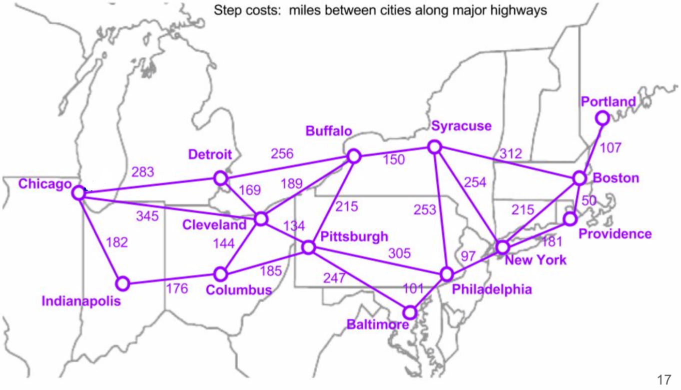
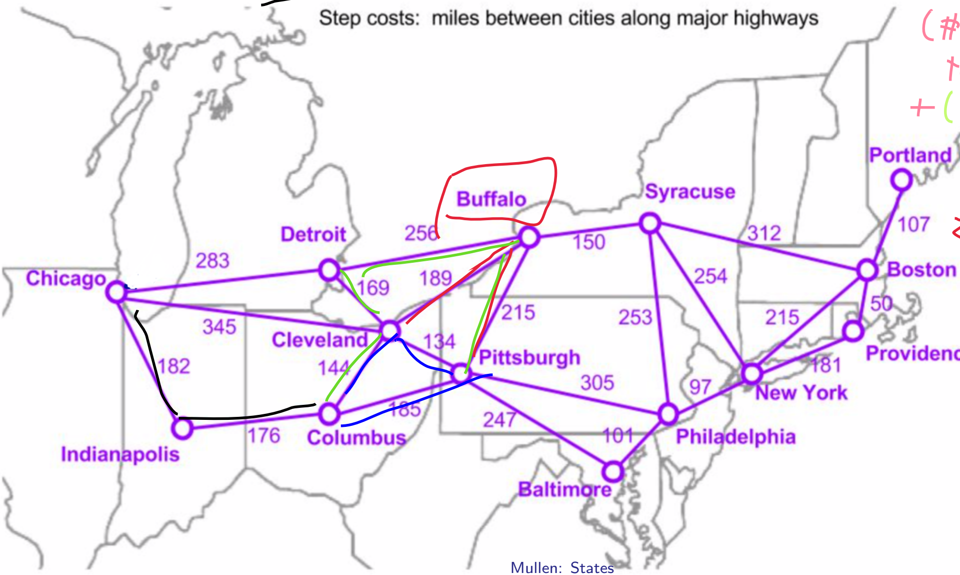
~~August 31~~ 9-2: ~~States~~ Breadth-First Searches

Opening Example: Consider map problem below, with the goal of traveling from Chicago to Pittsburgh. How many possible routes are there, without ever revisiting a city? What properties of this graph does that depend on?



Opening Sol'n: a hard counting problem

"informed"

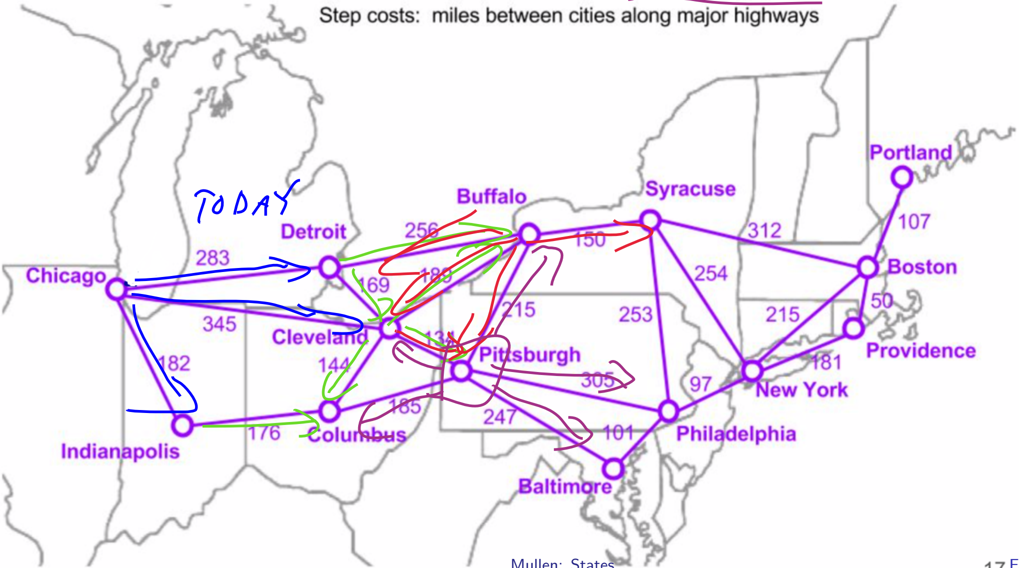


Step costs: miles between cities along major highways

1) paths don't g.t Buffalo
(# of ways to CLE)
+ (# of ways Columbus)

2) (# ways to Buffalo) · (ways Buffalo to PITT)

Opening Sol'n: a hard counting problem (Backsolve?)



Announcements and To-Dos

Graphs vs. Trees

shows all at once

"unfold" from 1 location

Announcements:

1. Homework 1 posted later this week.
2. Make sure you have both Piazza (`piazza.com/colorado/fall2020/csci3202`) and canvas (`https://canvas.colorado.edu/courses/66935`) access!

Last time we learned:

1. About state spaces.

To do:

1. Prepare for more "agents" next time, plus eyes open for HW1 posting!

States Recap:

Last time we talked about **states.**

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

We are exploring two possible representations of state spaces: graphs and trees. For a graph:

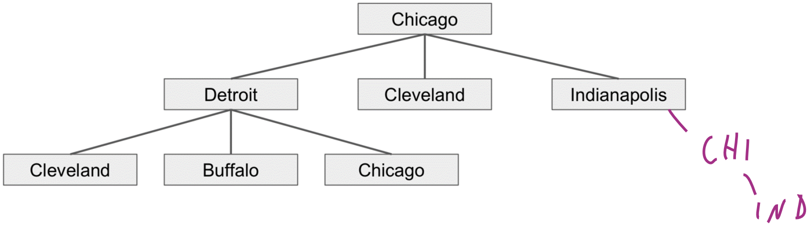
1. Each state is a vertex on the graph
2. Directed edges connect states by corresponding agent actions

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!

repeats itself if edges are bidirectional

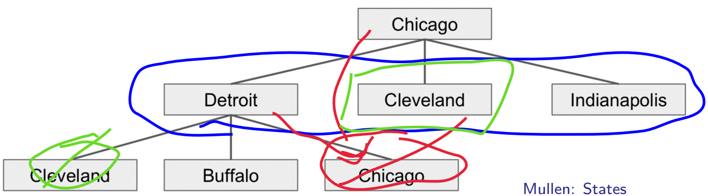


Search Trees

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**.

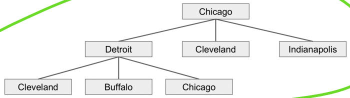
initialize: Chicago frontier
{ } explored



- set of frontier, step 1

Trees vs Graphs

Graph



Chicago

Detroit

Cleveland

Indianapolis

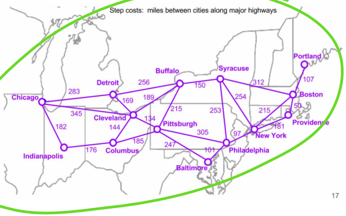
Cleveland

Buffalo

Chicago

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

Trees

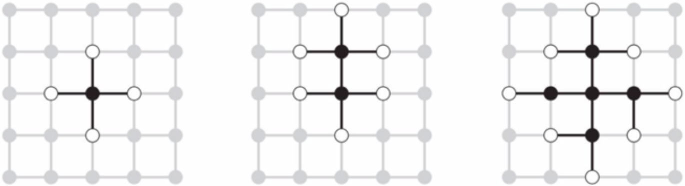


- Nodes show states, but correspond to **paths** (may be duplicate)
- Children are successors.
- Leaves are the frontier compared to an internal explored set.
- Root is initial state
- Construct on demand, as little as possible

Tree-like problems

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

1. Most generally, a tree can be used to capture any kind of graph
2. It can also be used to capture any decision process with finitely many decisions at any given point
3. We try to exploit that tree representations of problems are often quite redundant! A board game ”state” that follows moves $A \rightarrow B$ and $B \rightarrow C$ may be covered with just evaluating $A \rightarrow C$ directly!



Searching

The general goal of a **search** algorithm is to navigate through the state space to find an ideal **path** to a goal state or states.

1. An **uninformed** search provides no additional information about states beyond that in the problem definition.
2. In an **informed** search, we have some prior knowledge of which non-goal states are “more promising” than others. Ex: taking pieces in chess.

Searching

The general goal of a **search** algorithm is to navigate through the state space to find an ideal **path** to a goal state or states.

1. An **uninformed** search provides no additional information about states beyond that in the problem definition.
2. In an **informed** search, we have some prior knowledge of which non-goal states are "more promising" than others. Ex: taking pieces in chess.

Two major variables in a search algorithm over a graph are:

Definition: The **branching factor**, represented by b, is the maximum number of successors of any node. (Same as max # of outgoing edges).

→ graph: shortest path

Definition: The **depth**, represented by d, is the depth (in the search tree) of the shallowest goal node. This node requires the fewest number of agent actions to (possibly) reach.

Searching Algorithmic Goals

As with any algorithm, we're very interested in best-case, average-case, and worst-case behavior. In particular, we should ask about:

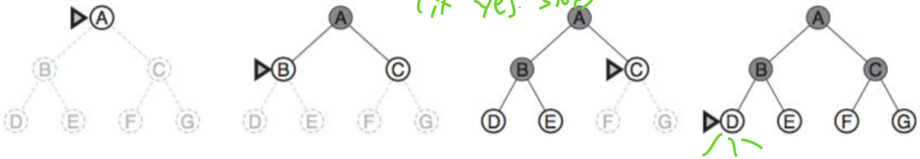
1. **Completeness:** is the algorithm guaranteed to find a solution, when one exists?
2. **Optimality:** is the algorithm guaranteed to find the optimal solution, i.e. that with the lowest path cost?
3. **Time Complexity:** how long does it take - number of operations - to find a solution?
4. **Space complexity:** how much memory is needed to find the solution?

Breadth-First Search (BFS)

Breadth-first search or BFS is an uninformed search algorithm based on tree representations of problems.

1. At each iterative step, expand all nodes at the current depth.
2. Then, proceed to the next layer, in FIFO-style (first in, first out) → order of frontier/exploration
3. Apply a goal test to each node *as it is generated*

(if yes: stop)

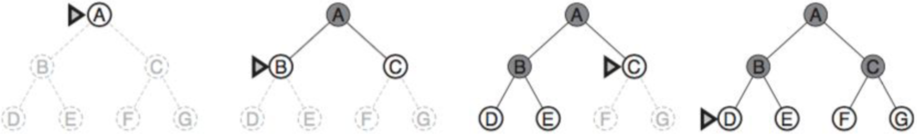


Explored:	{}	{a}	{a, b}	{a, b, c}
Frontier:	[A]	[B, C]	[C, d, e]	[D, E, F, G]

Breadth-First Search (BFS)

Breadth-first search or BFS is an uninformed search algorithm based on tree representations of problems.

1. At each iterative step, expand all nodes at the current depth.
2. Then, proceed to the next layer, in FIFO-style (first in, first out)
3. Apply a goal test to each node *as it is generated*



Explored:

Frontier:

BFS Example:

Consider our map problem, with the goal of traveling from Chicago to Pittsburgh. How does BFS solve this problem, if states are sorted alphabetically?

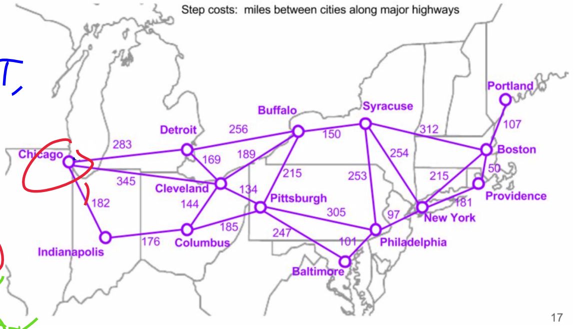
explored frontier

Step 0: { } {CHI}

Step 1; CHI: {CHI} {~~CLE~~, DET, IND}

{CHI, CLE}

{DET, IND, BUF, COL, ~~DET~~, ~~CHI~~ PITT} → stop!



BFS Example:

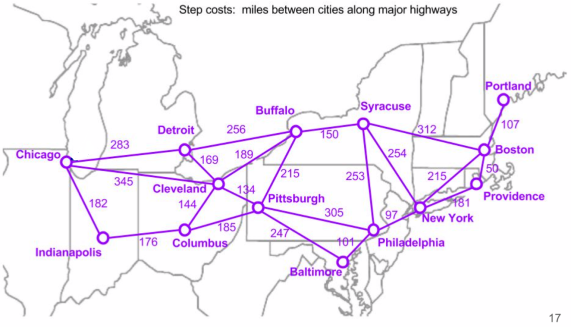
Consider our map problem, with the goal of traveling from Chicago to Pittsburgh. How does BFS solve this problem, if states are sorted alphabetically?

0 The root.

- 0a Unfold Chicago, discovering: {Cleveland, Detroit, Indianapolis}.
- 0b Explored set is: {Chicago}.
- 0c Check if those 3 are goal states.

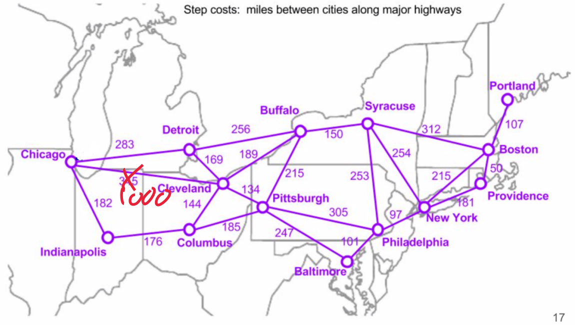
1 Level 1.

- 1a Unfold first new state (Cleveland), discovering {Buffalo, Pittsburgh, Columbus}. We ignore link to Detroit, because it's already in the explored set.
- 1b Explored set is now: {Chicago, Cleveland}.
- 1c Terminate BFS: solution of Pittsburgh is found!
- 1d (we would unfold I~~N~~D DET here, if needed)



BFS steps

We don't seem to be using our distance between each city! If we changed the step cost function, would this change our BFS result?



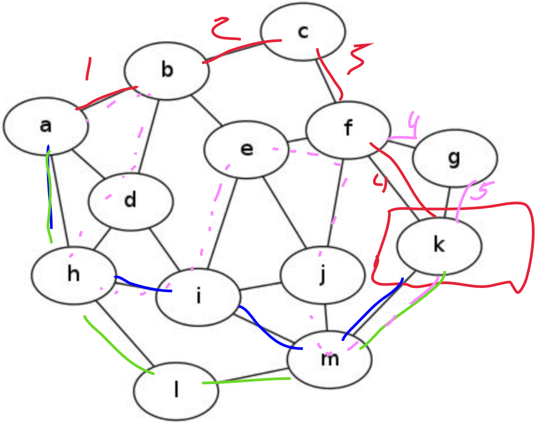
Step costs: miles between cities along major highways

What if we added a major traffic jam and tripled the time needed to move from Chicago to Cleveland?

BFS practice

Example,

Number the nodes in the search graph according to the order in which they would be expanded using BFS to find a path from a to k. Assume that nodes within a layer are expanded in alphabetical order.

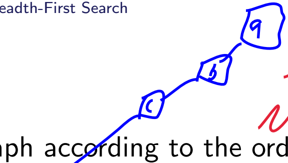


BFS practice

See: a-b-c-f-k

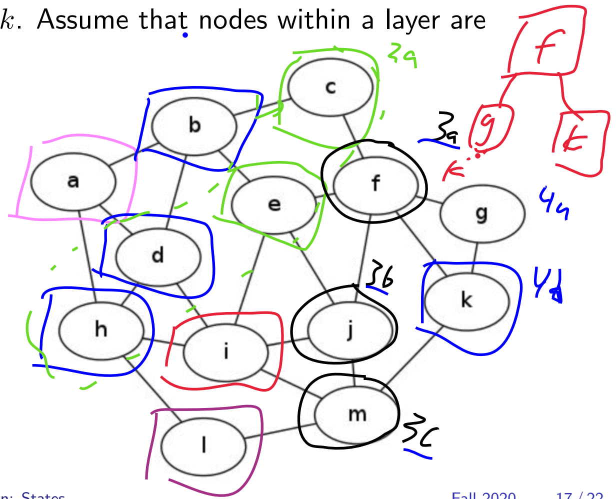
NOT: a-b-e-f-k

a-h-e-m-k



Example,

Number the nodes in the search graph according to the order in which they would be expanded using BFS to find a path from a to k. Assume that nodes within a layer are expanded in alphabetical order.



0 Explored: {}. Frontier: {a}.
1 Explored: {a}. Frontier: {b,d,h}.
2a Explored: {a,b}. Frontier: {c,d,e,h}.
2b Explored: {a,b,d}. Frontier: {c,e,h,i}.
2c Explored: {a,b,d,h}. Frontier: {c,e,i,l}.

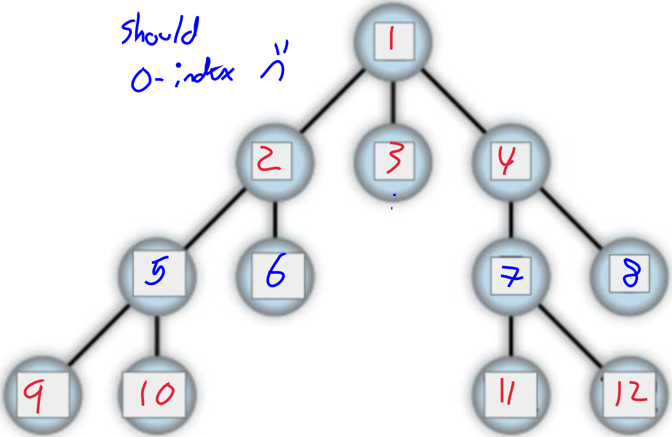
b=5

BFS practice

Example,

Number the nodes in the search tree according to the order in which they would be expanded using BFS. Assume that the goal is never found, and nodes within each layer move from left to right.

should 0-index ?!



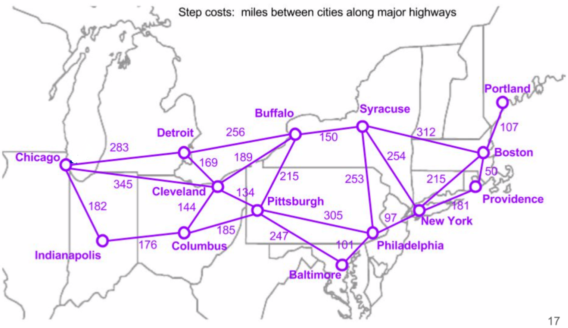
1) TOP-DOWN

2) LEFT-TO-RIGHT

1) explored

2) frontier order

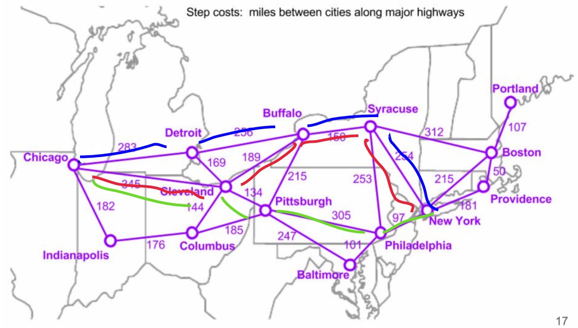
BFS properties



Is BFS **complete?** (does it find a route?)

Is BFS **optimal?** (does it find the best route?)

BFS properties

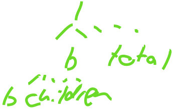

Step costs: miles between cities along major highways

Is BFS **complete?** Yes! We will go layer-by-layer until we eventually find *some* route to any city we want.

, good for no-edge weights   edge weights/utility

Is BFS **optimal?** Not really! Our graph includes distances/times and we're not using them: BFS will find (one of) the routes that passes through the least number of cities, but not the one that takes the least time. Consider e.g. Chicago to NYC!

BFS complexity

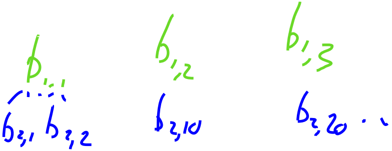


Time Complexity Suppose each layer generates b nodes, where we call b the *branching factor*. Suppose also the problem has $\underline{d}$ total layers.

0 Layer 0 (the root) has $b^0 = 1$ node.

1 Layer 1 generates $b^{\boxed{1}} = b$ nodes.

2 Layer 2 generates b from each of these, for $\underline{b^{\boxed{2}}}$ new nodes.



⋮

Layer k: b^k

BFS complexity

Time Complexity Suppose each layer generates b nodes, where we call b the *branching factor*. Suppose also the problem has d total layers.

0 Layer 0 (the root) has $b^0 = 1$ node.

1 Layer 1 generates $b^1 = b$ nodes.

$d \rightarrow (b^d)$

2 Layer 2 generates b from each of these, for b^2 new nodes.

$\vdots$

k Layer k generates b from each of the nodes in layer $k-1$, for b^k new nodes.

We end up with the sum of these, or a total of $1 + b + b^2 + \ldots b^d = \boldsymbol{O(b^d)}$

Space Complexity:

Geometric series!

BFS complexity

Time Complexity Suppose each layer generates b nodes, where we call b the *branching factor*. Suppose also the problem has d total layers.

0 Layer 0 (the root) has $b^0 = 1$ node.

1 Layer 1 generates $b^1 = b$ nodes.

2 Layer 2 generates b from each of these, for b^2 new nodes.

$\vdots$

k Layer k generates b from each of the nodes in layer $k-1$, for b^k new nodes.

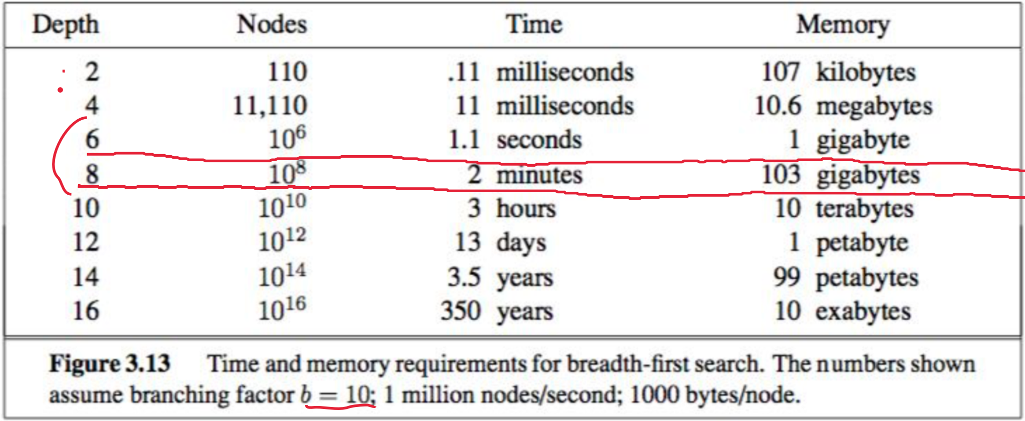
We end up with the sum of these, or a total of $1 + b + b^2 + \ldots b^d = \boldsymbol{O(b^d)}$

Space Complexity: Since we need to save every node in the explored set, if there are no redundant paths we have to save the entire penultimate later of this tree: this contained $\boldsymbol{O(b^{d-1})}$.

We *also* then need to store a list of every node on the frontier which was again $\boldsymbol{O(b^d)}$.

BFS complexity

In practice, even with b=10, the are often large proportions of redundant states: these may in practice make b much smaller.



Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

Figure 3.13 Time and memory requirements for breadth-first search. The numbers shown assume branching factor $b = 10$; 1 million nodes/second; 1000 bytes/node.

↑ ok!

↓ No BFS

Moving Forward

- This week:
  1. Play with “agent” nb some.
  2. Eyes open for upcoming HW.
- Next time: nb day on Friday, next lecture **deep** searching a week from now!