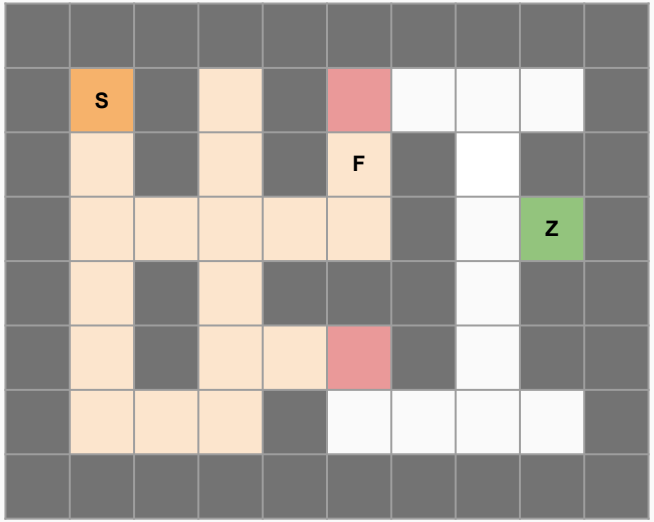
BFS in a Maze

```
queue = [
  (5,4),
  (2,5),
  (5,5)
]
```



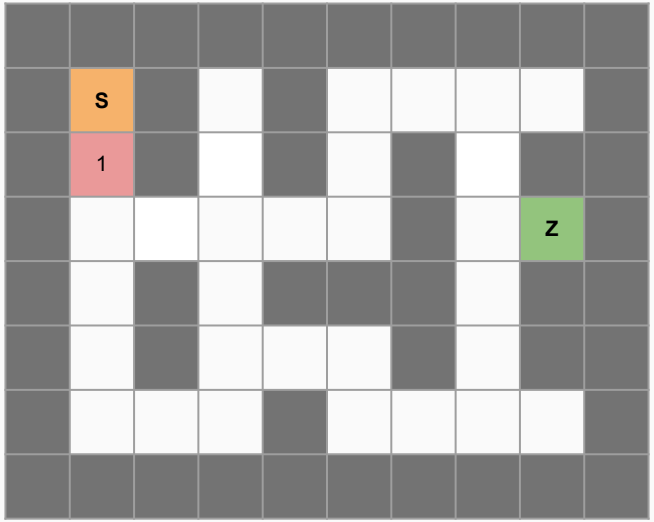
BFS in a Maze

```
queue = [
  (2,5),
  (5,5),
  (1,5)
]
```



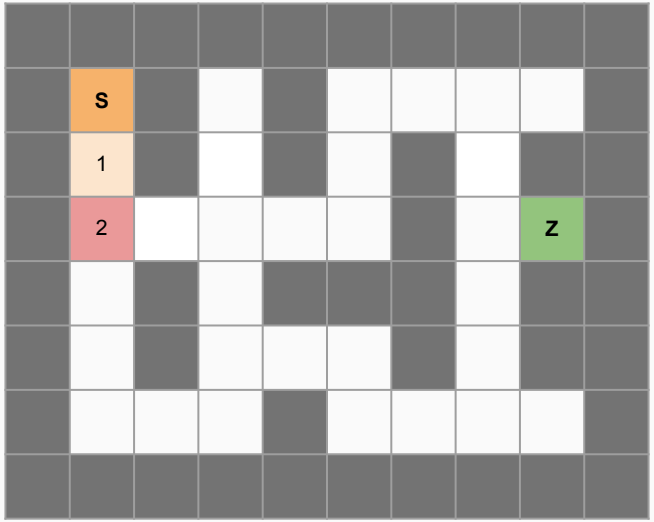
BFS in a Maze

```
queue = [
  (2,1)
]
```



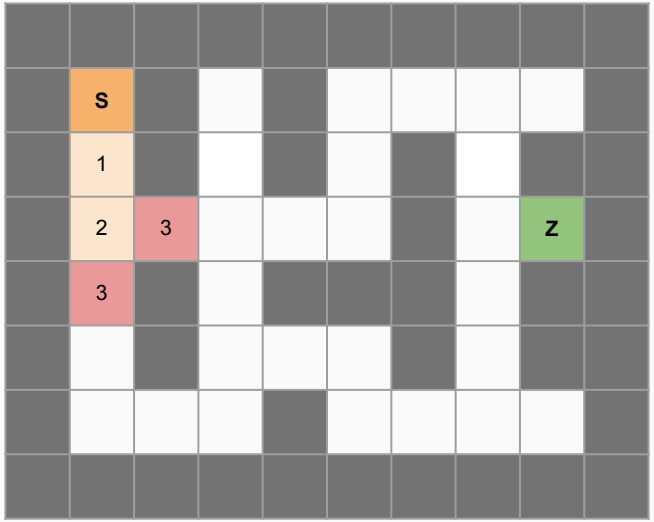
BFS in a Maze

```
queue = [
   (3,1)
]
```



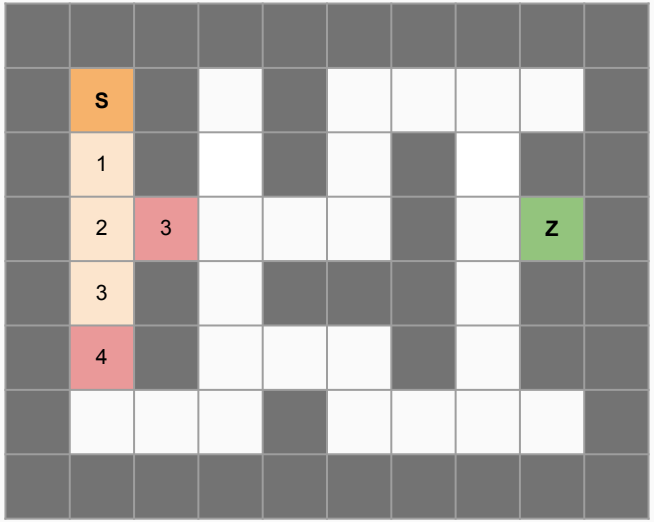
BFS in a Maze

```
queue = [
   (4,1),
   (3,2)
]
```



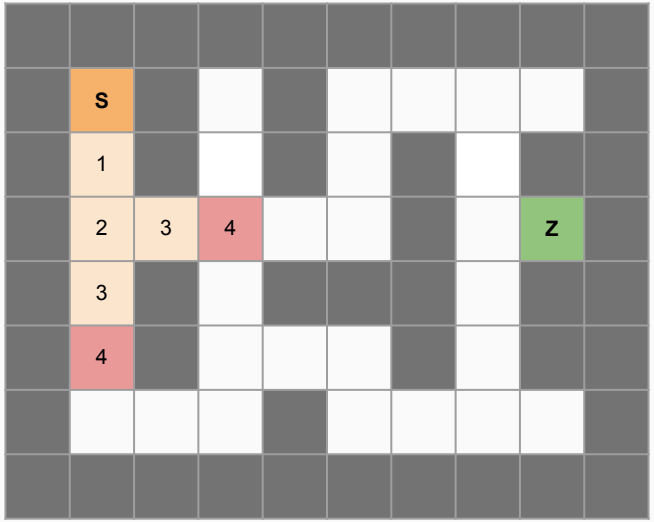
BFS in a Maze

```
queue = [
   (3,2),
   (5,1)
]
```



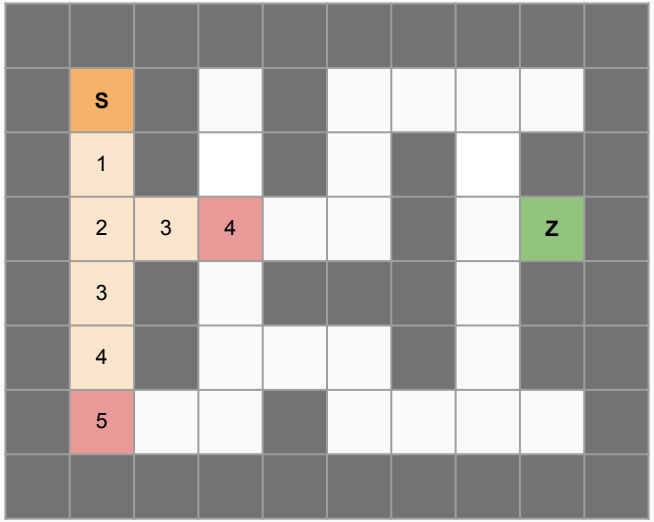
BFS in a Maze

```
queue = [
   (5,1),
   (3,3)
]
```



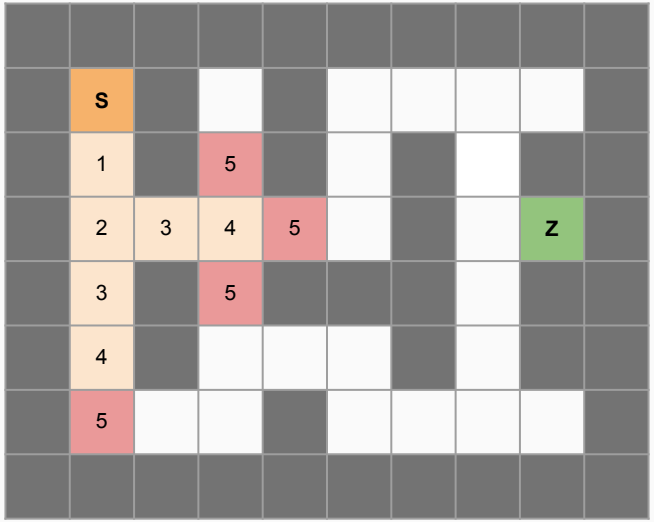
BFS in a Maze

```
queue = [
   (3,3),
   (6,1)
]
```



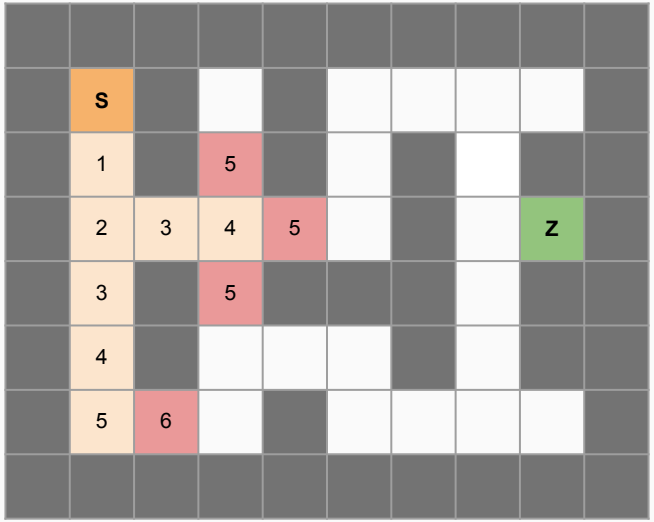
BFS in a Maze

```
queue = [
   (6,1),
   (2,3),
   (4,3),
   (3,4),
]
```



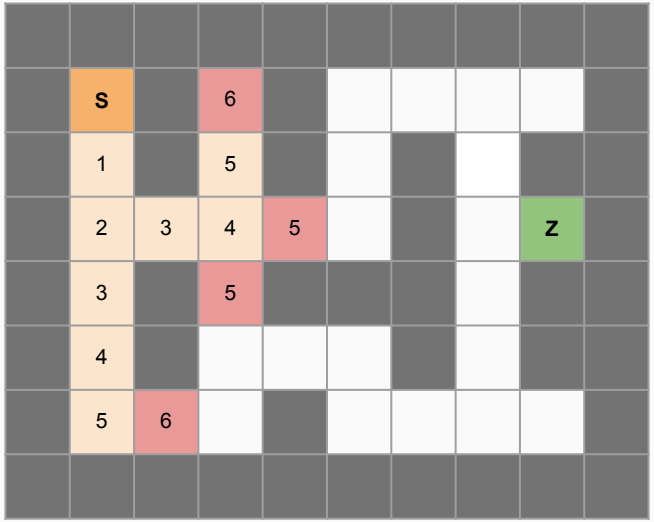
BFS in a Maze

```
queue = [
   (2,3),
   (4,3),
   (3,4),
   (6,2),
]
```



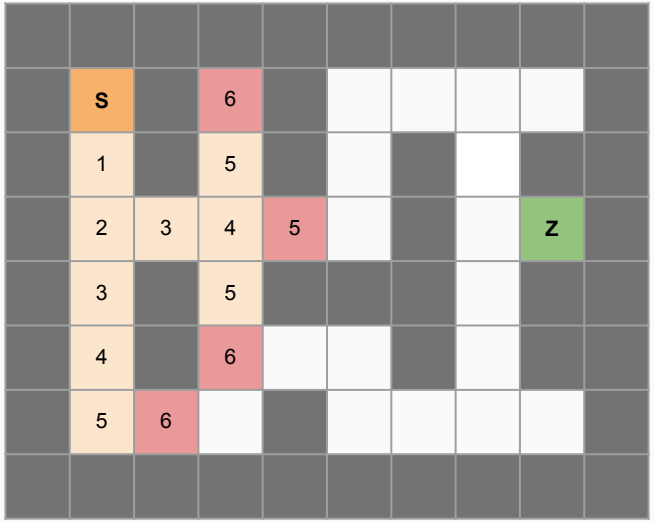
BFS in a Maze

```
queue = [
   (4,3),
   (3,4),
   (6,2),
   (1,3)
]
```



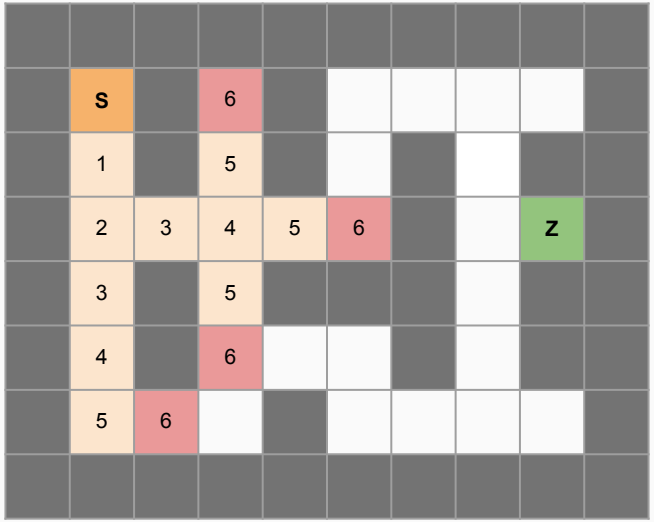
BFS in a Maze

```
queue = [
   (3,4),
   (6,2),
   (1,3),
   (5,3)
]
```



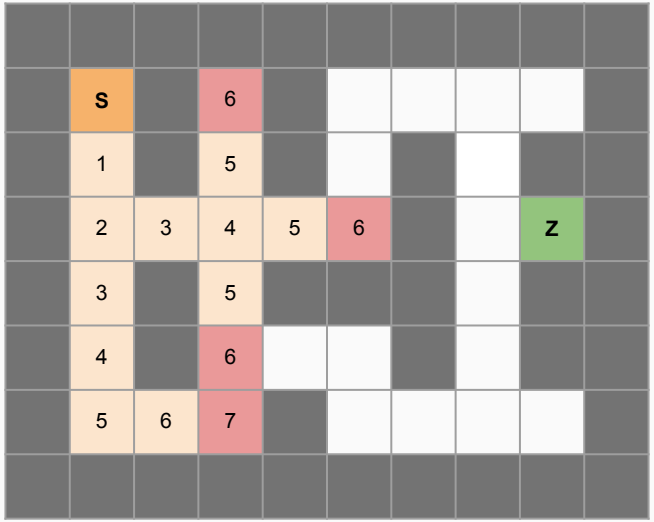
BFS in a Maze

```
queue = [
   (6,2),
   (1,3),
   (5,3),
   (3,5)
]
```



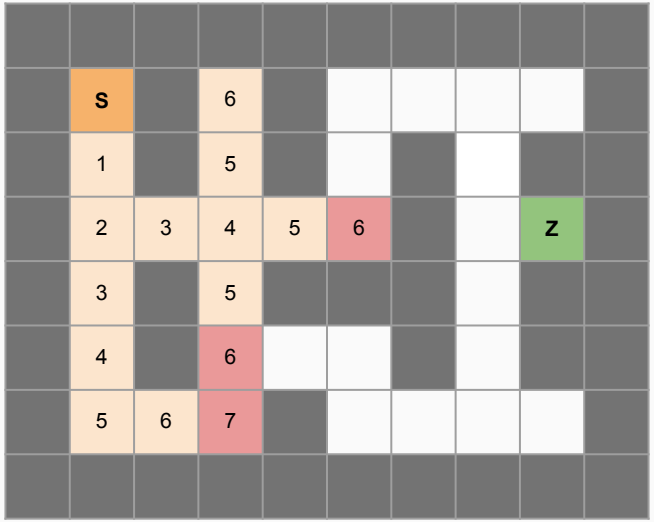
BFS in a Maze

```
queue = [
   (1,3),
   (5,3),
   (3,5),
   (6,3)
]
```



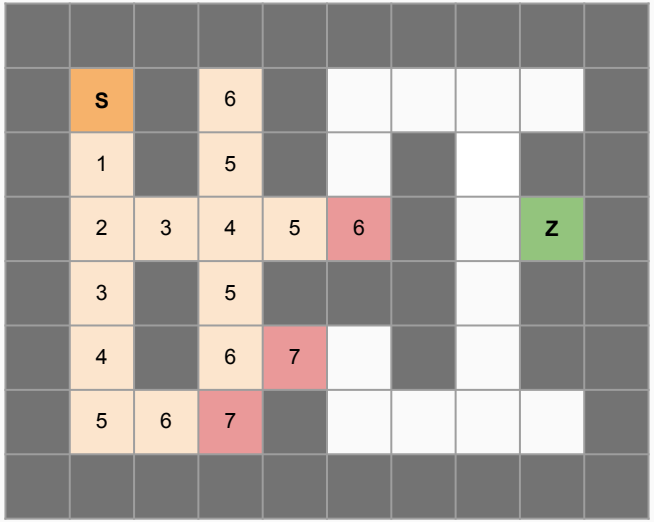
BFS in a Maze

```
queue = [
   (5,3),
   (3,5),
   (6,3)
]
```



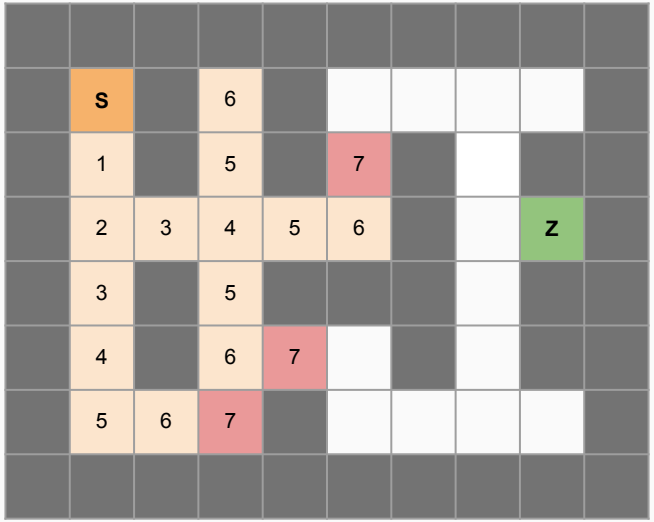
BFS in a Maze

```
queue = [
   (3,5),
   (6,3),
   (5,4)
]
```



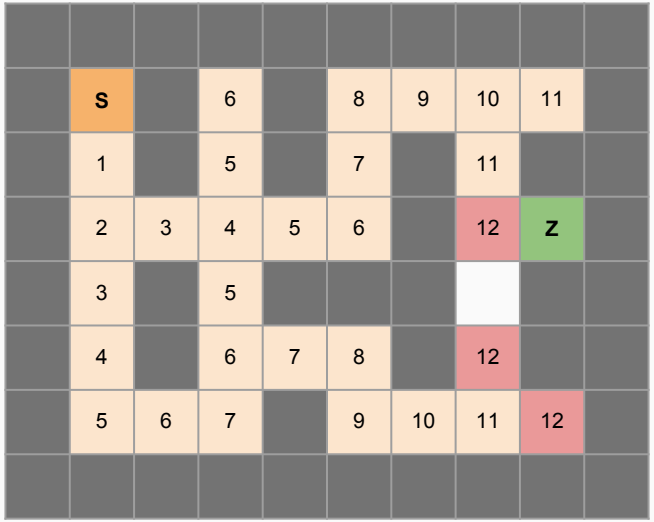
BFS in a Maze

```
queue = [
   (6,3),
   (5,4),
   (2,5)
]
```



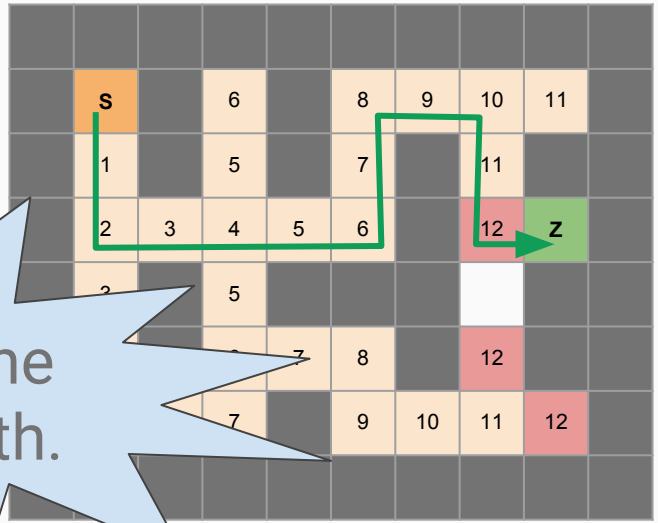
BFS in a Maze

```
queue = [
   (5,7),
   (6,8),
   (3,7)
]
```



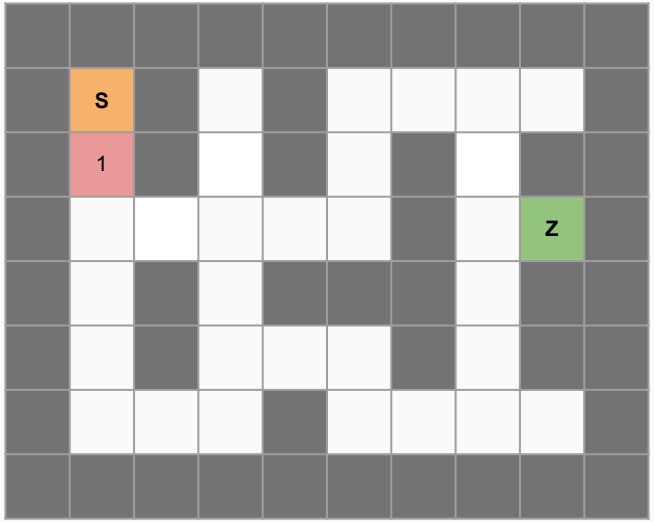
BFS in a Maze

```
queue = [
   (5,7),
   (6,8),
   (3,7)
]
```

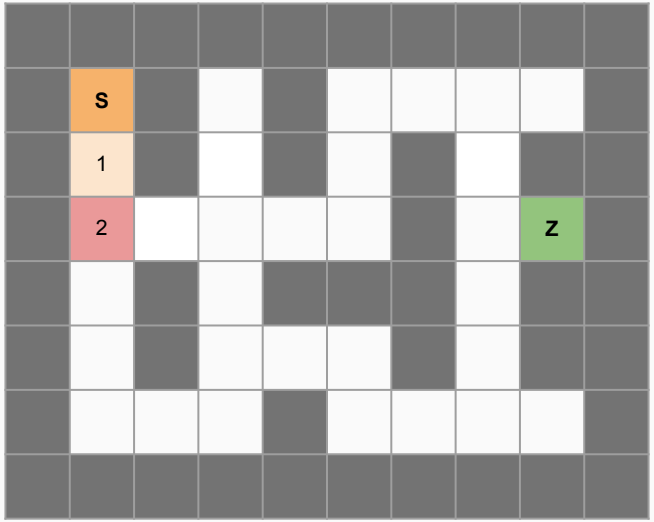


BFS finds the shortest path.

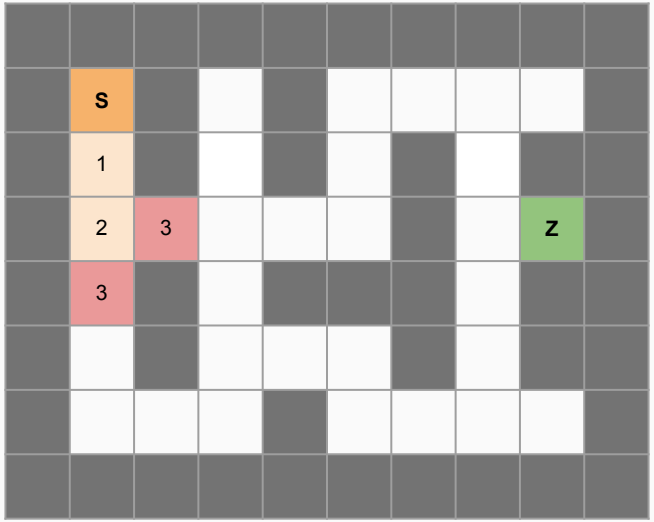
DFS in a Maze



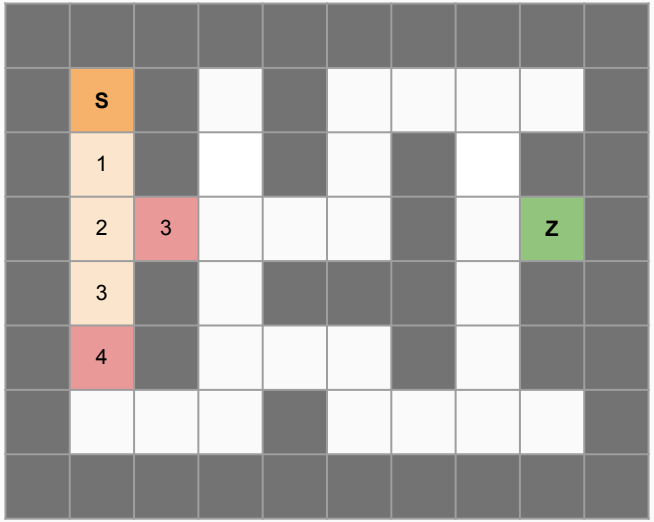
DFS in a Maze



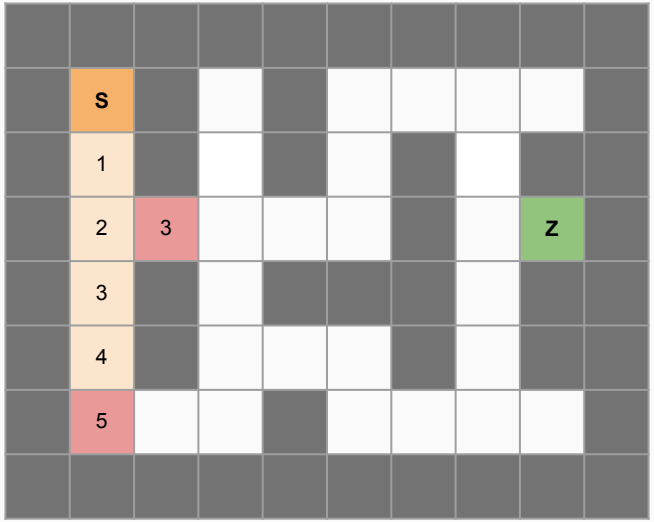
DFS in a Maze



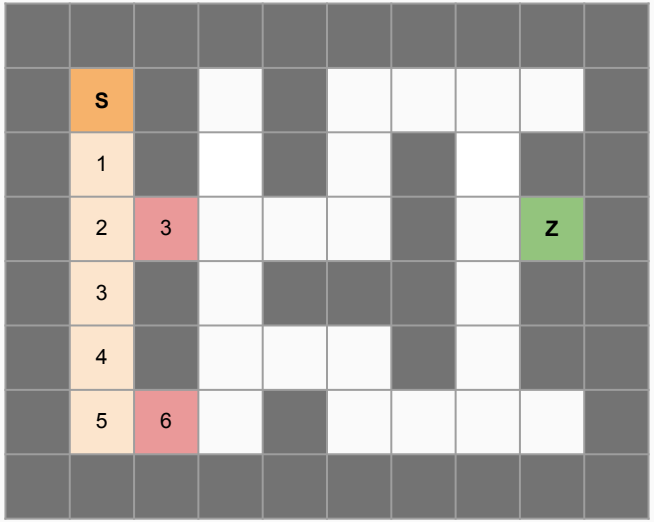
DFS in a Maze



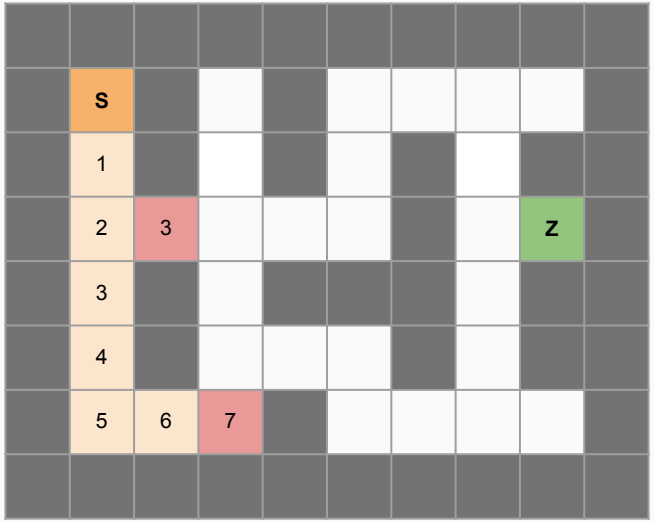
DFS in a Maze



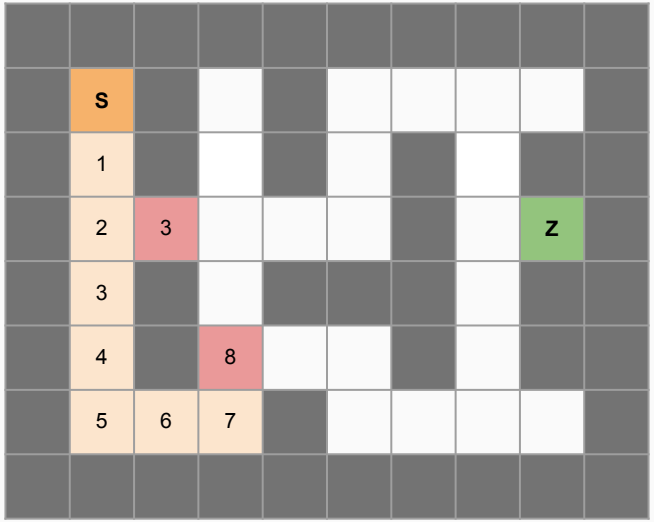
DFS in a Maze



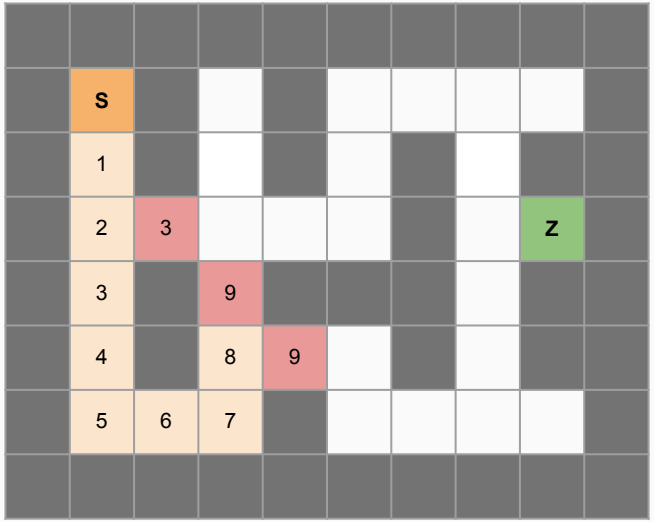
DFS in a Maze



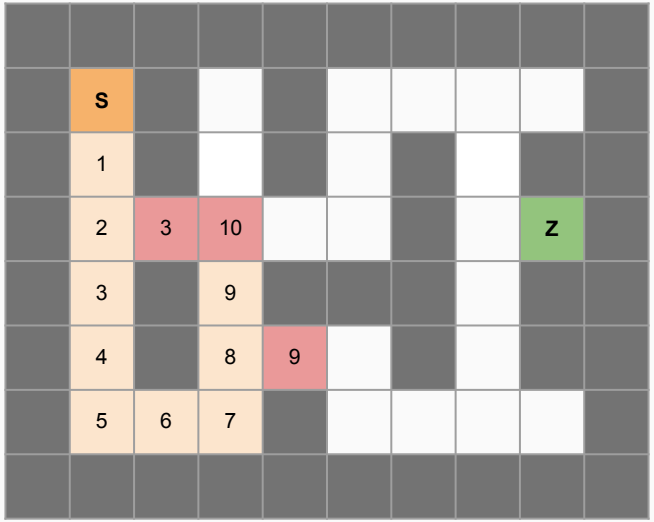
DFS in a Maze



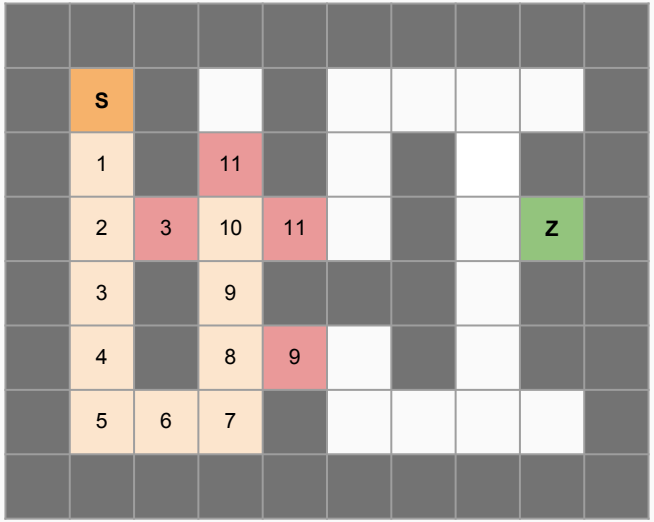
DFS in a Maze



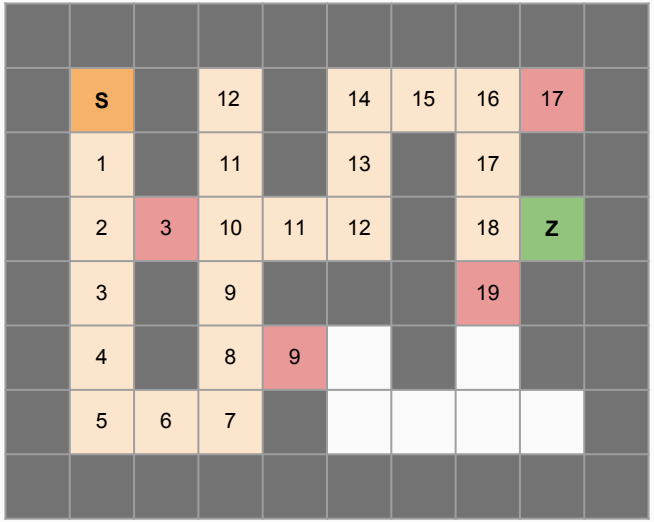
DFS in a Maze



DFS in a Maze

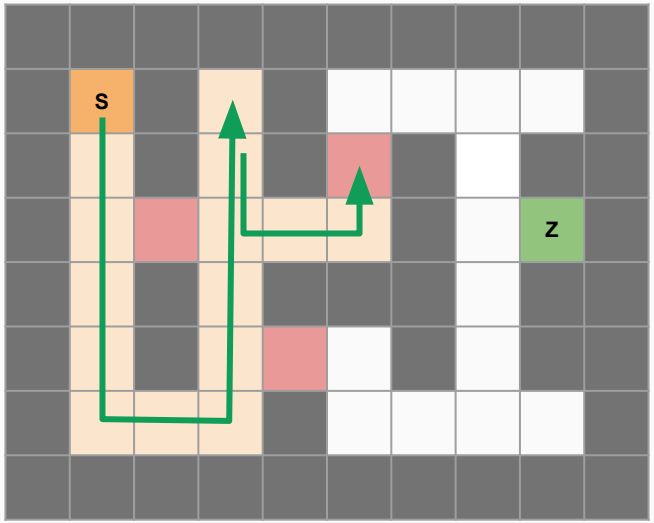


DFS in a Maze

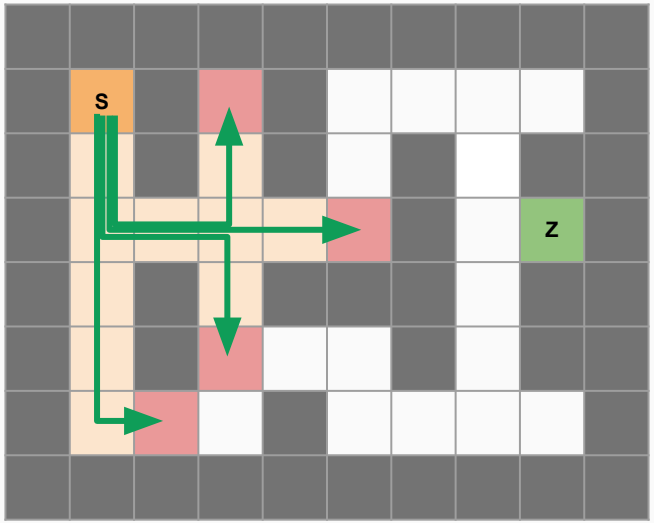


DFS vs. BFS

DFS:

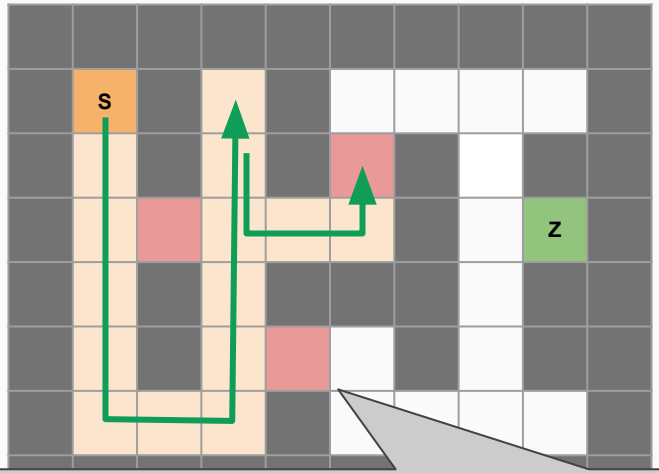


BFS:



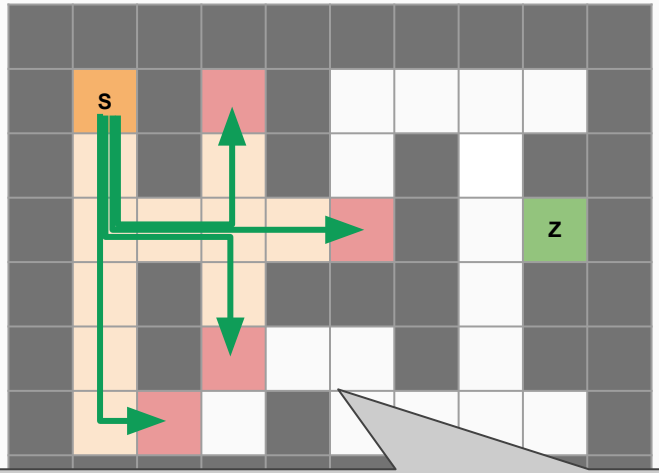
DFS vs. BFS

DFS:



Depth-First-Search:
Searches as far as possible,
returns when stuck.

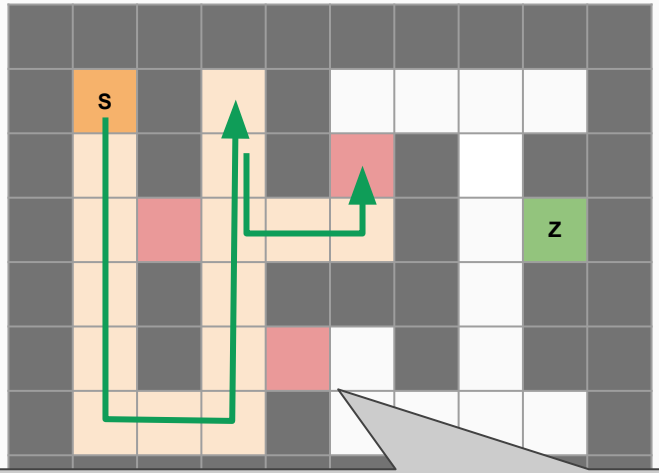
BFS:



Breadth-First-Search:
Searches all paths at the same time.

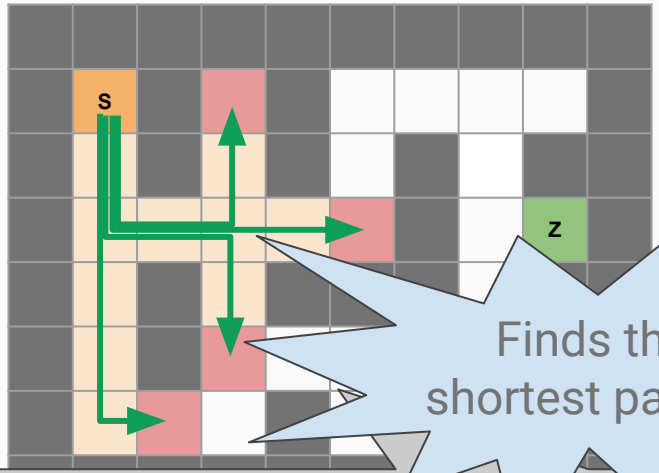
DFS vs. BFS

DFS:



Depth-First-Search:
Searches as far as possible,
returns when stuck.

BFS:



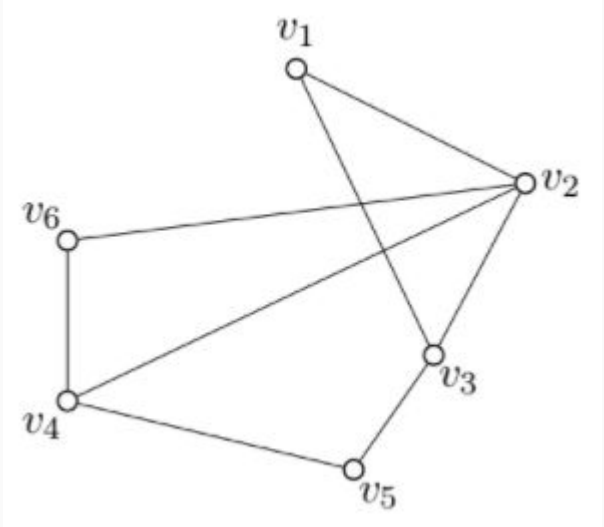
Finds the
shortest path :-)

Breadth-First-Search:
Searches all paths at the same time.

Searching Graphs:

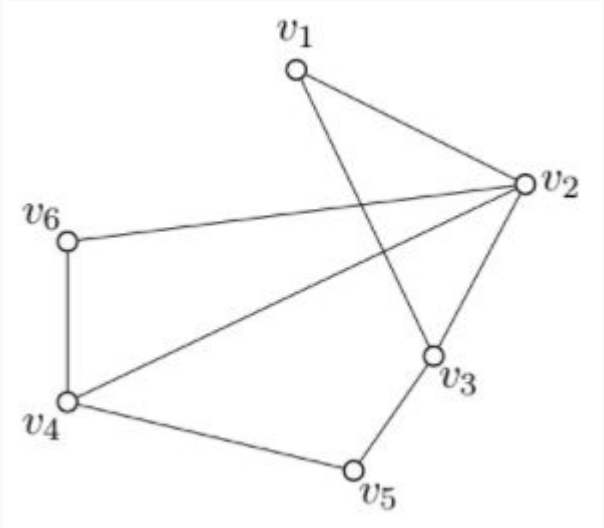
```
mark all vertices not visited
mark start as visited
vertices to look at = {start}

while we still have vertices to look at {
   take one of them
   for all its unvisited neighbors n {
      mark n visited
      put n into vertices to look at
   }
}
```



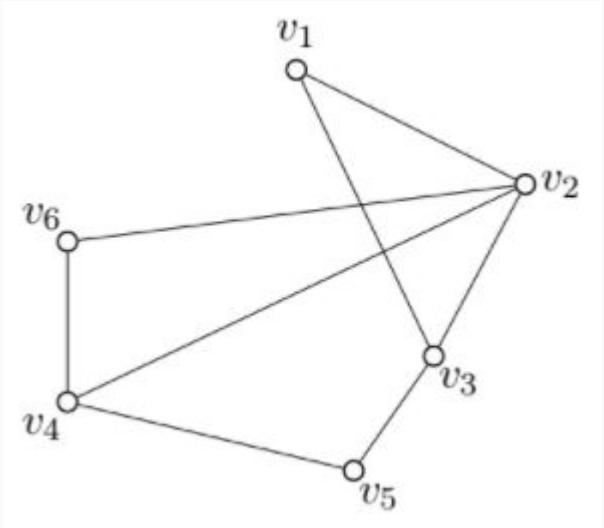
Searching Graphs: DFS

```
vector<int> stack;
visited[start] = true; stack.push_back(start);
while !stack.empty() {
  v = stack.pop()
  for (int n : a[v]) {
    if (!visited[n]) {
      visited[n] = true;
      stack.push_back(n);
    }
  }
}
```

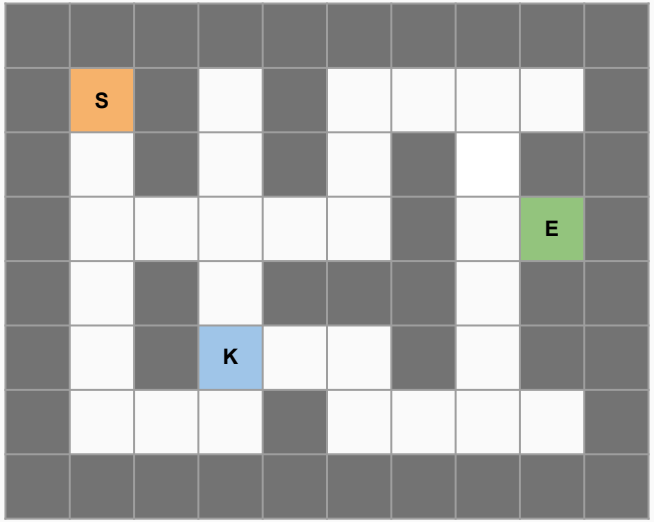


Searching Graphs: BFS

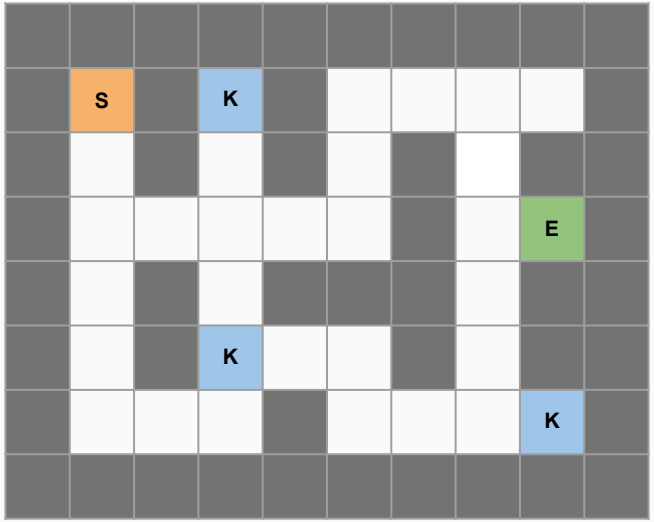
```
queue<int> q;
visited[start] = true; q.push_back(start);
while !q.empty() {
  v = q.pop_front()
  for (int n : a[v]) {
    if (!visited[n]) {
      visited[n] = true;
      q.push_back(n);
    }
  }
}
```



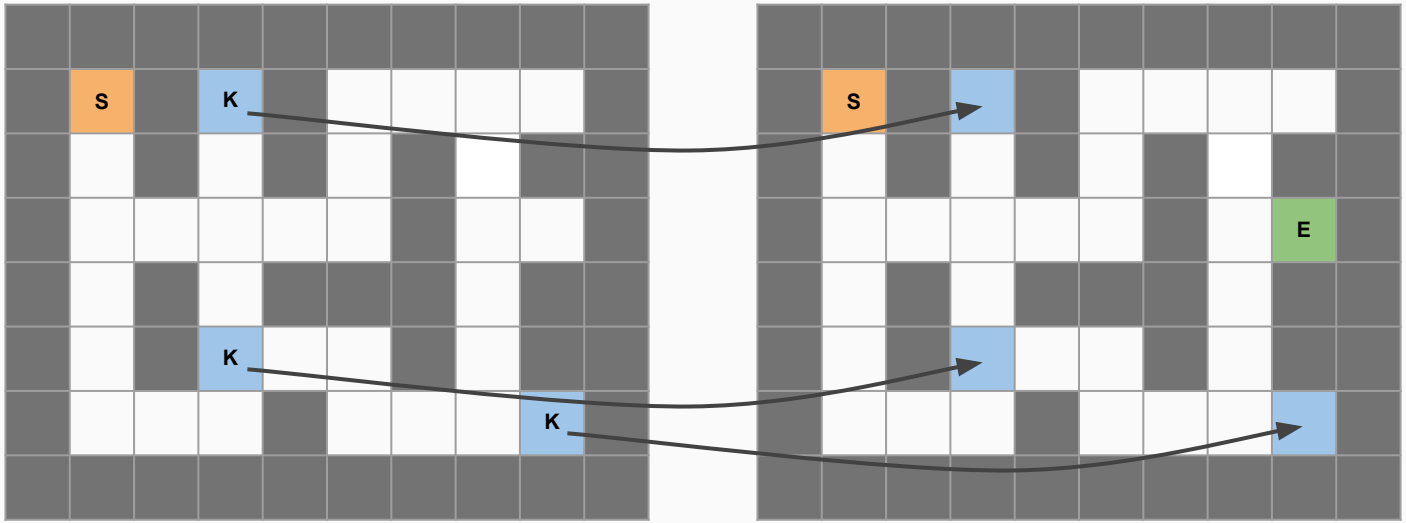
Problem:
Shortest Path in a Maze, Key Opens Exit



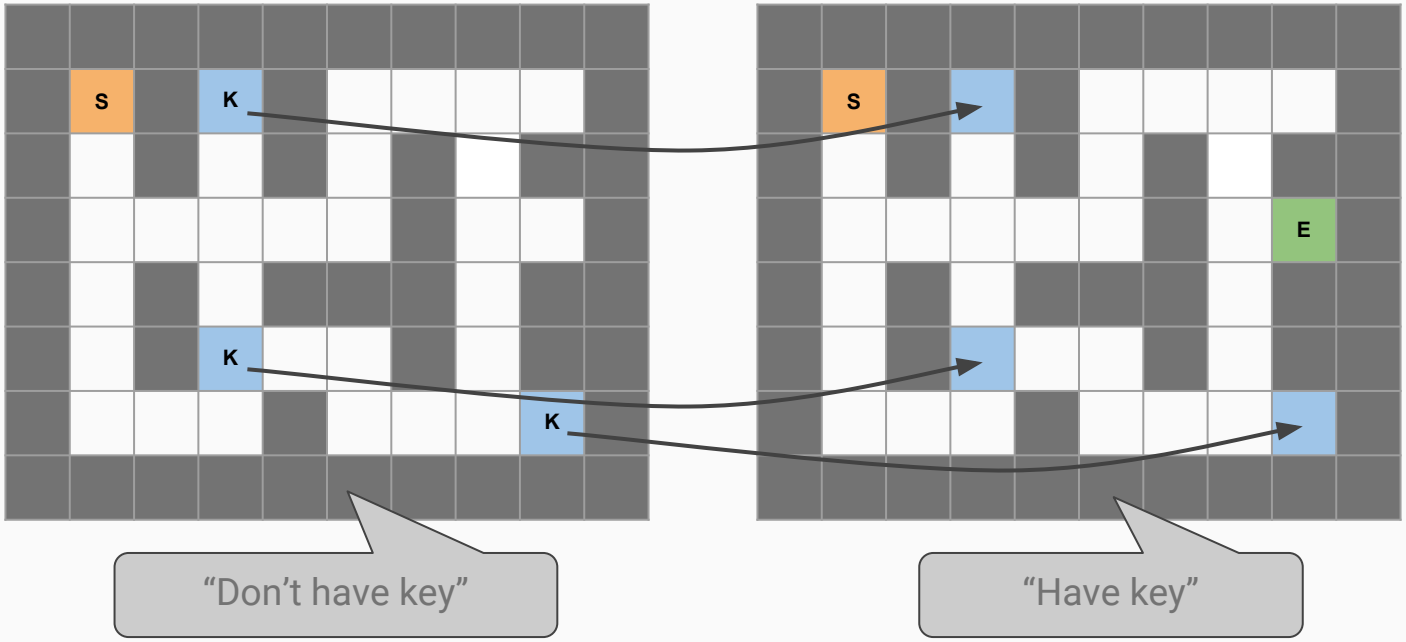
Problem:
Shortest Path in a Maze, Key Opens Exit



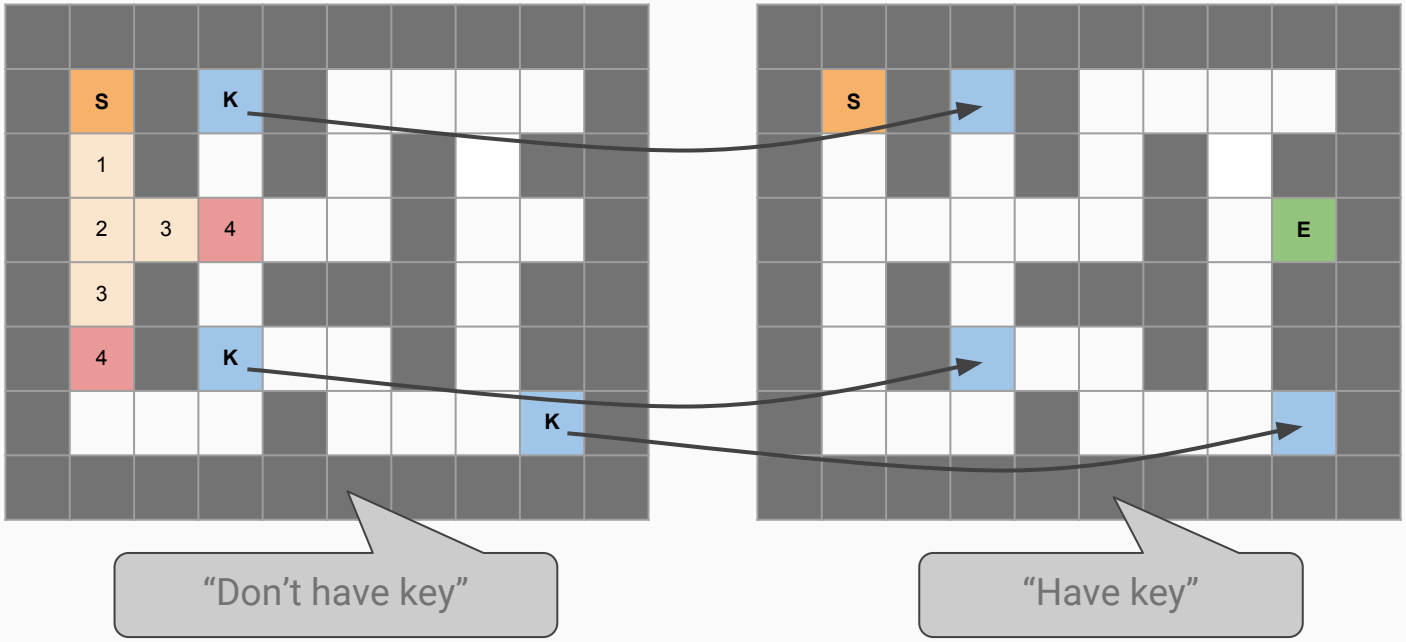
Problem:
Shortest Path in a Maze, Key Opens Exit



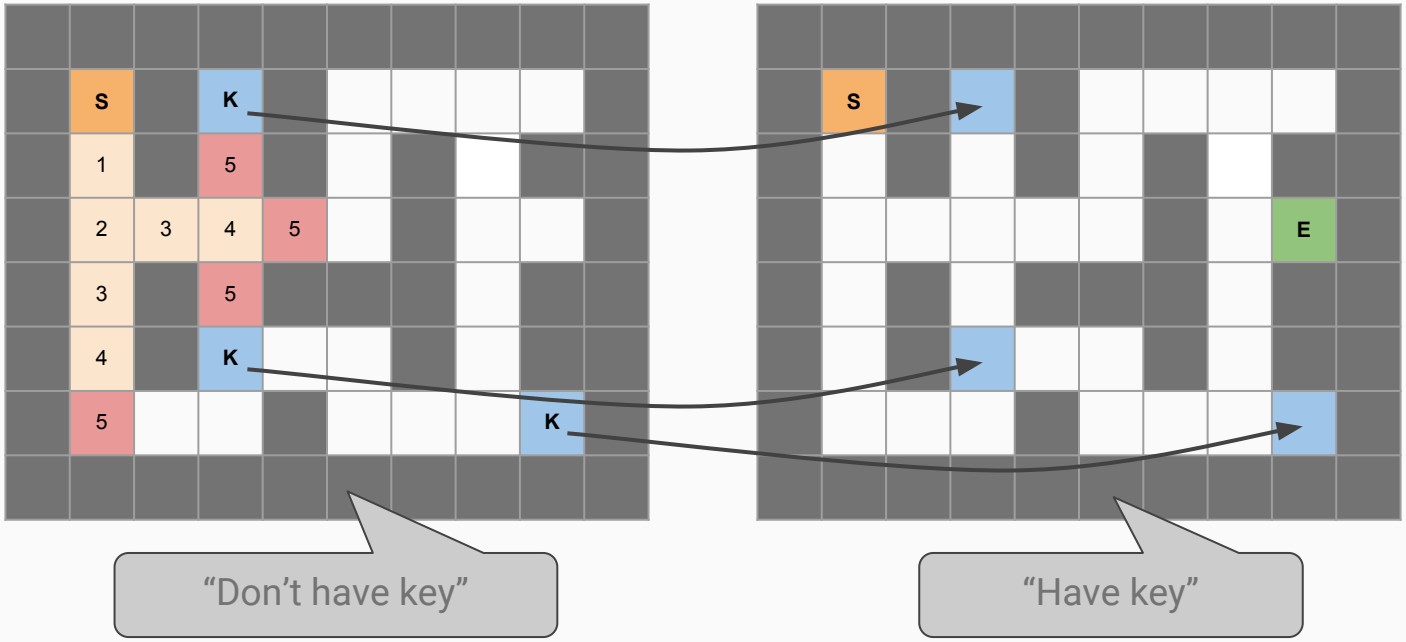
Problem:
Shortest Path in a Maze, Key Opens Exit



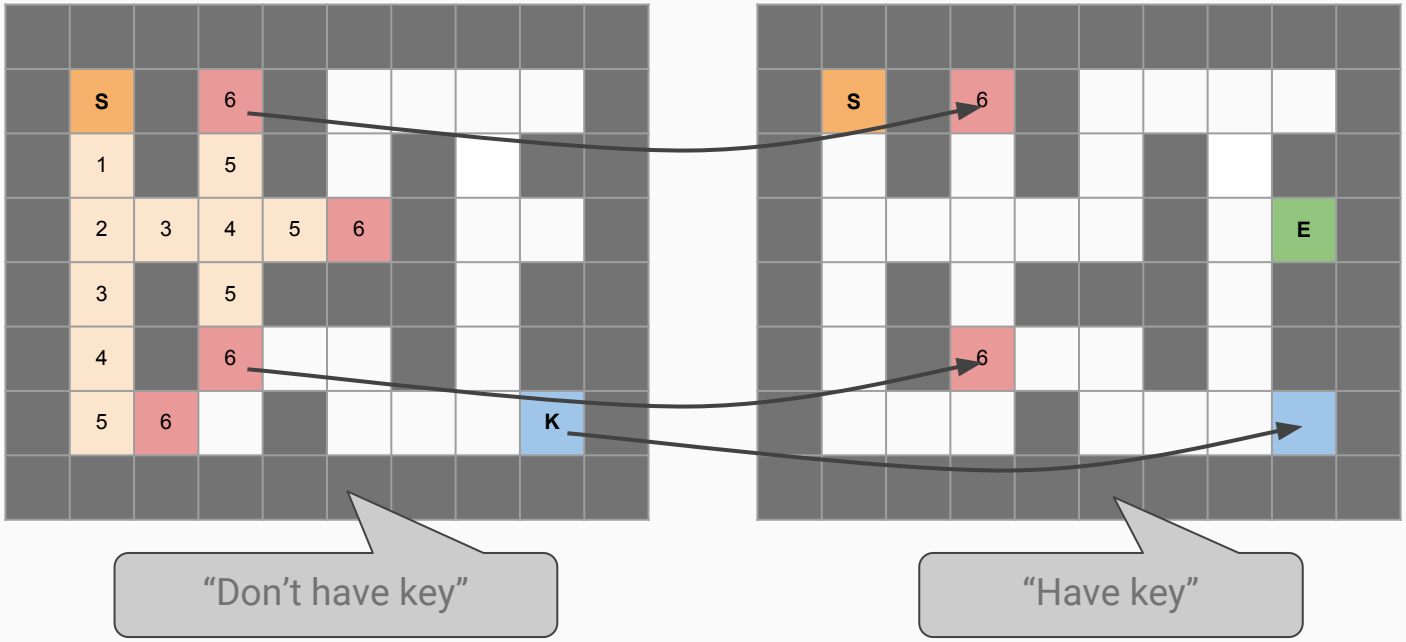
Problem:
Shortest Path in a Maze, Key Opens Exit



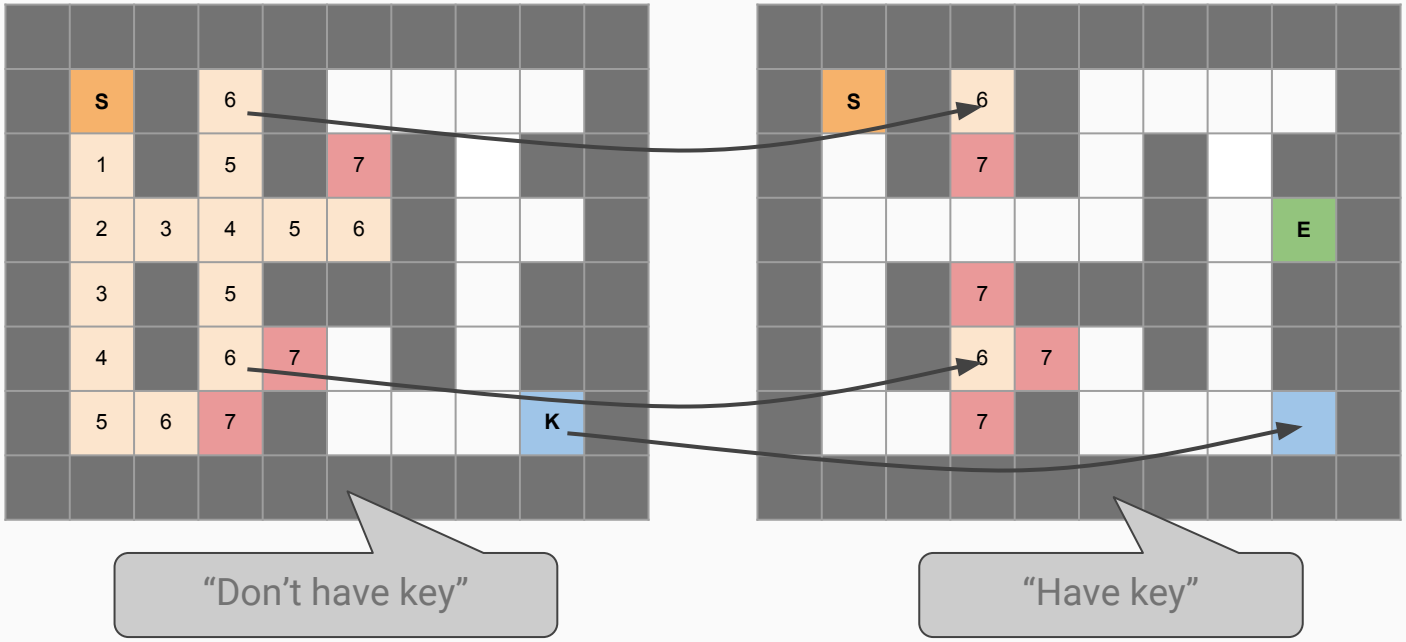
Problem:
Shortest Path in a Maze, Key Opens Exit



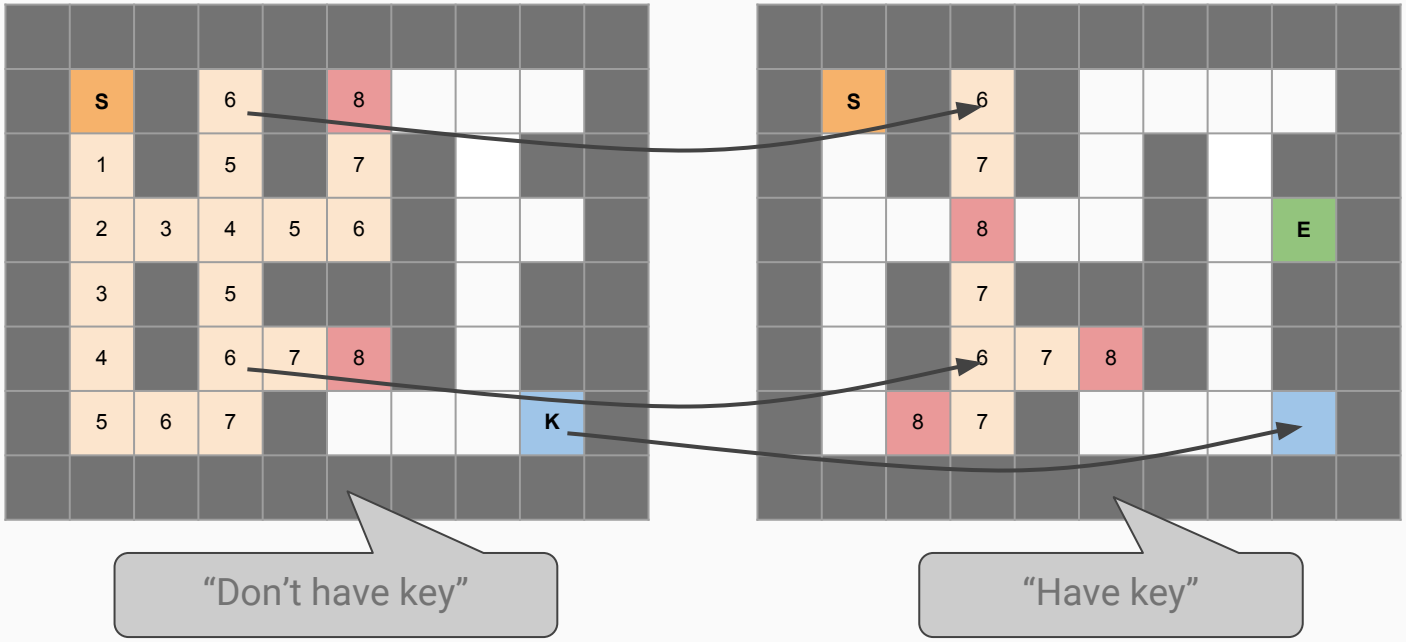
Problem:
Shortest Path in a Maze, Key Opens Exit



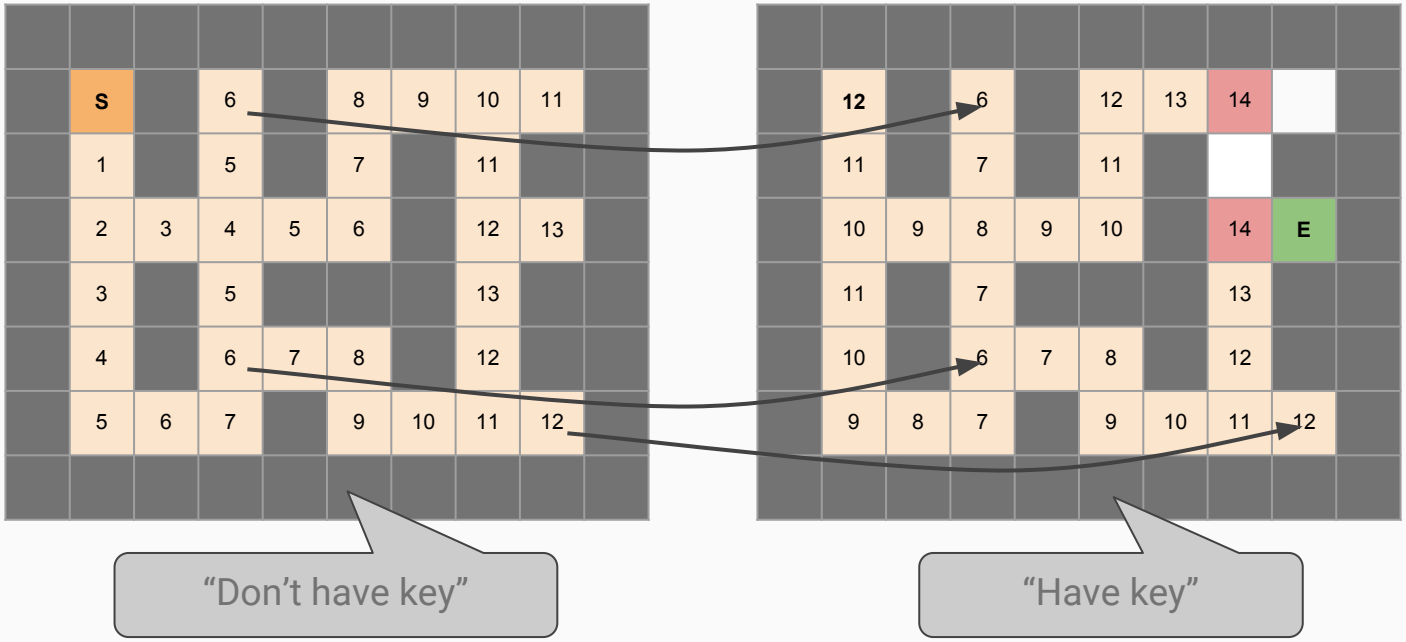
Problem:
Shortest Path in a Maze, Key Opens Exit



Problem:
Shortest Path in a Maze, Key Opens Exit



Problem:
Shortest Path in a Maze, Key Opens Exit



Problem:
Shortest Path in a Maze, Key Opens Exit

Have a key?

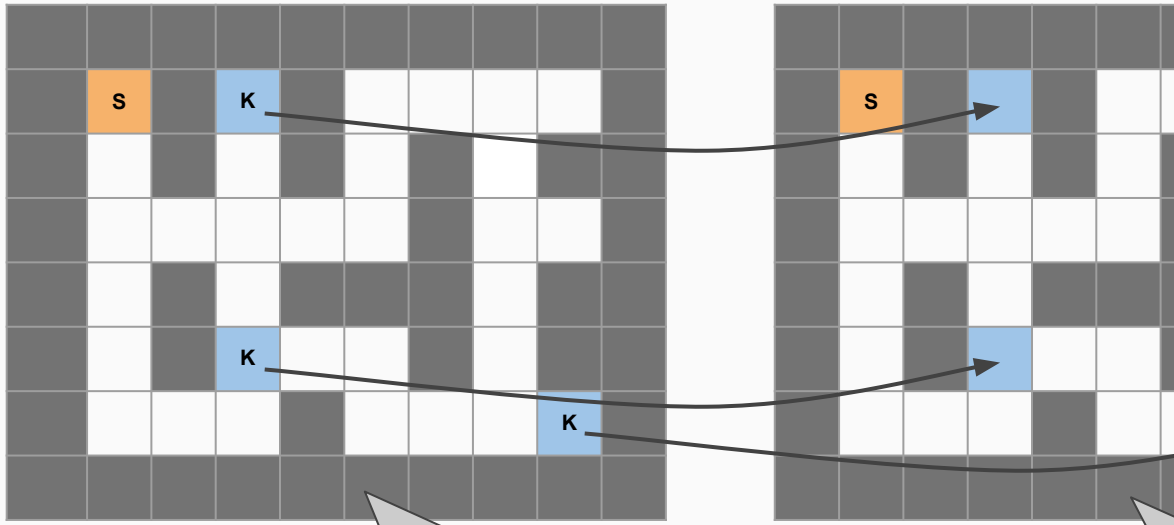
Position: <int, int, bool>

Start = start_x, start_y, false

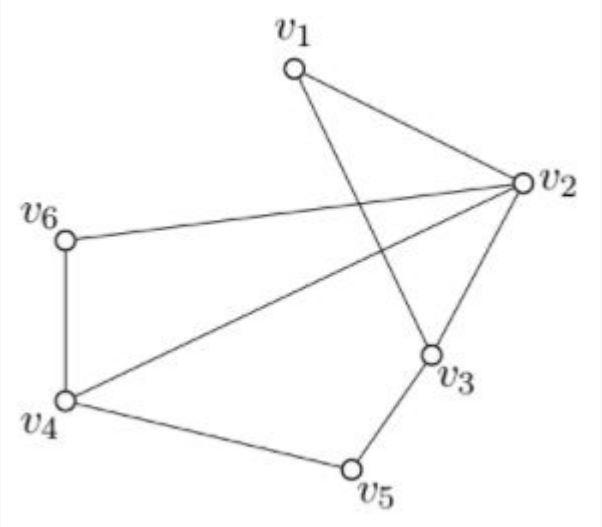
End = end_x, end_y, true

Neighbours of (i,j,k) are:

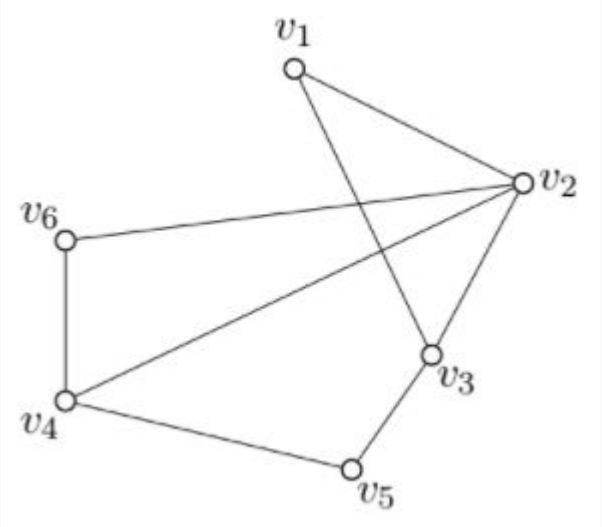
- (i, j+1, k || m[i][j+1] == 'K'),
- (i, j-1, k || m[i][j-1] == 'K'),
- (i+1, j, k || m[i+1][j] == 'K'),
- (i-1, j, k || m[i-1][j] == 'K'),



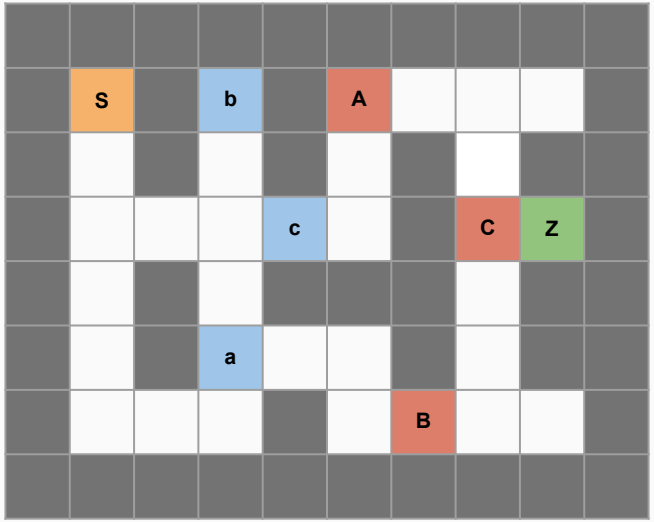
Problem:
Shortest Path from A to B



Problem:
Shortest Path from A to B of Even Length

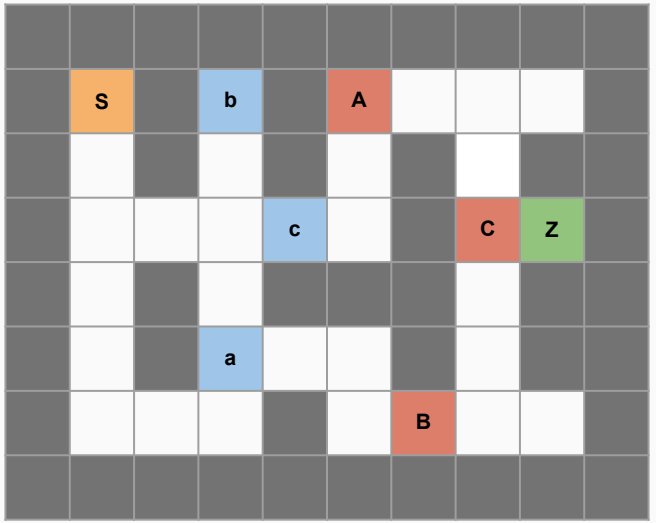


Problem:
Shortest Path in a Maze, Keys Open Doors



Problem:
Shortest Path in a Maze, Keys Open Doors

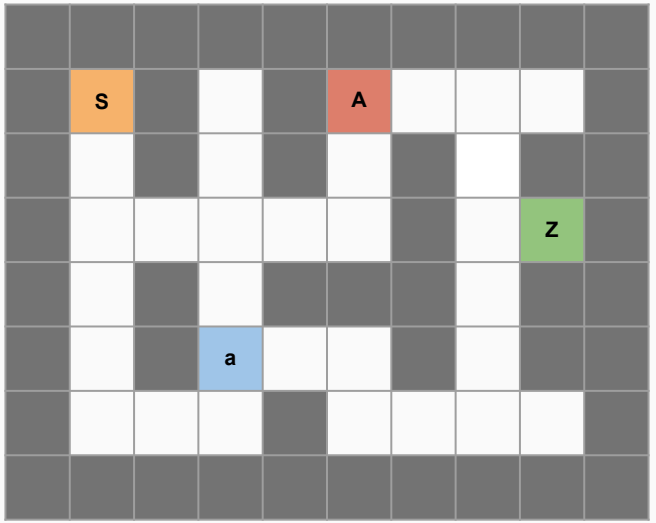
Position: ?



Problem:
Shortest Path in a Maze, Keys Open Doors

Position: <int, int, bool>

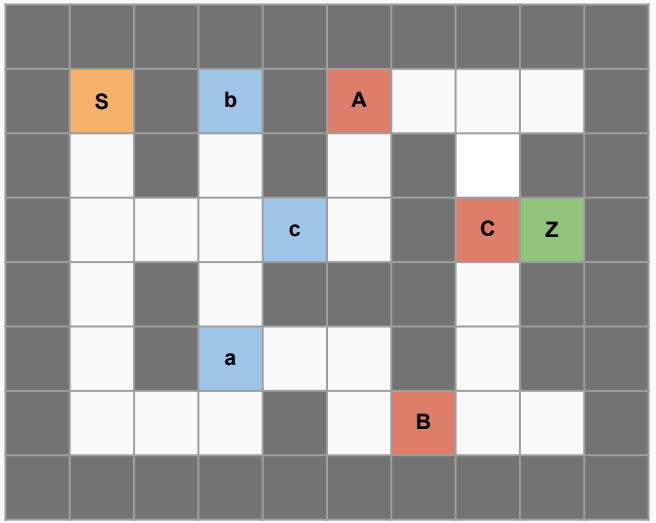
<row, column, have_key?>



Problem:
Shortest Path in a Maze, Keys Open Doors

Position: <int, int, bool, bool, bool>

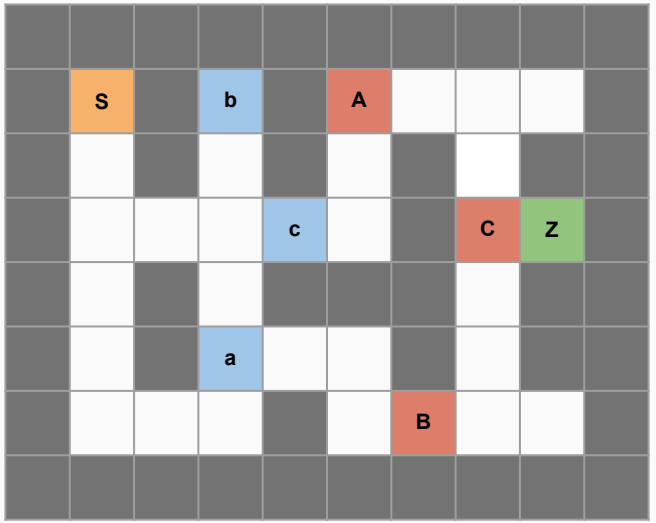
<row, column, have_key_A, have_key_B, have_key_C>



Problem:
Shortest Path in a Maze, Keys Open Doors

Position: <int, int, bool, bool, ..., bool>

<row, column, have_key_A, have_key_B, ..., have_key_X>



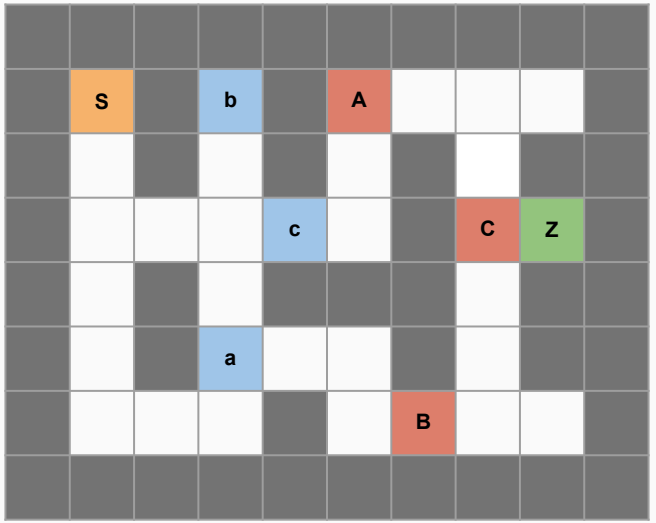
Problem:
Shortest Path in a Maze, Keys Open Doors

Position: <int, int, vector<bool>>

<row, column, which keys I have>

The neighbors of (x, y, keys) are:

(nx, ny, keys) for all neighbors (nx, ny) of (x, y) such that
(nx, ny) is free or a door for which we have a key,
(nx, ny, keys') for all neighbors (nx, ny) of (x, y) such that
(nx, ny) has key K, where keys' is keys with
keys'[K] := true



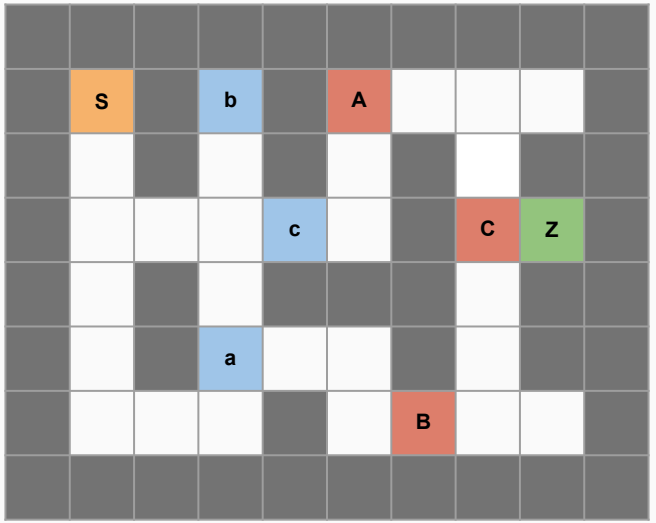
Problem:
Shortest Path in a Maze, Keys Open Doors

Position: <int, int, int>

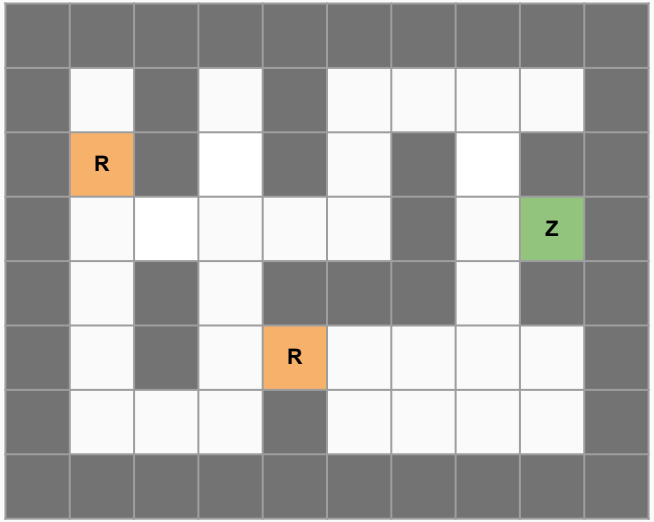
<row, column, bitmap of which keys I have>

The neighbors of (x, y, keys) are:

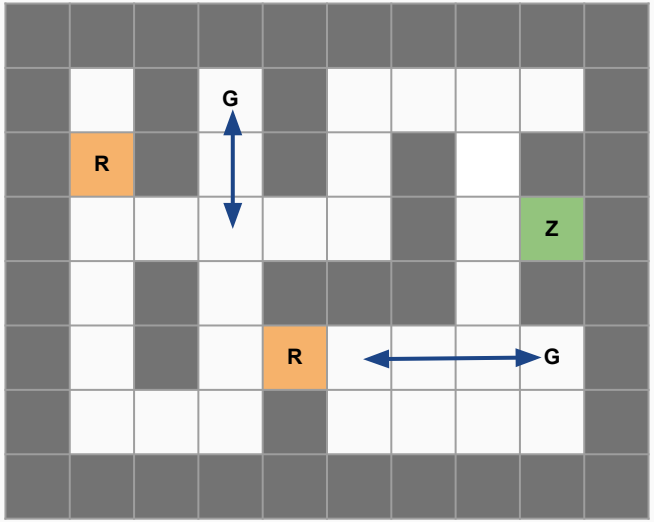
(nx, ny, keys) for all neighbors (nx, ny) of (x, y) such that
(nx, ny) is free or a door for which we have a key,
(nx, ny, keys | (1 << K)) for all neighbors (nx, ny) such that
(nx, ny) has key K



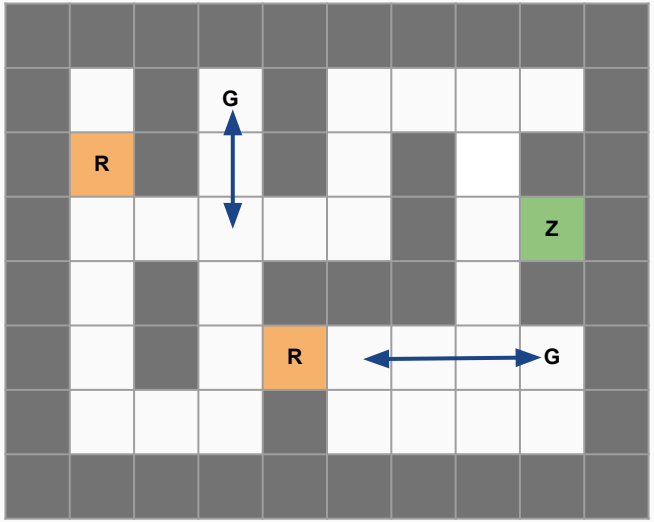
Problem:
Two Robots in a Maze



Problem:
Two Robots in a Maze, with Guards

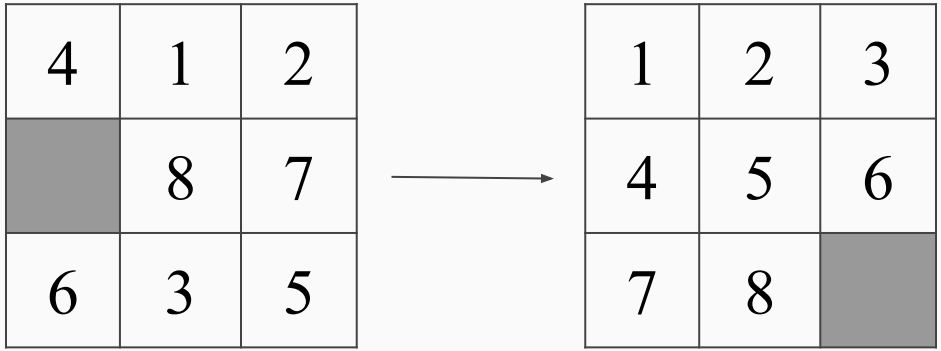


Problem:
Two Robots in a Maze, with Guards



IOI 2003

Problem:
The 8-Puzzle



Problem:
The 8-Puzzle

Still just a BFS!

Position: ???

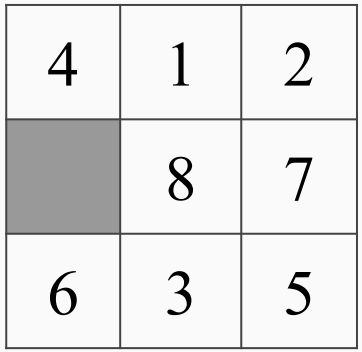
Image: https://www.geeksforgeeks.org/wp-content/uploads/image6.png

Problem:
The 8-Puzzle

Still just a BFS!

Position: string

"412x87635"



How to set up array ‘visited’?
How to generate all positions?

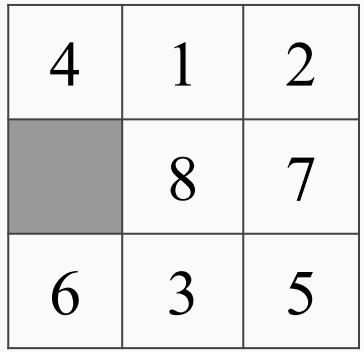
Image: https://www.geeksforgeeks.org/wp-content/uploads/image6.png

Problem:
The 8-Puzzle

Still just a BFS!

Position: string

"412x87635"



visited = map<string, bool>

Generate new positions
only when needed.

#positions ≤ 9! ~ 360’000

Problem:
Cups

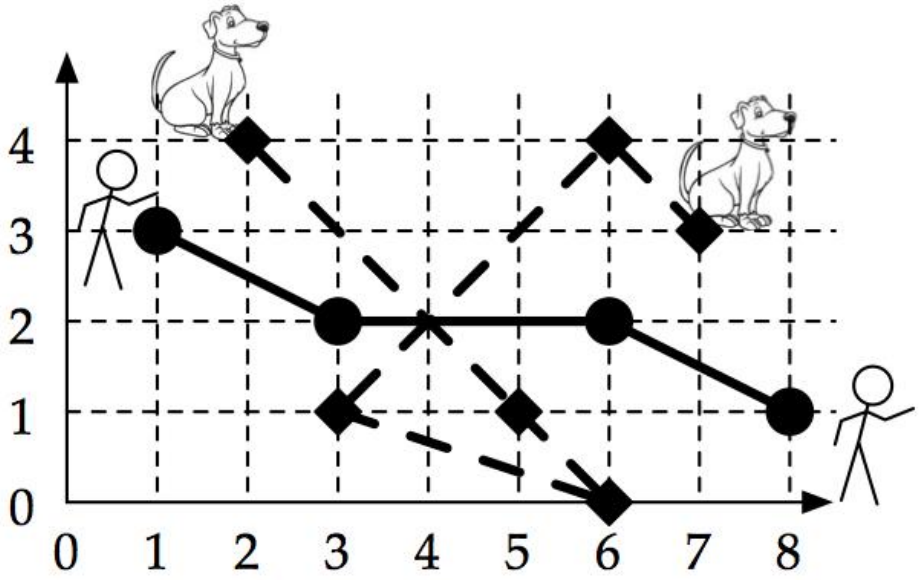
Have = Three measuring cups,
with volumes A, B, C < 200ml.

Task = measure K ml with
as few operations as possible.

Problem:
Dog Walk

Have = Prescribed path for you and for your dog.
Every second, each of you can
-- move to the next point,
-- move to the previous point,
-- stay put.

Task = find minimum length of leash to make the walk possible.



Summary

Summary

List of edges:

```
e = vector<pair<int
```

Adjacency lists:

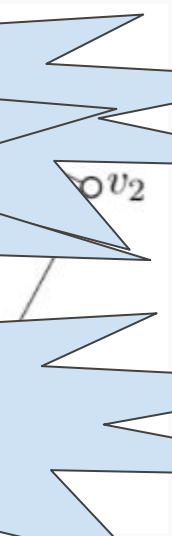
```
a = vector<vector<int
```

Adjacency matrix:

```
a = vector<vector<bool>>
```

Graphs are everywhere!

re are many other variations.



Summary

Master trick = find the “state space” of the problem.

The state space is a graph, but you do not have to store it explicitly.