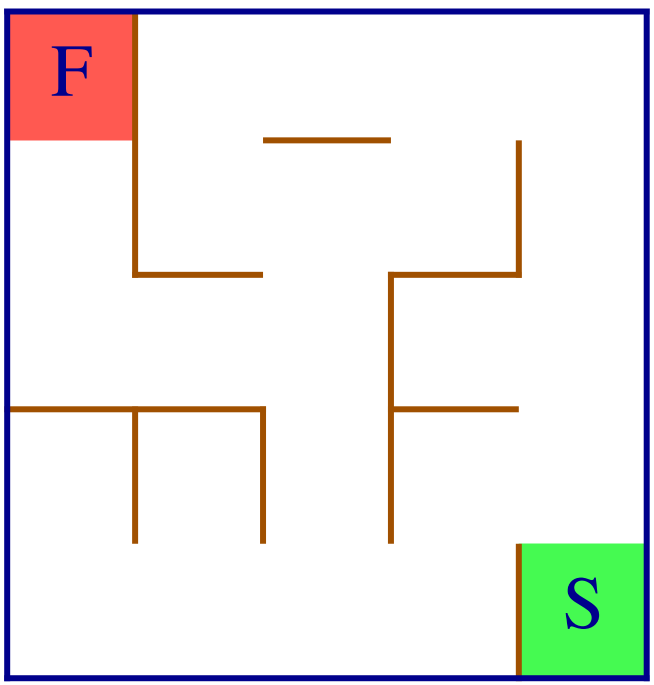
DFS and BFS – Edge List Representation

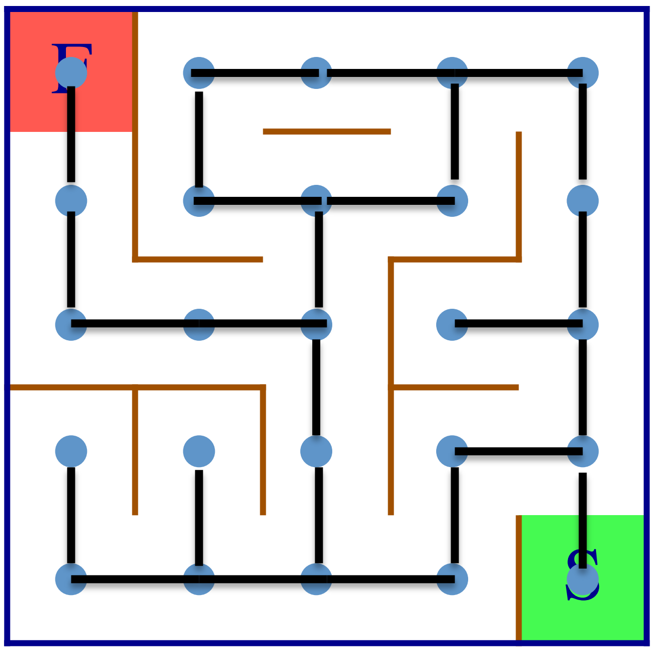
Application: Maze Path Finding

- 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



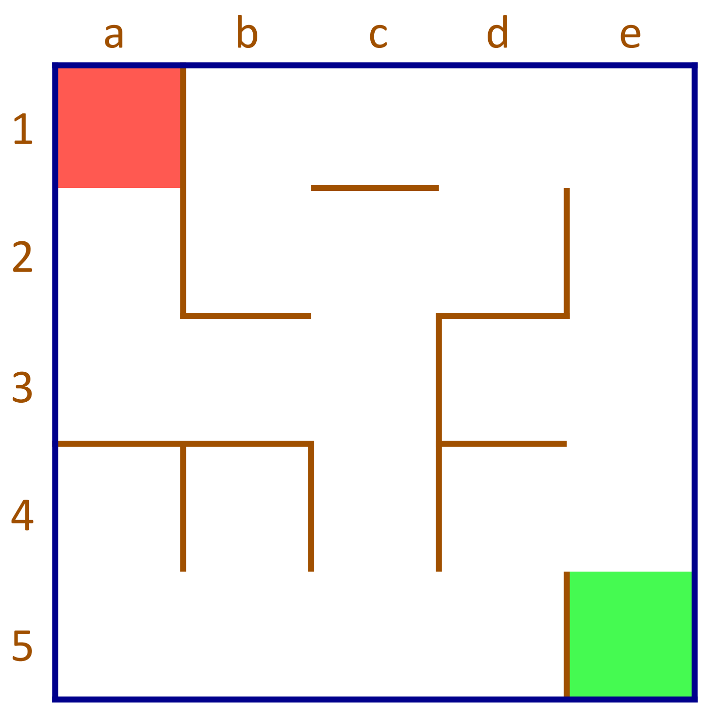
Application: Maze Path Finding

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

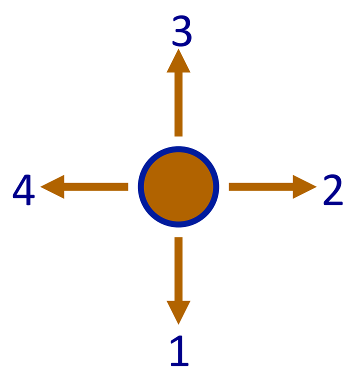


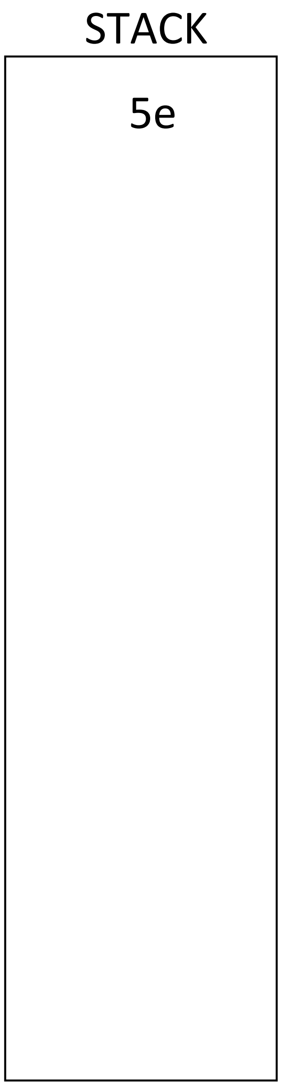
Application: Maze Path Finding Example

- Single-Source Reachability

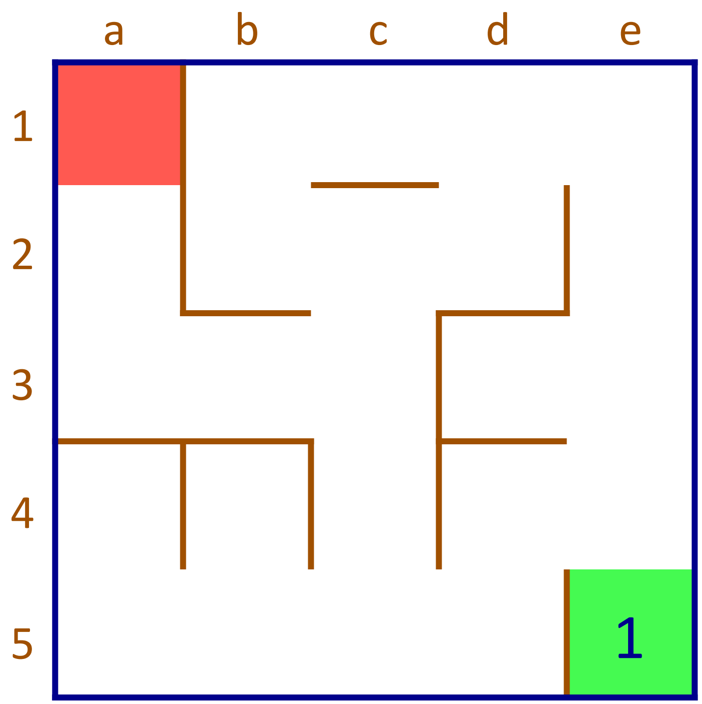


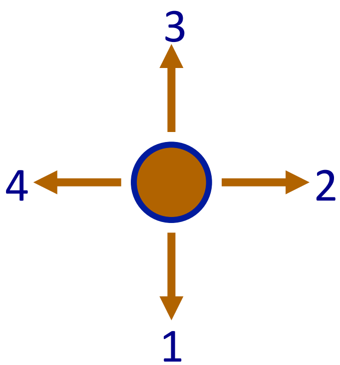
For consistency (order in which neighbors are **pushed** onto the stack)





Application: Maze Path Finding Example

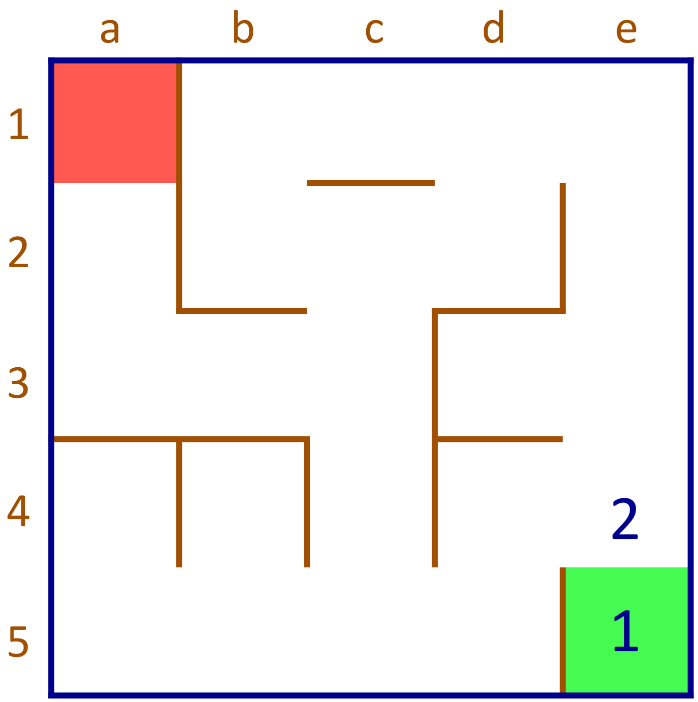


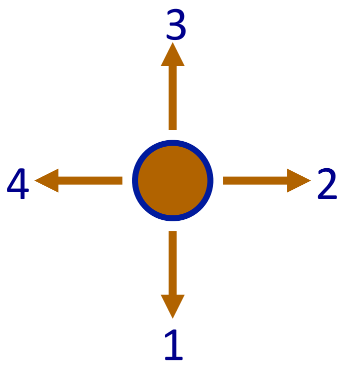


STACK

4e

Application: Maze Path Finding Example

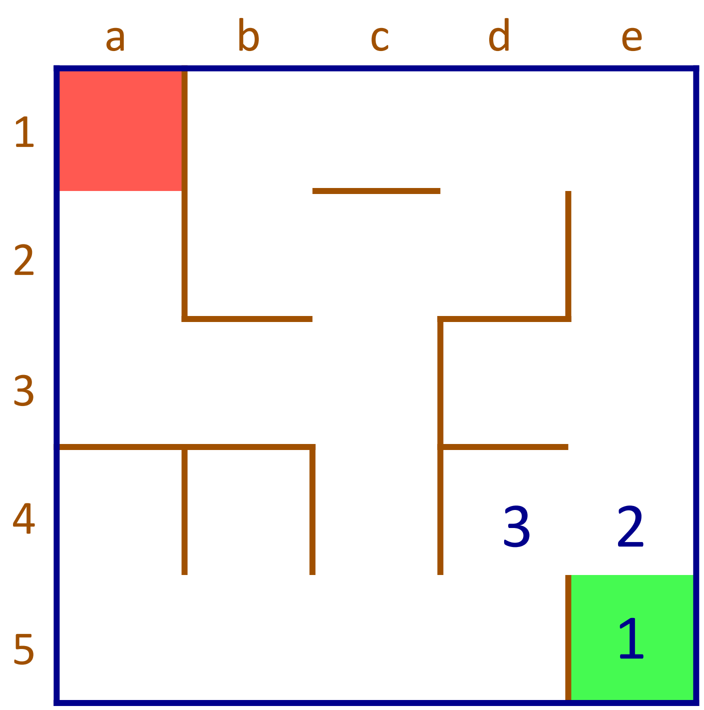


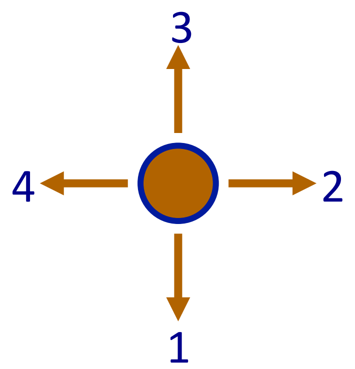


STACK

4d
3e

Application: Maze Path Finding Example

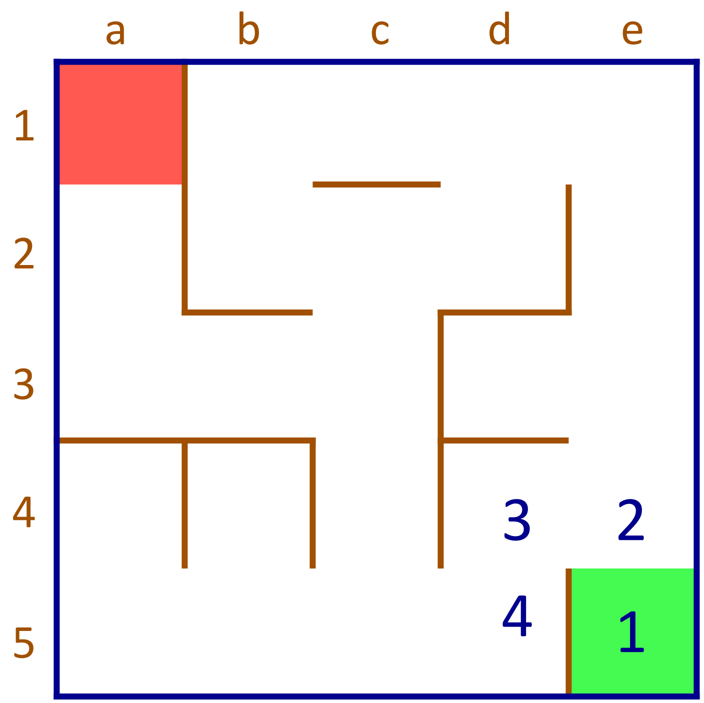


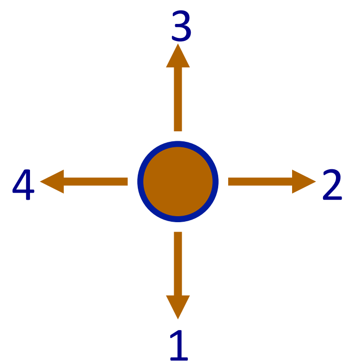


STACK

5d
3e

Application: Maze Path Finding Example

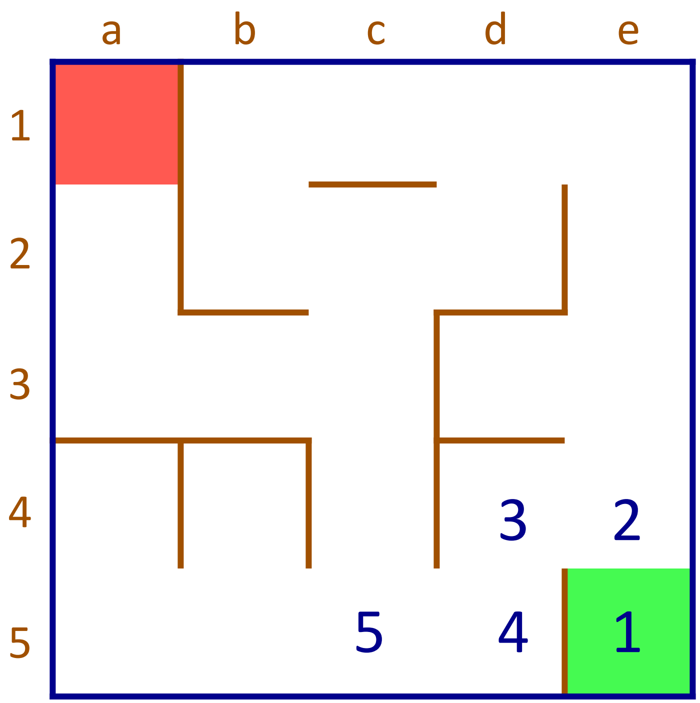


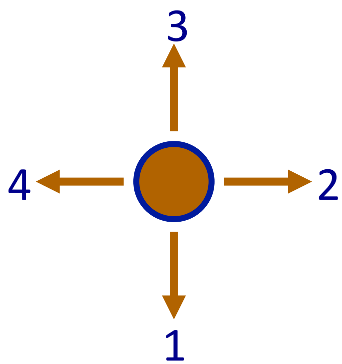


STACK

5c
3e

Application: Maze Path Finding Example

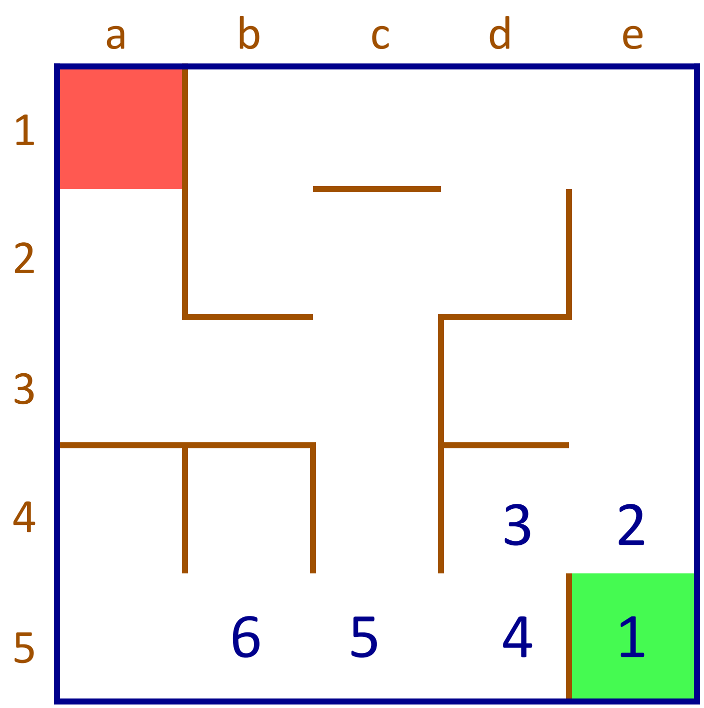


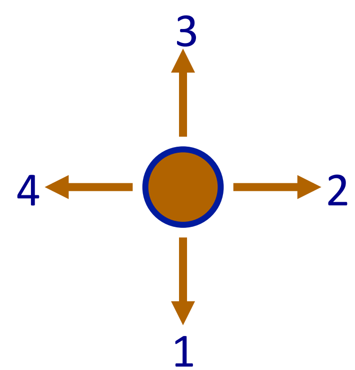


STACK

5b
4c
3e

Application: Maze Path Finding Example

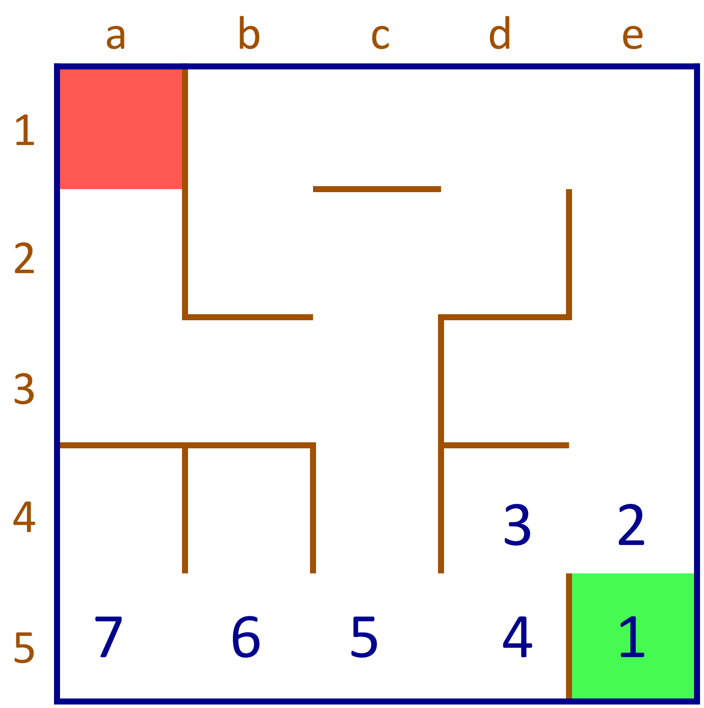


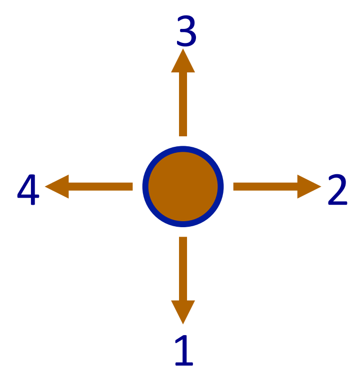


STACK

5a
4b
4c
3e

Application: Maze Path Finding Example

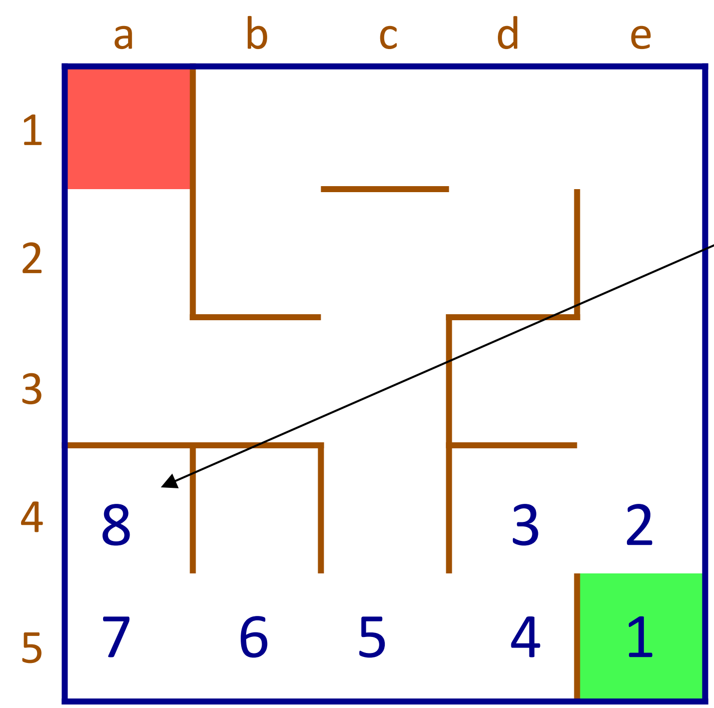


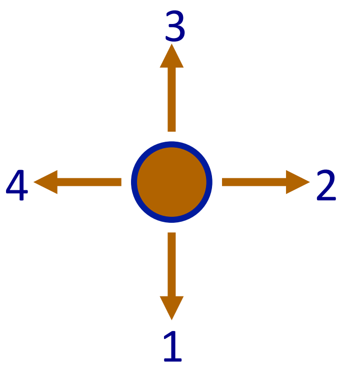


STACK

4a
4b
4c
3e

Application: Maze Path Finding Example

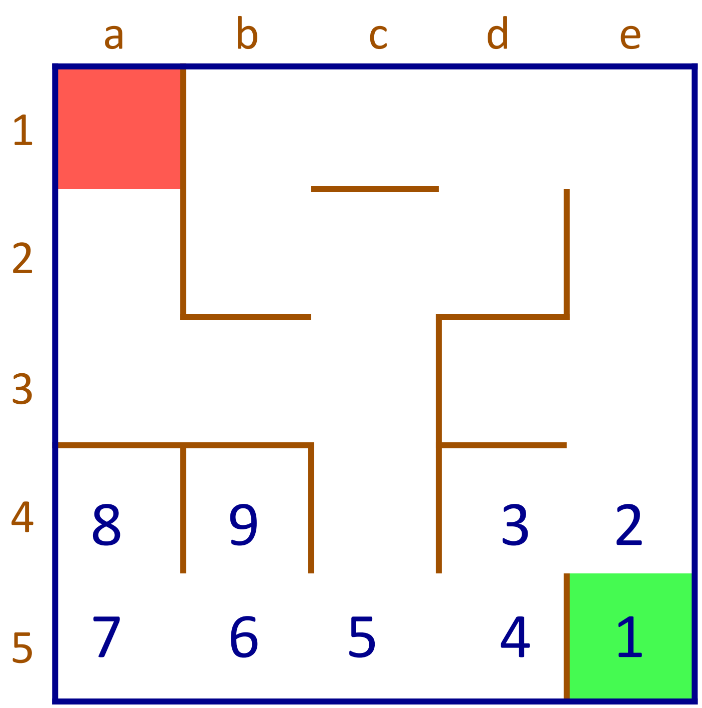


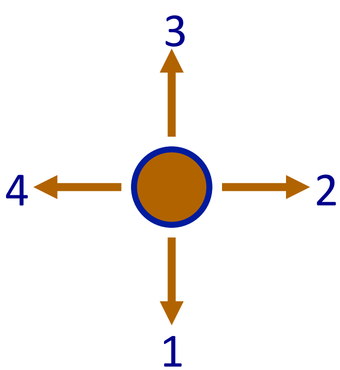


STACK

4b
4c
3e

Application: Maze Path Finding Example

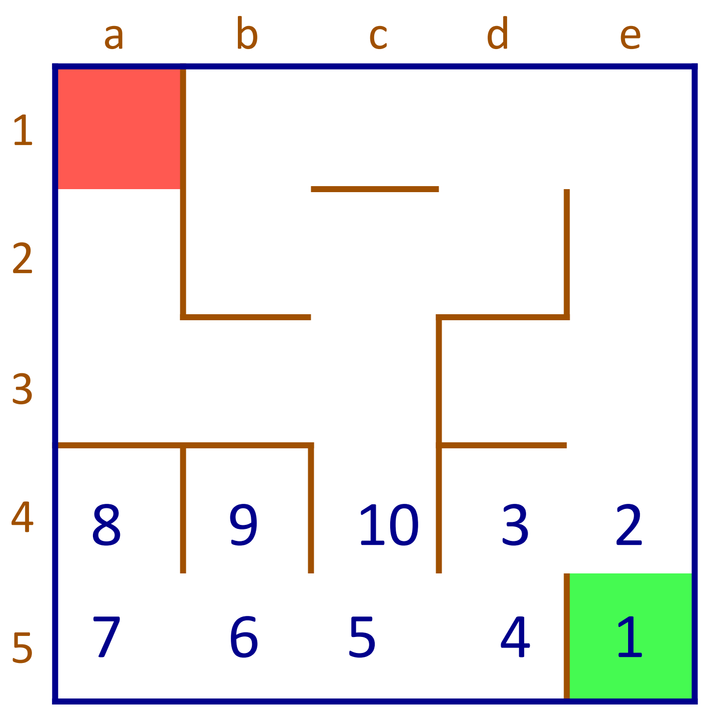


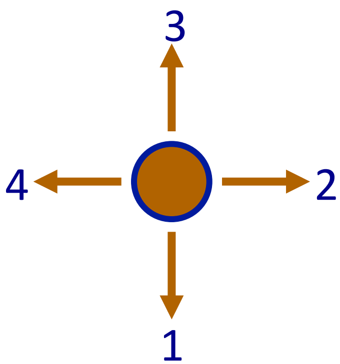


STACK

4c
3e

Application: Maze Path Finding Example

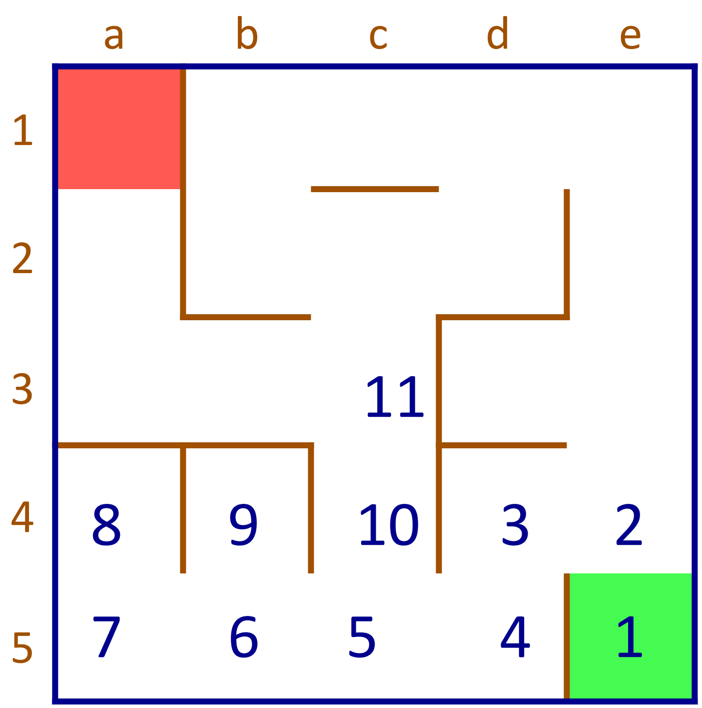


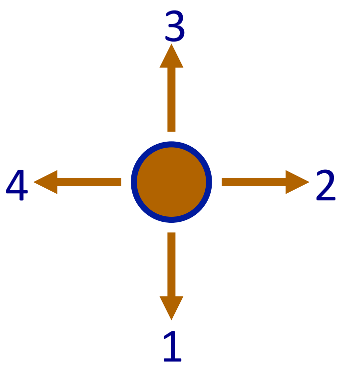


STACK

3c
3e

Application: Maze Path Finding Example

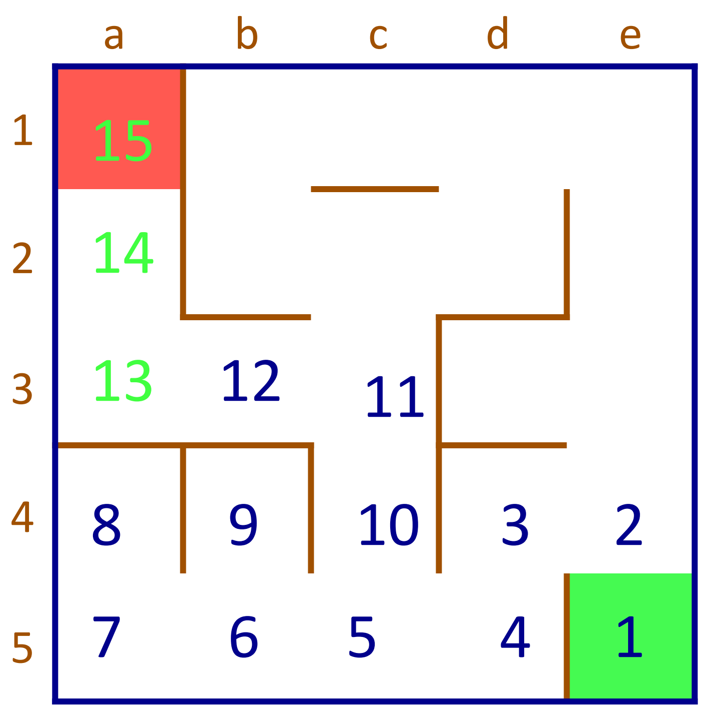


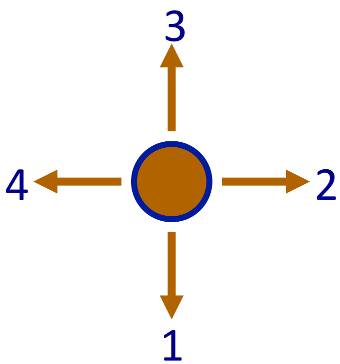


STACK

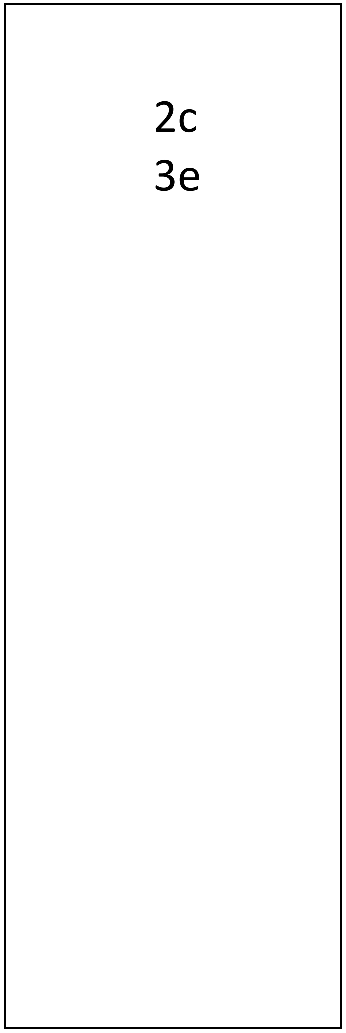
3b
2c
3e

Application: Maze Path Finding Example



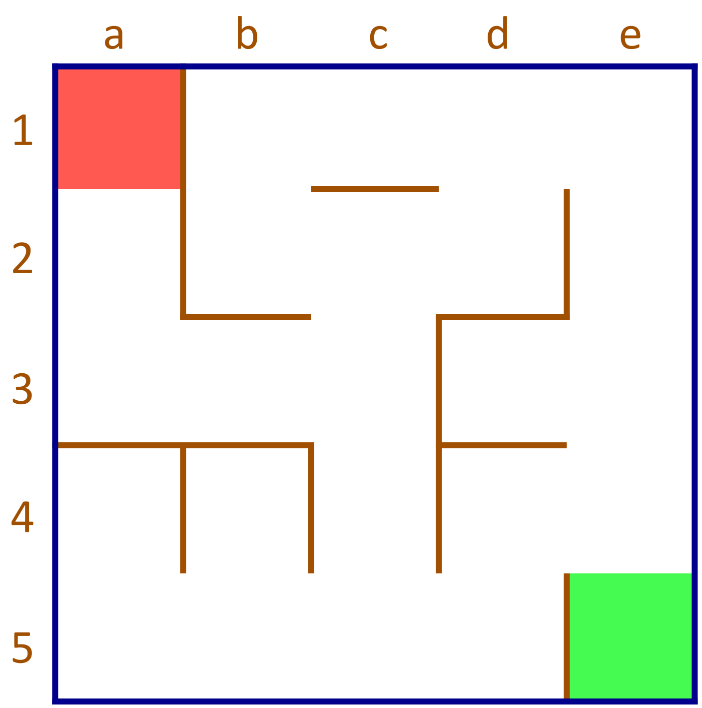


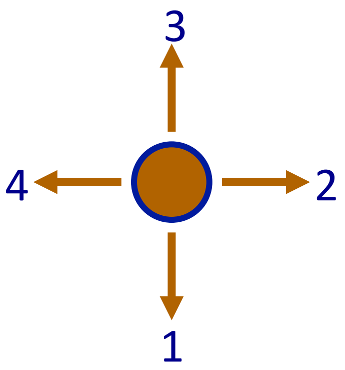
STACK



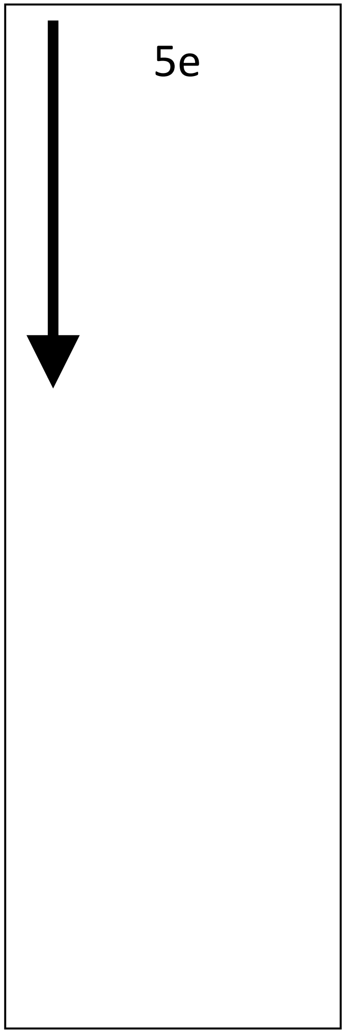
What happens if we use a Queue?

Application: Maze Path Finding Example

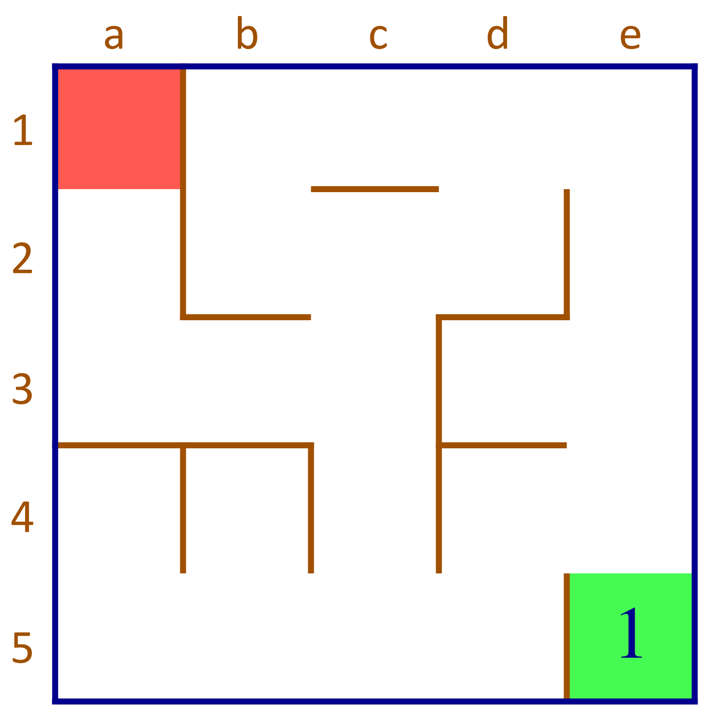


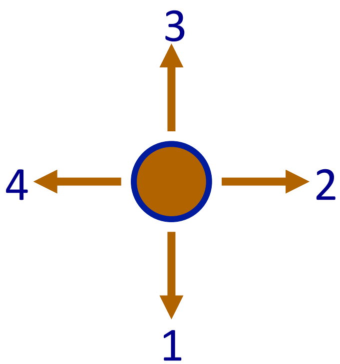


QUEUE

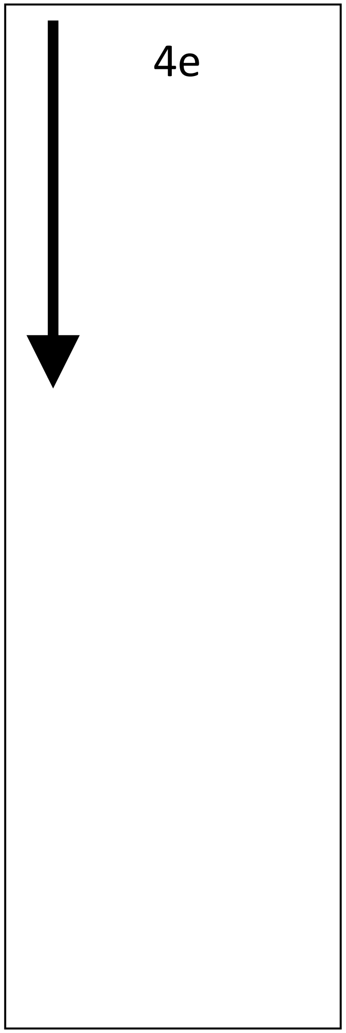


Application: Maze Path Finding Example

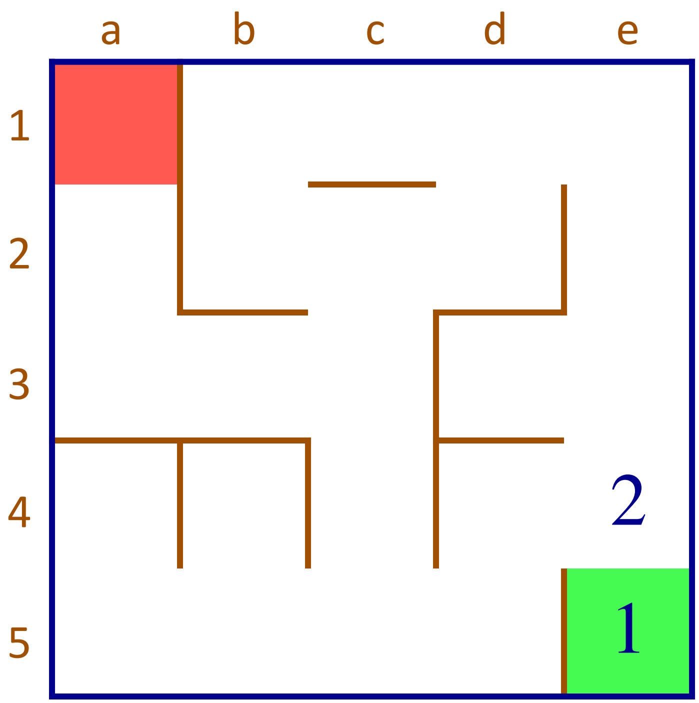


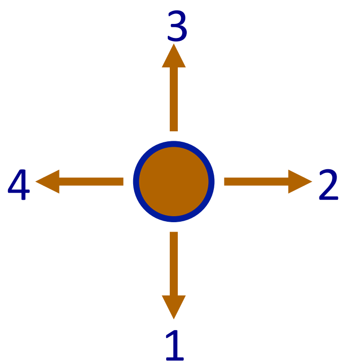


QUEUE

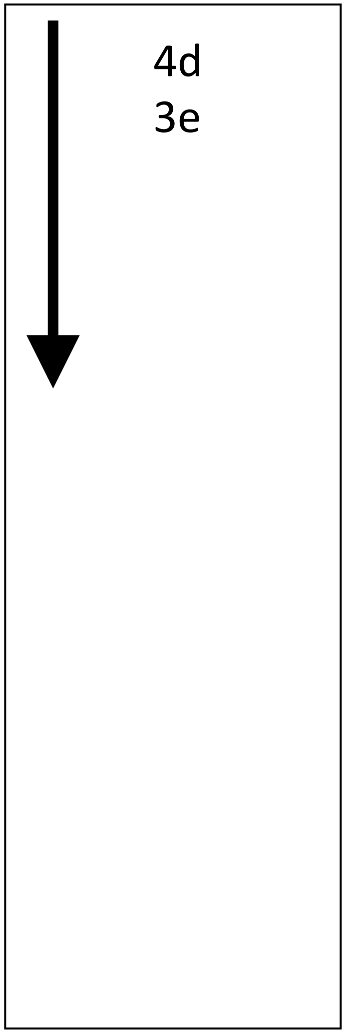


Application: Maze Path Finding Example

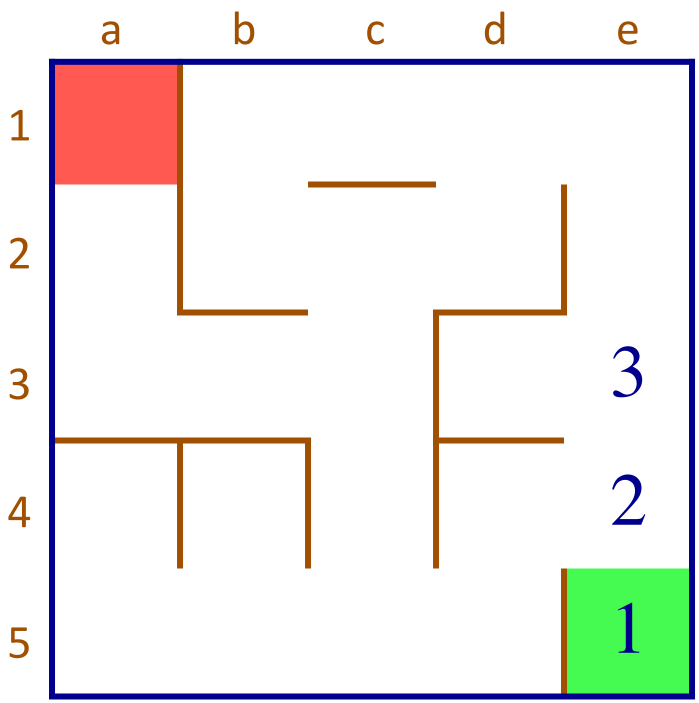


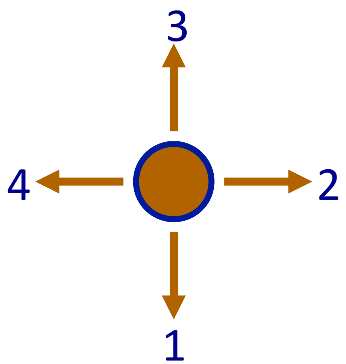


QUEUE

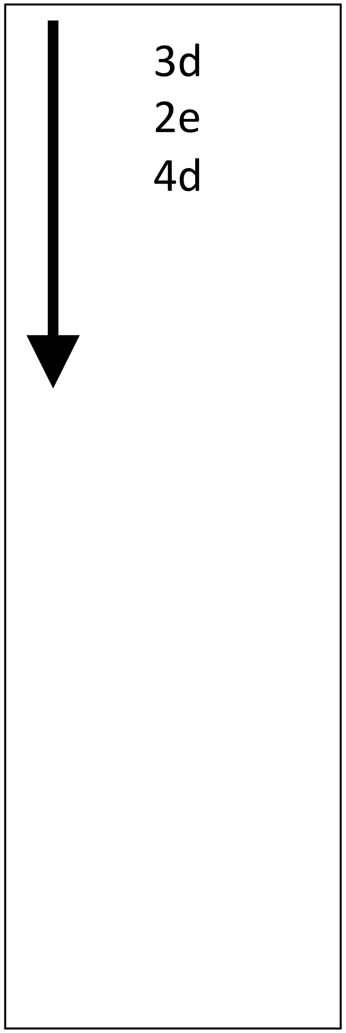


Application: Maze Path Finding Example

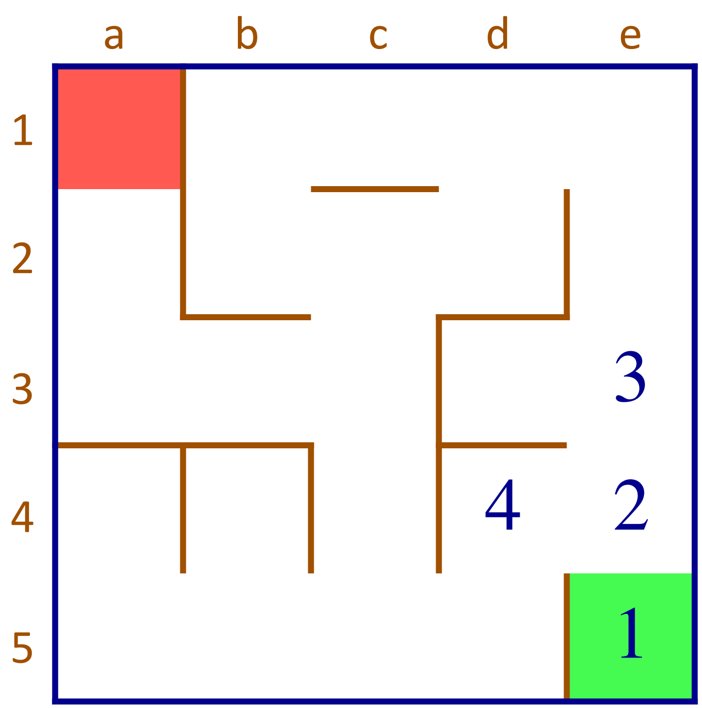


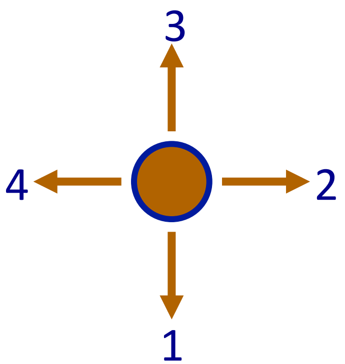


QUEUE



Application: Maze Path Finding Example

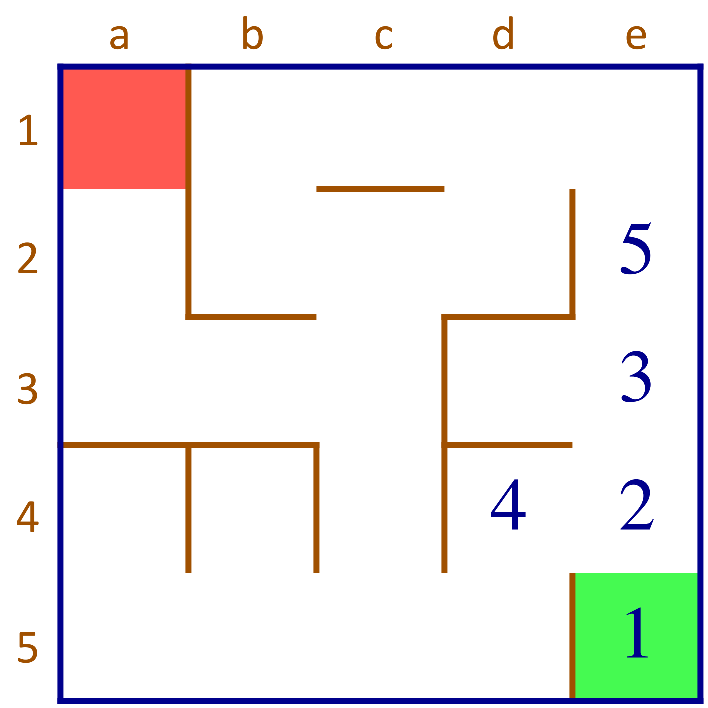


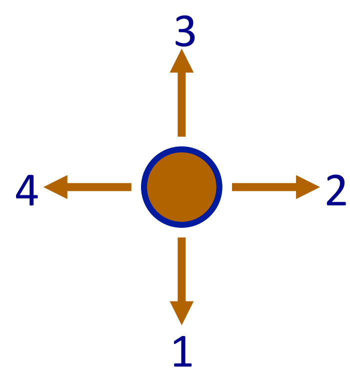


QUEUE

5d
3d
2e

Application: Maze Path Finding Example

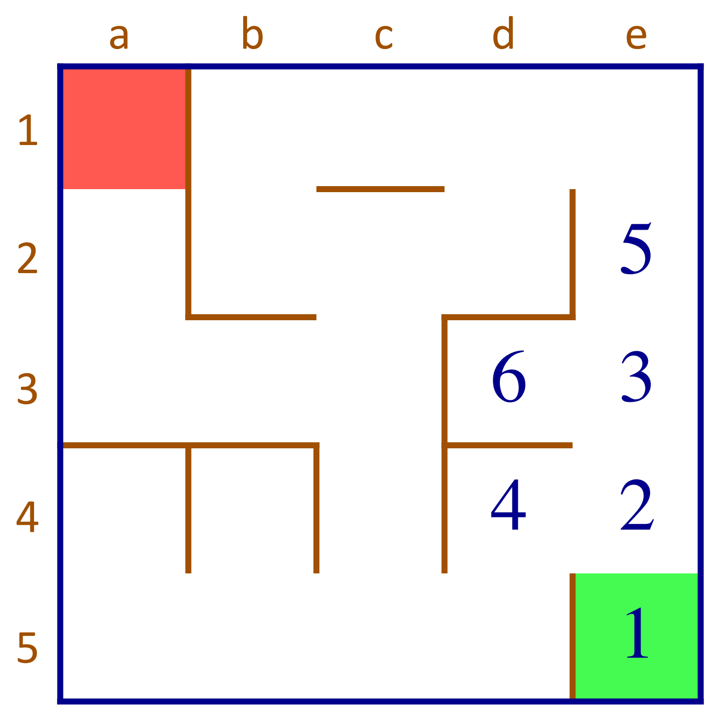


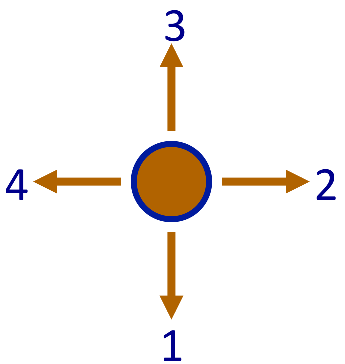


QUEUE

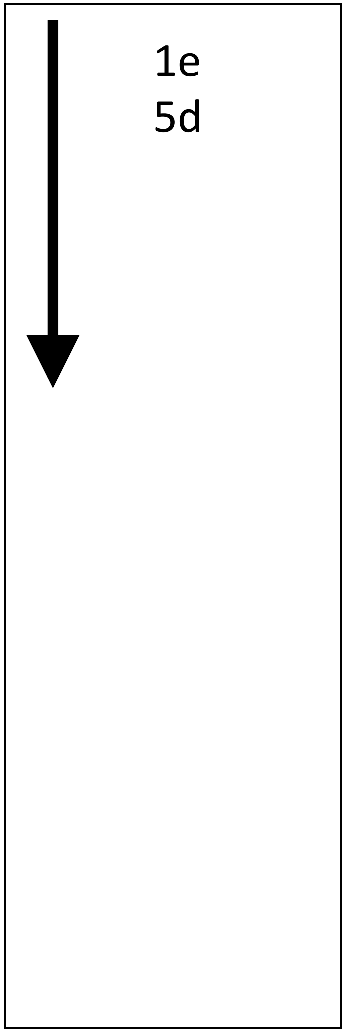
1e
5d
3d

Application: Maze Path Finding Example

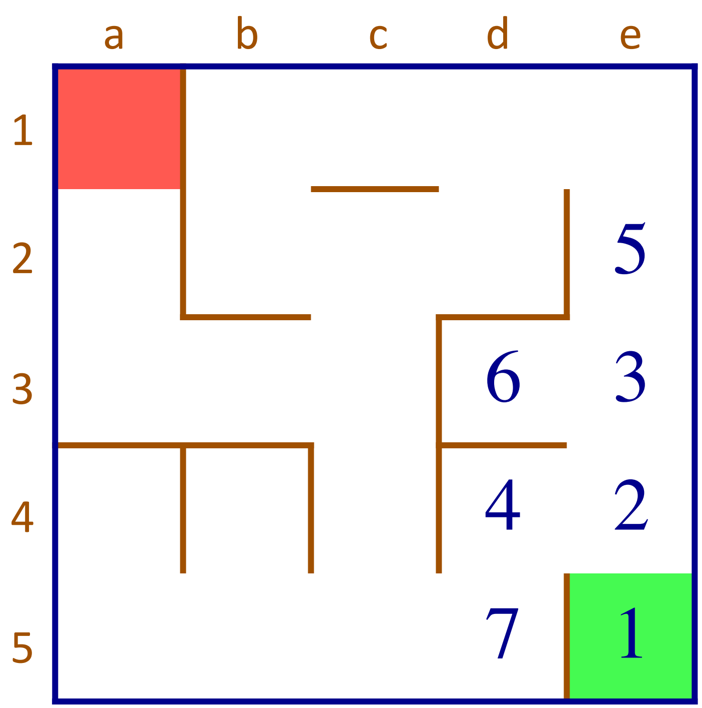


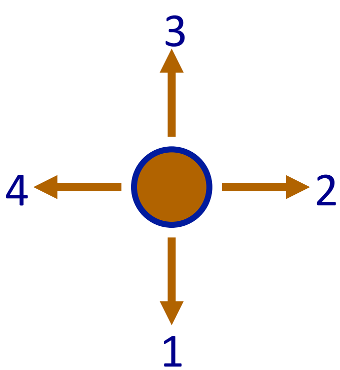


QUEUE

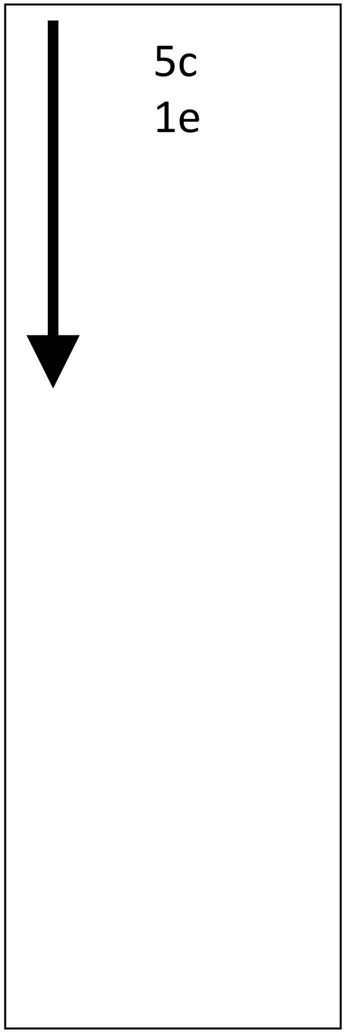


Application: Maze Path Finding Example

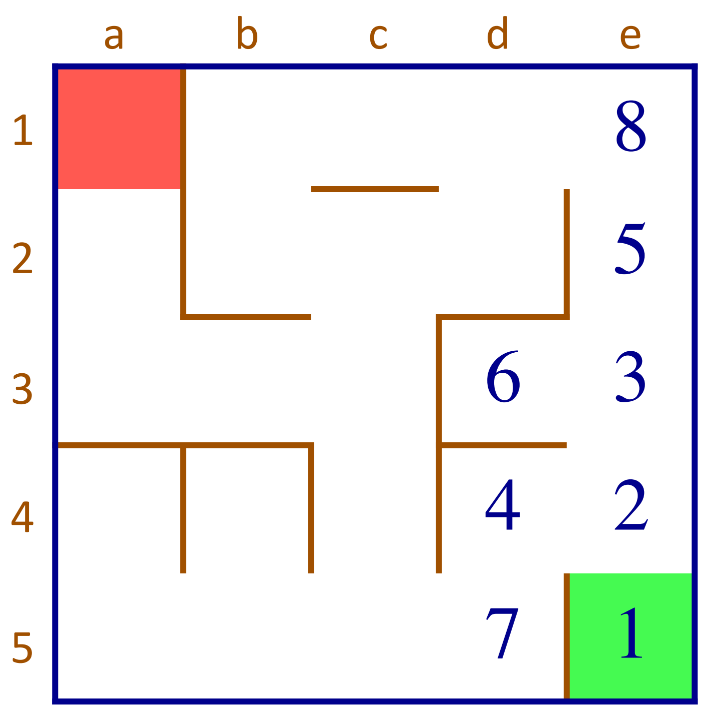


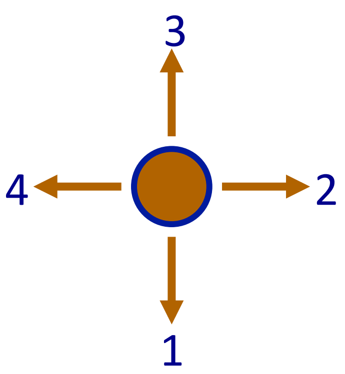


QUEUE

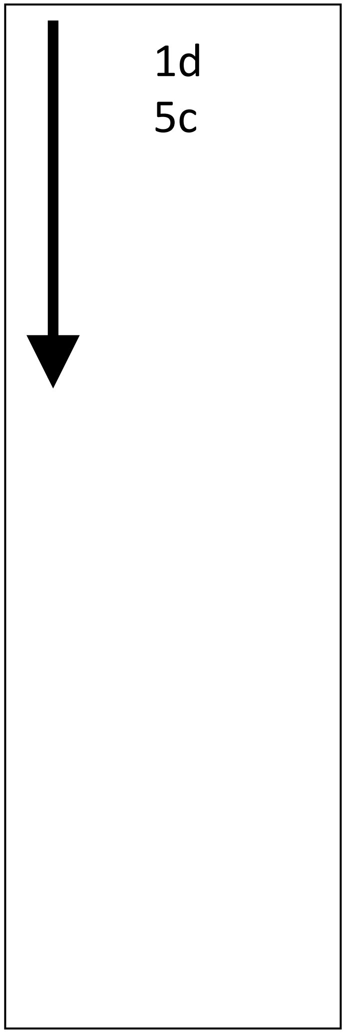


Application: Maze Path Finding Example

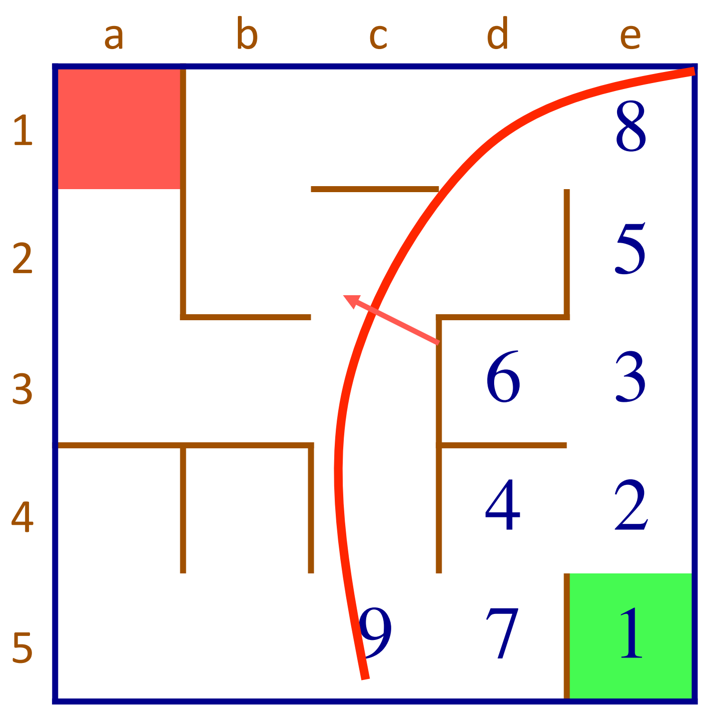


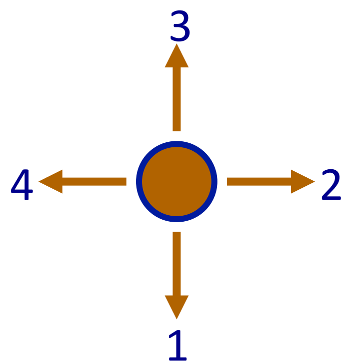


QUEUE



Application: Maze Path Finding Example

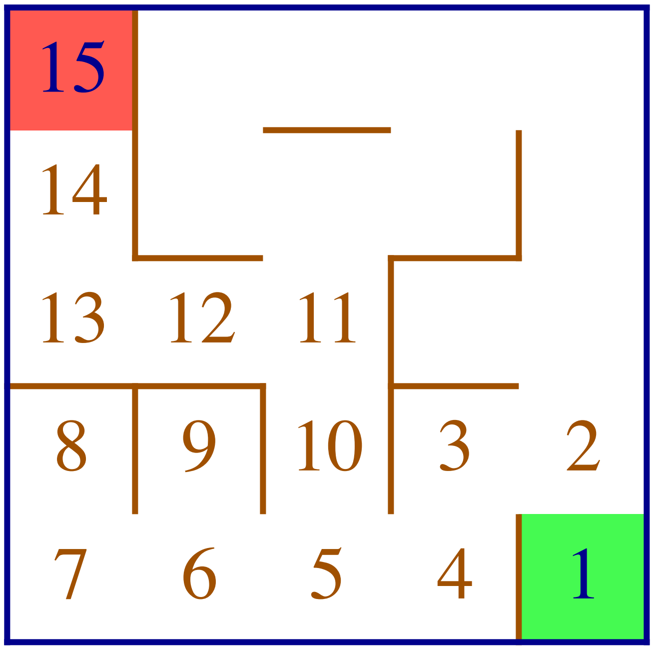




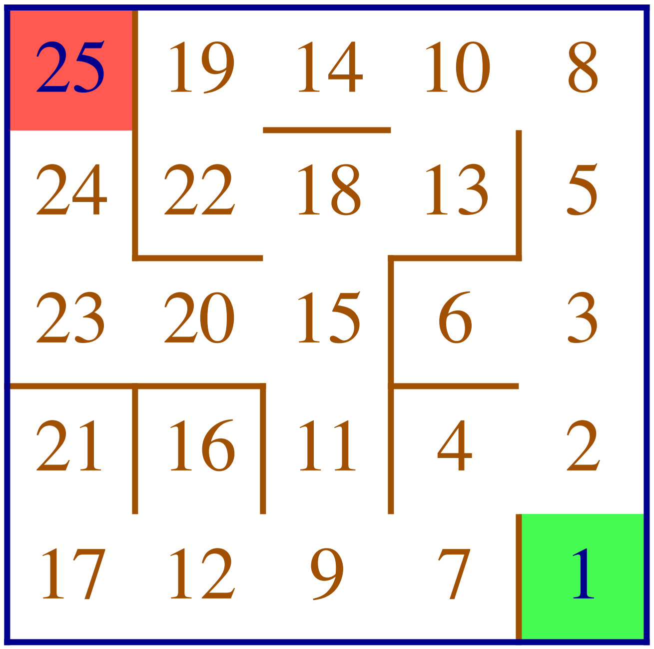
QUEUE

5b
4c
1d

Application: Maze Path Finding



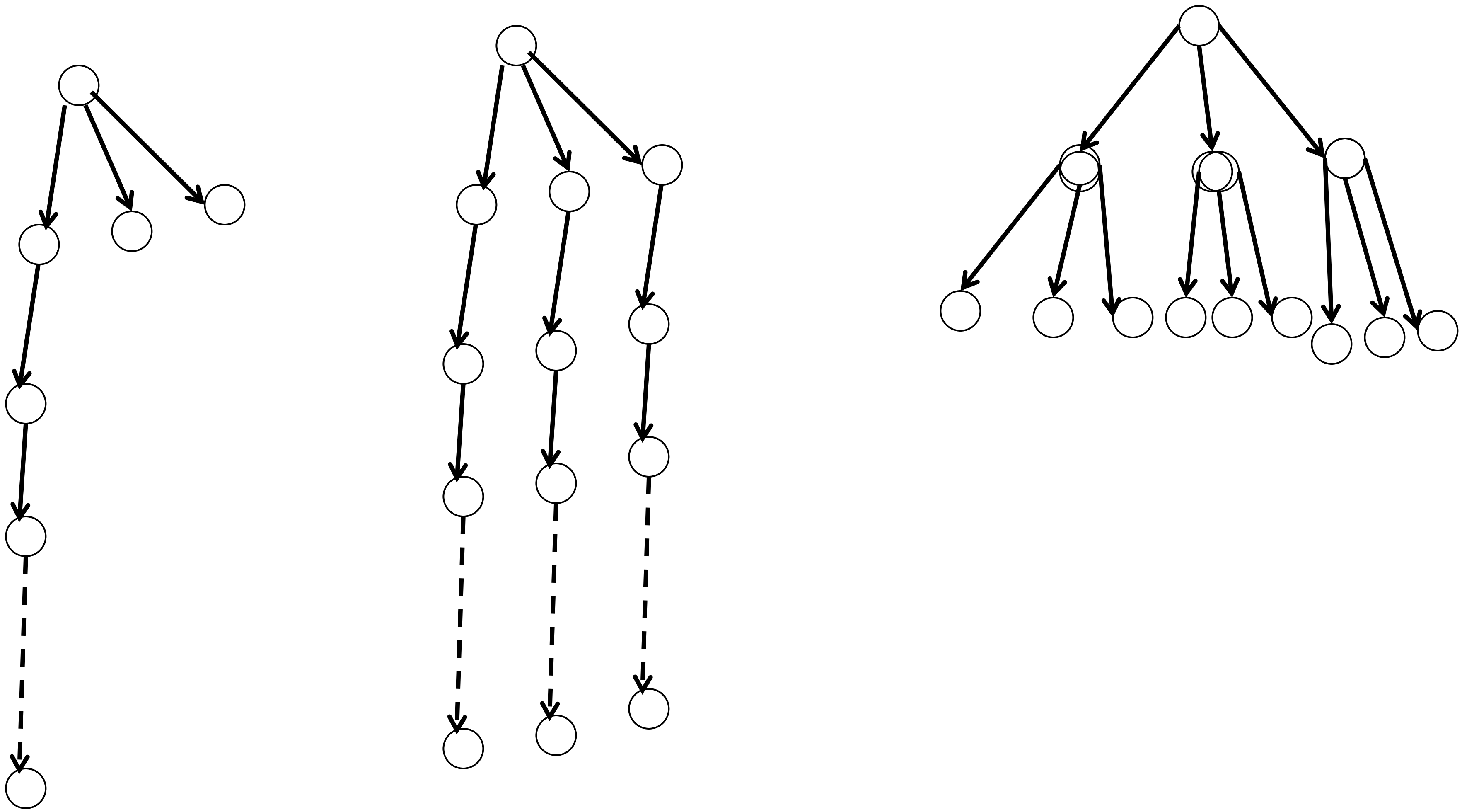
Depth-First (**Stack**)



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
  - If there are some very deep paths, DFS could spend a lot of time going down them
  - If it's a very broad/wide tree, BFS could require a lot of memory on the queue
  - If you need to find a shortest path, BFS guarantees is
  - Are solutions near top of the tree?
    - BFS may find it more quickly
    - e.g. Search a family tree for distant ancestor who was alive a long time ago
  - Are solutions at the leaves
    - DFS can find it more quickly
    - e.g. Search a family tree for someone who's still alive

Implementation Variations

- Can easily do DFS recursively
- Can avoid “Reachable” in both DFS/BFS by instead, adding a ***color*** field to each node
  - white: unvisited
  - gray: considered (on queue, stack)
  - black: reachable
- Store additional information to use in solving other important graph problems