WORKSHEET

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

Name _____ Class _____ Date _____

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

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

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

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))
```

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

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 = []
```

Plan your program here, then type it in and press Run.

Do it on the robot

www.bugbotlab.com/learn/a4-3-depth-first-traversal/

The simulator checks it and tells you when it passes. Nothing to install, no account.

Challenges

1. Make `dfs` also count the edges it follows to reach a new vertex. How does the count compare with the number of vertices visited, and why?
2. Write `has_path(start, goal)` that stops as soon as `goal` is visited, and returns True or False.
3. Reverse every neighbour list. Does the DFS order change? Does the set of vertices reached?