NUS | Computing

National University of Singapore

CS3243 Tutorial 1

Agents, Problems and Uninformed Search

Gu Zhenhao

January 25, 2023

About Me

Hello, I am Gu Zhenhao (Gary)!

- Year 2, MComp (CS) candidate,
- **Interests**: Algorithms & Theory, Computational Biology, open-world/MOBA games,
- **E-mail**: guzh@nus.edu.sg
- **Telegram**: @garygzh
- **GitHub**: GZHoffie

About this Tutorial

- **Time**: 10:00 – 11:00 A.M. every Wednesday.
- **Venue**: COM3-01-25.
- **Content**: Review key concepts in lectures and discuss problem-solving recipes.

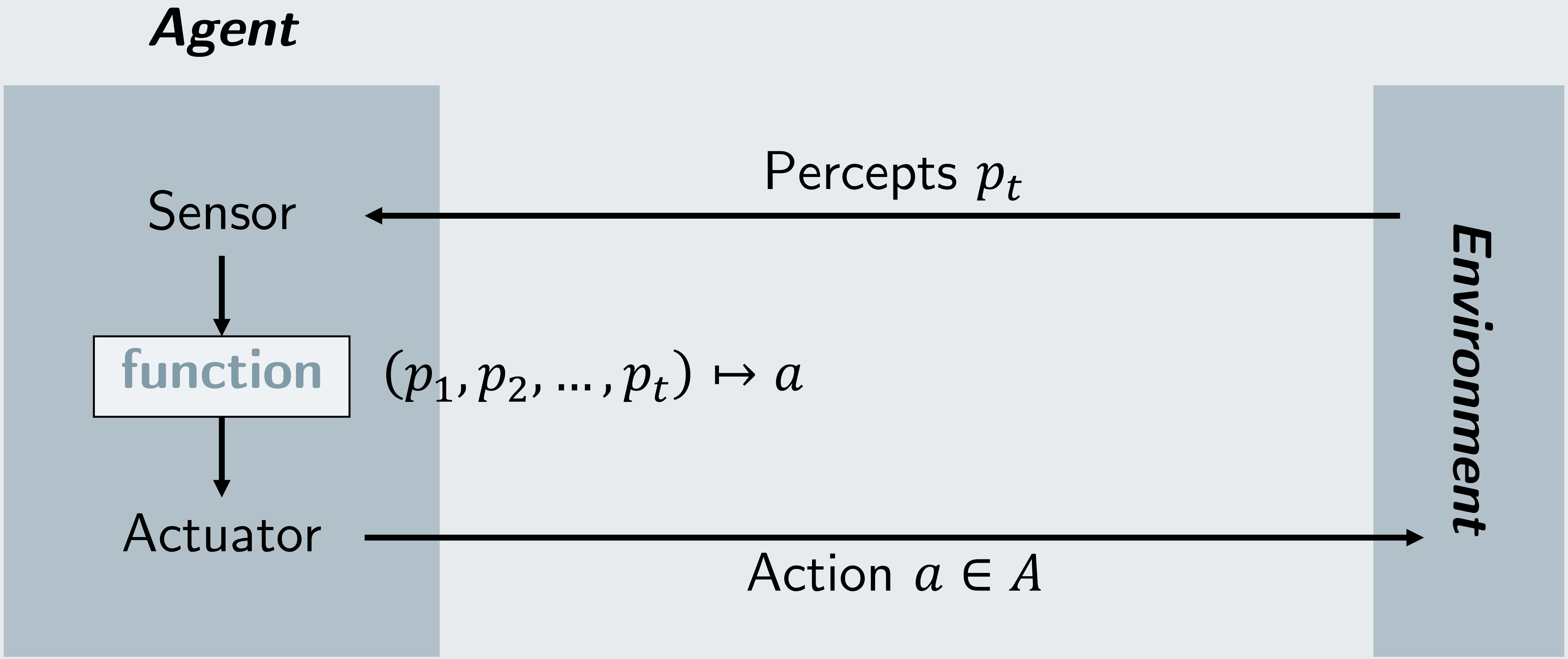
About this Tutorial

- **Slides**: will be uploaded to Telegram group and my repository.
- **Notes**:
  1. Attendance & participation will be taken. (5% of course grade)
  2. Interaction in class and discussion in telegram groups are all counted as participation.
  3. Hand in tutorial assignment (in paper), and attempt all questions before class.

Intelligent Agents

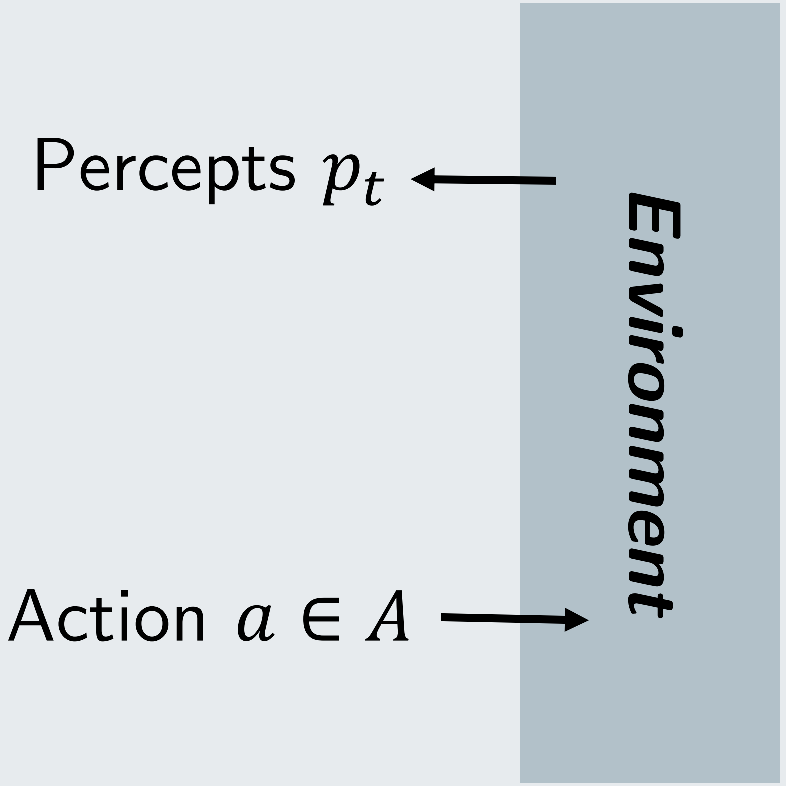
What does a general AI model look like?

Agent-Environment Interaction



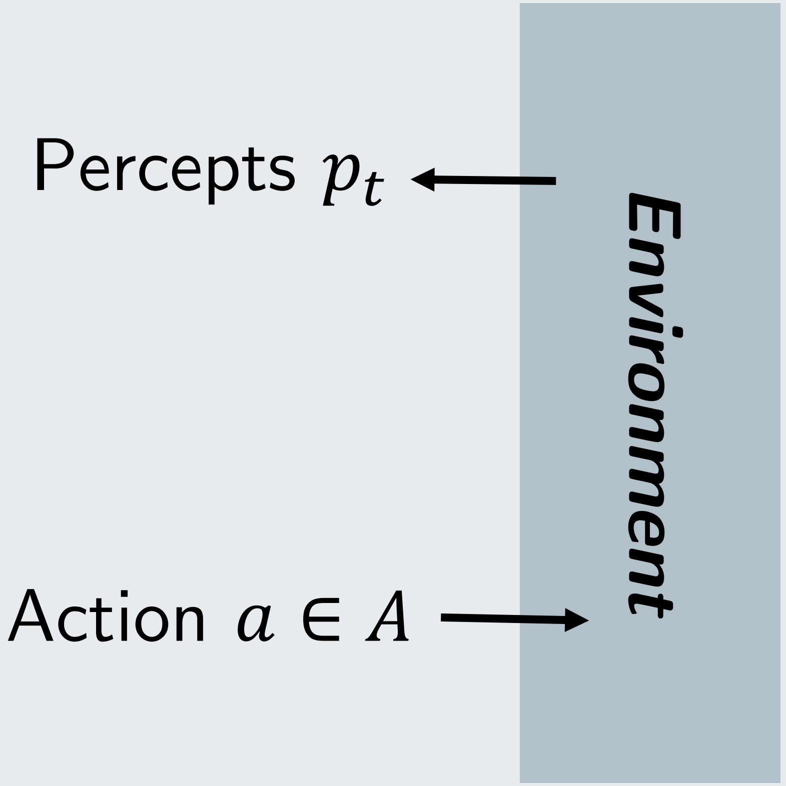
Types of Environment

- **Fully observable / Partially observable.**
  Are there hidden features in the state?



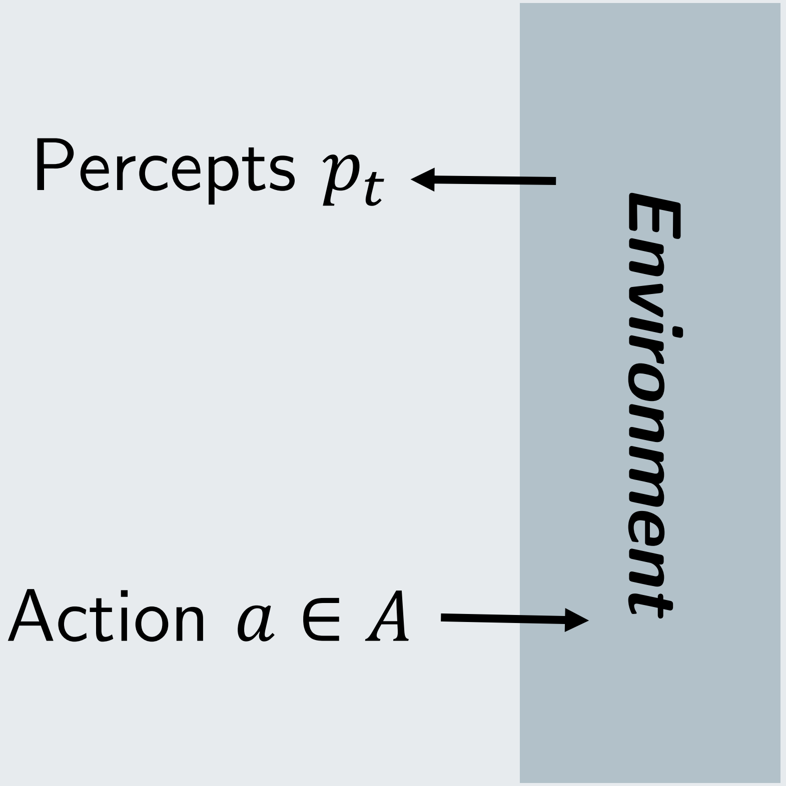
Types of Environment

- Fully observable / Partially observable.
- **Deterministic / Stochastic.**
  Are there any randomness involved?



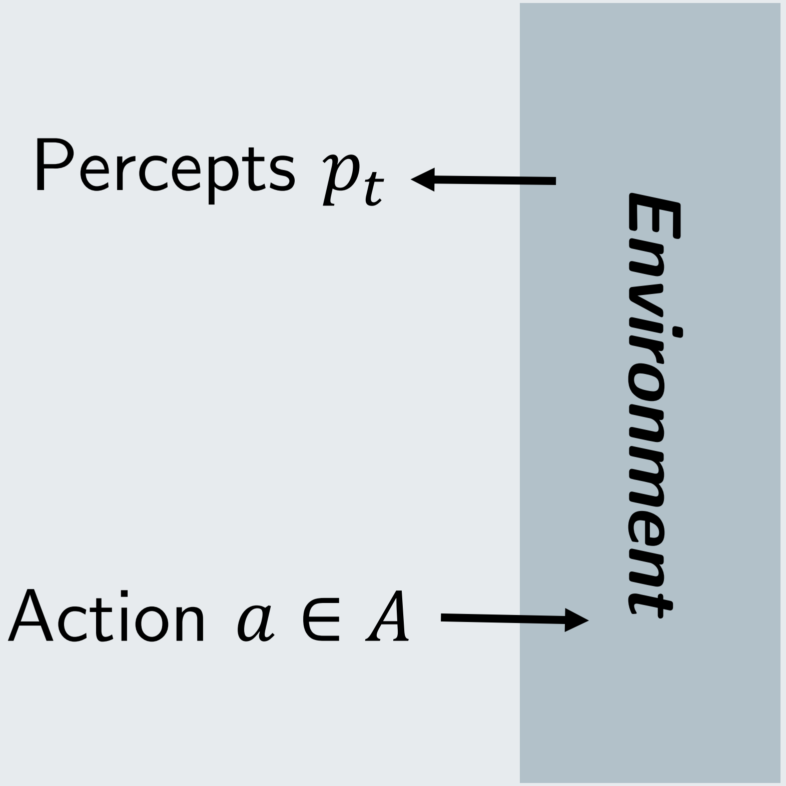
Types of Environment

- Fully observable / Partially observable.
- Deterministic / Stochastic.
- **Episodic / Sequential.**
  Will the current action affect future decisions?



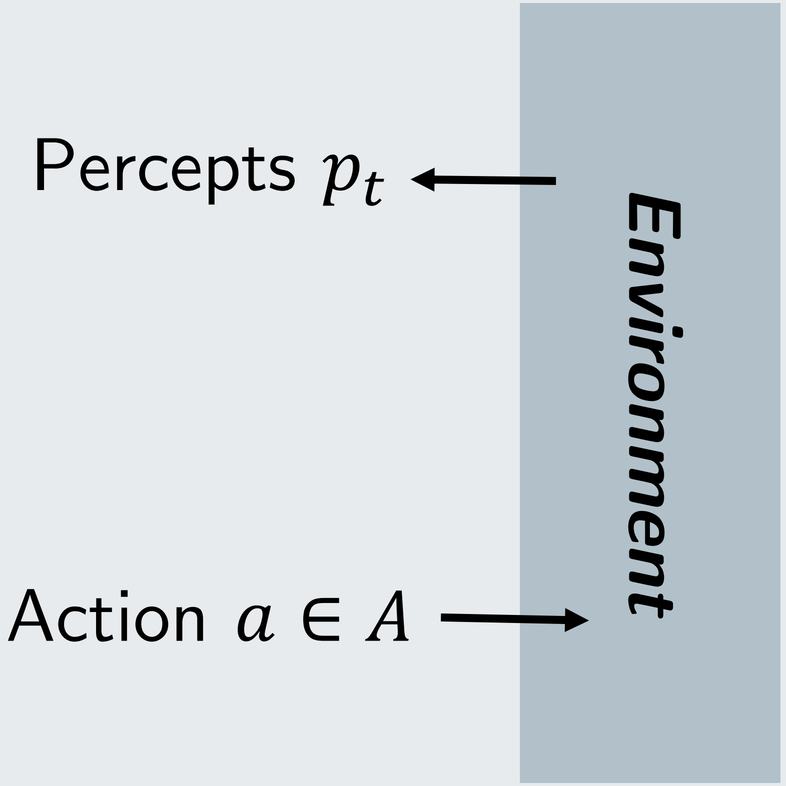
Types of Environment

- **Fully observable / Partially observable.**
- **Deterministic / Stochastic.**
- **Episodic / Sequential.**
- **Discrete / Continuous.**
  Are state, time, percepts and actions measured using discrete variables?



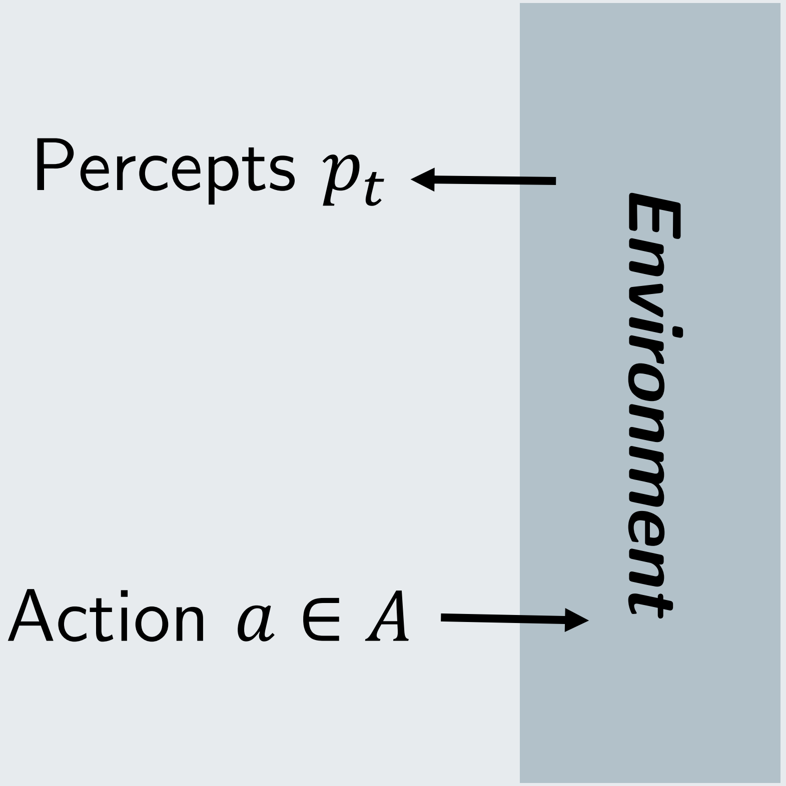
Types of Environment

- Fully observable / Partially observable.
- Deterministic / Stochastic.
- Episodic / Sequential.
- Discrete / Continuous.
- **Single-agent / Multi-agent.**
  Are there multiple entities that can influence the environment?



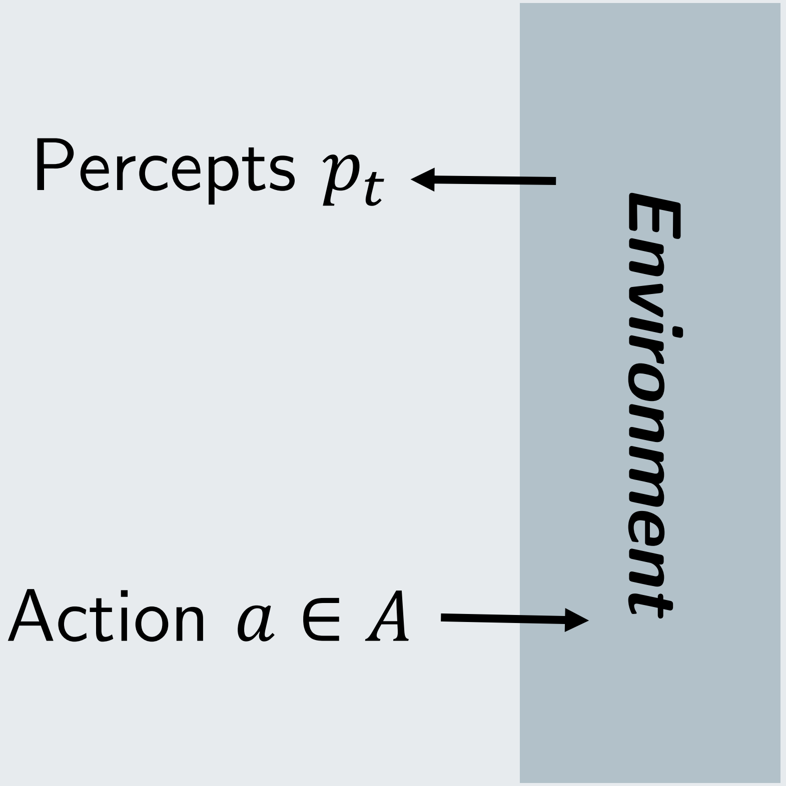
Types of Environment

- Fully observable / Partially observable.
- Deterministic / Stochastic.
- Episodic / Sequential.
- Discrete / Continuous.
- Single-agent / Multi-agent.
- **Static / Dynamic.**
  Can the environment change when agent is deciding on the action?



Types of Environment

- Fully observable / Partially observable.
- Deterministic / Stochastic.
- Episodic / Sequential.
- Discrete / Continuous.
- Single-agent / Multi-agent.
- Static / Dynamic.
- **Known / Unknown.**
  Does the agent know the rules of the environment?



Problem 1.a

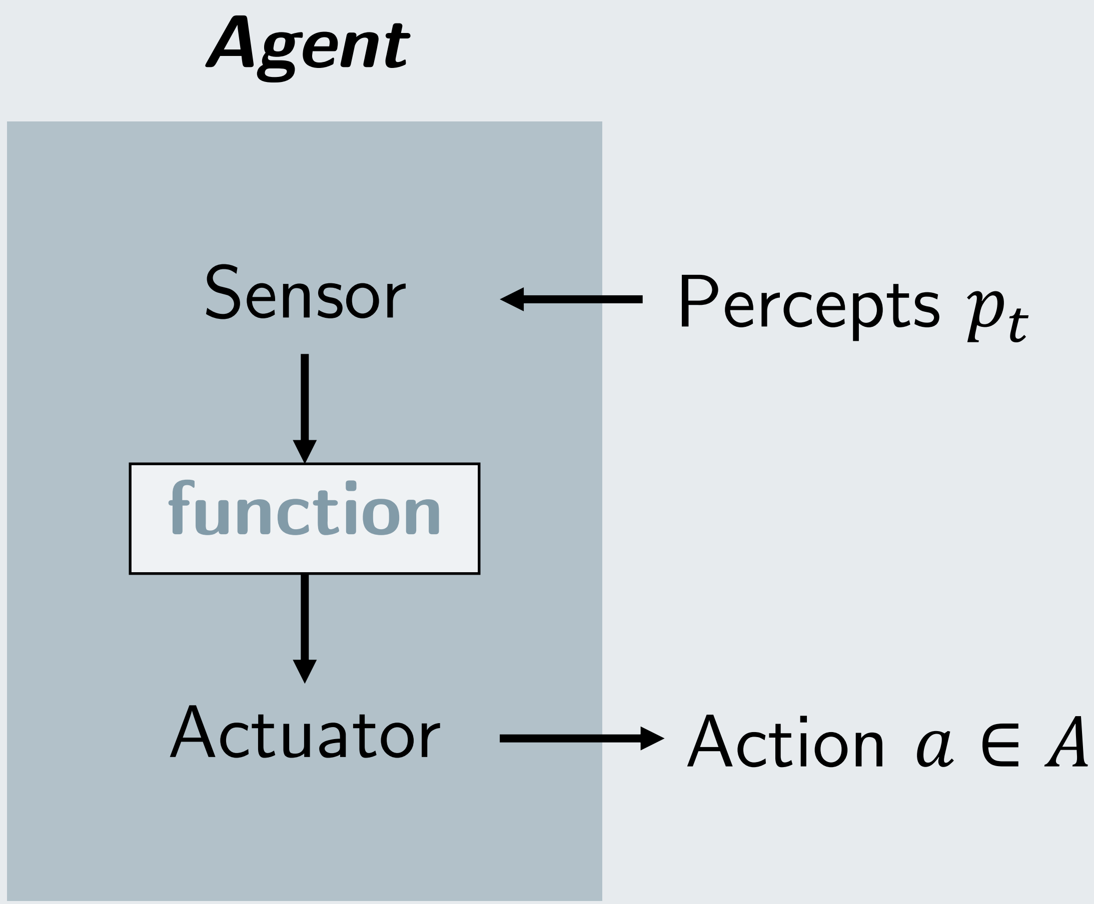
2	5			3		9		1
	1				4			
4		7				2		8
		5	2					
				9	8	1		
	4				3			
			3	6			7	2
	7							3
9		3				6		4

Figure. A Sudoku Puzzle.

- Fully observable / Partially observable.
- Deterministic / Stochastic.
- Episodic / Sequential. *
- Discrete / Continuous.
- Single-agent / Multi-agent.
- Static / Dynamic.
- Known / Unknown.

** Might be sequential for human players while episodic for computers.*

Types of Agent



- **Simple reflex agents.**
  Choose action a solely based on p_t.
- **Model-based reflex agents.**
  Simulate on a model of environment.
- **Goal-based agents.**
  Search and plan with a goal in mind.
- **Utility-based agents.**
  chooses the action that maximizes the expected utility of outcomes.

Searching

How to solve a problem using graph search algorithms?

Building Search Problems

- **Assumptions**: fully-observable, discrete, known, deterministic environment.
- **Steps**:
  1. **What should a node contain?**

     Determine state representation s_i, and all possible actions $A = \{a_1, \dots a_k\}$.
  2. **How to find a node's neighbours?**

     Determine the environment's transition model $T, \left(s_i, a_j\right) \mapsto s_i'$.
  3. **What are the cost for edges?**

     Define the $\mathtt{cost}(s_i, a_j, s_i')$ function that returns cost of an action a_j.
  4. **What are our goals?**

     Define the $\mathtt{isGoal}(s_i)$ function to check if a state s_i is our goal.

Problem 1.b

$a = (1,3)$

2	5	~~8~~	7	3	6	9	4	1
6	1	9	8	2	4	3	5	7
4	3	7	9	1	5	2	6	8
3	9	5	2	7	1	4	8	6
7	6	2	4	9	8	1	3	5
8	4	1	6	5	3	7	2	9
1	8	4	3	6	9	5	7	2
5	7	6	1	4	2	8	9	3
9	2	3	5	8	7	6	1	4

Figure. A Complete Sudoku Puzzle.

- **States**:
  A: matrix of size 9×9, each slot containing either null or one number in [1, 9].
  - **Initial state**: the complete, filled matrix.
  - **goal state**: any state (preferably still have unique solution and sufficiently sparse).
- **Actions**:
  (i, j): Replace a (non-empty) slot $A_{i,j}$ with null.
- **Transition model**:
  $T\big(A, (i, j)\big)$ = matrix A with slot (i, j) cleared.

Problem 4

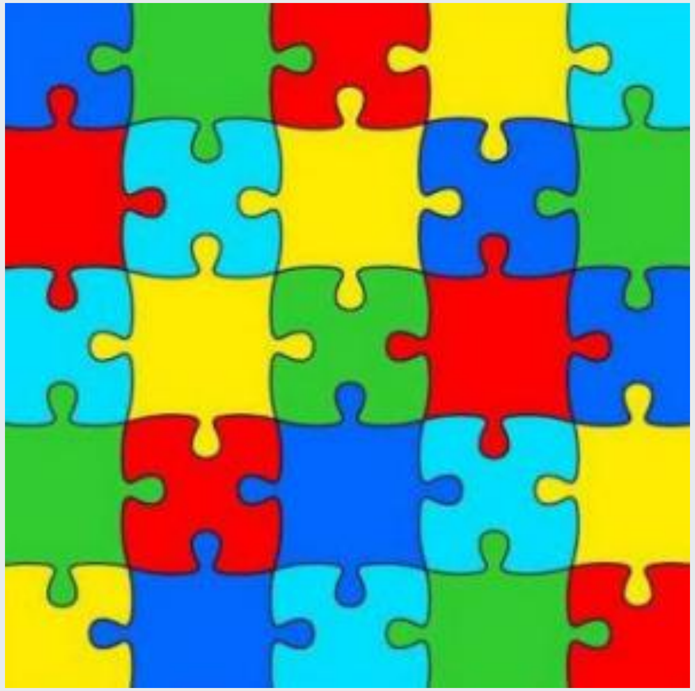


Figure. A Jigsaw Puzzle.

Many right answers... here's one.

- **States**: a graph showing connection of pieces.
  - **Initial state**: no connections.
  - **goal state**: all pieces are connected to the right number of pieces.
- **Actions**: (legally) connecting two pieces.
- **Transition model**:
  - returns the new graph with the added edges.
  - **step cost**: 1.

Pause and Ponder

2	5			3		9		1
	1				4			
4		7				2		8
		5	2					
				9	8	1		
	4				3			
			3	6			7	2
	7							3
9		3				6		4

Figure. A Sudoku Puzzle.

Question: could you define the state, action and transaction model for the problem of *solving a sudoku puzzle*?

Bonus: can you minimize branching factor and maximum depth by your definition?

** Feel free to discuss on the Telegram group!*

Solving Search Problems

```
traversal(G, s):
    F.push(s);
    while F is not empty do
        current = F.pop();
        visit(current);
        for u in current.neighbors do
            if not u.visited then
                F.push(u);
```

For DFS: F is a stack
For BFS: F is a queue
For UCS: F is a priority queue

For IDS/DLS: also check u.*depth* before inserting to F.

Solving Search Problems

```
traversal(G, s):
    F.push(s);
    while F is not empty do
        current = F.pop();
        visit(current);
        for a in actions(current.state) do
            u = Node(T(current.state, a), current)
            if not u.visited then
                F.push(u);
```

Neighbors **constructed** by transition model T.

Question: why do we use transition model instead of an adjacency matrix/list?

Solving Search Problems

```
traversal(G, s):
    F.push(s);
    while F is not empty do
        current = F.pop();
        if isGoal(current.state) then return current.getPath()
        for a in actions(current.state) do
            u = Node(T(current.state, a), current)
            if not u.visited then
                F.push(u);
```

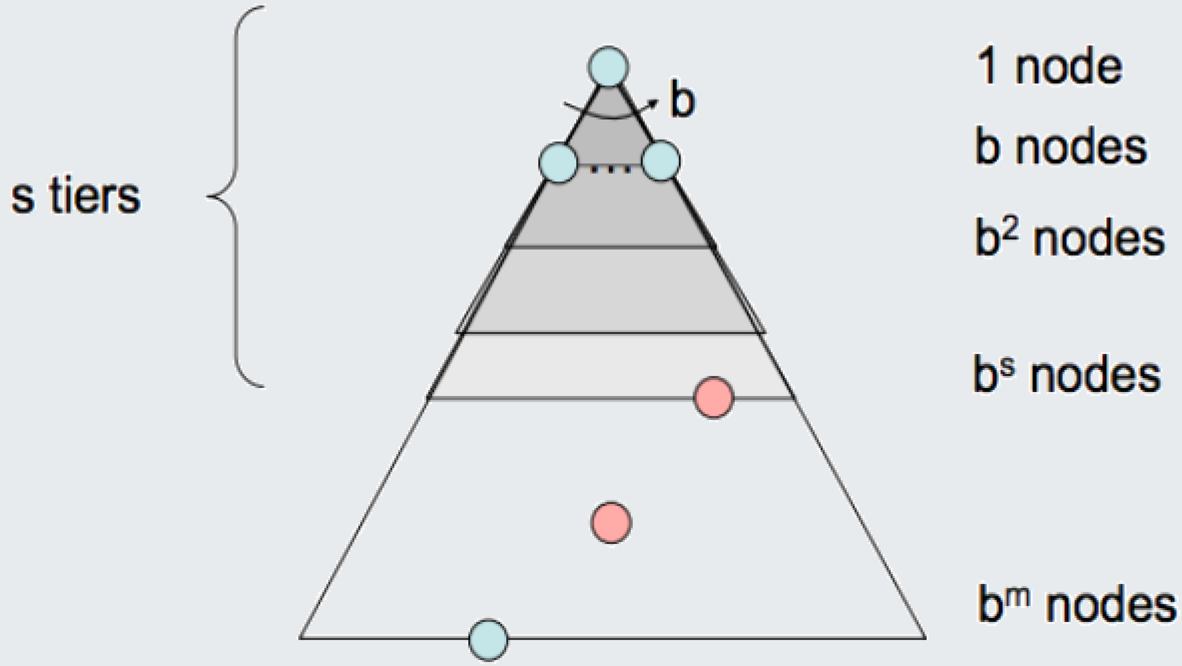
Terminate when we reach the goal.

Optimizations

```
traversal(G, s):
    F.push(s); reached = {s};
    while F is not empty do
        current = F.pop();
        if isGoal(current.state) then return current.getPath()
        for a in actions(current.state) do
            u = Node(T(current.state, a), current)
            if u not in reached then
                F.push(u); reached.insert(u);
```

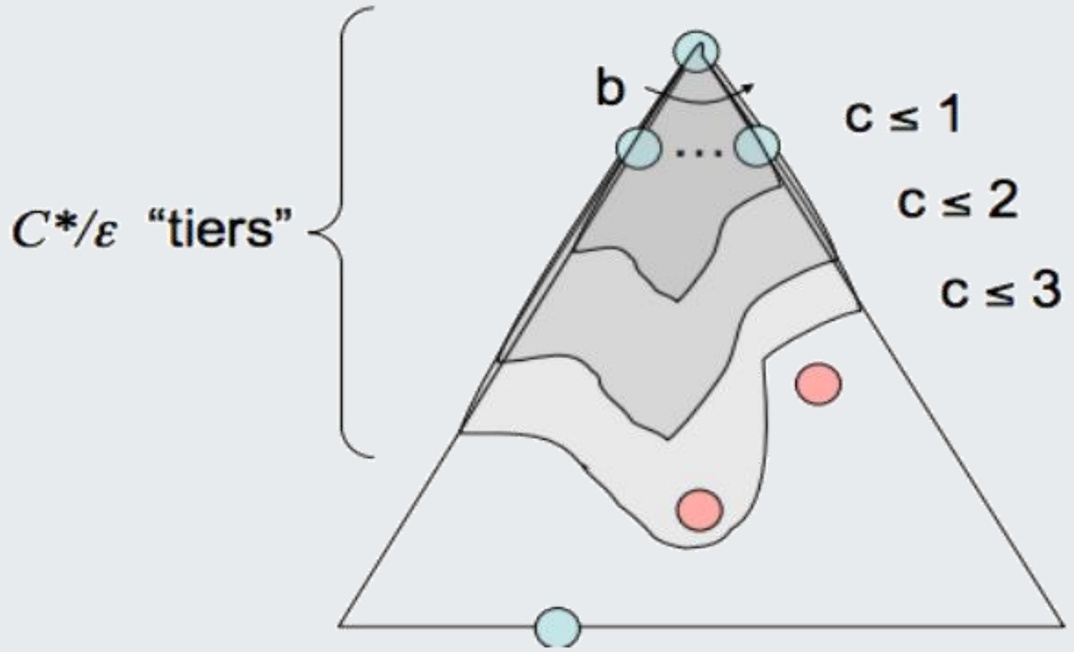
Optimizations: Tracking visited nodes with hash tables; early goal check; ...

Breadth-First Search



- **Intuition**: finds goal with smallest number of actions.
- **Pros**:
  - always finds a solution as long as it exists.
- **Cons**:
  - fails for trees with infinite branches.
  - doesn't take cost into account.
  - very time & memory-consuming.

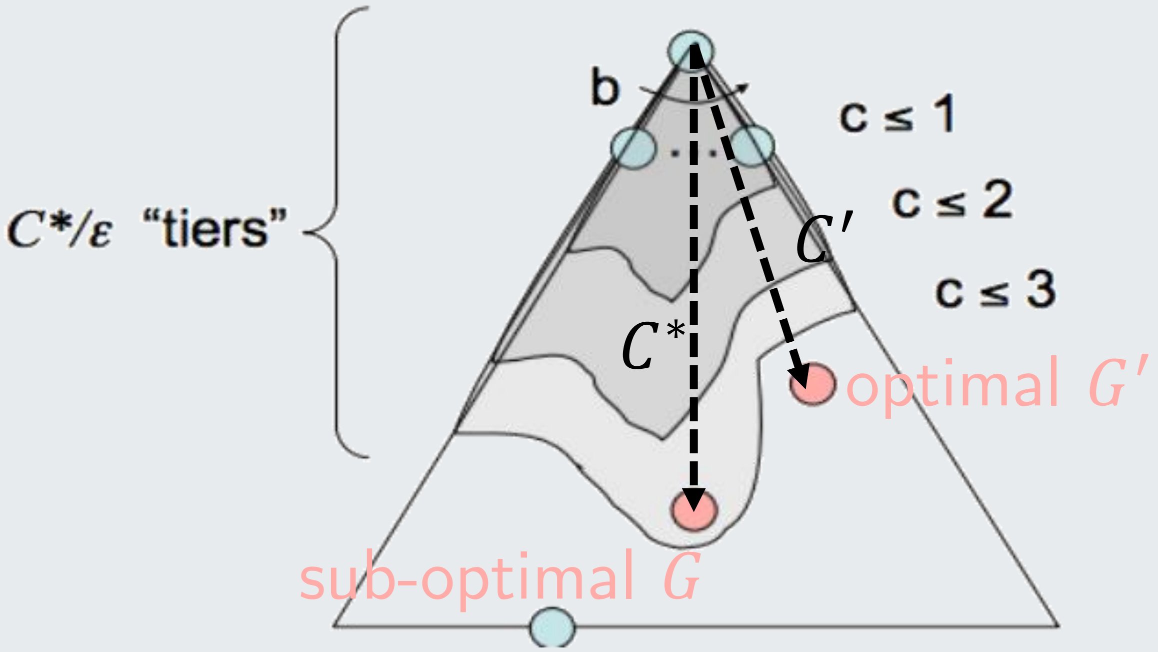
Uniform Cost Search



- **Intuition**: finds goal with smallest total action cost.
- **Pros**:
  - complete, and optimal for non-negative costs.
- **Cons**:
  - might also be memory-consuming.

Question: is UCS still complete/optimal given negative/zero costs?

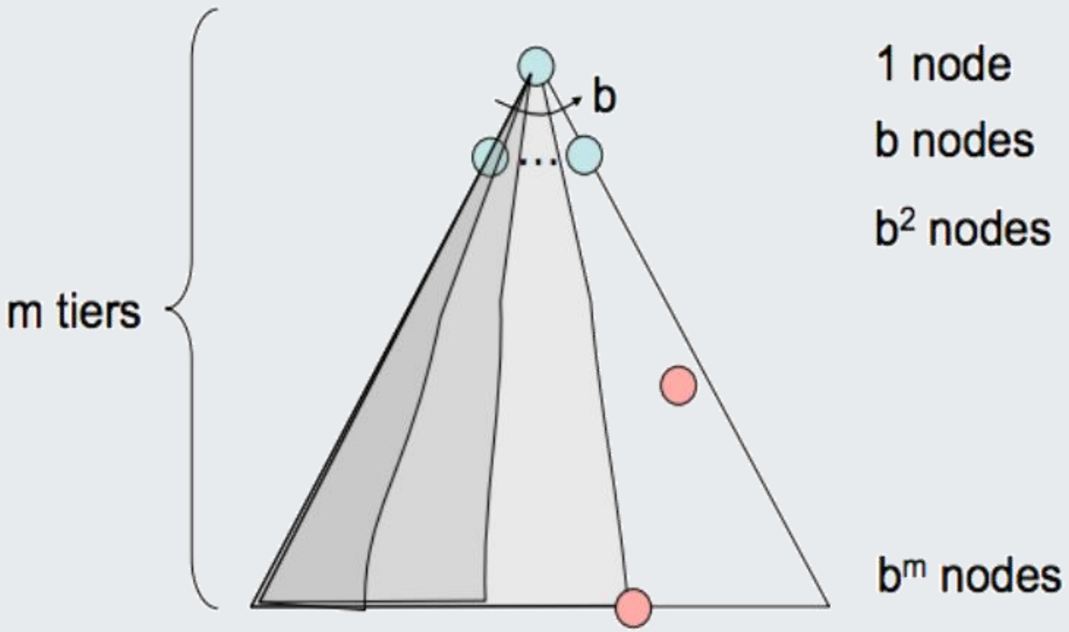
Problem 3



- **Goal**: prove optimality of UCS.
- **Idea**: by the greedy property of UCS.
  - UCS always explore nodes with smallest path cost first.
  - When adding a node at the end of path, the cost of path never decreases ($\varepsilon > 0$).

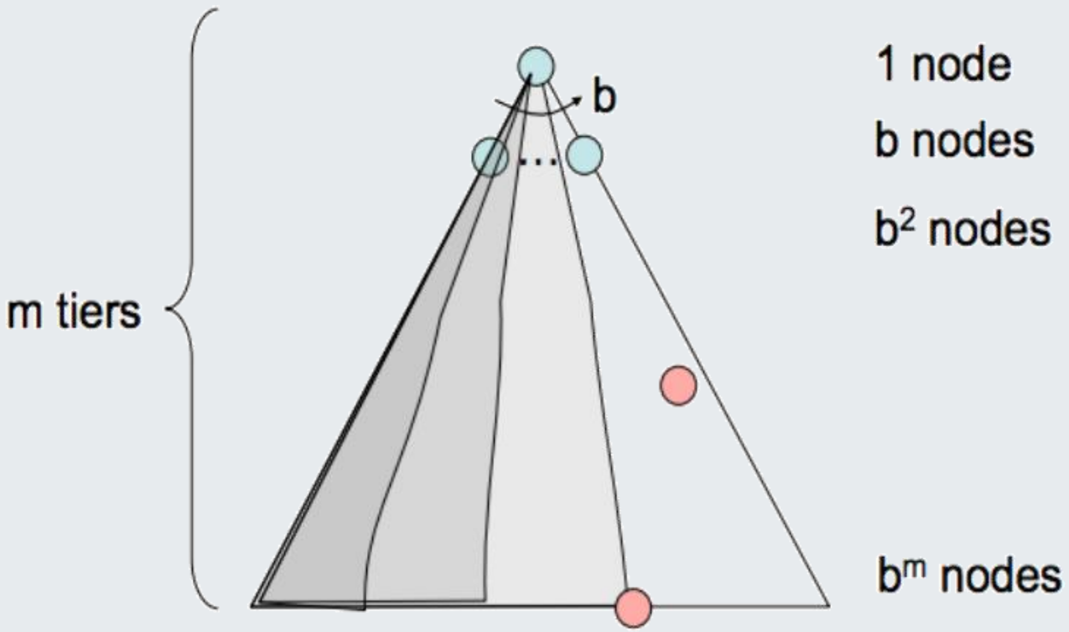
$\Rightarrow$ Nodes not explored must have larger path cost.

Depth-First Search



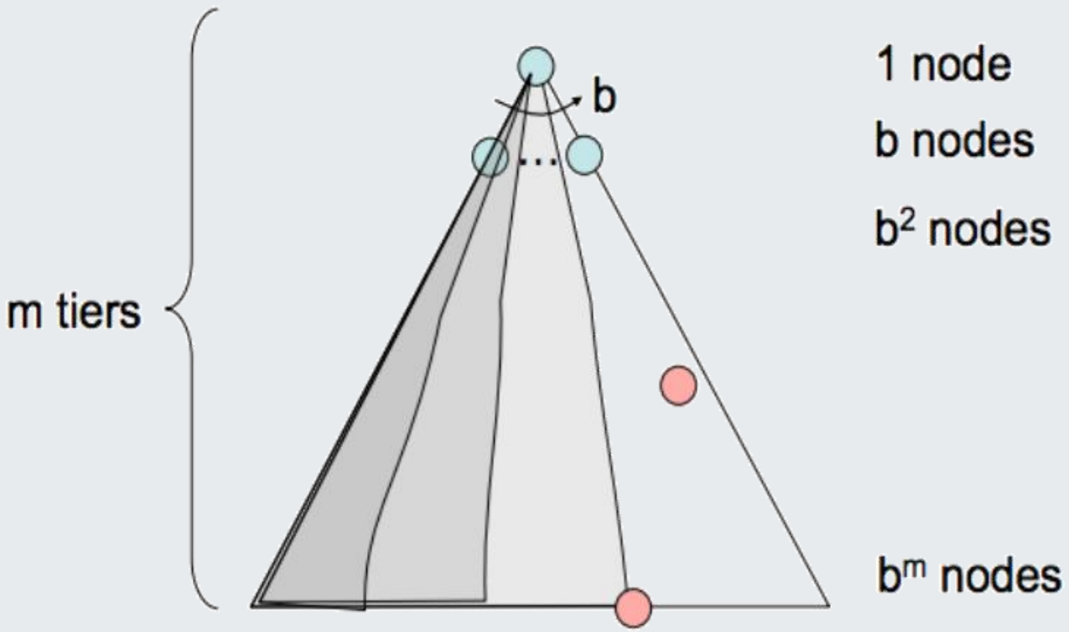
- **Intuition**: try with full sequences of actions, until a sequence leads to a goal.
- **Pros**:
  - more memory-saving.
- **Cons**:
  - fails for trees with infinite depth.
  - usually finds sub-optimal goals.

Depth Limited Search



- **Intuition**: DFS, but depth is limited to ℓ.
- **Pros**:
  - prevents DFS from going too far on one sequence of actions.
- **Cons**:
  - may not find a solution within depth limit.

Iterative Deepening Search



- **Intuition**: DLS, but the depth limit increase by 1 per iteration.
- **Pros**:
  - Get merits of both DFS and BFS.
  - preferred if search space is large and depth of solution is unknown.
- **Cons**:
  - Extra work rerunning top levels of the tree.

Summary

	Time	Space	Complete?	Optimal?
BFS	$O(b^d)$	$O(b^d)$	Yes (if b, d finite)	Yes (if costs uniform)
UCS	$O(b^{1+\lfloor C^*/\varepsilon \rfloor})$	$O(b^{1+\lfloor C^*/\varepsilon \rfloor})$	Yes (if $\varepsilon > 0$ large)	Yes
DFS	$O(b^m)$	$O(bm)$	No (unless b, m finite)	No
DLS	$O(b^\ell)$	$O(b\ell)$	No	No
IDS	$O(b^d)$	$O(bd)$	Yes (if b, d finite)	Yes (if costs uniform)

Notations:

b: the branching factor.

d: depth of shallowest solution.

C^*: cost of optimal solution.

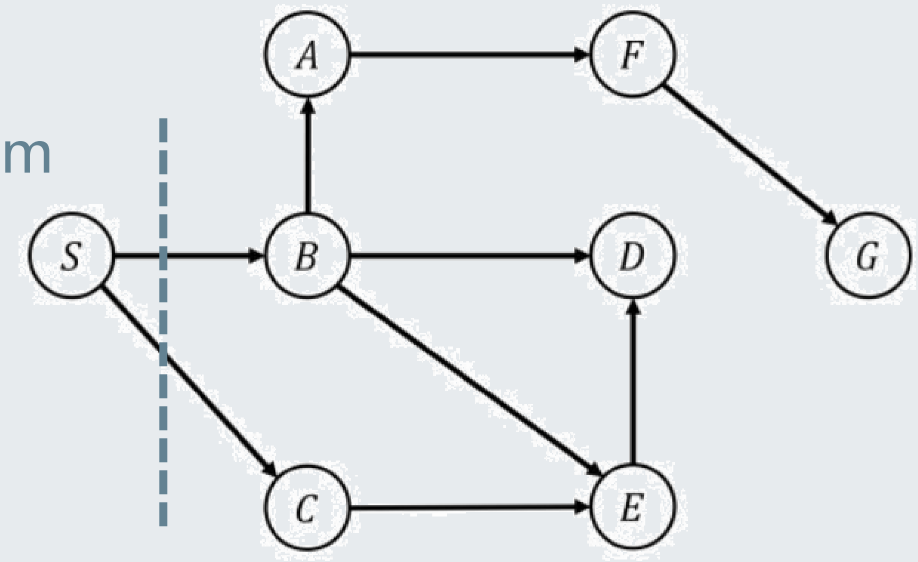
ε: minimum action cost.

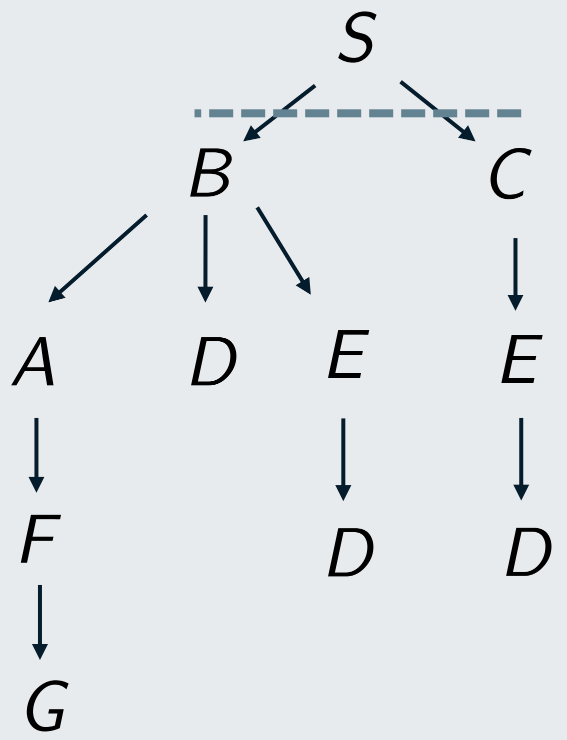
m: maximum depth of search tree.

ℓ: the depth limit.

Problem 2.b

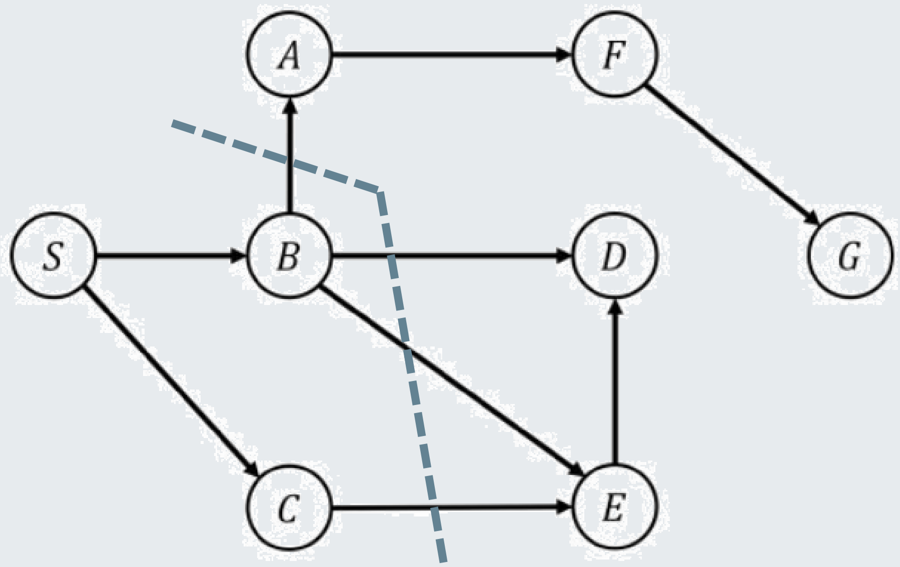
popped from frontier

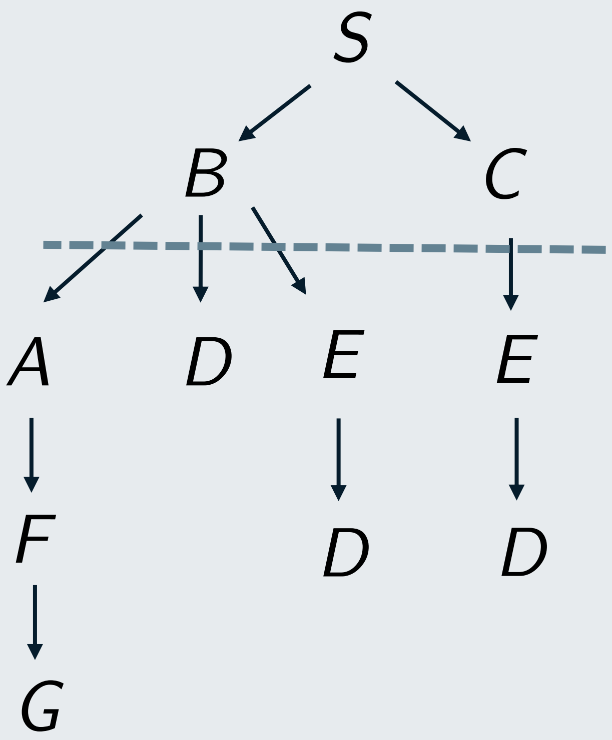




	Graph search	Tree search
BFS	S-	S-
DFS		

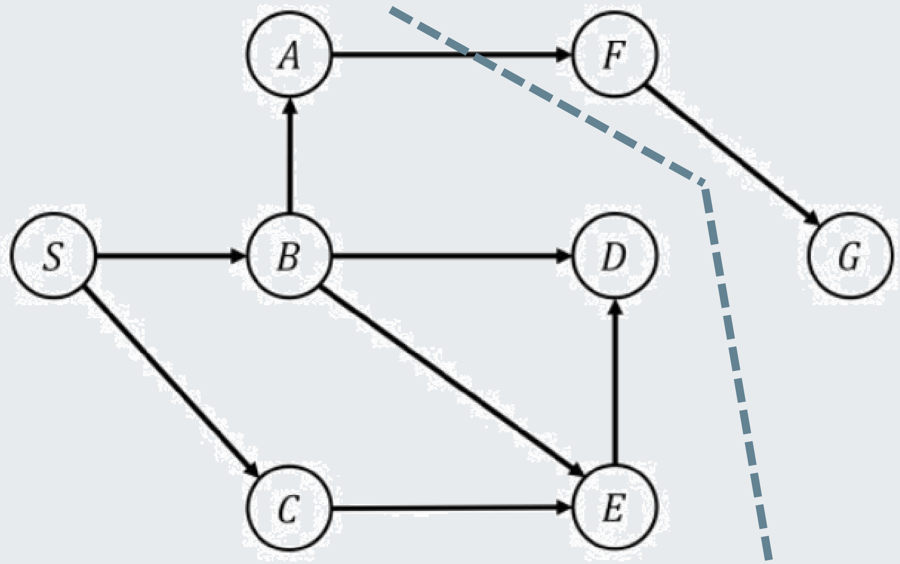
Problem 2.b

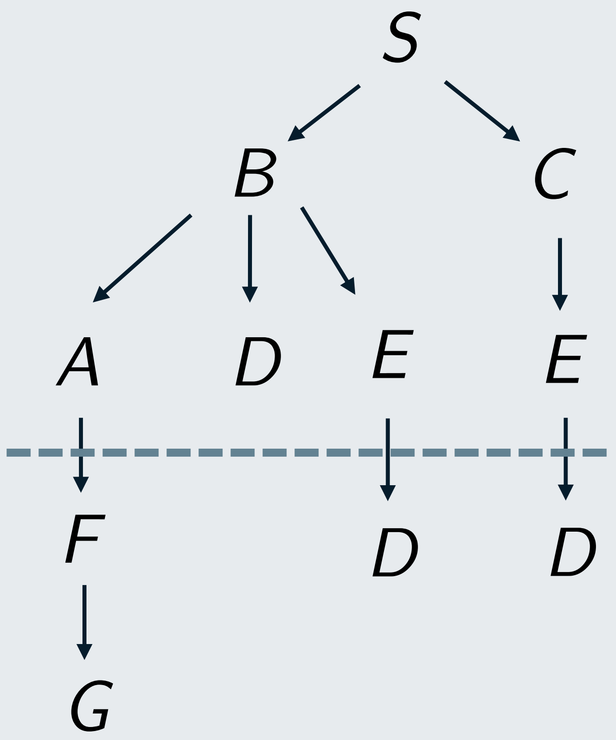




	Graph search	Tree search
BFS	*S-B-C*	*S-B-C*
DFS		

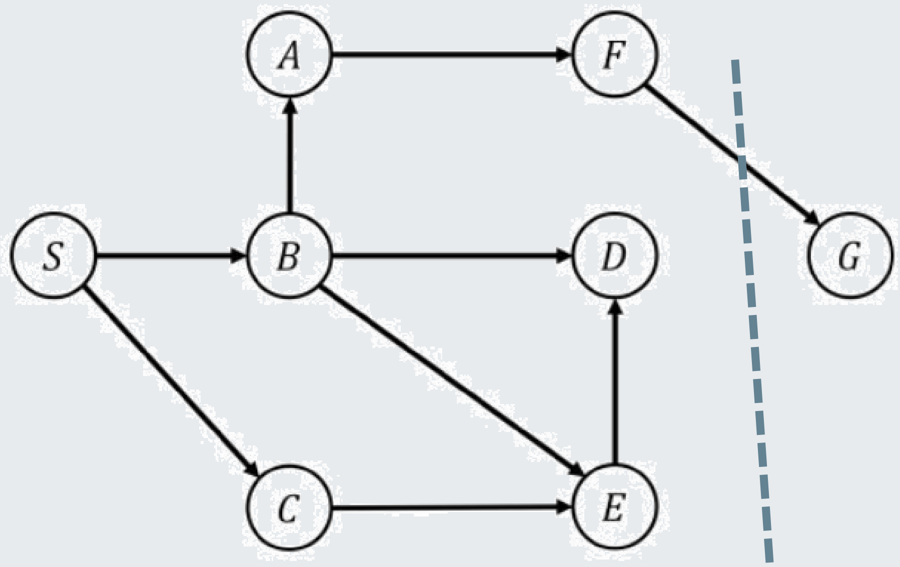
Problem 2.b

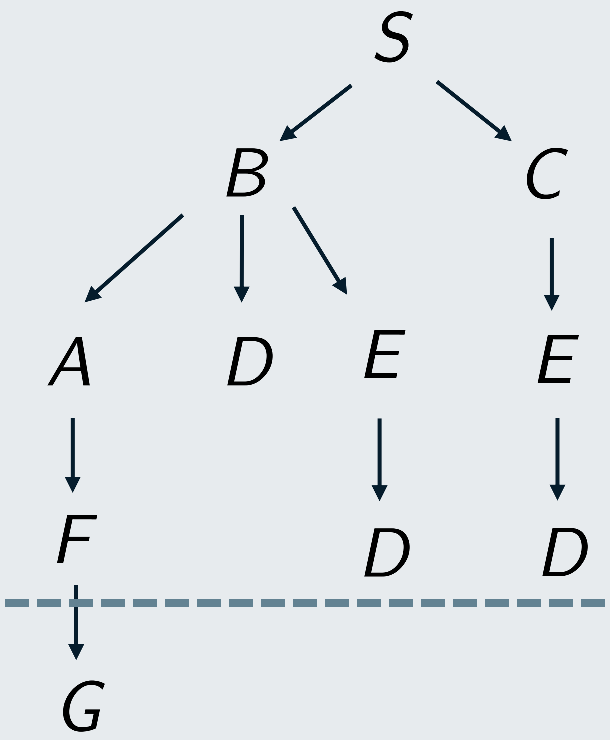




	Graph search	**Tree search**
BFS	*S-B-C-A-D-E*	*S-B-C-A-D-E-E*
DFS		

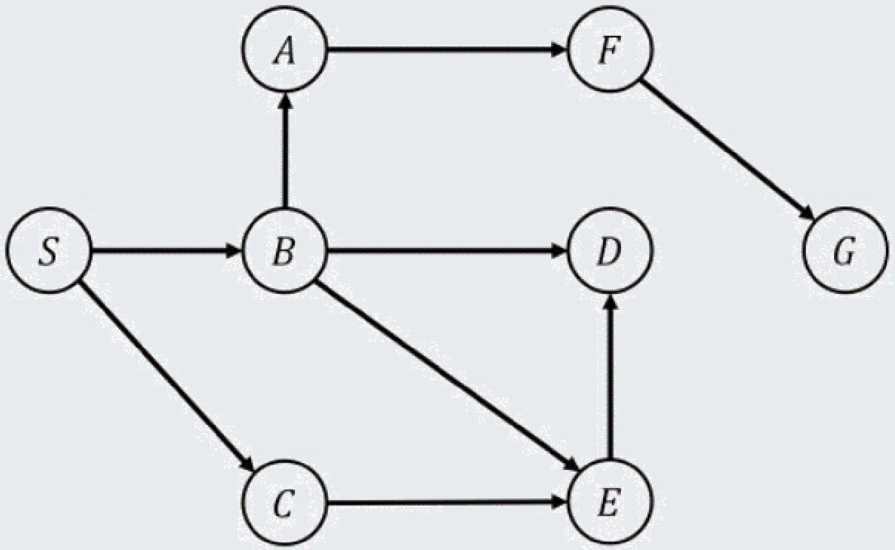
Problem 2.b

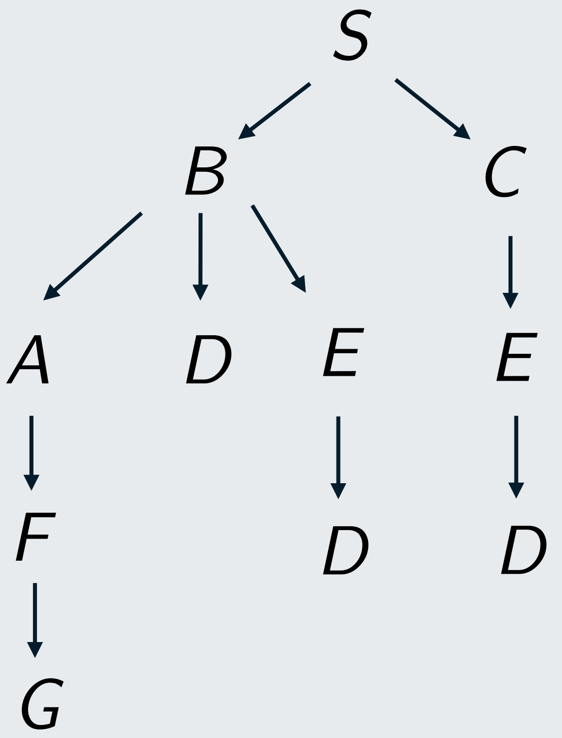




	Graph search	Tree search
BFS	*S-B-C-A-D-E-F*	*S-B-C-A-D-E-E-F-D-D*
DFS		

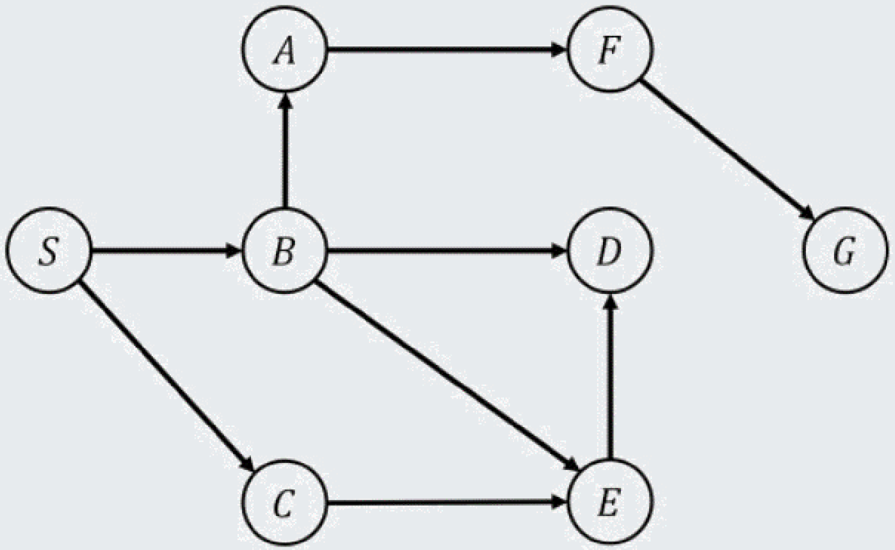
Problem 2.b

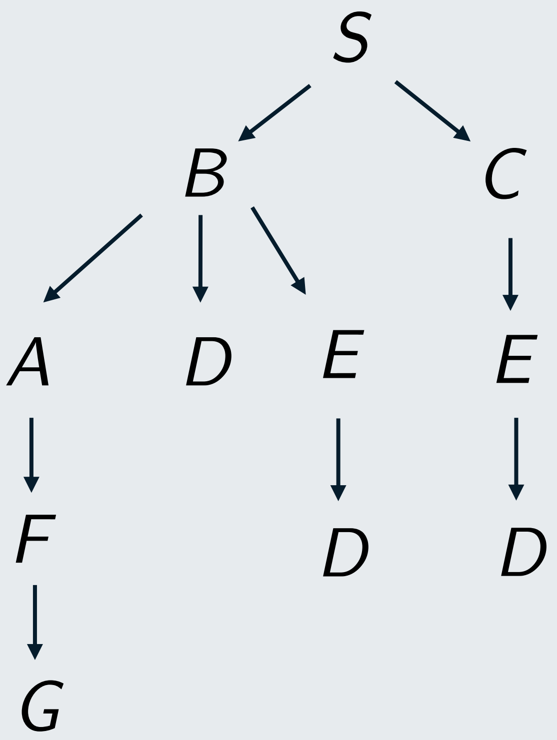




	Graph search	**Tree search**
BFS	*S-B-C-A-D-E-F-G*	*S-B-C-A-D-E-E-F-D-D-G*
DFS		

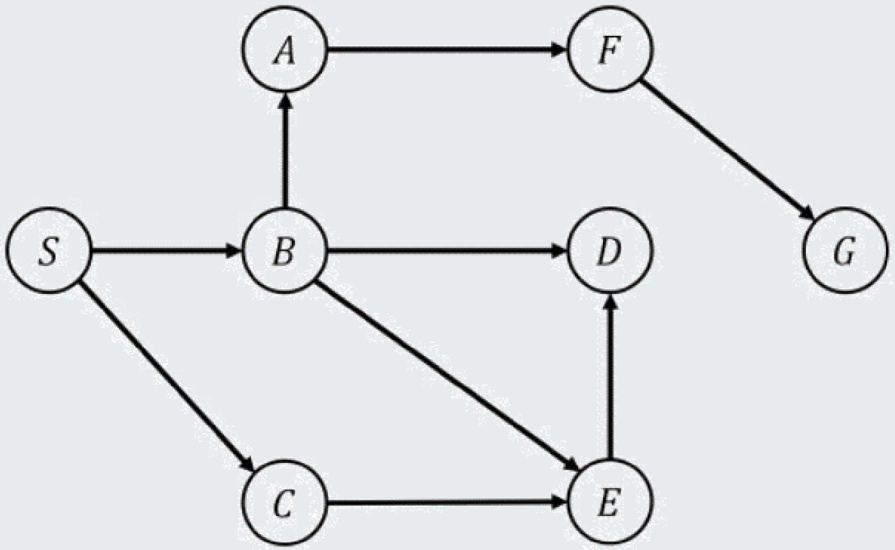
Problem 2.b



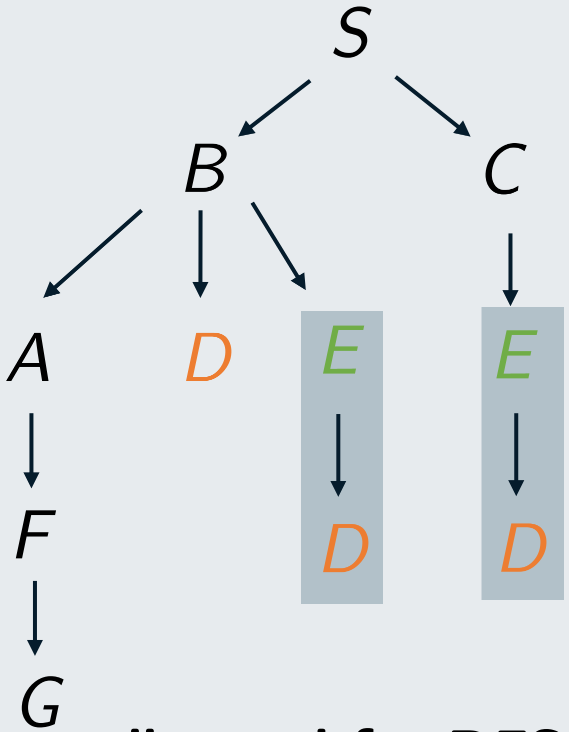


	Graph search	**Tree search**
BFS	*S-B-C-A-D-E-F-G*	*S-B-C-A-D-E-E-F-D-D-G*
DFS	*S-C-E-D-B-A-F-G*	*S-C-E-D-B-E-D-D-A-F-G*

Problem 2.a



usually used for BFS



usually used for DFS

- Tree search may explore redundant paths, and the same state multiple times.
- Graph search will not explore visited state (unless via a more optimal path).

End of File

Thank you very much for your attention!

References

- D. Ler, "Introduction: Problem Environments & Intelligent Agents“, 2023. [Online].
- D. Ler, "Uninformed Search: Problem-solving Agents & Path Planning“, 2023. [Online].
- S. Russell and P. Norvig, "Artificial Intelligence: A Modern Approach," 3rd ed., Prentice Hall, 2010.
- N. Sharma. "Introduction to Artificial Intelligence”, 2022. [Online]. Available:https://inst.eecs.berkeley.edu/~cs188/fa22/assets/notes/cs188-fa22-note01.pdf.