ANSWERS

A8.6 Karnaugh maps

Boolean algebra and logic circuits · A level · OCR H446 1.4.3, AQA 7517 4.6.5.1, Eduqas A500QS 1.2 · about 25 min

What this lesson is about

Gray code order, grouping rules for two to four variables, wrap-around groups, and reading off the expression.

- The idea
- Gray code order
- A two-input map
- A three-input map
- A four-input map
- The grouping rules
- Drawing a map by program

Questions

5 marks in all

1. In what order are the columns of a four-column Karnaugh map labelled? [1 mark]

- ☐ **A** 00, 01, 11, 10
- ☐ **B** 00, 01, 10, 11
- ☐ **C** 00, 10, 01, 11
- ☐ **D** 11, 10, 01, 00 only

> **Answer:** A. Gray code order, so that neighbouring columns differ in exactly one bit.

2. Which of these are allowed as groups on a Karnaugh map? [1 mark]

Tick every answer that is true.

- ☐ **A** A group of 4 cells in a square
- ☐ **B** A group of 3 cells in a row
- ☐ **C** A group of 2 cells that wraps from the left edge to the right edge
- ☐ **D** A group of 2 cells on a diagonal
- ☐ **E** Two groups that overlap

> **Answer:** A, C, E. Groups are rectangles of 1, 2, 4, 8 or 16 cells; they may wrap round and overlap, but never be diagonal.

3. In a four-input Karnaugh map, how many literals does a group of 4 cells give? [1 mark]

> **Answer:** 2. Each doubling removes one letter: 16 cells is 0 literals, 8 is 1, 4 is 2, 2 is 3, 1 is 4.

4. A three-input map (rows A, columns BC) has 1s in columns BC = 00 and BC = 10 in both rows, and 0s elsewhere. What is the expression? [1 mark]

- ☐ **A** ¬C
- ☐ **B** ¬B
- ☐ **C** A ∧ ¬C
- ☐ **D** B ∨ C

> **Answer:** A. The two columns are neighbours by wrap-around; across the group A and B change, but C is always 0.

5. Why does a Karnaugh map use Gray code order rather than binary order? [1 mark]

- ☐ **A** So that neighbouring cells differ in exactly one input, and can be combined
- ☐ **B** So that the map has fewer cells
- ☐ **C** Because binary order has no 11
- ☐ **D** So that the 1s end up in the corners

> **Answer:** A. Two terms that differ in one input combine and lose that input; Gray code puts every such pair side by side.

The task: draw the map, read the groups

`f(a, b, c, d)` takes four bits (each 0 or 1) and returns 1 if the row number 8a + 4b + 2c + d is in the list `ONES`, otherwise 0. First print the Karnaugh map of `f` as four lines, one per value of AB in Gray code order (00, 01, 11, 10), each giving the cells for CD in Gray code order (00, 01, 11, 10), in exactly this form: `AB=00: 1 0 0 1` Then read the groups off your map and write `simplified(a, b, c, d)` as a single `return` line using at most five of `and`, `or` and `not` in total. It may not use `ONES` or call `f`. Finally, check it on all sixteen combinations, treating any true result as 1 and any false result as 0, and print `matches: True` if it agrees with `f` on every one, or `matches: False` if not.

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

ONES = [0, 2, 8, 10, 11, 14, 15]

def f(a, b, c, d):
    return 1 if 8 * a + 4 * b + 2 * c + d in ONES else 0

def simplified(a, b, c, d):
    return 0
```

The hint students can ask for: The rows and the columns both go in Gray code order, so neighbours differ by one bit. Print the map first and look at it: find the biggest groups of 1s, remembering that the map wraps round at the edges. Each group gives one AND term; OR them together, then compare against f on all sixteen rows.

A solution

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

ONES = [0, 2, 8, 10, 11, 14, 15]

def f(a, b, c, d):
    return 1 if 8 * a + 4 * b + 2 * c + d in ONES else 0

def simplified(a, b, c, d):
    return (not b and not d) or (a and c)

GRAY = ["00", "01", "11", "10"]
for ab in GRAY:
    cells = []
    for cd in GRAY:
        a, b, c, d = int(ab[0]), int(ab[1]), int(cd[0]), int(cd[1])
        cells.append(str(f(a, b, c, d)))
    print(f"AB={ab}: " + " ".join(cells))

ok = True
for n in range(16):
    a, b, c, d = n >> 3 & 1, n >> 2 & 1, n >> 1 & 1, n & 1
    if (1 if simplified(a, b, c, d) else 0) != f(a, b, c, d):
        ok = False
print(f"matches: {ok}")
```

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