ANSWERS

A4.3 Depth-first traversal

Trees and graphs · A level · OCR H446 2.3.1, AQA 7517 4.3.1.1 · about 25 min

BugBotLab

What this lesson is about

Going deep and backtracking, recursively and with a stack; tracing it and what it is used for.

- The idea
- Recursive depth-first traversal
- With a stack of your own
- What depth-first traversal is for

Questions

5 marks in all

1. Which data structure does an iterative depth-first traversal use? [1 mark]

- ☐ **A** A stack
- ☐ **B** A queue
- ☐ **C** A priority queue
- ☐ **D** A hash table

> **Answer:** A. The most recently found vertex is explored next: last in, first out. A recursive version uses the call stack.

2. An undirected graph has the edges A-B, A-C and B-D. Put the vertices in the order a depth-first traversal from A visits them, trying neighbours in alphabetical order. [1 mark]

Number the lines 1 to 4 to put them in the right order.

___ C
___ D
___ A
___ B

> **Answer:**
>
> ```
> A
> B
> D
> C
> ```
>
> From A it goes to B, then as deep as possible to D, then backtracks to A and visits C.

3. Which application does AQA's specification give for depth-first search? [1 mark]

- ☐ **A** Navigating a maze
- ☐ **B** Finding the shortest path in an unweighted graph
- ☐ **C** Sorting a list of numbers
- ☐ **D** Searching a sorted array

> **Answer:** A. Follow one passage to its end, then backtrack. Shortest paths in unweighted graphs are the breadth-first application.

4. What does this program print? [1 mark]

```
GRAPH = {"P": ["Q", "S"], "Q": ["P", "R"], "R": ["Q", "S"], "S": ["P", "R"]}
visited = []
def dfs(v):
    visited.append(v)
    for n in GRAPH[v]:
        if n not in visited:
            dfs(n)
dfs("P")
print(" ".join(visited))
```

Answer:

```
P Q R S
```

P, then Q, then Q's unvisited neighbour R, then R's unvisited neighbour S. S was reached through the cycle, not directly from P.

5. Why does a depth-first traversal mark vertices as visited? [1 mark]

- ☐ **A** Without the marks it could go round a cycle forever
- ☐ **B** To make it find the shortest path
- ☐ **C** To sort the vertices into order
- ☐ **D** Because a stack cannot hold the same item twice

Answer: A. A cycle leads back to a vertex already seen. The mark stops the traversal entering it again.

The task: what can be reached

The graph in `GRAPH` is undirected and stored as an adjacency list: each vertex's neighbours, in the order to try them. Write a function `dfs` that traverses the graph depth-first from a start vertex, recursively or with your own stack, and records the order in which the vertices are visited. Try the neighbours in the order the list gives them. Print `DFS order:` followed by the vertices in the order visited from A, separated by single spaces. Then print `not reachable:` followed by every vertex that was never visited, in alphabetical order, separated by single spaces.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

GRAPH = {
    "A": ["B", "C"],
    "B": ["A", "D", "E"],
    "C": ["A", "F"],
    "D": ["B"],
    "E": ["B", "F"],
    "F": ["C", "E"],
    "G": ["H"],
    "H": ["G"],
}

visited = []
```

The hint students can ask for: Mark a vertex visited the moment you arrive, then try its neighbours in the order the list gives them, going as deep as you can down each before trying the next. Anything never marked could not be reached.

A solution

```
from bugbot import *
connect()

GRAPH = {
    "A": ["B", "C"],
    "B": ["A", "D", "E"],
    "C": ["A", "F"],
    "D": ["B"],
    "E": ["B", "F"],
    "F": ["C", "E"],
    "G": ["H"],
    "H": ["G"],
}

def dfs(graph, vertex, visited):
    visited.append(vertex)
    for nxt in graph[vertex]:
        if nxt not in visited:
            dfs(graph, nxt, visited)
    return visited

order = dfs(GRAPH, "A", [])
print("DFS order:", " ".join(order))
print("not reachable:", " ".join(v for v in sorted(GRAPH) if v not in order))
```

Any program that meets the task's checks is marked correct in the simulator; this is one way, not the only way.