Review – State Space Search

- Strategy – Discover the "best" (shortest, cheapest, quickest, etc) path from the initial state to a goal state.
- State: A snapshot of the world.
- State space: The set of all possible states. (graph)

Review – State Space Search

- Node: A node is a node in the search tree.
- Search tree: Shows how the current search is proceeding.
- Frontier: Holds nodes. Think of the frontier as the set of all possible nodes the algorithm might want to examine next.
- Reached: → helps us keep track of the best cost so far for each state.

  Map : state → node

Review – Uniform Cost Search

- aka Dijkstra's algorithm
- Frontier = priority queue
  - Sorted by g(n): "Path-cost" = the total cost of all actions taken from initial state to n.
- Always expand lowest g(n) node on the frontier.
- Time/Space: → $O(b^d)$

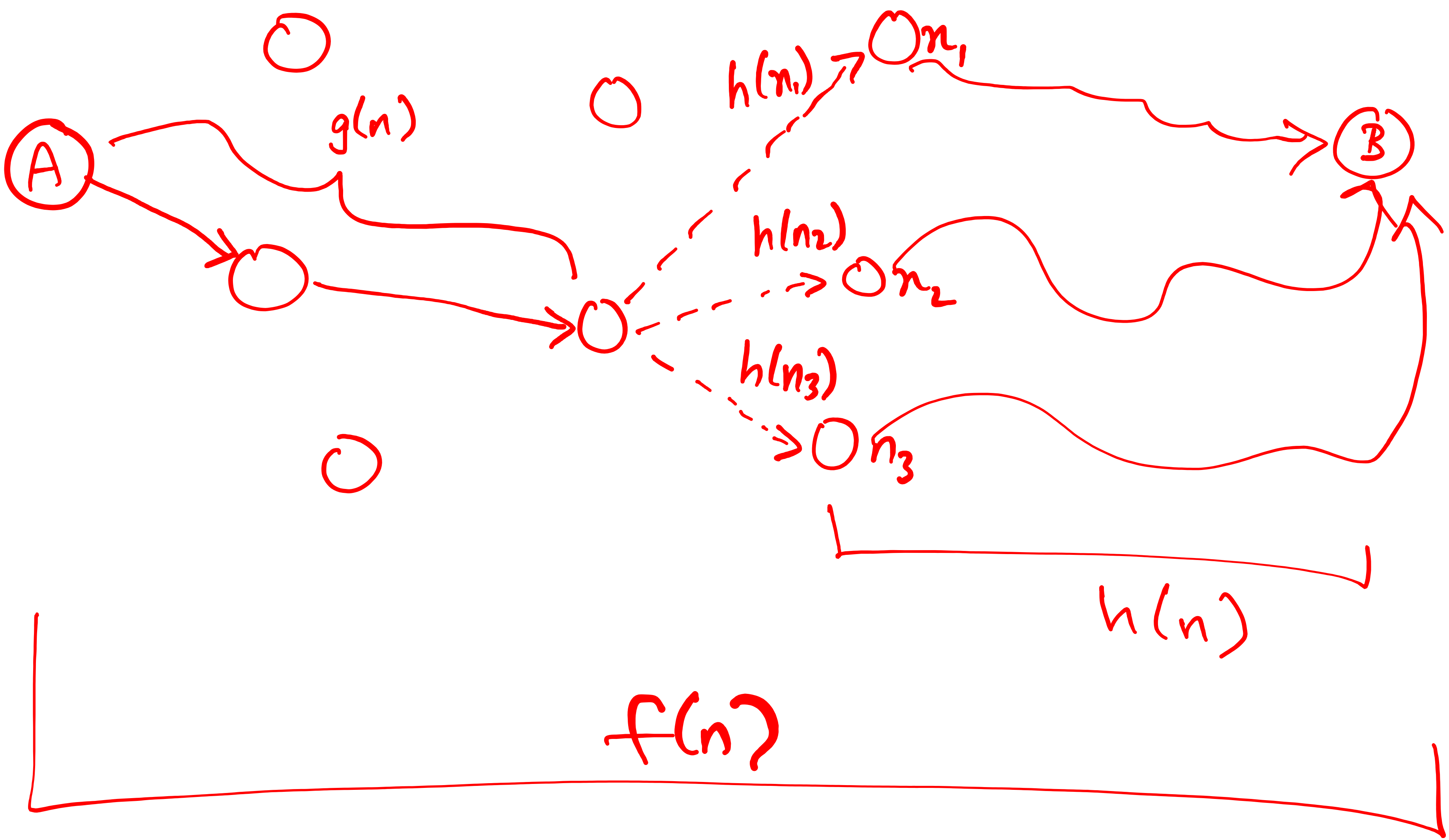
  b – branching factor
  d – depth of the solution in the search tree
- Complete? Yes Optimal? Yes

A* and variations

Algorithm A
A2
A3
A4
A*

- Same algorithm as uniform-cost search./Dijkstra
- Uses a different evaluation function to sort the priority queue.
- Need a heuristic function, h(n).
  - h(n) = **Estimate** of lowest-cost path from node n to a goal state.
  - In other words = an **estimate** of the distance remaining.

Visualizing a heuristic function

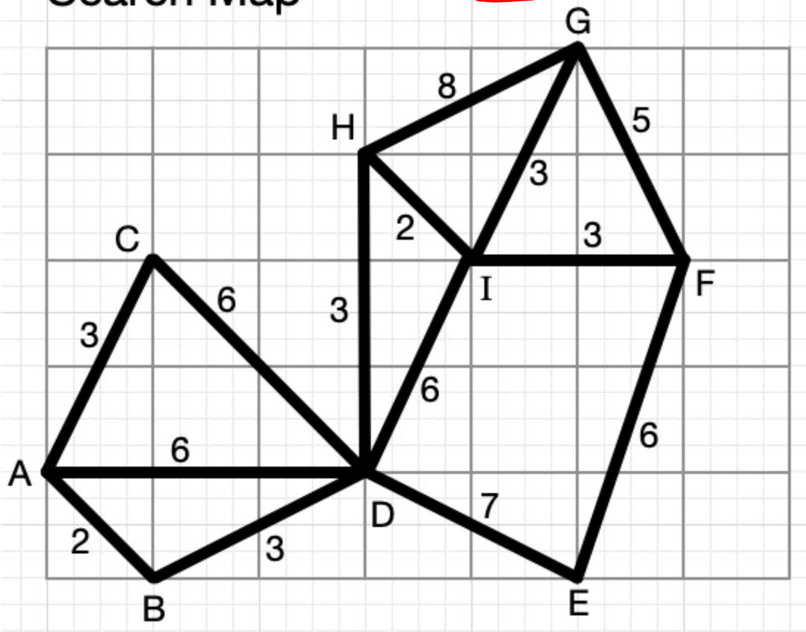


A* Algorithm

- Sort priority queue by a function f(n), which should be the *estimated* lowest-cost path through node n.
- How do we define f(n)?
  - Remember: g(n) = sum of costs from start state to node n.
  - h(n) = Estimate of lowest-cost path from node n to a goal state.
  - f(n) = g(n) + h(n)

Search Map

I → A



h(n) estimates for distance from n to A

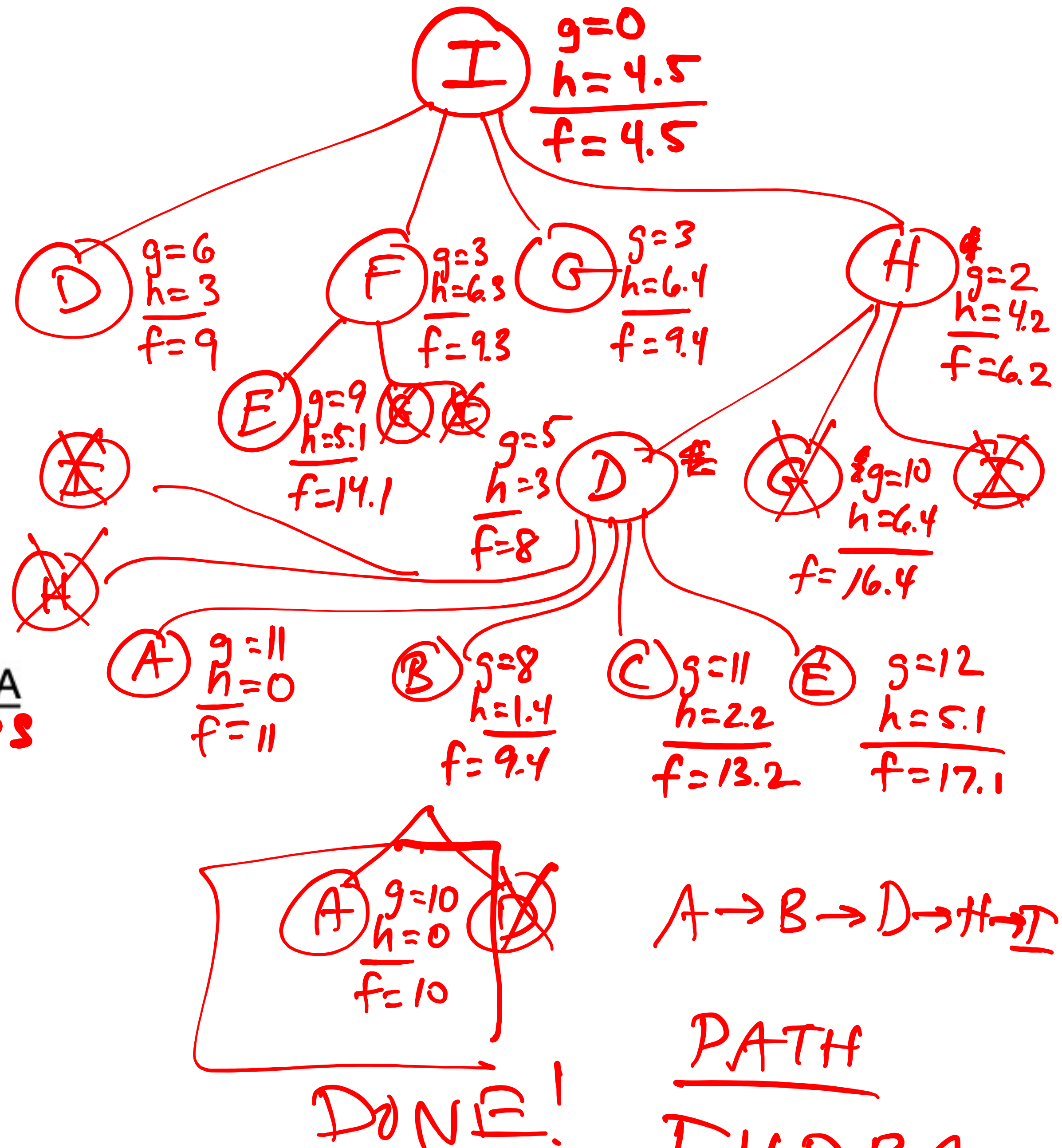
Straight-line distances

n	h(n)
A	0
B	~~1.4~~ 100
C	2.2
D	3
E	5.1
F	6.3
G	6.4
H	4.2
I	4.5

~~B (9.4)~~
C(13.2)
E(~~17.1~~) 14.1

Frontier

D ~~(9)~~ ~~(8)~~
~~F (9.3)~~
~~G (9.4)~~
~~H (6.2)~~
A ~~(11)~~(10)



A → B → D → H → I

PATH

IHDBA

Cost = 10

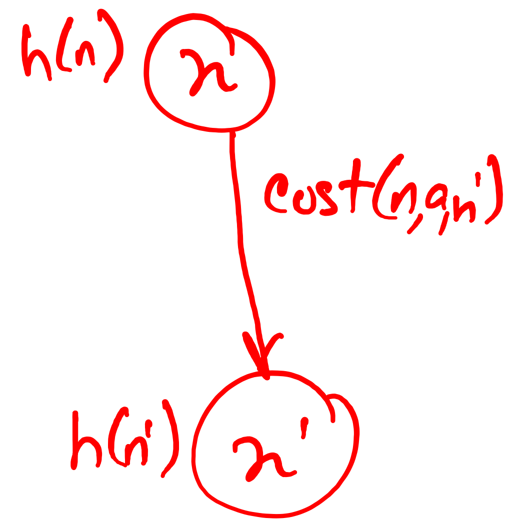
Properties of A*

Complete?

Yes

Optimal?

Yes*, only if your heuristic is AND
- admissible
- consistent



Admissibility: h(n) never over-estimates the true cost from n to the goal.

Consistent: h(n) must decrease along the path from the initial state to the goal state according to:

$$\forall n \quad h(n) \leq cost(n, a, n') + h(n')$$

$$h(n) - h(n') \leq cost(n, a, n')$$

Heuristics

- A heuristic function h(n) is ***admissible*** if it never over-estimates the true lowest cost to a goal state from node n.
- Equivalent: h(n) must always be less than or equal to the true cost from node n to a goal.
- What happens if we just set h(n) = 0 for all n?

↳ Dijkstra/UCS

Heuristics

- A heuristic function h(n) is ***consistent*** if values of h(n) along any path in the search tree are non-decreasing.
- Equivalent definition of consistency: given a node n, and an action which takes you from n to node n':

  h(n) <= cost(n, a, n') + h(n')

  h(n) – h(n') <= cost(n, a, n')
- Consistency implies admissibility (but not the other way around).
- Difficult to invent (natural) heuristics that are admissible but not consistent.

A* Algorithm

- A* is optimal if h(n) is consistent (and therefore admissible).
  - If your search space is a tree, A* only needs an admissible heuristic to be optimal, but this is uncommon.

Where do heuristics come from?

8-puzzle

h(

	1	3
8	2	4
7	5	6

) ⟶

1	2	3
4	5	6
7	8	

initial

h(n) = an estimate of n → goal

⟶ # of tiles out of place.

h(n) = an estimate of # of moves left.

⟶ ~~Man~~ Manhattan distance of ~~#~~ each tile to its correct location.

h(

1	2	3
4	5	6
7		8

)