

CSE12 - Lecture 8

Monday, October 16, 2023 8:00 AM

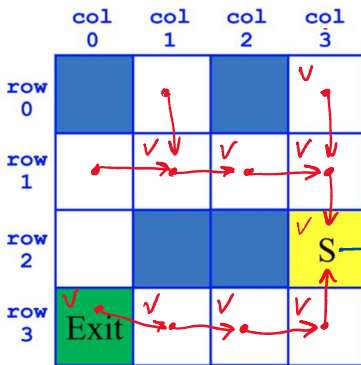
PA2 due Wed @ 8am

2D Search

Lecture 8

Breadth-First Search (BFS)

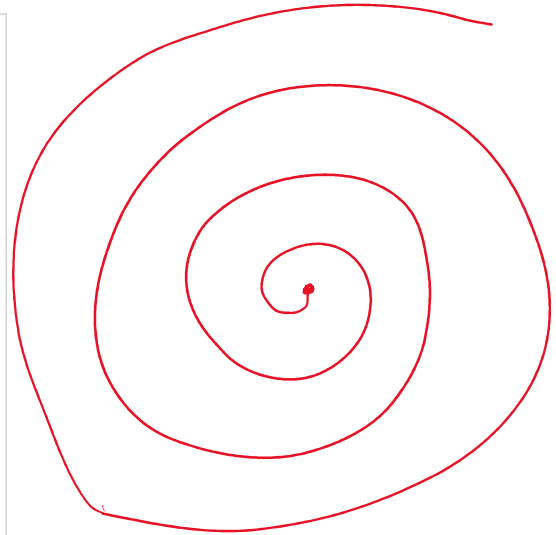
Guaranteed shortest path



SearchForTheExit

Initialize a Queue to hold Squares as we search
 Mark starting square as visited
 Enqueue starting square on Queue
 While Queue is not empty
 Dequeue square sq from Queue
 Mark sq as visited
 If sq is the Exit, we're done!
 For each of square's unvisited neighbors (S, W, N, E):
 Set neighbor's previous to sq
 Enqueue neighbor to Queue

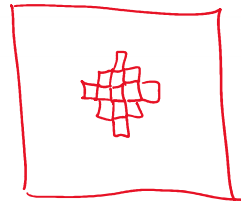
```
class Square {
    boolean visited;
    Square previous;
}
class Node {
    boolean visited;
    Node Node;
}
```



Run through the SearchForTheExit algorithm. Draw the queue.

front → ~~(2,2)~~ ~~(3,2)~~ ~~(1,3)~~ ~~(3,2)~~ ~~(1,2)~~ ~~(0,3)~~ ~~(3,1)~~ ~~(1,1)~~ (3,0) (1,0) (0,1)

(3,0) → (3,1) → (3,2) → (3,3) → (2,3) → null } stack to reverse
 exit start



How many nodes were visited? 9

How many total squares were added to the queue? 11

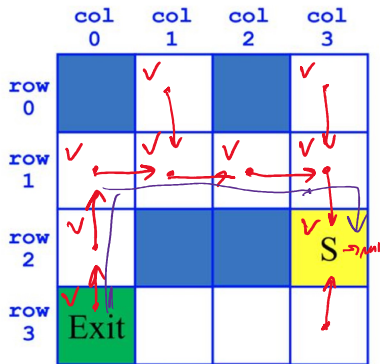
Was this the shortest path? Yes

Name: _____ PID: _____ Code: 6644

Depth-First Search (DFS)

Will always find a path. Possibly faster.

```
class Square {
    boolean visited;
    Square previous;
}
```



SearchForTheExit

Initialize a **Stack** to hold Squares as we search
Mark starting square as visited

Push starting square on **Stack**

While **Stack** is not empty

Pop square sq from **Stack**

 Mark sq as visited

 If sq is the Exit, we're done!

 For each of square's unvisited neighbors (S, W, N, E):

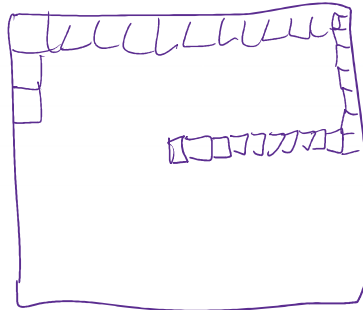
 Set neighbor's previous to sq

Push neighbor to **Stack**

Run through the SearchForTheExit algorithm. Draw the stack.

bottom → ~~(2,3)~~ ~~(3,3)~~ ~~(1,3)~~ ~~(1,2)~~ ~~(0,3)~~ ~~(1,1)~~ ~~(1,0)~~ ~~(0,1)~~ = top

~~(2,0)~~ ~~(3,0)~~ Exit



How many nodes were visited? 9

How many total squares were added to the stack? 10

Was this the shortest path? No

A*

Dijkstra's