ANSWERS

A4.2 Adjacency matrix and adjacency list

Trees and graphs · A level · OCR H446 1.4.2, AQA 7517 4.2.4.1, Eduqas A500QS 1.1 · about 25 min

What this lesson is about

Two ways to store a graph, and choosing between them for dense and sparse graphs.

- The adjacency matrix
- The adjacency list
- Which to use

Questions

6 marks in all

1. An adjacency matrix is symmetric about its leading diagonal. What does that tell you? [1 mark]

- ☐ **A** The graph is undirected, or every edge has a matching edge the other way
- ☐ **B** The graph is weighted
- ☐ **C** The graph has no cycles
- ☐ **D** Every vertex has the same degree

> **Answer:** A. Row u, column v equals row v, column u, so every connection works both ways.

2. How many cells does the adjacency matrix of a graph with 50 vertices have? [1 mark]

> **Answer:** 2500. One row and one column per vertex: 50 × 50.

3. A road map has 10,000 junctions, and each junction joins about 3 roads. Which representation suits it, and why? [1 mark]

- ☐ **A** An adjacency list, because the graph is sparse
- ☐ **B** An adjacency matrix, because the graph is sparse
- ☐ **C** An adjacency matrix, because it uses less memory
- ☐ **D** An adjacency list, because it tests for an edge in one look

> **Answer:** A. A matrix would need 100,000,000 cells, almost all empty. The list stores only the roads that exist.

4. What is an advantage of an adjacency matrix over an adjacency list? [1 mark]

- ☐ **A** Checking whether two particular vertices are joined takes one look
- ☐ **B** It uses less memory for a sparse graph
- ☐ **C** Adding a new vertex needs no extra space
- ☐ **D** Listing a vertex's neighbours never looks at more cells than it has neighbours

> **Answer:** A. matrix[u][v] answers directly. A list has to be searched.

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

```
VERTICES = ["A", "B", "C"]
matrix = [[0, 1, 1],
          [0, 0, 1],
          [1, 0, 0]]
for i in range(3):
    out = []
    for j in range(3):
        if matrix[i][j] == 1:
            out.append(VERTICES[j])
    print(VERTICES[i], out)
```

Answer:

```
A ['B', 'C']
B ['C']
C ['A']
```

Each row of the matrix becomes that vertex's list: the columns holding 1 are the vertices its edges go to.

6. What word describes a graph that has only a small number of edges compared with the number it could have? [1 mark]

Answer: sparse. A sparse graph suits an adjacency list; a dense graph, with many edges, suits a matrix.

The task: matrix and list

The graph in `EDGES` is **directed** and weighted, as drawn below: `("A", "B", 4)` is an edge from A to B of weight 4, and there is no edge from B to A unless one is listed. `VERTICES` gives the order of the matrix's rows and columns. Build an adjacency matrix called `matrix`, a list of lists with `matrix[i][j]` holding the weight of the edge from `VERTICES[i]` to `VERTICES[j]` and 0 where there is no edge, and print it one row per line: the vertex, a colon, then the row's values separated by single spaces, such as `A: 0 4 0 7`. Then build an adjacency list as a dictionary, with each vertex's neighbours in the order their edges appear in `EDGES`, and print it one vertex per line in `VERTICES` order, in the form `A -> B(4) D(7)`. Build both from `EDGES`: do not type the rows.

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

VERTICES = ["A", "B", "C", "D"]
EDGES = [("A", "B", 4), ("A", "D", 7), ("B", "C", 3), ("C", "A", 2), ("D", "C", 5), ("B",
"D", 1)]

n = len(VERTICES)
```

The hint students can ask for: The matrix needs a row and a column for every vertex, starting full of zeros; each edge sets the one cell in its from-row and to-column. The list gives each vertex its own list, and each edge adds to the list of the vertex it leaves.

A solution

```
from bugbot import *
connect()

VERTICES = ["A", "B", "C", "D"]
EDGES = [("A", "B", 4), ("A", "D", 7), ("B", "C", 3), ("C", "A", 2), ("D", "C", 5), ("B",
"D", 1)]

n = len(VERTICES)
matrix = [[0] * n for i in range(n)]
for u, v, w in EDGES:
    matrix[VERTICES.index(u)][VERTICES.index(v)] = w
for i in range(n):
    print(VERTICES[i] + ":", " ".join(str(x) for x in matrix[i]))

adj = {v: [] for v in VERTICES}
for u, v, w in EDGES:
    adj[u].append((v, w))
for u in VERTICES:
    print(u, "->", " ".join(f"{v}({w})" for v, w in adj[u]))
```

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