ANSWERS

U9.4 A* and the heuristic

BugBotLab

Planning · University · about 40 min

What this lesson is about

Guessing what is left to pay, why the guess must never be too high, and why greedy best first is not A*.

- Admissible
- Consistent
- Why greedy best first is not A*
- Weighted A*

Questions

7 marks in all

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

```
import math
CELL = 5.0
goal = (34, 34)
c = (30, 20)
dx, dy = abs(c[0] - goal[0]), abs(c[1] - goal[1])
octile = CELL * (max(dx, dy) + (math.sqrt(2) - 1) * min(dx, dy))
euclid = CELL * math.hypot(dx, dy)
manhattan = CELL * (dx + dy)
print(round(octile, 1), round(euclid, 1), round(manhattan, 1))
```

Answer:

```
78.3 72.8 90.0
```

With dx = 4 and dy = 14 the cheapest eight-connected route on an empty grid is 4 diagonals and 10 straights, 78.3 cm, which octile gives exactly. Euclidean is below it; Manhattan, at 90, overestimates it.

2. A* runs on an eight-connected grid with the Manhattan heuristic dx + dy. Is that heuristic admissible? [1 mark]

- ☐ **A** No: a diagonal step costs sqrt(2) but Manhattan counts it as 2, so it can overestimate the remaining cost
- ☐ **B** Yes: Manhattan distance never exceeds the true cost on any grid
- ☐ **C** Yes, because Manhattan is consistent
- ☐ **D** No, because it is smaller than the Euclidean distance

Answer: A. Manhattan is admissible only on a four-connected grid. On eight neighbours it overestimates, and A* can return a longer route than necessary.

3. What is the difference between greedy best first search and A*? [1 mark]

- ☐ **A** Greedy orders the queue by h alone, dropping g, so it ignores what has already been spent and is not optimal
- ☐ **B** Greedy uses an admissible heuristic and A* does not
- ☐ **C** Greedy expands more cells than A* but finds the same path
- ☐ **D** Greedy uses a stack instead of a priority queue

Answer: A. On this map greedy expands about 250 cells for a 410 cm route; A* expands about 680 for 331 cm. The g term is what remembers the cost so far.

4. Weighted A* with w = 1.5 is run on a map where the optimal path is 331 cm. What is the longest path, in cm, it is guaranteed never to exceed? [1 mark]

> **Answer:** 496.5 (accept within 0.1). With f = g + w h, the path is never worse than w times optimal: 1.5 x 331 = 496.5 cm.

5. Which of these heuristics are admissible for A* on an eight-connected grid? [1 mark]

Tick every answer that is true.

- ☐ **A** h = 0
- ☐ **B** Euclidean distance
- ☐ **C** Octile distance
- ☐ **D** Manhattan distance
- ☐ **E** Twice the octile distance

> **Answer:** A, B, C. Zero, Euclidean and octile never exceed the true remaining cost. Manhattan overestimates diagonals, and doubling octile overestimates everywhere.

6. What does a consistent heuristic buy A* beyond admissibility? [1 mark]

- ☐ **A** f never decreases along a path, so a node's g is final the first time it is popped and nodes never need reopening
- ☐ **B** A* then returns the optimal path, which an admissible heuristic does not guarantee
- ☐ **C** A* then expands fewer cells than greedy best first
- ☐ **D** A* then works with negative edge costs

> **Answer:** A. Admissibility already guarantees optimality. Consistency saves putting improved nodes back on the queue.

7. Why is octile distance a better heuristic than Euclidean distance on an eight-connected grid? [1 mark]

- ☐ **A** It is still admissible and closer to the true cost, so A* expands fewer cells
- ☐ **B** Euclidean distance is not admissible on a grid
- ☐ **C** Octile distance always equals the true cost, even with walls
- ☐ **D** Octile distance makes A* ignore the walls

> **Answer:** A. Octile is the exact cost on an empty grid, as tight as possible without overestimating. Walls only make the true cost larger.

The task: the same answer, less work

Run both searches over the same grid, eight neighbours, plain step costs of 5 cm and 5 root 2 cm, and print `cost:` (which should be identical either way), `dijkstra:` and `a star:`, the number of cells each expanded.

```
from bugbot import *
import heapq
import math
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: Run both searches over the same grid with plain step costs (5 cm straight, 5 root 2 diagonal) and count the expansions of each. A* is Dijkstra with the queue ordered by cost so far plus the octile estimate of what is left. If the two costs differ, the heuristic is overestimating.

A solution

```
from bugbot import *
import heapq
import math
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)]
STEPS = [(1, 0), (-1, 0), (0, 1), (0, -1), (1, 1), (1, -1), (-1, 1), (-1, -1)]

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)

def octile(c):
    dx, dy = abs(c[0] - goal[0]), abs(c[1] - goal[1])
    return CELL * (max(dx, dy) + (math.sqrt(2) - 1) * min(dx, dy))

def search(h):
    """h is the estimate of what is left: return zero from it and this is Dijkstra."""
    g = {start: 0.0}
    queue = [(h(start), start)]
    done = set()
    expanded = 0
    while queue:
        _f, cur = heapq.heappop(queue)
```

```
        if cur in done:
            continue
        done.add(cur)
        expanded += 1
        if cur == goal:
            break
        for dx, dy in STEPS:
            nxt = (cur[0] + dx, cur[1] + dy)
            if not (0 <= nxt[0] < N and 0 <= nxt[1] < N) or grid[nxt[0]][nxt[1]]:
                continue
            step = CELL * math.hypot(dx, dy)
            if g[cur] + step < g.get(nxt, 1e18):
                g[nxt] = g[cur] + step
                heapq.heappush(queue, (g[nxt] + h(nxt), nxt))
    return g[goal], expanded

flat_cost, flat_expanded = search(lambda c: 0.0)
star_cost, star_expanded = search(octile)
print("cost:", round(star_cost, 1))
print("dijkstra:", flat_expanded)
print("a star:", star_expanded)
print("same answer:", abs(flat_cost - star_cost) < 1e-6)
```

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