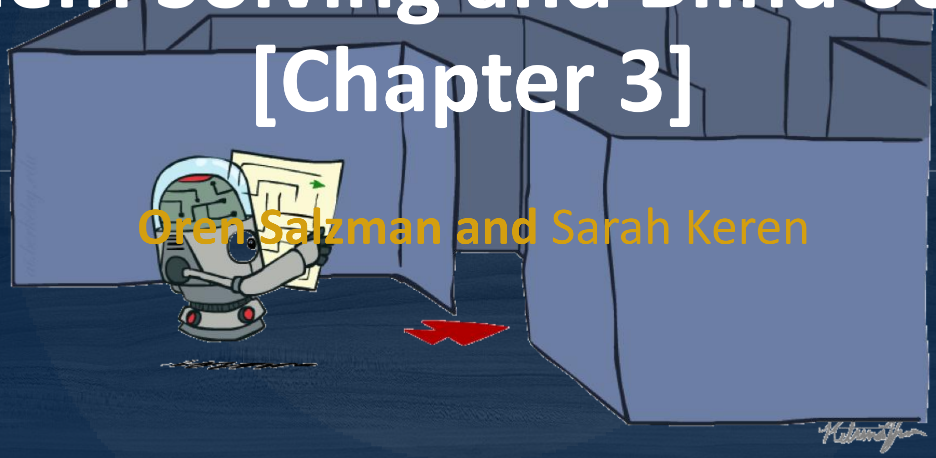
Introduction to AI – 236501
Problem Solving and Blind Search
[Chapter 3]



Oren Salzman and Sarah Keren

Slides adapted from ai.berkeley.edu by

Dan Klein, Pieter Abbeel and Anca Dragan

and from Shaul Markovitz @ Technion

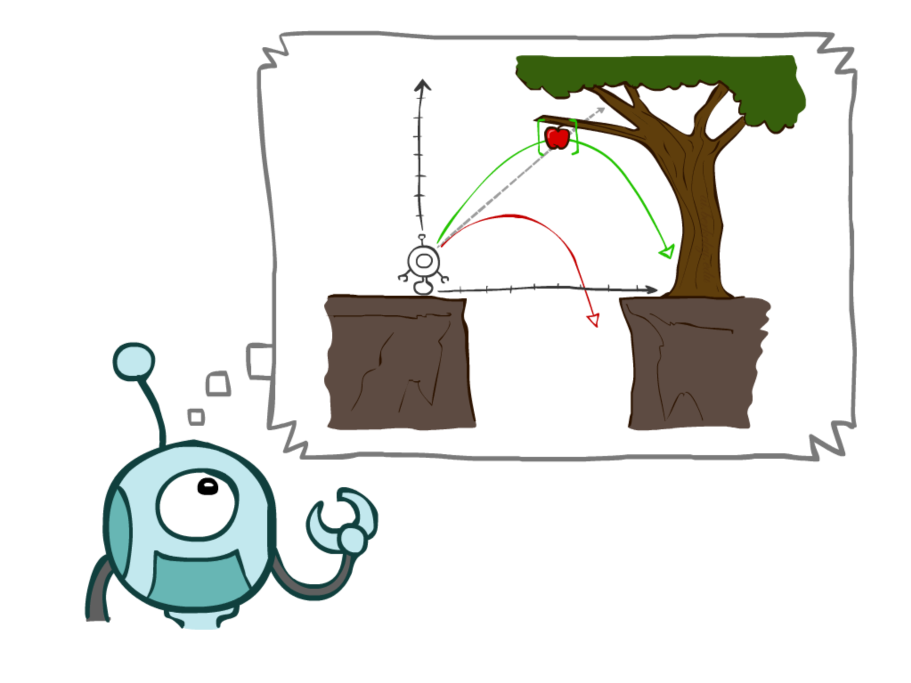
TECHNION
Israel Institute of Technology

The Henry and Marilyn Taub
Faculty of Computer Science

Today

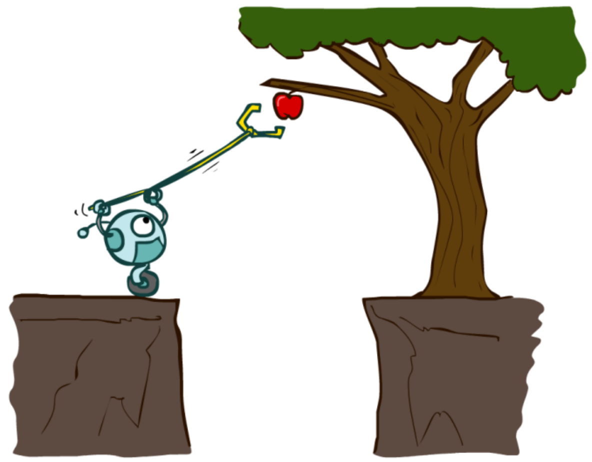
- Agents that Plan Ahead

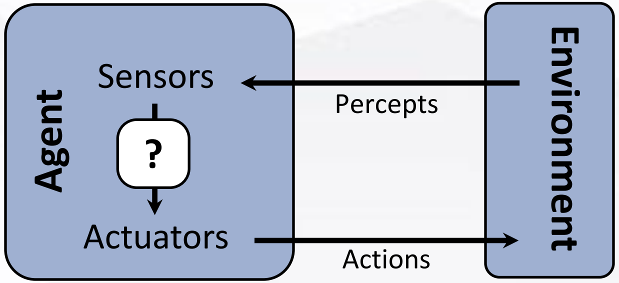
- Search Problems



Designing Rational Agents

- An **agent** is an entity that **perceives** and **acts**.
- A **rational** agent selects actions that **maximize** its (expected) **utility**.
- Characteristics of the **percepts**, **environment**, and **action space** dictate techniques for selecting rational actions





How do we design the rational agent's decision procedure?

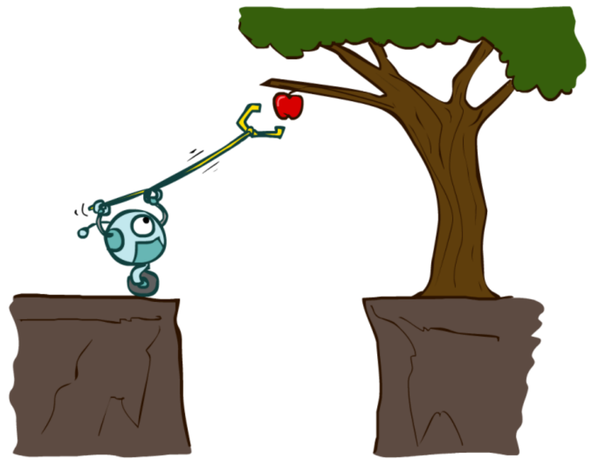
- An agent's **decision procedure** maps a world's **state** and a desired **goal** to an **action**
- We can hardcode the mapping

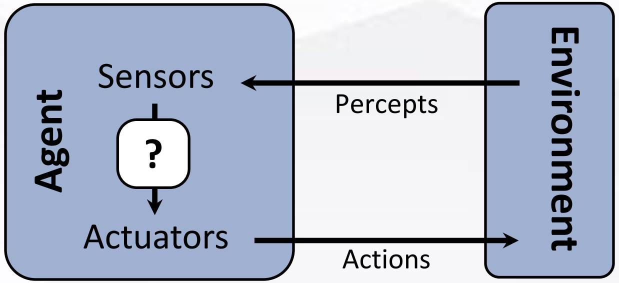
  $$\text{State} \times \text{Goal} \rightarrow \text{Action}$$

  - May be an infinite table
- We can hardcode the mapping

  $$\text{Percept} \times \text{Goal} \rightarrow \text{Action}$$

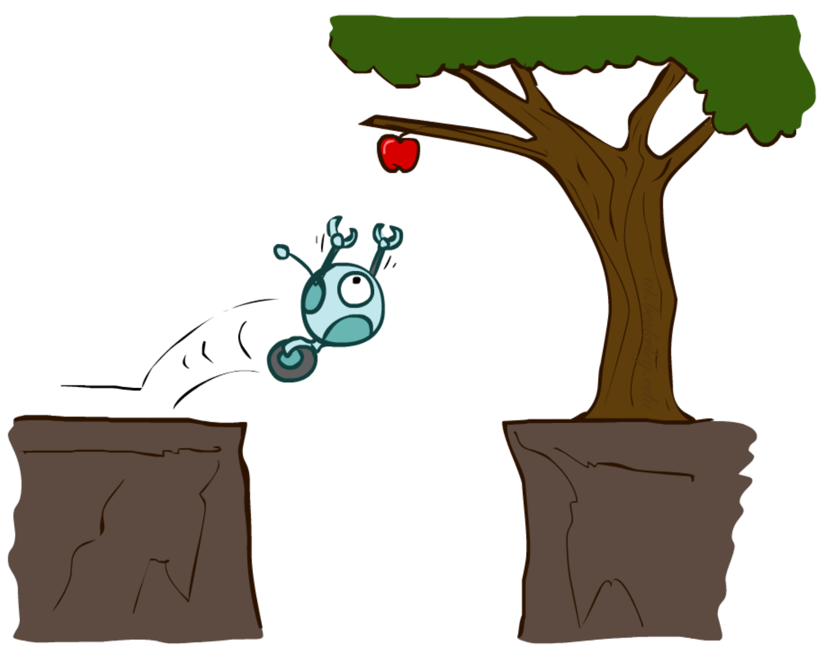
  - May (still) be an infinite table
  - This is called a **reflex agent**





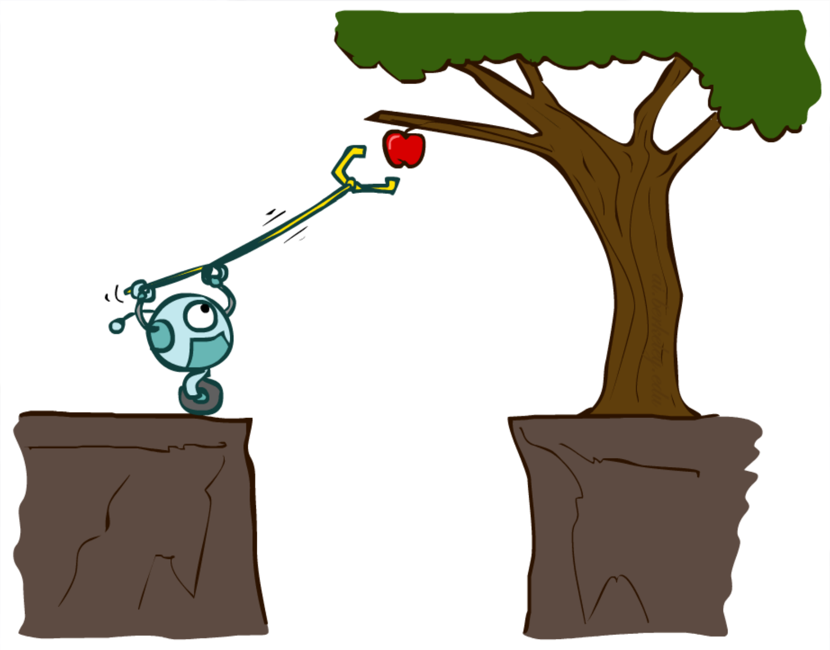
Reflex Agents

- Reflex agents:
  - Choose action based on current percept (and maybe memory)
  - May have memory or a model of the world's current state
  - Do not consider the future consequences of their actions
  - Consider how the world IS



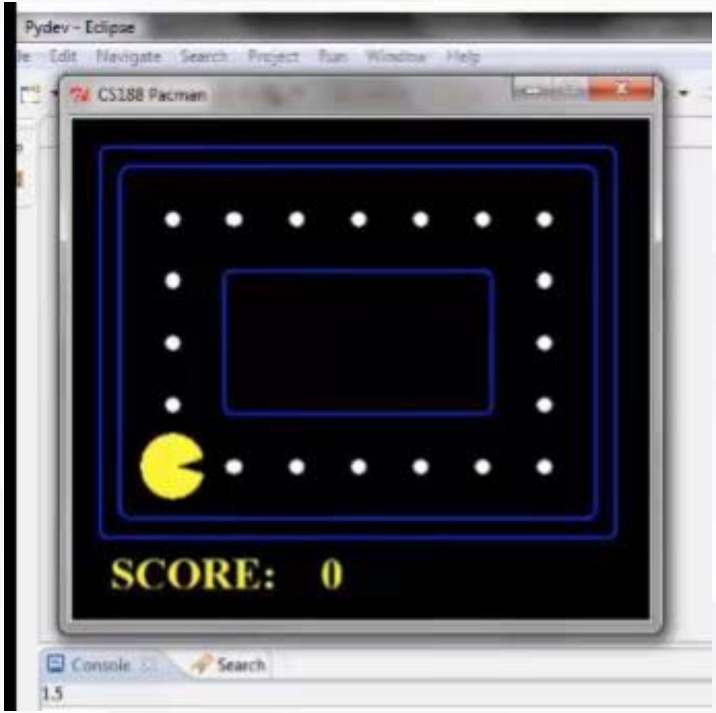
Planning Agents

- 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
- Planning vs. replanning



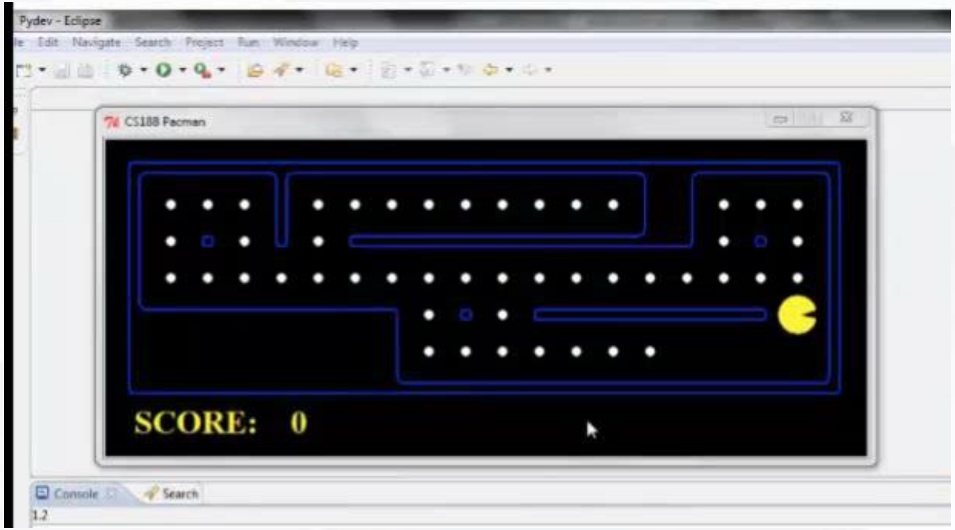
Video of Demo Reflex - Optimal

- Reflex agent
  - Move in the direction of the nearest uneaten dot
  - You can think ahead all you want, still end up doing the same thing
  - Can be rational



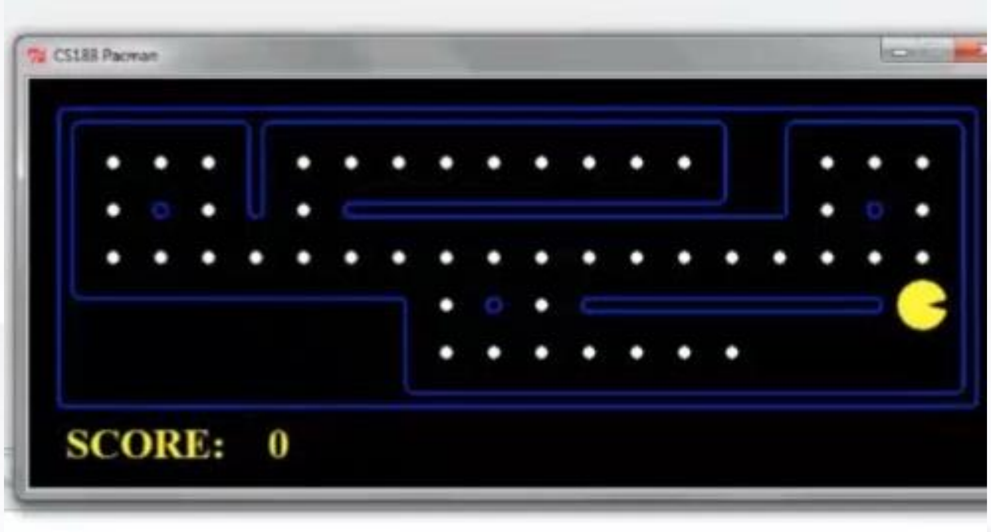
Video of Demo Reflex – incomplete

- Reflex agent
  - Move in the direction of the nearest uneaten dot
  - Can be incomplete
  - No planning
- Reflex agents not generally optimal, but they can be depending on circumstance.



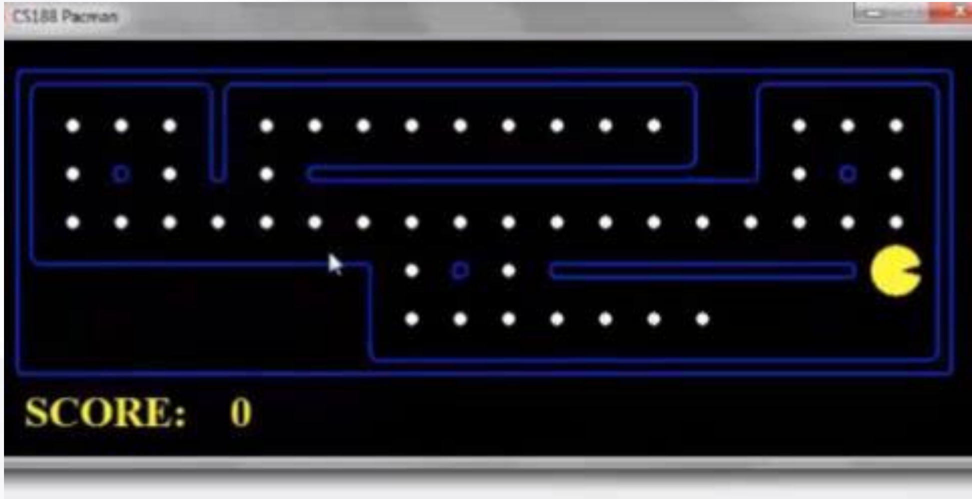
Video of Demo replaning reflex agent – sub optimal

- Replaning reflex agent
  - Move in the direction of the nearest uneaten dot
  - Clears board
  - Non-optimal
  - Fast



Video of Demo mastermind– Optimal

- Mastermind agent
  - Thinking, thinking, thinking…
  - Not just clear NEXT dot, but all dots
  - Know to end with dead end so it doesnt have to back up



Problem-Solving Agents

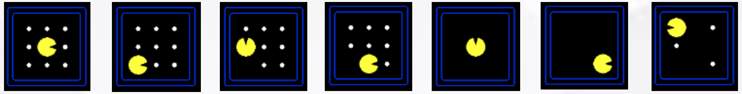
TECHNION
Israel Institute of Technology

The Henry and Marilyn Taub
Faculty of Computer Science

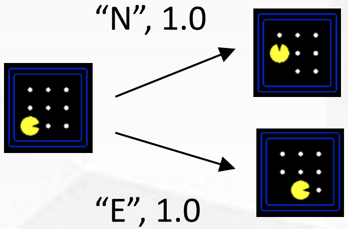
How do we formalize a search problem?

Search Problems

- A **search problem** consists of:
  - A **state space**



  - A **successor** function (with actions, costs) or a set of **operators**

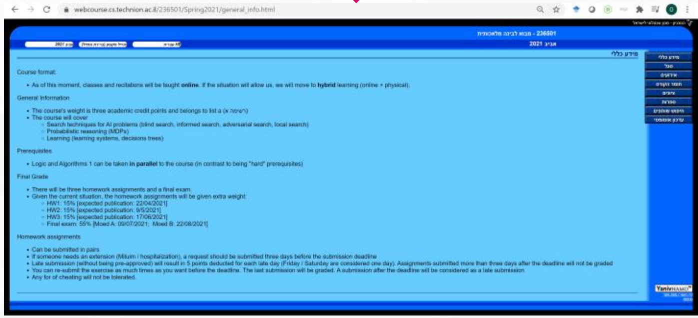


  - A **start state** and a **goal test**

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

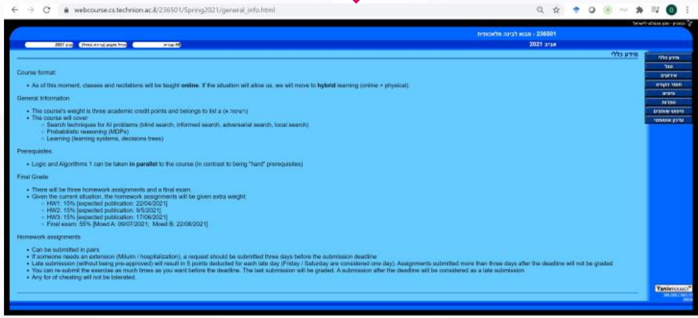
Example: searching the internet

- **Problem**: Given the Technion's homepage, find the series of clicks that will get you to the homepage of Intro To AI



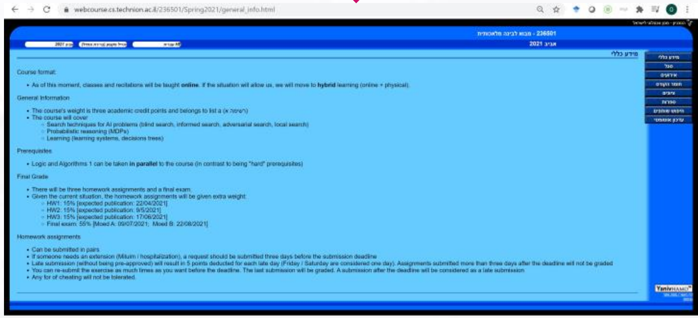
Example: searching the internet

- **Problem**: Given the Technion's homepage, find the series of clicks that will get you to the homepage of Intro To AI
- **State space:**



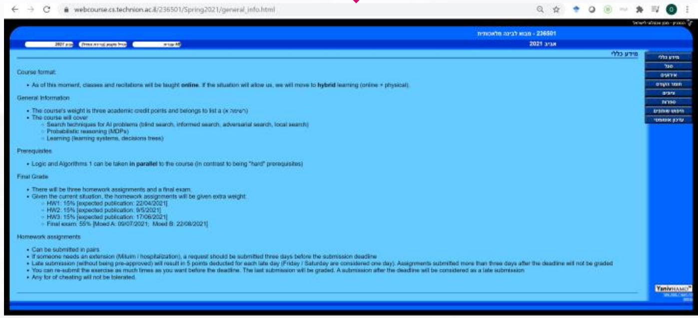
Example: searching the internet

- **Problem**: Given the Technion's homepage, find the series of clicks that will get you to the homepage of Intro To AI
- **State space:** all URLs



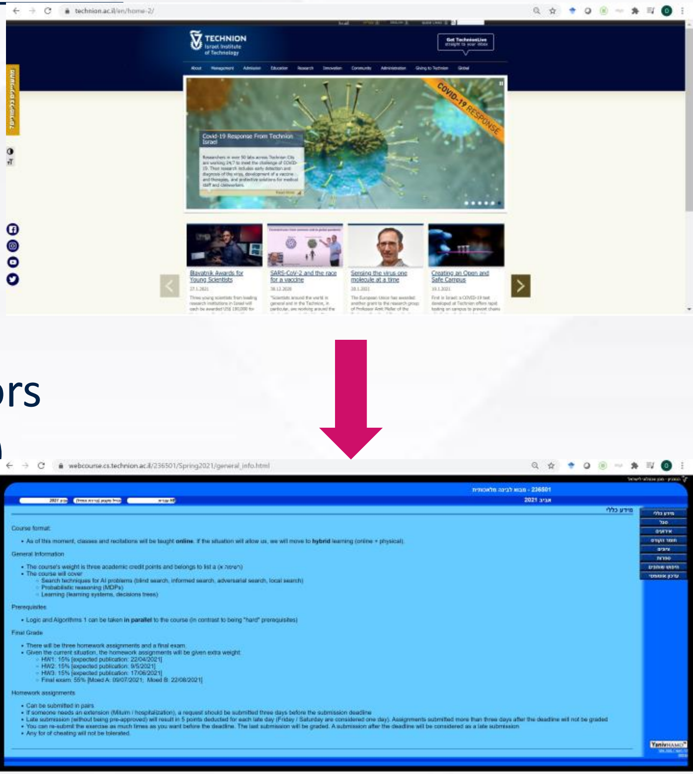
Example: searching the internet

- **Problem**: Given the Technion’s homepage, find the series of clicks that will get you to the homepage of Intro To AI
- **State space:** all URLs
- **Successor function:**



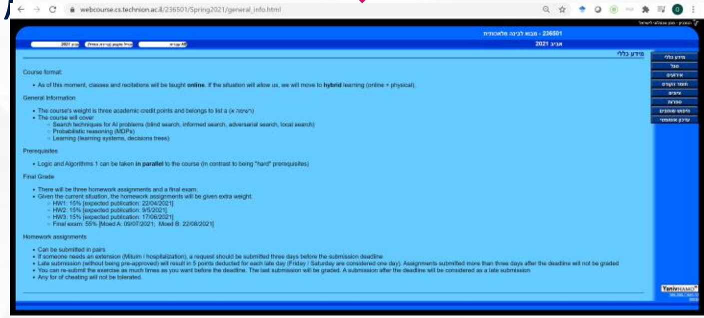
Example: searching the internet

- **Problem**: Given the Technion's homepage, find the series of clicks that will get you to the homepage of Intro To AI
- **State space:** all URLs
- **Successor function:** for each page the set of successors is defined by the links available at that page (cost = 1)



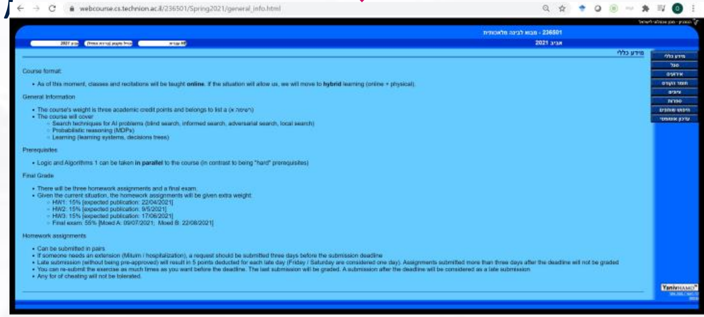
Example: searching the internet

- **Problem**: Given the Technion’s homepage, find the series of clicks that will get you to the homepage of Intro To AI
- **State space:** all URLs
- **Successor function:** for each page the set of successors is defined by the links available at that page (cost = 1)
- **Start state**:
- **Goal state**:



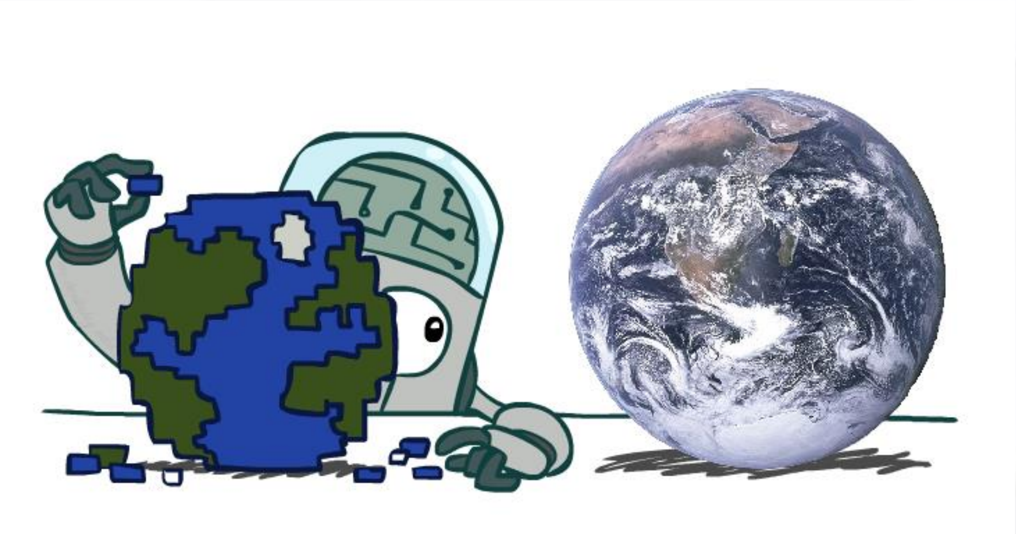
Example: searching the internet

- **Problem**: Given the Technion's homepage, find the series of clicks that will get you to the homepage of Intro To AI
- **State space:** all URLs
- **Successor function:** for each page the set of successors is defined by the links available at that page (cost = 1)
- **Start state**: https://www.cs.technion.ac.il/
- **Goal state**: https://webcourse.cs.technion.ac.il/236501/



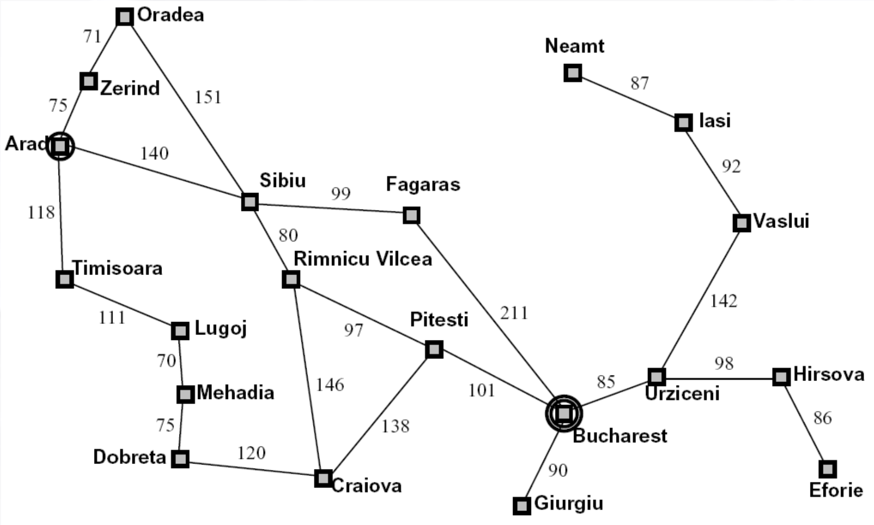
Search Problems Are Models

- Models aren't perfect
  - Too detailed, you can't solve
  - Not detailed enough, doesn't solve



Example: Traveling in Romania

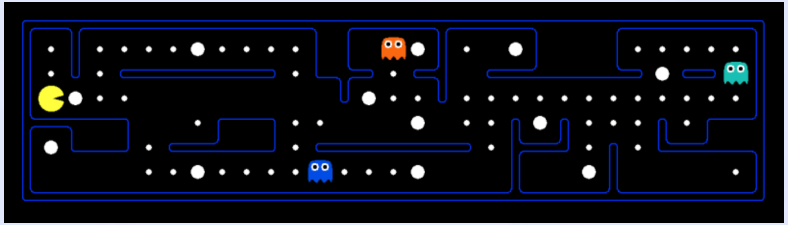
- State space:
  - Cities
- Successor function:
  - Roads: Go to adjacent city with cost = distance
- Start state:
  - Arad
- Goal test:
  - Is state == Bucharest?



- Solution?

What's in a State Space?

The world state includes every last detail of the environment

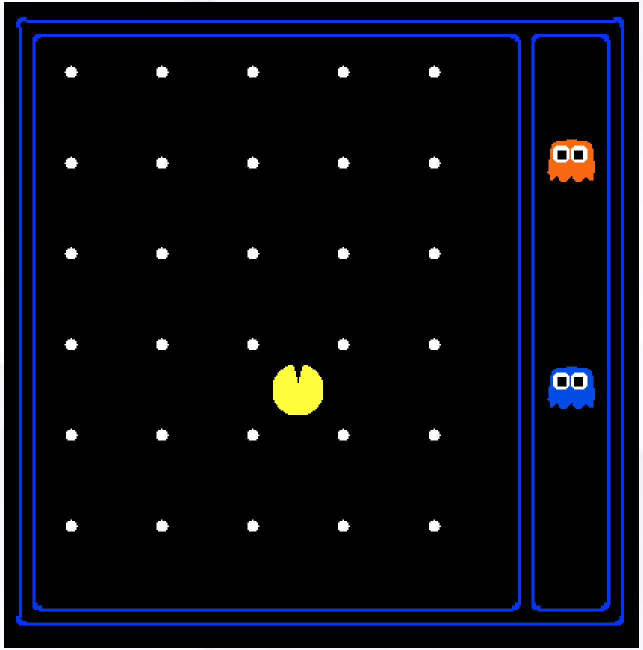


A search state keeps only the details needed for planning (abstraction)

- Problem: Path finding
  - 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: 12X10 = 120
  - Food count: 30
  - Ghost positions: 12
  - Agent facing: NSEW
- How many
  - World states?
    $120 x (2^{30}) x (12^2) x 4$
  - States for path finding?
    120
  - States for eat-all-dots?
    $120 x (2^{30})$



Some assumptions

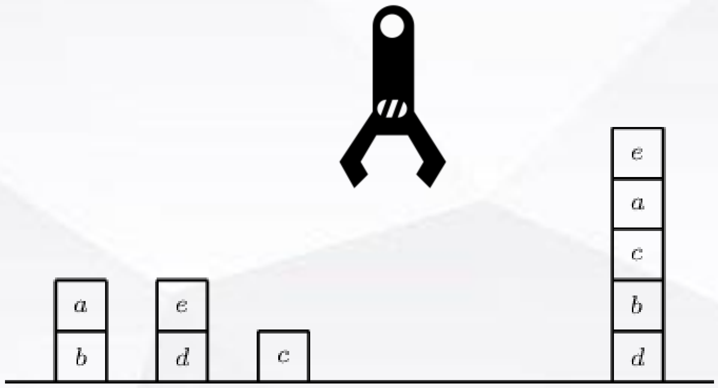
- Set of actions is **finite** and bounded by some constant
  - In practice, they are often parametrized and thus infinite (e.g., steering angle of a car)
  - In such settings we can **discretize** the set of actions
- State of agent is **observable**
  - i.e., agent knows its state
- Actions are **deterministic**
  - i.e., agent knows what will happen when it does something
- We will relax some of these assumptions along the course

State Space Search Problem : Classical Planning

- Formal definition:
  - a finite and discrete **state space** S
  - a known **initial state** $s_0 \in S$
  - a non-empty set of goal states $S_G \subseteq S$ (or a **goal test**)
  - set of **actions/operators** $A(s) \in A$ applicable in each state $s \in S$
  - A **successor** function – a **deterministic** state transition function $f(a, s)$ such that $s' = f(s, a)$ stands for the state resulting from applying an applicable action a at state s
  - A **cost function** $c(a, s)$ associating a **positive cost** to applying action a in s
- Objective
  - Find a plan, a sequence of applicable actions, that leads from the initial state to a goal state.
  - Optimality criteria - typically interested in shortest/ cheapest plan (but other considerations are possible)

Factored State Space Definition: Using Features

- Typically, it is not possible to explicitly specify the **state space** S
- Instead, **feature set** X is used to represent the state space such that a state is described via a set of random variables X= $x_1, \ldots, x_n$ such that each variable takes on values in some finite domain $Dom(x_i)$.
- Can be Boolean or multi-valued
- Example:
  - Position of block a
  - Position of block c
  - ...
  - Is gripper empty



Formalizing a search problem

Example: Missionaries and cannibals problem

- **Problem**:
  - 3 missionaries and 3 cannibals must cross a river using a boat that carries at most 2 people
  - If there are missionaries present on the bank, they cannot be outnumbered by cannibals (if they were, the cannibals would eat the missionaries)
  - The boat cannot cross the river by itself with no people on board
- How do we represent the search problem?

Example: Missionaries and cannibals problem

- **Take (1)**:
  - A state is a 7-tuple $< L_1, L_2, L_3, L_4, L_5, L_6, B >$
  - $L_i \in \{\text{Left}, \text{Right}, \text{Boat}\}$ location of person i
  - $B \in \{\text{Left}, \text{Crossing}, \text{Right}\}$ location of boat
  - Start state: $< \text{Right}, \text{Right}, \text{Right}, \text{Right}, \text{Right}, \text{Right}, \text{Right} >$
  - Goal state: $< \text{Left}, \text{Left}, \text{Left}, \text{Left}, \text{Left}, \text{Left}, \text{Left} >$
- How many states do we have?

"Hmm ... maybe I'll just have a salad."

Example: Missionaries and cannibals problem

- **Take (1)**:
  - A state is a 7-tuple $< L_1, L_2, L_3, L_4, L_5, L_6, B >$
  - $L_i \in \{\text{Left}, \text{Right}, \text{Boat}\}$ location of person i
  - $B \in \{\text{Left}, \text{Crossing}, \text{Right}\}$ location of boat
  - Start state: $< \text{Right}, \text{Right}, \text{Right}, \text{Right}, \text{Right}, \text{Right}, \text{Right} >$
  - Goal state: $< \text{Left}, \text{Left}, \text{Left}, \text{Left}, \text{Left}, \text{Left}, \text{Left} >$
- How many states do we have? $3^7 = 2187$
- Unnecessary information
  - We don't care **which** missionary / cannibal is on which side
  - We don't care about the state of the boat while crossing

"Hmm ... maybe I'll just have a salad."

Example: Missionaries and cannibals problem

- **Take (2)**:
  - A state is a 3-tuple $< M_L, M_C, B >$
  - $M_L \in \{0, \ldots, 3\}$ number of missionaries on left bank
  - $M_C \in \{0, \ldots, 3\}$ number of cannibals on left bank
  - $B \in \{\mathrm{Left}, \mathrm{Right}\}$ location of boat
  - Start state: $< 0, 0, \mathrm{Right} >$
  - Goal state: $< 3, 3, \mathrm{Left} >$
- How many states do we have?

"Hmm ... maybe I'll just have a salad."

Example: Missionaries and cannibals problem

- **Take (2)**:
  - A state is a 3-tuple $< M_L, M_C, B >$
  - $M_L \in \{0, \ldots, 3\}$ number of missionaries on left bank
  - $M_C \in \{0, \ldots, 3\}$ number of cannibals on left bank
  - $B \in \{\text{Left}, \text{Right}\}$ location of boat
  - Start state: $< 0, 0, \text{Right} >$
  - Goal state: $< 3, 3, \text{Left} >$
- How many states do we have? $4 \times 4 \times 2 = 32$

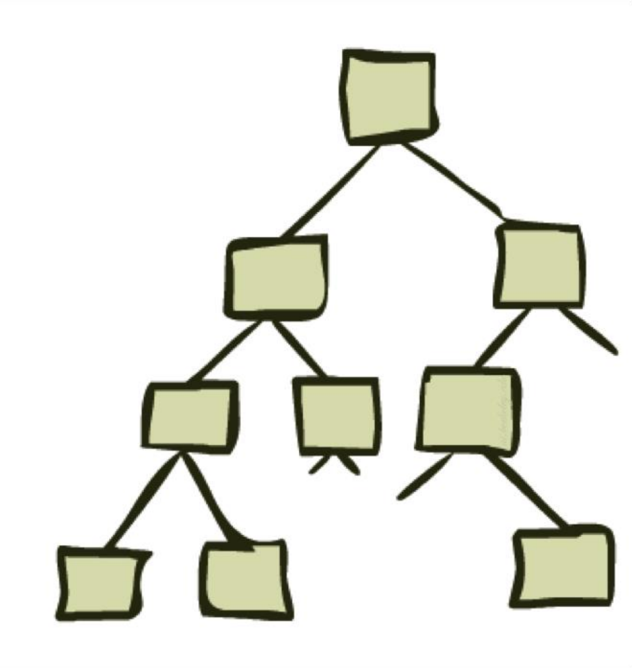
"Hmm ... maybe I'll just have a salad."

Search algorithms - preliminaries

TECHNION
Israel Institute of Technology

The Henry and Marilyn Taub
Faculty of Computer Science

State Space Graphs and Search Trees

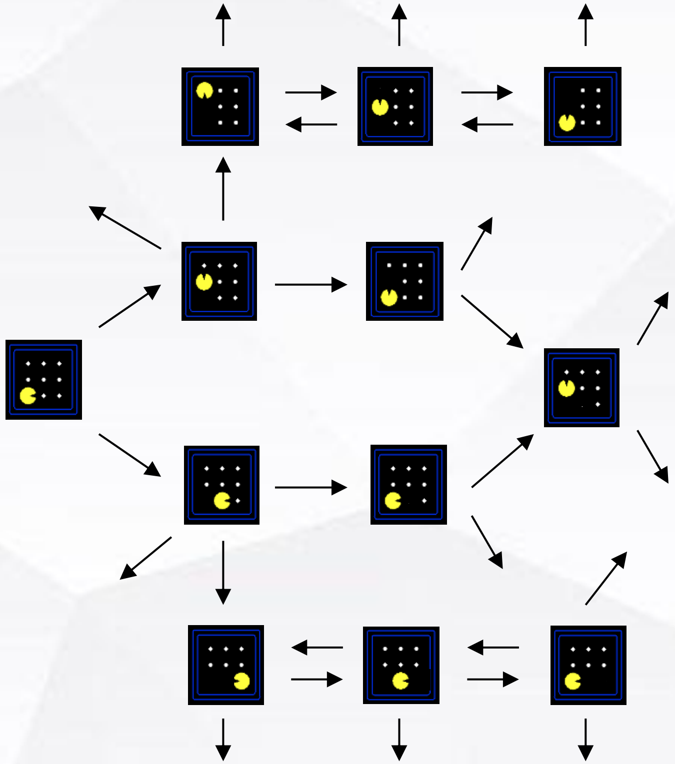


State-Space Graphs

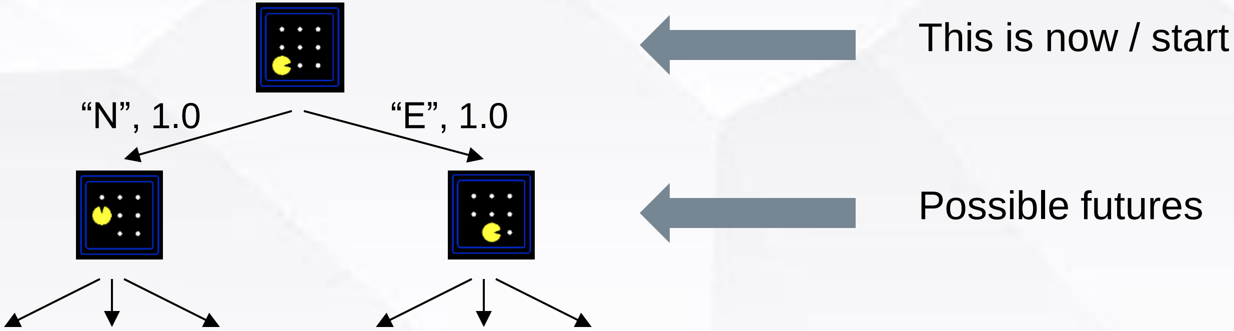
- State-space graph: A mathematical representation of a search problem
  - Nodes are (abstracted) world configurations
  - Arcs represent successors (action results)
  - The goal test is a set of goal nodes (maybe only one)

- 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
  - The graph may be **infinite**
  - The graph is given **implicitly**



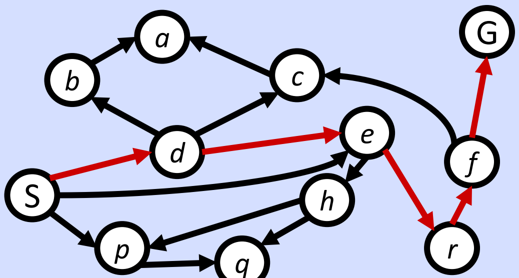
Search Trees



- 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

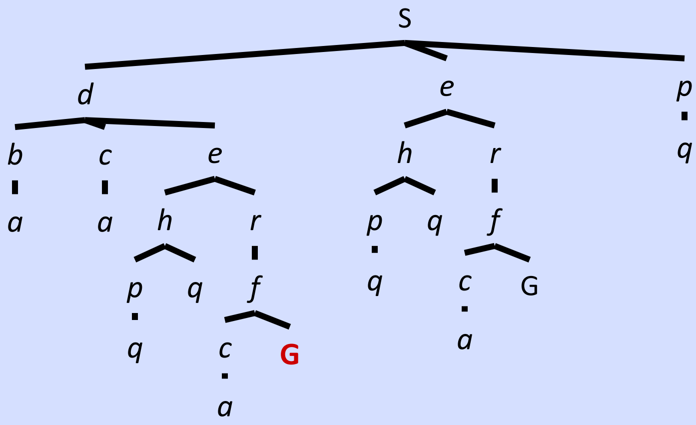
State Space Graph



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

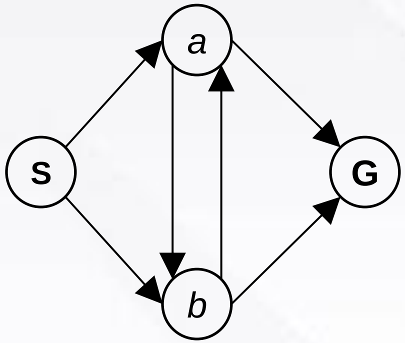
We construct both on demand – and we construct as little as possible.

Search Tree

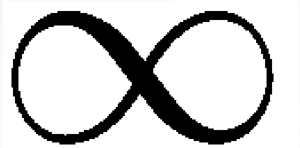


State Space Graphs vs. Search Trees

Consider this 4-state graph:

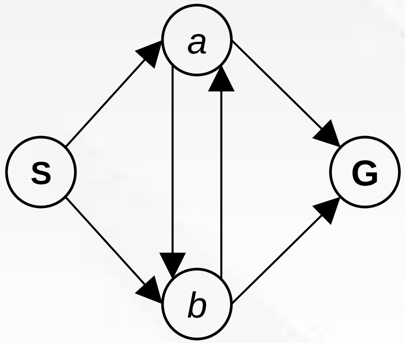


How big is its search tree (from S)?

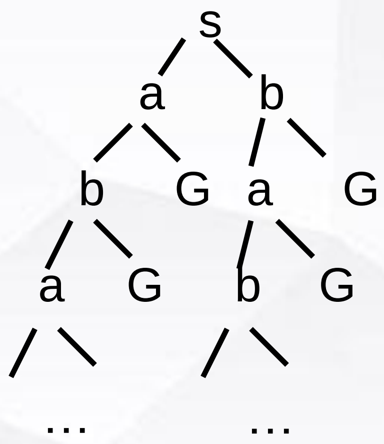


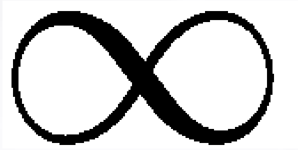
State Space Graphs vs. Search Trees

Consider this 4-state graph:



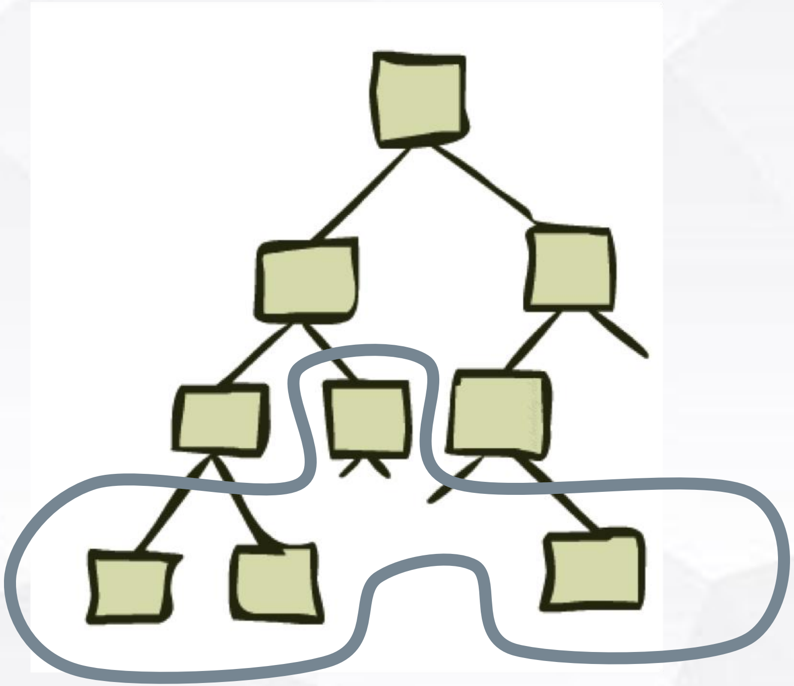
How big is its search tree (from S)?



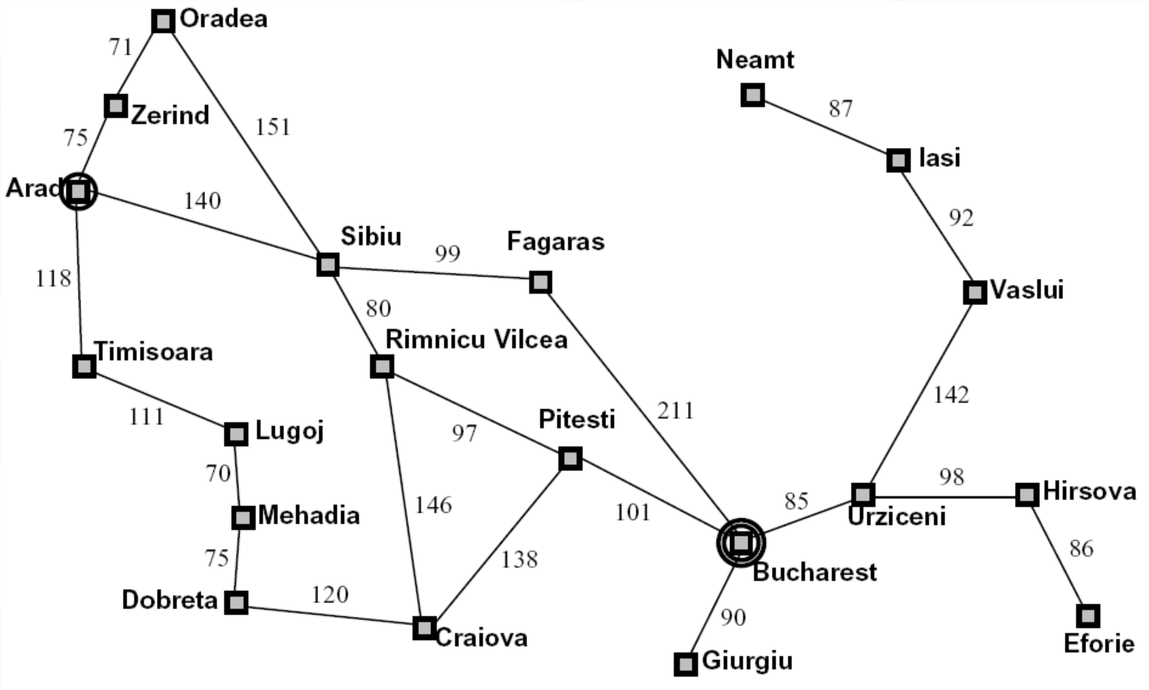


Important: Lots of repeated structure in the search tree!

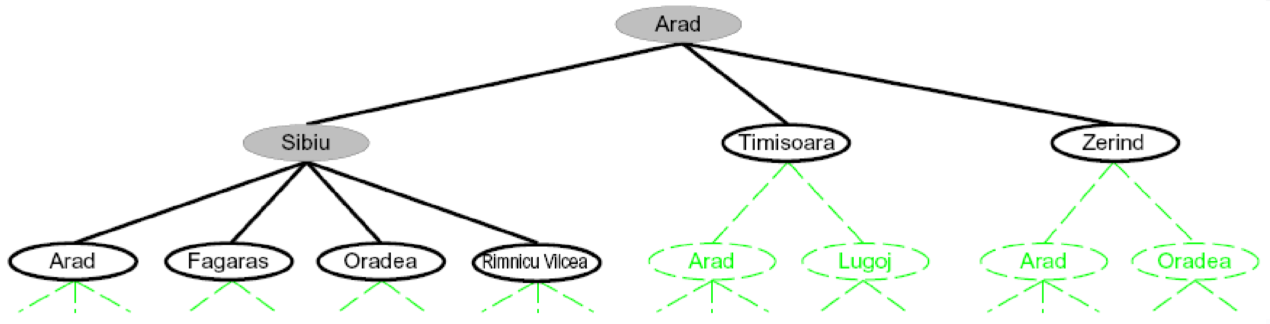
Tree Search



Search Example: Romania



Searching with a Search Tree



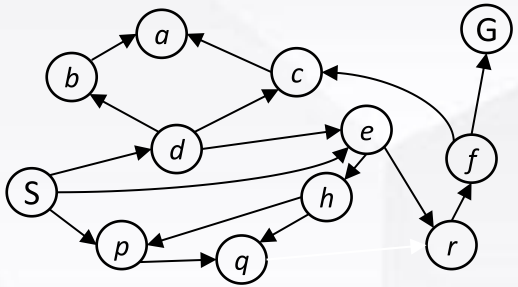
- Search:
  - Expand out potential plans (tree nodes)
  - Maintain a fringe of partial plans under consideration
  - Try to expand as few tree nodes as possible

General Tree Search

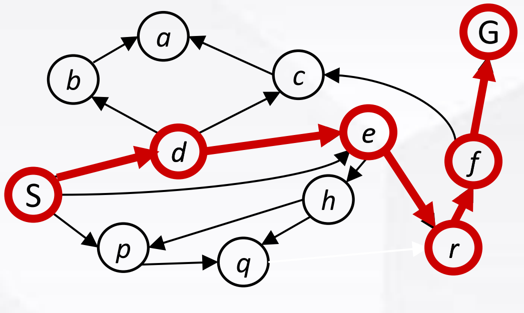
```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end
```

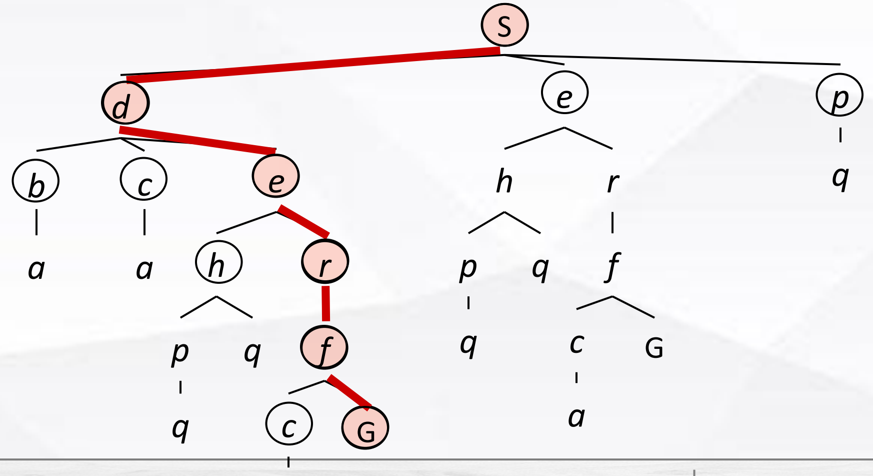
- Important ideas:
  - Fringe
  - Expansion
  - Exploration strategy
- Main question: which fringe nodes to explore?

Example: Tree Search



Example: Tree Search





~~s~~

~~s → d~~

s → e

s → p

s → d → b

s → d → c

~~s → d → e~~

s → d → e → h

~~s → d → e → r~~

~~s → d → e → r → f~~

s → d → e → r → f → c

~~s → d → e → r → f → G~~

Measuring performance of a search algorithm

- We care about two main (conflicting) **metrics**
  - Algorithm's **resources**: running time, memory consumption
  - **Quality** of solution: Optimal, bounded suboptimal, unbounded
- Typically, the more resources we invest, the higher quality the solution we can obtain

Algorithm's resources

- **Running time**
  - What we actually care about
  - Problematic metric – sensitive to hardware, compiler, optimization, implementation language etc.
  - Alternative measure: The number of **nodes expanded**

Algorithm’s resources

- **Running time**
  - What we actually care about
  - Problematic metric – sensitive to hardware, compiler, optimization, implementation language etc.
  - Alternative measure: The number of **nodes expanded**
- **Memory consumption**
  - **Critical resource** – may deem a search algorithm worthless
  - An algorithm that expands **1,000,000 nodes / second** and uses **10 bytes / node** can run out of memory in less than **2 minutes**

Quality of solution

- We will want to minimize the **cost** of a solution
- E.g., plan the path for a robot; action costs correspond to the robot's energy consumption; minimize total energy consumption

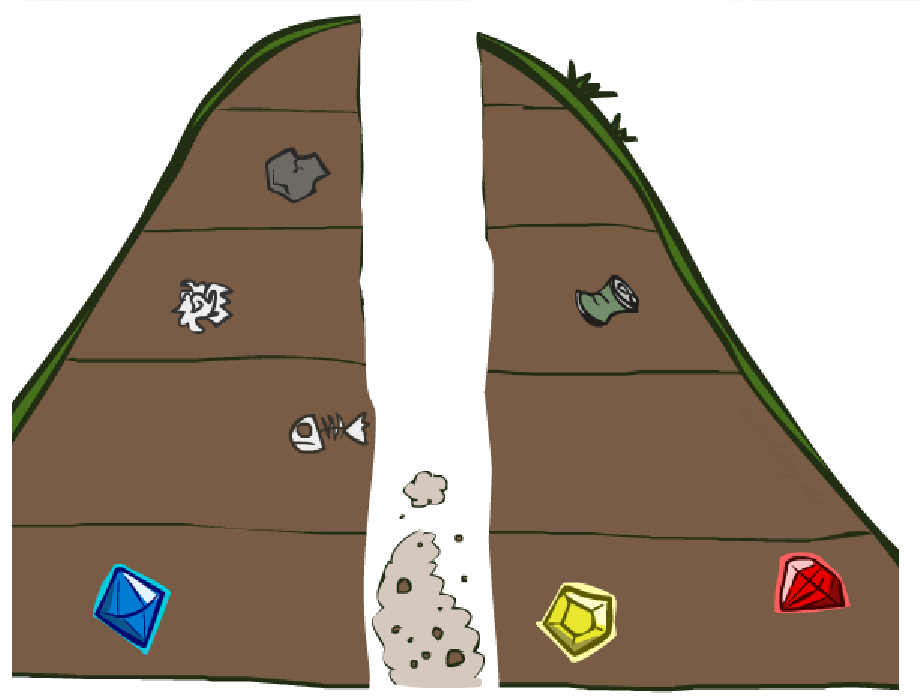
General Tree Search (recap)

```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end
```

- Important ideas:
  - Fringe
  - Expansion
  - Exploration strategy

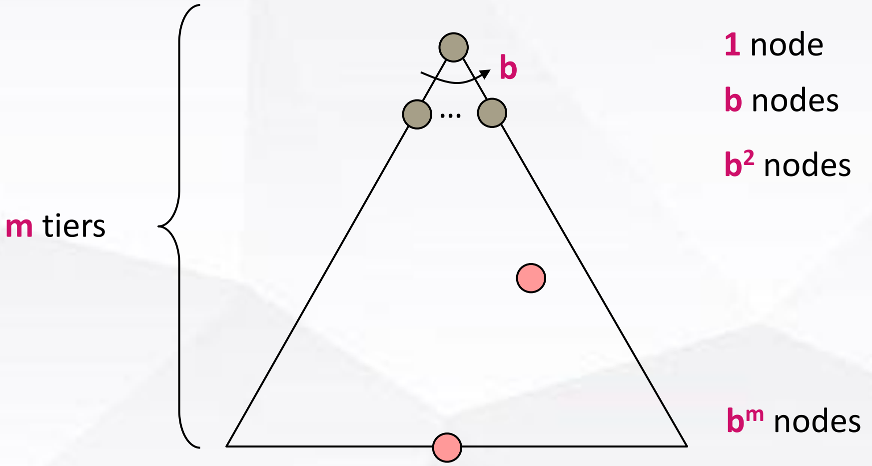
- Main question: which fringe nodes to explore?

Search Algorithm Properties



Search Algorithm Properties

- **Complete**: Guaranteed to find a solution if one exists?
- **Optimal**: Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?
- Cartoon of search tree:
  - **b** is the branching factor
  - **m** is the maximum depth
  - solutions at various depths



- Number of nodes in entire tree?
  - $1 + b + b^2 + \ldots + b^m = O(b^m)$

Breadth-First Search

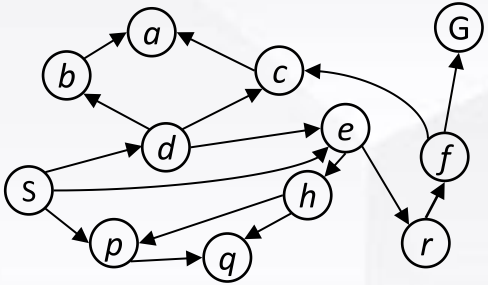
TECHNION
Israel Institute of Technology

The Henry and Marilyn Taub
Faculty of Computer Science

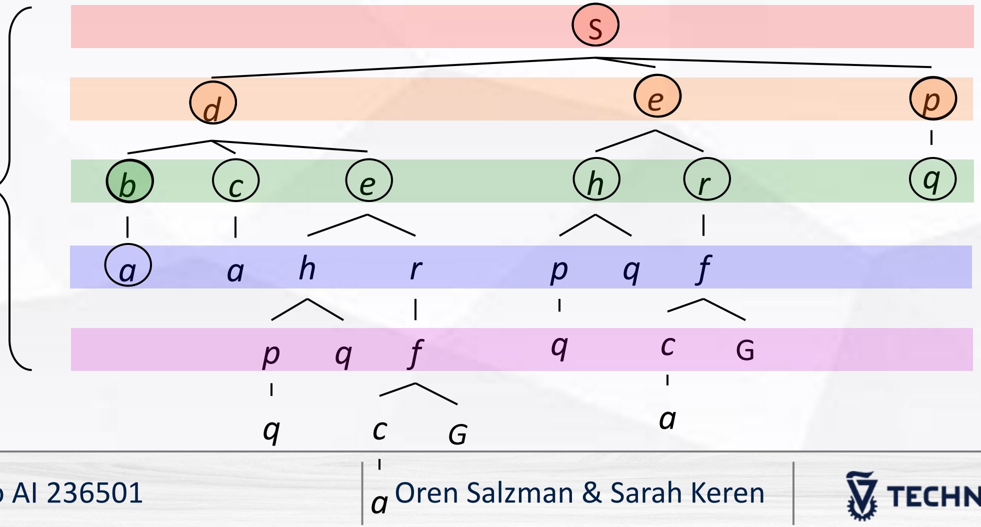
Breadth-First Search

Strategy: *expand a shallowest node first*

Implementation: *Fringe is a FIFO queue*

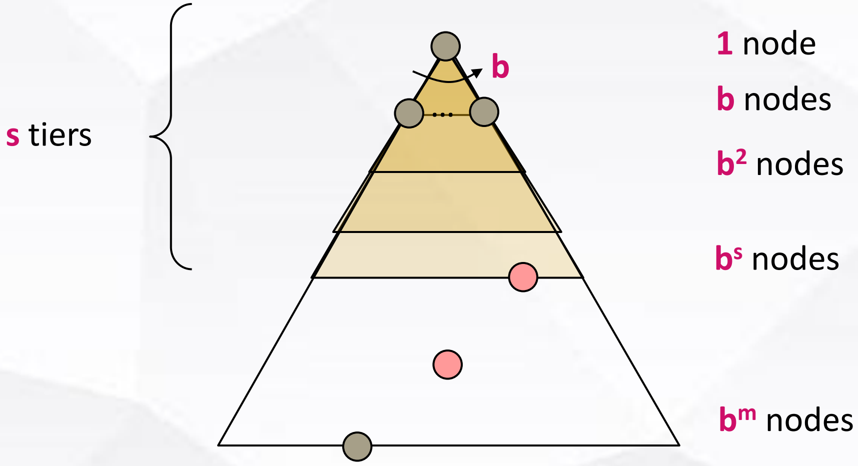


Search Tiers



Breadth-First Search (BFS) Properties

- **What nodes does BFS expand?**
  - Processes all nodes above shallowest solution
  - Let depth of shallowest solution be s
  - Search takes time **$O(b^s)$**



- **How much space does the fringe take?**
  - Has roughly the last tier, so **$O(b^s)$**
- **Is it complete?**
  - **s** must be finite if a solution exists
- **Is it optimal?**
  - Only if costs are all 1 (more on costs later)

BFS Memory requirements

- Assume that BFS expands **1,000,000 nodes / sec** for a problem with branching factor **b=10** and each node uses **1000 bytes node**

Depth	No. of nodes	Time	Memory
2	111	0.1 ms	11 KB
4	11,111	11ms	11 MB
6	~10^6	1 sec	~1 GB
8	~10^8	100 sec	~100 GB
10	~10^10	3 hours	~10 TB
12	~10^12	11 days	~1 PB
14	~10^14	3 years	~100 PB

Time requirements are reasonable

Memory requirements are not

Depth-First Search

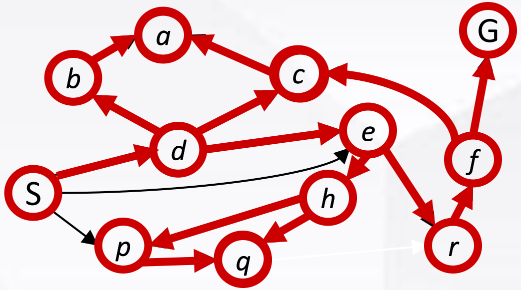
TECHNION
Israel Institute of Technology

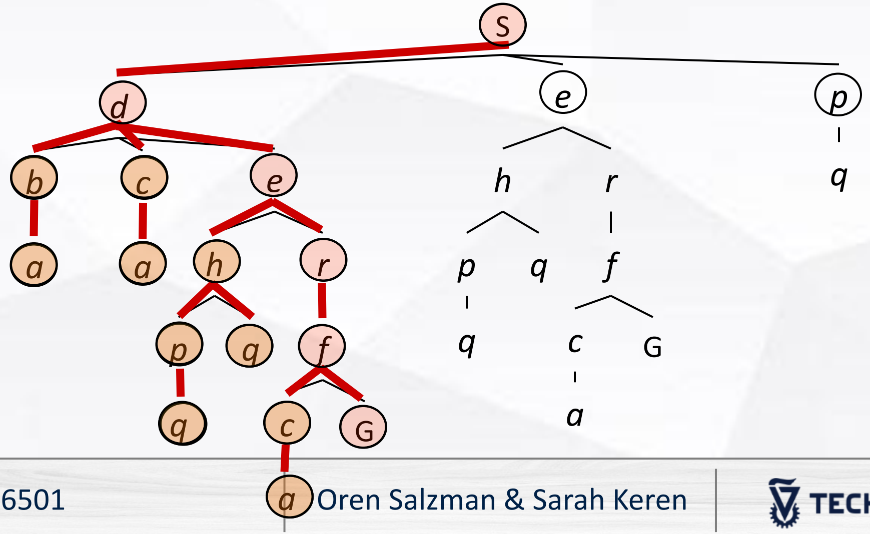
The Henry and Marilyn Taub
Faculty of Computer Science

Depth-First Search

Strategy: *expand a deepest node first*

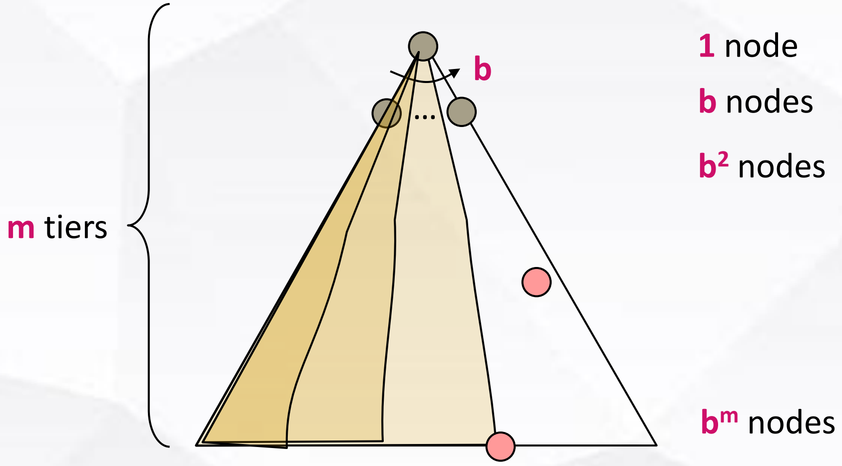
Implementation: *Fringe is a LIFO stack*





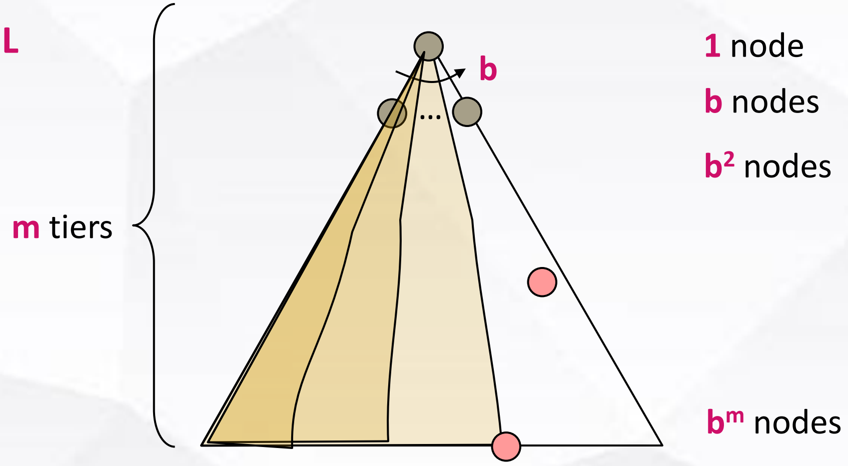
Depth-First Search (DFS) Properties

- What nodes DFS expand?
  - Some left prefix of the tree.
  - Could process the whole tree!
  - If **m** is finite, takes time **$O(b^m)$**
- **How much space does the fringe take?**
  - Only has siblings on path to root, so **$O(bm)$**
- **Is it complete?**
  - **m** could be infinite
  - To avoid this, we can call the algorithm with a bound **L** on the maximum depth – this is called **`DFS-L`**
- **Is it optimal?**
  - No, it finds the "leftmost" solution, regardless of depth or cost



Depth-First Search Variants

- **DFS-L**
  - Same as **DFS** but with a maximum tree depth **L**
- **DFS-L-BT**
  - Same as **DFS-L** but with the following modification:
  - At every node, we run an operator **O** over the same node
  - When we backtrack, we undo the previous operator
  - We only need to store one node in memory (with some additional book keeping)

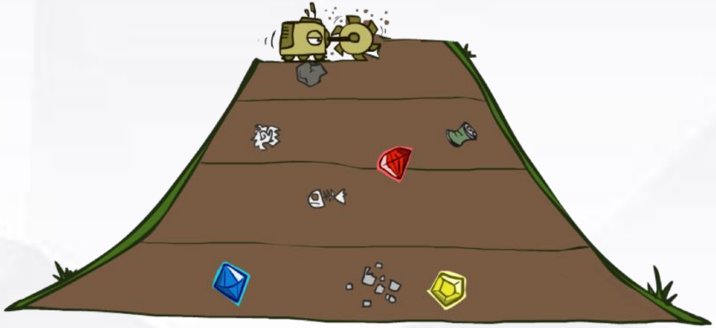


BFS vs. DFS

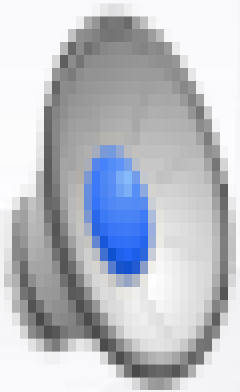
TECHNION
Israel Institute of Technology

The Henry and Marilyn Taub
Faculty of Computer Science

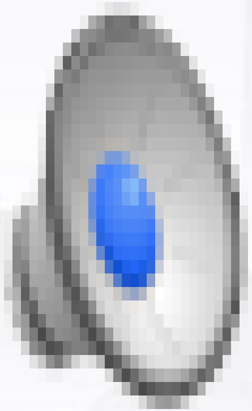
Quiz: DFS vs BFS



Video of Demo Maze Water DFS/BFS (part 1)



Video of Demo Maze Water DFS/BFS (part 2)



BFS vs. DFS (−L)

	BFS	DFS	DFS-L
Complete	YES	NO*	NO**
Optimal#	YES	NO	NO
Memory	Exponential in solution length	Infinite	Linear in L

BFS vs. DFS (-L)

	BFS	DFS	DFS-L
Complete	YES	NO*	NO**
Optimal#	YES	NO	NO
Memory	Exponential in solution length	Infinite	Linear in L

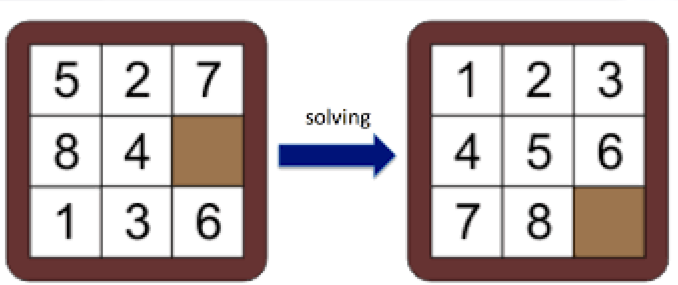
* If tree is unbounded, can run infinitely even if a solution exists

** If **L** is less than the solution depth, no solution is returned even though one exists

Assuming uniform cost edges

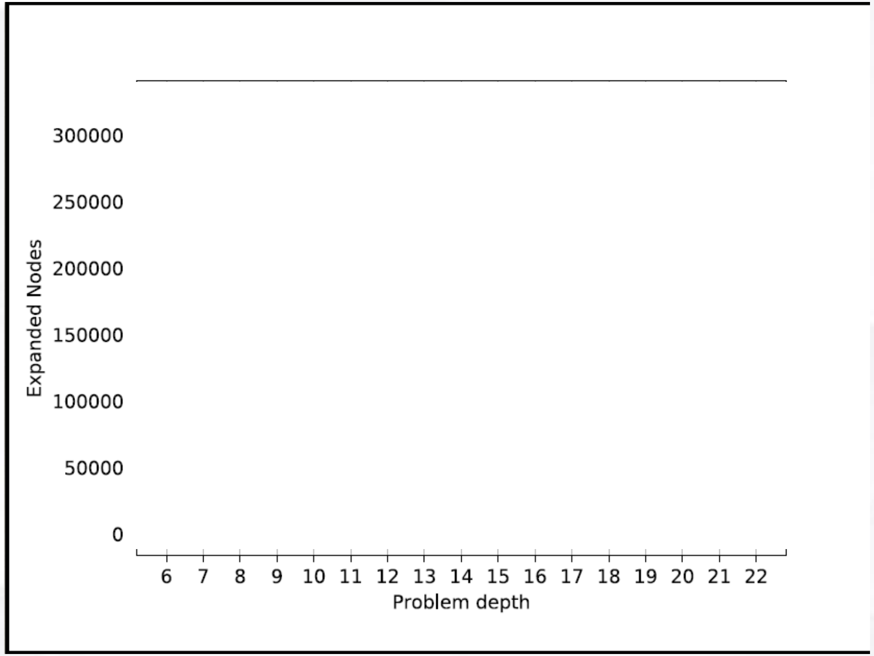
Empirical demonstration

- Python implementation by Shaul Markovitz of `BFS`, `DFS-L` and `DFS-L-BT`
- Random **3X3** puzzles whose solution-lengths range between **6** and **22** (**20** problem instances per solution length)
- `DFS-L` was run with **L=22** (for all instances)



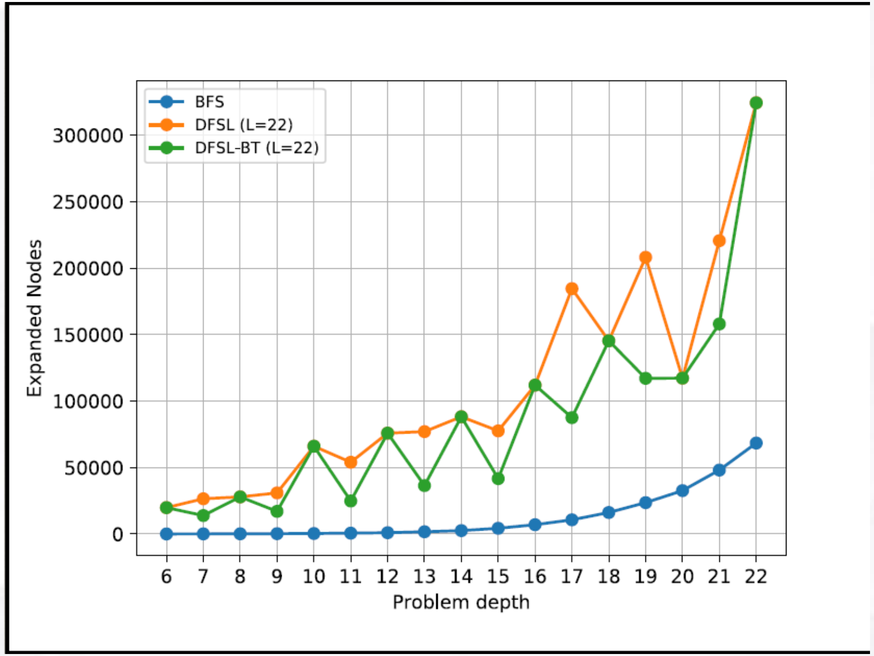
Empirical demonstration

- **Expanded nodes** as a function of **problem depth**

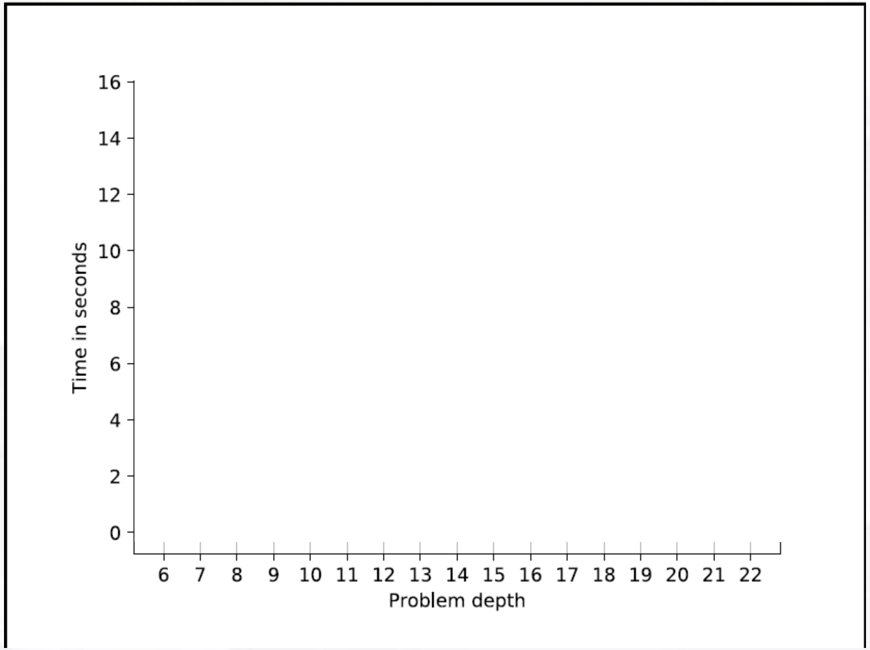


Empirical demonstration

- **Expanded nodes** as a function of **problem depth**

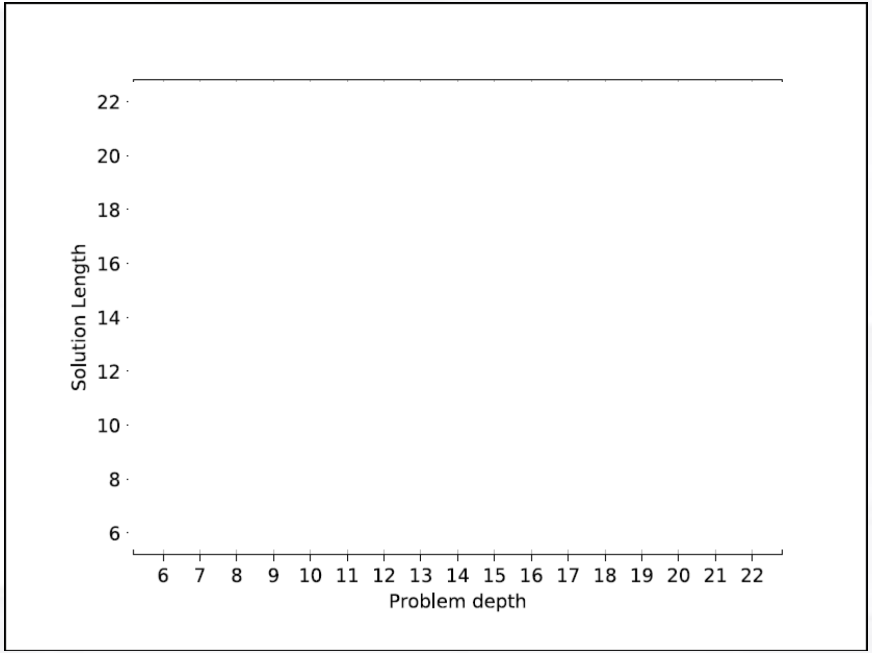


- **Time** as a function of **problem depth**



Empirical demonstration

- **Solution length** as a function of **problem depth**



Empirical demonstration

- **Memory footprint** as a function of **problem depth**