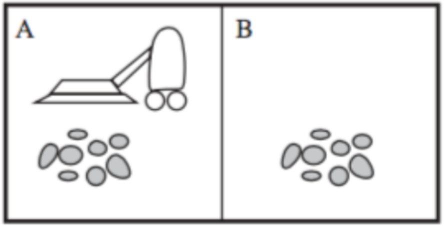
August 31: States

Opening Example: Consider the two-location vacuum world problem. At any given time, we may have a tuple that contains the vacuum's location, whether tile 1 is clean or dirty, and whether tile 2 is clean or dirty. How many distinct such tuples are possible?



1 2 3

[A, ['Dirty', 'Dirty']]

action: clean

OR

move to B

Counting Review

vacuum AND tile 1 AND tile 2

Definition: The *product rule* handles *and* statements. If a procedure can be broken down into two tasks and there are n_1 distinct ways to perform the first task and n_2 distinct ways to handle the second task, then there are

$\boxed{n_1 \cdot n_2}$ ways to do both Task 1 and 2

Definition: The *sum rule* handles (exclusive) *or* statements. If a procedure can either be done in n_1 ways **or** n_2 ways (where none of the n_1 and n_2 are the same), then there are

$\boxed{n_1 + n_2}$ ways total to do the task

The Subtraction Rule (aka the **Inclusion-Exclusion Principle**), which handles the *inclusive or*. If a task can be done in n_1 or n_2 ways, then the number of ways to do the task is $n_1 + n_2$ *minus* the number of ways to do the task that are common between n_1 and n_2.

Opening Sol'n

Opening Example: Consider the two-location vacuum world problem. At any given time, we may have a tuple that contains the vacuum's location, whether tile 1 is clean or dirty, and whether tile 2 is clean or dirty. How many distinct such tuples are possible?
Solution:

A or B
1 + 1

C or D

C or D

ways

$$\underset{\text{loc of vacuum}}{\underline{2}} \cdot \underset{\text{status A}}{\underline{2}} \cdot \underset{\text{status B}}{\underline{2}} = 2^3 = 8.$$

obj

Opening Sol'n

Opening Example: Consider the two-location vacuum world problem. At any given time, we may have a tuple that contains the vacuum's location, whether tile 1 is clean or dirty, and whether tile 2 is clean or dirty. How many distinct such tuples are possible?

Solution: There are **2** locations, and **4** possible dirt configurations: 1 with both dirty, 1 with both clean, and 2 with one dirty and one clean. The total possible tuples include both a vacuum status *and* a dirt status, so we multiply these:

$$\underbrace{2}_{\text{Vac \#ways}} \cdot \boxed{\underbrace{4}_{\text{dirt \# ways}}}$$

1 way for both dirty

$$C(2,2) = \frac{2!}{2!0!}$$

2 way for 1xC & 1xD

$$C(2,1) = \frac{2!}{1!1!}$$

Announcements and To-Dos

A clean dirty

move to B

A dirty clean

suck!

Announcements:

1. Homework 1 posted later this week.

Last time we learned:

1. We started the "agents" notebook. (To be Cont'd Friday)

To do:

1. Peek at/play with rest of "agents" notebook.

Agents Underview:

We talked about a few types of task environments:

1. Descriptors for environments included ways to describe the number of agents, whether they worked in unison, whether we had to worry about randomness (including in the future states, in our ability to observe the world correctly, or as a direct result of our actions),

And a few types of agents:

1. *Reflex* agents only use their current knowledge to decide an action: they don't predict or calculate the future. These can be simple or model-based.
2. *Goal* and *utility* based agents try to move or act towards a pre-specified end result: either a binary end-result or in a "high score" way.

States

Definition: The *sample space* or *state space* is the set of all possible work configurations, or *states*. We may also refer to them as *states-of-the-world* (SOW).

Definition: A *state space graph* a mathematical representation of the problem:

1. Each state is a vertex on the graph
2. Directed edges connect states by corresponding agent actions

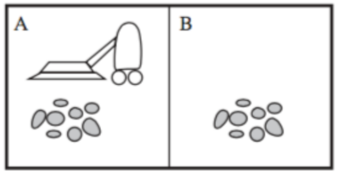
States Graphs

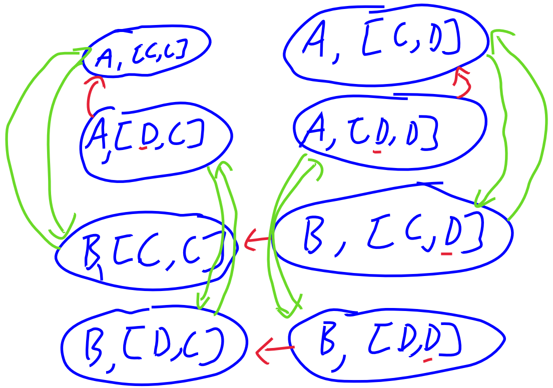
C=clean
D=dirty

loc, A Co/D
loc B Co, D

L [,]
↑ A or B of vacuum

Example: Construct a state space graph for the two-agent Vacuum world:

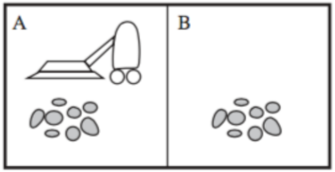




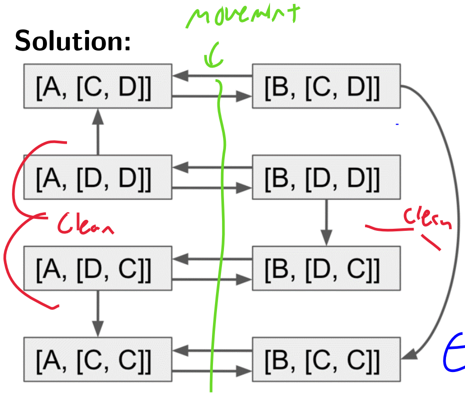
Actions: 1 Suck/clean (if D)
2 move

States Graphs

Example: Construct a state space graph for the two-agent Vacuum world:



Solution:



GOAL

← this row

Counting States

In general, the appropriate algorithm to use depends on the number of states. Do we want to search *all* possible states? Or just some? How many could we skip?

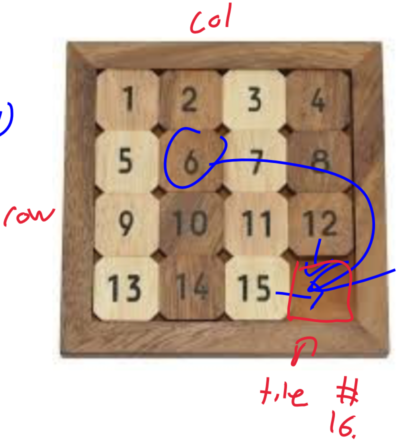
Example: Consider this tile-sliding game. How many possible states can the board be in (assuming no half-moved tiles)?

Each loc. has one tile (or null)

16 distinct   distinct

OR

each tile has exactly one location.



[[0,0] empty tile]
[0,1] , —

Counting States

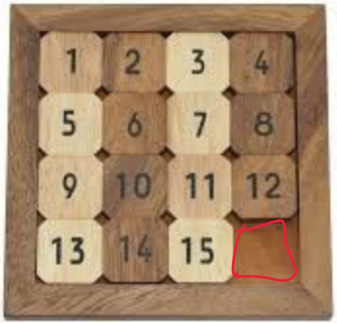
In general, the appropriate algorithm to use depends on the number of states. Do we want to search *all* possible states? Or just some? How many could we skip?

Example: Consider this tile-sliding game. How many possible states can the board be in (assuming no half-moved tiles)?

Solution: This is an ordering problem, so it's a permutation. We can treat the blank space as a "null tile" or the 16th object that we're ordering, so there are $P(16, 16) = 16!$ ways.

- col

row

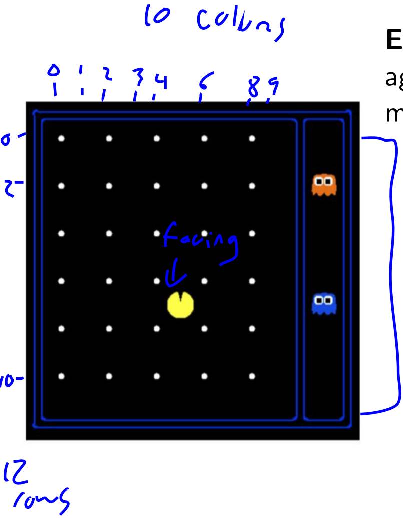


[3,3] , empty

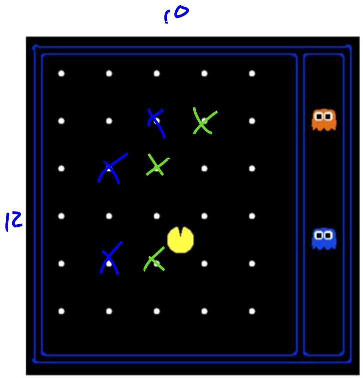
Recall that we can think of this as an *and* statement for counting:

$$\underbrace{16}_{\text{pos. tiles in loc 1}} \overbrace{\cdot}^{\text{and then}} \underbrace{15}_{\text{pos. tiles in loc 2}} \overbrace{\cdot}^{\text{and then}} \underbrace{14}_{\text{pos. tiles in loc 3}} \quad \ldots$$

(NB: we could alternatively track each tile's location instead of each location's tile!)



Example: What is the size of the state space for this Pac-Man agent? White dots represent consumable food, the ghosts can move, and the total grid is 10 by 12.



Example: What is the size of the state space for this Pac-Man agent? White dots represent consumable food, the ghosts can move, and the total grid is 10 by 12.

Solution: Let's count each object:

1. Pac Man has 10x12 locations, and 4 possible "facings."
2. The ghosts have 12 locations each. if overlap possible
3. There are 30 (5 x 6) food ~~items~~ locations, and *each* can either be eaten or uneaten. This represents 2^{30} states of just the foods alone.

These are all *and* counts in our SOW, so the total is:

$$\underbrace{120}_{Pac} \cdot \overbrace{4}^{Facing} \cdot \underbrace{12}_{Blue} \cdot \overbrace{12}^{Red} \cdot \underbrace{2^{30}}_{food} = 74,217,034,874,880$$

ghosts: no overlap: $(12 \cdot 11)$

$= (12^2 - 12)$

Intro to Searching

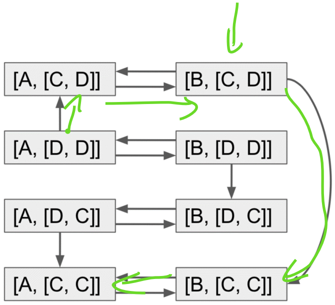
Whether it's a game or deciding on the optimal decision in a complex world (robotics, policy-making, etc.), we may have to navigate through this entire state space to try to find a way from where we start to some best state or set of states.

Definition: A *search problem* consists of:

1. state space
2. transition model
3. actions
4. initial state
5. goal

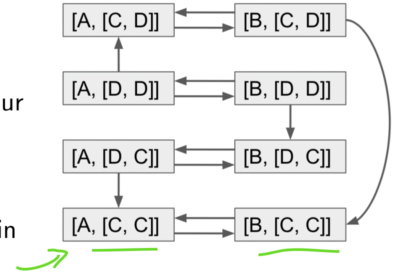
Searching

1. **State Space**
   a) Contains all possible ways the world could look: forms a directed graph (vertices/nodes)
   b) A **path** in the state space is a sequence of states connected by actions.
   c) A **path cost function** assigns a numeric cost (often in *utility*) to each path.
   d) We often sum the **step costs** to compare paths.
2. **Transition Model**
   a) a function that returns state_new resulting from the action taken in state_old.
   b) A *successor* is any state reachable from a given state by a single action.



Searching

3. **Actions**
   a) These are the actual choices presented to the agent.
4. **Initial States**
   a) The one, often unique, starting state: e.g. [A, 'dirty'] for our vacuum.
5. **Goal test**
   a) Determines whether a given state is the goal state
   b) There may be multiple goal states: e.g. winning positions in Chess.

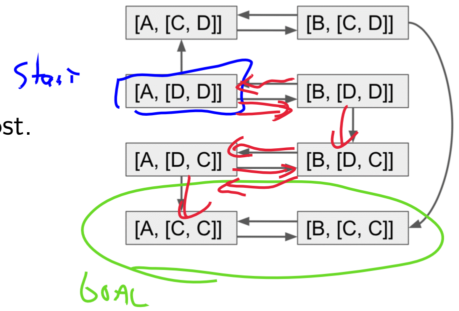


Searching

A **solution** to a problem is a sequence of actions that leads from the initial state to the goal state.

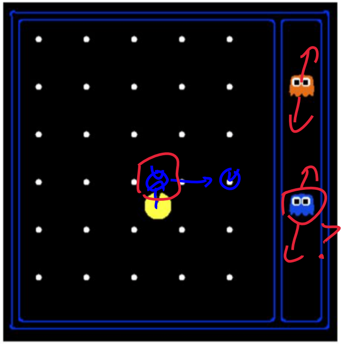
An **optimal solution** is a solution with the lowest path cost.

Sol:



The Full Problem: Pac Man

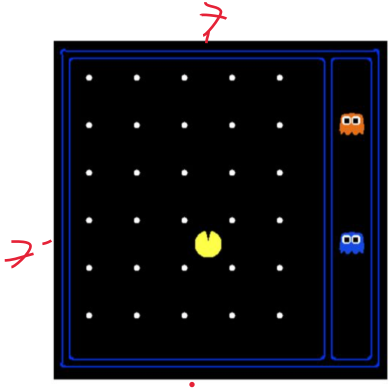
1. state space
2. transition model → what happens when we push direction
3. actions – directions
4. initial state
5. goal



1) If we push up:
   1) PM moves
   2) PM may eat food
   3) Ghost may move to us

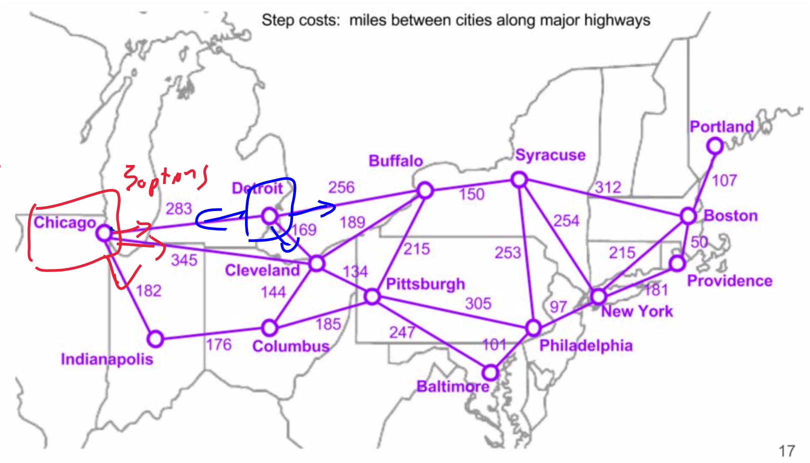
The Full Problem: Pac Man

1. state space : Loc/facing of Pac-Man, foods, ghosts
2. transition model : Update locations, food statuses
3. actions : Move N/S/E/W, maybe an 'eat' button?
4. initial state : row 7, col 5, N, all food uneaten, etc.
5. goal : all food status 'eaten'



A Travel Problem

1. state space
   vertices
2. transition model
   arrive @ new city
3. actions
   drive to new loc.
4. initial state
5. goal



A Travel Problem

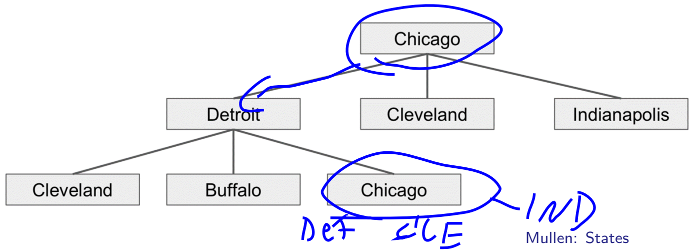
1. state space
2. transition model
3. actions
4. initial state
5. goal

1. state space : List of cities
2. transition model : roads
3. actions : drive on a road
4. initial state : e.g. ‘Chicago’
5. goal : e.g. is_state()==’Boston’

Search Trees

A search tree:

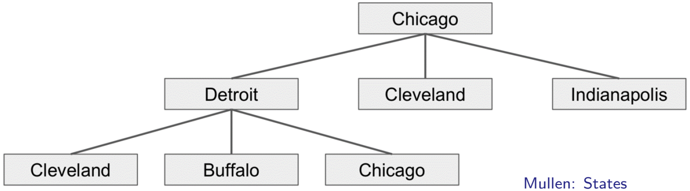
1. Is a "what if" tree of plans and outcomes.
2. The initial state is the root node.
3. Children of each node are its successors
4. For most problems, we neither can nor want to build the whole tree!



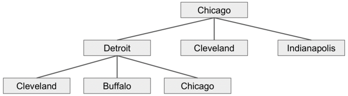
Search Trees

As we unfold a search tree:

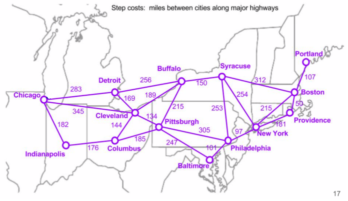
- We are tasked with expanding from the current state and asking where to explore more:
- We can **generate** a new set of states :
  Leaves go on the **frontier**, **internal vertices** are the **explored set**.
- Our selection strategy will change which leaves we might explore further
- We should track the explored set to **avoid loops and redundant paths**.



Trees vs Graphs



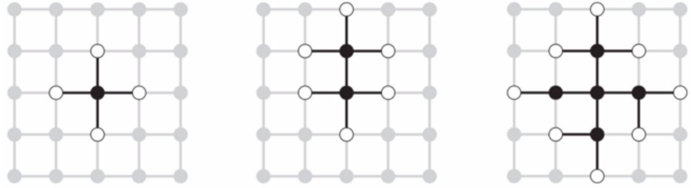
- Mathematical representation of search problem
- Nodes are world configurations
- Arcs and edges represent successors (action results)
- The goal test is a set of nodes
- Each state only occurs once



- Nodes show states, but correspond to **paths**
- Children are successors.
- Leaves are the frontier compared to an internal explored set.
- Root is initial state
- Construct on demand, as little as possible

Moving Forward

We can take these types of problems to capture behavior on e.g. a 2D lattice (or board)



- This week:
  1. Play with “agent” nb some. ←
  2. Eyes open for upcoming HW.
- Next time: **Searching** state spaces begins!

12