ANSWERS

U9.2 A grid is a graph

BugBotLab

Planning · University · about 35 min

What this lesson is about

Cells are nodes, neighbours are edges, and breadth first search is the shortest path when every step costs the same.

- The grid used from here on
- Choosing the cell size
- Breadth first search
- Four neighbours or eight

Questions

6 marks in all

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

```
rows = ["S..#",
        ".#.#",
        "...G"]
H, W = len(rows), len(rows[0])
start, goal = (0, 0), (2, 3)
came = {start: None}
queue = [start]
expanded = 0
while queue:
    cur = queue.pop(0)
    expanded += 1
    if cur == goal:
        break
    r, c = cur
    for nxt in ((r + 1, c), (r - 1, c), (r, c + 1), (r, c - 1)):
        nr, nc = nxt
        if 0 <= nr < H and 0 <= nc < W and rows[nr][nc] not in "#" and nxt not in came:
            came[nxt] = cur
            queue.append(nxt)
steps, cur = 0, goal
while came[cur] is not None:
    cur = came[cur]
    steps += 1
print(steps, expanded)
```

Answer:

```
5 9
```

Four-connected breadth first finds a 5 step path, the Manhattan distance from (0, 0) to (2, 3), and on this small map every free cell comes off the queue by the time the goal does, 9 including the goal.

2. Breadth first search is run on an eight-connected grid. What is wrong with the path it returns? [1 mark]

☐ **A** It counts a diagonal step as one, the same as a straight step, so the path has fewest steps but need not be shortest

☐ **B** It may not find a path even when one exists

☐ **C** It visits cells more than once and may loop for ever

☐ **D** It returns a path that passes between diagonal cells through a wall

> **Answer:** A. A diagonal step is sqrt(2) cells long and breadth first cannot say so. With unequal edge costs, use Dijkstra.

3. On the U9 grid (5 cm cells), a four-connected path runs from cell (6, 6) to cell (34, 34) with no detours. How long is it, in cm? [1 mark]

> **Answer:** 280. Four-connected paths are Manhattan: 28 cells across plus 28 cells up is 56 cells, and 56 x 5 = 280 cm. The straight line is 198 cm, so the path is about 41 percent longer than it.

4. In the breadth first code, the dictionary came does two jobs. What are they? [1 mark]

☐ **A** It is the visited set, and it records which cell each cell was reached from so the path can be walked back

☐ **B** It stores the queue, and it stores the cost of each cell

☐ **C** It stores the free cells, and it stores the walls

☐ **D** It records the goal, and it counts the steps

> **Answer:** A. A cell is in came once it has been discovered, and following came from the goal back to None recovers the path.

5. On the U9 grid, what is the x coordinate, in cm, of the centre of cell (13, 20)? [1 mark]

> **Answer:** 67.5 (accept within 0.01). The centre of cell (i, j) is (5i + 2.5, 5j + 2.5), so x = 5 x 13 + 2.5 = 67.5 cm.

6. The cell size is made much larger and breadth first now reports no route across the mat. Why? [1 mark]

☐ **A** No cell centre lands in the passage between the walls, so the passage has disappeared from the graph

☐ **B** Breadth first is not complete on coarse grids

☐ **C** The queue overflows with large cells

☐ **D** Larger cells make every edge cost different

> **Answer:** A. The planner is not wrong given the map it was handed. Too large a cell loses narrow passages; too small multiplies the cells by the square.

The task: breadth first across the mat

Build the grid, run breadth first from cell (6, 6) to cell (34, 34) with four neighbours, and print `steps:`, how many moves long the path is, and `expanded:`, how many cells came off the front of the queue before the goal did.

```
from bugbot import *
connect()

CELL = 5.0
N = 40
INFLATE = 8.0
WALLS = [(70.0, 0.0, 8.0, 115.0), (125.0, 85.0, 8.0, 115.0)]
```

The hint students can ask for: Build the 40 by 40 grid described on the page, mark a cell blocked when its centre is inside a wall grown by 8 cm or within 8 cm of the mat edge, then run breadth first from cell (6, 6) to cell (34, 34) with four neighbours per cell. Count the pops, and walk the parents back to get the length.

A solution

```
from bugbot import *
connect()

CELL = 5.0
N = 40
INFLATE = 8.0
WALLS = [(70.0, 0.0, 8.0, 115.0), (125.0, 85.0, 8.0, 115.0)]

def blocked(i, j):
    x, y = i * CELL + CELL / 2, j * CELL + CELL / 2
    if x < INFLATE or y < INFLATE or x > 200 - INFLATE or y > 200 - INFLATE:
        return True
    for ox, oy, ow, oh in WALLS:
        if ox - INFLATE <= x <= ox + ow + INFLATE and oy - INFLATE <= y <= oy + oh +
INFLATE:
            return True
    return False

grid = [[blocked(i, j) for j in range(N)] for i in range(N)]
start, goal = (6, 6), (34, 34)

came = {start: None}
queue = [start]
head = 0
expanded = 0
while head < len(queue):
    cur = queue[head]
    head += 1
    expanded += 1
    if cur == goal:
        break
    for dx, dy in ((1, 0), (-1, 0), (0, 1), (0, -1)):
        nxt = (cur[0] + dx, cur[1] + dy)
        if 0 <= nxt[0] < N and 0 <= nxt[1] < N and not grid[nxt[0]][nxt[1]] and nxt not in
came:
            came[nxt] = cur
            queue.append(nxt)
```

```
path, cur = [], goal
while cur is not None:
    path.append(cur)
    cur = came[cur]
print("steps:", len(path) - 1)
print("expanded:", expanded)
```

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