# Finding the Shortest Path **Greedy Best-First Search**

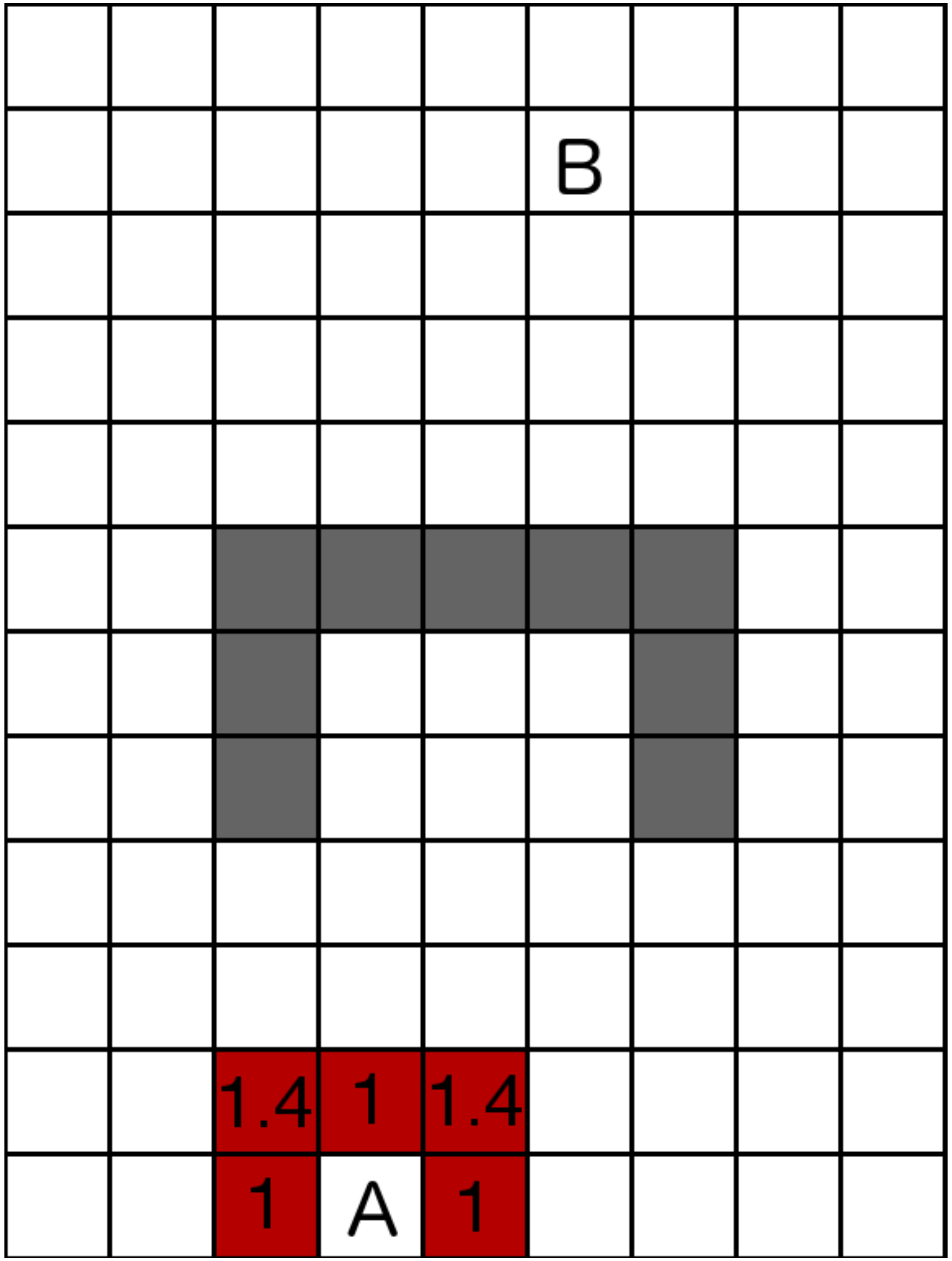
Anton Gerdelan <gerdela@scss.tcd.ie>

# Problem 1: Optimal Path

- Many paths possible from **A** to **B** in some graph **G**
- Adding total cost of all weights in a path gives its length
- Shortest or **optimal path(s)** have smallest overall cost
- Some search algorithms will find an optimal path
    - Dijkstra
    - Breadth-First Search

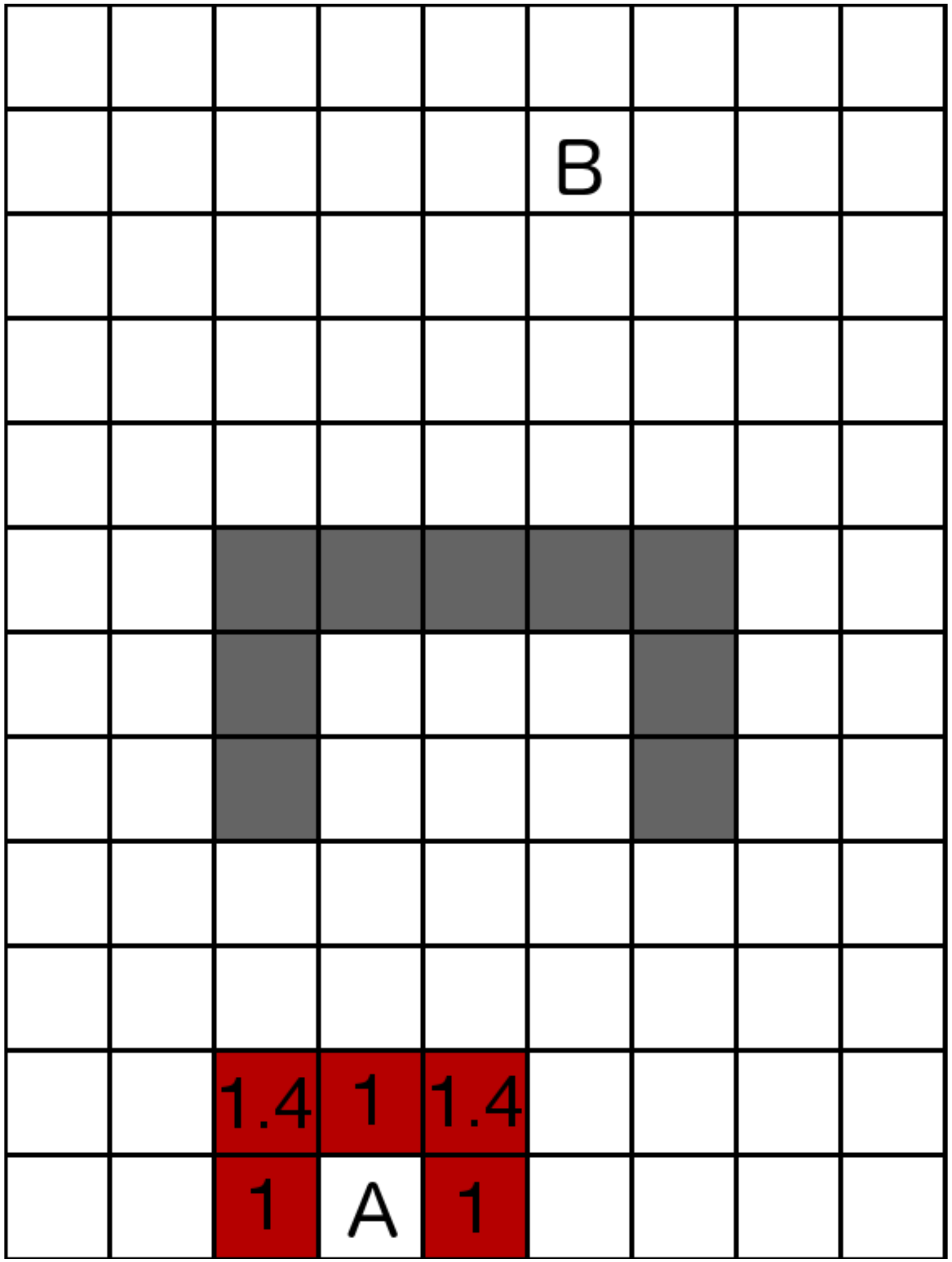
# Problem 2: Computation

- Mathematics usually interested in proof of path
- For real applications we want to compute a path quickly
    - Video games, self-driving cars, electricity budget…
- Some algorithms are not efficient
    - Breadth-First Search (equal frontier in all directions)
    - Dijkstra - only somewhat guided towards goal



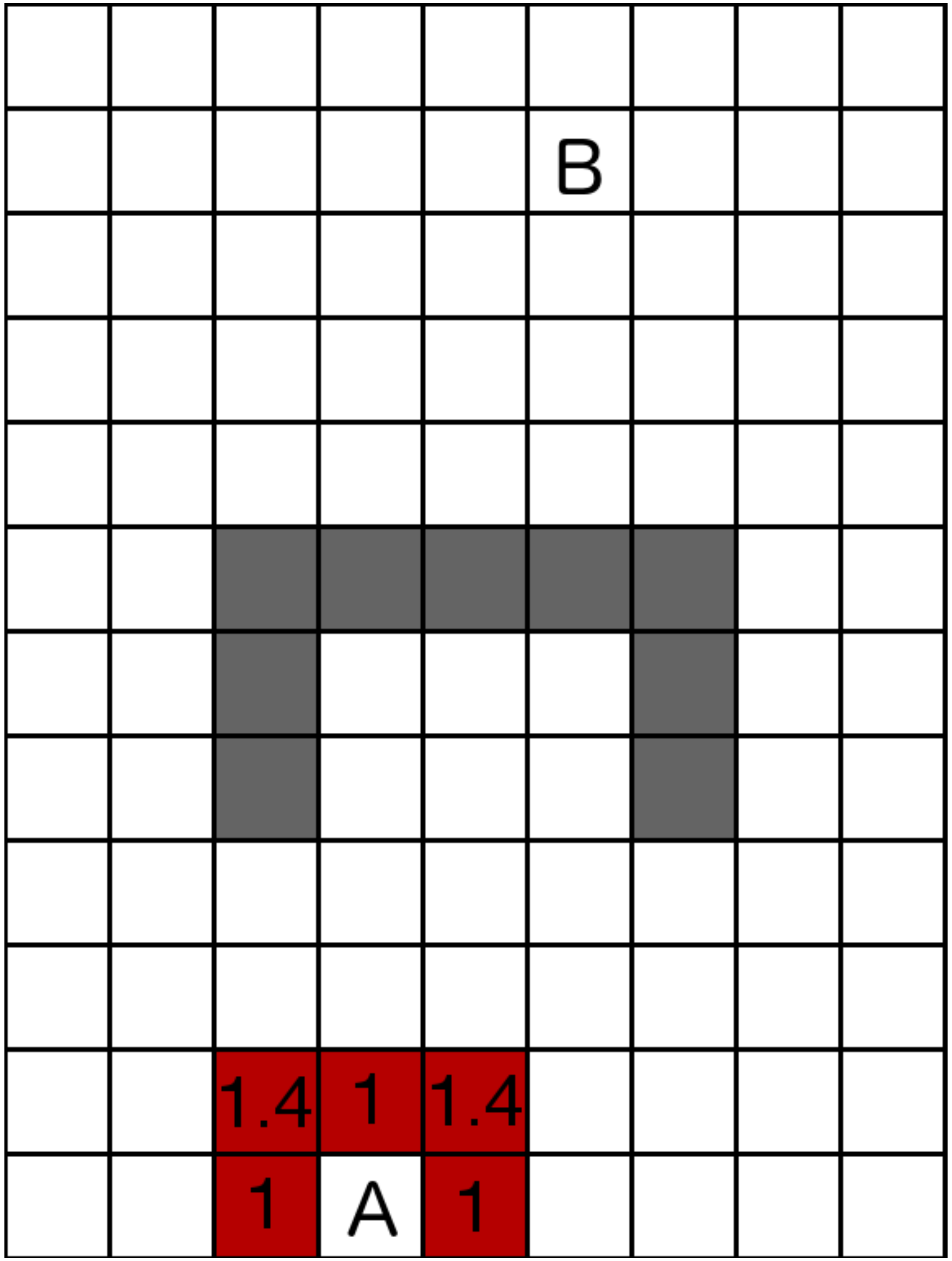
*adjacent edge weights*

- Which nodes will
  - DFS visit next?
  - BFS visit next?
  - Dijkstra's visit next?

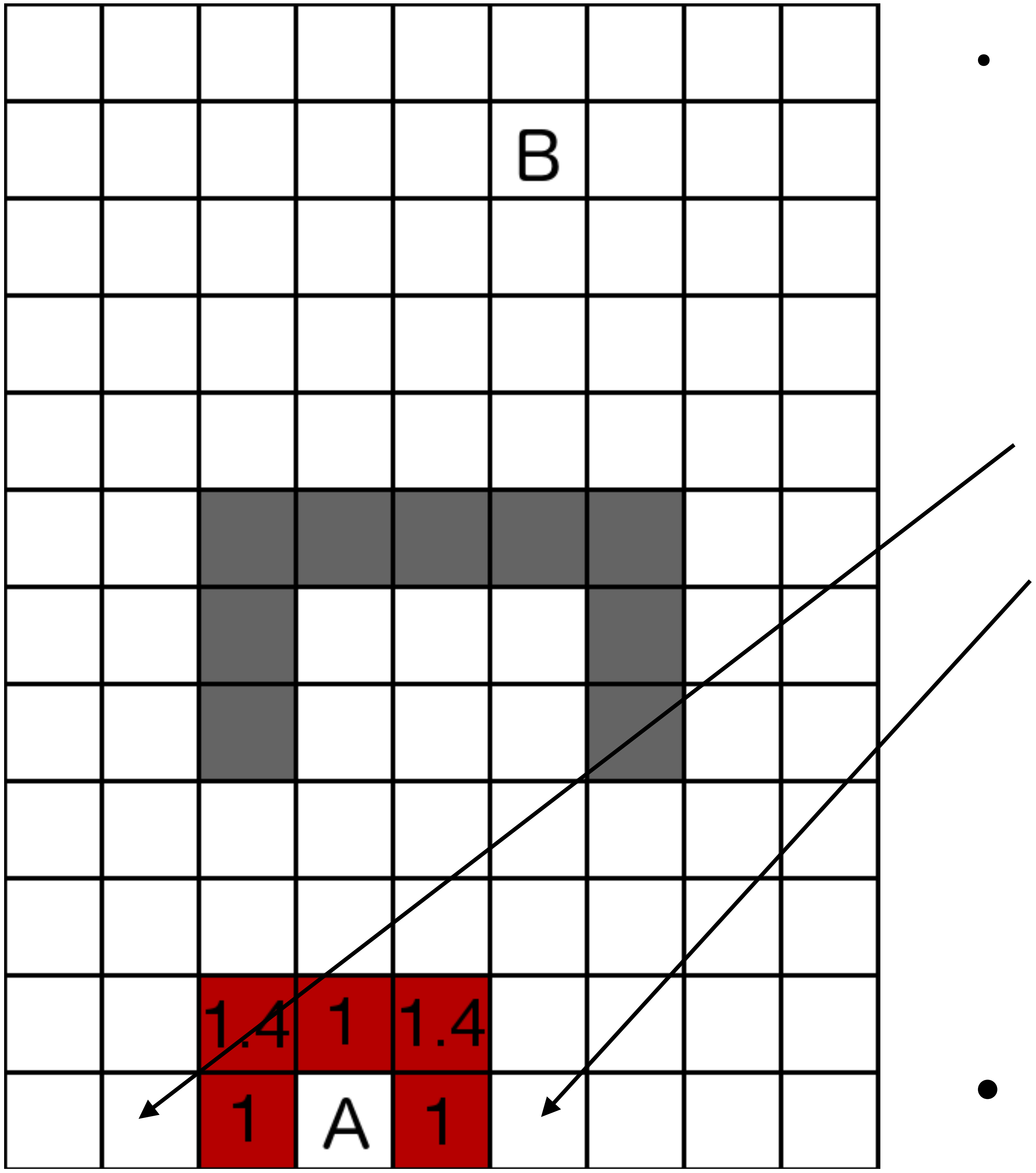


*adjacent edge weights*

- **Q 1.** How can we write an algorithm that **prefers** edges more likely to be on the shortest path?
- *Suggestions?*



- Greedy Best-First Search
  - what does **greedy** mean again for algorithms?
  - create a **heuristic** value to rank choices
  - heuristic is a guess cost
  - usually **pessimistic** - why?



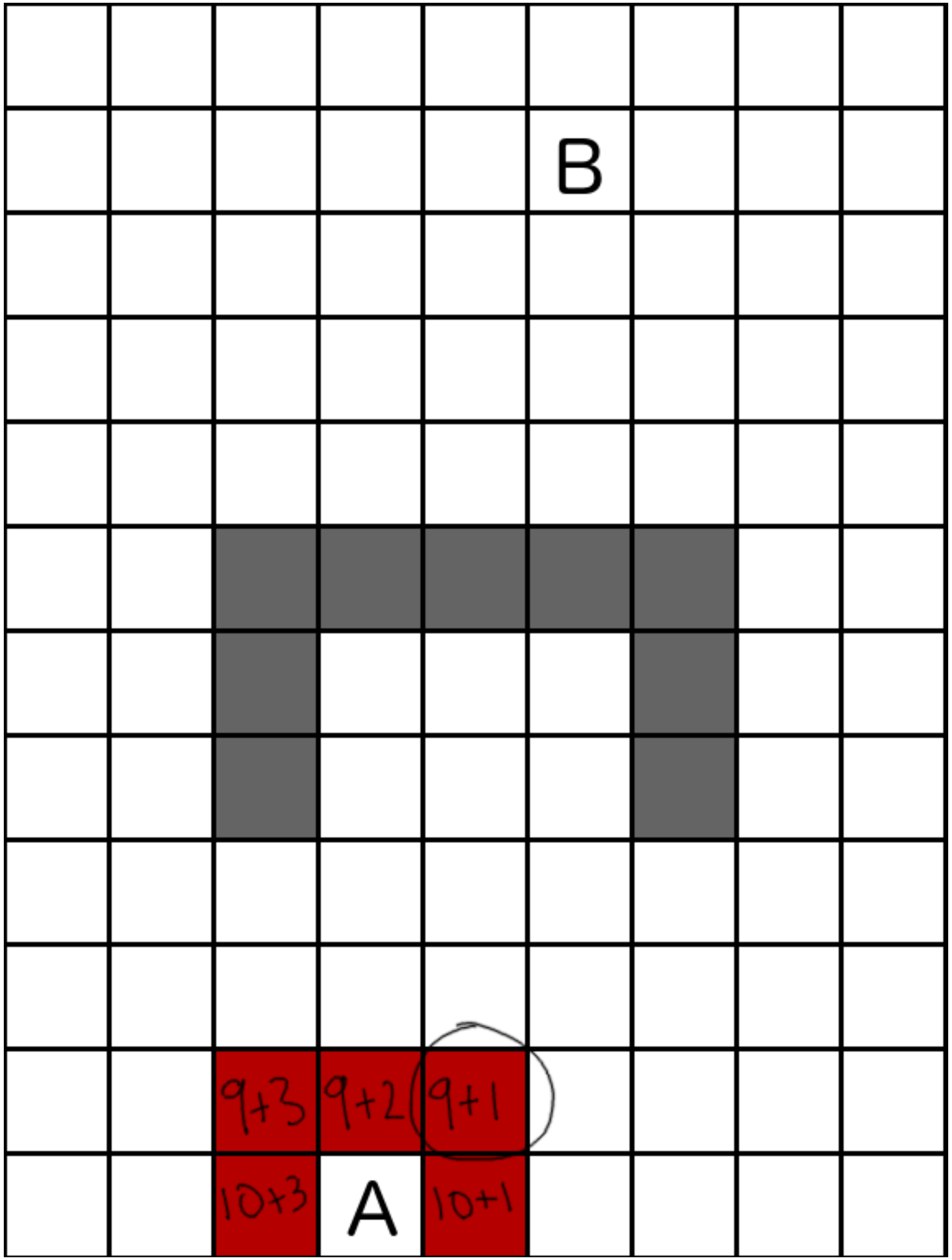
- **Manhattan Distance**
  - city blocks across + city blocks up
  - 4 + 10 = 14
  - 0 + 10 = 10
  - useful heuristic
    - simple
    - pessimistic
- alt: "As the crow flies"
  - why is this worse?

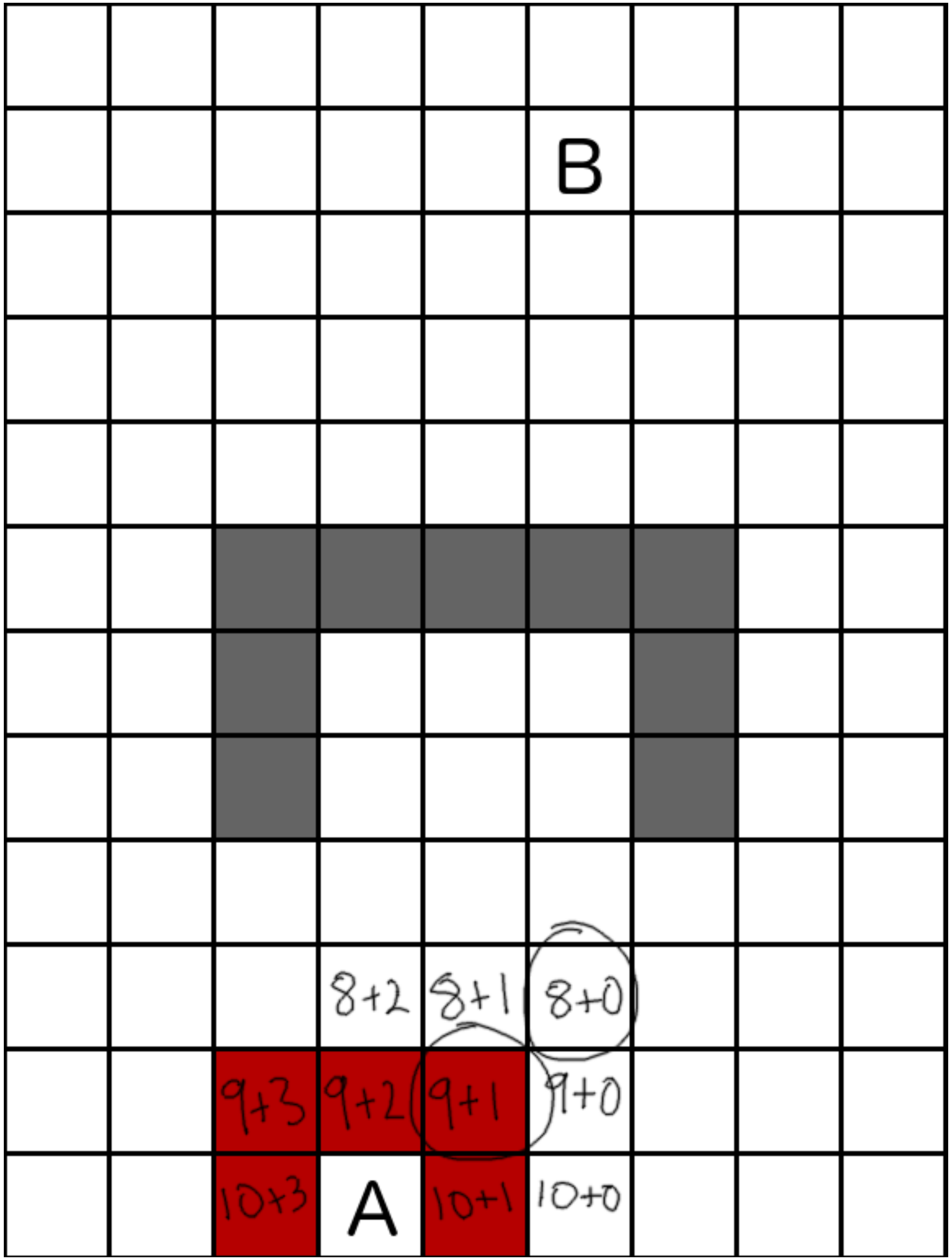
# Greedy Best-First Search

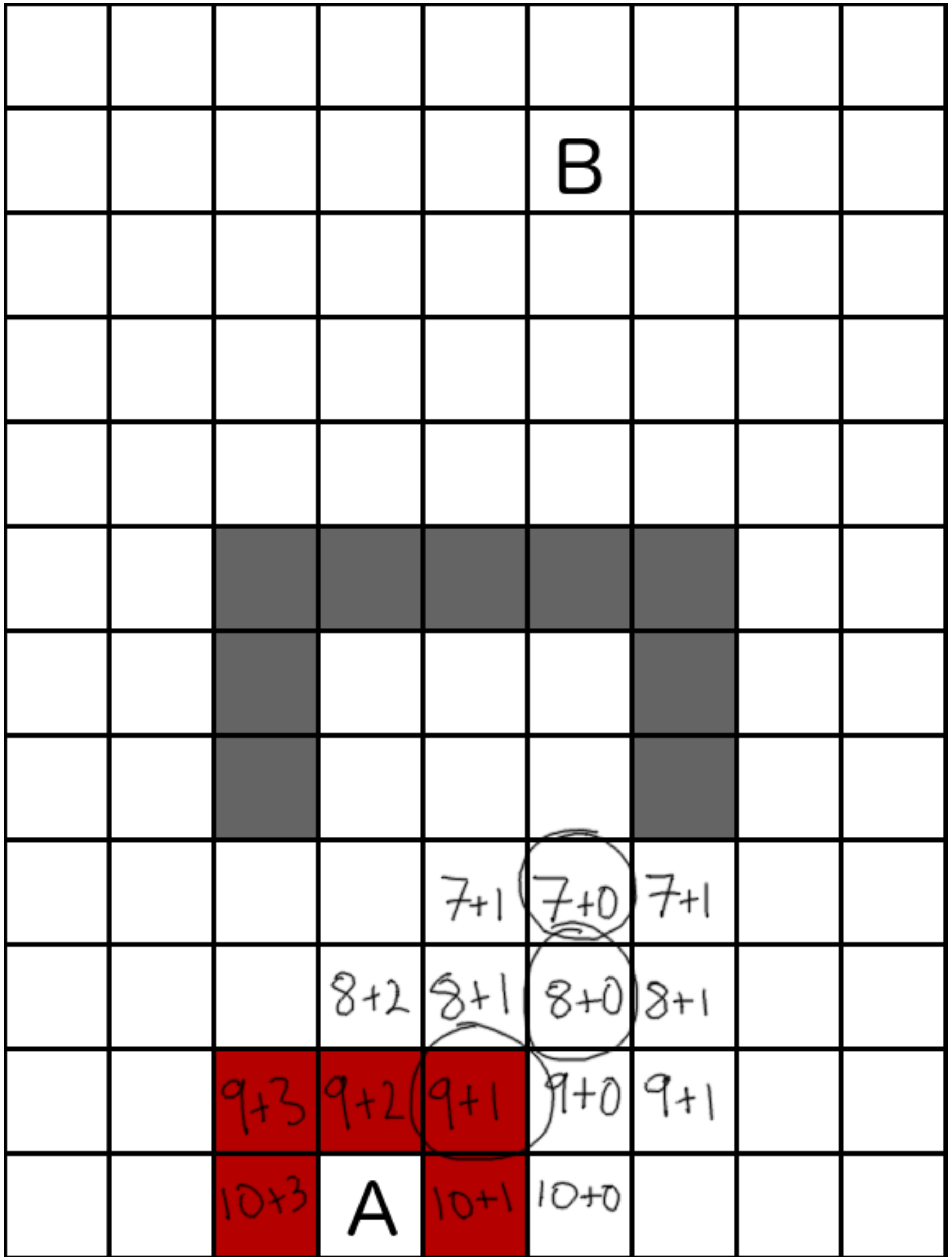
- Like Breadth-First Search except…
- queue of choices are ranked using a heuristic
- **priority queue** - insertion sort or a heap ADT?
- the parent stays in the queue so that it can back-track
- stops when goal state found
    - **Q. why is this unusual?**

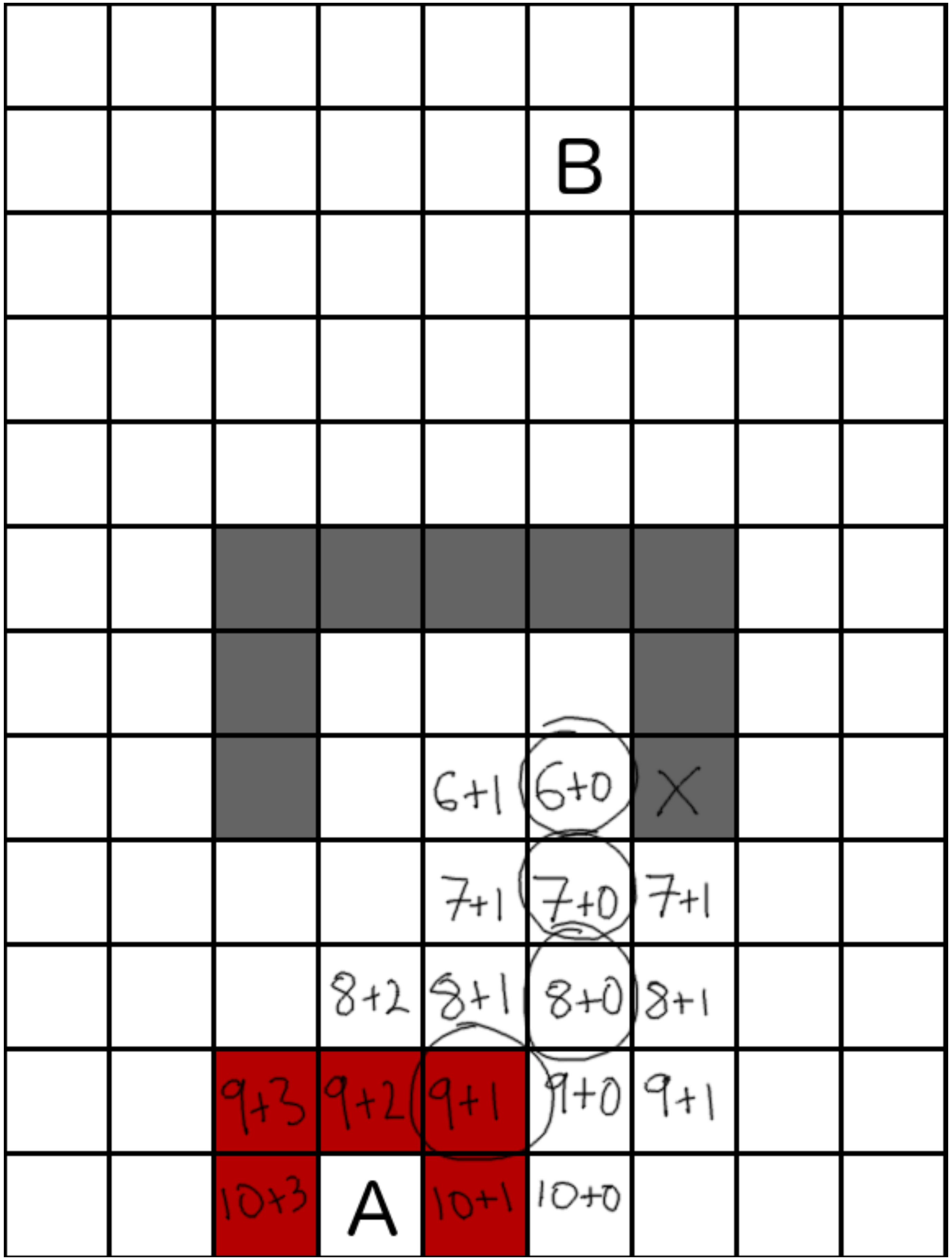
# Greedy Best-First Search

- usually many fewer nodes visited than BFS and Dijkstra
- does not guaranty a shortest path like Dijkstra's
- vulnerable to **local maxima** traps

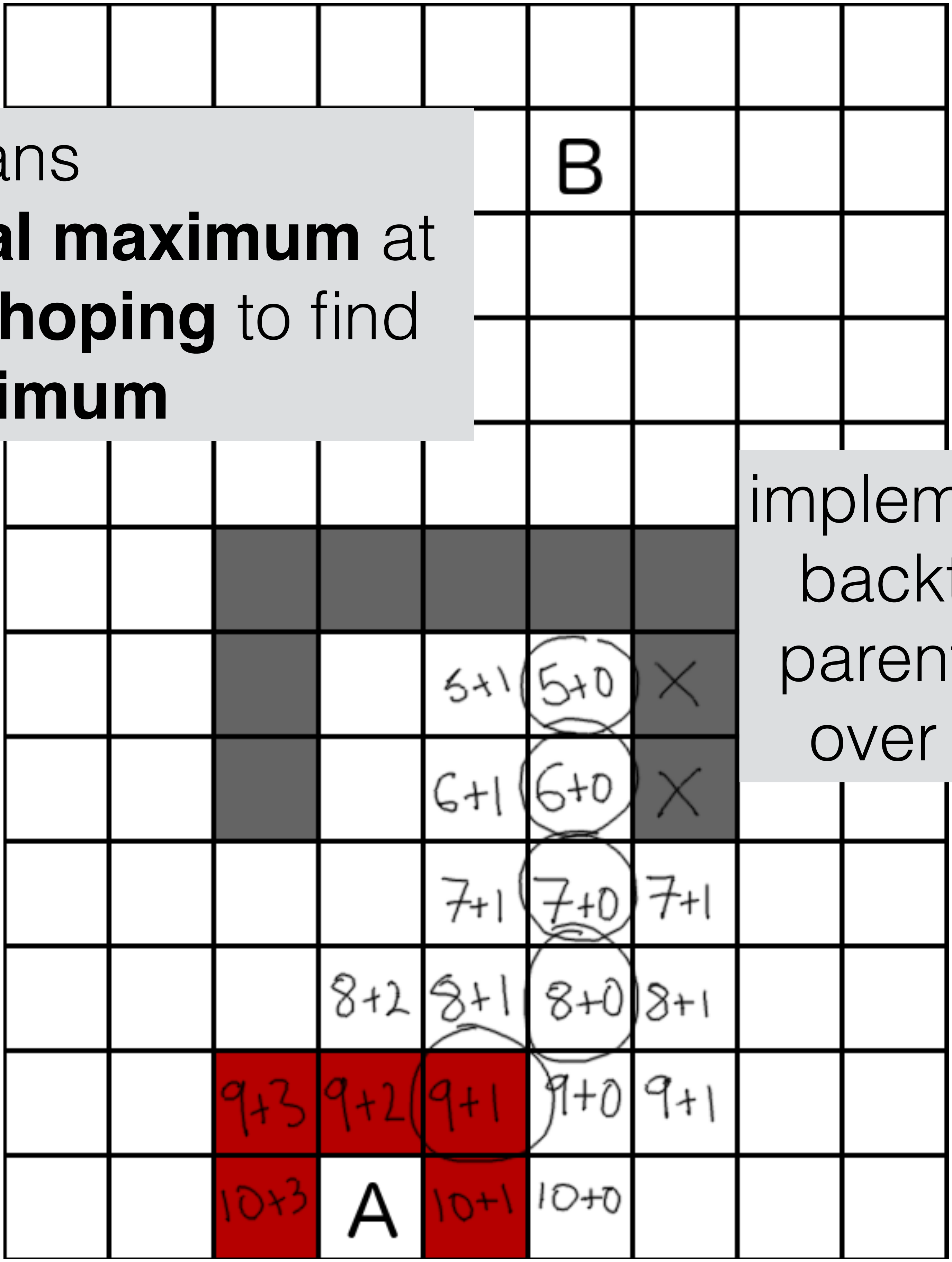

B
9+3
9+2
9+1
10+3
A
10+1


B
8+2
8+1
8+0
9+3
9+2
9+1
9+0
10+3
A
10+1
10+0

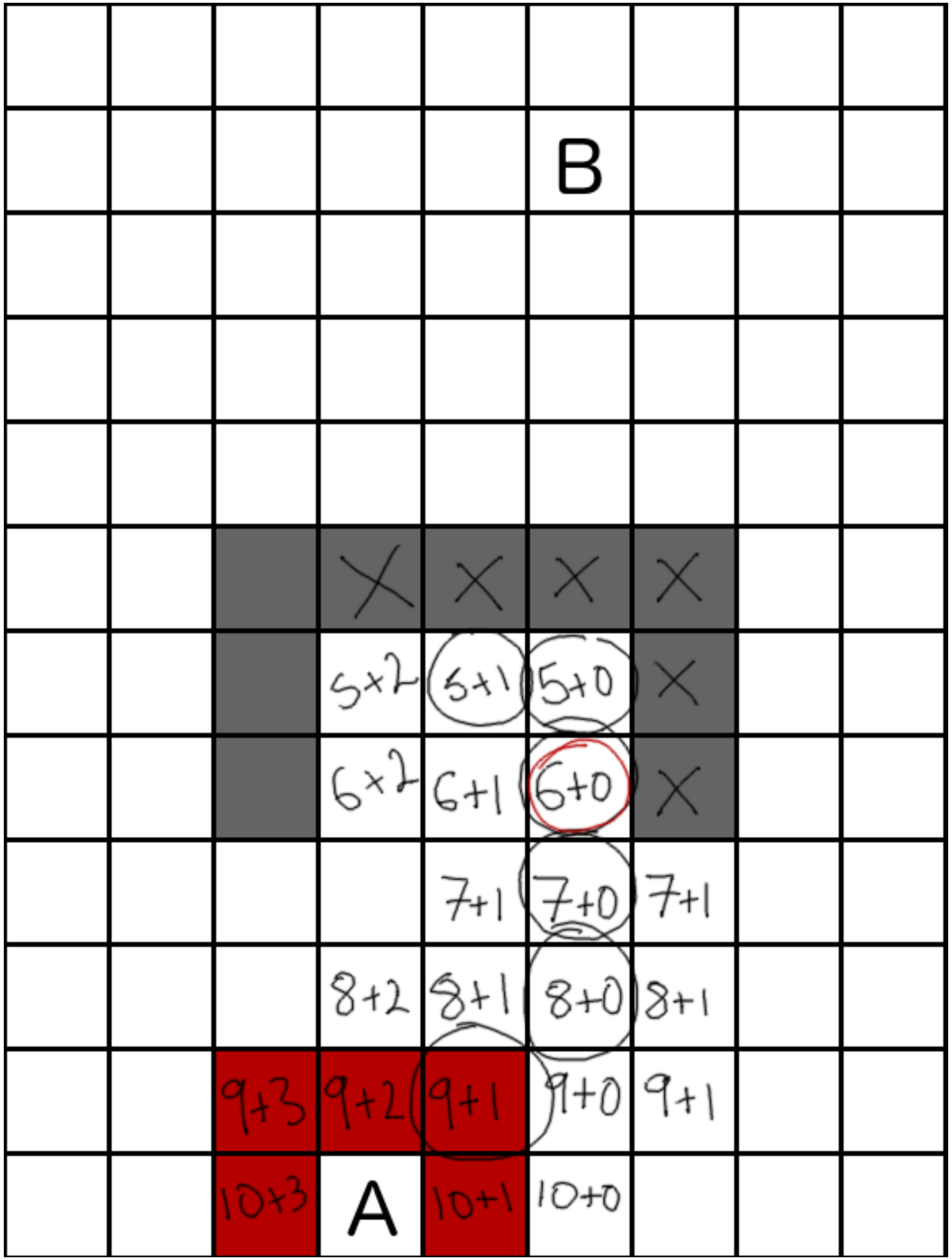

B
7+1
7+0
7+1
8+2
8+1
8+0
8+1
9+3
9+2
9+1
9+0
9+1
10+3
A
10+1
10+0


B
6+1
6+0
X
7+1
7+0
7+1
8+2
8+1
8+0
8+1
9+3
9+2
9+1
9+0
9+1
10+3
A
10+1
10+0

**greedy** means choose **local maximum** at each stage **hoping** to find **global maximum**



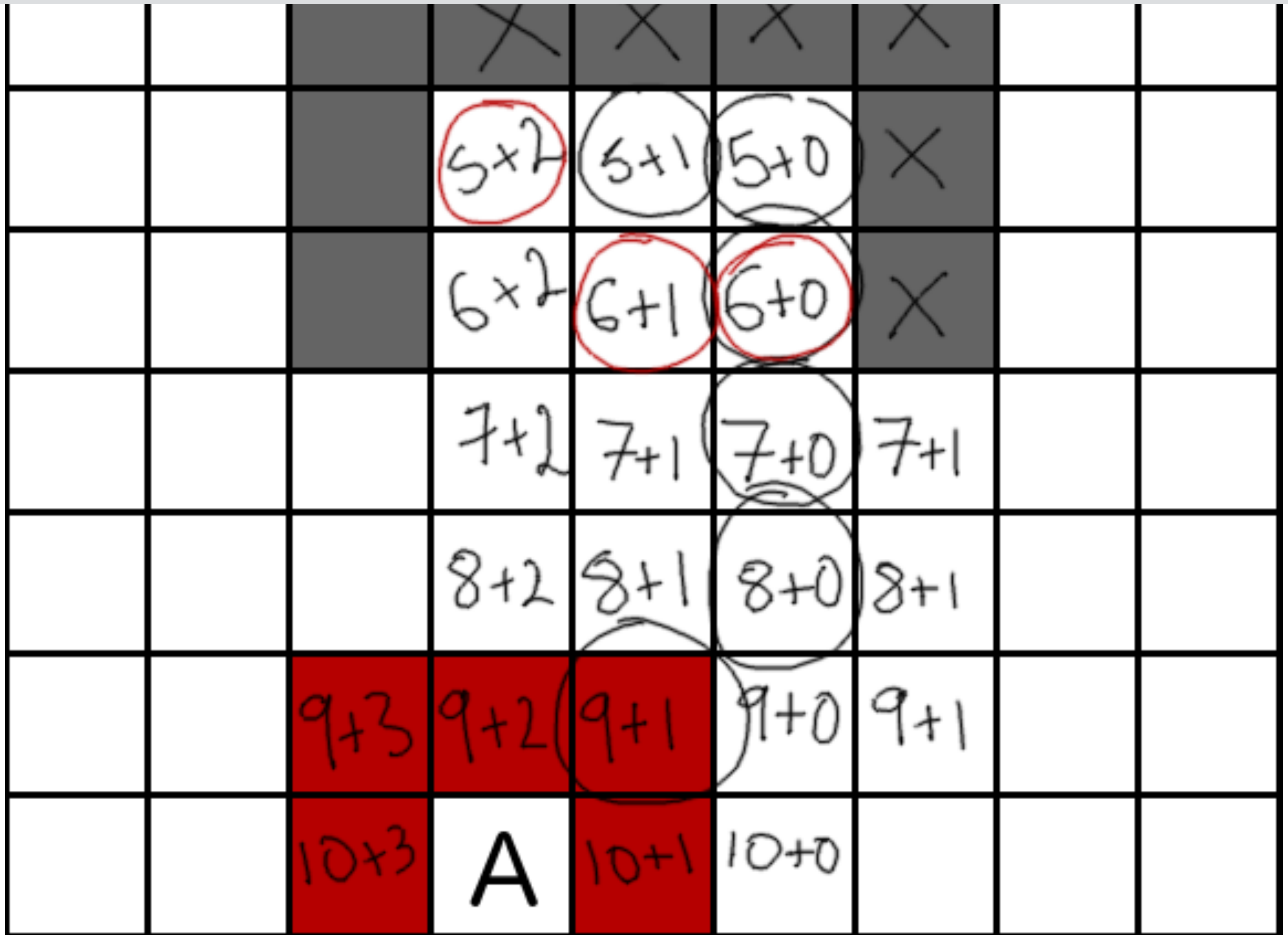
implementation may backtrack here if parent has priority over equal child



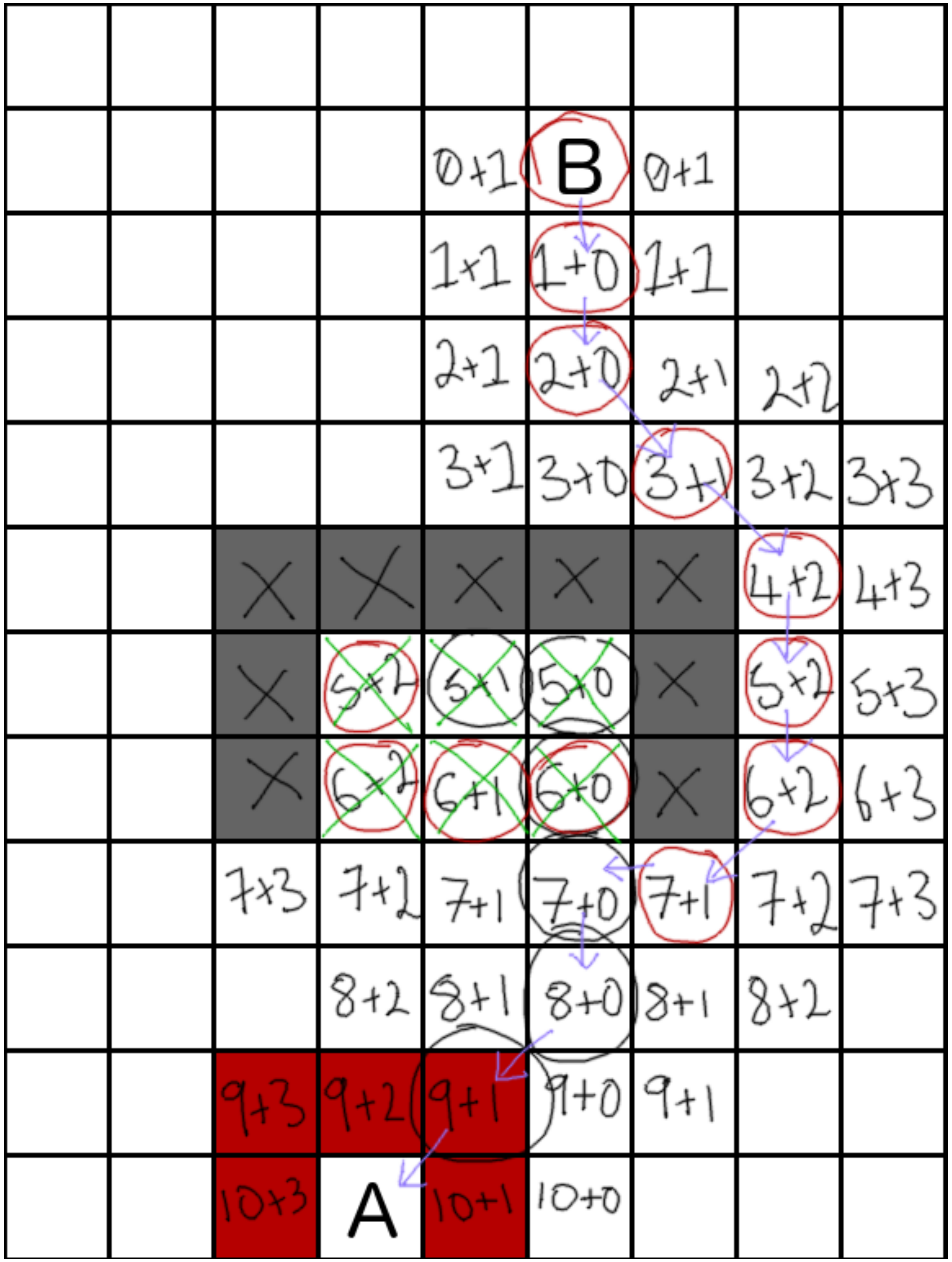
back-track
to parent

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | |
| | | | | | B | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | ▨ | X | X | X | X | | |
| | | ▨ | 5+2 | 5+1 | 5+0 | X | | |
| | | ▨ | 6+2 | 6+1 | 6+0 | X | | |
| | | | | 7+1 | 7+0 | 7+1 | | |
| | | | 8+2 | 8+1 | 8+0 | 8+1 | | |
| | | 9+3 | 9+2 | 9+1 | 9+0 | 9+1 | | |
| | | 10+3 | A | 10+1 | 10+0 | | | |

- lots of back-tracking in maxima traps (dead-ends)
- each node **stores its parent** to allow back-track
- investigated nodes can be flagged to prevent infinite loops - the **closed list**
- frontier is the **open list**



- at halt work backwards through parents to get path
- wasted time in trap
- very few nodes are investigated overall
- BFS would have visited nearly every node



| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | |
| | | | | 0+1 | B | 0+1 | | |
| | | | | 1+1 | 1+0 | 1+1 | | |
| | | | | 2+1 | 2+0 | 2+1 | 2+2 | |
| | | | | 3+1 | 3+0 | 3+1 | 3+2 | 3+3 |
| | | X | X | X | X | X | 4+2 | 4+3 |
| | | X | 5+2 | 5+1 | 5+0 | X | 5+2 | 5+3 |
| | | X | 6+2 | 6+1 | 6+0 | X | 6+2 | 6+3 |
| | | 7+3 | 7+2 | 7+1 | 7+0 | 7+1 | 7+2 | 7+3 |
| | | | 8+2 | 8+1 | 8+0 | 8+1 | 8+2 | |
| | | 9+3 | 9+2 | 9+1 | 9+0 | 9+1 | | |
| | | 10+3 | A | 10+1 | 10+0 | | | |

# Greedy Best-First Search

- add a **heuristic** to Breadth-First Search prioritise strongly
- narrows frontier
- finds path to goal in far fewer steps
- path may not be the shortest path
- **greedy** = short-sighted
    - vulnerable to **local maxima** traps