Do not write outside the box

10 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 6 shows a maze and **Figure 7** shows one possible representation of this maze. Position 1 in **Figure 6** corresponds to vertex 1 in **Figure 7** and is the entrance to the maze. Position 7 in **Figure 6** is the exit to the maze and corresponds to vertex 7. Dead ends have been represented by the symbol ——┤ in **Figure 7**.

Figure 8 shows a simplified undirected graph of this maze with dead ends omitted.

Figure 6

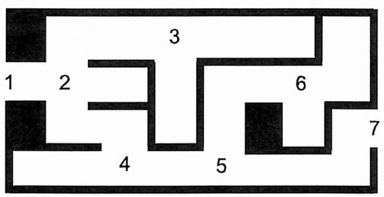
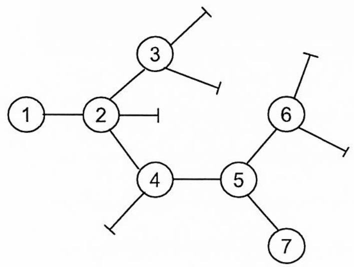
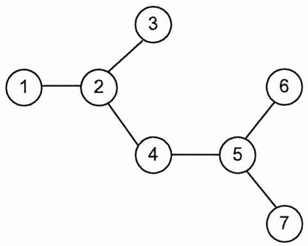


Figure 7



Representation of maze including dead ends

Figure 8



Graph representing maze with dead ends omitted

Do not write outside the box

10 (a) The graph in **Figure 8** is a tree.

State **one** property of the graph in **Figure 8** that makes it a tree.

....

....

(1 mark)

10 (b) The graphs of some mazes are not trees.

Describe a feature of a maze that would result in its graph **not** being a tree.

....

....

(1 mark)

10 (c) Complete the table below to show how the graph in **Figure 8** would be stored using an adjacency matrix.

(2 marks)

Question 10 continues on the next page

Turn over ▸

Do not write outside the box

10 (d) (i) What is a *recursive routine*?

....

....

(1 mark)

10 (d) (ii) To enable the use of recursion a programming language must provide a stack.

Explain what this stack will be used for and why a stack is appropriate.

....

....

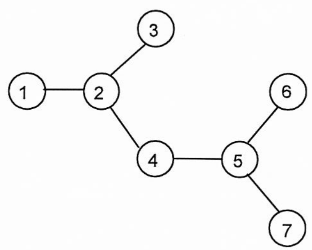
....

....

(2 marks)

Figure 8 from page 20 is repeated here so that you can answer Question 10(e) without having to turn back in the question booklet.

Figure 8 (repeated)



10 (e) 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 (vertex 1) to the exit (vertex 7).

The recursive routine in **Figure 9** is to be used to explore the graph in **Figure 8**. It has two parameters, `V` (the current vertex) and `EndV` (the exit vertex).

Do not write outside the box

Figure 9

```
Procedure DFS(V, EndV)
  Discovered[V] ← True
  If V = EndV Then Found ← True
  For each vertex U which is connected to V Do
    If Discovered[U] = False Then DFS(U, EndV)
  EndFor
  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(1,7)`.

The details of each call and the values of the variables `V`, `U` and `EndV` have already been entered into the table for you. The letter `F` has been used as an abbreviation for `False`. You should use `T` as an abbreviation for `True`.

				Discovered							CompletelyExplored							
Call	V	U	EndV	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	Found
	-	-		F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
DFS(1,7)	1	2	7															
DFS(2,7)	2	1	7															
		3	7															
DFS(3,7)	3	2	7															
DFS(2,7)	2	4	7															
DFS(4,7)	4	2	7															
		5	7															
DFS(5,7)	5	4	7															
		6	7															
DFS(6,7)	6	5	7															
DFS(5,7)	5	7	7															
DFS(7,7)	7	5	7															
DFS(5,7)	5	-	7															
DFS(4,7)	4	-	7															
DFS(2,7)	2	-	7															
DFS(1,7)	1	-	7															

(5 marks)

12

Turn over ▸