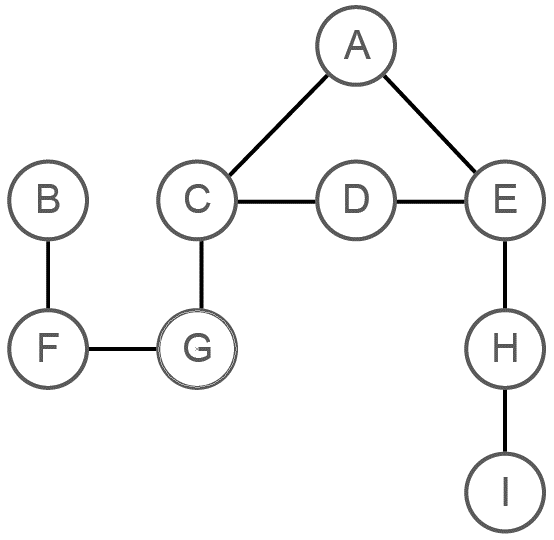
Homework 5 Graph traversal algorithms

1. Write down **two** possible orders of nodes visited, starting at A, for **each** of the following traversals:

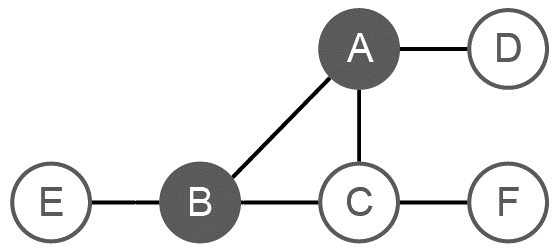
(i) depth-first traversal [2]

(ii) breadth-first traversal [2]



2. (a) Trace the order of nodes visited when the following graph is traversed using a depth-first traversal, showing the list of visited nodes and the use of the stack at each point. The first two steps are done for you. Visit C next, and stop when you have shown the contents of the stack and visited list at the next five steps. [5]

(b) What will be the next node visited after F? [1]



|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | A |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A |  | AB |  |  |  |  |  |  |  |  |  |  |

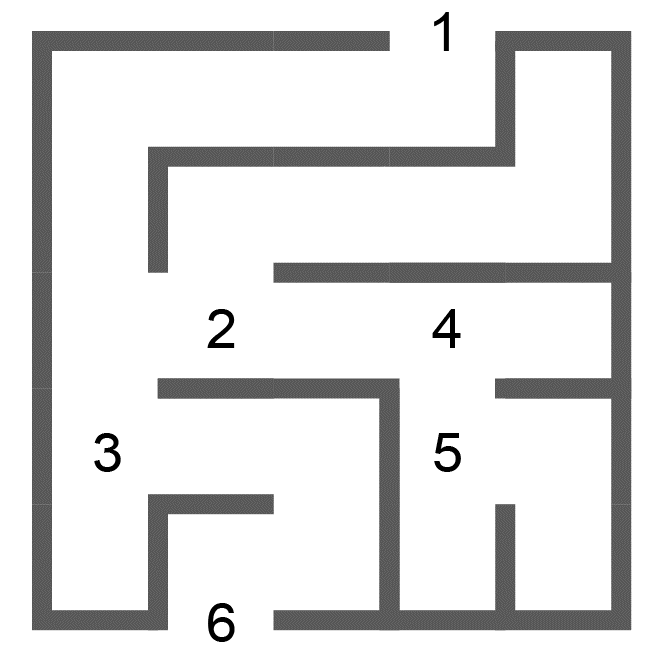
The final stage is an empty stack, which means every node has been visited.

3. A graph can be drawn to represent a maze. In such a graph, each graph vertex represents one of the following:

* the entrance to or exit from the maze
* a place where more than one path can be taken
* a dead end.

Edges connect the vertices according to the paths in the maze.

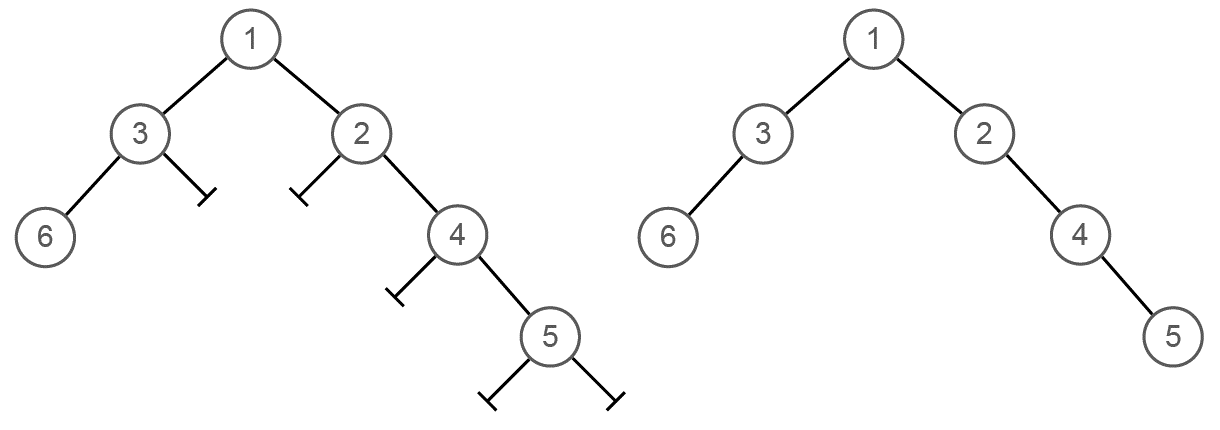
Figure 1 shows a maze.



*Figure 1*

Figure 2 shows one possible representation of this maze with dead ends marked.

Figure 3 shows a simplified undirected graph of this maze with dead ends omitted.



*Figure 2 Figure 3*

(a) Complete the table below to show how the graph in Figure 3 would be stored using an adjacency list. [2]

|  |  |  |  |
| --- | --- | --- | --- |
| **1** |  |  | [2,3] |
| **2** |  |  |  |
| **3** |  |  |  |
| **4** |  |  |  |
| **5** |  |  |  |
| **6** |  |  | [3] |

(b) A recursive routine can be used to perform a depth-first search of the graph that represents the maze to test if there is a route from the entrance 1 to the exit 6.

The routine is shown below. It has two parameters, v (the current vertex) and endV (the exit vertex)

procedure DFS(v, endV)

discovered[v] = True

if v == endV then found = True

for each neighbour u of v

if discovered[u] == False then DFS(u, endV)

next neighbour

completelyExplored[v] = True

endprocedure

Complete the trace table below to show how the discovered and completelyExplored flag arrays and the variable found are updated by the algorithm when it is called using DFS.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | **discovered** | | | | | | **completelyExplored** | | | | | |  |
|  | Call | v | u | endV | [1] | [2] | [3] | [4] | [5] | [6] | [1] | [2] | [3] | [4] | [5] | [6] | Found |
| 1 |  | - | - | 6 | F | F | F | F | F | F | F | F | F | F | F | F | F |
| 2 | DFS(1,6) | 1 | 2 | 6 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 | DFS(2,6) | 2 | 1 | 6 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  | 4 | 6 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5 | DFS(4,6) | 4 | 2 | 6 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 6 |  |  | 5 | 6 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 7 | DFS(5,6) | 5 | 4 | 6 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 8 | DFS(4,6) | 4 | - | 6 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 9 | DFS(2,6) | 2 | - | 6 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 10 | DFS(1,6) | 1 | 3 | 6 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 11 | DFS(3,6) | 3 | 1 | 6 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 12 |  | 3 | 6 | 6 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 13 | DFS(6,6) | 6 | 3 | 6 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 14 | DFS(3,6) | 3 | - | 6 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 15 | DFS(1,6) | 1 | - | 6 |  |  |  |  |  |  |  |  |  |  |  |  |  |

[5]

Total 17 marks