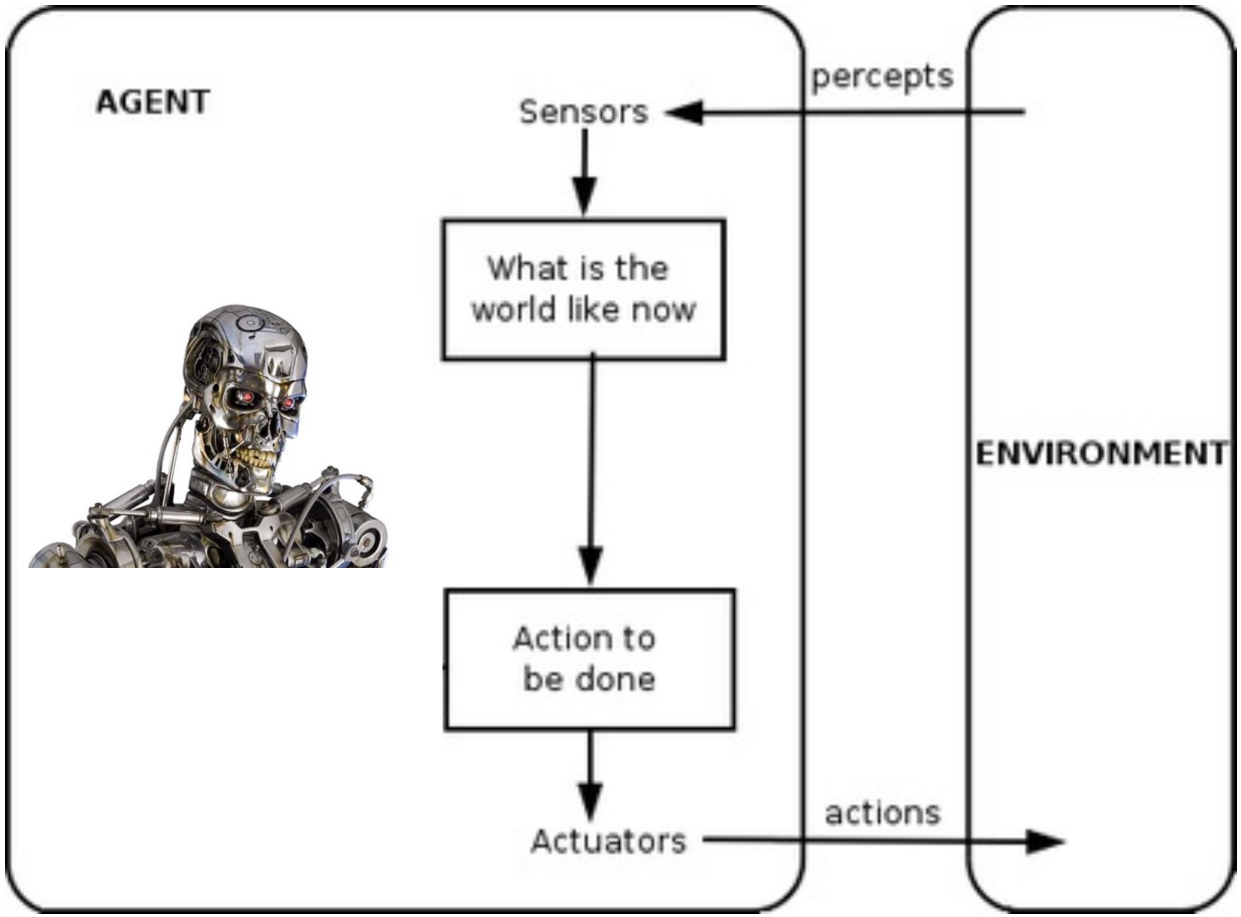
Agents



Agents interact with their environment through sensors and actuators.

Rational Agents

- Rational agent:
  - For every possible percept sequence, a rational agent should
  - select an action that is expected to maximize its performance measure,
  - given evidence provided by the percept sequence and whatever built-in knowledge the agent has.

Rational Agents

- Rational agent:
  - For every possible percept sequence, a rational agent should
  - select an action that is **expected to maximize its performance measure**,
  - given evidence provided by the percept sequence and whatever built-in knowledge the agent has.

Rational Agents

- Rational agent:
  - For every possible percept sequence, a rational agent should
  - select an action that is expected to maximize its performance measure,
  - given **evidence provided by the percept sequence and whatever built-in knowledge the agent has**.

Rational Agents

- Rational agent:
  - **For every possible percept sequence**, a rational agent should
  - select an action that is expected to maximize its performance measure,
  - given evidence provided by the percept sequence and whatever built-in knowledge the agent has.

Environments

- Fully-observable vs partially-observable
- Single agent vs multiple agents
- Deterministic vs non-deterministic
- Episodic vs sequential
- Static or dynamic
- Discrete or continuous

State Space Search

Environments

- Fully-observable vs partially-observable
- Single agent vs multiple agents
- Deterministic vs stochastic
- Episodic vs sequential
- Static or dynamic
- Discrete or continuous

Overview

- Problem-solving as search
- How to formulate an AI problem as search.
- Uninformed search methods

What is search? (3.1)

What is search?

Environmental factors needed

- **Static** — The world does not change on its own, and our actions don't change it.
- **Discrete** — A finite number of individual states exist rather than a continuous space of options.
- **Observable** — States can be determined by observations.
- **Deterministic** — Action have certain outcomes.

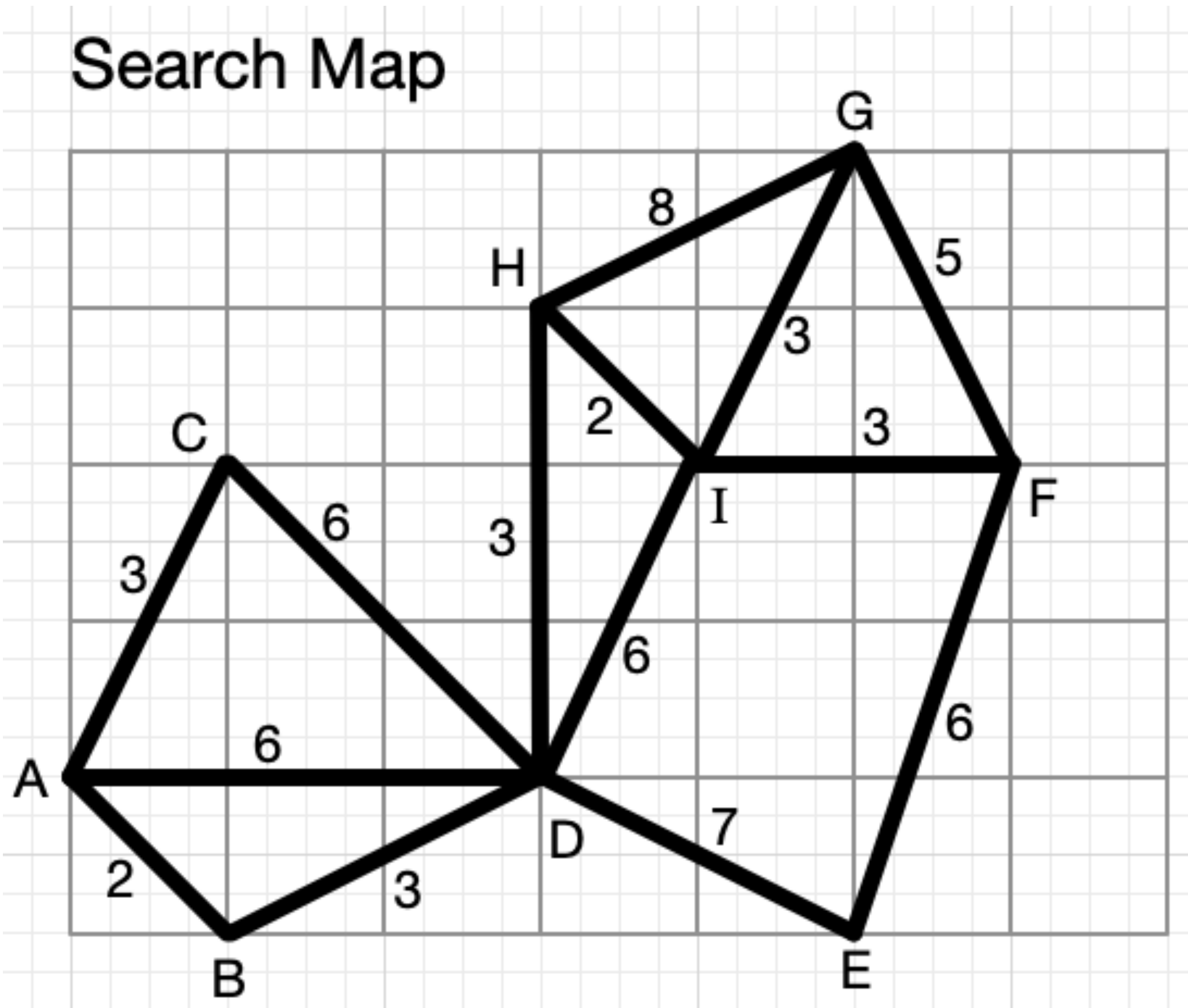
- The **environment** is all the information about the world that remains constant while we are solving the problem.
- A **state** is the set of properties that define the current conditions of the world our agent is in.
  - Think of this as a *snapshot* of the world at a given point in time.
  - The entire set of possible states is called the **state space**.
- The **initial state** is the state the agent begins in.
- A **goal state** is a state where the agent may end the search.
- Agents move from state to state by taking **actions.** Moving from state to state has an associated **cost.**

- How does an agent know what actions are possible in a state?
  - Imagine a function **ACTIONS(*s*)** that returns the set of actions possible in a state ***s***.
- How does an agent know what state they go to when they take an action?
  - Imagine a function **RESULT(s, a)** that returns the new state **s'** that you end up in when taking action **a** from state **s**.
- How does an agent know when they have reached a goal state?
  - Imagine a function **IS-GOAL(s)** that returns true/false.
- How does an agent know the cost of moving from one state to another?
  - Imagine a function **ACTION-COST(s, a, s')** which returns the cost of taking action **a** in state **s** and moving to state **s'**.

Formulating problems as search (3.2)

- *Canonical problem*: route-finding
  - Route-finding with traveling salesperson problem.
- Sliding block puzzle (almost any kind of game or puzzle can be formulated this way).
- Roomba problem.

Formulate navigation problem



Formulate navigation problem

Formulate 8-puzzle problem

Formulate Roomba problem

Formulate Roomba problem

Formulating problems as search

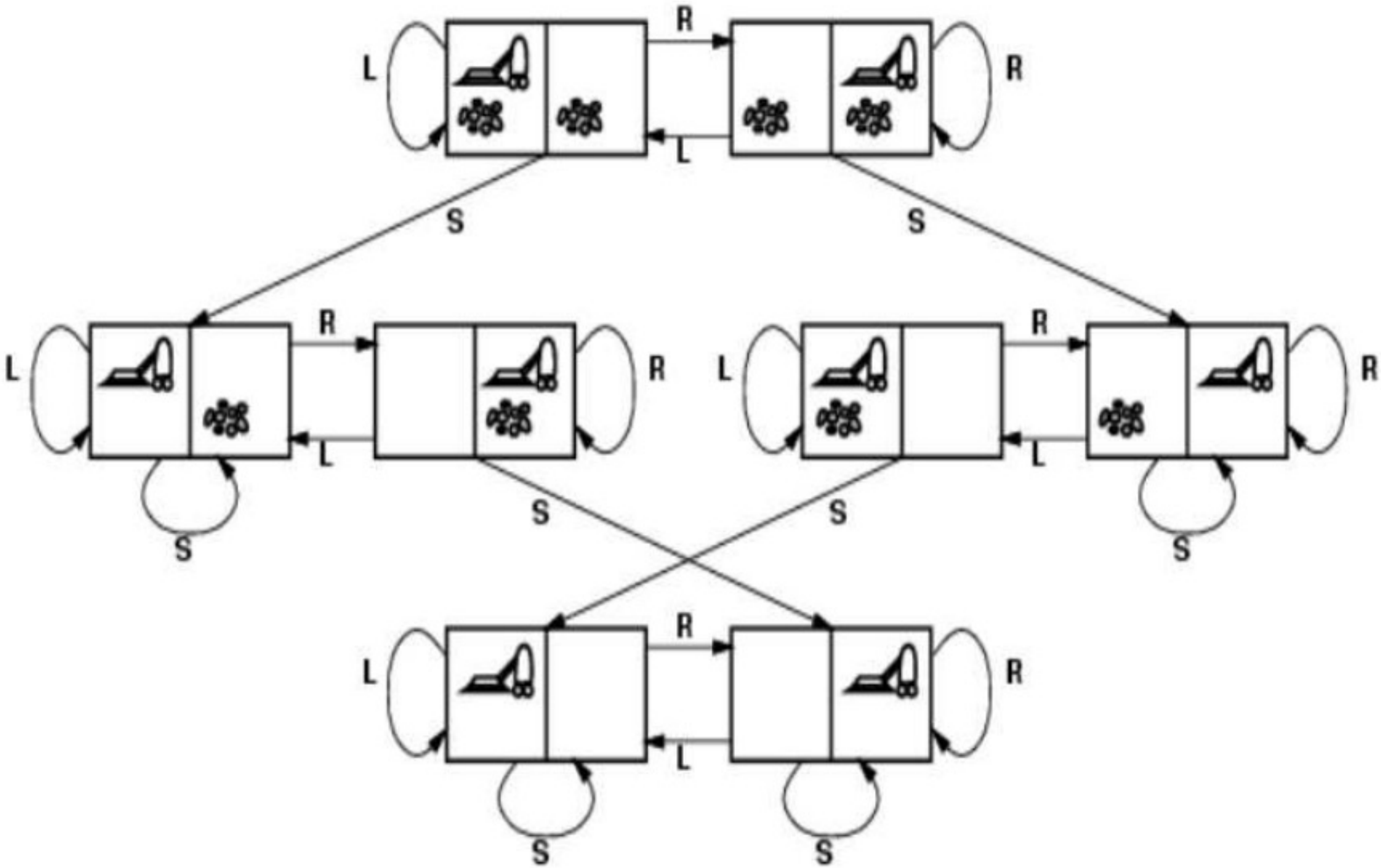
- A solution to a search problem is a ***path*** between the initial state and a goal state.
- The ***quality*** of a solution is measured by path cost, which is the sum of all the individual costs along the way.
- ***Optimal solutions*** have the lowest cost of any possible path.

- Side note:
- Consider whether the search space forms a tree or a graph.
  - Often there are faster versions of these algorithms for searching trees.

Recap

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

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



Generic search algorithms (3.3)

- All search algorithms work in essentially the same manner:
- Start with initial state.
- Generate all possible successor states (*a.k.a.* "expanding a node."
- Pick a new node to expand.
- Continue until we find a goal state.

- There are two simultaneous graph-like structures used in search algorithms:
  - (1) Graph (or tree) of 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' data structures

- ***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 **states** to **nodes**.
  - Used to quickly access the priorities of states stored in the frontier to see if the algorithm has found a better priority.

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**
- **Space complexity**

Uninformed search methods

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

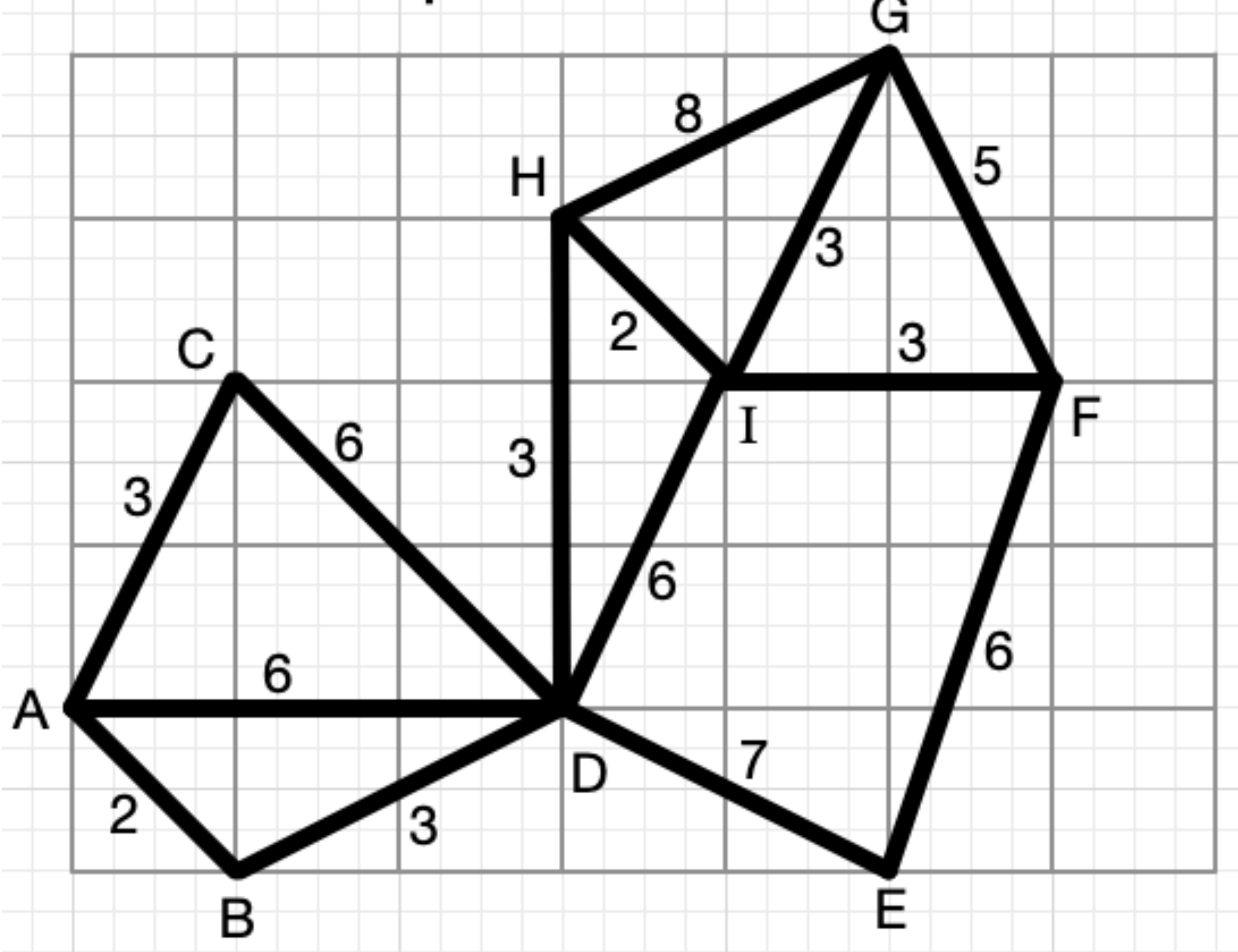
Uninformed search algorithms

- Breadth-first search
  - Variant — Uniform-cost search
- Depth-first search

Breadth-first search

- Choose shallowest node for expansion.
- Data structure for frontier?
  - Queue (regular, FIFO)
- Complete? Optimal? Time? Space?

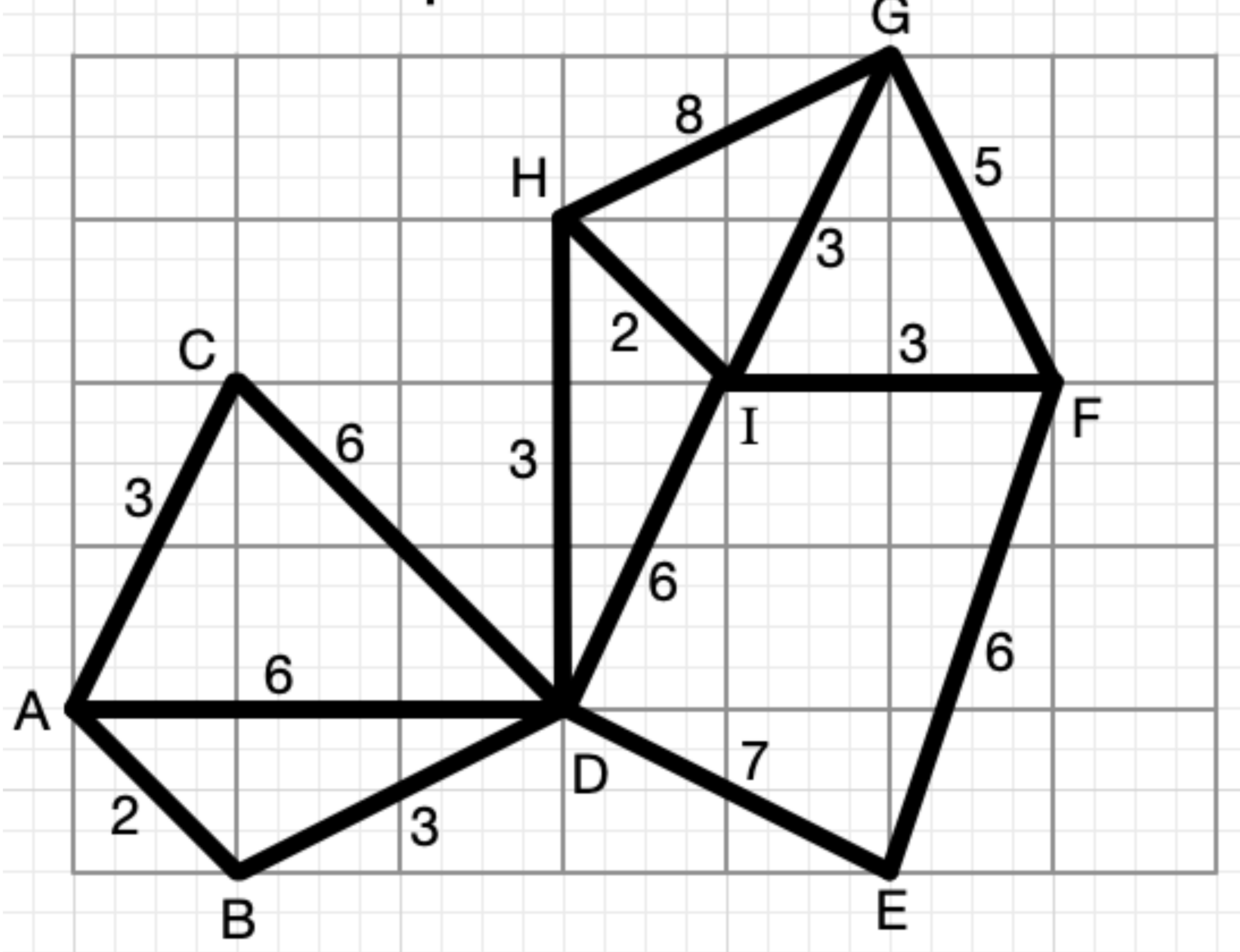
Search Map



Depth-first search

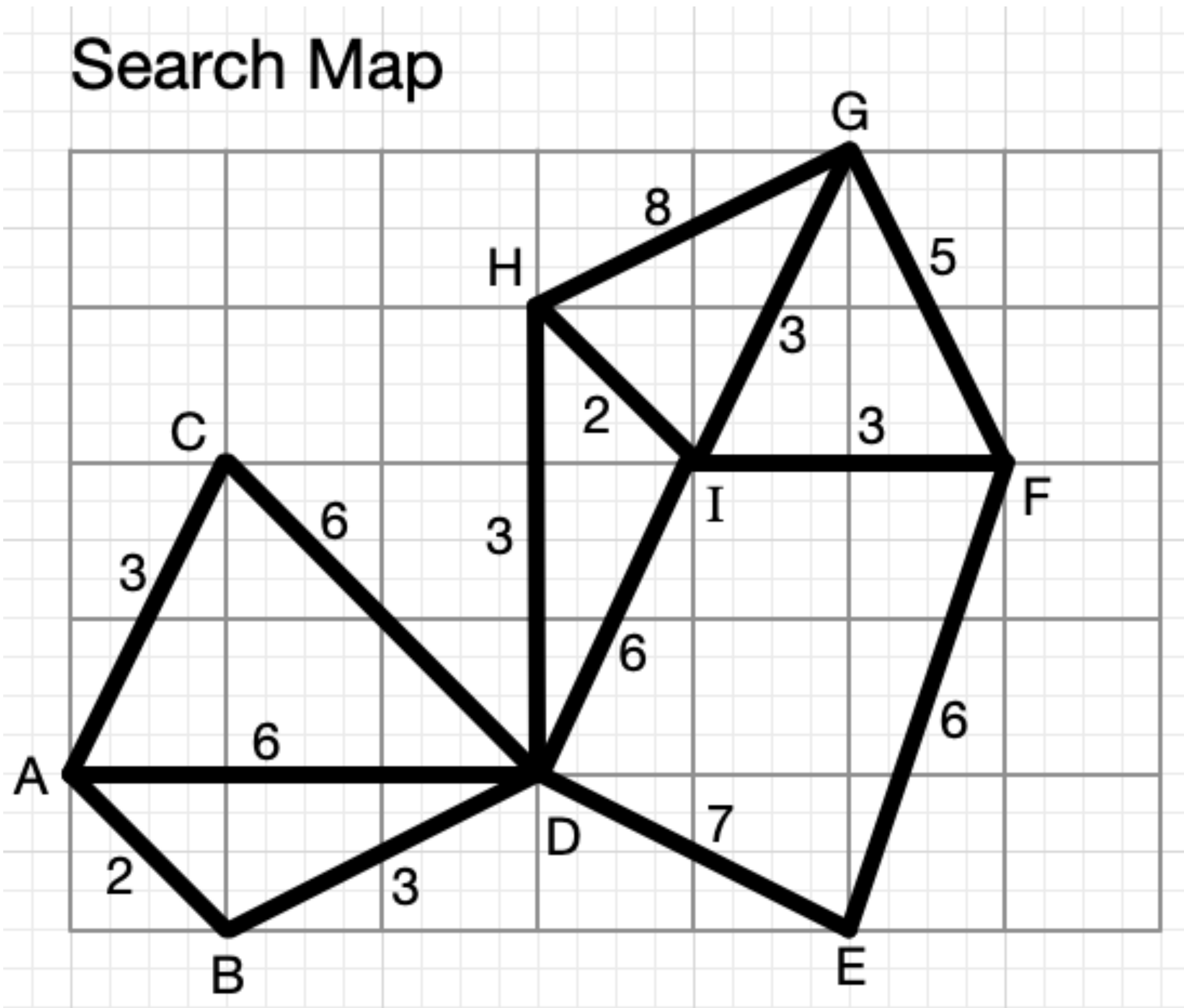
- Choose deepest node to expand.
- Data structure for frontier?
  - Stack (or just use recursion)
- Complete? Optimal? Time? Space?

Search Map



Uniform-cost search

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


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

Uniform-cost search

- Choose node with lowest path cost g(n) for expansion.
- 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.
- Complete? Optimal? Time? Space?

Review – State Space Search

- Strategy – Discover the best (shortest, cheapest, quickest, etc) path from the initial state to a goal state.
- State:
- State space:

Review – State Space Search

- Node:
- Search tree:
- Frontier:
- Reached:

Review – Uniform Cost Search

- aka Dijkstra's algorithm
- Frontier = priority queue
  - Sorted by g(n):

- Always expand lowest g(n) node on the frontier.
- Time/Space:
- Complete? Optimal?

A* and variations

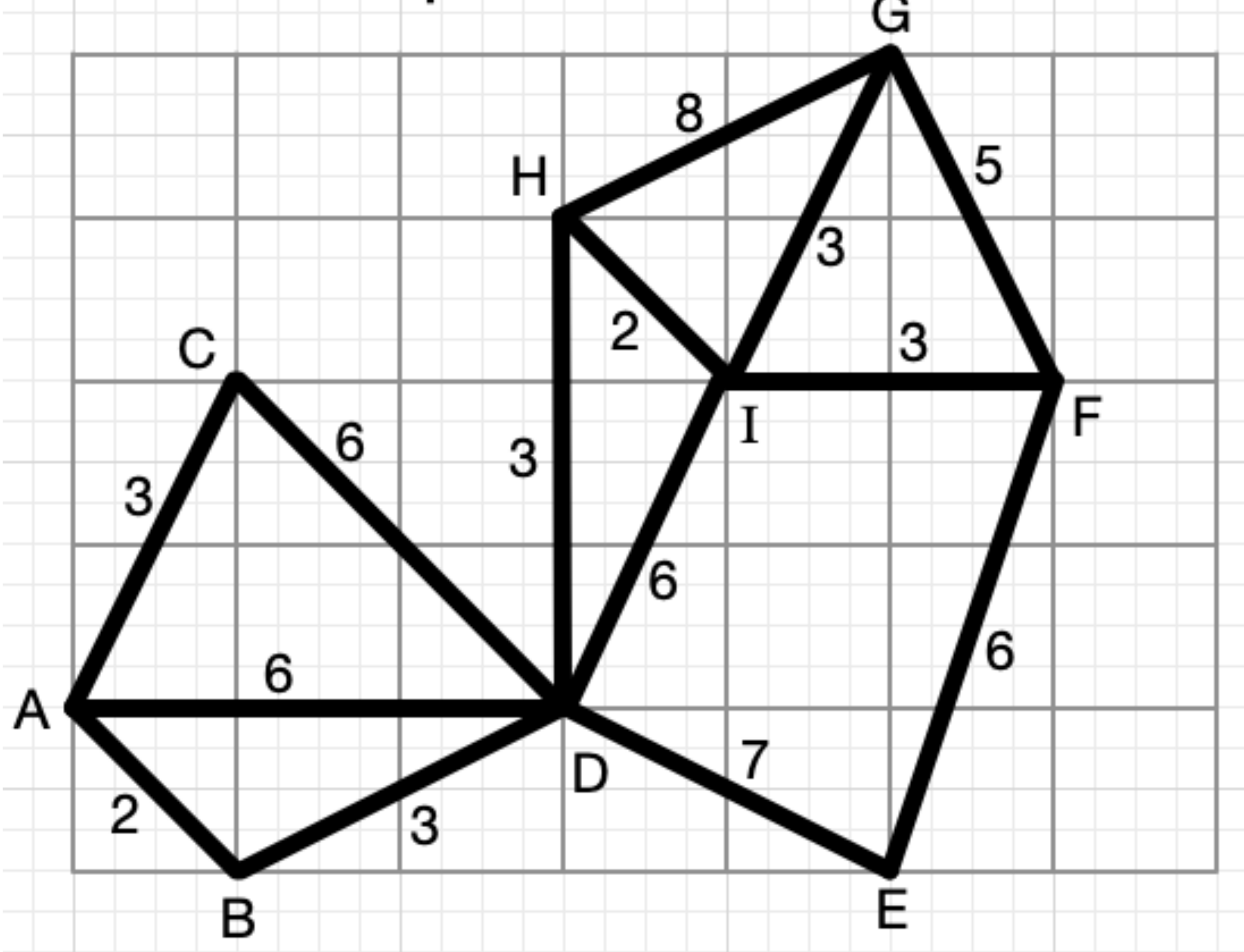
- Same algorithm as uniform-cost search.
- Uses a different evaluation function to sort the priority queue.
- Need a heuristic function, h(n).
  - h(n) = **Estimate** of lowest-cost path from node n to a goal state.
  - In other words = an **estimate** of the distance remaining.

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



h(n) estimates

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

Properties of A*

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?

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?

Where do heuristics come from?

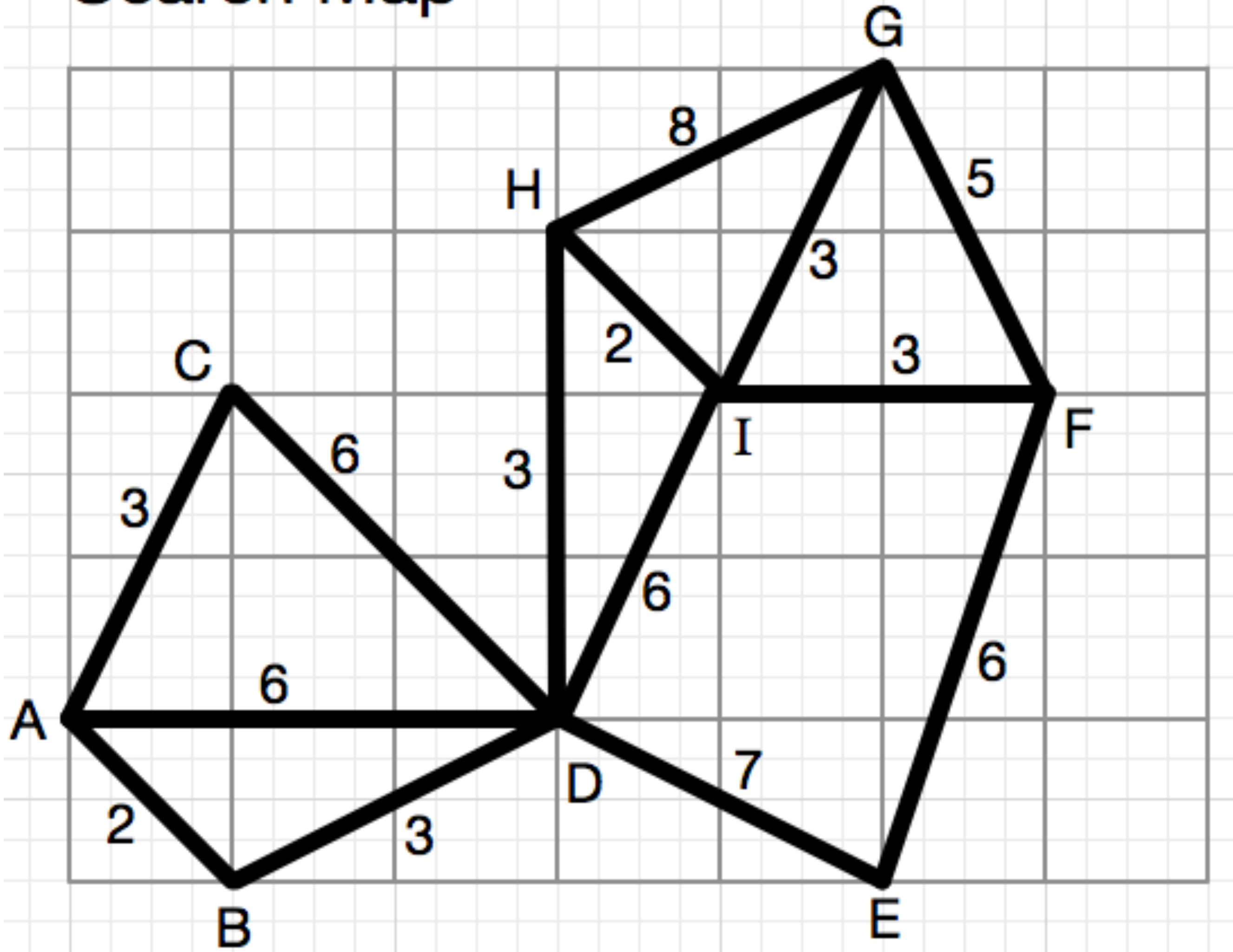
Greedy best-first search

- Use just h(n) to sort priority queue.
- Optimal?
- Complete?

Summary

- Uniform cost search (Dijkstra) [sort by g(n)]
  - Complete and optimal.
- A* [sort by f(n) = g(n) + h(n)]
  - Complete and optimal, assuming an admissible and consistent heuristic.
- Greedy best first search [sort by h(n)]
  - Complete, but not optimal.

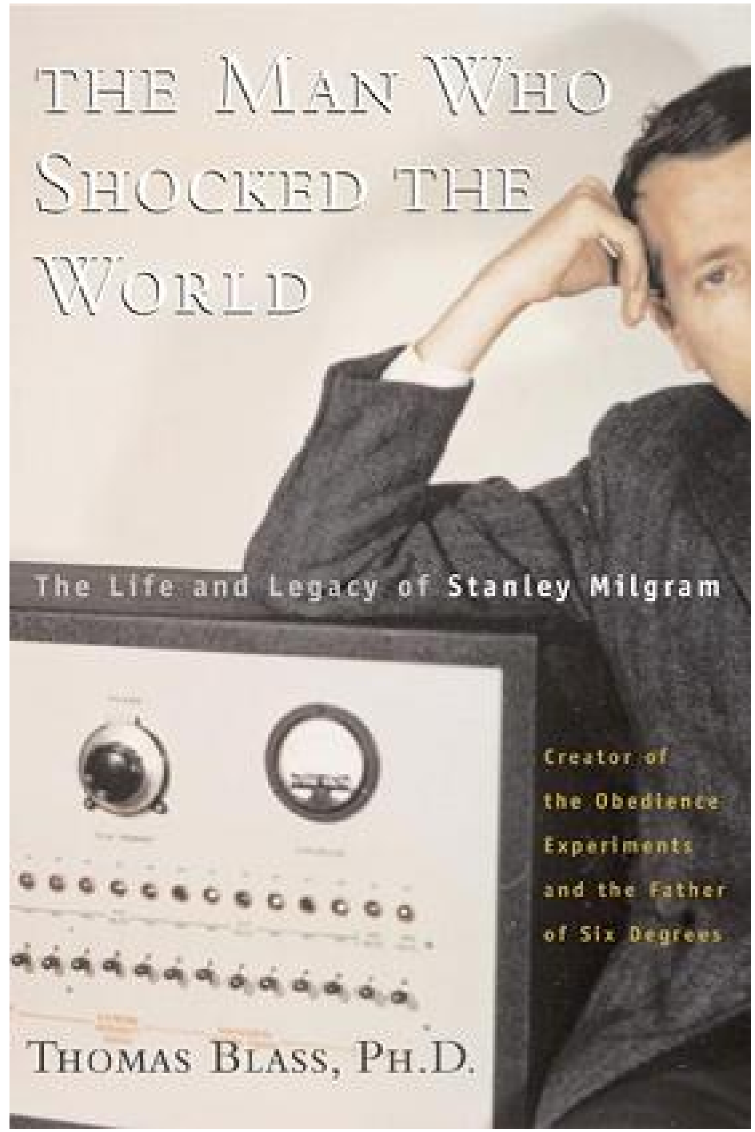
Search Map



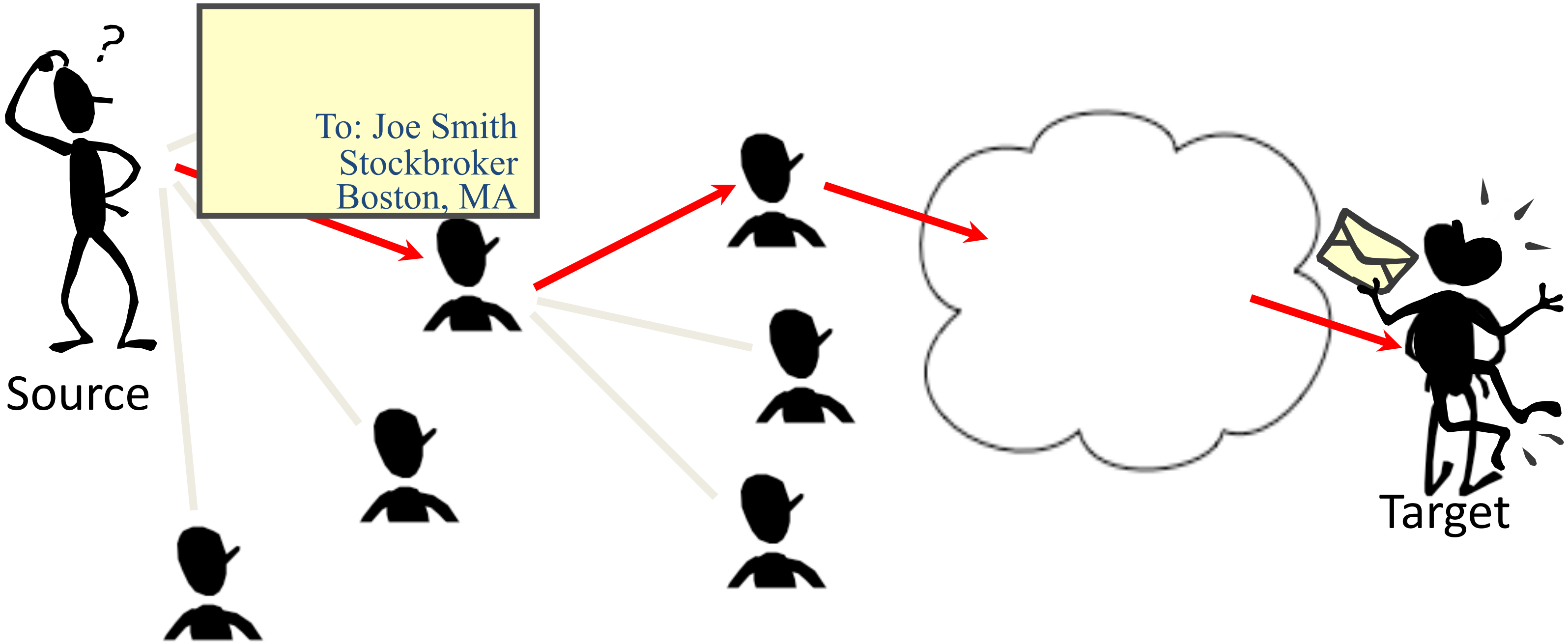
h(n) estima

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

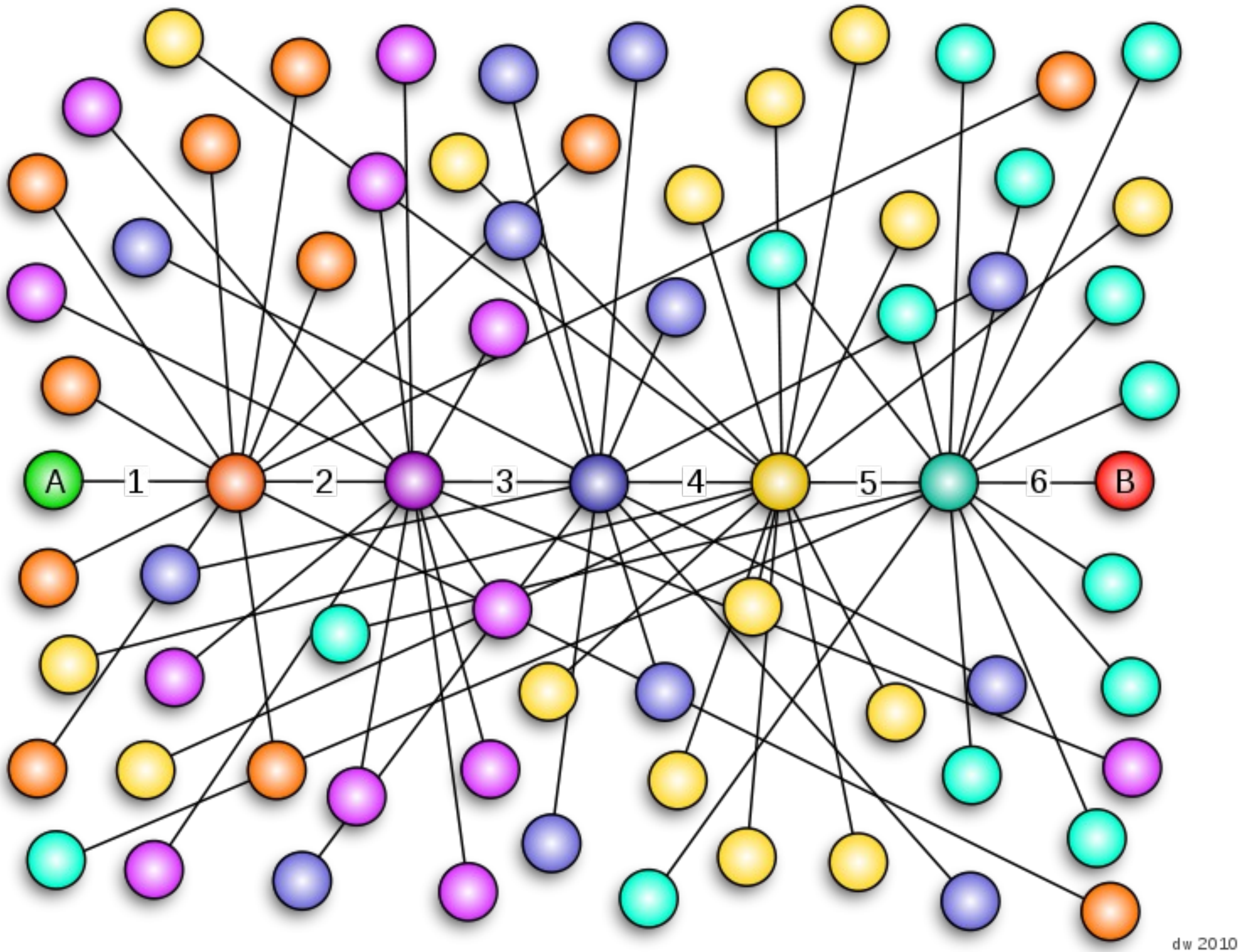
Stanley Milgram



Travers & Milgram (1969)



- 296 letters
- 22% reached target
- Median chain length = 6



A
1
2
3
4
5
6
B
dw 2010