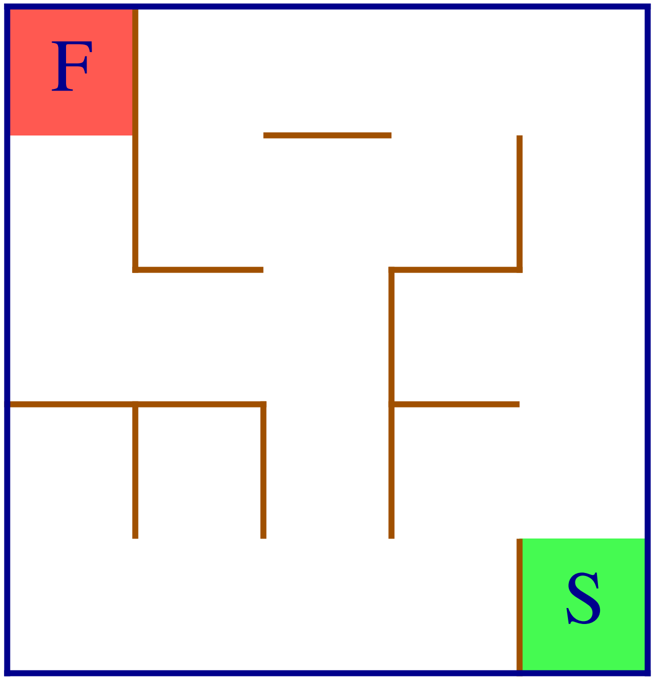
CS261 Data Structures

DFS and BFS

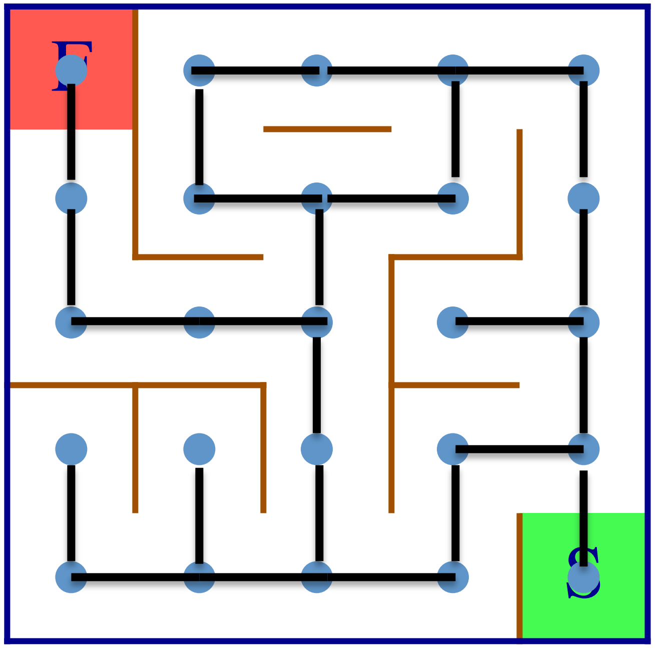
Application: Maze Path Finding

- Find a path from start to finish in a maze:
  - – Easily represent a maze as a graph
  - – Compute single source (S) reachability, stopping when get to F



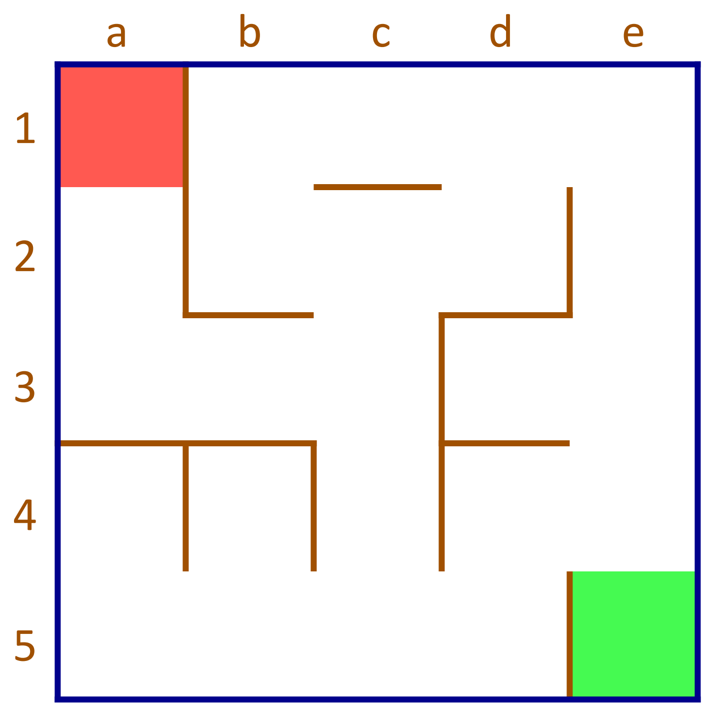
Application: Maze Path Finding

- Find a path from start to finish in a maze:
  - Easily represent a maze as a graph

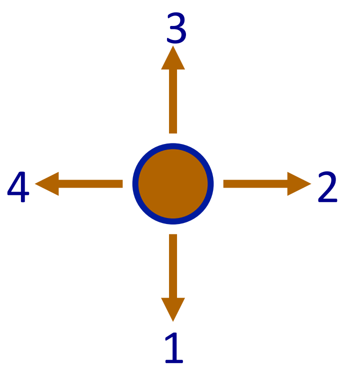


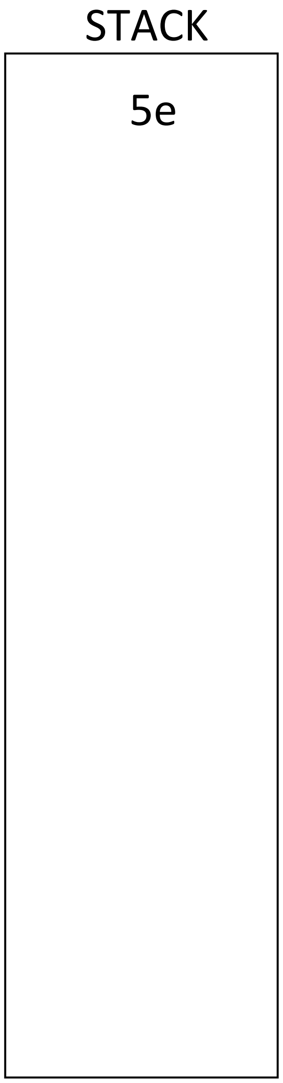
Application: Maze Path Finding Example

- Single-Source Reachability

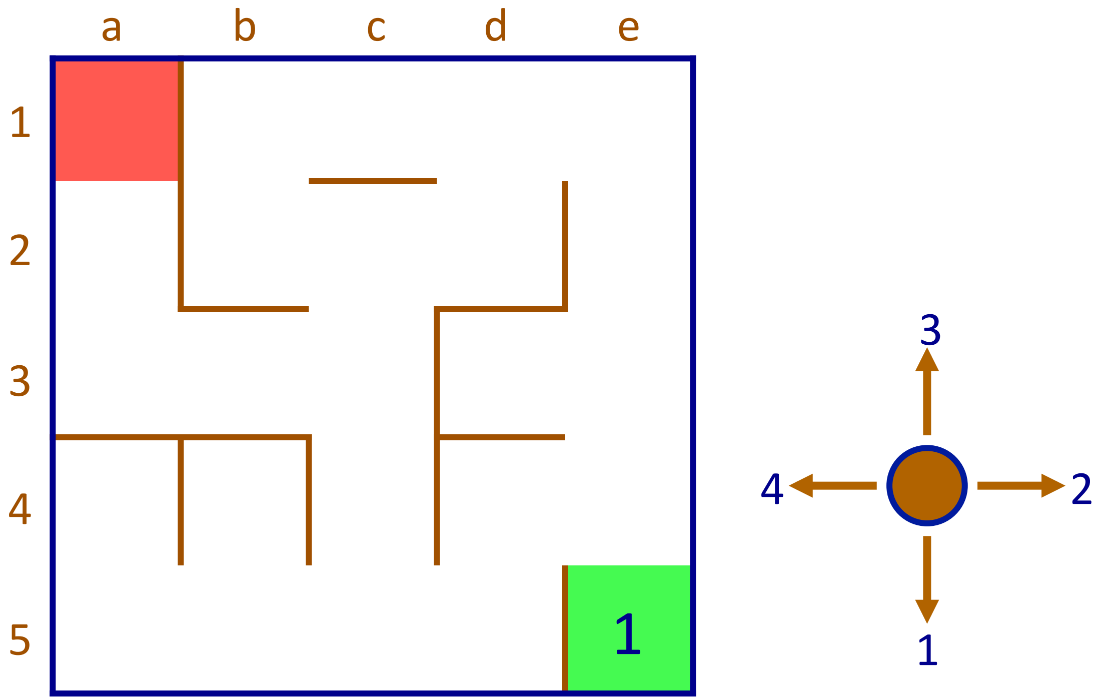

a
b
c
d
e
1
2
3
4
5

For consistency (order in which neighbors are **pushed** onto the stack)





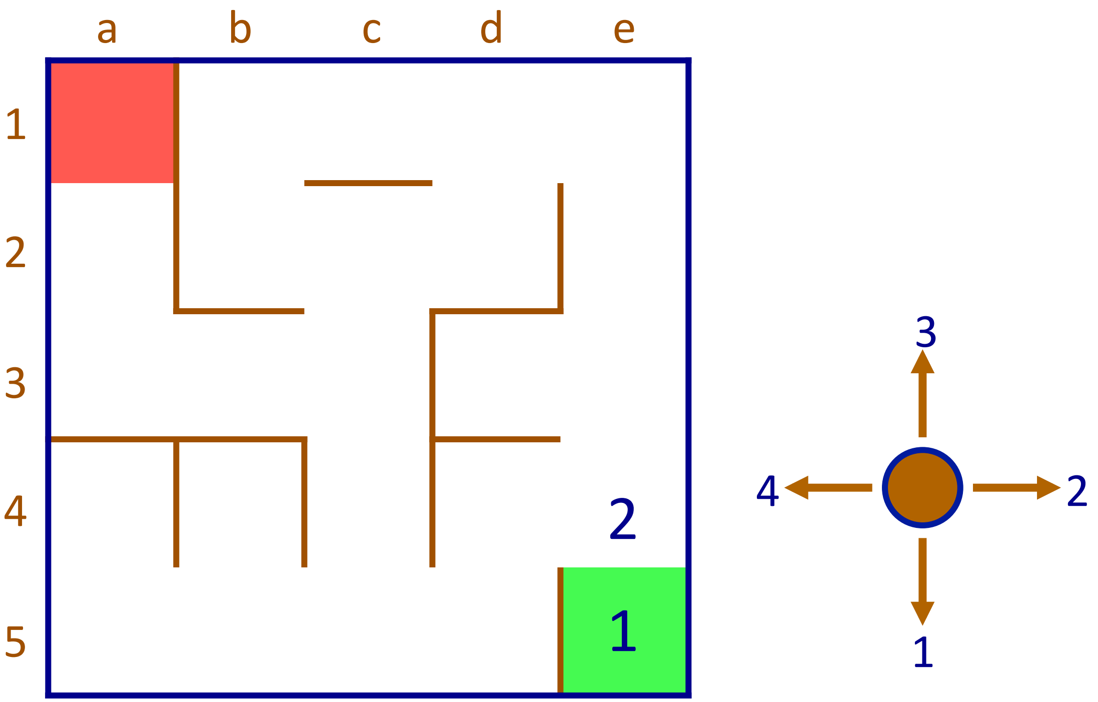
Application: Maze Path Finding Example


a
b
c
d
e
1
2
3
4
5
1
3
4
2
1

STACK

4e

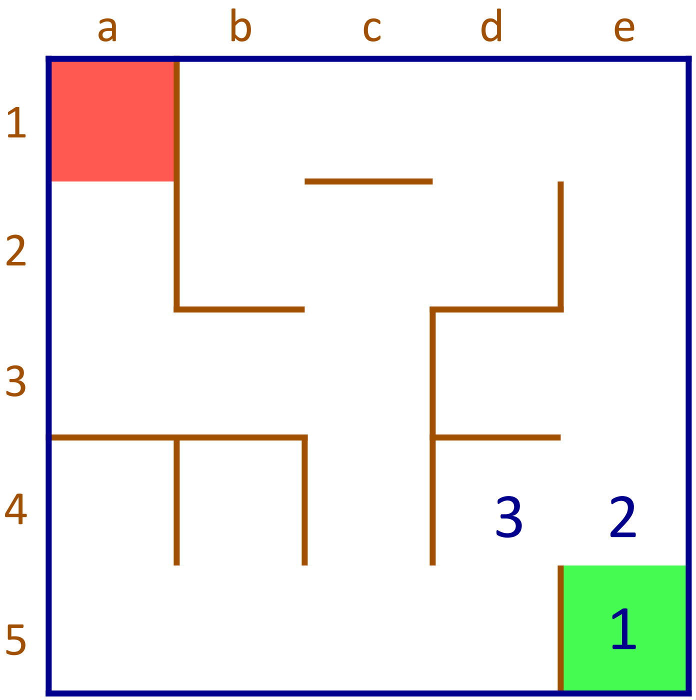
Application: Maze Path Finding Example

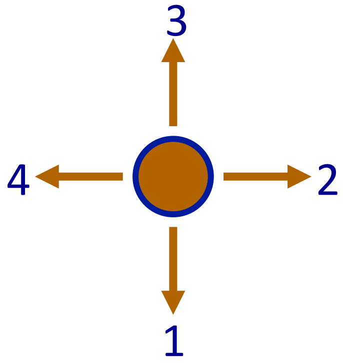

a
b
c
d
e
1
2
3
4
5
2
1
3
4
2
1

STACK

4d
3e

Application: Maze Path Finding Example

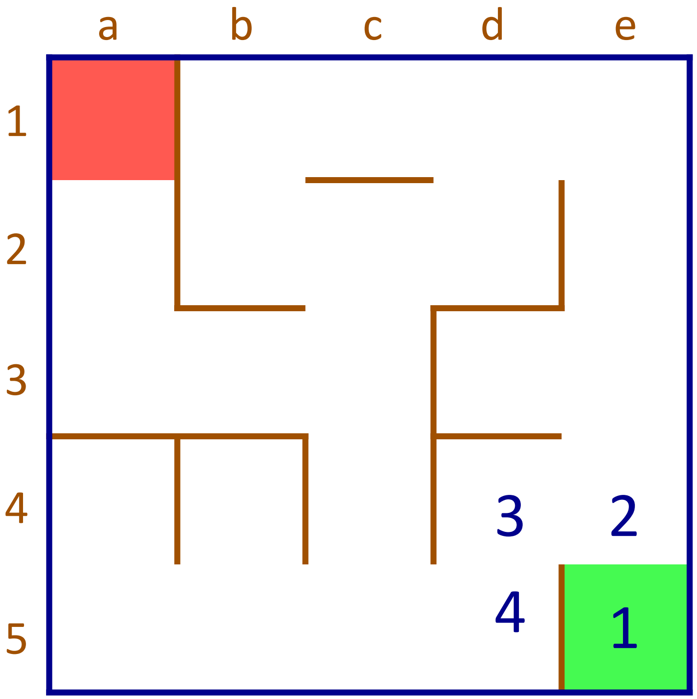

a
b
c
d
e
1
2
3
4
5
3
2
1

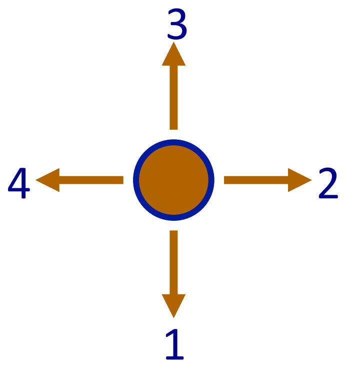


STACK

5d
3e

Application: Maze Path Finding Example

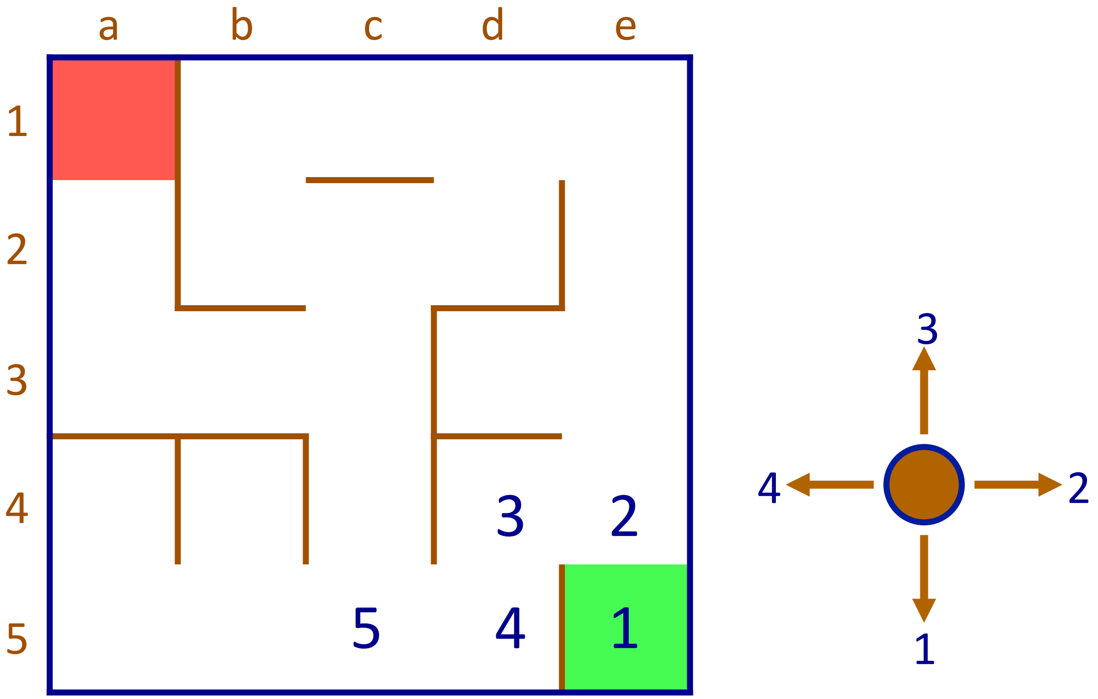

a
b
c
d
e
1
2
3
4
5
3
2
4
1



STACK

5c
3e

Application: Maze Path Finding Example

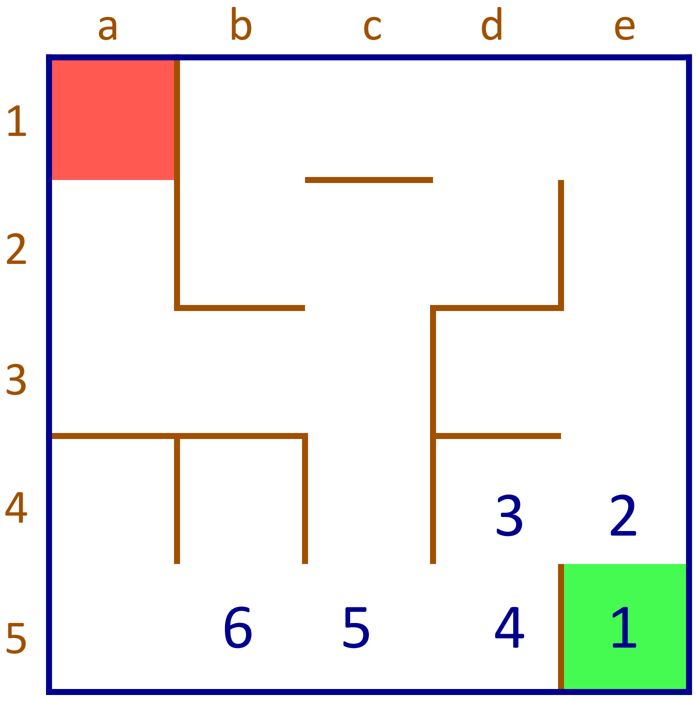


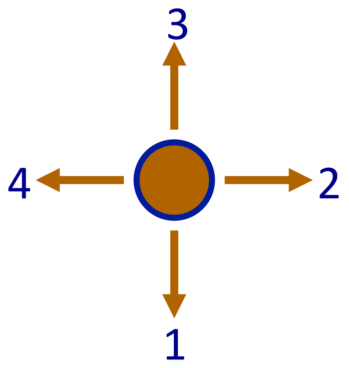
a
b
c
d
e
1
2
3
4
5
3
2
5
4
1
3
4
2
1

STACK

5b
4c
3e

Application: Maze Path Finding Example

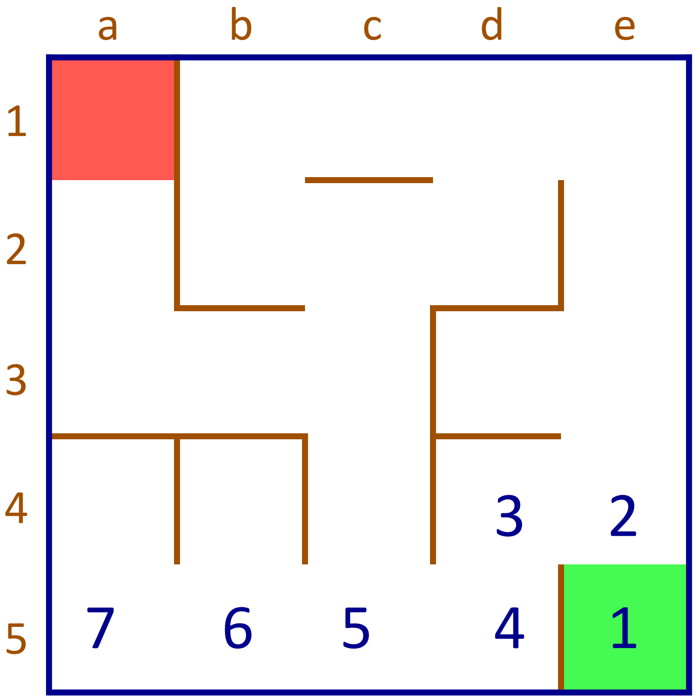

a
b
c
d
e
1
2
3
4
5
3
2
6
5
4
1

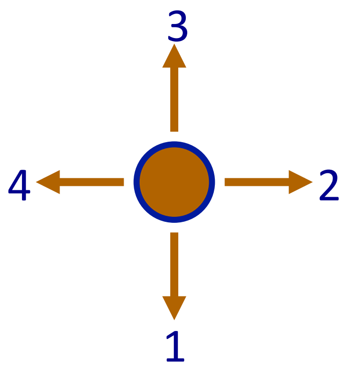


STACK

5a
4b
4c
3e

Application: Maze Path Finding Example

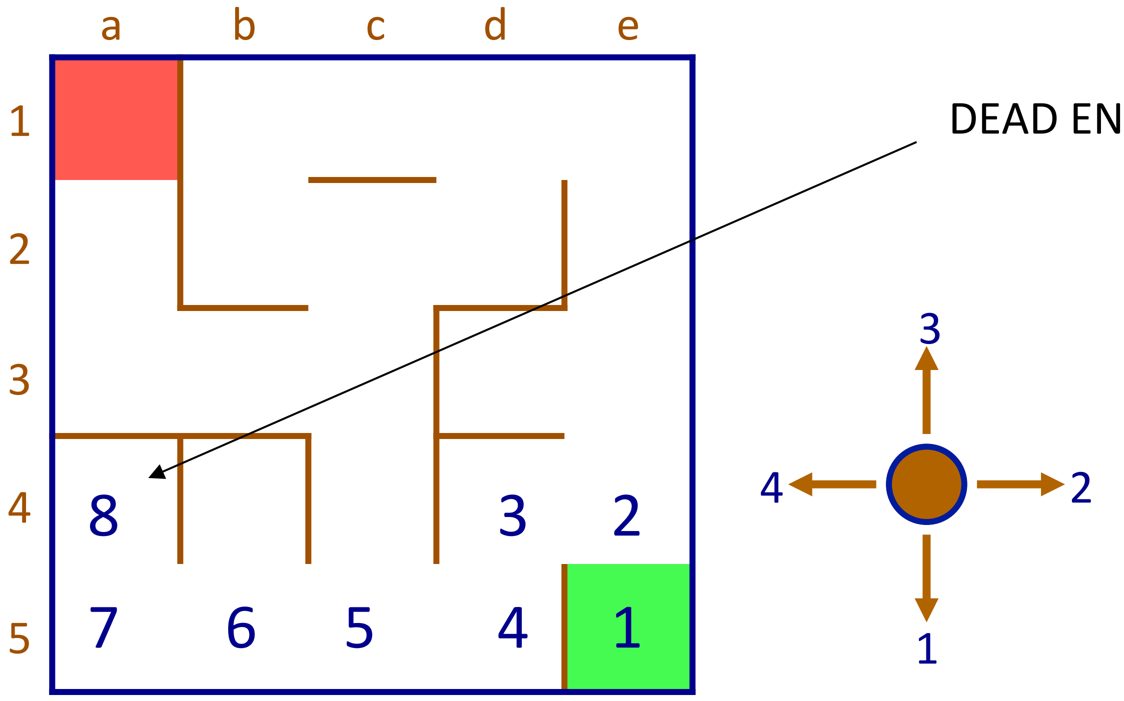

a
b
c
d
e
1
2
3
4
5
3
2
7
6
5
4
1



STACK

4a
4b
4c
3e

Application: Maze Path Finding Example

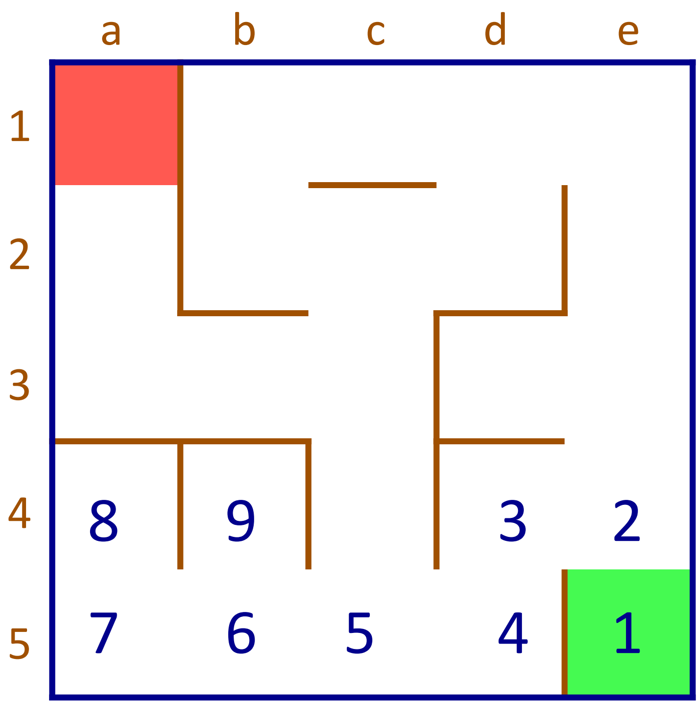

a
b
c
d
e
1
2
3
4
5
8
3
2
7
6
5
4
1
3
4
2
1

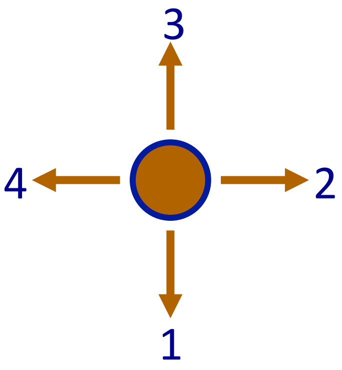
DEAD END!!

STACK

4b
4c
3e

Application: Maze Path Finding Example

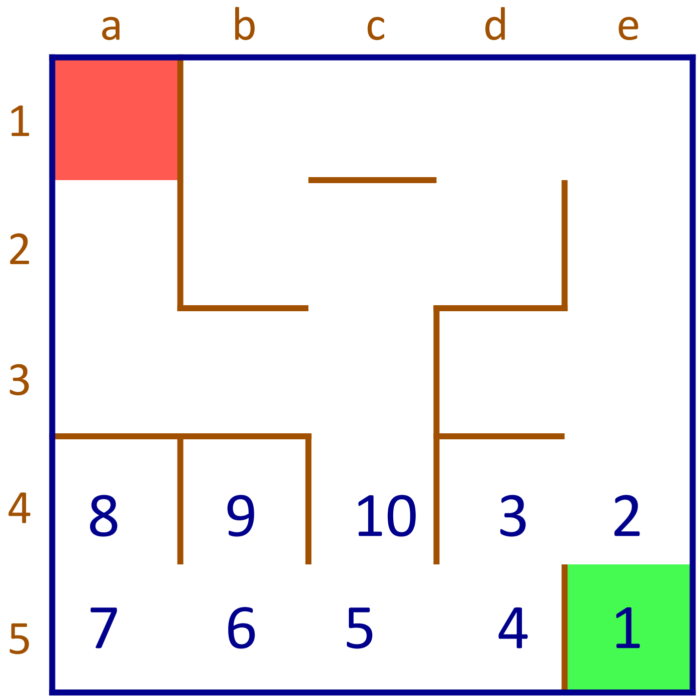

a
b
c
d
e
1
2
3
4
5
8
9
3
2
7
6
5
4
1

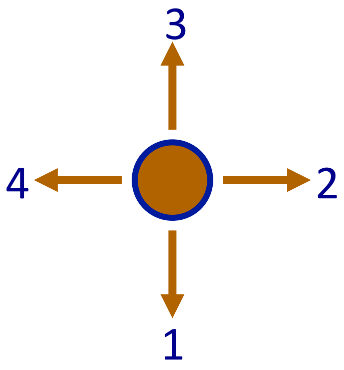


STACK

4c
3e

Application: Maze Path Finding Example

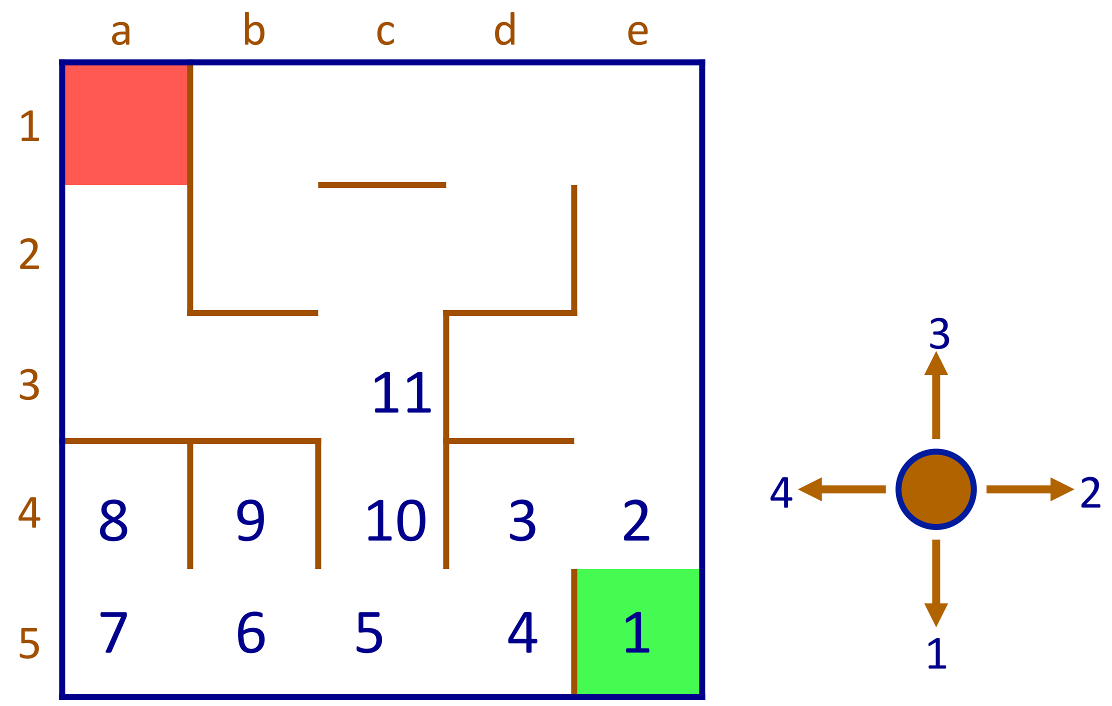

a
b
c
d
e
1
2
3
4
5
8
9
10
3
2
7
6
5
4
1



STACK

3c
3e

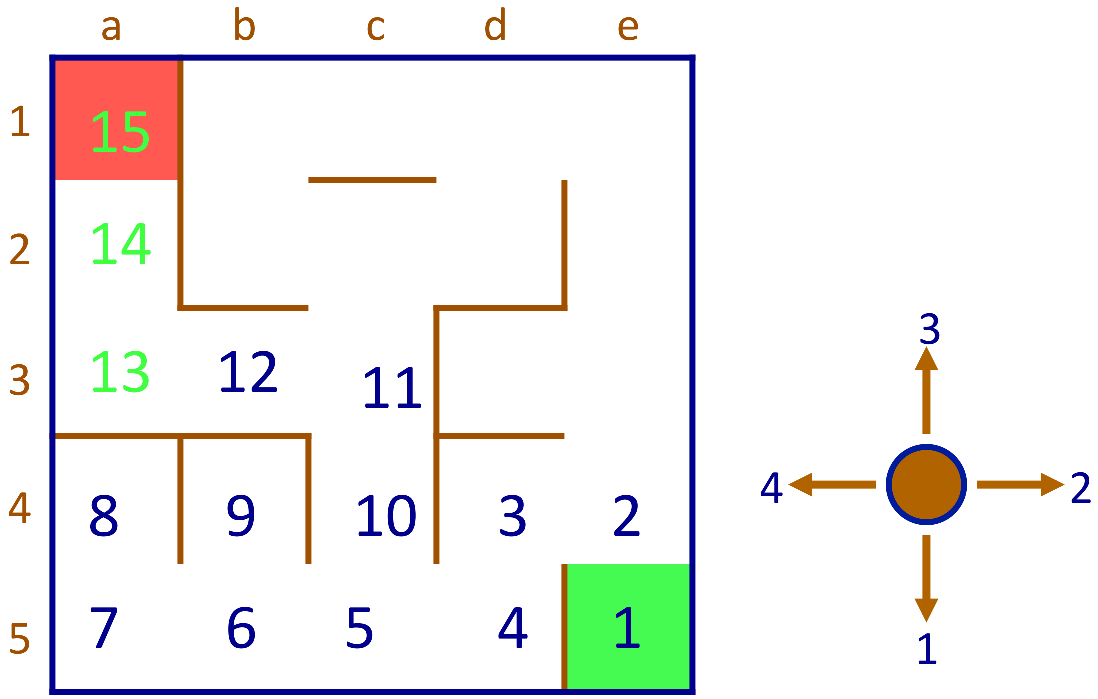
Application: Maze Path Finding Example


a
b
c
d
e
1
2
3
4
5
11
8
9
10
3
2
7
6
5
4
1
3
4
2
1

STACK

3b
2c
3e

Application: Maze Path Finding Example

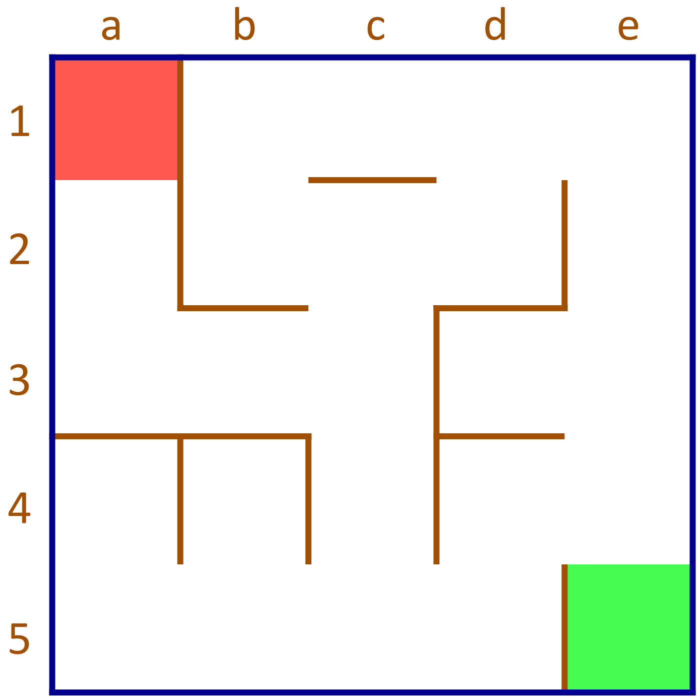

a
b
c
d
e
1
2
3
4
5
15
14
13
12
11
8
9
10
3
2
7
6
5
4
1
3
4
2
1

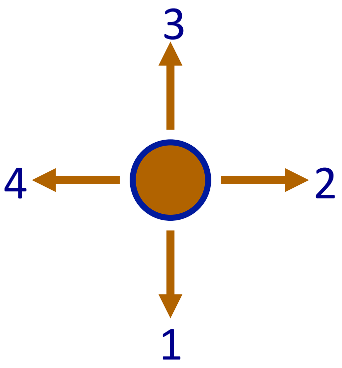
STACK

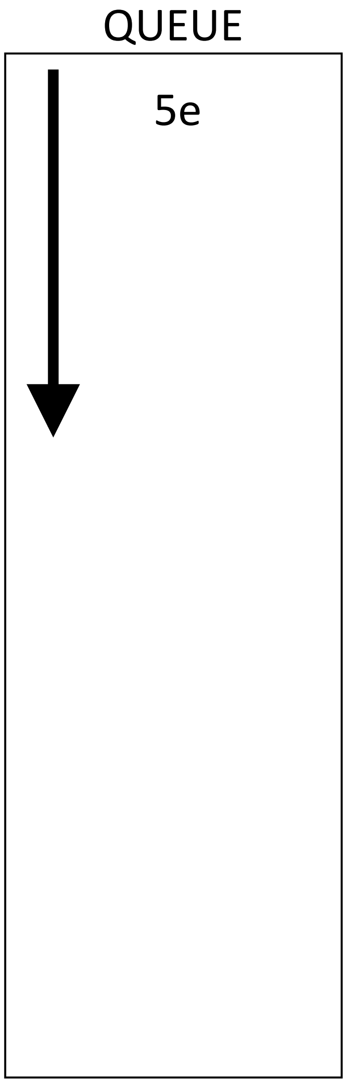
2c
3e

What happens if we use a Queue?

Application: Maze Path Finding Example

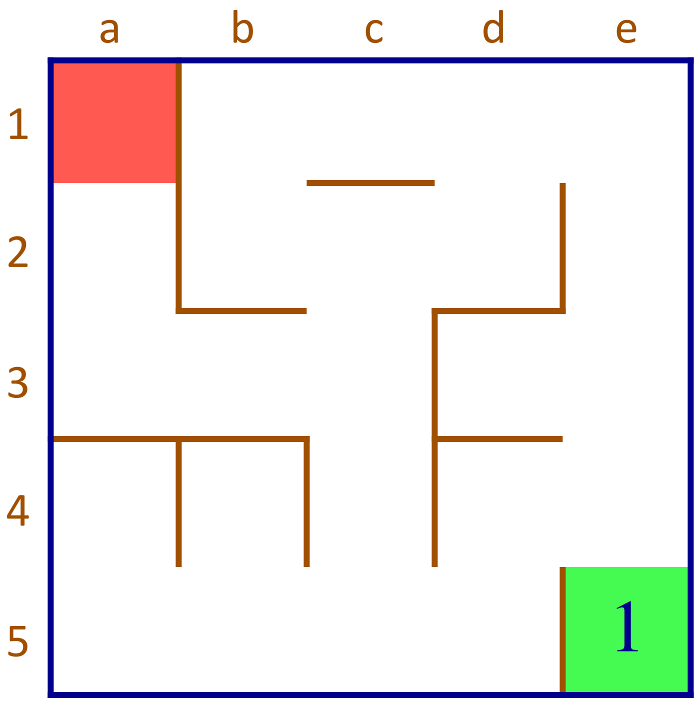

a
b
c
d
e
1
2
3
4
5

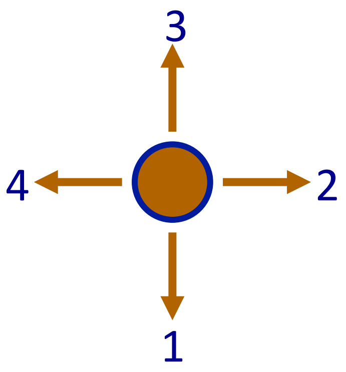


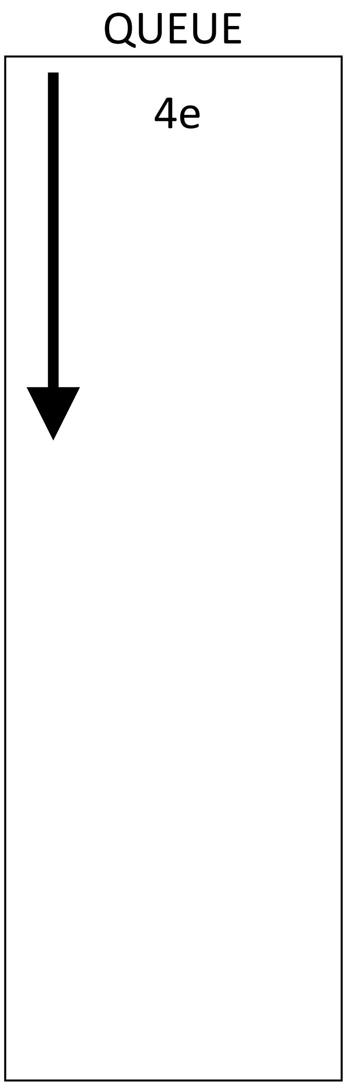

QUEUE

5e

Application: Maze Path Finding Example


a
b
c
d
e
1
2
3
4
5
1

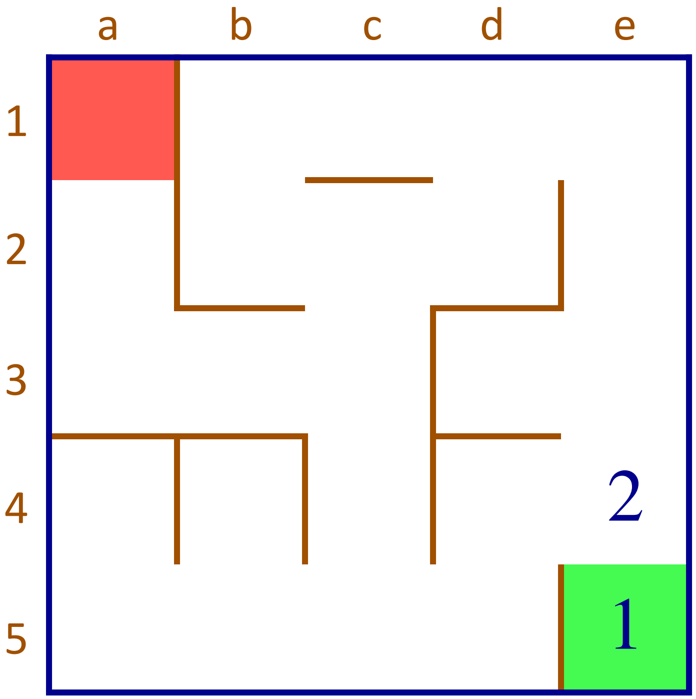


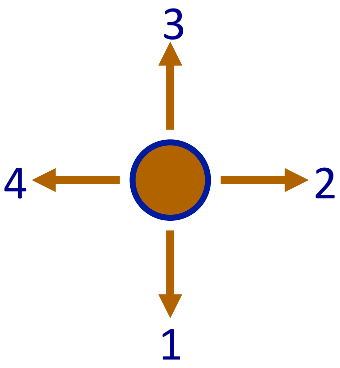


QUEUE

4e

Application: Maze Path Finding Example

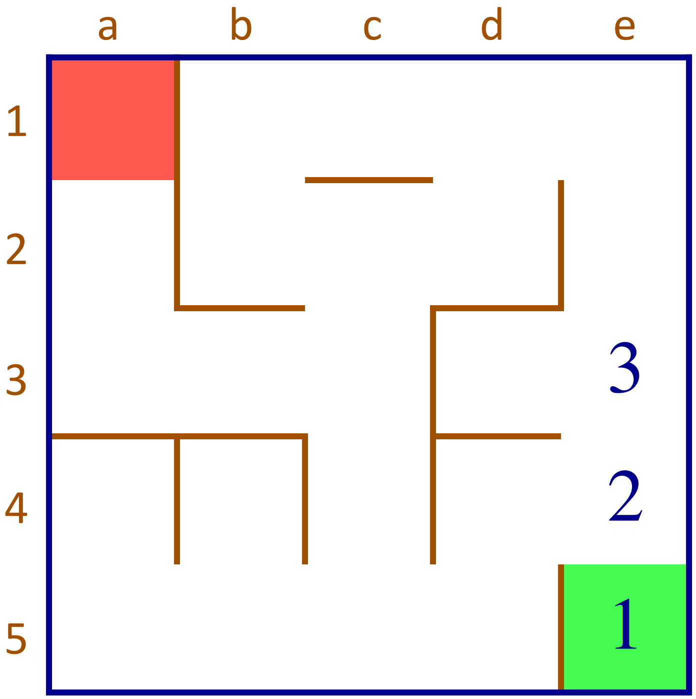

a
b
c
d
e
1
2
3
4
5
2
1

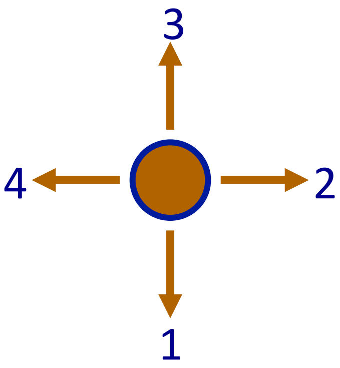


QUEUE

4d
3e

Application: Maze Path Finding Example

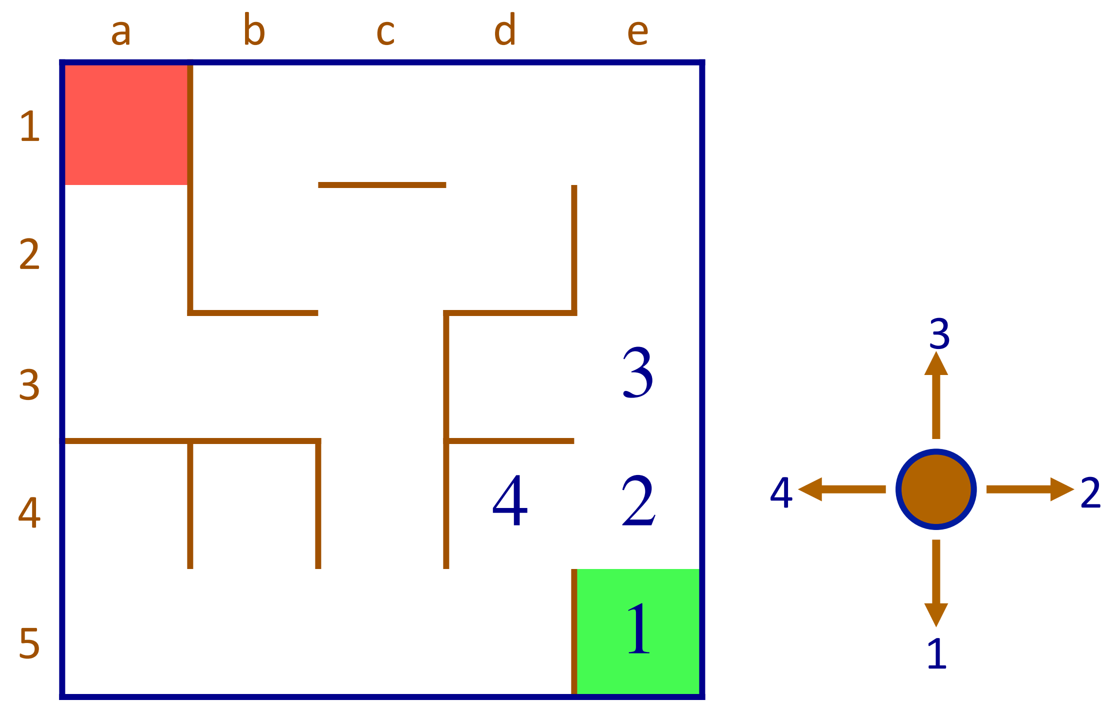

a
b
c
d
e
1
2
3
4
5
3
2
1



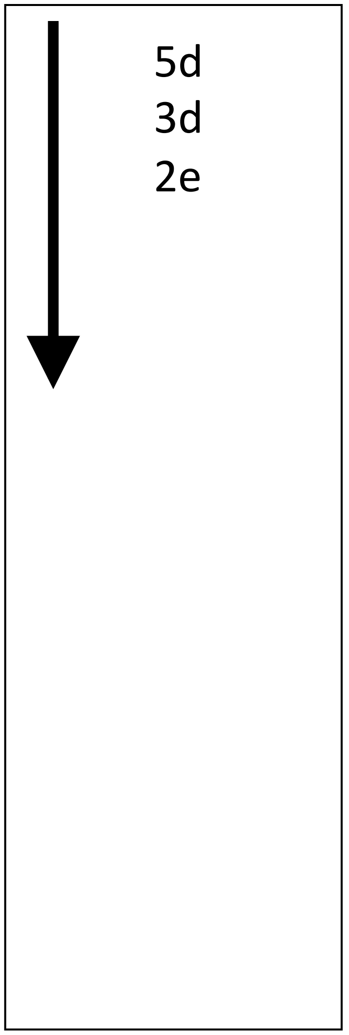
QUEUE

3d
2e
4d

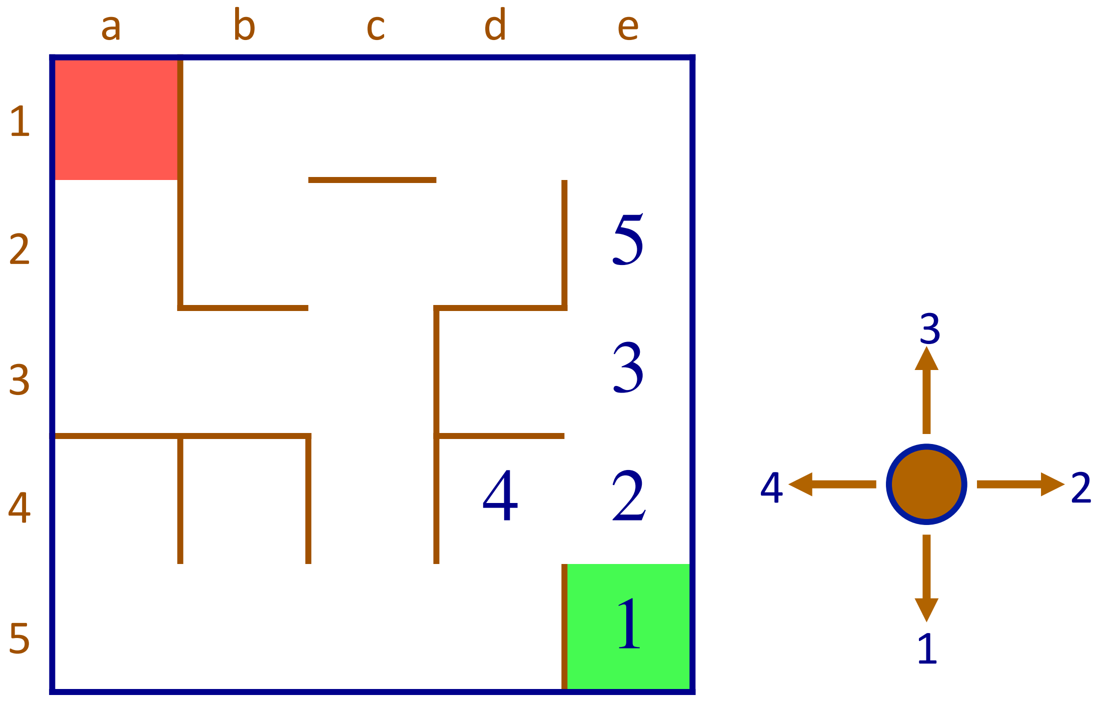
Application: Maze Path Finding Example


a
b
c
d
e
1
2
3
4
5
3
4
2
1
3
4
2
1

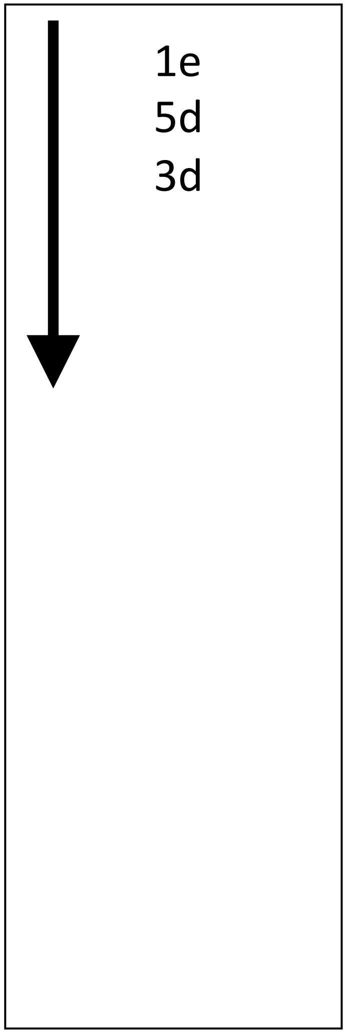
QUEUE



Application: Maze Path Finding Example

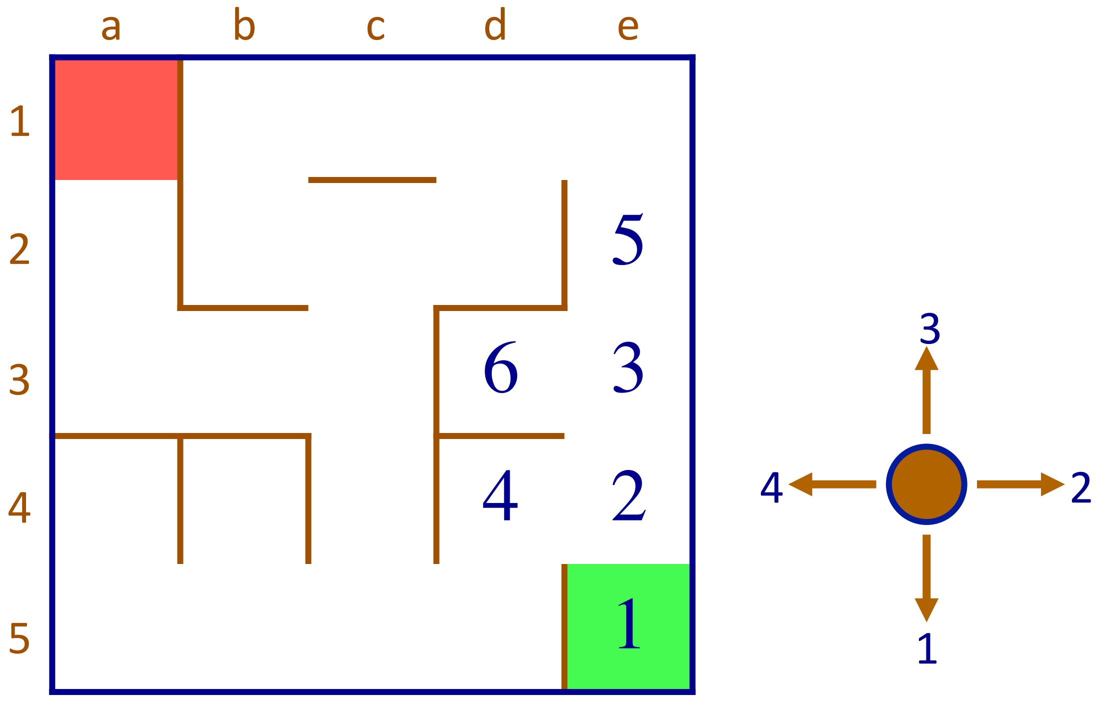

a
b
c
d
e
1
2
3
4
5
5
3
4
2
1
3
4
2
1

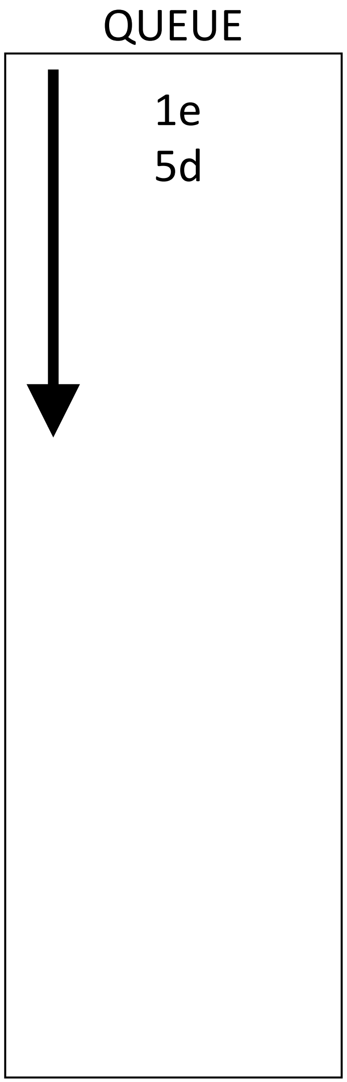
QUEUE



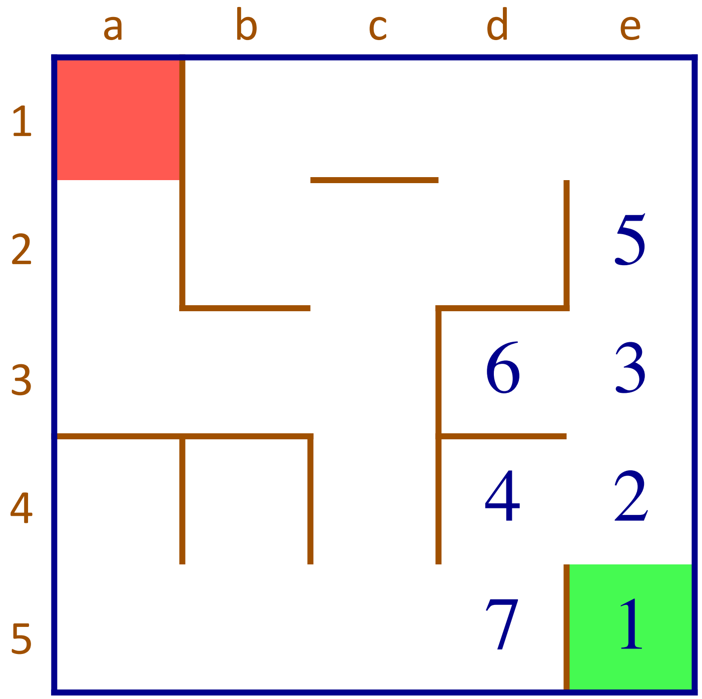
1e
5d
3d

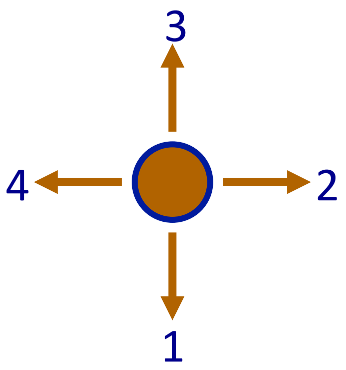
Application: Maze Path Finding Example


a
b
c
d
e
1
2
3
4
5
5
6
3
4
2
1
3
4
2
1



Application: Maze Path Finding Example

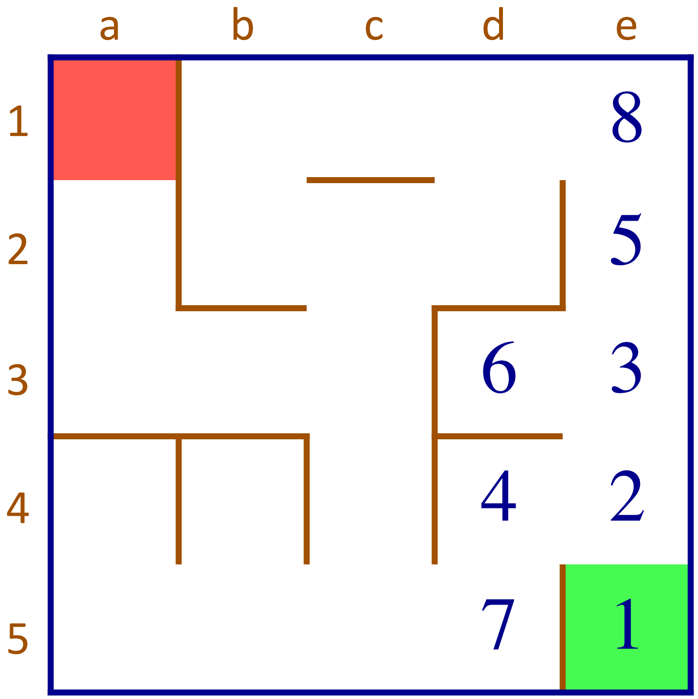

a
b
c
d
e
1
2
3
4
5
5
6
3
4
2
7
1

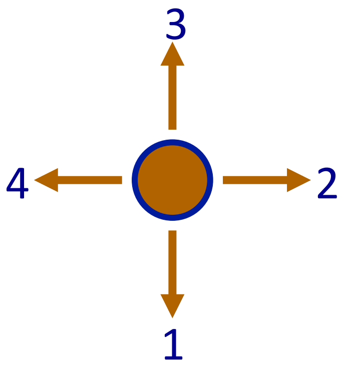


QUEUE

5c
1e

Application: Maze Path Finding Example

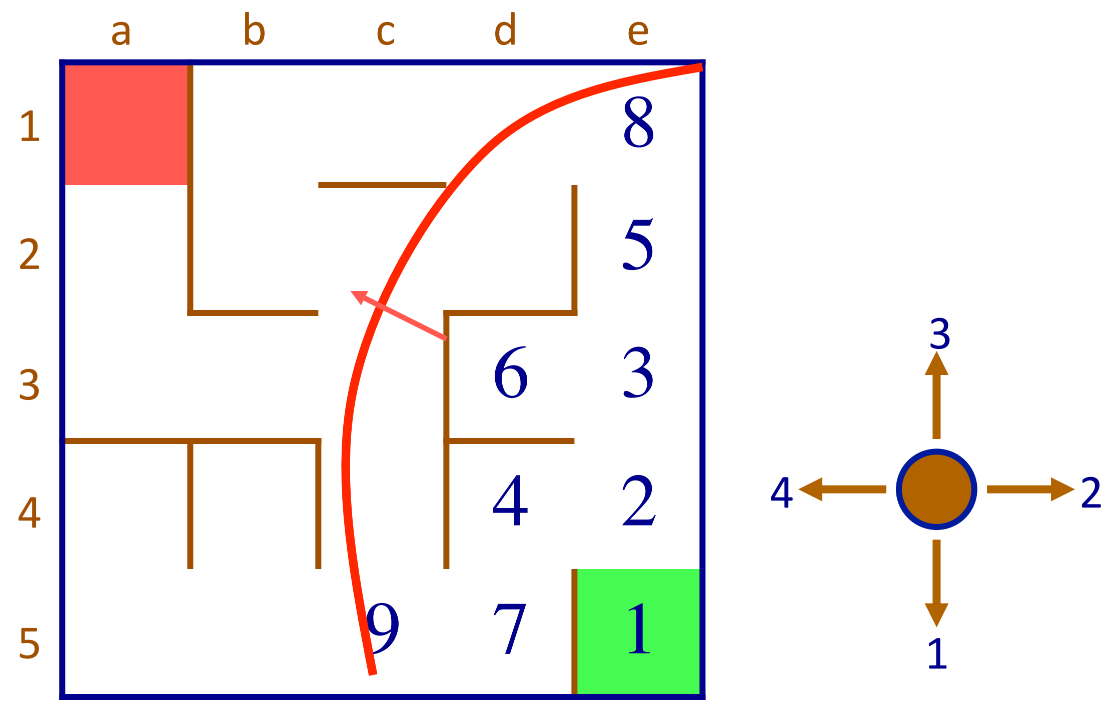

a
b
c
d
e
1
2
3
4
5
8
5
6
3
4
2
7
1



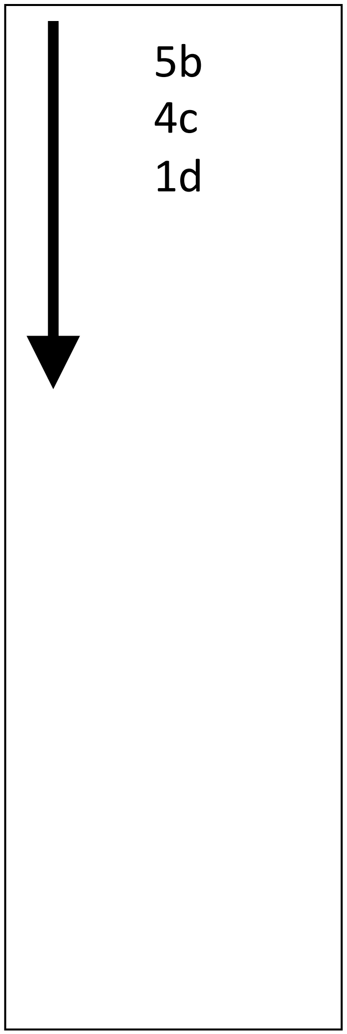
QUEUE

1d
5c

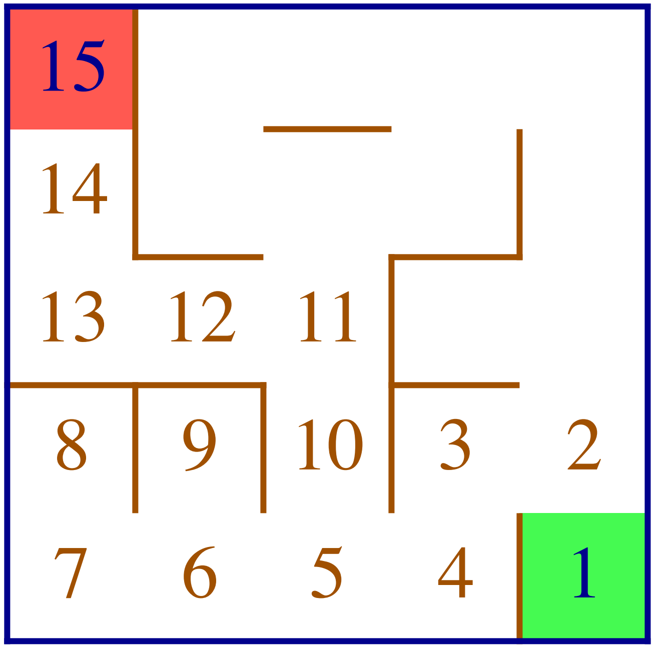
Application: Maze Path Finding Example


a
b
c
d
e
1
2
3
4
5
8
5
6
3
4
2
9
7
1
3
4
2
1

QUEUE

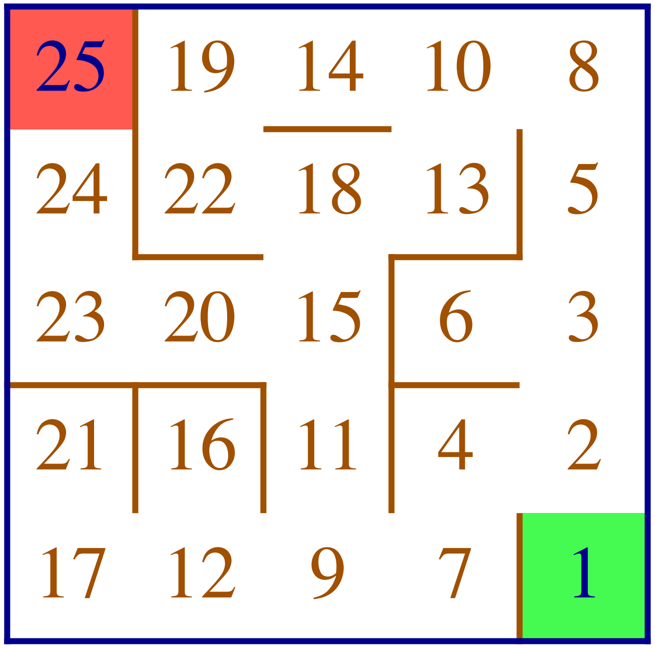


Application: Maze Path Finding



15				
14				
13	12	11		
8	9	10	3	2
7	6	5	4	1

Depth-First (**Stack**)



25	19	14	10	8
24	22	18	13	5
23	20	15	6	3
21	16	11	4	2
17	12	9	7	1

Breadth-First (**Queue**)

DFS vs. BFS

- DFS like a single person working a maze
- BFS like a wave flowing through a maze
- DFS can take an unfortunate route and have to backtrack a long way, and multiple times
- DFS can get lucky and find the solution very quickly
- BFS may not find it as quickly, but will ***always*** find it
- Because BFS first checks all paths of length 1, then of length 2, then of length 3, etc….it's guaranteed to find a path containing the least steps from start to goal (if it exists)
- What if there's one infinite path….DFS may go down it…but BFS will not get stuck in it

Time Complexity: DFS/BFS

- O(V+E) time in both cases
  - Key observation: Edge list scanned once for each vertex, so scans E edges

```
Initialize set of reachable vertices and add $v_i$ to a stack
While stack is not empty
    Get and remove (pop) last vertex $v$ from stack
    if vertex v is not in reachable,
        add it to reachable
        For all neighbors, $v_j$, of $v$, if $v_j$ is NOT in reachable
            add to stack
```

Space Complexithy: DFS/BFS

- What about space?
  - BFS must store all vertices on a Queue at most once
  - DFS uses a Stack and stores all vertices on the stack at most once
  - In both cases, O(V) space worst case
  - In practice, BFS may take up more space because it looks at all paths of a specific length at once. e.g. if search a deep tree , BFS will store lots of long potential paths

DFS vs. BFS :In practice

- Depends on the problem
  - What is the graph/tree like (deep, wide?)
  - Are solutions rare or frequent?
  - If solution is close to the root, BFS might be better and will find it quickly
  - If solution is rare and may be deep, DFS might look a long time
  - If solutions are frequent but deep in tree, BFS may take too much memory!