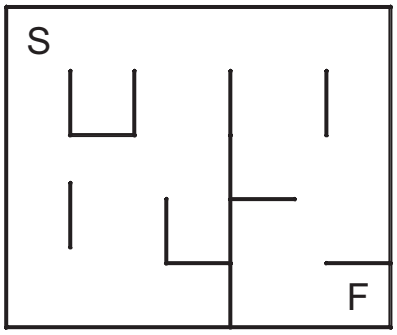
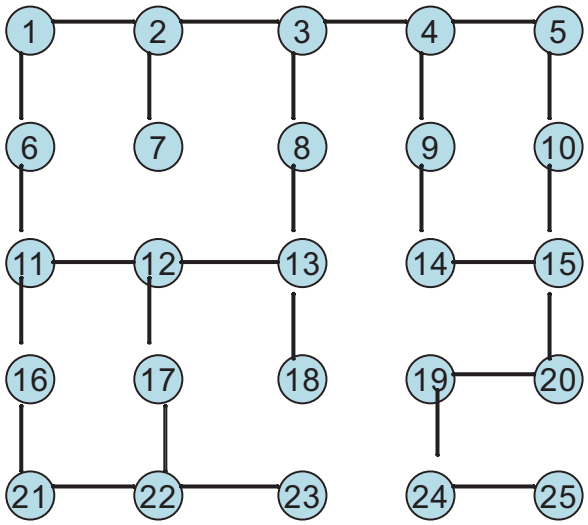
Worksheet 41: Depth First and Breadth First Search

Many different types of data can be represented by a graph. For example, consider the maze shown at right. We can encode the maze as an undirected graph by assigning each cell a value, and forming an arc when you can move from one cell to the next. The resulting graph is as follows:





The *reachablility problem* asks what cells (vertices) can be reached starting from the initial vertex. Of course, as anybody who has tried traversing a maze knows, it is not a simple matter of walking a fixed path, since you frequently have a choice of two or more unexplored alternatives. You will often find yourself in a dead-end, and must backtrack to investigate another possibility.

This problem can be expressed in data structure form as the following. You are given a graph and an initial vertex. The result you seek is a *set* of reachable vertices. To discover this you will use another container of vertices known to be reachable but possibly not yet explored. The reachability algorithm can be expressed in pseudo-code as follows

```
findReachable (graph g, vertex start) {
  create a set of reachable vertices, initially empty. call this r.
  create a container for vertices known to be reachable. call this c
  add start vertex to container c
  while the container c is not empty {
    remove first entry from the container c, assign to v
    if v is not already in the set of reachable vertices r {
      add v to the reachable set r
      add the neighbors of v to the container c if they are not already reachable
    }
  }
  return r
}
```

What is interesting about this algorithm is that the container c can be either a stack or a queue. The resulting search will be very different depending upon the data structure is selected. If a stack is used, it is termed *depth-first search*. If a queue is used, it is termed

breadth-first search. To explore this, simulate the algorithm on the graph given, and record the vertices in r in the order that they are placed into the collection. For the purpose of being able to reproduce a trace, please push the neighbors onto the stack (or add to the end of the queue), in clockwise order starting at the node to directly to the north.

Depth first search (stack version)

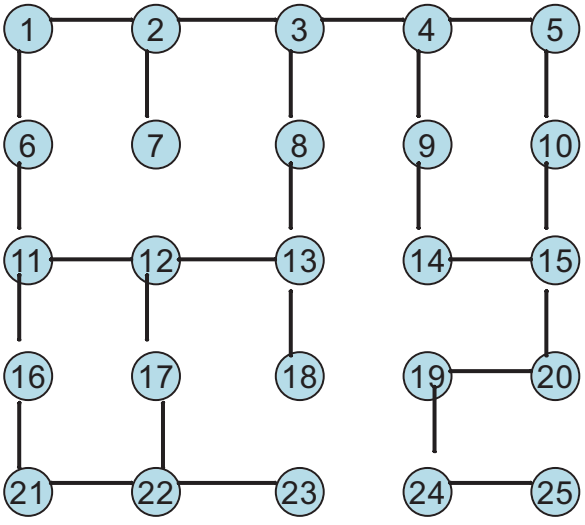
r:

Stack:

Breadth first search (queue version)

r:

Queue:



Using your finger, trace the sequence of vertices in the graph in the order that they are listed in r. Notice that the stack version moves in more of a continuous line, while the queue version seems to jump all over the graph. One way to visualize the two is that a depth first search is like a person walking through the maze. As long as they can move forward, they continue. It is only when they reach a dead-end that they move back to a point where there was a previous choice, selecting a different alternative. A breadth-first search, on the other hand, is like pouring a bottle of ink on the initial vertex. The ink moves from cell to cell, uniformly moving in all possible directions at the same time. Eventually the ink will find all reachable locations.

NOTE: I have not completed the iterations. I have provided 12 and 14 iterations for DFS and BFS respectively.

Solution: 11 iterations of DFS

Iteration	Stack(T—B)	Reachable
0	1	
1	6,2	1
2	11,2	1,6
3	16,12,2	1,6,11
4	21,12,2	1,6,11,16
5	22,12,2	1,6,11,16,21
6	23,17,12,2	1,6,11,16,21,22
7	17,12,2	1,6,11,16,21,22,23
8	12, 12, 2	1,6,11,16,21,22,23,17
9	13, 12, 2	1,6,11,16,21,22,23,17,12
10	18,8,12,2	1,6,11,16,21,22,23,17,12,13
11	8,12,2	1,6,11,16,21,22,23,17,12,13,18

13 iterations of BFS

Iteration	Queue (F---B)	Reachable
0	1	
1	2,6	1
2	6,3,7	1,2
3	3,7,11	1,2,6
4	7,11,4,8	1,2,6,3
5	11,4,8	1,2,6,3,7
6	4,8,12,16	1,2,6,3,7,11
7	8,12,16,5,9	1,2,6,3,7,11,4
8	12,16,5,9,13	1,2,6,3,7,11,4,8
9	16,5,9,13,13,17	1,2,6,3,7,11,4,8,12
10	5,9,13,13,17,21	1,2,6,3,7,11,4,8,12,16
11	9,13,13,17,21,10	1,2,6,3,7,11,4,8,12,16,5
12	13,13,17,21,10,14	1,2,6,3,7,11,4,8,12,16,5,9
13	13,17,21,10,14,18	1,2,6,3,7,11,4,8,12,16,5,9,13

Notice the size difference in the queue vs. the stack as the iterations proceed.