WORKSHEET

U9.3 Dijkstra and the cost of a step

Planning · University · about 35 min

Name _____ Class _____ Date _____

What this lesson is about

When steps cost different amounts, the cheapest route is not the shortest one, and a queue sorted by cost finds it.

- The algorithm
- A cost map with clearance in it
- Uniform cost search, and the name

Questions

6 marks in all

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

```
import heapq
EDGES = {"S": [("A", 1), ("B", 2)], "A": [("G", 5)], "B": [("C", 2)], "C": [("G", 1)], "G":
[]}
best, came, done = {"S": 0}, {"S": None}, set()
queue = [(0, "S")]
while queue:
    cost, cur = heapq.heappop(queue)
    if cur in done:
        continue
    done.add(cur)
    if cur == "G":
        break
    for nxt, w in EDGES[cur]:
        price = cost + w
        if price < best.get(nxt, 1e9):
            best[nxt] = price
            came[nxt] = cur
            heapq.heappush(queue, (price, nxt))
path, n = [], "G"
while n is not None:
    path.append(n)
    n = came[n]
print(best["G"], "".join(reversed(path)))
```

2. In Dijkstra's algorithm, when is a node's cost final? [1 mark]

- ☐ **A** The first time it is popped off the priority queue
- ☐ **B** The first time it is pushed onto the priority queue
- ☐ **C** When all of its neighbours have been pushed
- ☐ **D** When the goal is reached

3. Why does Dijkstra's algorithm give wrong answers on a graph with a negative edge cost? [1 mark]

- ☐ **A** A node popped as final could later be reached more cheaply through the negative edge
- ☐ **B** The priority queue cannot store negative numbers
- ☐ **C** Negative costs make the graph disconnected
- ☐ **D** The lazy deletion trick removes the negative edges

4. With the U9.3 cost map, a robot takes a diagonal step on the 5 cm grid into a cell whose centre is 12 cm from a wall. What does the step cost, to two decimal places? [1 mark]

---

5. A cell is already on the queue and a cheaper route to it is found. What do most implementations do? [1 mark]

- ☐ **A** Push a second entry with the new cost, and skip the old one when it is popped because the node is already done
- ☐ **B** Search the heap for the old entry and change its cost
- ☐ **C** Ignore the cheaper route, since the node was pushed first
- ☐ **D** Restart the search from the start

6. Adding a clearance penalty to the cost map changed the route to run down the middle of the corridor. What changed in the algorithm? [1 mark]

- ☐ **A** Nothing: only the cost function changed, and the cheapest path under the new costs is a different route
- ☐ **B** Dijkstra switched to a greedy search
- ☐ **C** The heuristic became admissible
- ☐ **D** The graph gained extra edges down the middle

The task: the cheapest way across

Same grid as U9.2, now with eight neighbours and the cost map above. Print `cost:`, the total cost of the cheapest path, and `min gap:`, the smallest distance from any cell on that path to a wall.

```
from bugbot import *
import heapq
import math
connect()

CELL = 5.0
N = 40
INFLATE = 8.0
NEAR = 20.0
WALLS = [(70.0, 0.0, 8.0, 115.0), (125.0, 85.0, 8.0, 115.0)]
```

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

Do it on the robot

www.bugbotlab.com/learn/u9-3-dijkstra/

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

Challenges

1. Run it again with the penalty turned off (multiplier 1 everywhere) and compare both the cost and the minimum gap. How much did the clearance cost you in distance?
2. Make the penalty smooth: `1 + 4 * max(0, 1 - gap/25)`. Does the path move further from the walls, or just sit differently?
3. Add a cost for turning by penalising a step whose direction differs from the one before it. What does that need you to change about what a node is?