Recap

- What things do we need to define in order to formulate a problem as a search problem?

Environment
static info that doesn't change

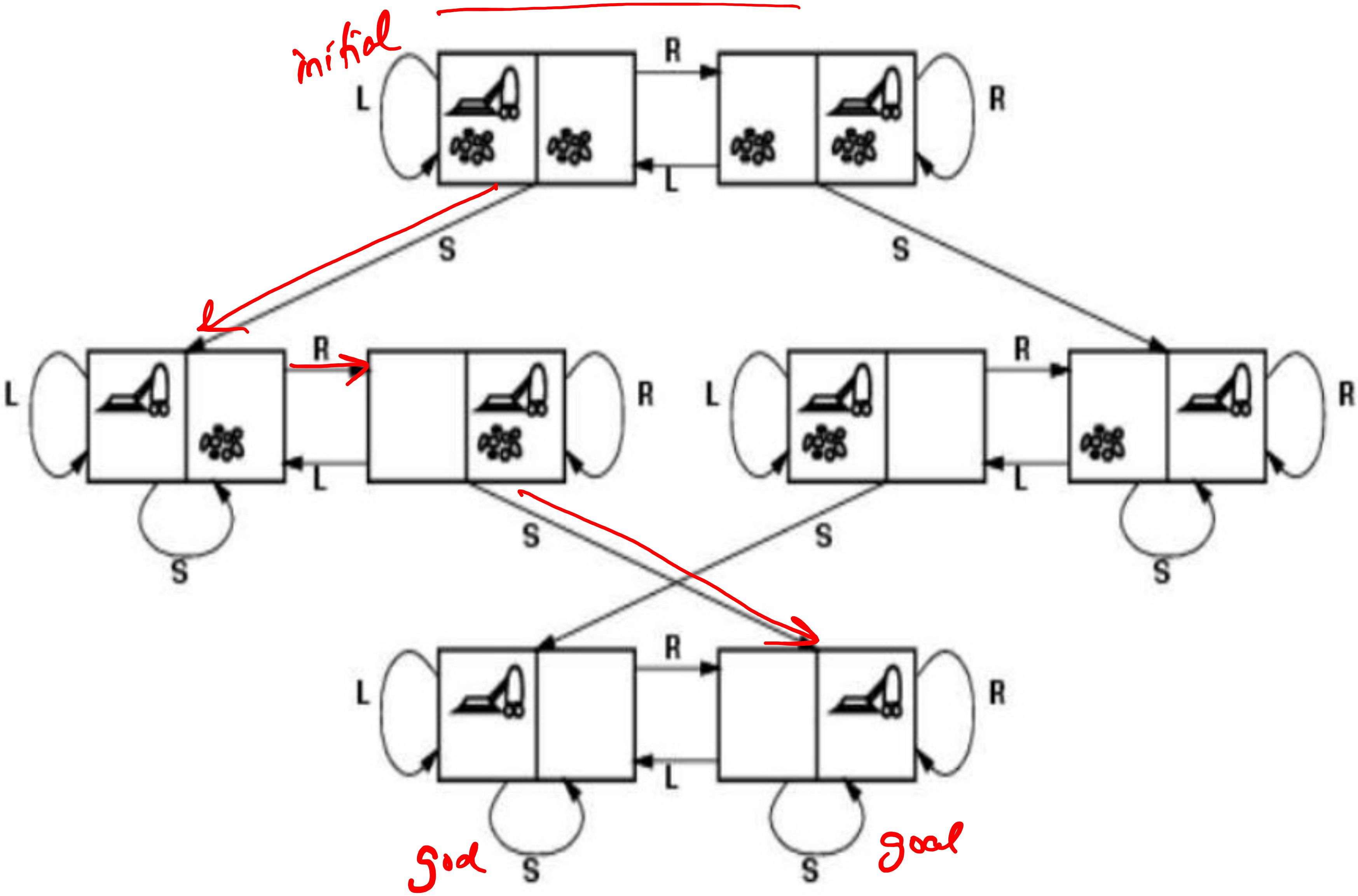
State
info that is changing

Initial / goal state

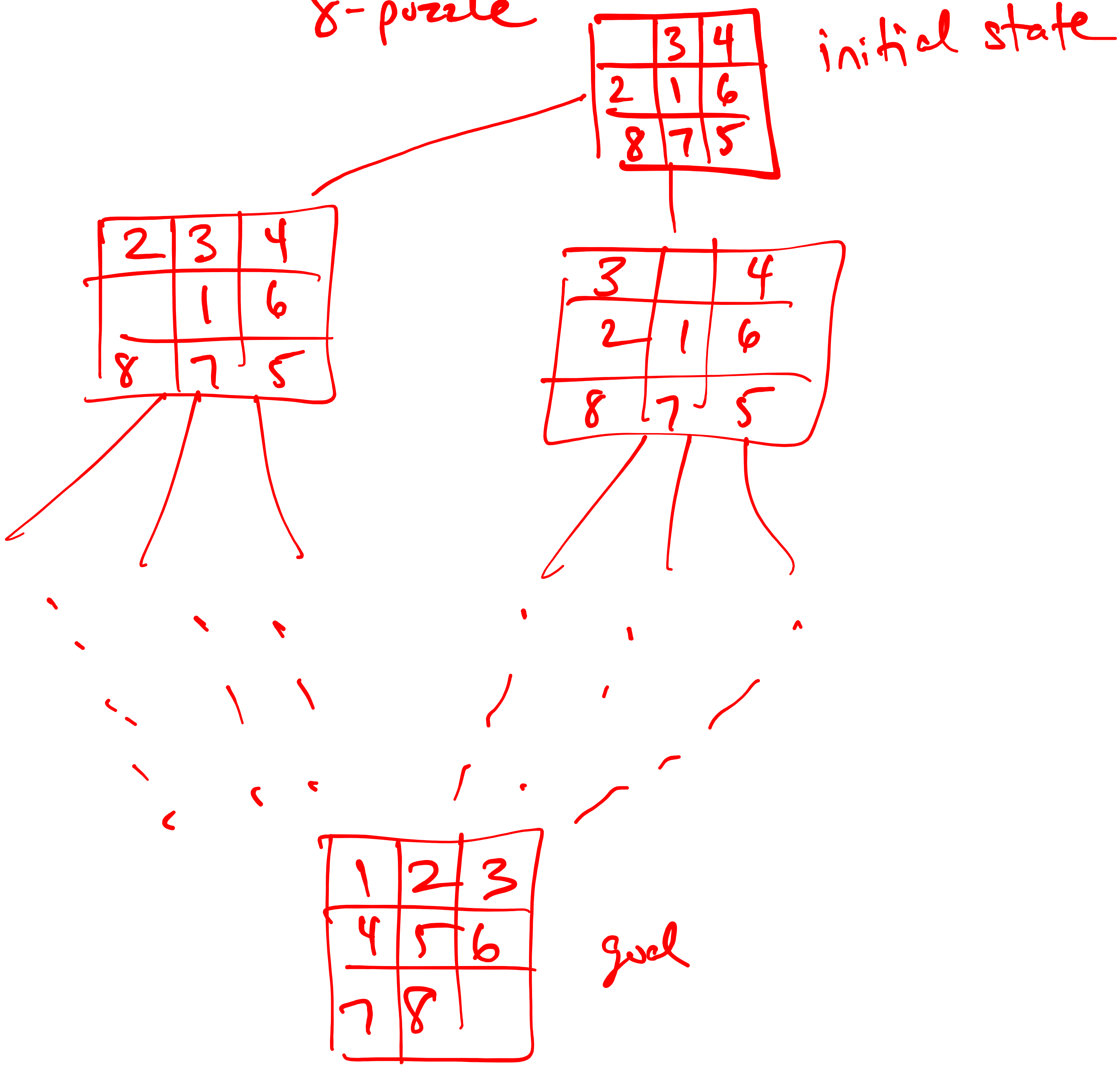
Actions

Costs

- Always a good idea to try to visualize the graph of the search space.

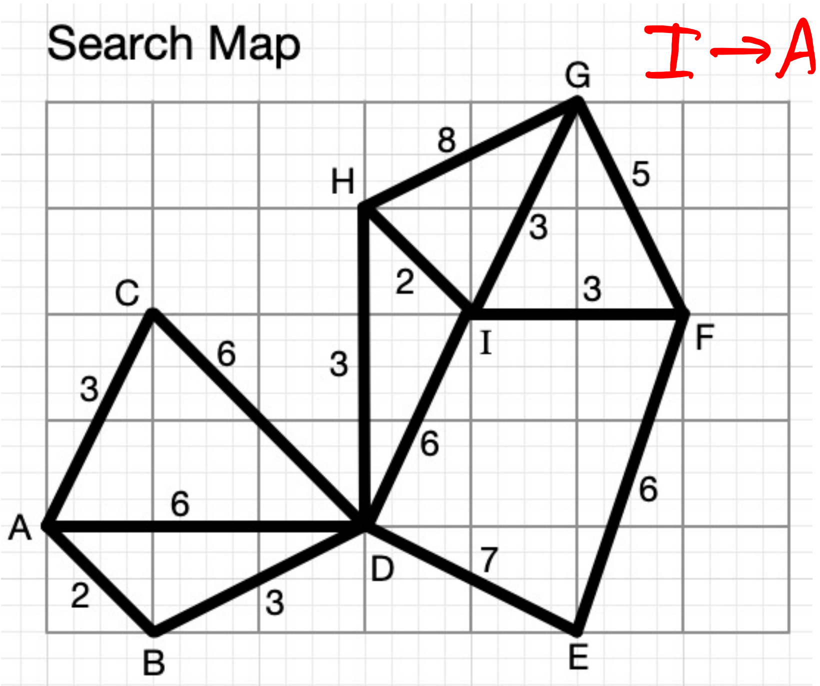


8-puzzle



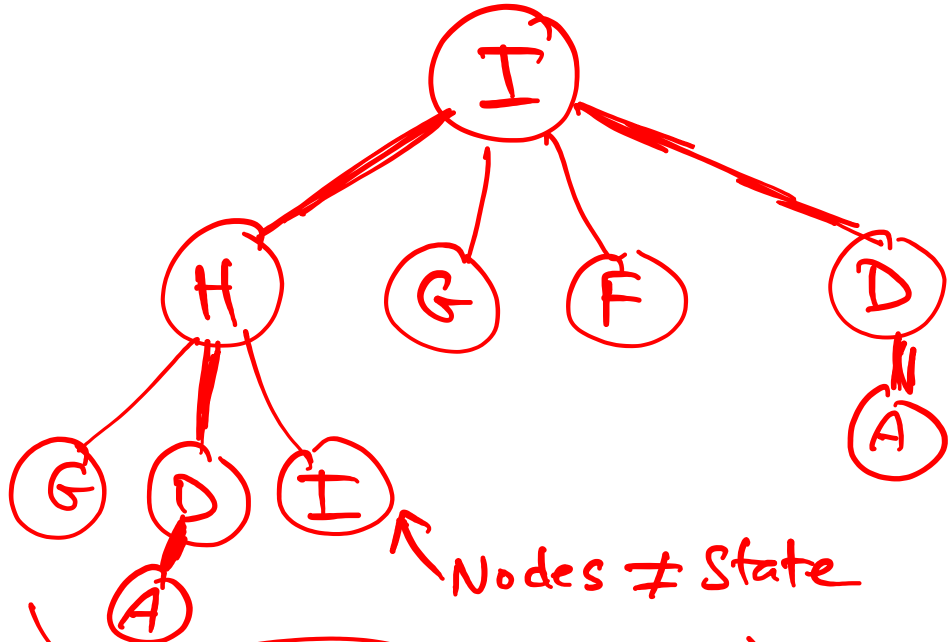
Generic search algorithms (3.3)

- All search algorithms work in essentially the same manner:
- Start with initial state.
- Generate all possible (next) successor states (*a.k.a.* "expanding a node."
- Pick a new node to expand. ← Step that is different for each algorithm.
- Continue until we find a goal state.

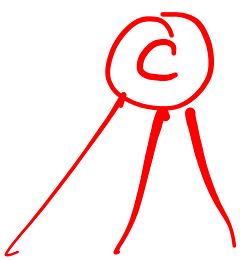


I → A

Start w/ initial state



Search tree



- There are two simultaneous graph-like structures used in search algorithms:
  - (1) Tree or graph of the underlying state space.
  - (2) Tree maintaining the record of the current search in progress (the ***search tree***).
- (1) does not depend on the current search being run.
- (1) is sometimes not even stored in memory (too big!)
- (2) always depends on the current search, and is always stored in memory. It is created on the fly during the running of the search algorithm.

Search tree

- A node ***n*** of the search tree stores:
  - a state (of the state space)
  - a pointer to the state's parent node (usually)
  - the action that got you from the parent to ***n*** (sometimes)
  - the path cost *g(**n**)*: cost of the path *so far* from the initial state to ***n***.

Generic search algorithms

(all based off of "best-first search")

- ***Frontier:*** a data structure storing the collection of nodes that are available to be examined next in the algorithm.
  - Often represented as a stack, queue, or priority queue.
- ***Reached:*** a map from nodes to states. Keeps track of which states have been examined already.

  explored set

  - Often stored using a data structure that enables quick look-up for membership tests.

    hash table

    Sorted tree

How do you evaluate a search algorithm?

- **Completeness** — Does the algorithm always find a solution if one exists?
- **Optimality** — Does the algorithm find the best solution?
- **Time complexity** – big-oh
- **Space complexity** – big-oh

Uninformed search methods

- These methods have no information about which nodes are on promising paths to a solution.
- Also called: *blind search*

Uninformed Search algorithms

- Breadth-first search ✓
- Uniform-cost search ✗
- Depth-first search ✓

```
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure
  node ← NODE(problem.INITIAL)
  if problem.IS-GOAL(node.STATE) then return node
  frontier ← a FIFO queue, with node as an element
  reached ← {problem.INITIAL}
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if problem.IS-GOAL(s) then return child
      if s is not in reached then
        add s to reached
        add child to frontier
  return failure
```

Breadth-first search

- Choose shallowest node for expansion. (in the search tree)
- Data structure for frontier?
  - Queue (regular)
- Complete? Optimal? Time? Space?

Complete? Yes

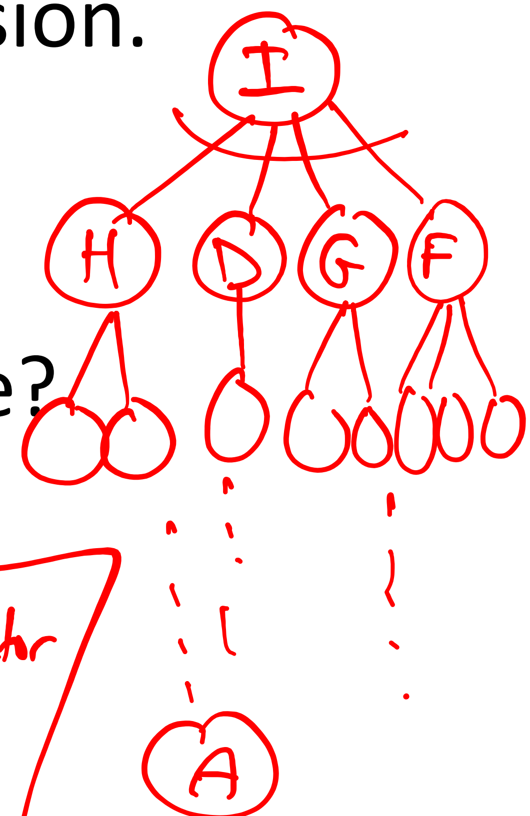
Optimal? No — Yes, only if all costs are equal

Time? $O(b^d)$

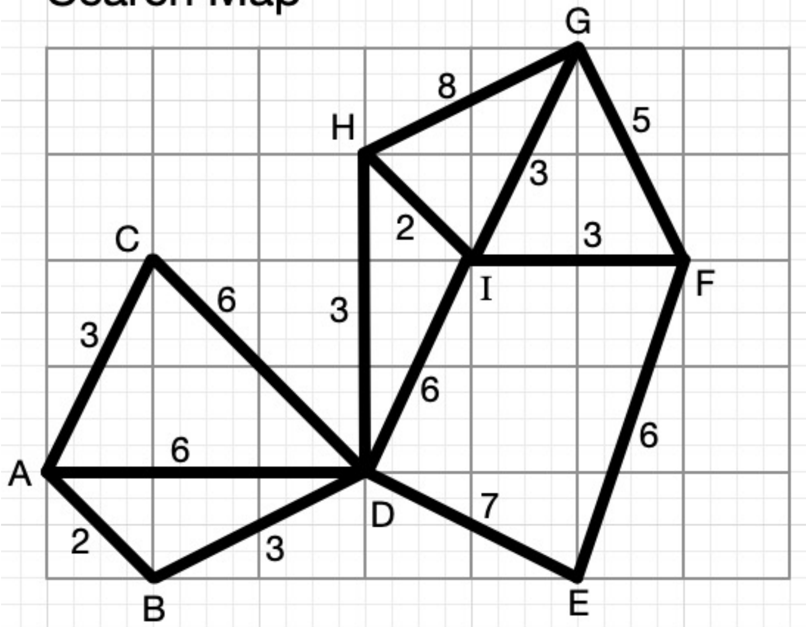
b: branching factor

d: depth of the search tree

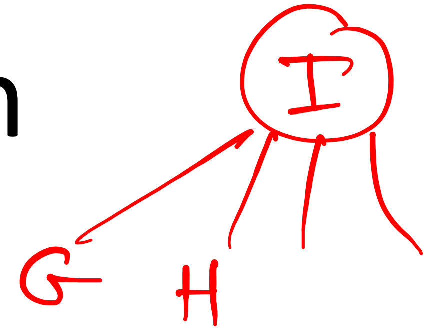
"n"



Search Map



Depth-first search

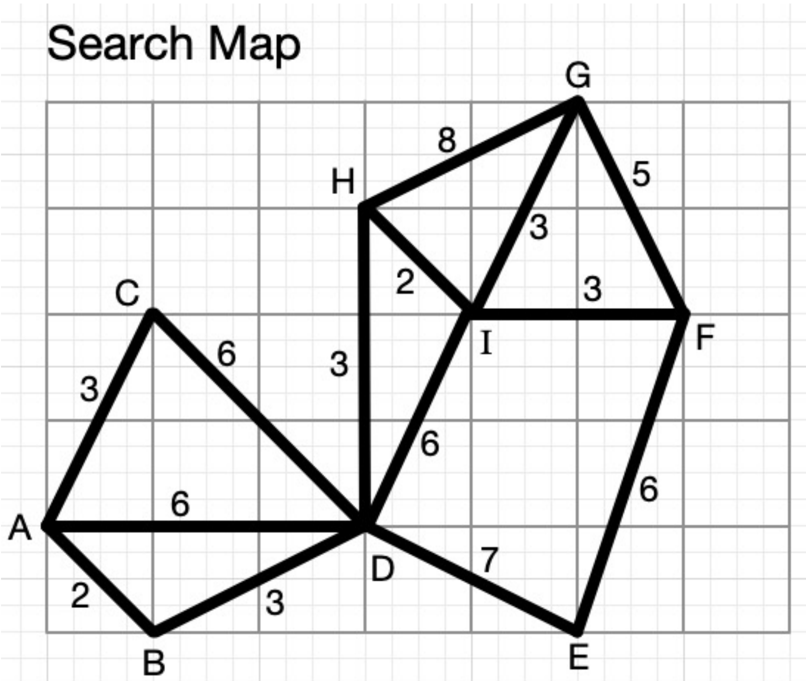


- Choose deepest node to expand.
- Data structure for frontier?
  - Stack (or just use recursion)
- Complete? Optimal? Time? Space?

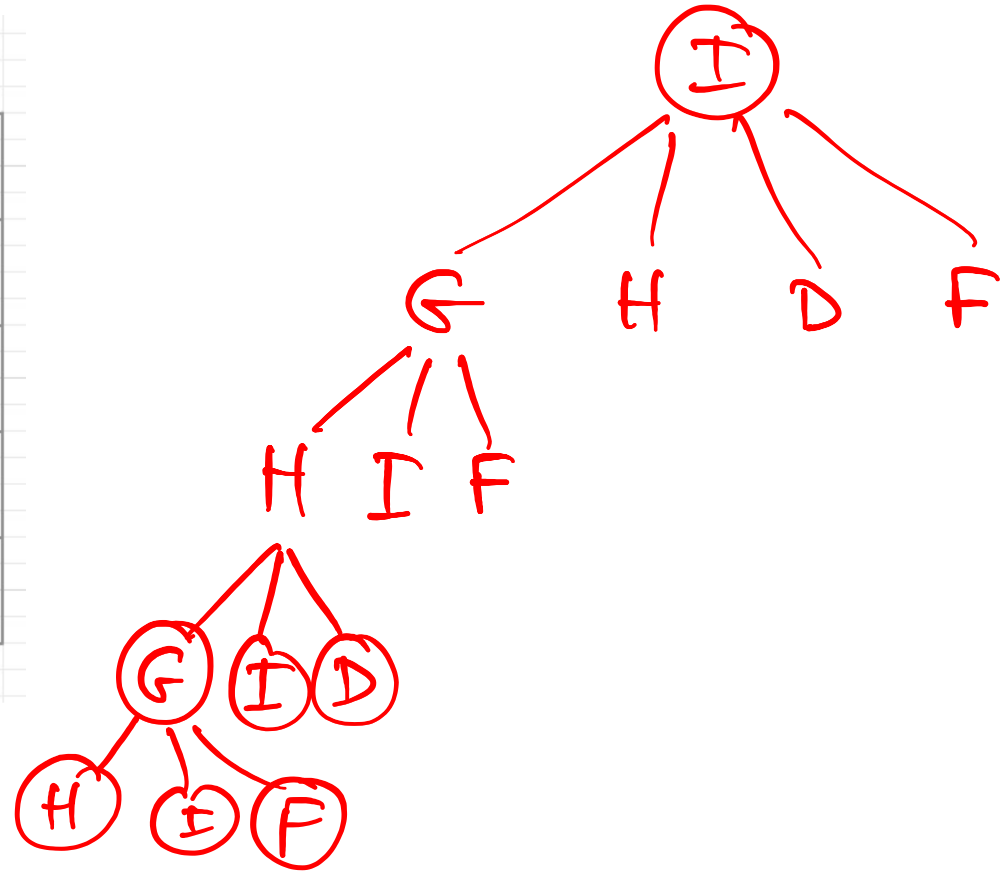
No, if we don't prevent loops

Yes, if we do prevent

No



Search Map
G
8
5
H
3
2
3
C
I
F
6
3
3
6
6
6
A
D
7
2
3
B
E



I
G
H
D
F
H
I
F
G
I
D
H
I
F

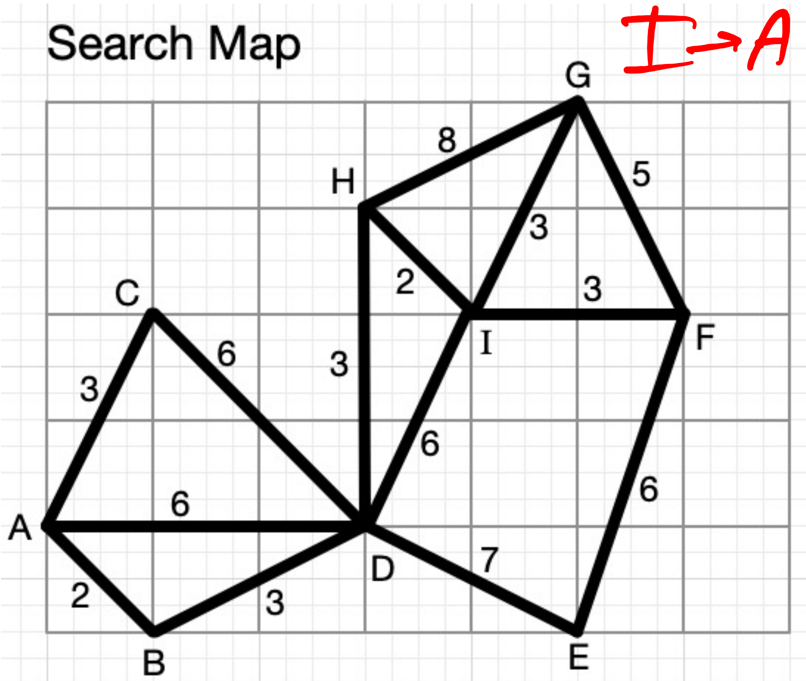
Dijkstra's alg =

Uniform-cost search

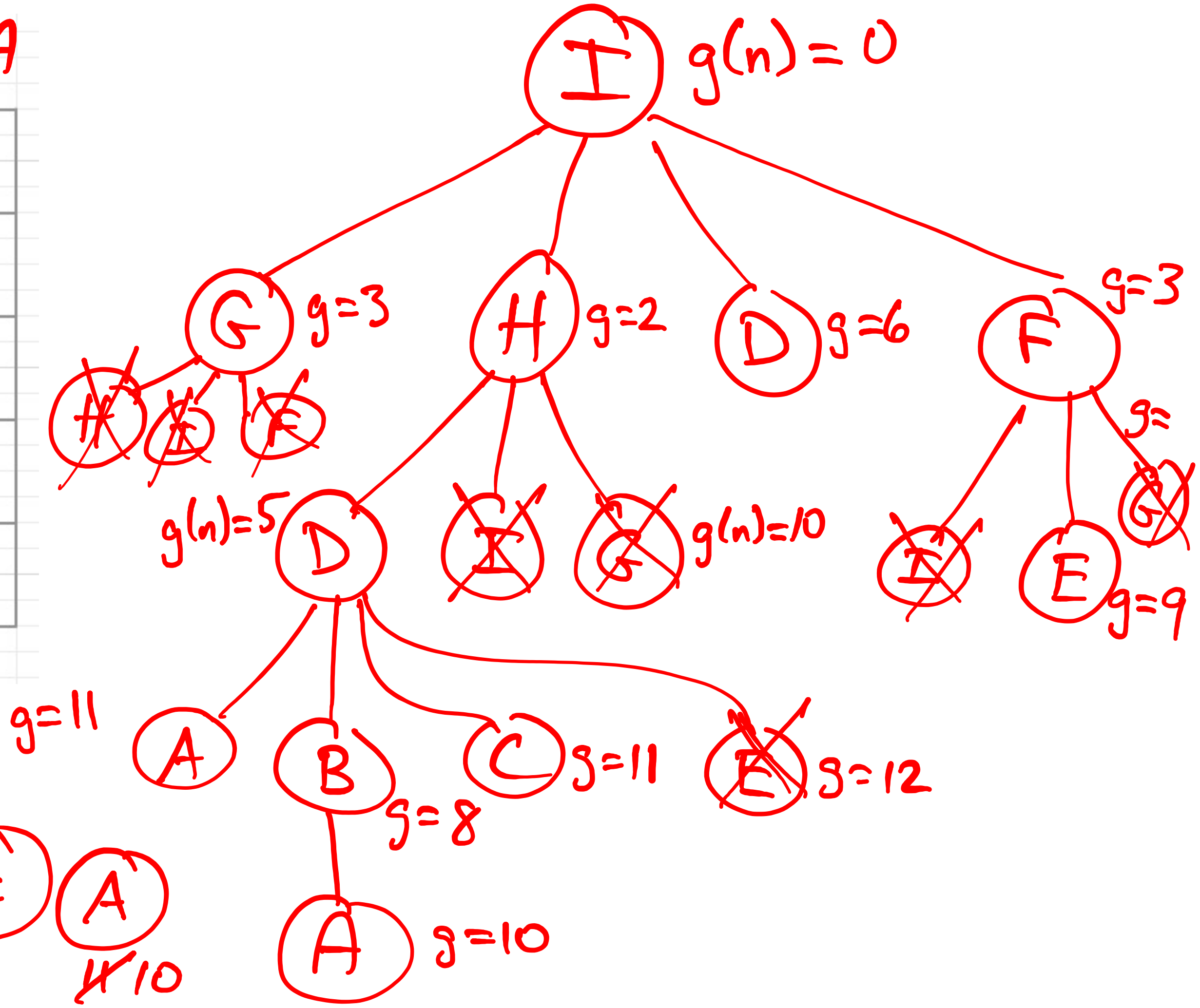
- Choose node with lowest path cost g(n) for expansion.

g(n): total path-cost from initial state to node n

- Data structure for frontier?
  - Priority queue
- Suppose we come upon the same state twice. Do we re-add to the frontier?
  - Yes, if lower path cost.



I→A



Frontier

~~G~~ 3, ~~H~~ 2, ~~D~~ ~~6~~ 5, ~~F~~ 3, E 9, A ~~11~~ 10

B 8, C 11

Reached

I 0, G 3, H 2, D ~~6~~ 5, F 3, E 9