CMSC 471
Artificial Intelligence

Search

KMA Solaiman – ksolaima@umbc.edu

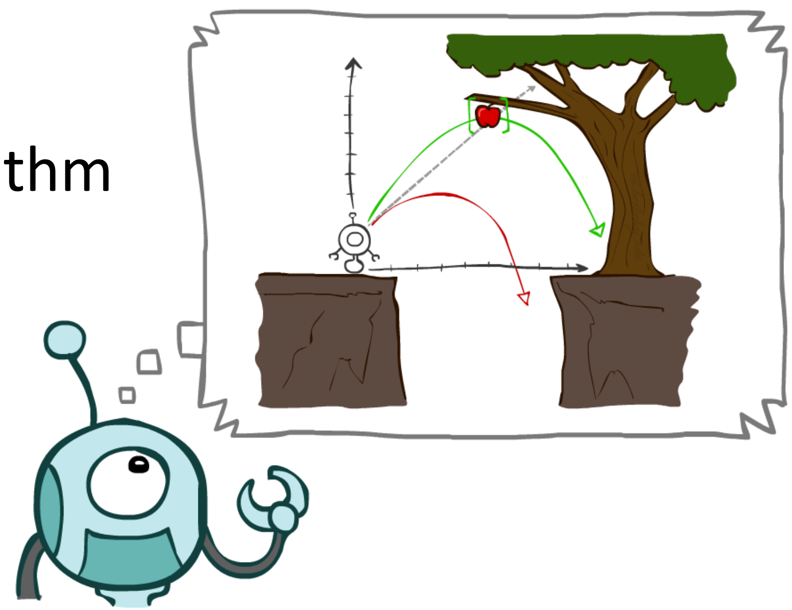
Slide adoption courtesy: Tim Finin, Frank Ferraro (UMBC),
Charles R. Dyer (UW-Madison), Dan Klein and Pieter Abbeel (CS188, UC Berkeley)

Today

- Agents that Plan Ahead
- Goal-based agents

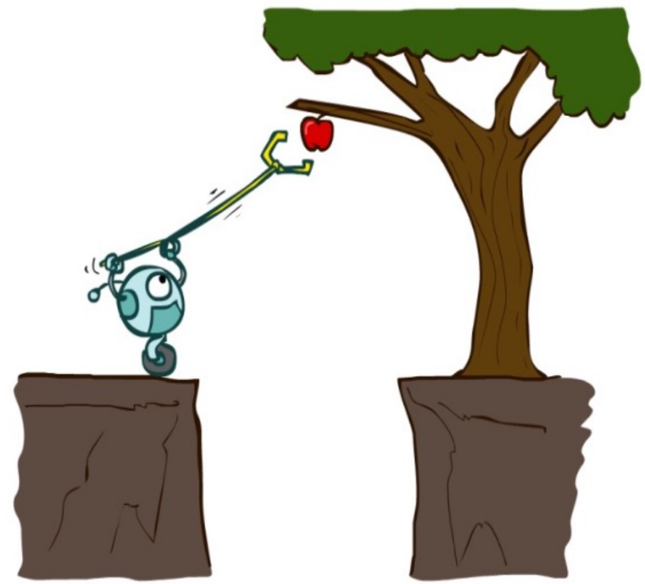
- Search Problems
- Generic state-space search algorithm

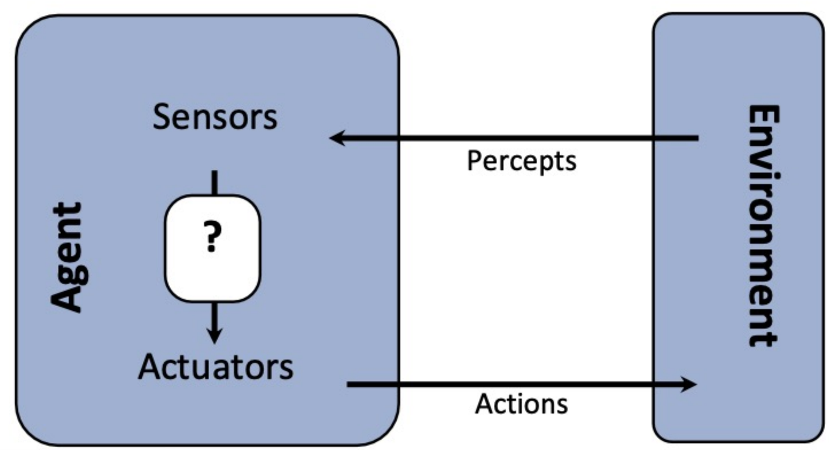
- Uninformed Search Methods
  - Depth-First Search
  - Breadth-First Search
  - Uniform-Cost Search



How do you design an intelligent agent?

- An **agent** is an entity that perceives and acts
- **Intelligent agents** perceive environment via **sensors** and act *rationally* on them with their **effectors**
- *Discrete* agents receive **percepts** one at a time, and map them to a sequence of discrete **actions**





Courtesy Tim Finin

Characteristics of environments

	Fully observable?	Deterministic?	Episodic?	Static?	Discrete?	Single agent?
Solitaire	No	Yes	Yes	Yes	Yes	Yes
Backgammon	Yes	No	No	Yes	Yes	No
Taxi driving	No	No	No	No	No	No
Internet shopping	No	No	No	No	Yes	No
Medical diagnosis	**No**	**No**	**No**	**No**	**No**	**Yes**

A **Yes** in a cell means that aspect is simpler; a **No** more complex

Courtesy Tim Finin

Characteristics of environments

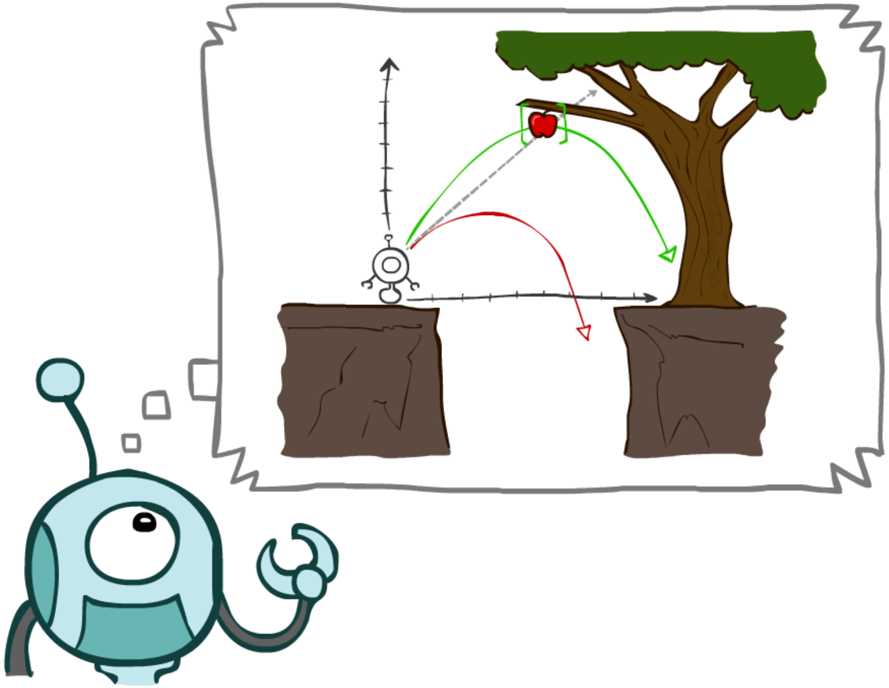
→ Lots of real-world domains fall into the hardest case!

	Fully observable?	Deterministic?	Episodic?	Static?	Discrete?	Single agent?
Solitaire	No	Yes	Yes	Yes	Yes	Yes
Backgammon	Yes	No	No	Yes	Yes	No
Taxi driving	No	No	No	No	No	No
Internet shopping	No	No	No	No	Yes	No
Medical diagnosis	**No**	**No**	**No**	**No**	**No**	**Yes**

A **Yes** in a cell means that aspect is simpler; a **No** more complex

Courtesy Tim Finin

Agents that Plan



Recap

simple

(0) Table-driven agents

Use percept sequence/action table to find next action. Implemented by a **lookup table**

(1) Simple reflex agents

Based on **condition-action rules**, stateless devices with no memory of past world states

(2) Agents with memory

represent states and keep track of past world states

(3) Agents with goals

Have a state and **goal information** describing desirable situations; can take future events into consideration

(4) Utility-based agents

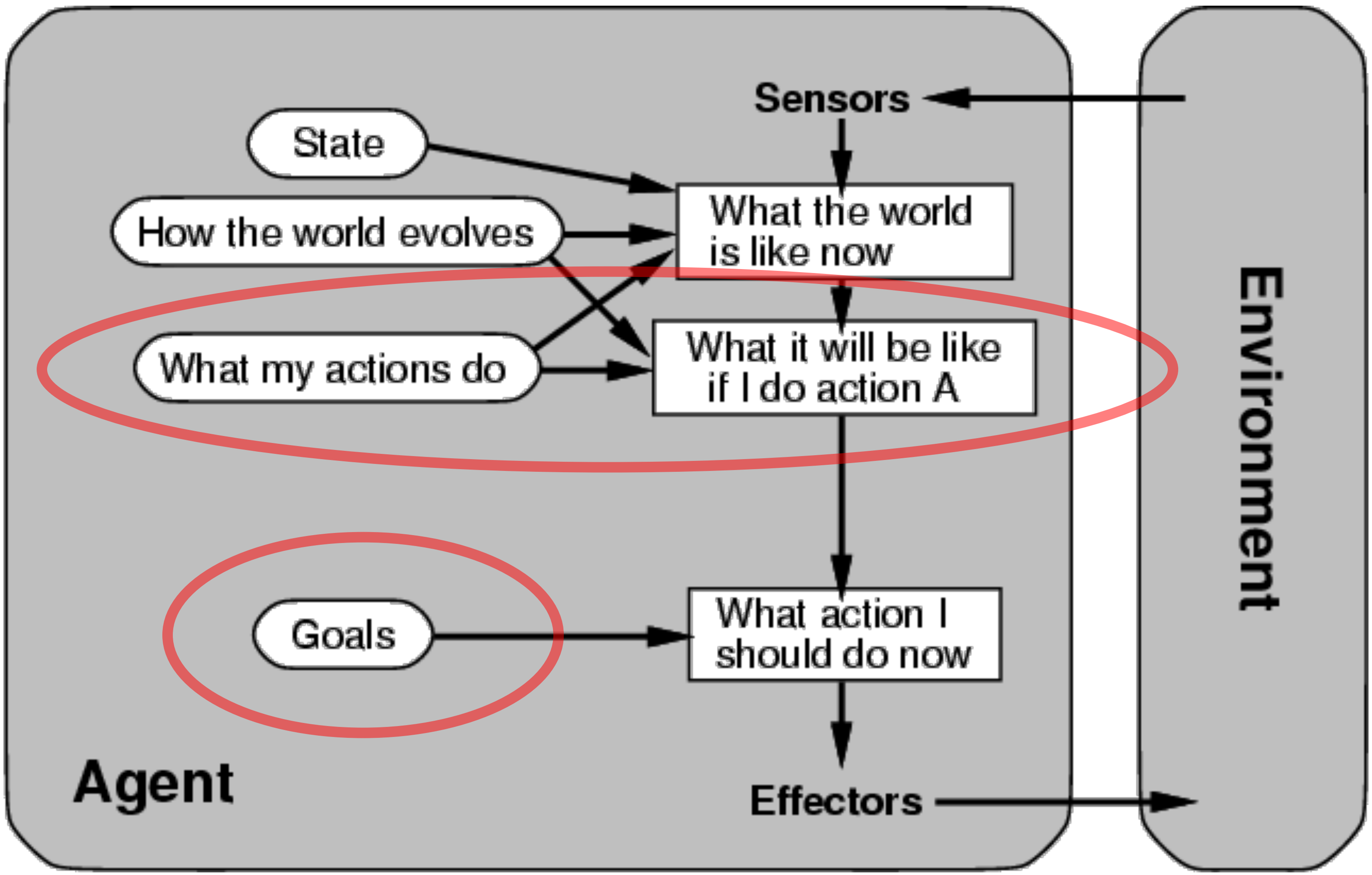
base decisions on **utility theory** in order to act rationally

complex

Recap

(3) Architecture for goal-based agent

state and **goal information** describe desirable situations allowing agent to take future events into consideration

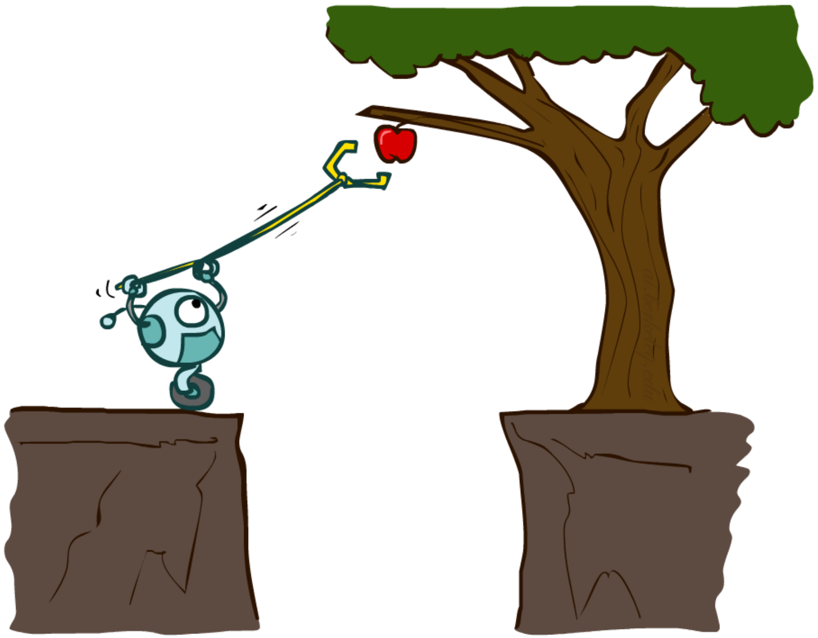


Courtesy Tim Finin

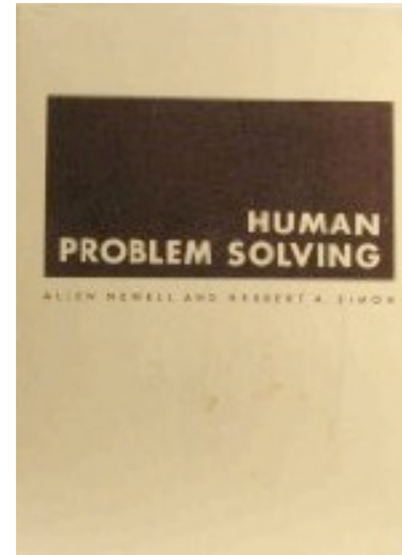
Planning Agents

- Planning agents:
  - Ask "what if"
  - Decisions based on (hypothesized) consequences of actions
  - Must have a model of how the world evolves in response to actions
  - Must formulate a goal (test)
  - Consider how the world WOULD BE

- Optimal vs. complete planning



Big Idea

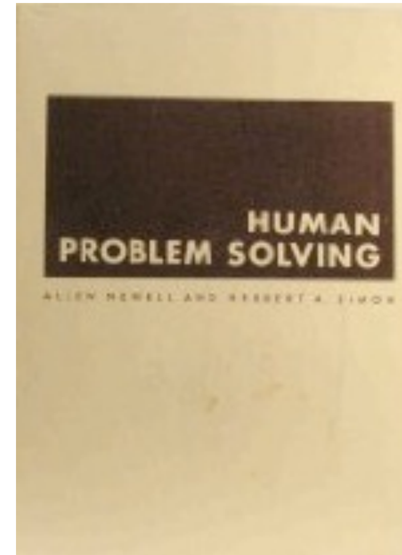


Allen Newell and Herb Simon developed the *problem space principle* as an AI approach in the late 60s/early 70s

"The rational activity in which people engage to solve a problem can be described in terms of (1) a set of **states** of knowledge, (2) **operators** for changing one state into another, (3) **constraints** on applying operators and (4) **control** knowledge for deciding which operator to apply next."

Newell A & *Simon* H A. *Human problem solving*. Englewood Cliffs, NJ: Prentice-Hall. 1972.

Big Idea



Allen Newell and Herb Simon developed the *problem space principle* as an AI approach in the late 60s/early 70s

"The rat … e a
problem … **tes**
of know … nto
another … 4)
control … pply
next."

We'll achieve this by formulating an appropriate graph and then applying graph search algorithms to it

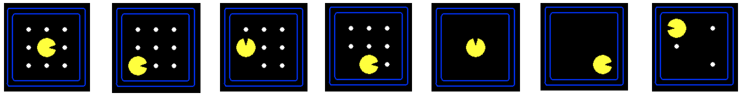
Newell A & *Simon* H A. *Human problem solving*. Englewood Cliffs, NJ: Prentice-Hall. 1972.

Search Problems: Key Terms

- A search problem consists of:

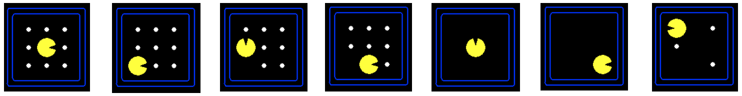
Search Problems: Key Terms

- A search problem consists of:
  - A state space



Search Problems: Key Terms

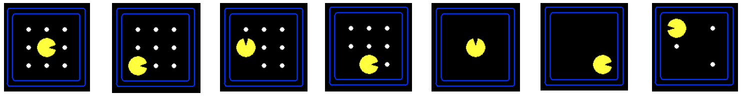
- A search problem consists of:
  - A state space



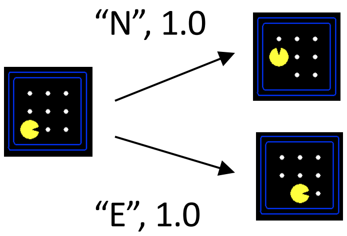
  - A successor function (with actions, costs)

Search Problems: Key Terms

- A search problem consists of:
  - A state space

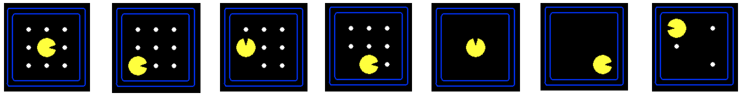


  - A successor function (with actions, costs)

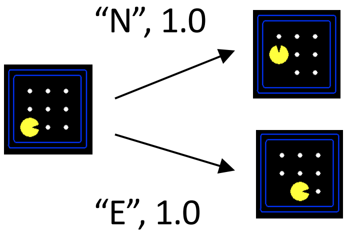


Search Problems: Key Terms

- A search problem consists of:
  - A state space



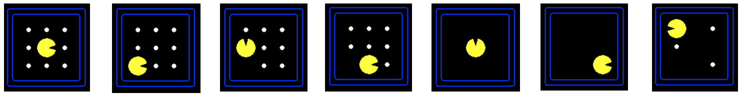
  - A successor function (with actions, costs)



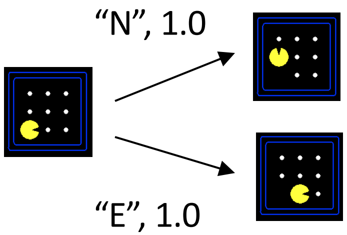
  - A start state and a goal test

Search Problems: Key Terms

- A search problem consists of:
  - A state space



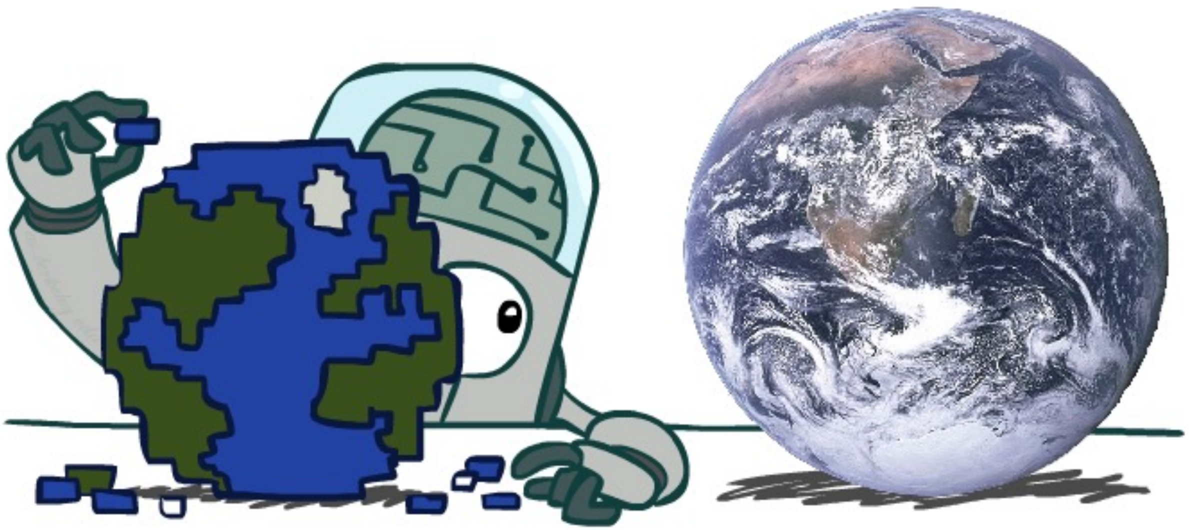
  - A successor function (with actions, costs)



  - A start state and a goal test
- A solution is a sequence of actions (a plan) which transforms the start state to a goal state

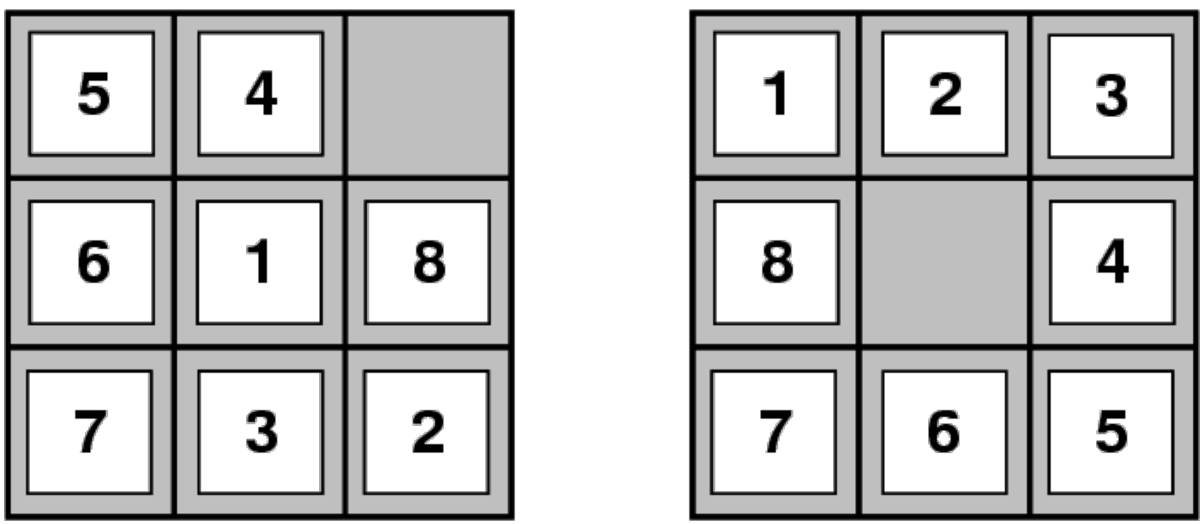
Search Problems Are Models

Search Problems Are Models



Example: 8-Puzzle

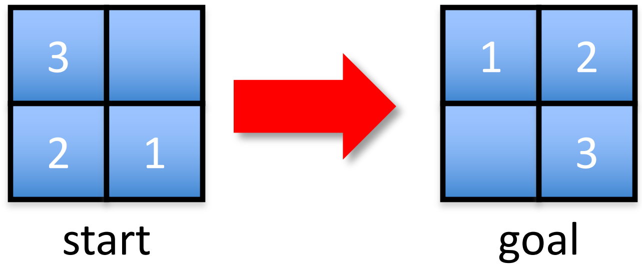
Given an initial configuration of 8 numbered tiles on a 3x3 board, move the tiles to produce a desired goal configuration

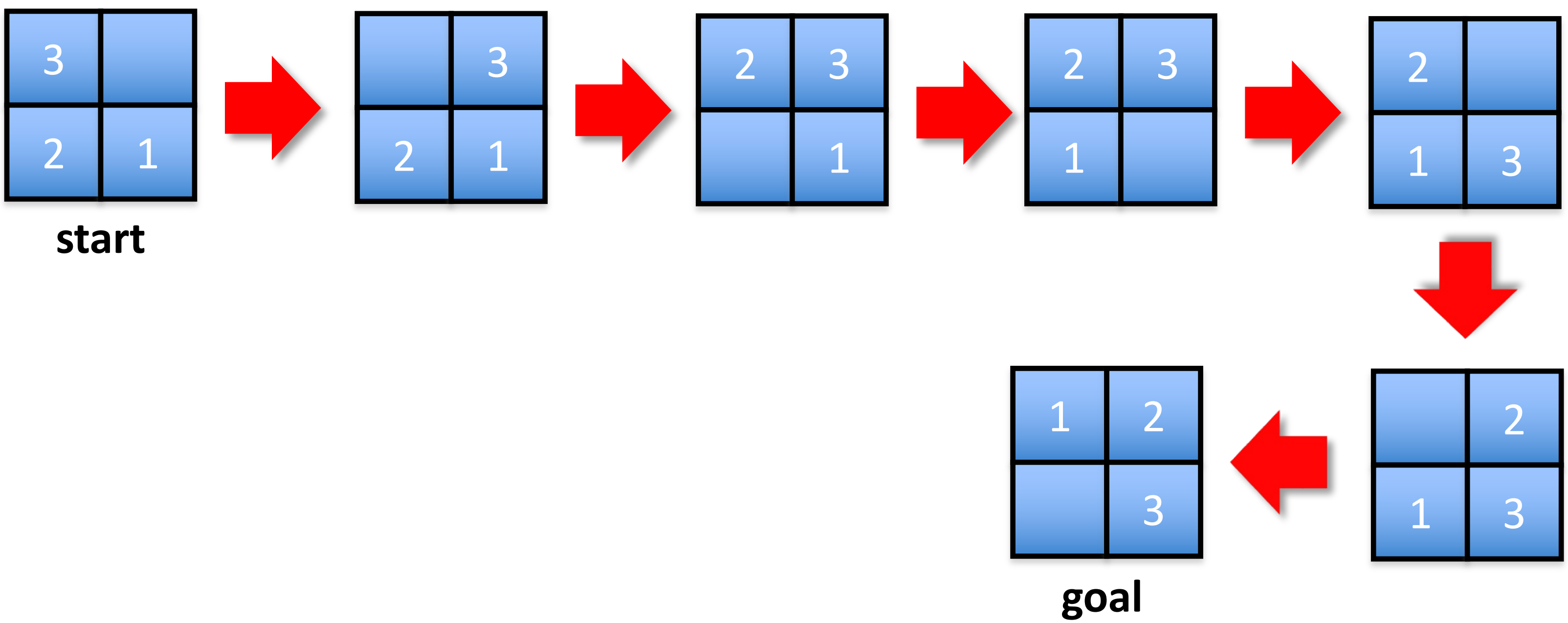


Start State

Goal State

Simpler: 3-Puzzle





Building goal-based agents

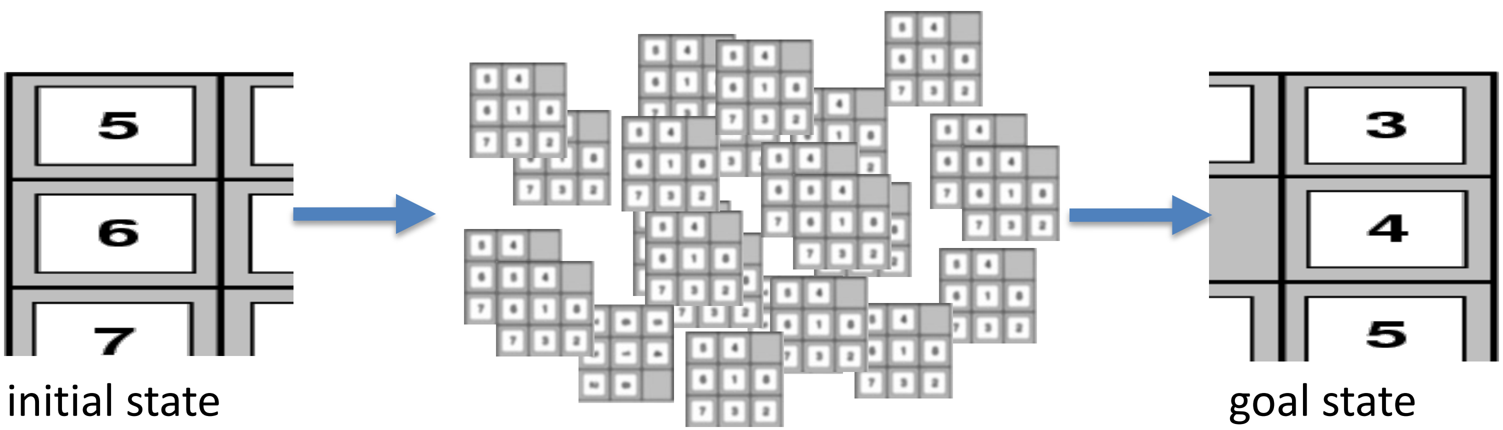
We must answer the following questions

– How do we represent the **state** of the “world”?

– What is the **goal** and how can we recognize it?

– What are the possible **actions**?

– What *relevant* information do we encoded to describe states, actions and their effects and thereby solve the problem?



Characteristics of 8-puzzle ?

	Fully observable?	Deterministic?	Episodic?	Static?	Discrete?	Single agent?
8-puzzle						

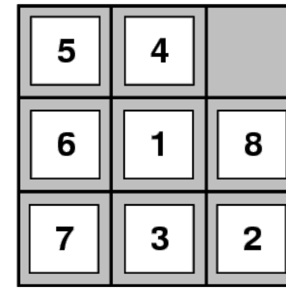
Characteristics of 8-puzzle

	Fully observable?	Deterministic?	Episodic?	Static?	Discrete?	Single agent?
8-puzzle	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**

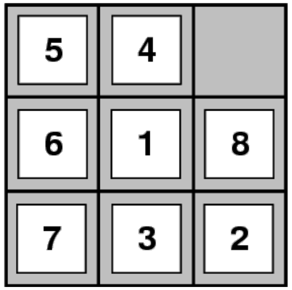
- All the Yes's mean it may be relatively easy!
- This is typical of the problems worked on in the 60s and 70s
- And the algorithms for solving them a state-space search model

Representing states / State-space

- State of an 8-puzzle?



Representing states / State-space



- State of an 8-puzzle?
  - A 3x3 array of integer in {0..8}
  - No integer appears twice
  - 0 represents the empty space

- In Python, we might implement this using a nine-character string: "`540681732`"
- And write functions to map the 2D coordinates to an index

What’s the goal to be achieved?

- Describe situation we want to achieve, a set of properties that we want to hold, etc.
- Defining a **goal test** function that when applied to a state returns True or False
- For our problem:

```
def isGoal(state):
    return state == “123405678”
```

What are the actions?

- **Primitive actions** for changing the state

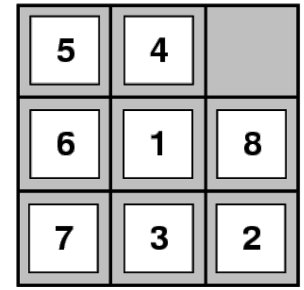
  In a **deterministic** world: no uncertainty in an action's effects (simple model)

- Given action and description of **current world state**, action completely specifies
  - Whether action ***can*** be applied to the current world (i.e., is it applicable and legal?) and
  - What state ***results*** after action is performed in the current world (i.e., no need for *history* information to compute the next state)

Representing actions

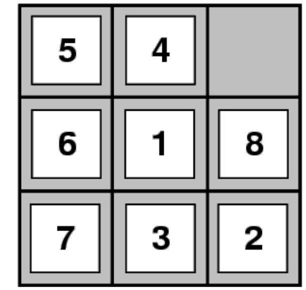
- Actions ideally considered as **discrete events** that occur at an **instant of time**
- Example, in a planning context
  - If state:inClass and perform action:goHome, then next state is state:atHome
  - There's no time where you're neither in class nor at home (i.e., in the state of "going home")

Representing actions



- Actions for 8-puzzle?

Representing actions



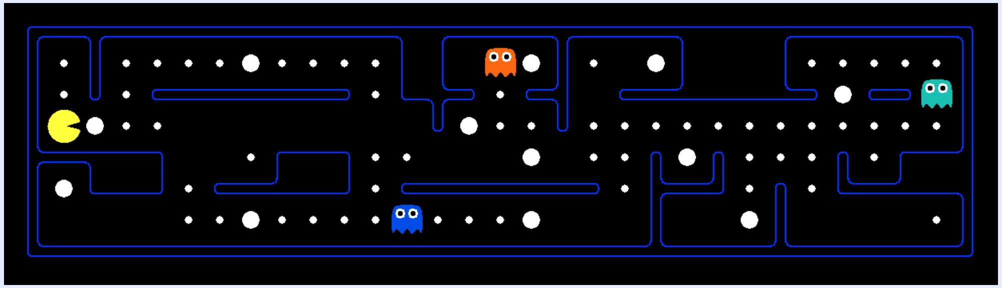
- Actions for 8-puzzle?

- Number of actions/operators depends on the **representation** used in describing a state
  - Specify 4 possible moves for each of the 8 tiles, resulting in a total of **4*8=32 operators**
  - Or: Specify four moves for "blank" square and we only need **4 operators**

- **Representational shift can simplify a problem!**

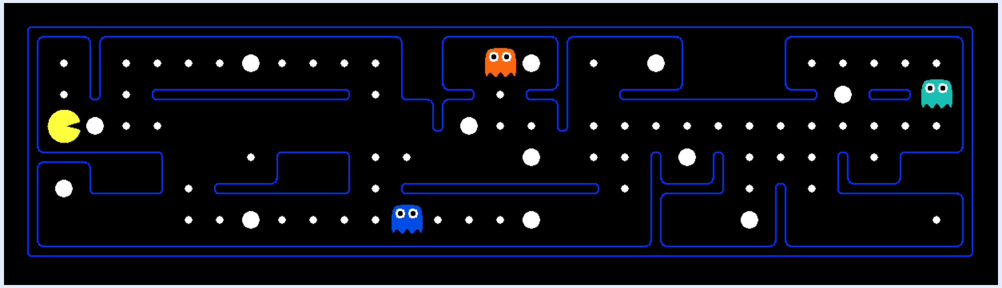
What’s in a State Space?

The world state includes every last detail of the environment



What's in a State Space?

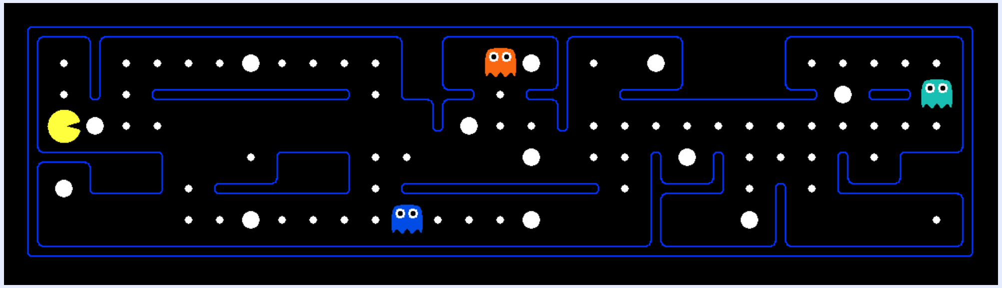
The world state includes every last detail of the environment



A search state keeps only the details needed for planning (abstraction)

What's in a State Space?

The world state includes every last detail of the environment

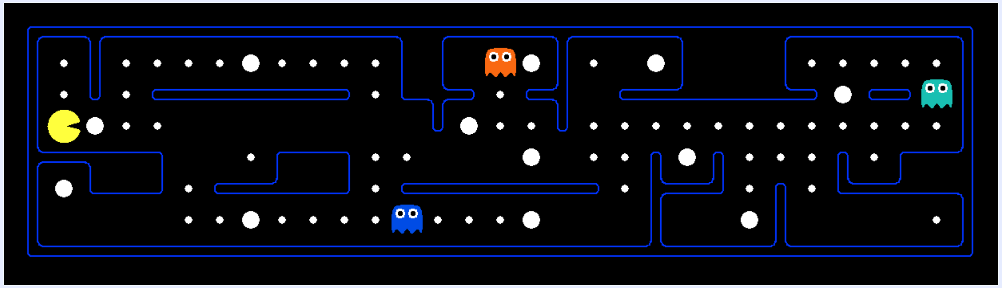


A search state keeps only the details needed for planning (abstraction)

- Problem: Pathing

What's in a State Space?

The world state includes every last detail of the environment

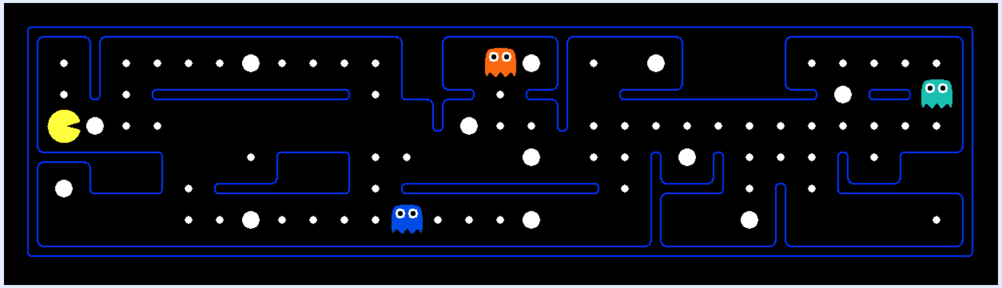


A search state keeps only the details needed for planning (abstraction)

- Problem: Pathing
  - States: (x,y) location

What's in a State Space?

The world state includes every last detail of the environment

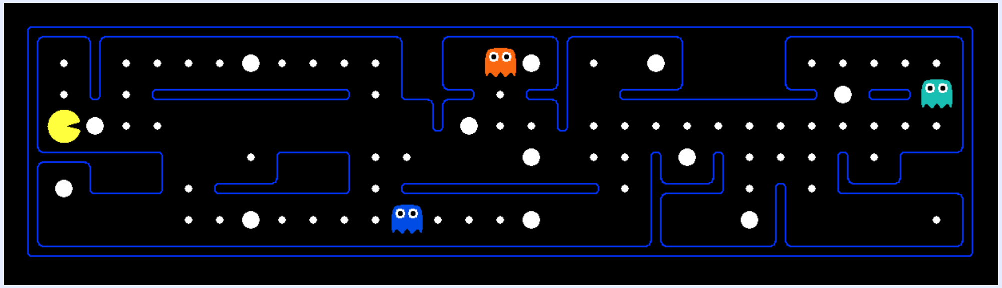


A search state keeps only the details needed for planning (abstraction)

- Problem: Pathing
  - States: (x,y) location
  - Actions: NSEW

What's in a State Space?

The world state includes every last detail of the environment

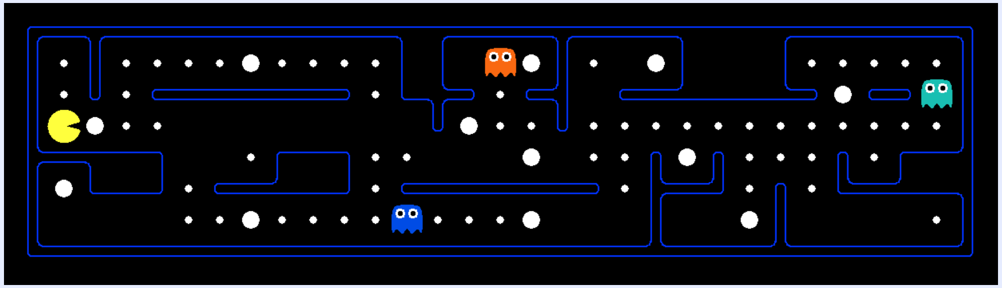


A search state keeps only the details needed for planning (abstraction)

- Problem: Pathing
  - States: (x,y) location
  - Actions: NSEW
  - Successor: update location only

What’s in a State Space?

The world state includes every last detail of the environment

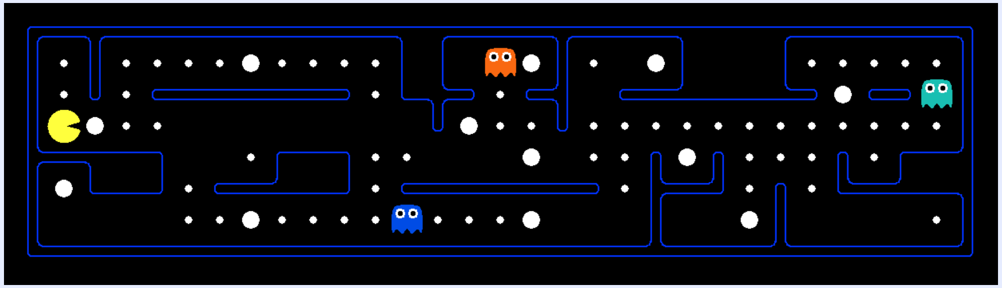


A search state keeps only the details needed for planning (abstraction)

- Problem: Pathing
  - States: (x,y) location
  - Actions: NSEW
  - Successor: update location only
  - Goal test: is (x,y)=END

What's in a State Space?

The world state includes every last detail of the environment

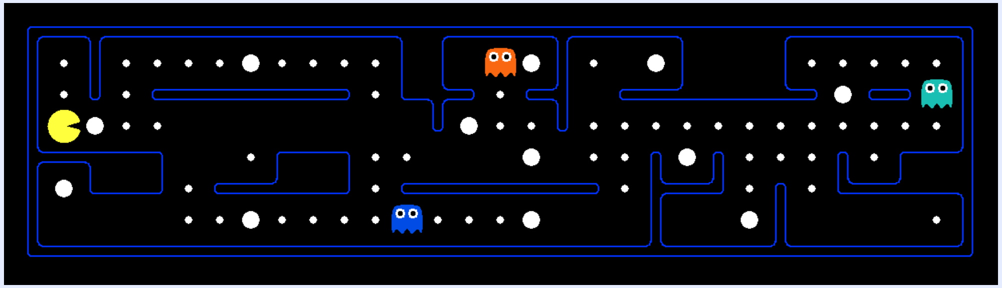


A search state keeps only the details needed for planning (abstraction)

- Problem: Pathing
  - States: (x,y) location
  - Actions: NSEW
  - Successor: update location only
  - Goal test: is (x,y)=END
- Problem: Eat-All-Dots

What's in a State Space?

The world state includes every last detail of the environment

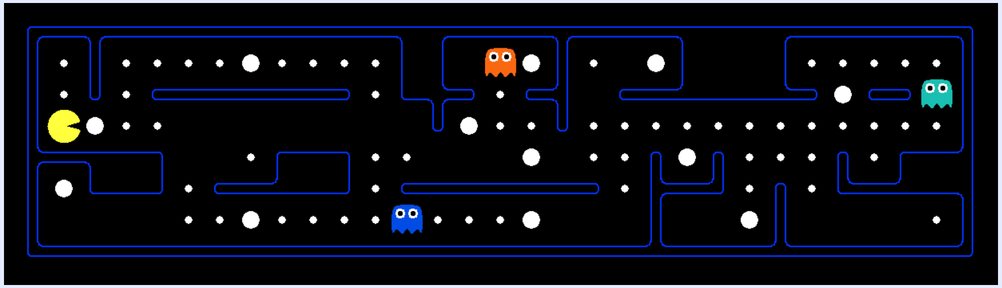


A search state keeps only the details needed for planning (abstraction)

- Problem: Pathing
  - States: (x,y) location
  - Actions: NSEW
  - Successor: update location only
  - Goal test: is (x,y)=END
- Problem: Eat-All-Dots
  - States: {(x,y), dot booleans}

What’s in a State Space?

The world state includes every last detail of the environment

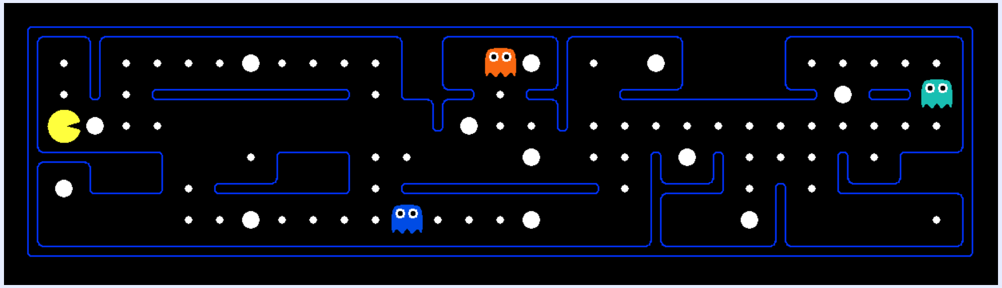


A search state keeps only the details needed for planning (abstraction)

- Problem: Pathing
  - States: (x,y) location
  - Actions: NSEW
  - Successor: update location only
  - Goal test: is (x,y)=END
- Problem: Eat-All-Dots
  - States: {(x,y), dot booleans}
  - Actions: NSEW

What's in a State Space?

The world state includes every last detail of the environment

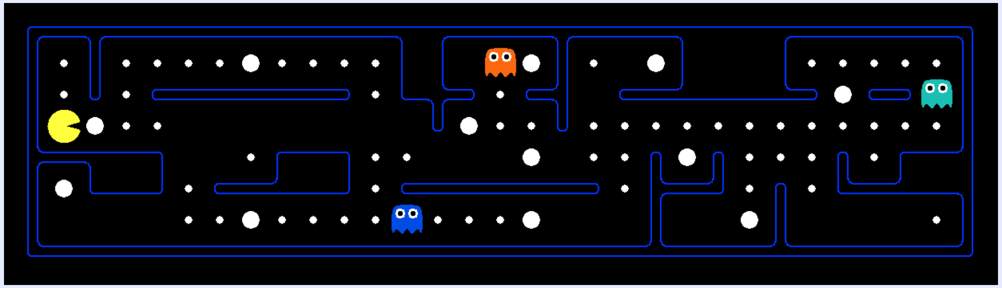


A search state keeps only the details needed for planning (abstraction)

- Problem: Pathing
  - States: (x,y) location
  - Actions: NSEW
  - Successor: update location only
  - Goal test: is (x,y)=END
- Problem: Eat-All-Dots
  - States: {(x,y), dot booleans}
  - Actions: NSEW
  - Successor: update location and possibly a dot boolean

What's in a State Space?

The world state includes every last detail of the environment

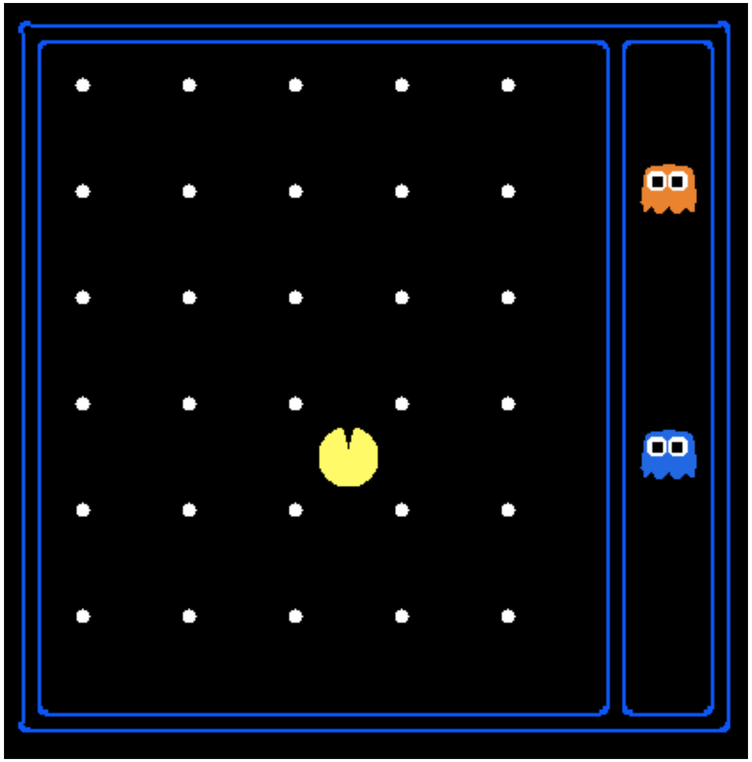


A search state keeps only the details needed for planning (abstraction)

- Problem: Pathing
  - States: (x,y) location
  - Actions: NSEW
  - Successor: update location only
  - Goal test: is (x,y)=END
- Problem: Eat-All-Dots
  - States: {(x,y), dot booleans}
  - Actions: NSEW
  - Successor: update location and possibly a dot boolean
  - Goal test: dots all false

State Space Sizes?

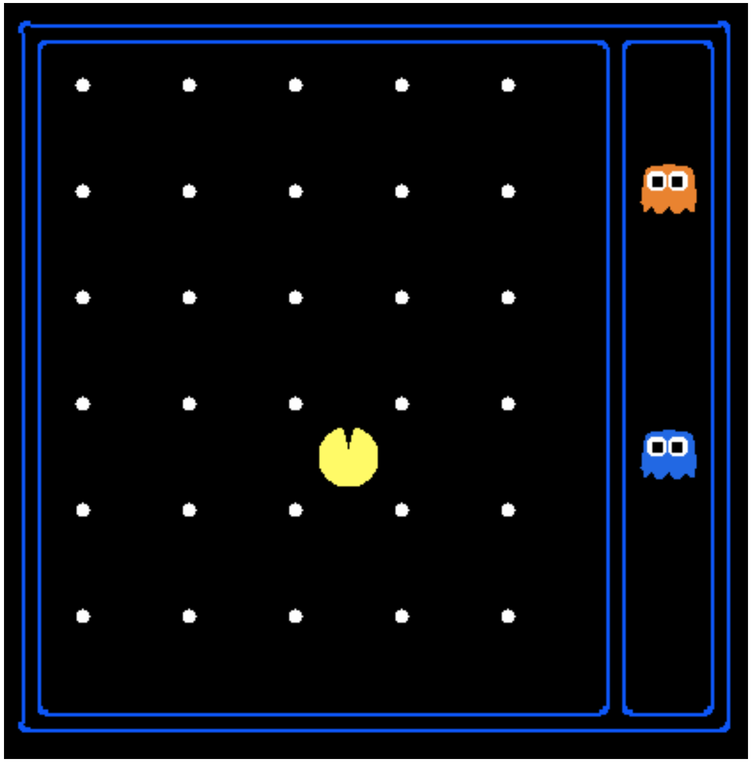
- World state:
  - Agent positions: 120
  - Food count: 30
  - Ghost positions: 12
  - Agent facing: NSEW



State Space Sizes?

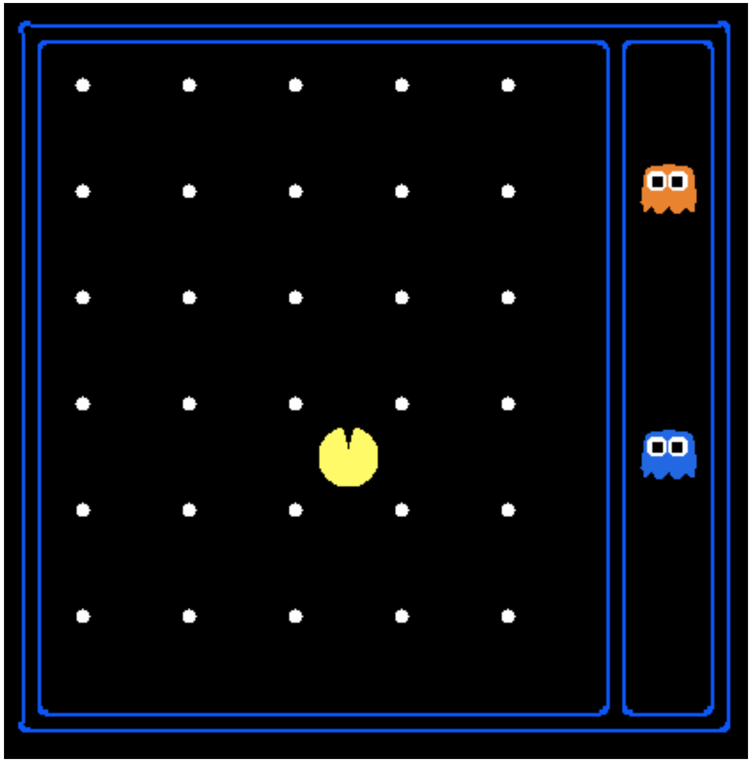
- World state:
  - Agent positions: 120
  - Food count: 30
  - Ghost positions: 12
  - Agent facing: NSEW

- How many



State Space Sizes?

- World state:
  - Agent positions: 120
  - Food count: 30
  - Ghost positions: 12
  - Agent facing: NSEW
- How many
  - World states?

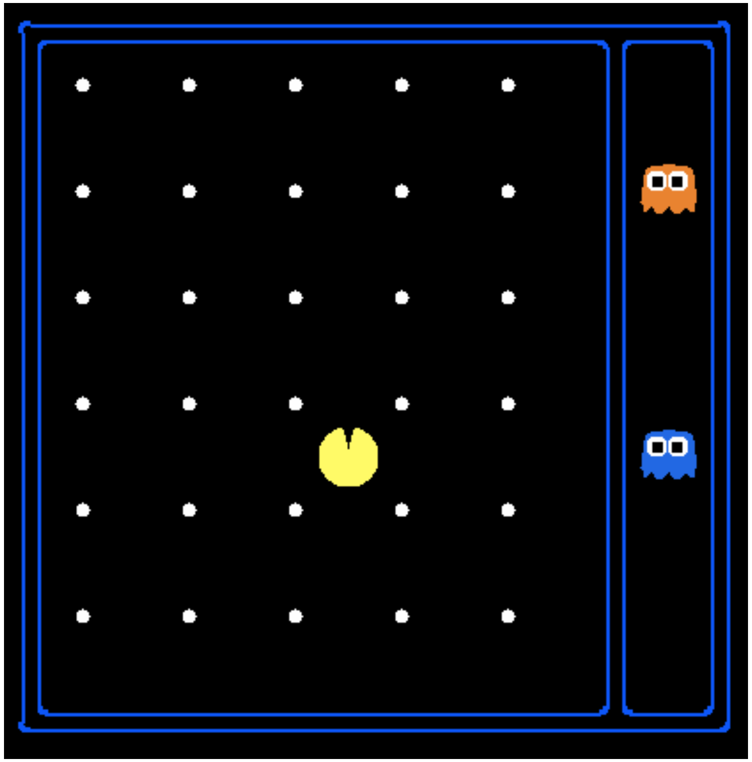


State Space Sizes?

- World state:
  - Agent positions: 120
  - Food count: 30
  - Ghost positions: 12
  - Agent facing: NSEW

- How many
  - World states?

    $120x(2^{30})x(12^{2})x4$

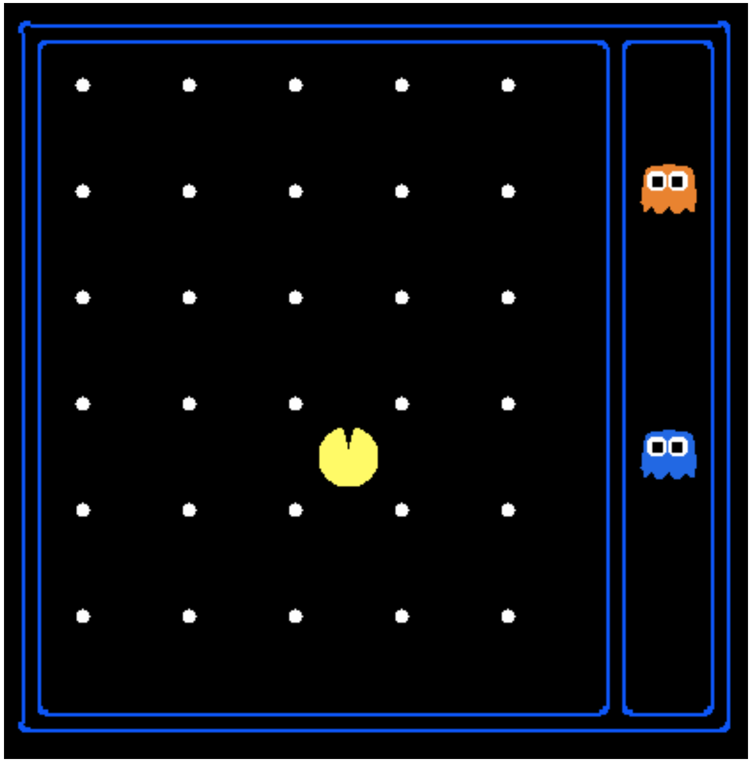


State Space Sizes?

- World state:
  - Agent positions: 120
  - Food count: 30
  - Ghost positions: 12
  - Agent facing: NSEW

- How many
  - World states?

    $120 \times (2^{30}) \times (12^2) \times 4$
  - States for pathing?

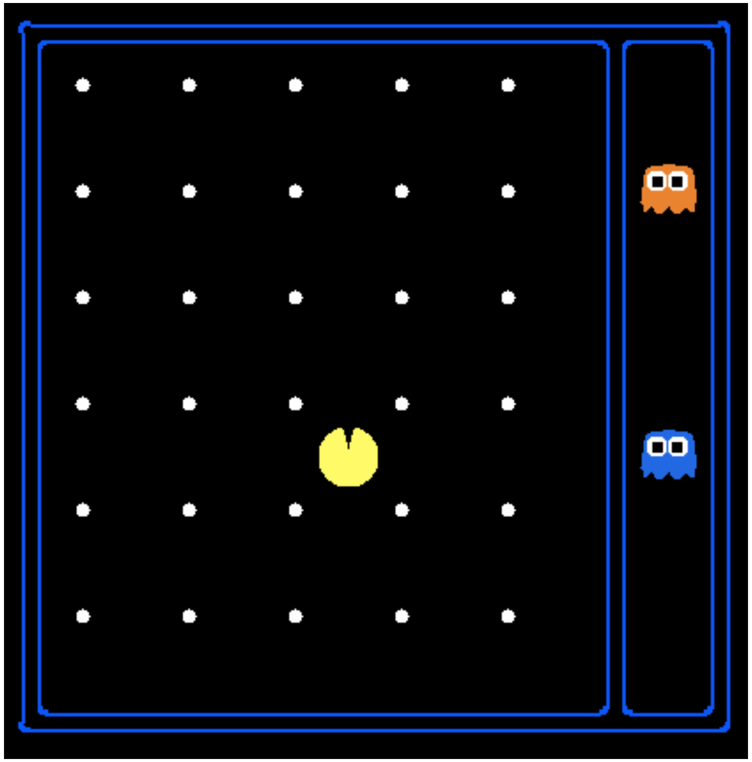


State Space Sizes?

- World state:
  - Agent positions: 120
  - Food count: 30
  - Ghost positions: 12
  - Agent facing: NSEW
- How many
  - World states?

    $120 \times (2^{30}) \times (12^2) \times 4$
  - States for pathing?

    120

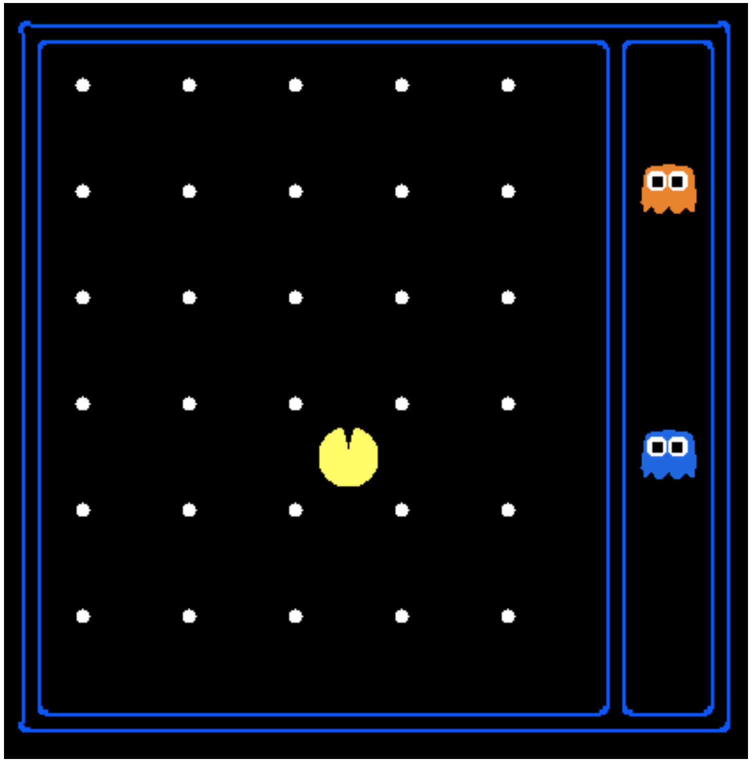


State Space Sizes?

- World state:
  - Agent positions: 120
  - Food count: 30
  - Ghost positions: 12
  - Agent facing: NSEW
- How many
  - World states?

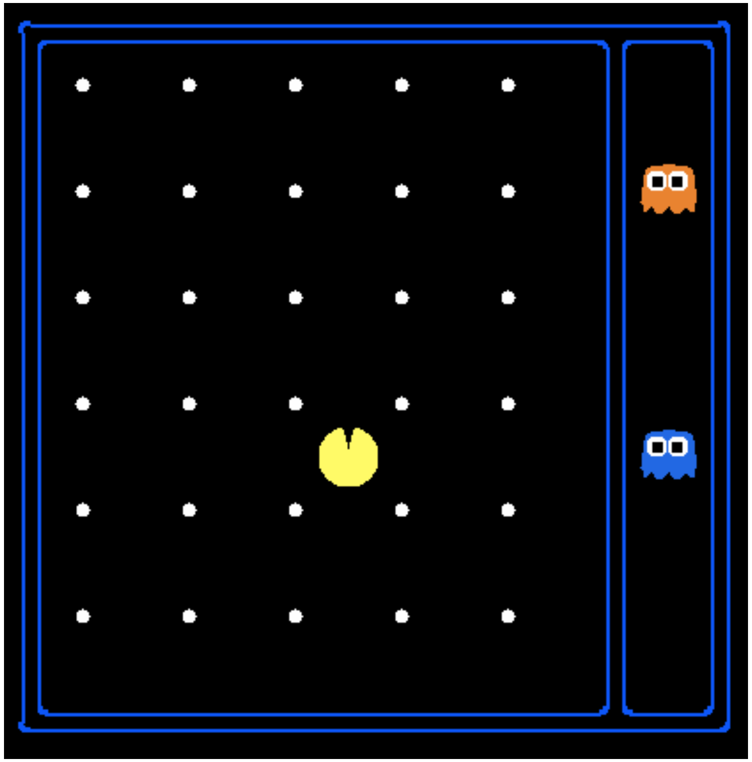
    $120 \times (2^{30}) \times (12^2) \times 4$
  - States for pathing?

    120
  - States for eat-all-dots?



State Space Sizes?

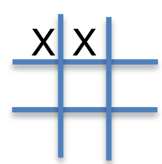
- World state:
  - Agent positions: 120
  - Food count: 30
  - Ghost positions: 12
  - Agent facing: NSEW
- How many
  - World states?
    $120x(2^{30})x(12^{2})x4$
  - States for pathing?
    120
  - States for eat-all-dots?
    $120x(2^{30})$

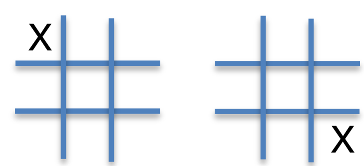


State Space Sizes?

- **Size of a problem** usually described in terms of possible **number of states**
  - Tic-Tac-Toe has about 3^9 states (19,683≈2*10^4)
  - Checkers has about 10^{40} states
  - Rubik's Cube has about 10^{19} states
  - Chess has about 10^{120} states in a typical game
  - Go has 2*10^{170}
  - Theorem provers may deal with an infinite space
- State space size ≈ solution difficulty

State Space Sizes?

- **Our estimates were loose upper bounds**
- How many **possible, legal** states does tic-tac-toe really have?
- Simple upper bound: nine board cells, each of which can be empty, O or X, so 3^9
- Only 593 states after eliminating
  - impossible states 

  - Rotations and reflections 

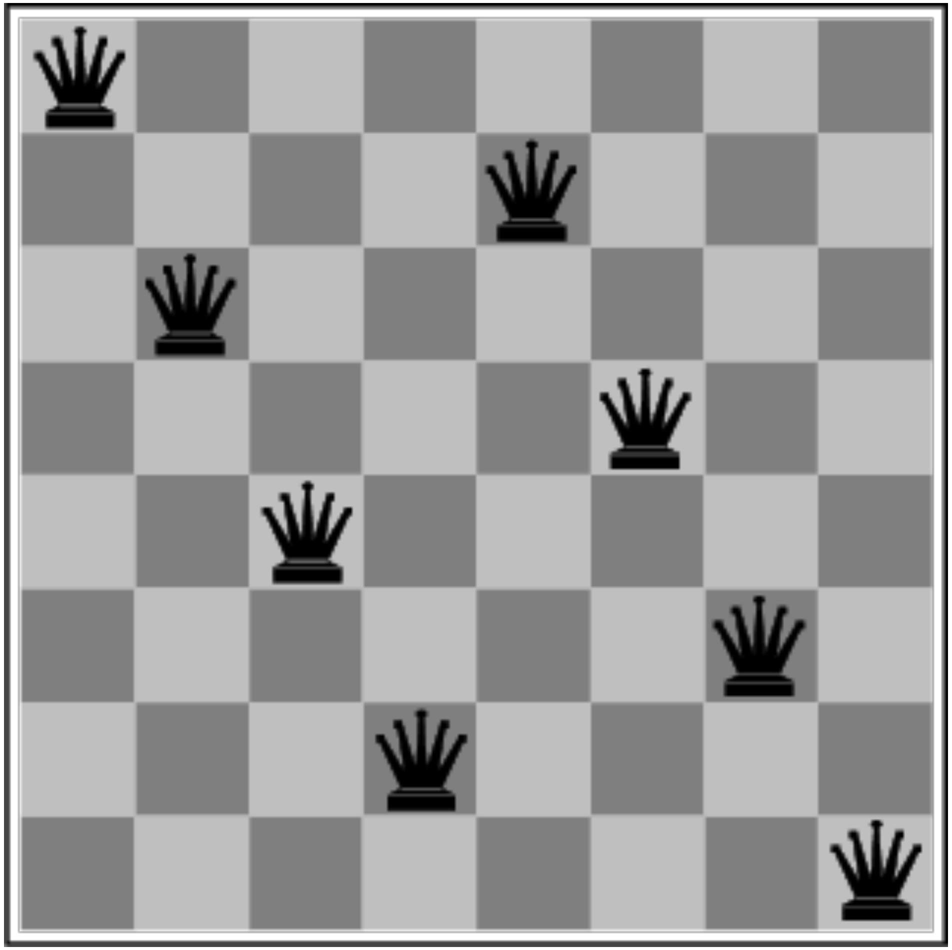
Some example problems

- Toy problems and micro-worlds
  - 8-Puzzle
  - Missionaries and Cannibals
  - Cryptarithmetic
  - 8-Queens Puzzle
  - Remove 5 Sticks
  - Water Jug Problem
- Real-world problems

Example: The 8-Queens Puzzle

Place eight queens on a chessboard such that no queen attacks any other

We can generalize the problem to a NxN chessboard



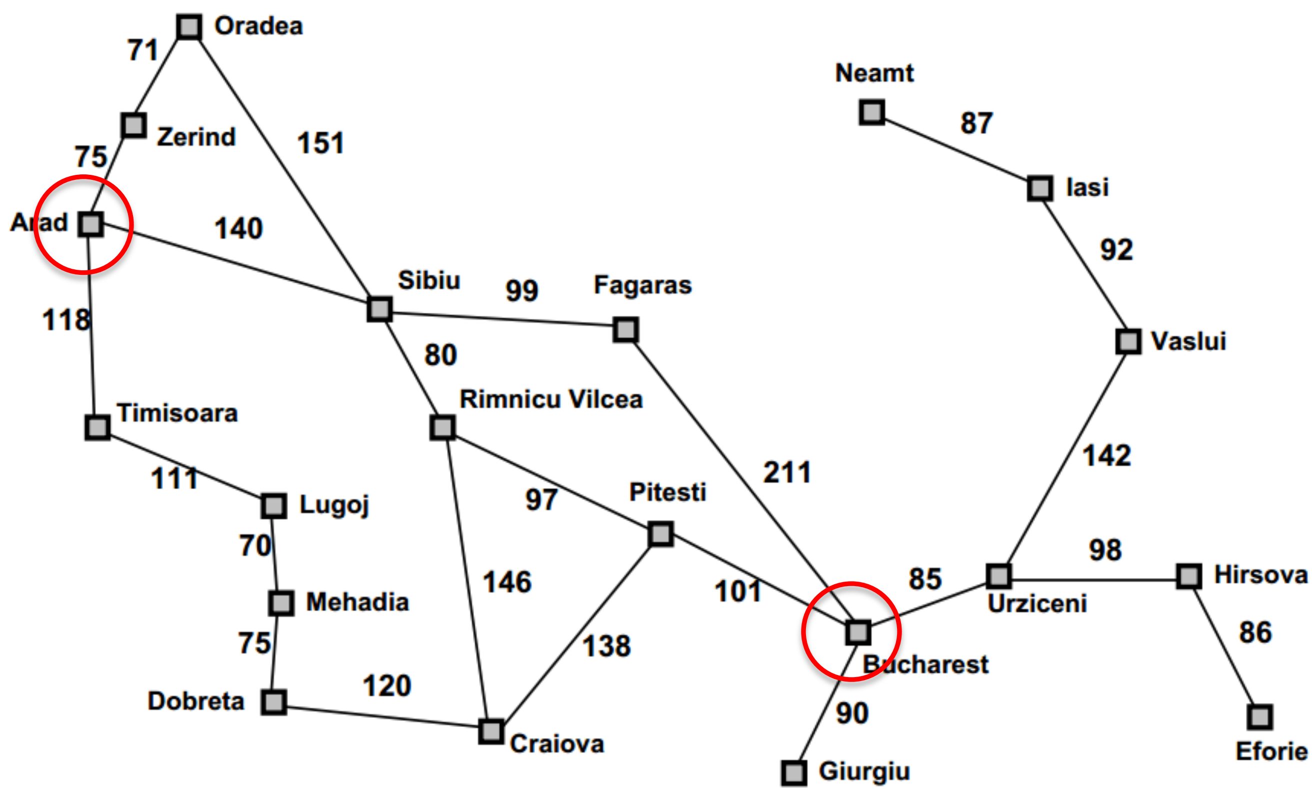
What are the states, goal test, actions?

Some more real-world problems

- Route finding
- Touring (traveling salesman)
- Logistics
- VLSI layout
- Robot navigation
- Theorem proving
- Learning

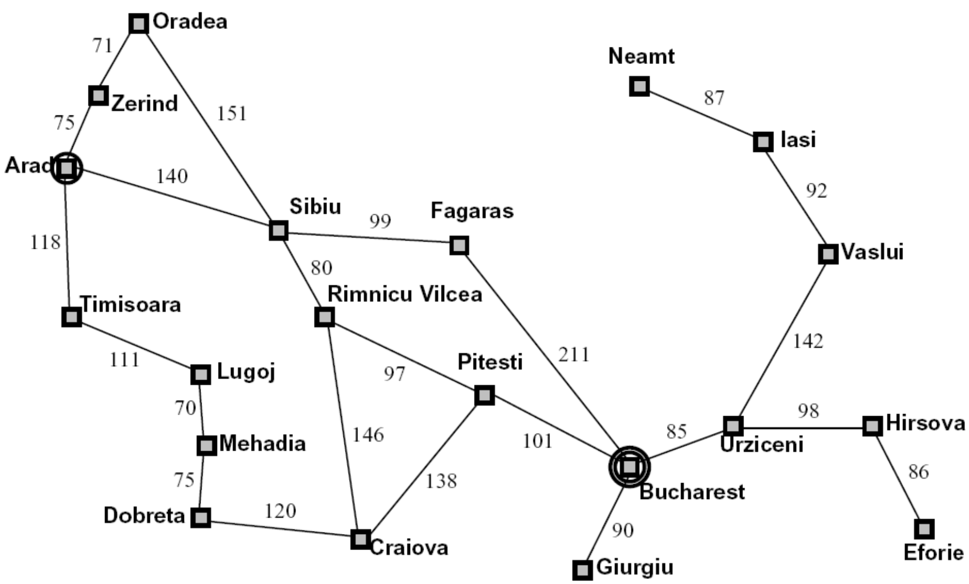
Route Planning

Find a route from Arad to Bucharest



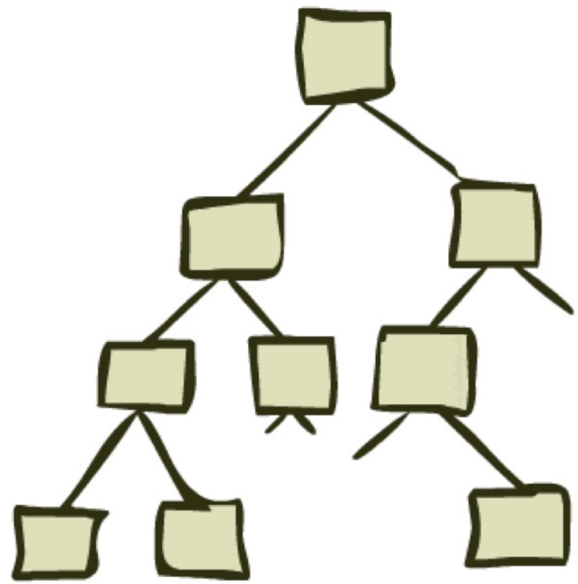
A simplified map of major roads in Romania used in our text

Example: Traveling in Romania



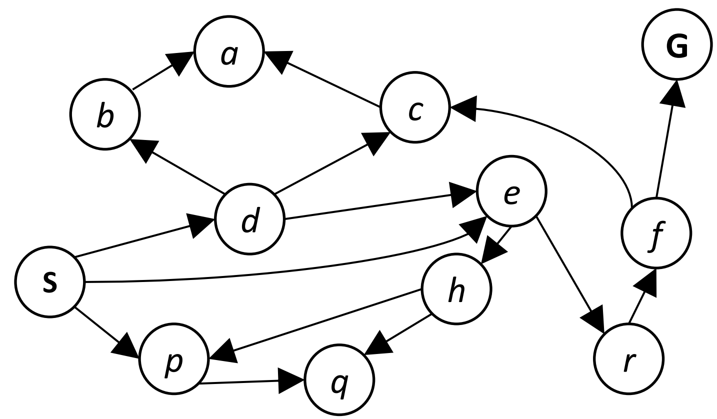
- State space:
  - Cities
- Successor function:
  - Roads: Go to adjacent city with cost = distance
- Start state:
  - Arad
- Goal test:
  - Is state == Bucharest?
- Solution?

State Space Graphs and Search Trees



State Space Graphs

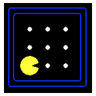
- State space graph: A mathematical representation of a search problem
  - Nodes are (abstracted) world configurations
  - Arcs represent transitions/ successors (action results)
  - The goal test is a set of goal nodes (maybe only one)
- In a state space graph, each state occurs only once!
- We can rarely build this full graph in memory (it's too big), but it's a useful idea



Tiny state space graph for a tiny search problem

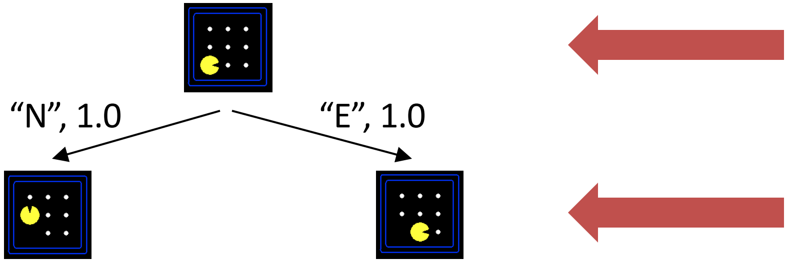
Search Trees

Search Trees



This is now / start

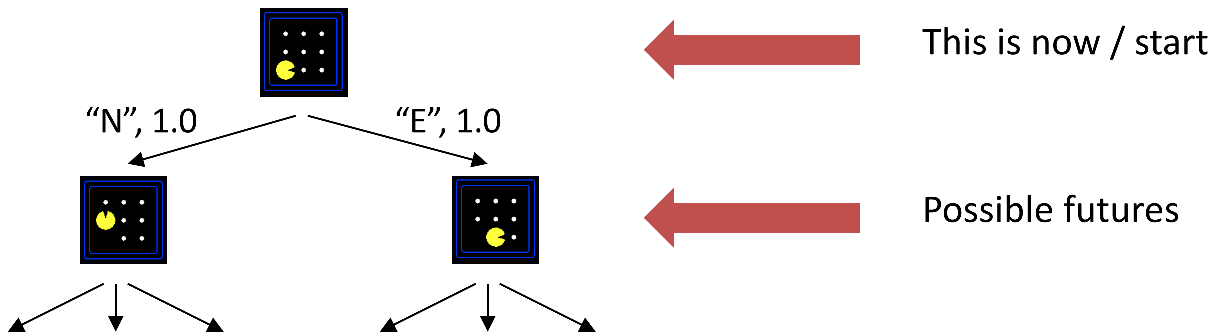
Search Trees



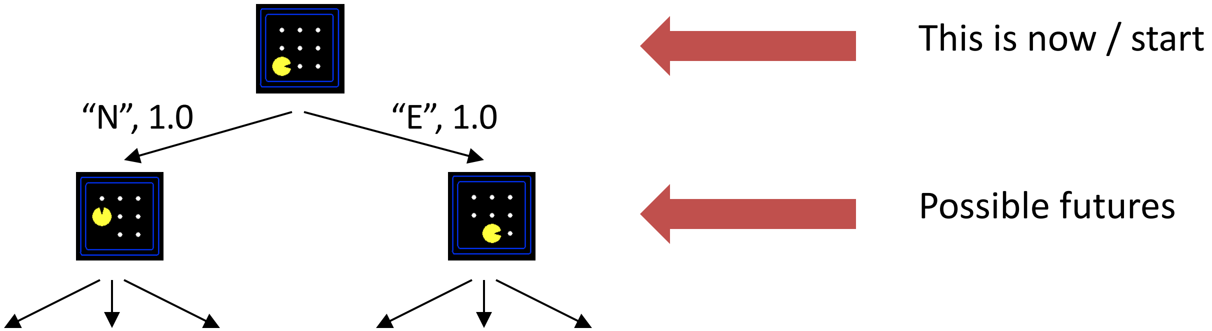
This is now / start

Possible futures

Search Trees



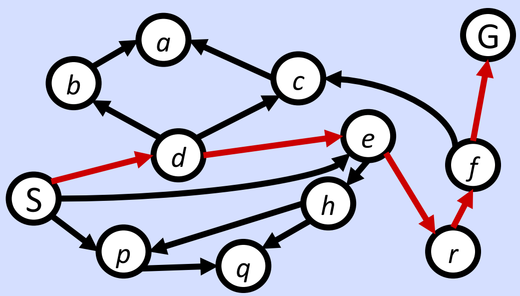
Search Trees



- A search tree:
  - A “what if” tree of plans and their outcomes
  - The start state is the root node
  - Children correspond to successors
  - Nodes show states, but correspond to PLANS that achieve those states
  - For most problems, we can never actually build the whole tree

State Space Graphs vs. Search Trees

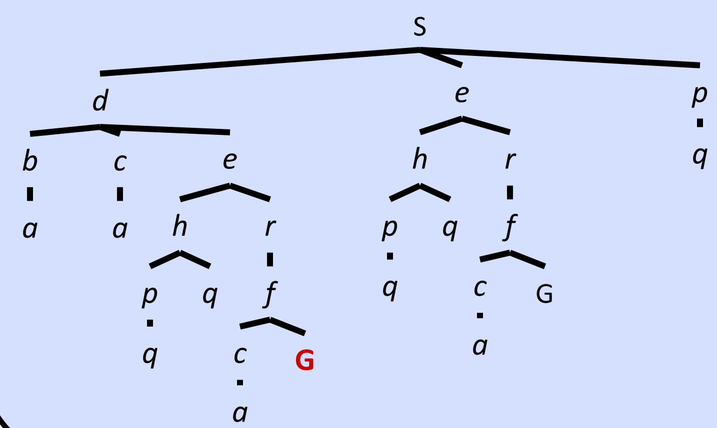
State Space Graph



Each NODE in in the search tree is an entire PATH in the state space graph.

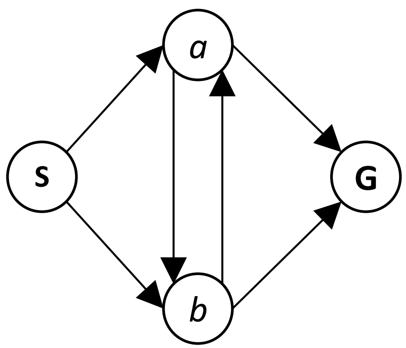
We construct the tree on demand – and we construct as little as possible.

Search Tree



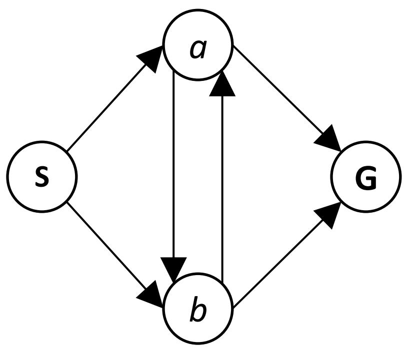
Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:



Quiz: State Space Graphs vs. Search Trees

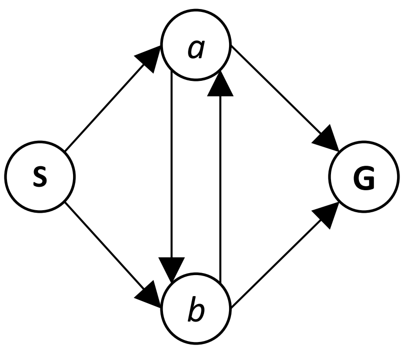
Consider this 4-state graph:



How big is its search tree (from S)?

Quiz: State Space Graphs vs. Search Trees

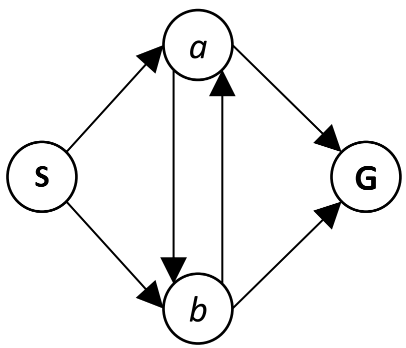
Consider this 4-state graph:



How big is its search tree (from S)?

Quiz: State Space Graphs vs. Search Trees

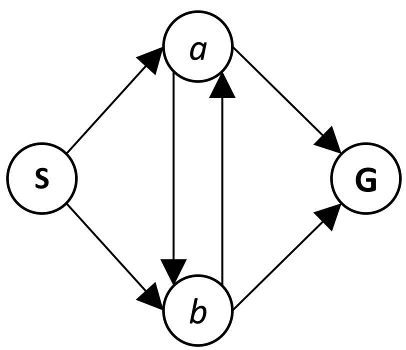
Consider this 4-state graph:



How big is its search tree (from S)?

Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:



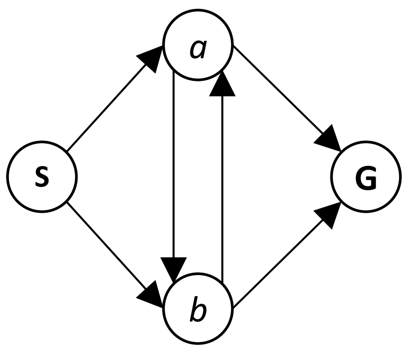
How big is its search tree (from S)?

S

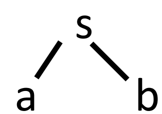
∞

Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:

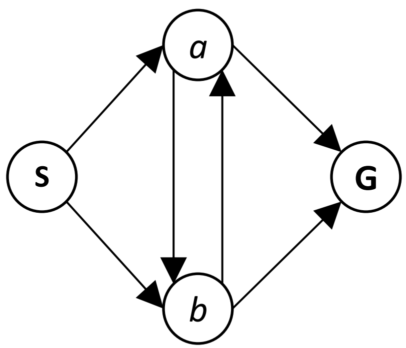


How big is its search tree (from S)?

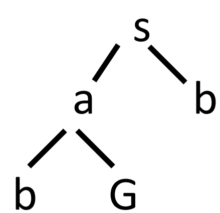


Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:



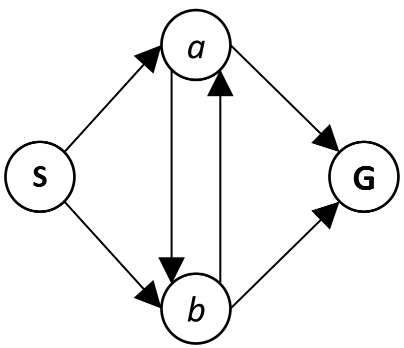
How big is its search tree (from S)?



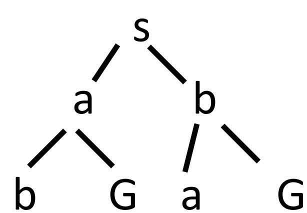
∞

Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:

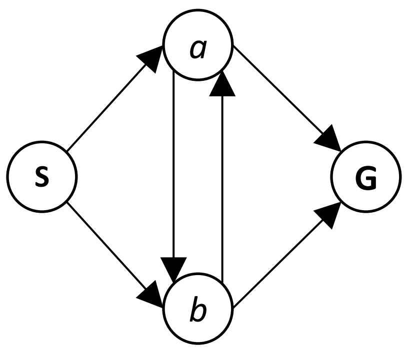


How big is its search tree (from S)?

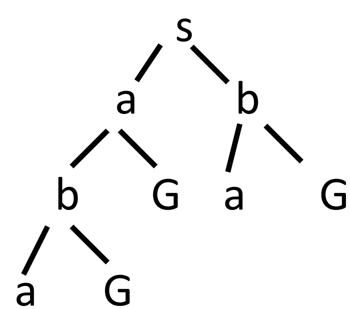


Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:

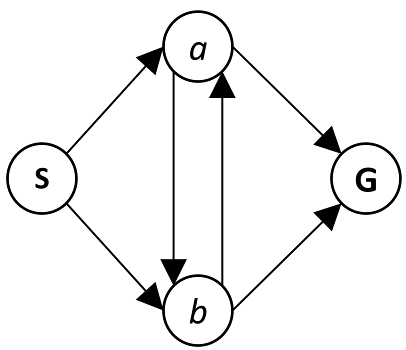


How big is its search tree (from S)?

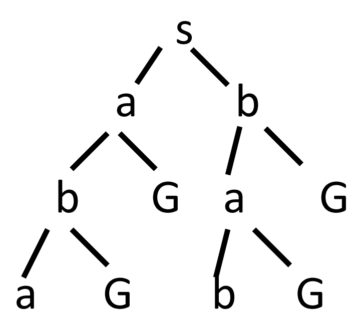


Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:

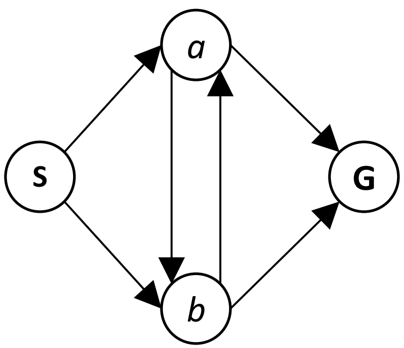


How big is its search tree (from S)?

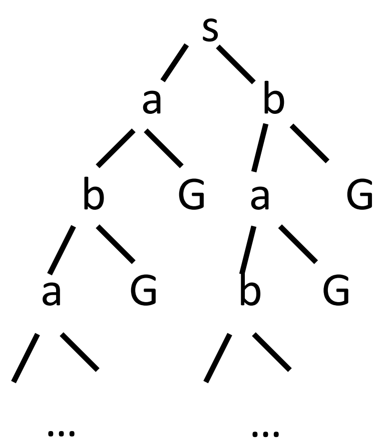


Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:



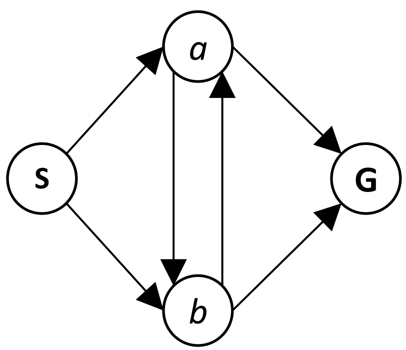
How big is its search tree (from S)?



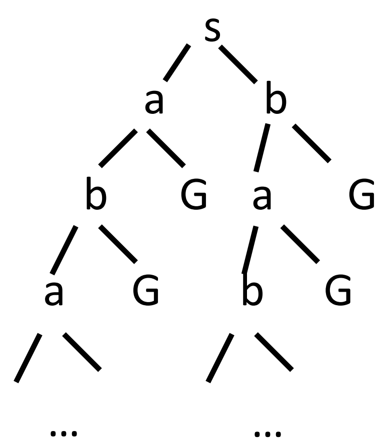
∞

Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:



How big is its search tree (from S)?



Important: Those who don’t know history are doomed to repeat it!

Formalizing search

- **Solution:** sequence of actions associated with a path from a start node to a goal node
- **Solution cost:** sum of the arc costs on the solution path
  - If all arcs have same (unit) cost, then solution cost is length of solution (number of steps)
  - Algorithms generally require that arc costs cannot be negative (why?)

A General Searching Algorithm

Core ideas:

1. Maintain a list of frontier (fringe) nodes
   1. Nodes coming *into* the frontier have been explored
   2. Nodes going out of the frontier have not been explored
2. Iteratively select nodes from the frontier and explore unexplored nodes from the frontier
3. Stop when you reach your **goal**

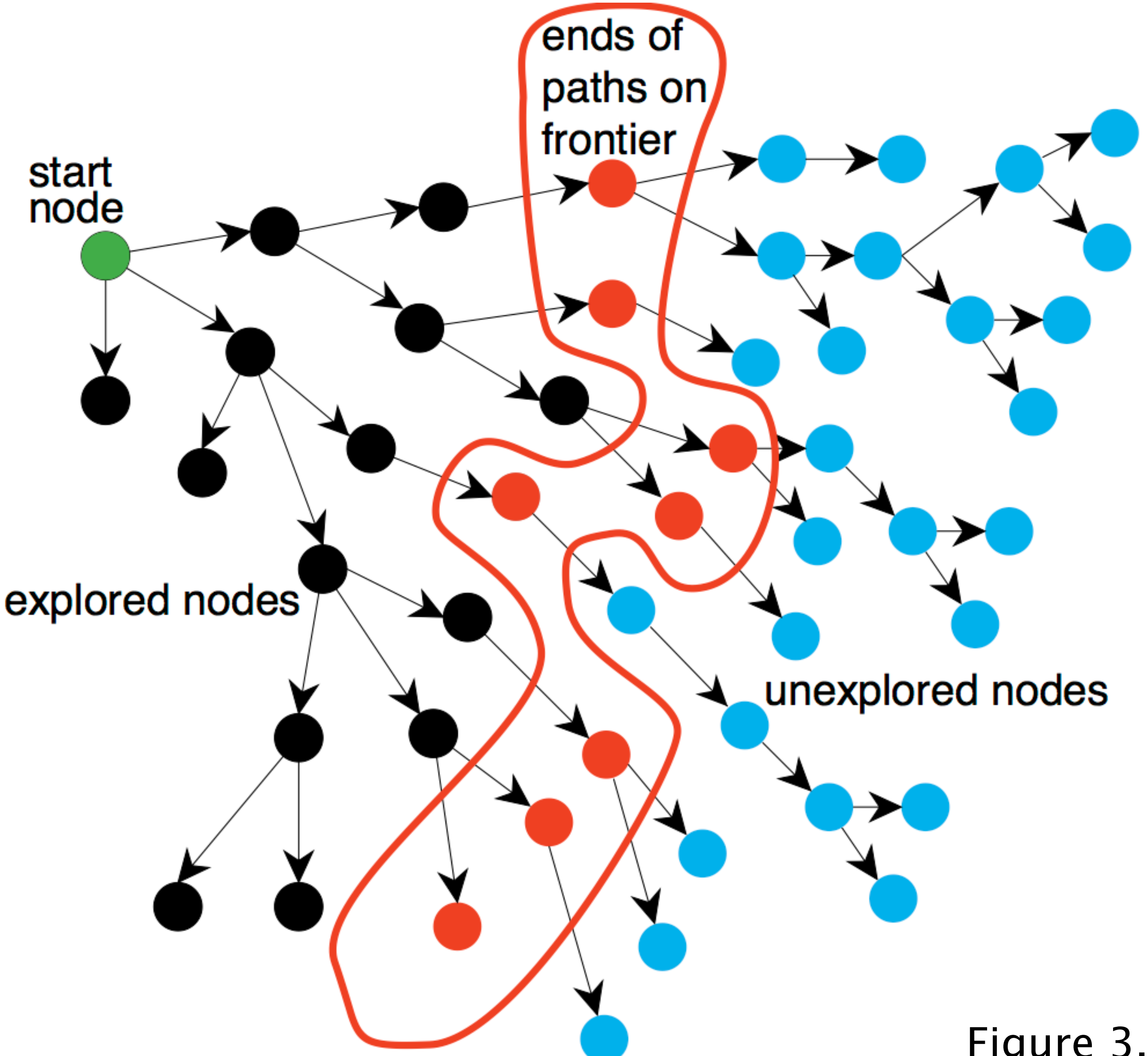


Figure 3.3

State-space search algorithm

;; problem describes the start state, operators, goal test, and operator costs

;; queueing-function is a comparator function that ranks two states

;; general-search returns either a goal node or failure

```
function general-search (problem, QUEUEING-FUNCTION)
  nodes = MAKE-QUEUE(MAKE-NODE(problem.INITIAL-STATE))
  loop
     if EMPTY(nodes) then return "failure"
     node = REMOVE-FRONT(nodes)
     if problem.GOAL-TEST(node.STATE) succeeds
        then return node
     nodes = QUEUEING-FUNCTION(nodes, EXPAND(node,
             problem.OPERATORS))
 end
```

;; Note: The goal test is NOT done when nodes are generated

;; Note: This algorithm does not detect loops

Key procedures to be defined

- EXPAND
  - Generate a node's successor nodes, adding them to the graph if not already there
- GOAL-TEST
  - Test if state satisfies all goal conditions
- QUEUEING-FUNCTION
  - Maintain ranked list of nodes that are candidates for expansion
  - Changing definition of the QUEUEING-FUNCTION leads to different search strategies

Informed vs. uninformed search

Uninformed search strategies (blind search)

- Use no information about likely *direction* of a goal
- Methods: breadth-first, depth-first, depth-limited, uniform-cost, depth-first iterative deepening, bidirectional

Informed search strategies (heuristic search)

- Use information about domain to (try to) (usually) head in the general direction of goal node(s)
- Methods: hill climbing, best-first, greedy search, beam search, algorithm A, algorithm A*

Evaluating search strategies

- **Completeness**
  - Guarantees finding a solution whenever one exists
- **Time complexity** (worst or average case)
- **Space complexity**
- **Optimality/Admissibility**

Evaluating search strategies

- **Completeness**
  - Guarantees finding a solution whenever one exists
- **Time complexity** (worst or average case)
  - Usually measured by *number of nodes expanded*
- **Space complexity**
- **Optimality/Admissibility**

Evaluating search strategies

- **Completeness**
  - Guarantees finding a solution whenever one exists
- **Time complexity** (worst or average case)
  - Usually measured by *number of nodes expanded*
- **Space complexity**
  - Usually measured by maximum size of graph/tree during the search
- **Optimality/Admissibility**

Evaluating search strategies

- **Completeness**
  - Guarantees finding a solution whenever one exists
- **Time complexity** (worst or average case)
  - Usually measured by *number of nodes expanded*
- **Space complexity**
  - Usually measured by maximum size of graph/tree during the search
- **Optimality/Admissibility**
  - If a solution is found, is it **guaranteed** to be an optimal one, i.e., one with minimum cost

NEXT CLASS
UNINFORMED SEARCH

EXTRA HELPER SLIDES

PRIMER ON GRAPHS

Remember: Graphs

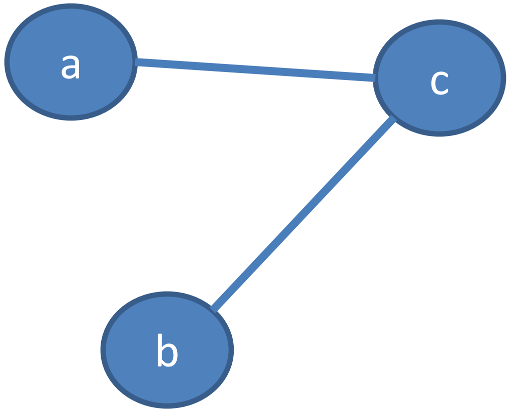
- A graph G = (E, V)
- V = set of vertices (nodes)
- E = set of edges between pairs of nodes, (x, y)

G can be:

- Undirected: order of (x, y) doesn't matter
  - These are symmetric
- Directed: order of (x, y) does matter
- Weighted: cost function $g(x, y)$
- (among other qualities)

Remember: Graphs

- A graph G = (E, V)
- V = set of vertices (nodes)
- E = set of edges between pairs of nodes

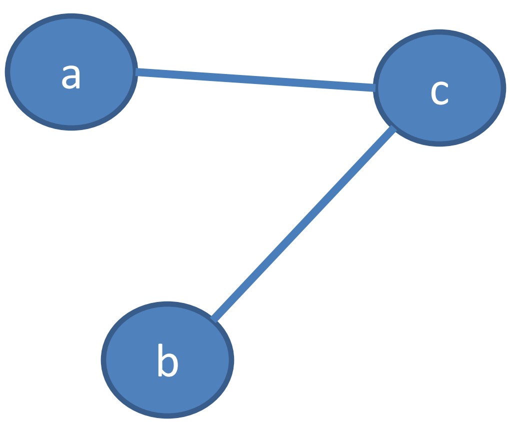


V= { ??? }

E = { ??? }

Remember: Graphs

- A graph G = (E, V)
- V = set of vertices (nodes)
- E = set of edges between pairs of nodes



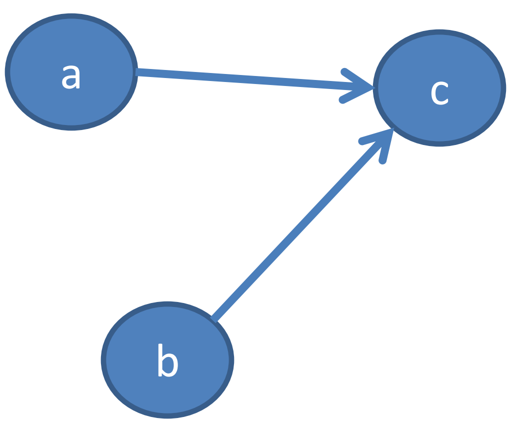
V= { a, b, c }

E = { (a, c), (b, c) }

undirected

Remember: Graphs

- A graph G = (E, V)
- V = set of vertices (nodes)
- E = set of edges between pairs of nodes

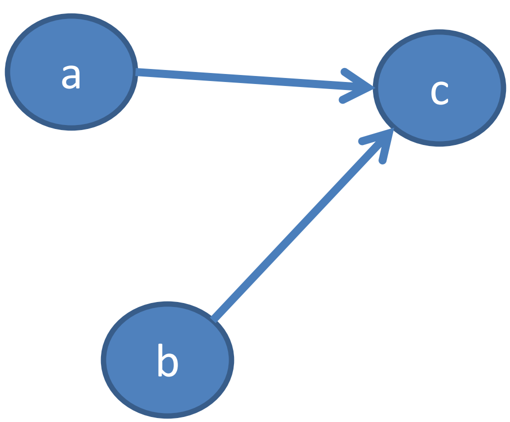


V= { ??? }

E = { ??? }

Remember: Graphs

- A graph G = (E, V)
- V = set of vertices (nodes)
- E = set of edges between pairs of nodes



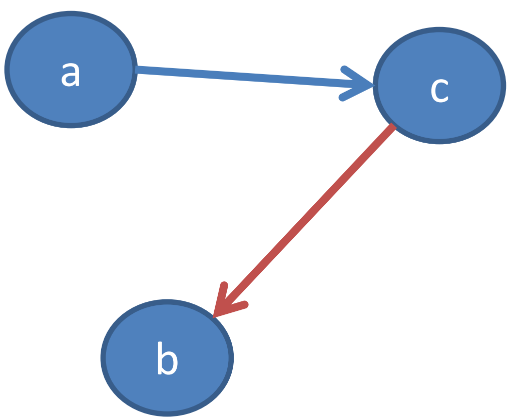
V= { a, b, c }

E = { (a, c), (b, c) }

directed

Remember: Graphs

- A graph G = (E, V)
- V = set of vertices (nodes)
- E = set of edges between pairs of nodes



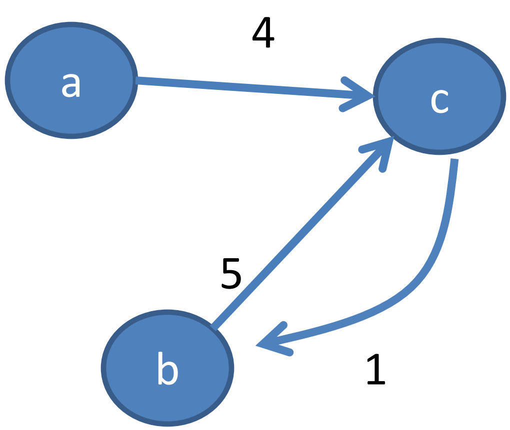
V= { a, b, c }

E = { (a, c), (c, b) }

directed

Remember: Graphs

- A graph G = (E, V)
- V = set of vertices (nodes)
- E = set of edges between pairs of nodes



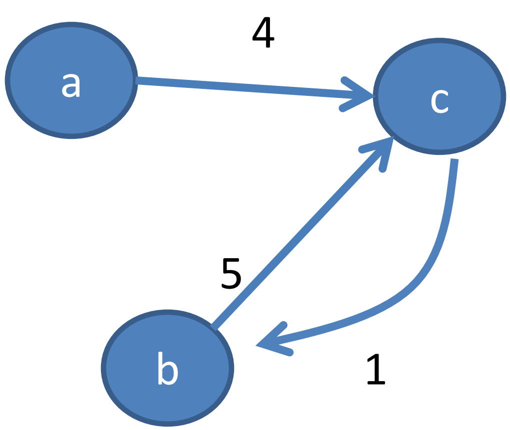
V= { ??? }

E = { ??? }

g = ???

Remember: Graphs

- A graph G = (E, V)
- V = set of vertices (nodes)
- E = set of edges between pairs of nodes



V= { a, b, c }

E = { (a,c), (b, c), (c, b) }

g = {(a, c): 4, (b, c): 5, (c, b): 1}

weighted, directed

MORE EXAMPLES

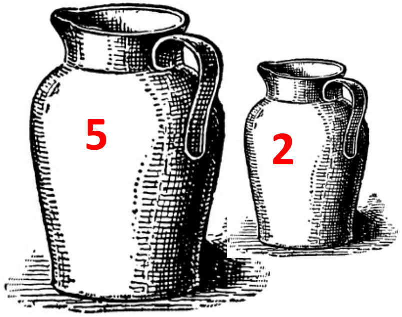
Water Jug Problem

- Two jugs J1 & J2 with capacity C1 & C2
- Initially J1 has W1 water and J2 has W2 water
  - e.g.: full 5 gallon jug and empty 2 gallon jug
- Possible actions:
  - Pour from jug X to jug Y until X empty or Y full
  - Empty jug X onto the floor
- Goal: J1 has G1 water and J2 G2
  - G1 or G2 can be -1 to represent any amount
- E.g.: initially full jugs with capacities 3 and 1 liters, goal is to have 1 liter in each

Example: Water Jug Problem

- Two jugs J1 and J2 with capacity C1 and C2
- Initially J1 has W1 water and J2 has W2 water
  - e.g.: a full 5-gallon jug and an empty 2-gallon jug
- Possible actions:
  - Pour from jug X to jug Y until X empty or Y full
  - Empty jug X onto the floor
- Goal: J1 has G1 water and J2 G2
  - G1 or G0 can be -1 to represent any amount

Example: Water Jug Problem



Given full 5-gal. jug and empty 2-gal. jug, fill 2-gal jug with one gallon

- State representation?
  - General state?
  - Initial state?
  - Goal state?
- Possible actions?
  - Condition?
  - Resulting state?

Action table

Name	Cond.	Transition	Effect

Example: Water Jug Problem

Given full 5-gal. jug and empty 2-gal. jug, fill 2-gal jug with one gallon

- State representation?
  - –General state?
  - –Initial state?
  - –Goal state?
- Possible actions?
  - –Condition?
  - –Resulting state?

Action table

Name	Cond.	Transition	Effect
dump1	$x>0$	$(x,y)\rightarrow(0,y)$	Empty Jug 1
dump2	$y>0$		Empty Jug 2
pour_1_2	$x>0$ & $y<C2$		Pour from Jug 1 to Jug 2
pour_2_1	$y>0$ & $X<C1$		Pour from Jug 2 to Jug 1

So…

- How can we represent the states?
- What’s an initial state
- How do we recognize a goal state
- What are the actions; how can we tell which ones can be performed in a given state; what is the resulting state
- How do we search for a solution from an initial state given a goal state
- What is a solution? The goal state achieved or a path to it?

Search in a state space

- Basic idea:
  - Create representation of initial state
  - Try all possible actions & connect states that result
  - Recursively apply process to the new states until we find a solution or dead ends
- We need to keep track of the connections between states and might use a
  - Tree data structure or
  - Graph data structure
- A graph structure is best in general…

Formalizing state space search

- A state space is a **graph** (V, E) where V is a set of **nodes** and E is a set of **arcs**, and each arc is directed from a node to another node
- **Nodes:** data structures with state description and other info, e.g., node's parent, name of action that generated it from parent, etc.
- **Arcs:** instances of actions, head is a state, tail is the state that results from action

Formalizing search in a state space

- Each arc has fixed, positive **cost** associated with it corresponding to the action cost
  - Simple case: all costs are 1
- Each node has a set of **successor nodes** corresponding to all legal actions that can be applied at node's state
  - **Expanding** a node = generating its successor nodes and adding them and their associated arcs to the graph
- One or more nodes are marked as **start nodes**
- A **goal test** predicate is applied to a state to determine if its associated node is a goal node

What does “search” look like for a particular problem?

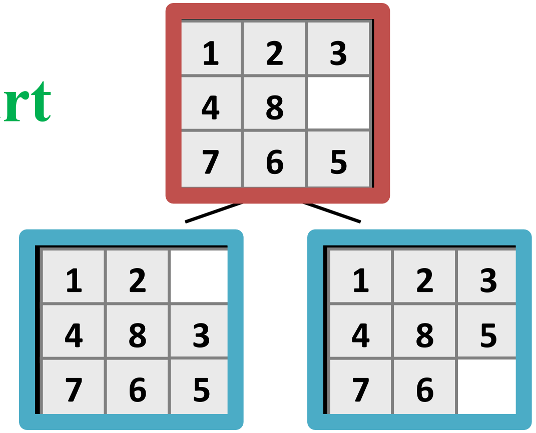
start

1	2	3
4	8	
7	6	5

goal

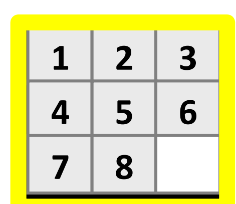
1	2	3
4	5	6
7	8	

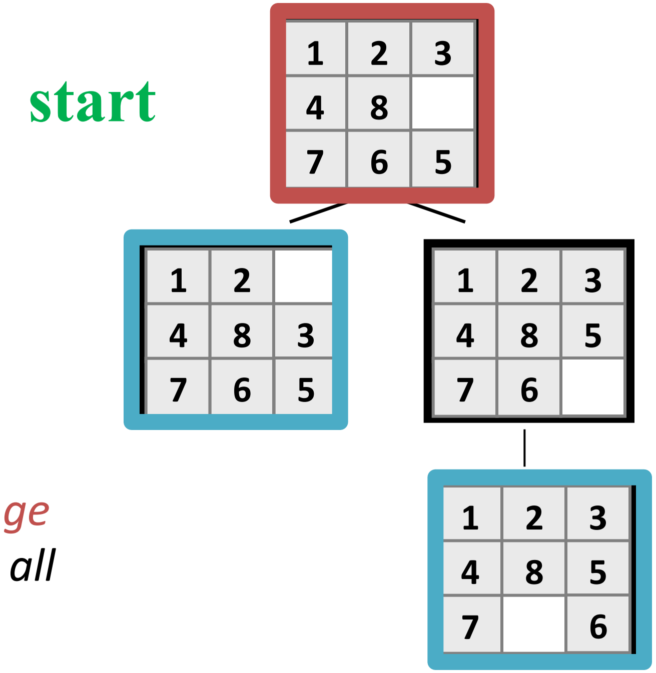
start



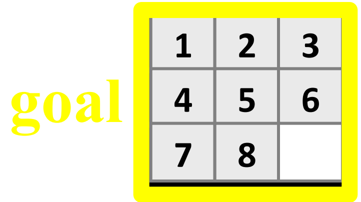
Expanding a node on the fringe
(taking a certain action)

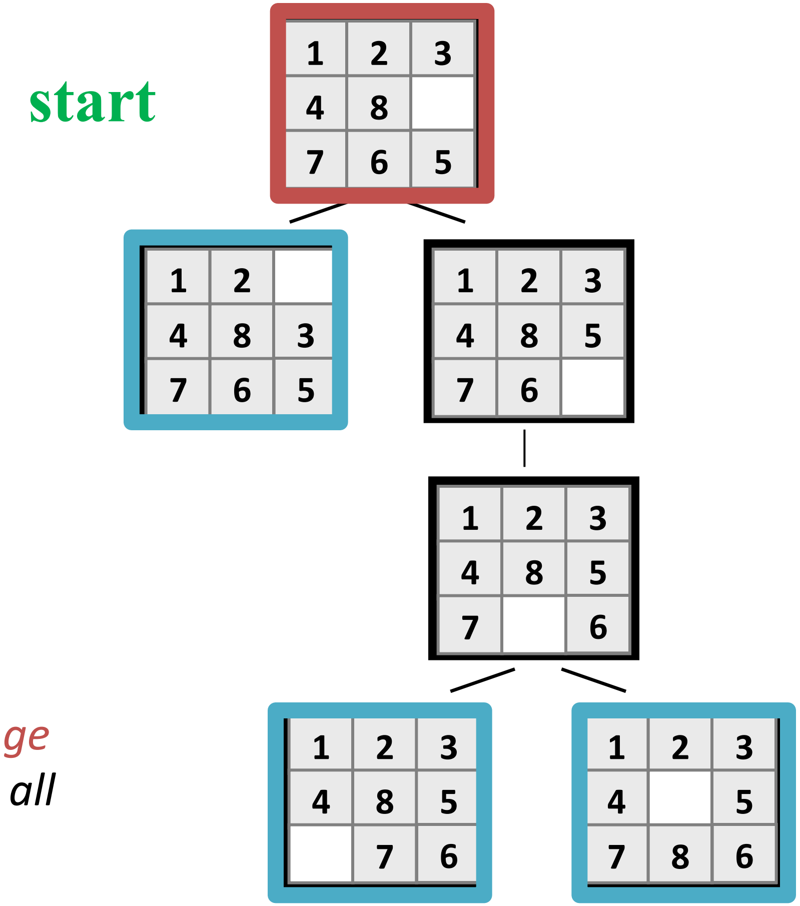
goal



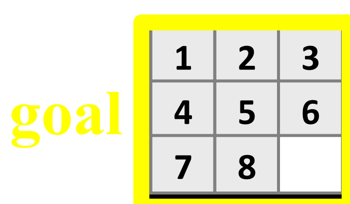


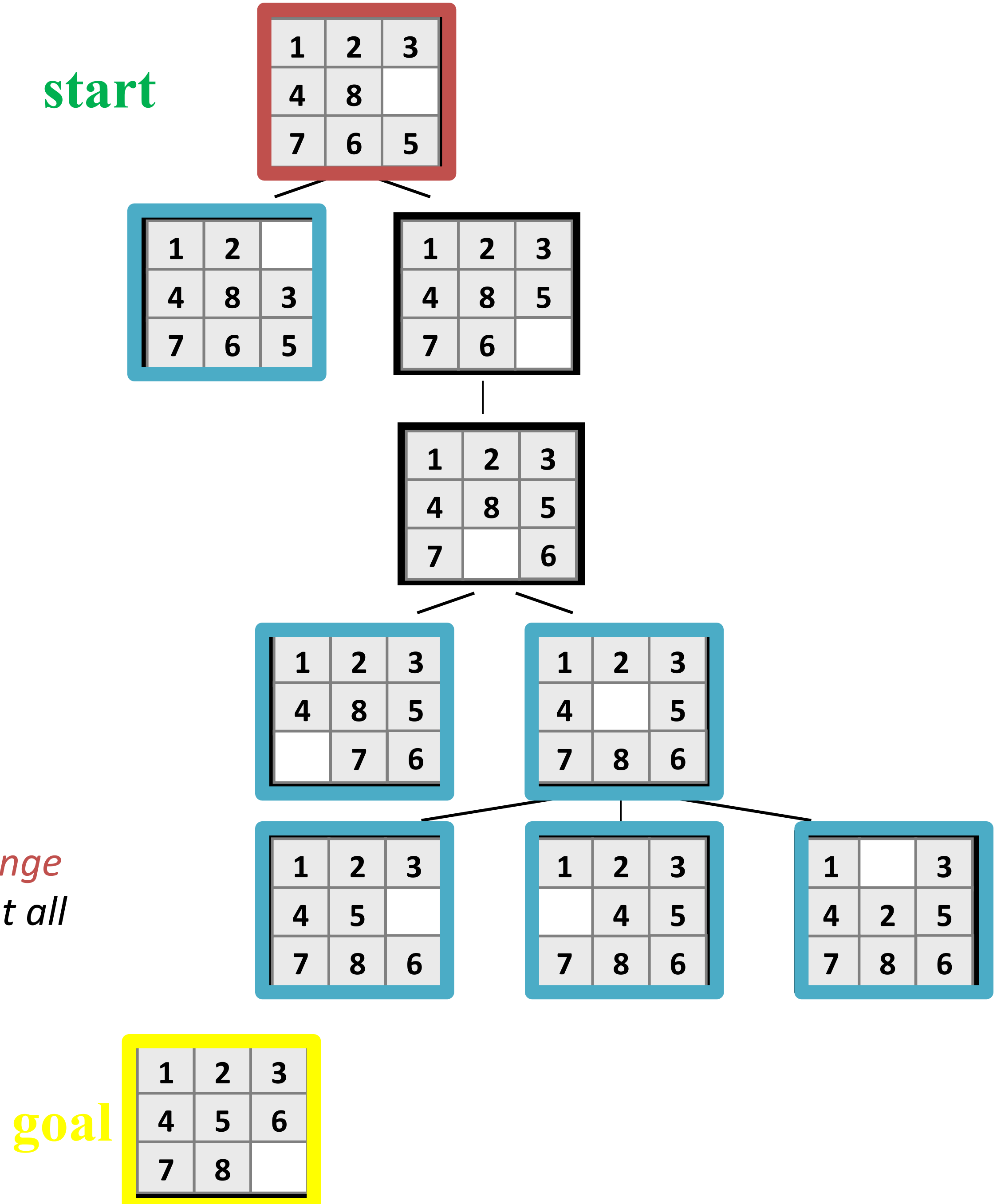
Expanding a node on the fringe (taking a certain action). Not all actions shown.



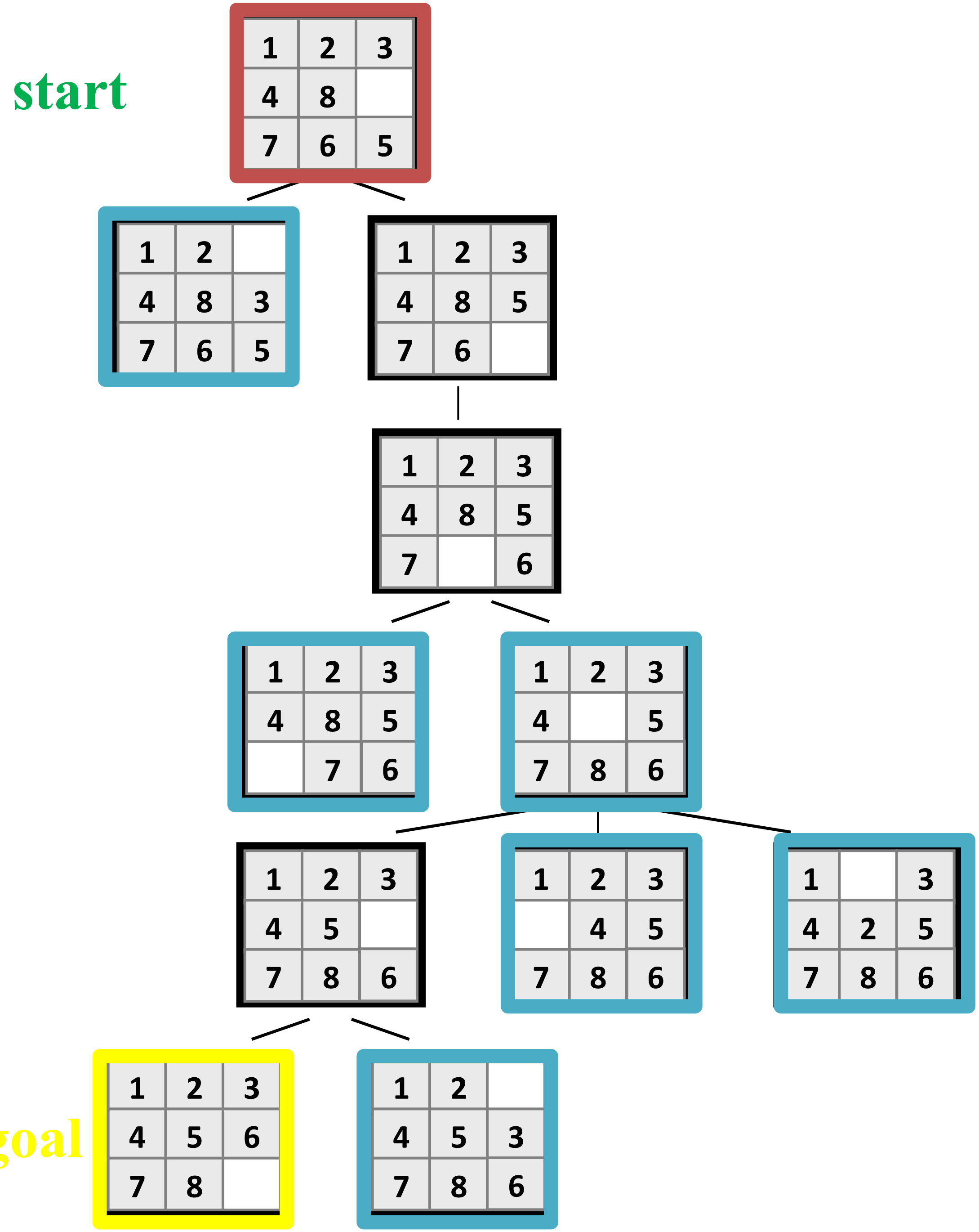


Expanding a node on the fringe (taking a certain action). Not all actions shown.





Expanding a node on the fringe (taking a certain action). Not all actions shown.


start
goal