LESSON 65

Trees and Graphs

80 minutes (2 periods) | Advanced Algorithms and Data Structures | Outcome 1.6 | Outcome 1.7 | Outcome 1.22 | Outcome 2.6 | Outcome 2.7 | Outcome 2.8 | Outcome 2.9

WHAT THIS LESSON DOES

In this lesson, you'll explore key data structures in computer science, learning to represent them with Python tools like dictionaries and lists. You'll implement depth-first and breadth-first search traversals and apply these skills to a maze path-finding project in VS Code.

CURRICULUM OUTCOMES

Outcome	What the specification asks for
1.6	explain the operation of a variety of algorithms
1.7	develop algorithms to implement chosen solutions
1.22	read, write, test, and modify computer programs
2.6	construct algorithms using appropriate sequences, selections/conditionals, loops and operators to solve a range of problems, to fulfil a specific requirement
2.7	implement algorithms using a programming language to solve a range of problems
2.8	apply basic search and sorting algorithms and describe the limitations and advantages of each algorithm
2.9	assemble existing algorithms or create new ones that use functions (including recursive), procedures, and modules

WHAT STUDENTS SHOULD BE ABLE TO DO

- Understand the fundamental concepts and differences between trees and graphs as data structures.
- Represent trees and graphs using Python data structures like dictionaries and lists.
- Implement depth-first search (DFS) and breadth-first search (BFS) traversal algorithms.
- Apply traversal algorithms to solve practical problems such as path-finding in networks.
- Recognise the applications of trees and graphs in real-world scenarios like social networks and hierarchical systems.

BEFORE THE LESSON

- Confirm VS Code and the Python interpreter run on the student machines, and that a plain .py file executes without setup fuss.
- Have students create a TreesGraphs folder to hold tree.py, graph.py and friend_finder.py, since the lesson builds one file at a time.
- Run the DFS and BFS graph.py code yourself first so you can point out the output pattern before students hit the recursion.

Trees and Graphs

HOW THE LESSON RUNS

- Introduction
- Understanding Trees
- Representing Trees in Python
- Understanding Graphs
- Depth-First Search (DFS)
- Implementing DFS
- Breadth-First Search (BFS)
- Implementing BFS
- Friend Finder
- DFS Solution
- BFS Solution
- Challenge
- Wrap Up

TEACHING MOVES

- 2. Understanding Trees. Draw the A-B-C-D-E-F tree on the board and label root, leaf, edge and height on it. A family tree is the reference students already hold, so anchor every term to it.
- 3. Representing Trees in Python. Show that the dictionary key is the parent and the list is its children. Read one line, 'A maps to B and C', aloud as a sentence so the structure and the picture line up.
- 4. Understanding Graphs. Contrast directly with the tree: a graph can have cycles and no fixed top. Trace a path that returns to its start to make the cycle concrete before any code.
- 5. Depth-First Search (DFS). Use the maze image: follow one path to the dead end, then backtrack. Physically trace it on the board graph with your finger before showing the recursive code.
- 6. Implementing DFS. Highlight the visited set as the thing that stops infinite loops on cycles. Walk through what happens the second time the code reaches an already-visited node.
- 7. Breadth-First Search (BFS). Describe BFS as a wave spreading level by level, contrasting it with the DFS deep dive. Stress that this is what gives BFS the shortest path in an unweighted graph.
- 8. Implementing BFS. Point out the queue where DFS used a stack or recursion. Trace the first two dequeue steps by hand so students see nodes coming out in distance order.
- 9. Friend Finder. Frame the switch to an undirected graph: friendships are mutual, so each edge appears in both people's lists. Point out Grace has no friends and Henry-Isabel are isolated, as test cases.
- 10. DFS Solution. Run a connected pair and a disconnected pair, such as Alice to Grace, so students see both a True and a False result and trust the function.
- 11. BFS Solution. Ask which of DFS or BFS you would trust to give the shortest chain of friends, and why. Draw the answer out before moving to the challenge.
- 12. Challenge. Steer students to store each node's parent as they enqueue it, then walk parents backwards from the goal. Remind them to return an empty list when no path exists.

WHAT TO WATCH FOR

- Students think trees and graphs are different things entirely, missing that a tree is just a graph with no cycles and one root. Show the same dictionary representation works for both. Ask what single property makes the tree example a tree, and elicit 'no cycles, one root'.
- Students believe DFS and BFS visit different nodes, rather than the same nodes in a different order. Run both on the same graph and compare the printed lists side by side. Same nodes, different sequence. Only the order differs.
- Students forget the visited set and write a traversal that loops forever on a cyclic graph. Temporarily delete the visited check in front of the class and let it hang or crash, then restore it. The failure teaches the reason.

DIFFERENTIATION

- Emerging: Give them tree.py only. Getting the dictionary to print its children correctly before any traversal is a complete success for one period. Let them trace DFS on paper with a pencil, ticking visited nodes, before they read the recursive code.
- Developing: Ask them to predict the printed order of BFS on paper, then run it and check. Prediction before execution builds the mental model.
- Proficient: Point them at the shortest-path challenge and then ask them to count how many nodes each of DFS and BFS visited, opening the door to Higher Level efficiency comparison. Have them add a disconnected person of their own and predict what each function returns before running it.