WORKSHEET

A5.8 A* search

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

BugBotLab

Name _____ Class _____ Date _____

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

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

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

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]

_____

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

_____

_____

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

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.",
    "....~~~~....",
    "....#####...",
    "............",
    "............",
]
```

Plan your program here, then type it in and press Run.

Do it on the robot

www.bugbotlab.com/learn/a5-8-a-star-search/

The simulator checks it and tells you when it passes. Nothing to install, no account.

Challenges

1. Is the Manhattan distance still admissible if moving onto a normal square cost 0.5? What would you change?
2. Multiply the heuristic by 3 and run again. Does A* still find the cheapest route? How many squares does it expand?

3. Allow diagonal moves costing 1.4. Why is the Manhattan distance no longer admissible, and what heuristic could you use instead?