Heuristic Search

The search techniques we have seen so far...

- Breadth first search
- Uniform cost search
- Depth first search
- Depth limited search
- Iterative Deepening
- Bi-directional Search

uninformed search
blind search

...are all too slow for most real world problems

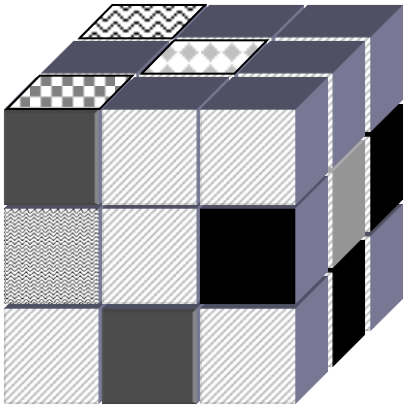
Sometimes we can tell that some states appear better that others...

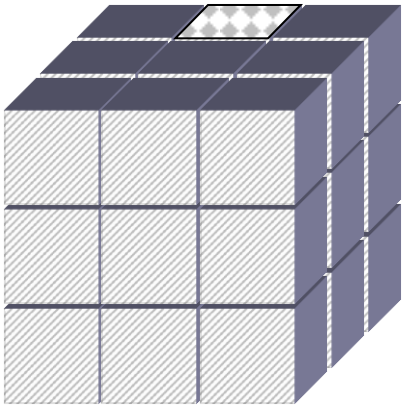
7	8	4
3	5	1
6	2	

1	2	3
4	5	6
7		8

FWD
C

D
FW C





...we can use this knowledge of the relative merit of states to guide search

Heuristic Search (informed search)

A **Heuristic** is a function that, when applied to a state, returns a number that is an estimate of the merit of the state, with respect to the goal.

In other words, the heuristic tells us approximately how far the state is from the goal state*.

Note we said “approximately”. Heuristics might underestimate or overestimate the merit of a state. But for reasons which we will see, heuristics that *only* underestimate are very desirable, and are called admissible.

*I.e Smaller numbers are better

Heuristics for 8-puzzle I

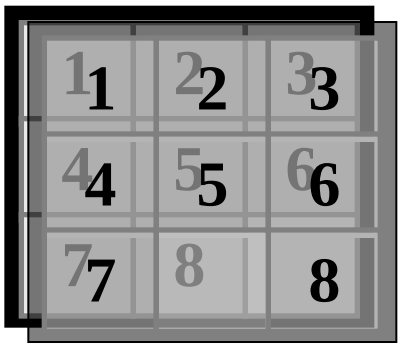
- The number of **misplaced tiles** (not including the blank)

Current State

1	2	3
4	5	6
7		8

Goal State

1	2	3
4	5	6
7	8	



N	N	N
N	N	N
N	Y	

In this case, only "**8**" is misplaced, so the heuristic function evaluates to 1.

In other words, the heuristic is *telling* us, that it *thinks* a solution might be available in just 1 more move.

Notation: $h(n)$ h(current state) = 1

Heuristics for 8-puzzle II

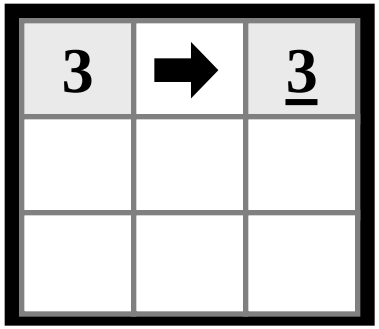
•The **Manhattan Distance** (not including the blank)

Current State

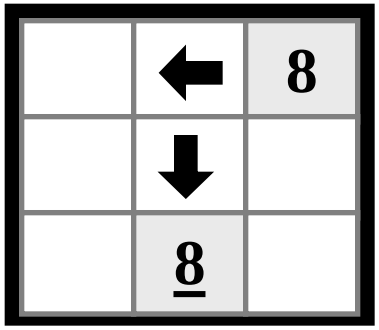
3	**2**	**8**
4	**5**	**6**
7	**1**	

Goal State

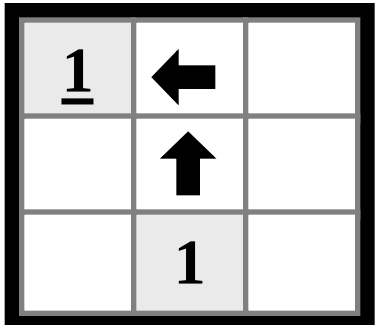
1	**2**	**3**
4	**5**	**6**
7	**8**	



2 spaces



3 spaces



3 spaces

Total 8

In this case, only the "**3**", "**8**" and "**1**" tiles are misplaced, by 2, 3, and 3 squares respectively, so the heuristic function evaluates to 8.

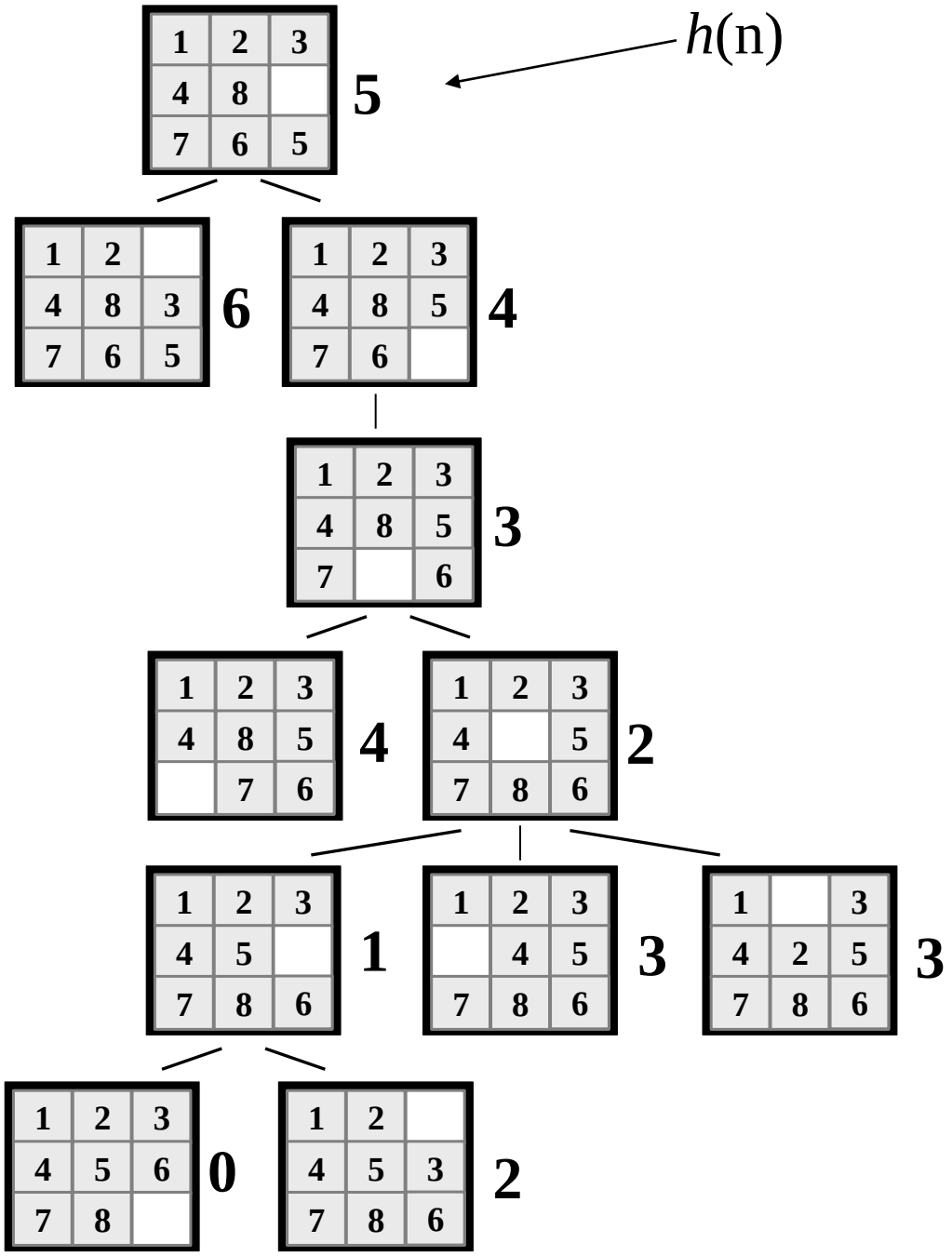
In other words, the heuristic is *telling* us, that it *thinks* a solution is available in just 8 more moves.

Notation: $h(n)$ $h(\text{current state}) = 8$

We can use heuristics to guide “hill climbing” search.

In this example, the Manhattan Distance heuristic helps us quickly find a solution to the 8-puzzle.

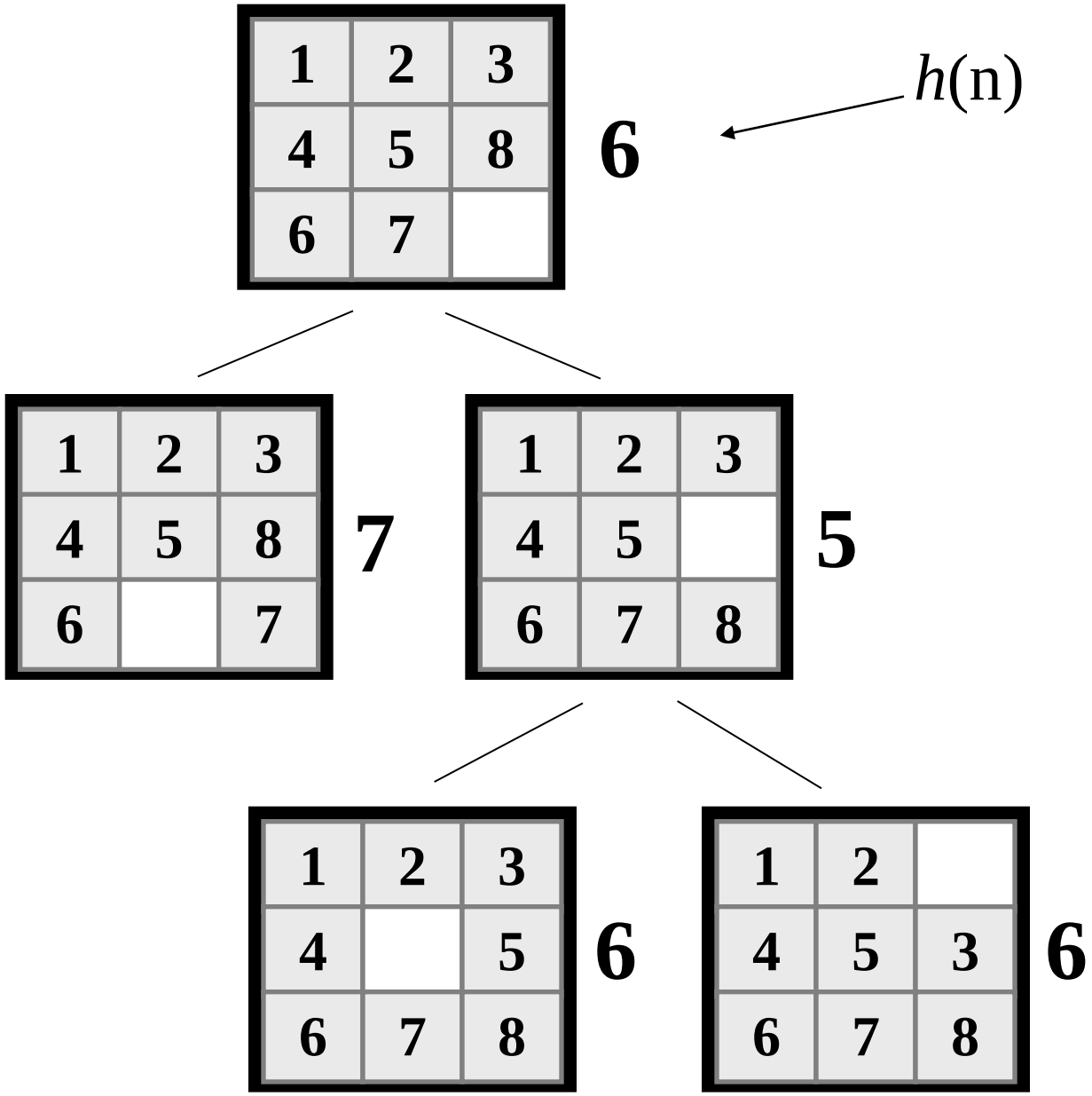
But “hill climbing has a problem...”



In this example, hill climbing does not work!

All the nodes on the fringe are taking a step "backwards" (local minima)

Note that this puzzle *is* solvable in just 12 more steps.



We have seen two interesting algorithms.

Uniform Cost

- Measures the cost to each node.
- Is optimal and complete!
- Can be very slow.

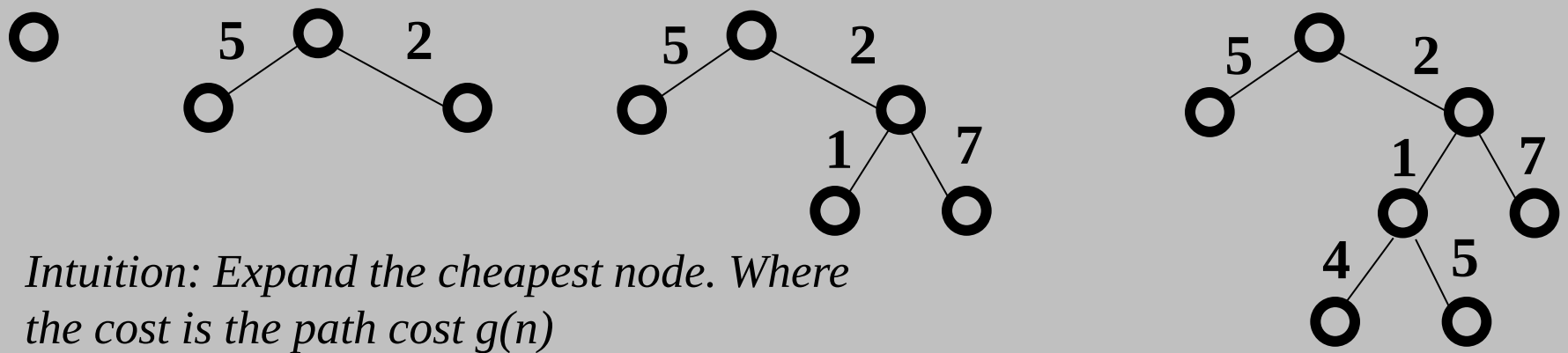
Hill Climbing

- Estimates how far away the goal is.
- Is neither optimal nor complete.
- Can be very fast.

Can we combine them to create an optimal and complete algorithm that is also very fast?

Uniform Cost Search

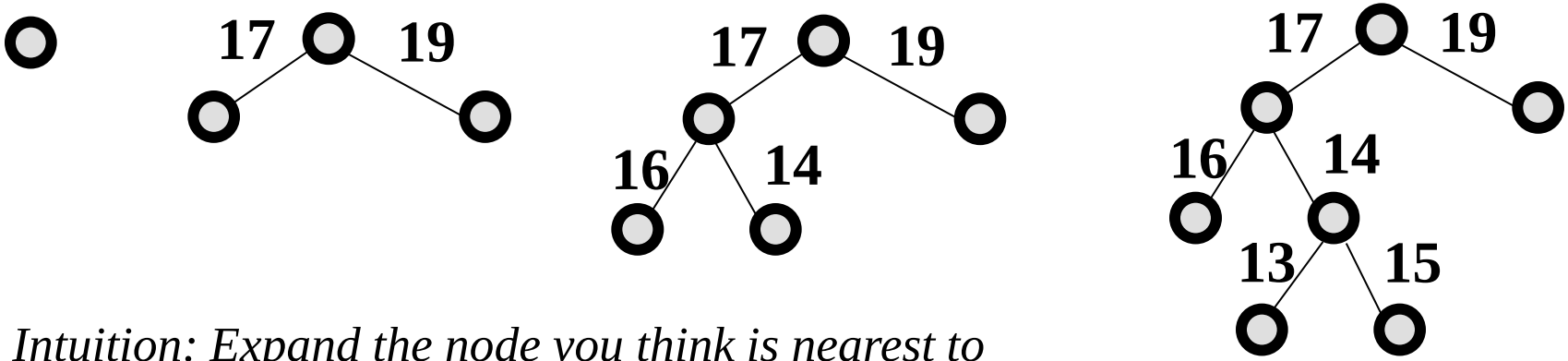
Enqueue nodes in order of cost



Intuition: Expand the cheapest node. Where the cost is the path cost g(n)

Hill Climbing Search

Enqueue nodes in order of estimated distance to goal



Intuition: Expand the node you think is nearest to goal. Where the estimate of distance to goal is h(n)

The A* Algorithm ("A-Star")

Enqueue nodes in order of estimate cost to goal, *f*(n)

g(n) is the cost to get to a node.
h(n) is the estimated distance to the goal.

$f(n) = g(n) + h(n)$

We can think of *f*(n) as the estimated cost of the cheapest solution that goes through node n

Note that we can use the general search algorithm we used before. All that we have changed is the queuing strategy.

If the heuristic is optimistic, that is to say, it never overestimates the distance to the goal, then…

A* is optimal and complete!

Informal *proof* outline of A* completeness

• Assume that every operator has some minimum positive cost, *epsilon* .

• Assume that a goal state exists, therefore some finite set of operators lead to it.

•Expanding nodes produces paths whose actual costs increase by at least epsilon each time. Since the algorithm will not terminate until it finds a goal state, it must expand a goal state in finite time.

Informal *proof* outline of A* optimality

• When A* terminates, it has found a goal state

• All remaining nodes have an estimate cost to goal (*f*(n)) greater than or equal to that of goal we have found.

•Since the heuristic function was optimistic, the actual cost to goal for these other paths can be no better than the cost of the one we have already found.

How fast is A*?

A* is the fastest search algorithm. That is, for any given heuristic, no algorithm can expand fewer nodes than A*.

How fast is it? Depends of the quality of the heuristic.

•If the heuristic is useless (ie *h*(n) is hardcoded to equal 0), the algorithm degenerates to uniform cost.

•If the heuristic is perfect, there is no real search, we just march down the tree to the goal.

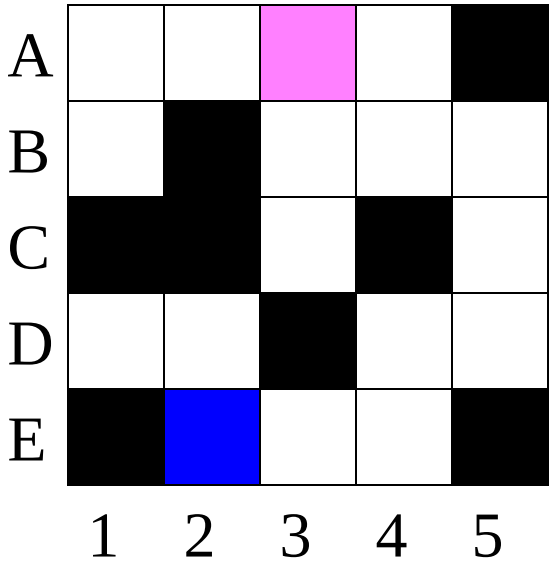
Generally we are somewhere in between the two situations above. The time taken depends on the quality of the heuristic.

What is A*'s space complexity?

A* has worst case $O(b^d)$ space complexity, but an iterative deepening version is possible (IDA*)

A Worked Example: Maze Traversal

Problem: To get from square **A3** to square **E2**, one step at a time, avoiding obstacles (black squares).

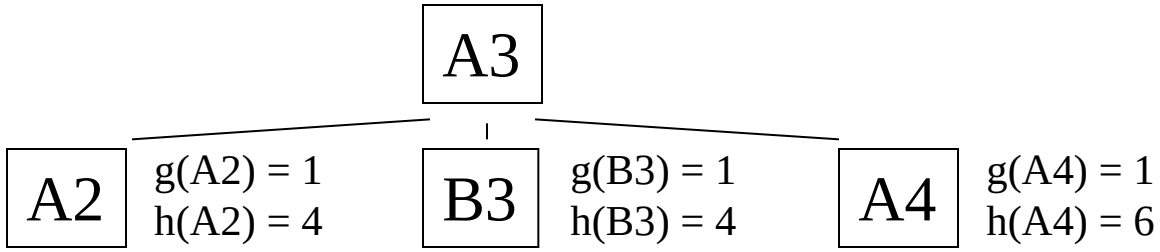


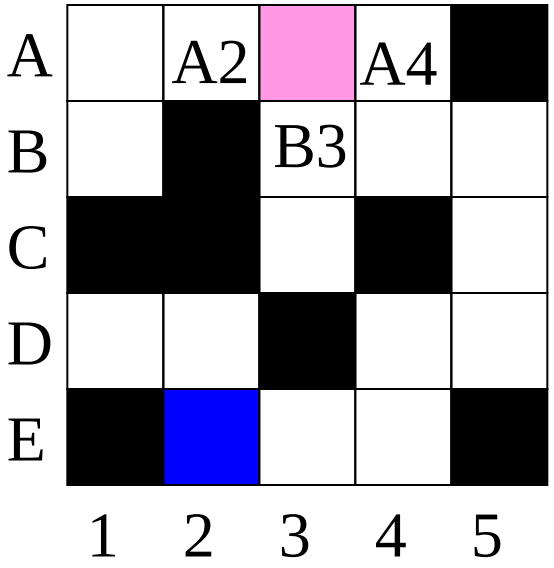
Operators: (in order)

- **`go_left`**(n)
- **`go_down`**(n)
- **`go_right`**(n)

each operator costs 1.

Heuristic: Manhattan distance

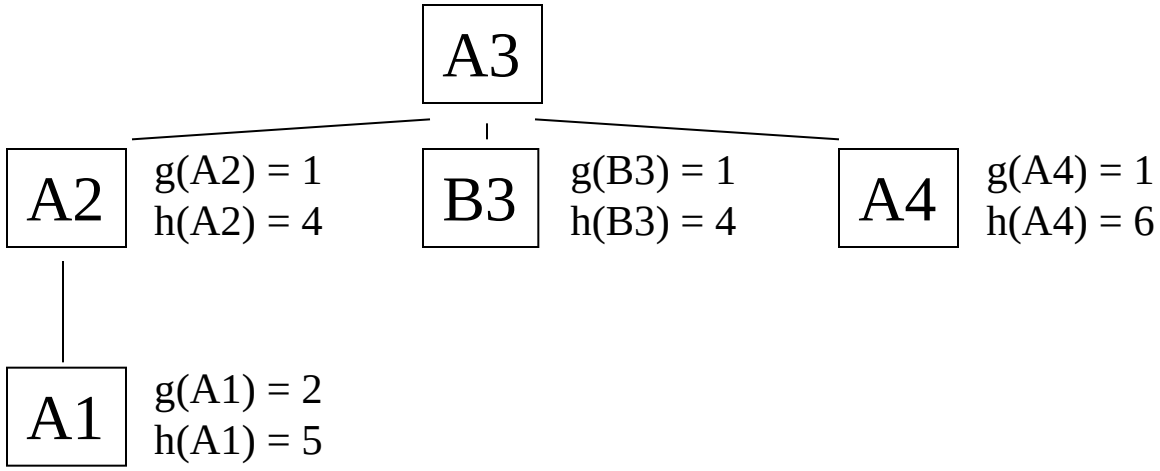


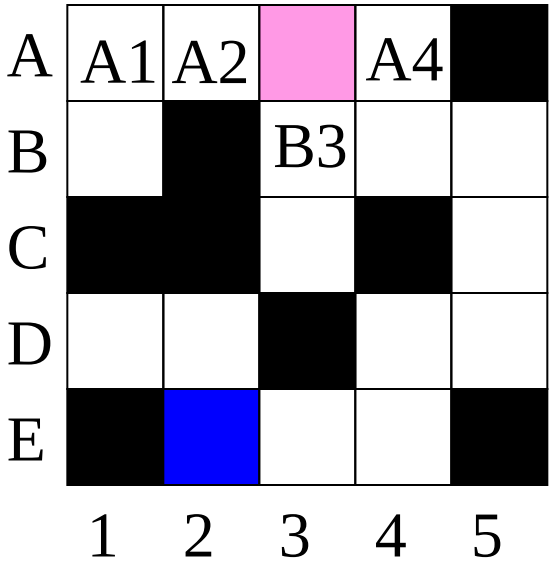


Operators: (in order)

- **`go_left`**(n)
- **`go_down`**(n)
- **`go_right`**(n)

each operator costs 1.

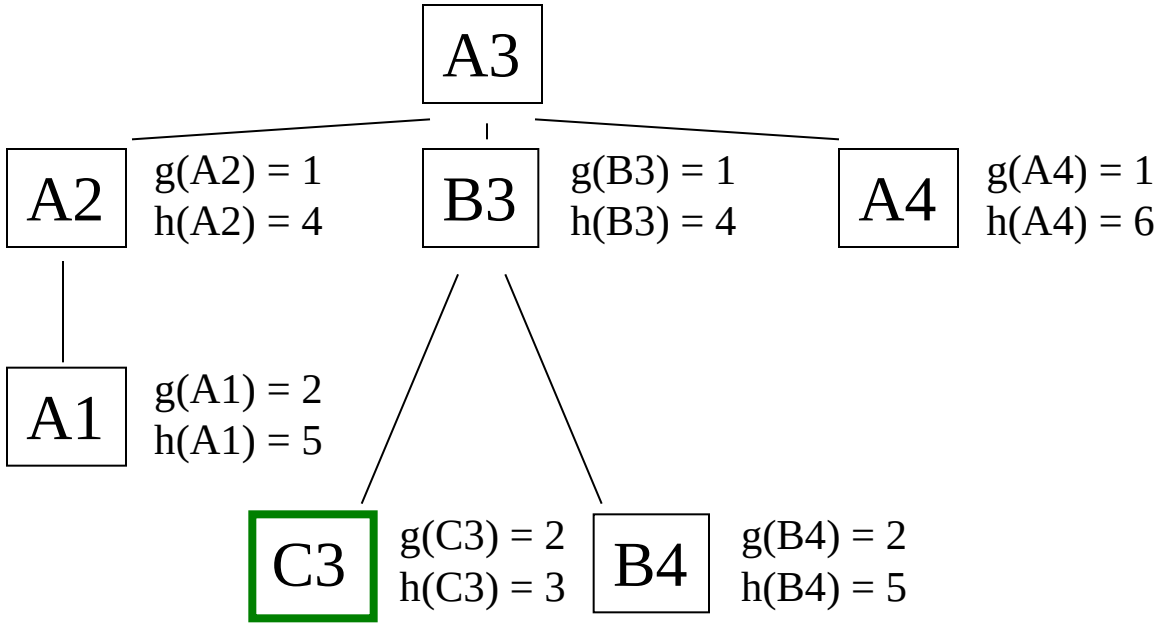


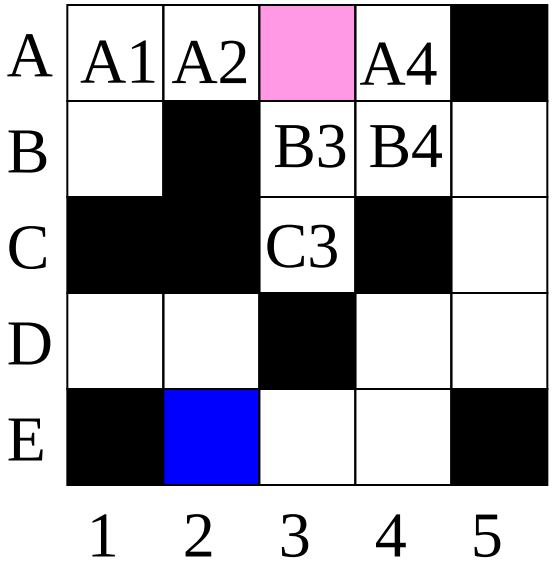


Operators: (in order)

- **`go_left`**(n)
- **`go_down`**(n)
- **`go_right`**(n)

each operator costs 1.

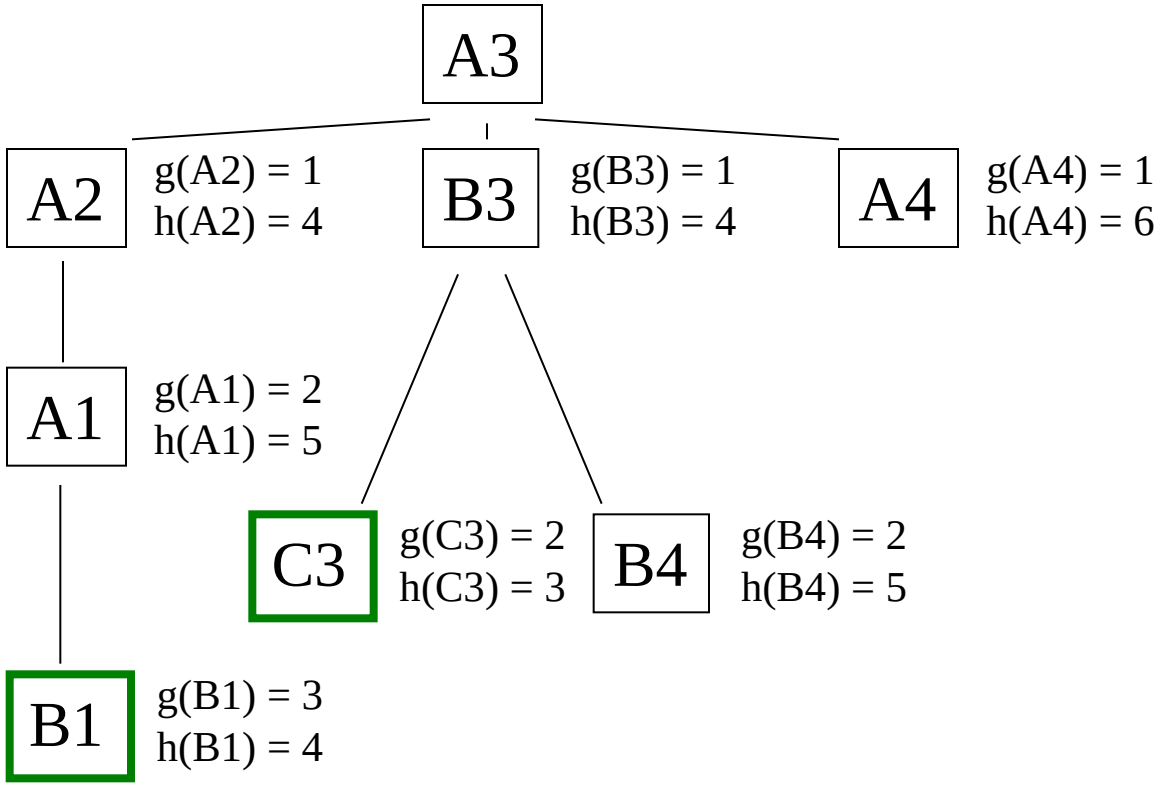


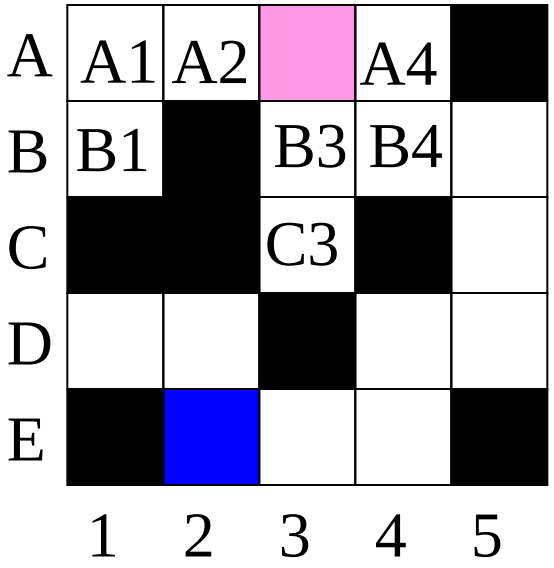


Operators: (in order)

- **`go_left`**(n)
- **`go_down`**(n)
- **`go_right`**(n)

each operator costs 1.

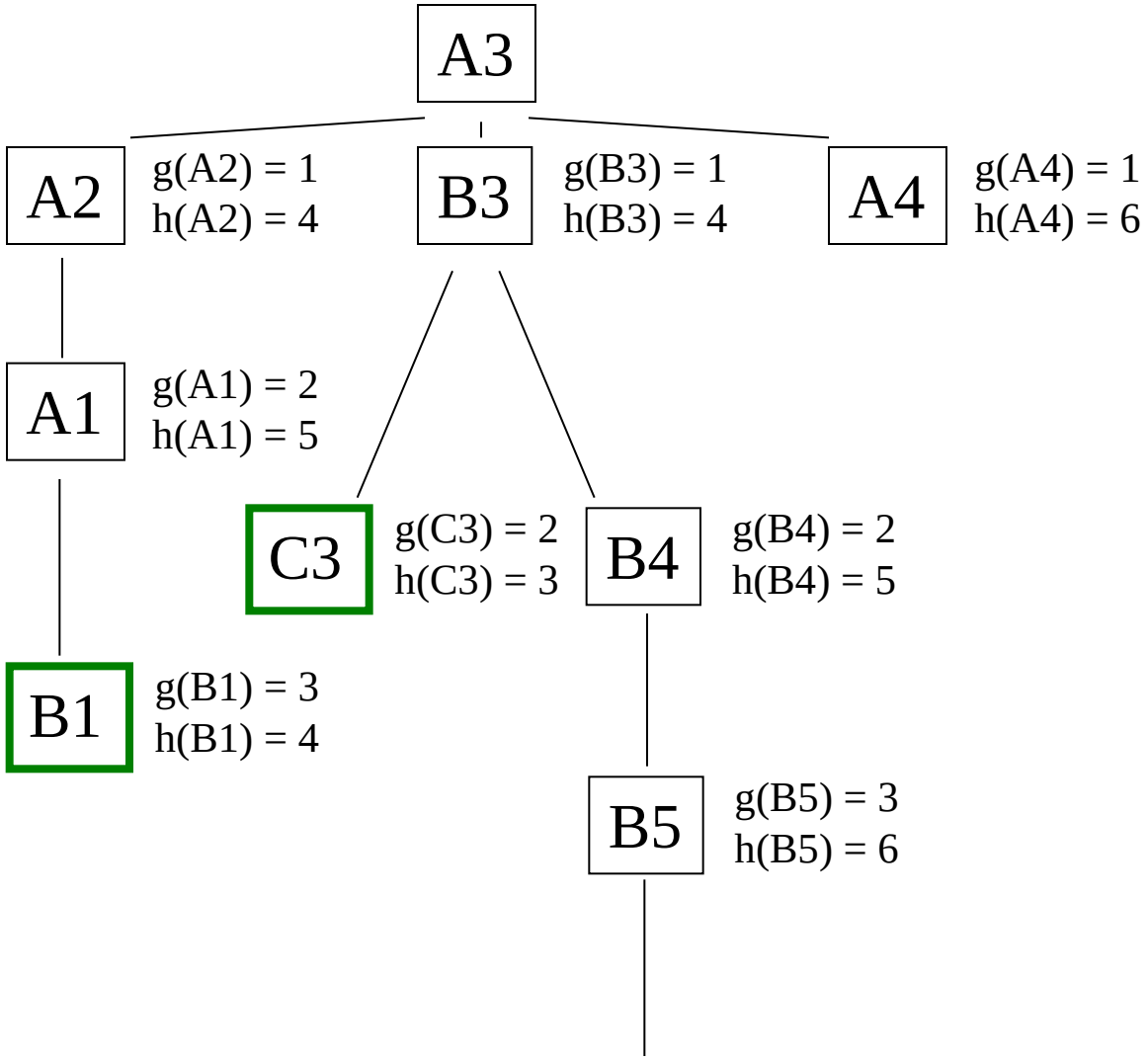


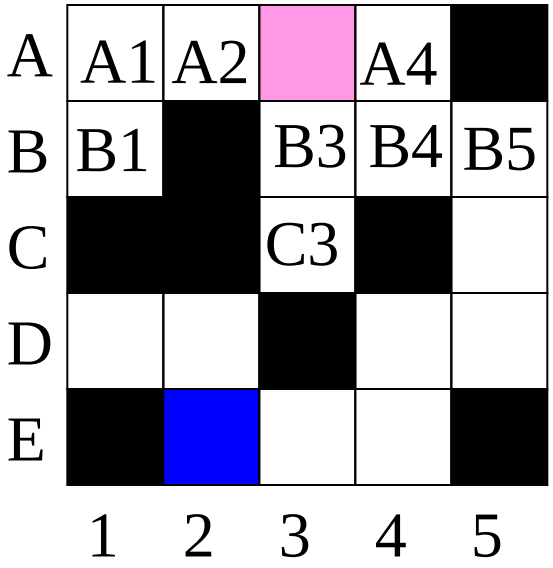


Operators: (in order)

- **`go_left`**(n)
- **`go_down`**(n)
- **`go_right`**(n)

each operator costs 1.





Operators: (in order)

- **`go_left`**(n)
- **`go_down`**(n)
- **`go_right`**(n)

each operator costs 1.