ANSWERS

# A5.8 A* search

Algorithms and complexity · A level · OCR H446 2.3.1 · about 30 min

## What this lesson is about

g, h and f, open and closed lists, admissible heuristics, tracing A*, and A* against Dijkstra on the mat's grid.

- g, h and f
- The algorithm
- A trace
- When is the answer guaranteed shortest?
- A* on the mat's grid
- Efficiency

## Questions

**6 marks in all**

**1.** In A*, what is f(n)? [1 mark]

- ☐ **A** g(n) + h(n): the cost so far plus the estimated cost to the goal
- ☐ **B** The cost so far only
- ☐ **C** The estimated cost to the goal only
- ☐ **D** The number of edges from the start

**Answer:** A. A* always expands the open vertex with the smallest f.

**2.** What makes a heuristic admissible? [1 mark]

- ☐ **A** It never overestimates the real cost to the goal
- ☐ **B** It always overestimates the cost to the goal
- ☐ **C** It is always exactly 0
- ☐ **D** It is worked out by searching the graph

**Answer:** A. With an admissible heuristic A* is guaranteed to find a shortest route.

**3.** What happens to A* if the heuristic is 0 for every vertex? [1 mark]

- ☐ **A** It behaves exactly like Dijkstra's algorithm
- ☐ **B** It never finds the goal
- ☐ **C** It becomes breadth-first search
- ☐ **D** It finds a route that is too long

**Answer:** A. f is then just g, so it always expands the smallest distance so far.

**4.** On a grid where moves are up, down, left or right, what is the Manhattan distance from square (row 2, column 3) to square (row 7, column 1)? [1 mark]

**Answer:** 7. The difference in rows is 5 and in columns is 2: 5 + 2 = 7.

**5.** What is the name for the list of vertices that A* has found but not yet expanded? [1 mark]

**Answer:** open list. Expanded vertices go on the closed list.

**6.** A robot needs the shortest distance from its base to every charging point on a map. Which algorithm fits best? [1 mark]

- ☐ **A** Dijkstra's algorithm, because one run gives distances to every vertex
- ☐ **B** A*, because it always expands fewer vertices
- ☐ **C** Linear search
- ☐ **D** Bubble sort

> **Answer:** A. A* aims at one destination. Dijkstra's algorithm settles the distance to every vertex in one run.

## The task: round the rough ground

The mat in the starter has rough squares ( `~` ) as well as walls ( `#` ). Entering a normal square ( `.` , `S` or `G` ) costs 1 and entering a rough square costs 3. Moves are up, down, left and right only. Write `a_star(grid, use_heuristic)` that returns a tuple `(cost, expanded)` : the cost of the cheapest route from `S` to `G` , and the number of squares expanded. Use the Manhattan distance as the heuristic when `use_heuristic` is `True` and 0 when it is `False` (which makes it Dijkstra's algorithm). To make the count exact: - always expand the open square with the smallest f; break a tie by the smaller h, and then by the smaller row and then column, which is what a heap of `(f, h, row, col)` tuples does; - a square is **expanded** when it is taken off the open list and closed. Skip (and do not count) a square that is already closed. Stop as soon as `G` is expanded, and count it. Print exactly three lines: - `cost: <n>` - `A* expanded: <n>` - `Dijkstra expanded: <n>`

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

import heapq

grid = [
    "............",
    "............",
    "....~~~~....",
    ".S..~~~~..G.",
    "....~~~~....",
    "....#####...",
    "............",
    "............",
]
```

**The hint students can ask for:** Keep a heap of (f, h, row, col) entries and a dictionary of the best g for each square. Each time round, pop the smallest, skip it if it is already closed, close and count it, and stop if it is G. For each open neighbour, g goes up by the cost of the square you step onto; push the neighbour only when that g beats the best so far. Run the same function twice, with the heuristic on and off.

## A solution

```
from bugbot import *
connect()

import heapq

grid = [
    "............",
    "............",
    "....~~~~....",
    ".S..~~~~..G.",
    "....~~~~....",
    "....#####...",
    "............",
    "............",
]

def a_star(grid, use_heuristic):
    for r, row in enumerate(grid):
        if "S" in row:
```

```
            start = (r, row.index("S"))
        if "G" in row:
            goal = (r, row.index("G"))
    def h(r, c):
        return abs(r - goal[0]) + abs(c - goal[1]) if use_heuristic else 0
    best = {start: 0}
    heap = [(h(*start), h(*start), start[0], start[1])]
    closed = set()
    while heap:
        f, hh, r, c = heapq.heappop(heap)
        if (r, c) in closed:
            continue
        closed.add((r, c))
        if (r, c) == goal:
            return best[goal], len(closed)
        for nr, nc in [(r - 1, c), (r + 1, c), (r, c - 1), (r, c + 1)]:
            if 0 <= nr < len(grid) and 0 <= nc < len(grid[0]) and grid[nr][nc] != "#" and
(nr, nc) not in closed:
                step = 3 if grid[nr][nc] == "~" else 1
                g = best[(r, c)] + step
                if g < best.get((nr, nc), float("inf")):
                    best[(nr, nc)] = g
                    heapq.heappush(heap, (g + h(nr, nc), h(nr, nc), nr, nc))
    return None, len(closed)

cost, expanded = a_star(grid, True)
print("cost:", cost)
print("A* expanded:", expanded)
print("Dijkstra expanded:", a_star(grid, False)[1])
```

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