

# CSE 250: Maze Exploration

## Lecture 17

Oct 11, 2023

# Reminders

- WA3 due Sun, Oct 15 at 11:59 PM

# The Stack ADT

A stack of objects on top of one another.

- **Push**

Put a new object on top of the stack.

- **Pop**

Remove the object from the top of the stack.

- **Top**

Peek at what's on top of the stack.

# The Queue ADT

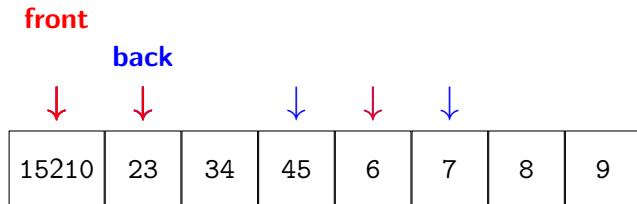
Outside of the US, "queueing" is lining up.

- **Enqueue** (`add(item)` or `offer(item)`)  
Put a new object at the end of the queue.
- **Dequeue** (`remove()` or `poll()`)  
Remove the object from the front of the queue.
- **Peek** (`element()` or `peek()`)  
Peek at what's at the front of the queue.

# Queues vs Stacks

- **Queue**  
First in, First out (FIFO)
- **Stack**  
Last in, First out (LIFO, FILO)

# Circular Buffer



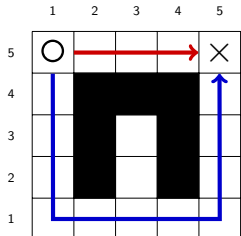
```

list.add(1)           list.remove() → 2   list.add(9)
list.add(2)           list.remove() → 3   list.add(10)
list.add(3)           list.remove() → 4   list.remove() → 6
list.remove() → 1    list.remove() → 5   list.remove() → 7
list.add(4)           list.add(7)         list.remove() → 8
list.add(5)           list.add(8)         list.remove() → 9
list.add(6)

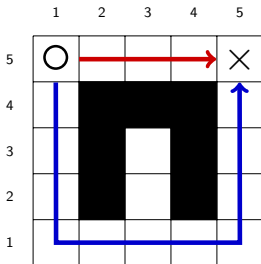
```

# Mazes

- ○ is the start, × 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**



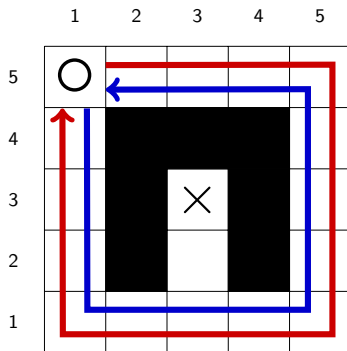
# Mazes



- How do we know which one is best?
- Are there any other problems?



# Mazes



- Priority order doesn't guarantee exploring the entire maze

# Formalizing Maze Solving

## ■ Inputs

- The map: An  $n \times m$  grid of filled/empty squares.
- The  $\circ$  is at position start
- The  $\times$  is at position dest

## ■ Goal

- Compute  $\text{steps}(\text{start}, \text{dest})$ , the minimum steps from start to end.

**How do we define steps?**

# Formalizing Maze Solving

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

# Formalizing Maze Solving

$$\text{steps}(\text{pos}, \text{dest}) = \begin{cases} 0 & \text{if } \text{pos} = \text{dest} \\ \infty & \text{if } \text{pos} \text{ is filled} \\ 1 + \text{min\_nearby}(\text{pos}, \text{dest}) & \text{otherwise} \end{cases}$$

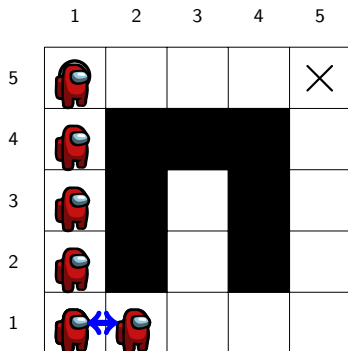
Where...

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

# Formalizing Maze Solving

```
1  public int steps(Point pos, Point dest)
2  {
3      if(pos == dest){ return 0; }
4      else if(is_filled(pos){ return ∞; }
5      else {
6          return 1 + Math.min(
7              steps(pos.moveDown, dest),
8              steps(pos.moveLeft, dest),
9              steps(pos.moveRight, dest),
10             steps(pos.moveUp, dest)
11         );
12     }
13 }
```

# Formalizing Maze Solving



Shortest Path:  $\infty$

# Formalizing Maze Solving

**Problem:** Infinite Loop

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

Mark nodes as visited so you don't visit them twice.

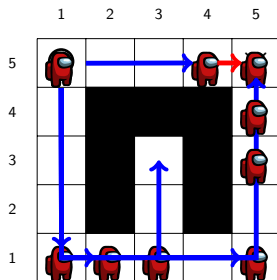
# Formalizing Maze Solving

```
1  public int steps(Point pos, Point dest)
2  {
3      if(pos == dest){ return 0; }
4      else if(is_filled(pos){ return ∞; }
5      else if(is_visited(pos){ return ∞; }
6      else {
7          mark_visited(pos);
8          return 1 + Math.min(
9              steps(pos.moveDown, dest),
10             steps(pos.moveLeft, dest),
11             steps(pos.moveRight, dest),
12             steps(pos.moveUp, dest)
13         );
14     }
15 }
```





# Formalizing Maze Solving



**Shortest:**  $1 + \min(\infty, \infty, \infty, \infty) = \infty$

Among Us Beans ©Innersloth LLC

# Formalizing Maze Solving

**Problem:** The first time you visit a node it may be via a longer path!

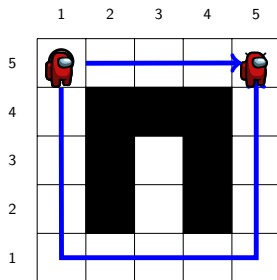
Unmark nodes as you leave them.

# Formalizing Maze Solving

```
1 public int steps(Point pos, Point dest)
2 {
3     if(pos == dest){ return 0; }
4     else if(is_filled(pos){ return ∞; }
5     else if(is_visited(pos){ return ∞; }
6     else {
7         mark_visited(pos);
8         int stepCount = 1 + Math.min(
9             steps(pos.moveRight, dest),
10            /* ... */
11        );
12        unmark_visited(pos);
13        return stepCount;
14    }
15 }
```



# Formalizing Maze Solving



**Shortest:**  $1 + \min(11, 3, \infty, \infty) = 4$

Among Us Beans ©Innersloth LLC

# Formalizing Maze Solving

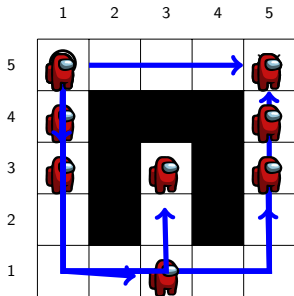
**Question:** What path did we take?

Track the current path in a `Stack`

# Formalizing Maze Solving

```
1  public Array[Point] steps(Point pos, Point dest, Stack visited)
2  {
3      if(pos == dest){ return visited.toArray(); } ←
4      else if(is_filled(pos){ return null; }
5      else if(visited.contains(pos){ return null; } ←
6      else {
7          visited.push(pos); ←
8          int steps = shortest_array(
9              steps(pos.moveRight, dest),
10             /* ... */
11             );
12         visited.pop(pos); ←
13     }
14 }
```

# Formalizing Maze Solving



(1,5)	(1,5)
(1,4)	<del>(2,5)</del>
(1,3)	<del>(3,5)</del>
(1,2)	<del>(4,5)</del>
(1,1)	<del>(5,5)</del>
(2,1)	<del>(2,1)</del>
(3,1)	(3,1)
(4,1)	(4,1)
(5,1)	(5,1)
(5,2)	(5,2)
(5,3)	(5,3)
(5,4)	(5,4)
(5,5)	(5,5)
✓	(3,2)
	(3,3)
	×

# Queues

**Thought Question:** How would we need to change the algorithm to use a queue instead of a stack?