ANSWERS

F10.4 Network topologies

Networks · GCSE · OCR J277 1.3.1, Edexcel 1CP2 4.1.8 · about 20 min

What this lesson is about

Star, bus, ring and mesh, and routing around a failure in a mesh of robots.

- Four topologies
- Routes through a mesh

Questions

6 marks in all

1. In a star topology, what happens if the central switch fails? [1 mark]

- ☐ **A** No device can communicate
- ☐ **B** Only one device is affected
- ☐ **C** Data takes another route
- ☐ **D** Nothing, the devices connect directly

> **Answer:** A. Every connection goes through the switch.

2. Why is a mesh network very reliable? [1 mark]

- ☐ **A** There are many routes, so data can go around a failed link
- ☐ **B** It has a single central switch
- ☐ **C** It uses less cable
- ☐ **D** Data travels in one direction

> **Answer:** A. Redundant links give alternative routes.

3. How many connections does a full mesh of 5 devices need? [1 mark]

> **Answer:** 10. $5 \times 4 \div 2$.

4. In a bus topology, what happens if the backbone cable breaks? [1 mark]

- ☐ **A** The whole network fails
- ☐ **B** Only one device is affected
- ☐ **C** Data goes the other way round
- ☐ **D** The switch reroutes it

> **Answer:** A. Every device shares the one cable.

5. In a ring topology, how does data travel? [1 mark]

- ☐ **A** From device to device around the loop, in one direction
- ☐ **B** Through a central switch
- ☐ **C** Along one shared cable to every device at once
- ☐ **D** Directly between every pair of devices

> **Answer:** A. Each device passes data on to the next.

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

```
links = {"A": ["B"], "B": ["A", "C"], "C": ["B"]}
seen = ["A"]
for robot in seen:
    for n in links[robot]:
        if n not in seen:
            seen.append(n)
print(seen)
```

Answer:

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

A reaches B, then B reaches C.

The task: route around a failure

Write `route(links, start, end)` that returns the shortest route as a list of robots, using a breadth-first search. Print `route: A-B-D-F` style for the route from A to F, joining the names with `-`, and send the same text by radio. Then the link between D and F fails: remove it from both robots' lists, find the route again, and print `after D-F fails: <route>`.

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

links = {"A": ["B", "C"], "B": ["A", "D"], "C": ["A", "D", "E"], "D": ["B", "C", "F"], "E":
["C", "F"], "F": ["D", "E"]}
```

The hint students can ask for: Breadth-first search: keep a queue of robots to visit and a record of which robot you reached each one from. When it is done, follow that record back from the end to the start and reverse it.

A solution

```
from bugbot import *
connect()
links = {"A": ["B", "C"], "B": ["A", "D"], "C": ["A", "D", "E"], "D": ["B", "C", "F"], "E":
["C", "F"], "F": ["D", "E"]}

def route(links, start, end):
    came_from = {start: None}
    queue = [start]
    while queue:
        robot = queue.pop(0)
        for neighbour in links[robot]:
            if neighbour not in came_from:
                came_from[neighbour] = robot
                queue.append(neighbour)
    path = []
    robot = end
    while robot is not None:
        path.append(robot)
        robot = came_from[robot]
    path.reverse()
    return path

text = "route: " + "-".join(route(links, "A", "F"))
print(text)
send(text)
links["D"].remove("F")
links["F"].remove("D")
print("after D-F fails:", "-".join(route(links, "A", "F")))
```

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