Problem-Solving as Search

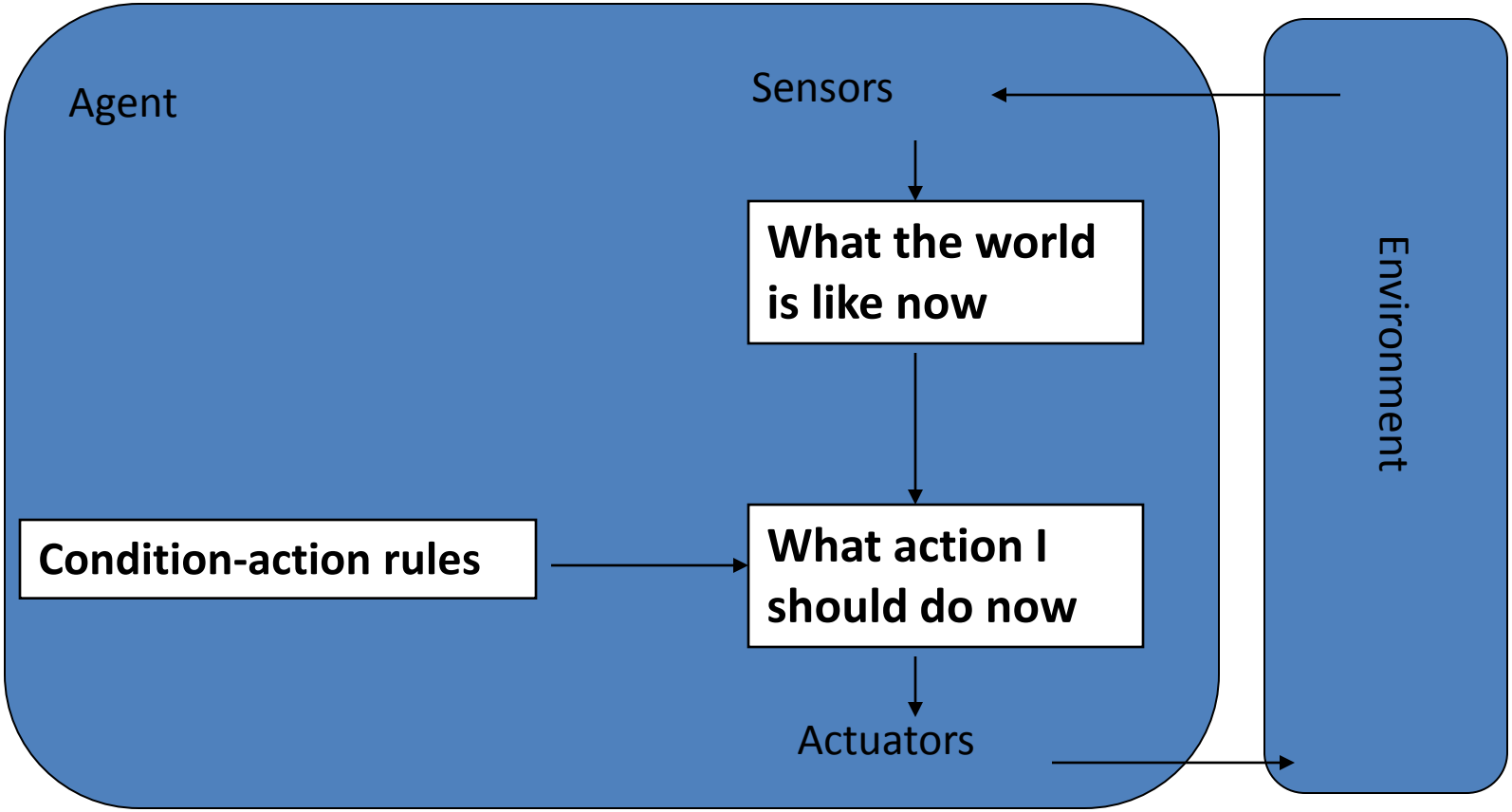
Intelligent Agents

Agent:
Anything that can be viewed as perceiving its **environment** through **sensors** and acting upon that environment through **actuators**.

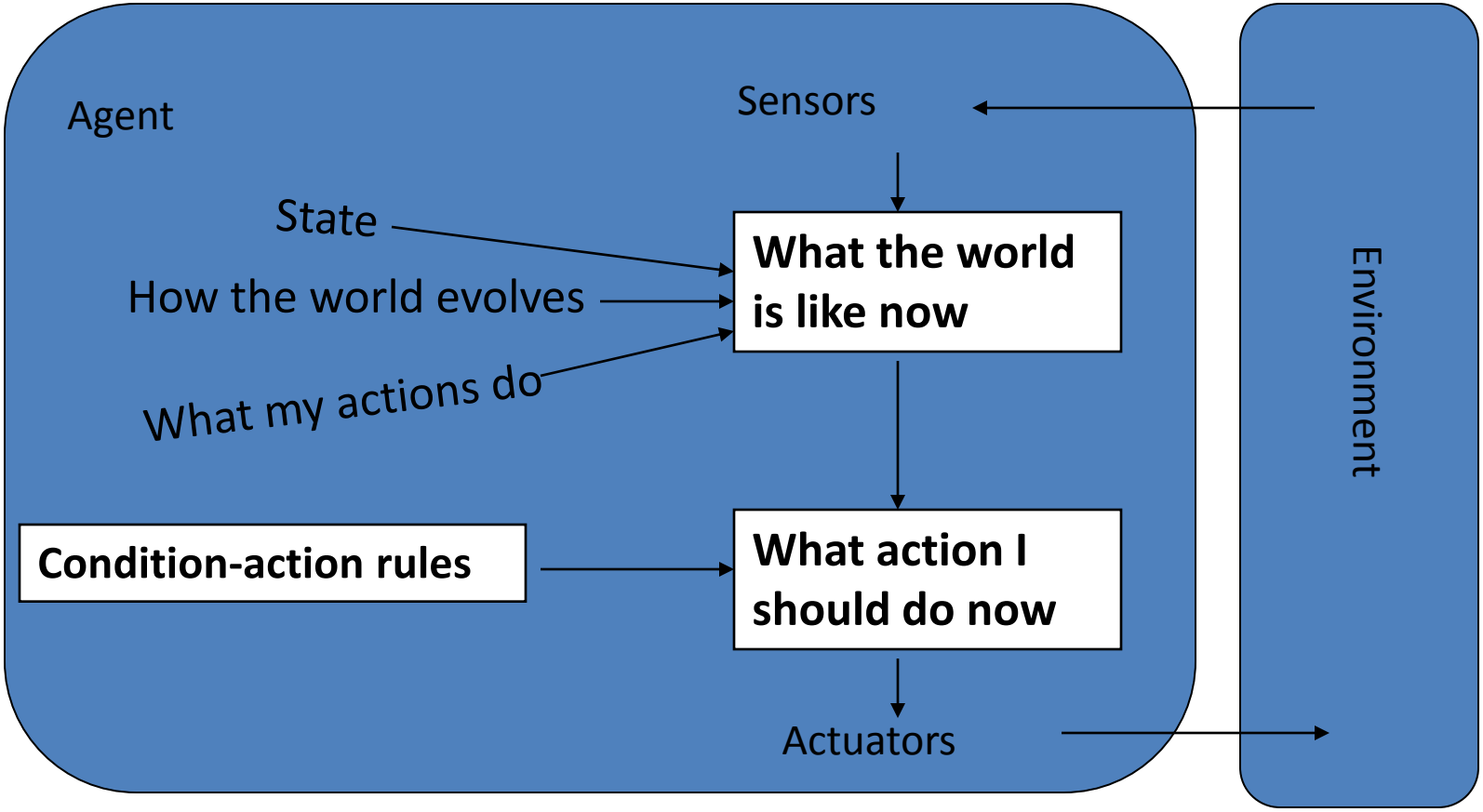
Agent Function:
Agent behavior is determined by the agent function that maps any given percept sequence to an action.

Agent Program:
The agent function for an artificial agent will be implemented by an agent program.

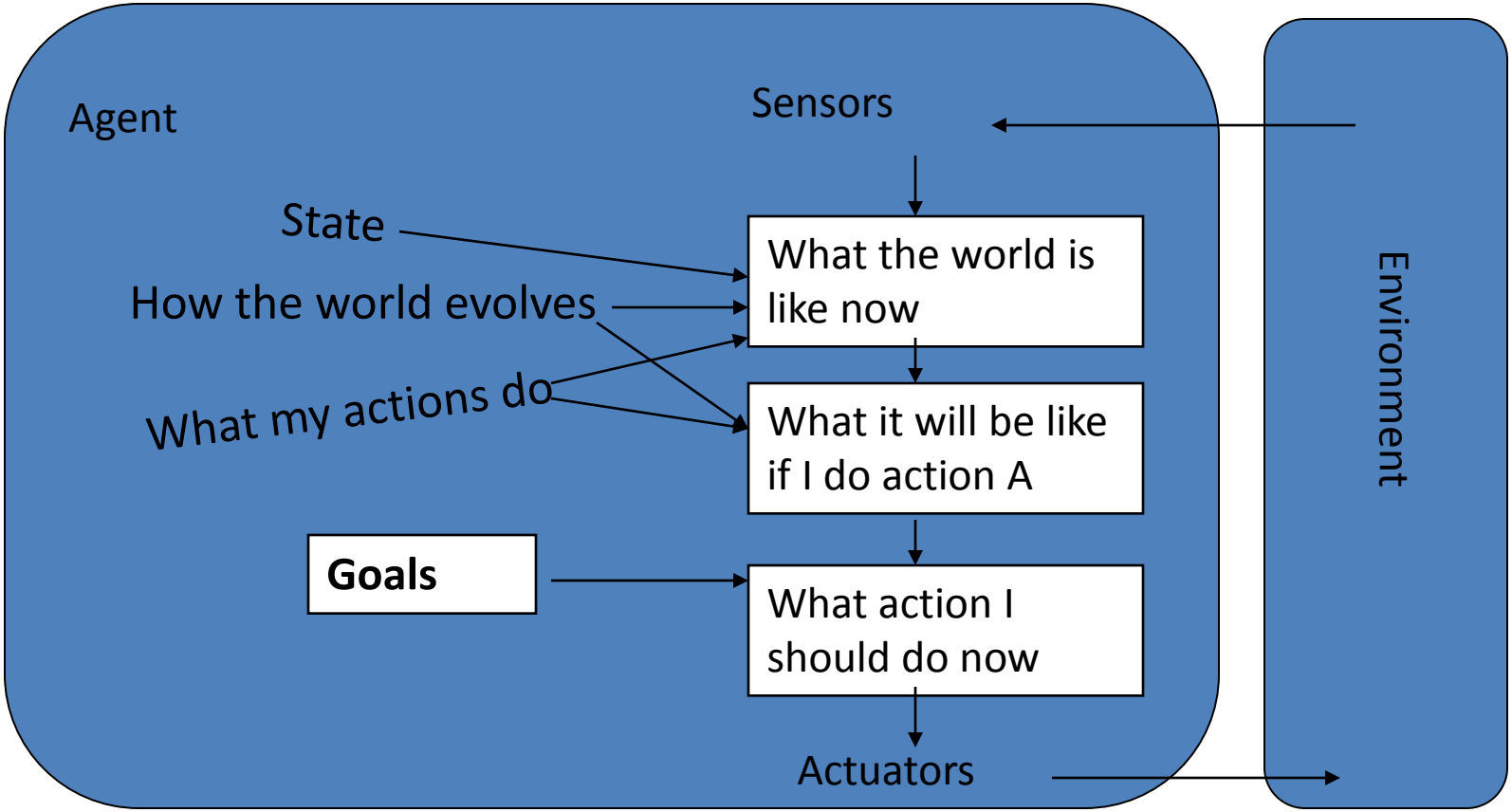
A Simple Reflex Agent



Agent with Model and Internal State



Goal-Based Agent

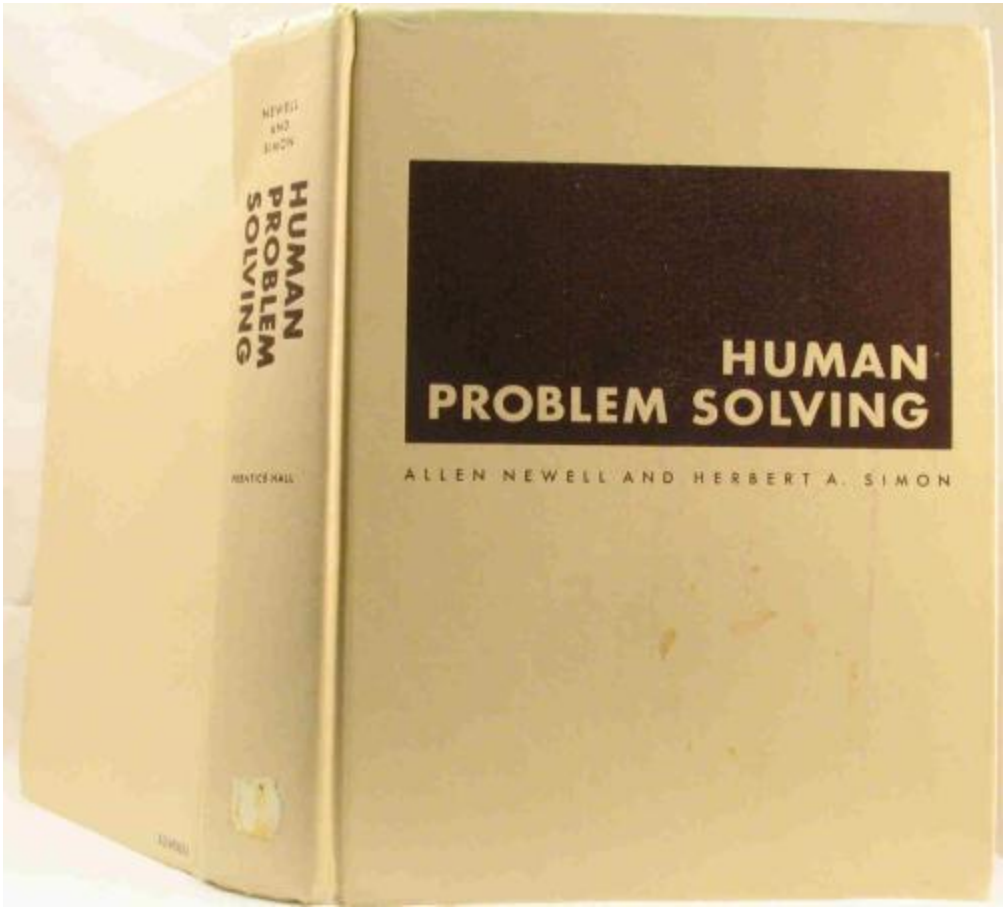


Schedule

- Search 
- Machine learning
- Knowledge based systems
- Discovery

Problem Solving as Search

- Search is a central topic in AI
  - Originated with Newell and Simon's work on problem solving.
  - Famous book: “Human Problem Solving” (1972)
- Automated reasoning is a natural search task
- More recently: Smarter algorithms
  - Given that almost all AI formalisms (planning, learning, etc.) are NP-complete or worse, some form of search is generally unavoidable (no “smarter” algorithm available).


NEWELL AND SIMON
HUMAN PROBLEM SOLVING
PRENTICE HALL
HUMAN PROBLEM SOLVING
ALLEN NEWELL AND HERBERT A. SIMON

Defining a Search Problem

State space - described by

initial state - starting state

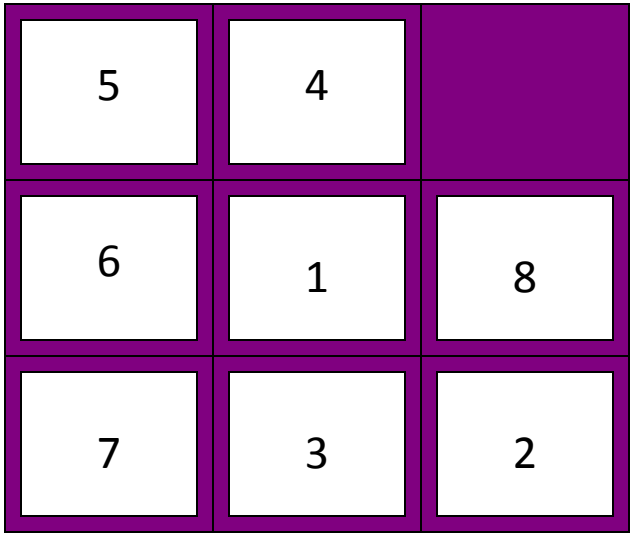
actions - possible actions available

successor function; operators - given a particular state *x*, returns a set of < *action*, *successor* > pairs

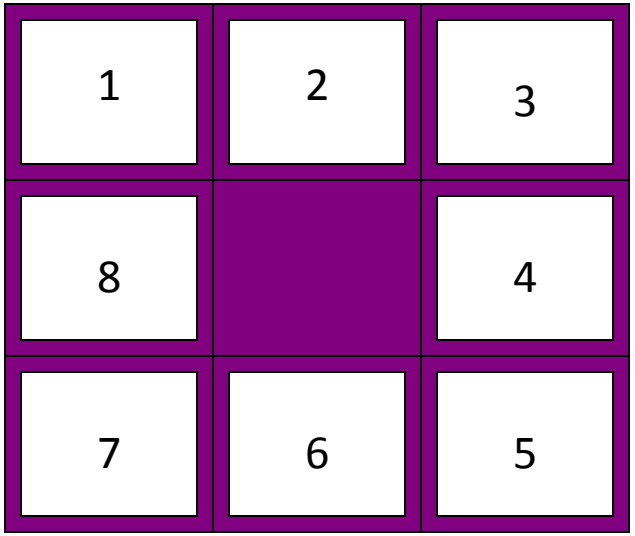
Goal test - determines whether a given state is a goal state (sometimes list, sometimes condition).

Path cost - function that assigns a cost to a path

The 8 Puzzle



Initial State



Goal State

Clicker

- **What is the size of the state space?**
  - **A. 4**
  - **B. 3x3**
  - **C. 9!**
  - **D. 9^9**
  - **E. Whatever**

Clicker

- **How many actions possible for each state (on average)?**
  - **A. ~1**
  - **B. ~4**
  - **C. ~9**
  - **D. ~9!**

Cryptarithmetic

```
  SEND
+ MORE
------
 MONEY
```

Find (non-duplicate) substitution of digits for letters such that the resulting sum is arithmetically correct.

Each letter must stand for a different digit.

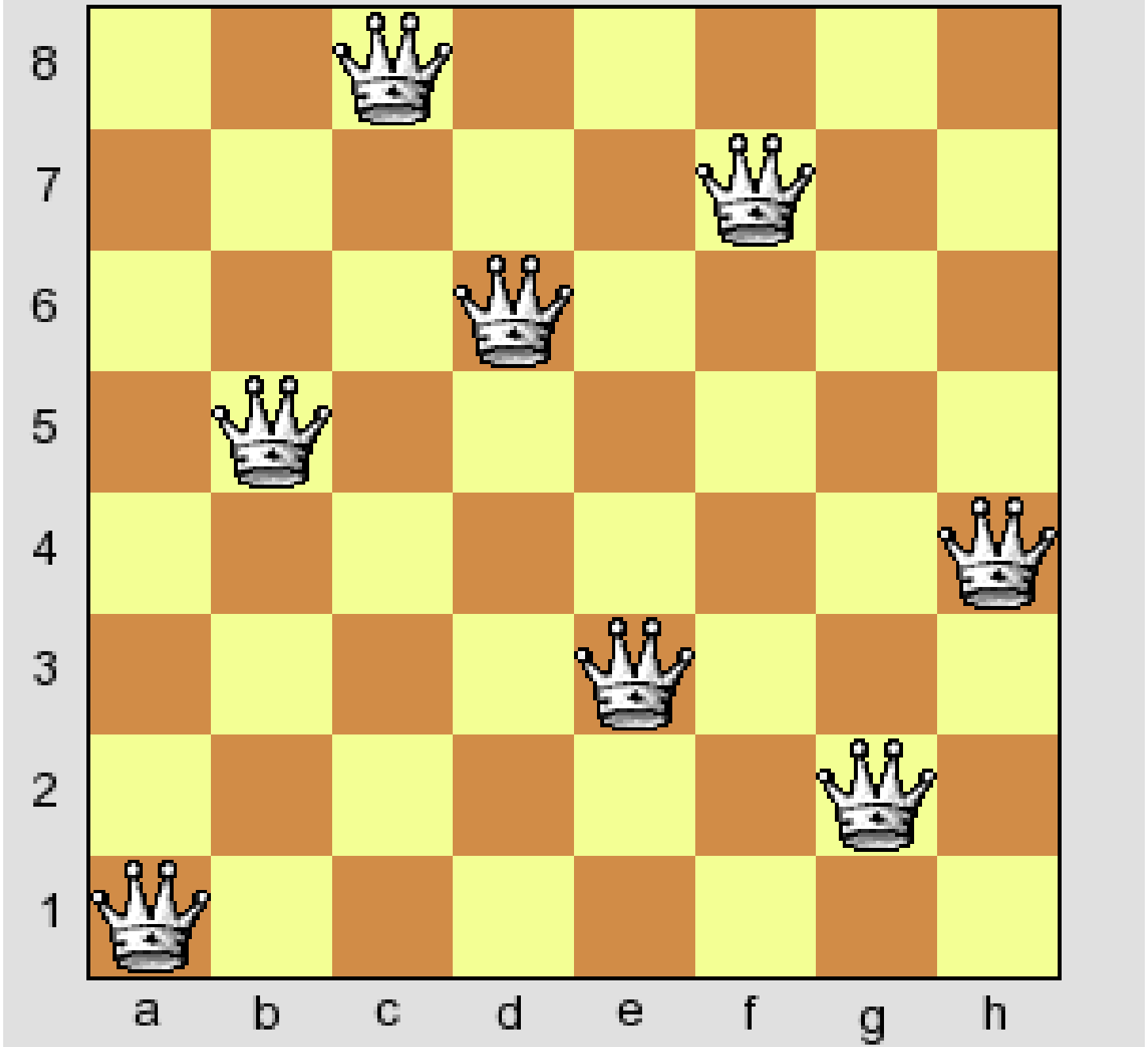
Solving a Search Problem: State Space Search

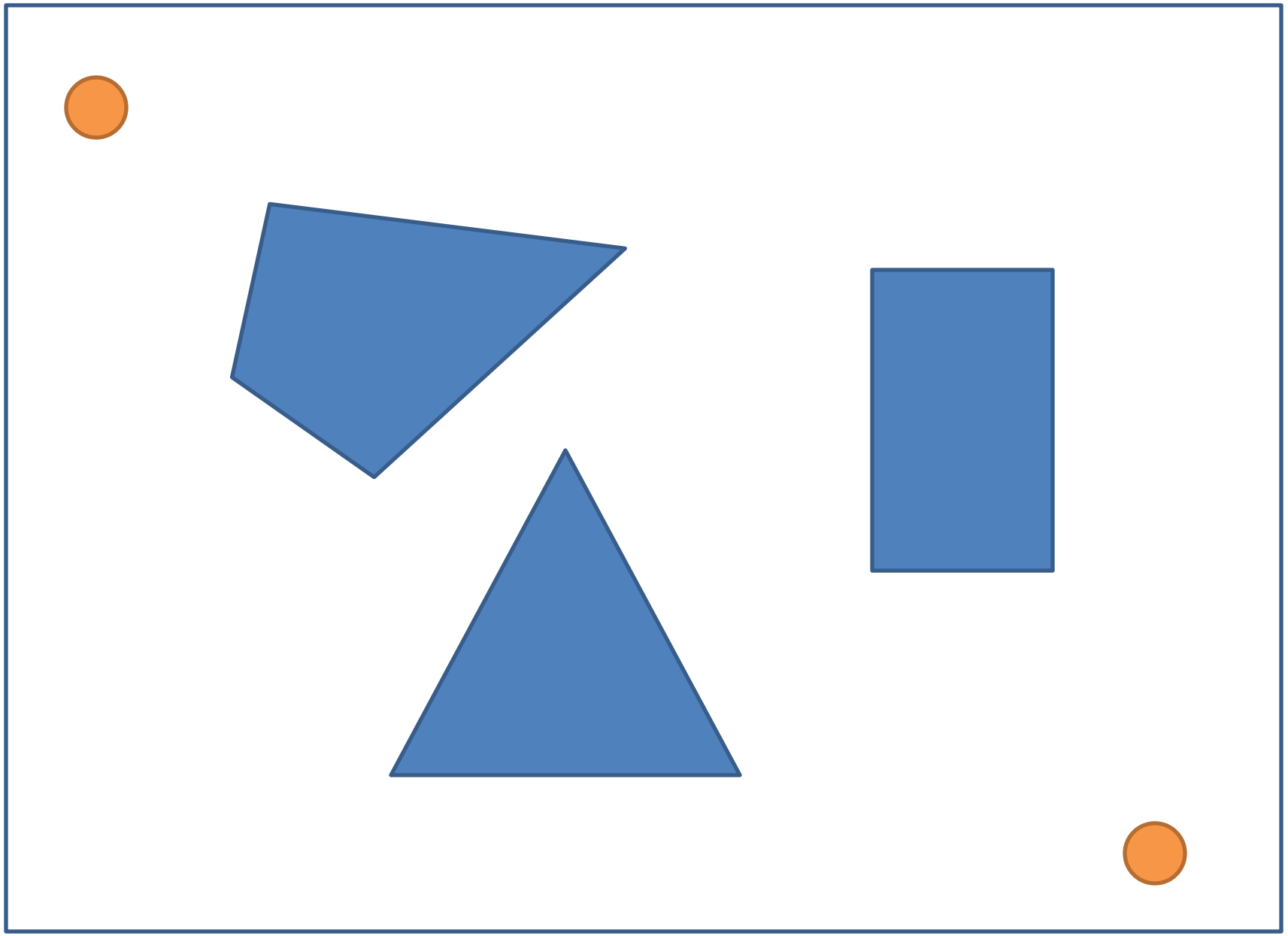
Input:

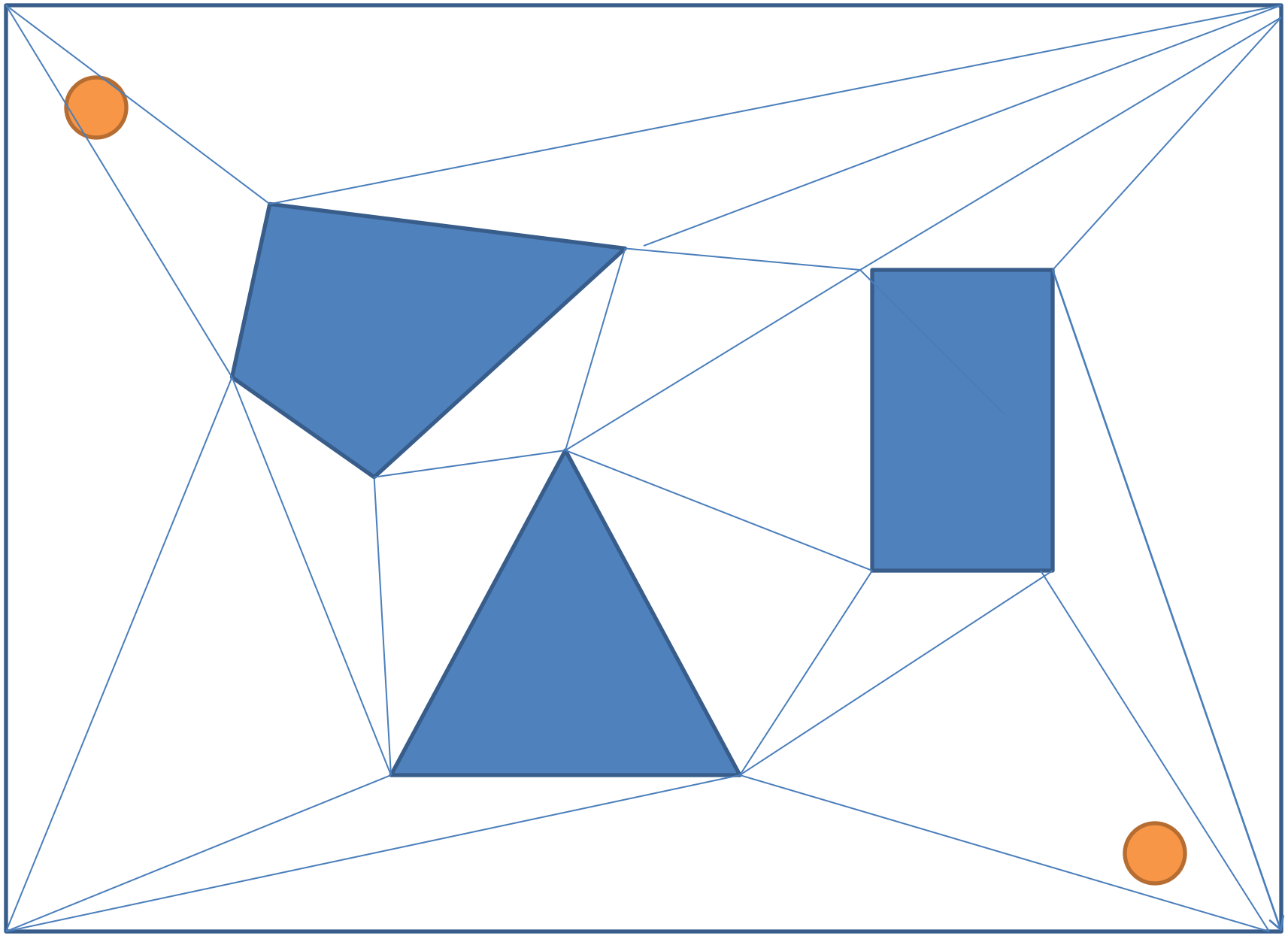
- Initial state
- Goal test
- Successor function
- Path cost function

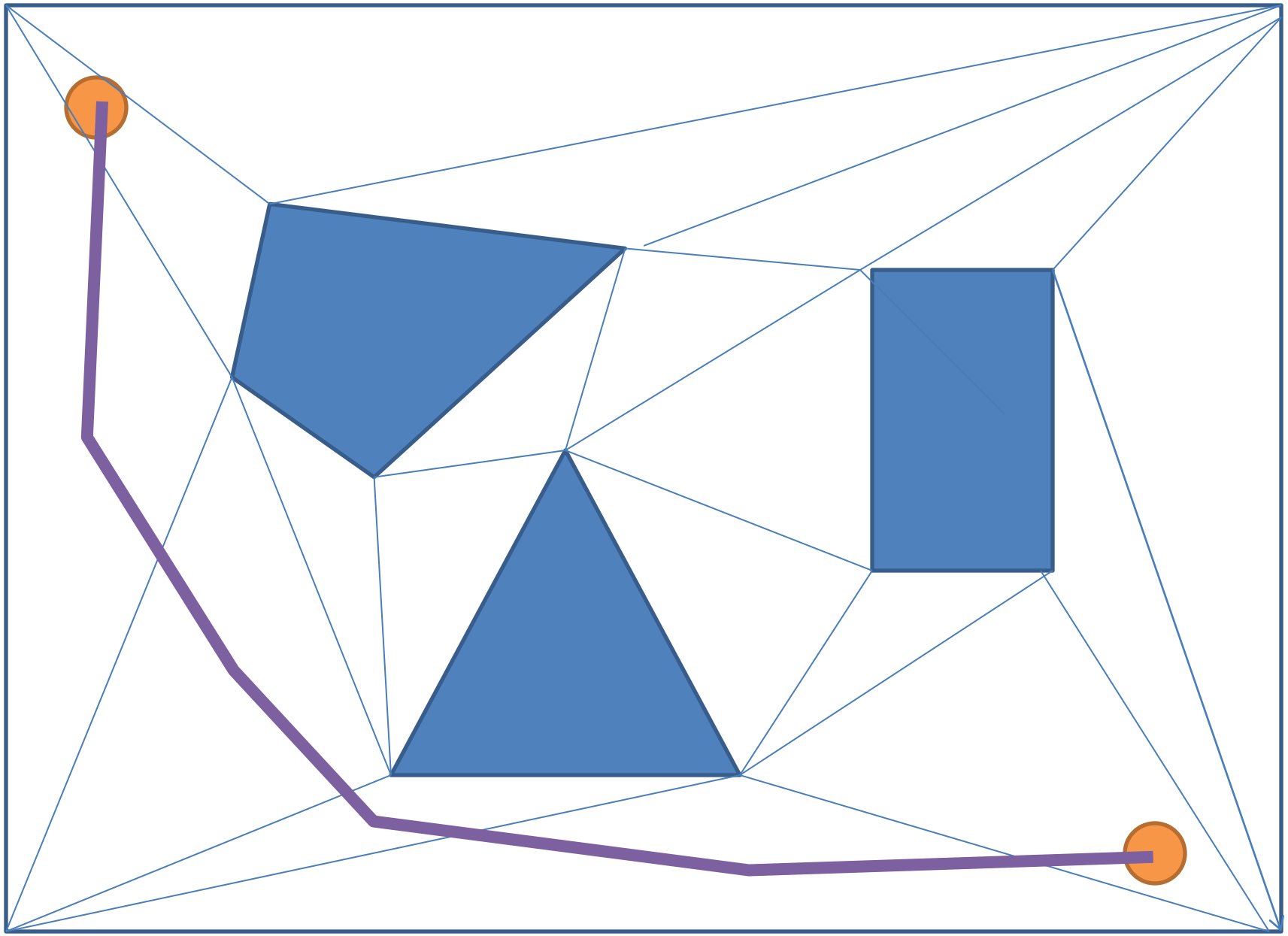
Output:

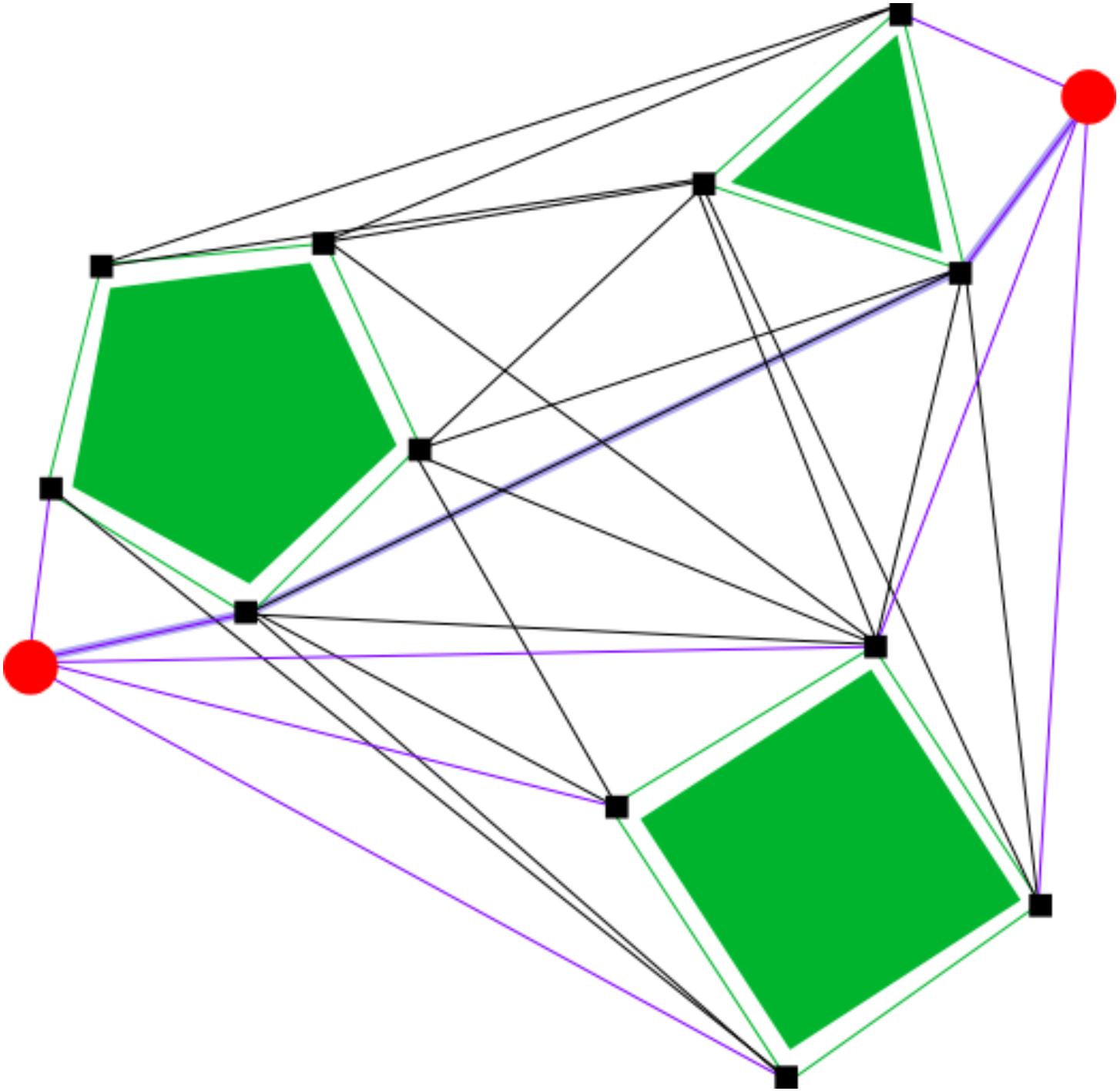
- Path from initial state to goal state.
- Solution quality is measured by the path cost.


8
7
6
5
4
3
2
1
a
b
c
d
e
f
g
h









Generic Search Algorithm

```
L = make-list(initial-state)
loop
   node = remove-front(L) (node contains path
                           of how the algorithm got
there)
   if goal-test(node) == true then
      return(path to node)
   S = successors (node)
   insert (S,L)
until L is empty
return failure
```

Search procedure defines a search tree

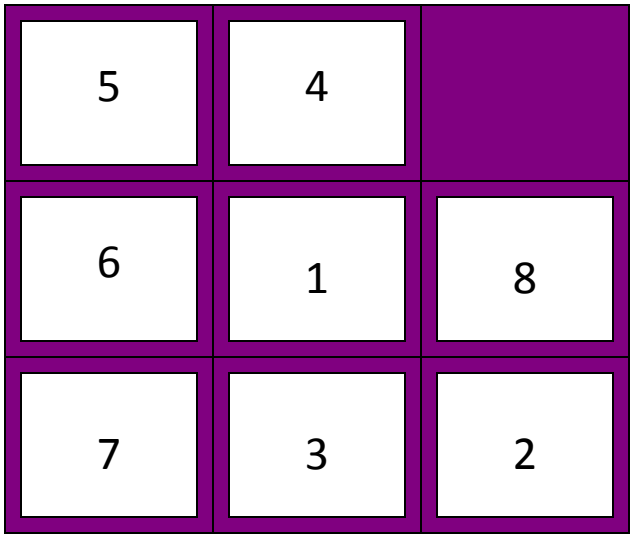
Search tree

- root node - initial state
- children of a node - successor states
- fringe of tree - L: states not yet expanded

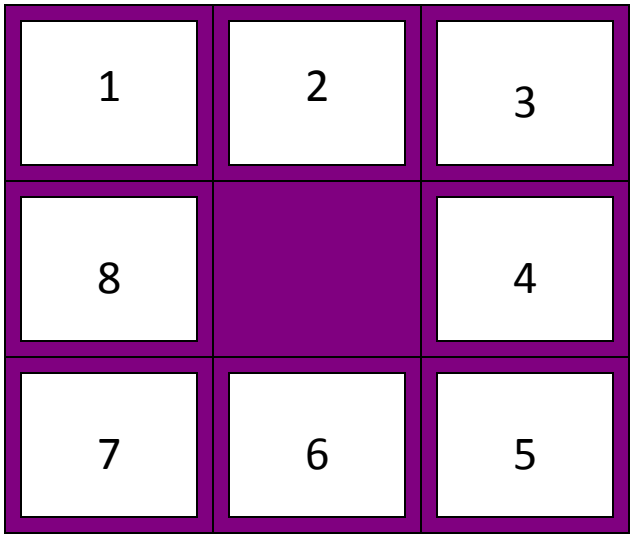
Search strategy - algorithm for deciding which leaf node to expand next.

- stack: Depth-First Search (DFS).
- queue: Breadth-First Search (BFS).

Solving the 8-Puzzle



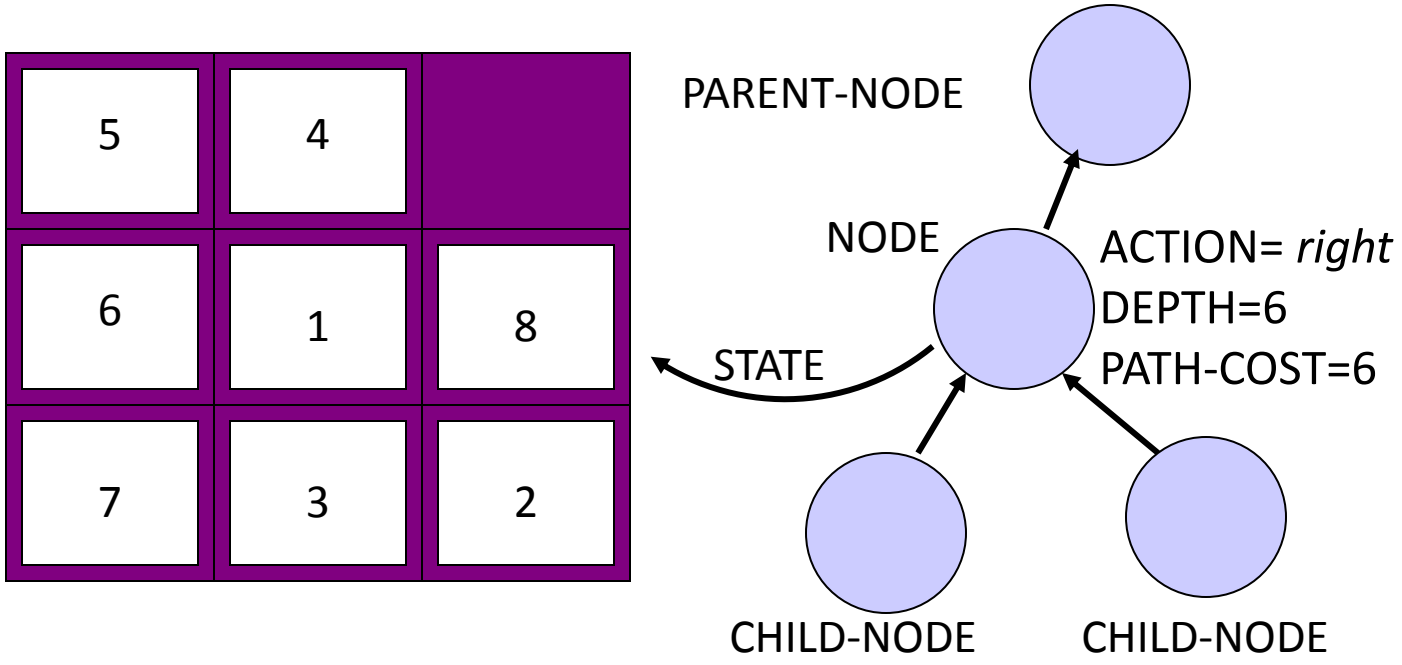
Start State



Goal State

What would the search tree look like after the start state was expanded?

Node Data Structure



Sliding Block Puzzles

- 8-puzzle (on 3x3 grid) has 181,440 states
  - Easily solvable from any random position
- 15-puzzle (on 4x4 grid) has ~1.3 Trillion states
  - Solvable in a few milliseconds
- 24-puzzle (on 5x5 grid) has ~10^{25} states
  - Difficult to solve

Evaluating a Search Strategy

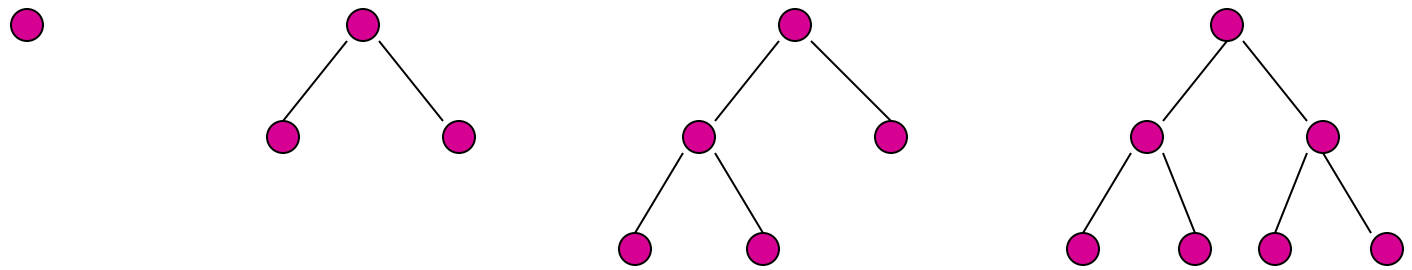
Completeness:
Is the strategy guaranteed to find a solution when there is one?

Time Complexity:
How long does it take to find a solution?

Space Complexity:
How much memory does it need?

Optimality:
Does strategy always find a lowest-cost path to solution? (this may include different cost of one solution vs. another).

Uninformed search: BFS

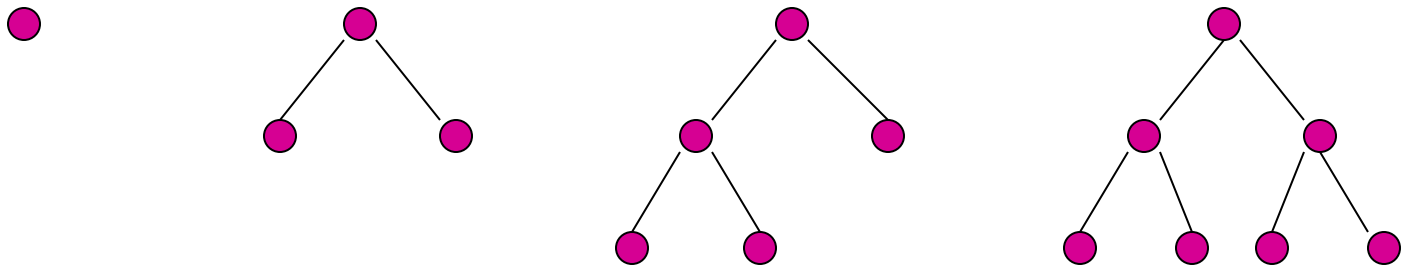


Consider paths of length 1, then of length 2, then of length 3, then of length 4,....

Time and Memory Requirements for BFS – O(b^{d+1})

Let b = branching factor, d = solution depth, then the maximum number of nodes *generated* is:

$$b + b^2 + \ldots + b^d + (b^{d+1} - b)$$



Time and Memory Requirements for BFS – $O(b^{d+1})$

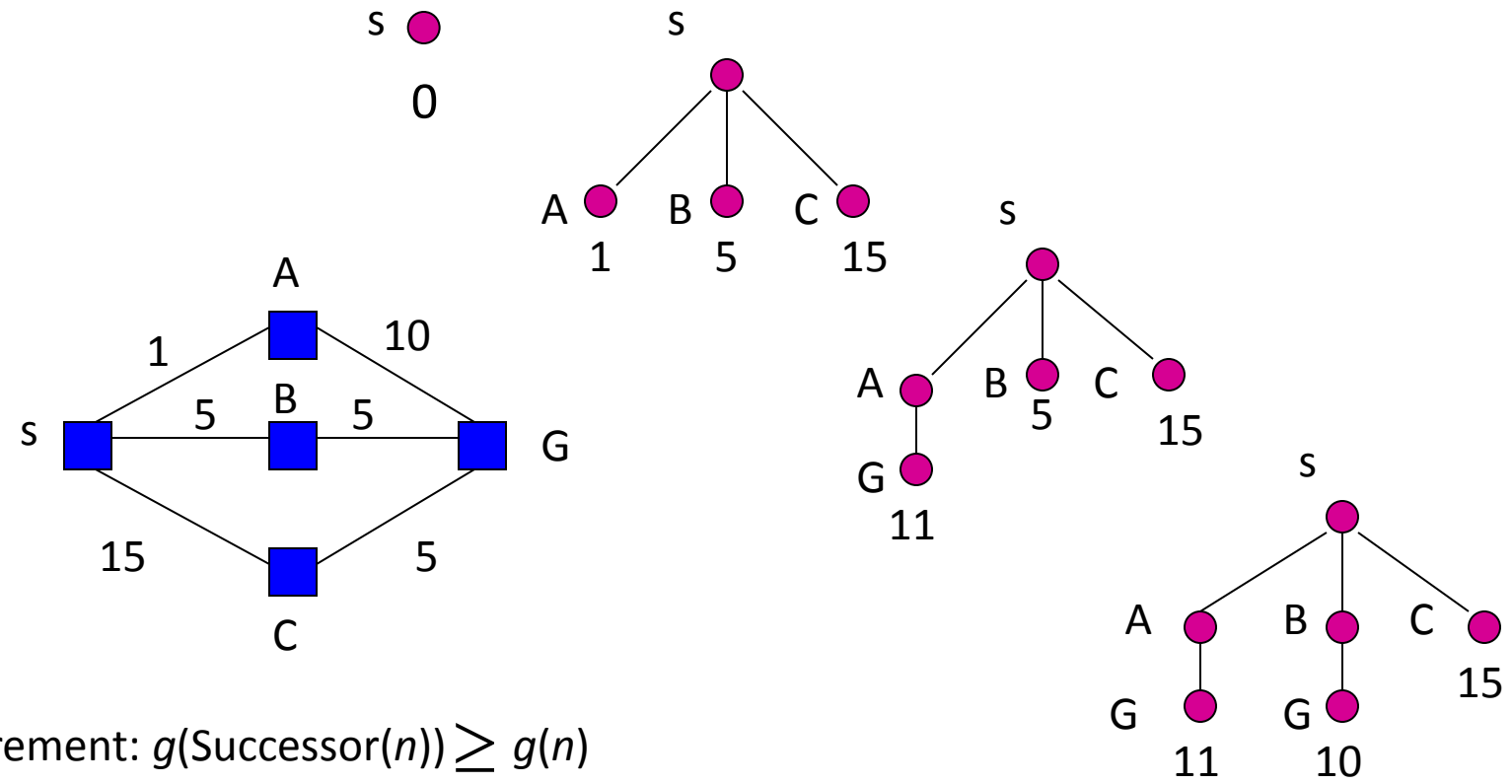
Example:

- b = 10
- 10,000 nodes/second
- each node requires 1000 bytes of storage

Depth	Nodes	Time	Memory
2	1100	.11 sec	1 meg
4	111,100	11 sec	106 meg
6	10^7	19 min	10 gig
8	10^9	31 hrs	1 tera
10	10^{11}	129 days	101 tera
12	10^{13}	35 yrs	10 peta
14	10^{15}	3523 yrs	1 exa

Uniform-cost Search

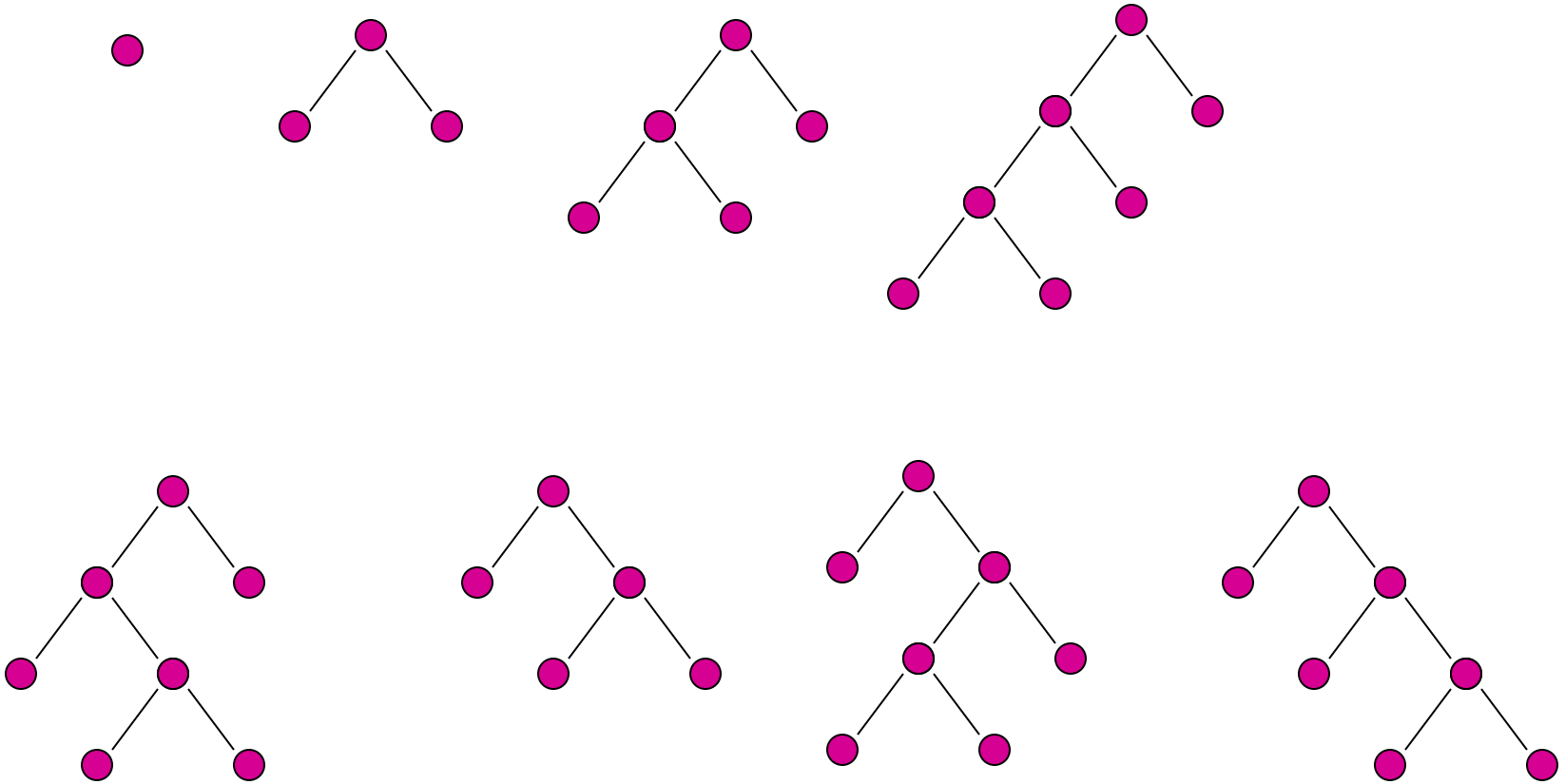
Use BFS, but always expand the lowest-cost node on the fringe as measured by path cost *g(n).*



Requirement: $g(\text{Successor}(n)) \geq g(n)$

Always expand lowest cost node in open-list.
Goal-test only before expansion, not after generation.

Uninformed search: DFS



DFS vs. BFS

	Complete	Optimal	Time	Space
BFS	YES	YES	$O(b^{d+1})$	$O(b^{d+1})$
DFS	Finite depth	NO	$O(b^{m})$	$O(bm)$

m is maximum search depth *d* is solution depth *b* is branching factor

Time

m = *d*: DFS typically wins
m > *d*: BFS might win
m **is infinite**: BFS probably will do better

Space

DFS almost always beats BFS

Which search should I use...

If there may be infinite paths?

B=BFS **D=DFS**

Which search should I use...

If goal is at a known depth?

B=BFS **D=DFS**

Which search should I use...

If there is a large (possibly infinite) branching factor?

B=BFS **D=DFS**

Which search should I use...

If there are lots of solutions?

B=BFS **D=DFS**

Backtracking Search

Idea:

DFS, but don't expand all *b* states before next level

Generate the next state as needed (e.g. from previous state)

Uses only O(m) storage

Important when space required to store each state is very large (e.g. assembly planning)

Iterative Deepening [Korf 1985]

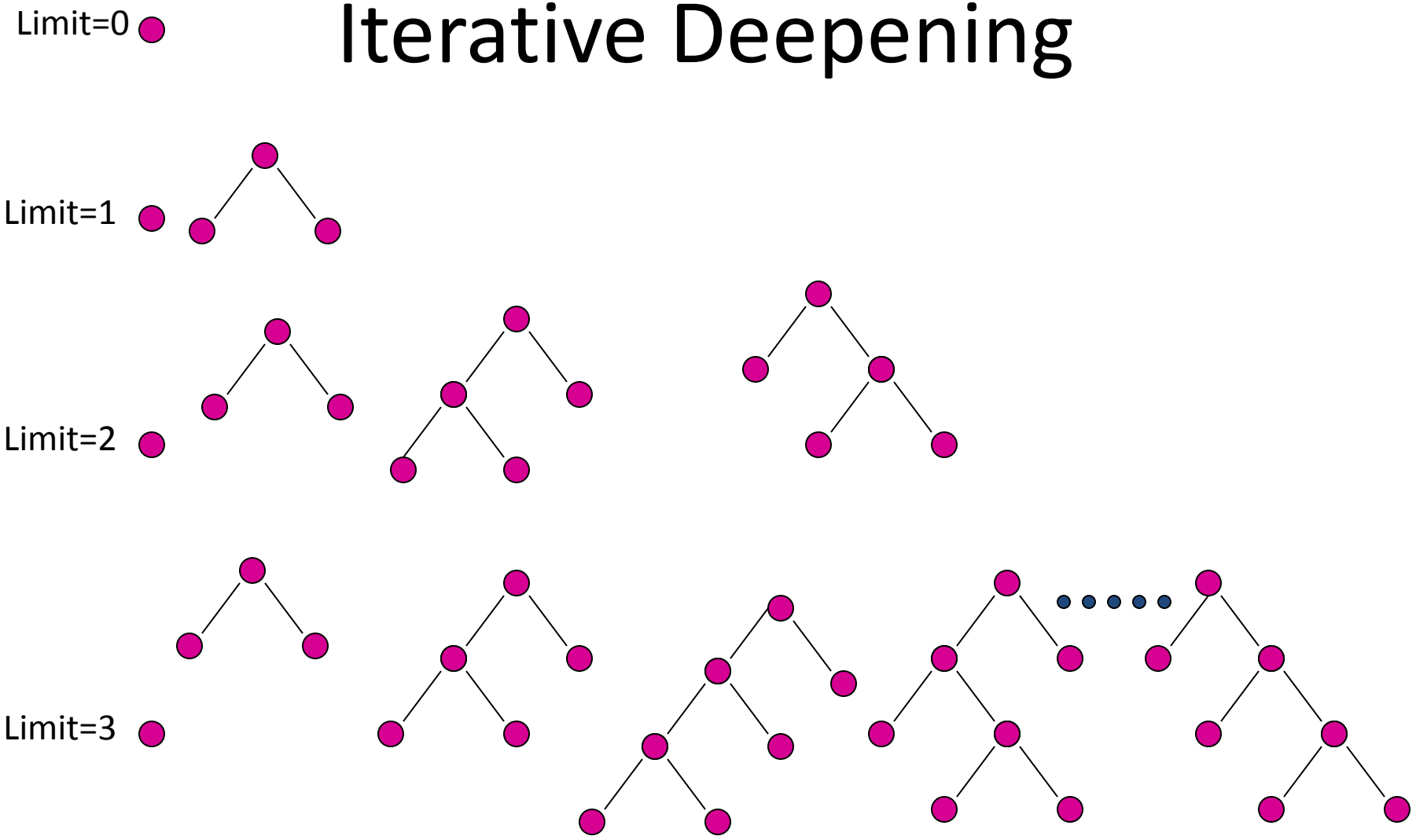
Idea:

Use an *artificial* depth cutoff, *c*.

If search to depth c succeeds, we're done.
If not, increase c by 1 and start over.

Each iteration searches using depth-limited DFS.

Iterative Deepening

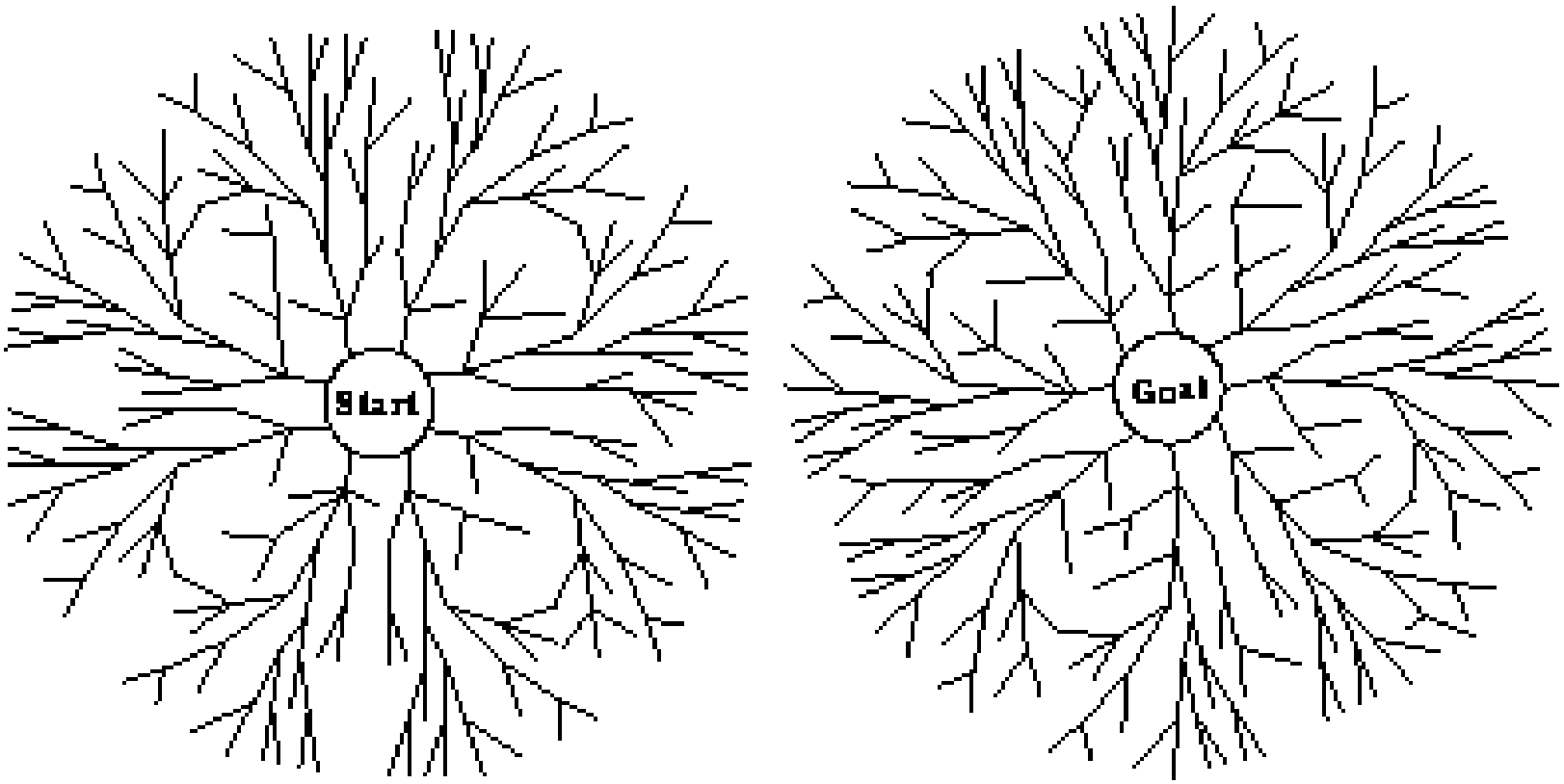


Cost of Iterative Deepening

space: *O(bd)* as in DFS, **time:** $O(b^d)$

b	ratio of IDS to DFS
2	3
3	2
5	1.5
10	1.2
25	1.08
100	1.02

Bidirectional Search



(from AIMA Figure 3.17)

Comparing Search Strategies

Criterion	Breadth-First	Uniform-Cost	Depth-First	Iterative Deepening	Bidirectional (if applicable)
Time	b^{d+1}	$b^{1+\frac{C^*}{\epsilon}}$	b^m	b^d	$b^{d/2}$
Space	b^{d+1}	$b^{1+\frac{C^*}{\epsilon}}$	bm	bd	$b^{d/2}$
Optimal?	Yes	yes	no	yes	yes
Complete?	Yes	Yes	No	Yes	Yes

***Note that many of the ``yes's'' above have caveats, which we discussed when covering each of the algorithms.