ITCS 451
Section 2

Lecture 1 Intro

Thanapon Noraset
Faculty of ICT, Mahidol University

Adapted from AIMA by Stuart Russell and Peter Norvig, and UC Berkeley CS188 by Dan Klein and Pieter Abbeel (ai.berkeley.edu)

Announcements & Agenda

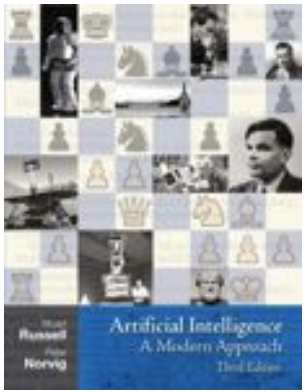
1. Please bring
   notebook (computer)
   next class.
2. also need pen or pencil.
3. Checkpoint 1
   due on Sun.
4. Feedback 1

1. Why study AI ?
2. What is AI?
3. Course Logistics
4. Agent, Environment, states

Credit

This class is based on

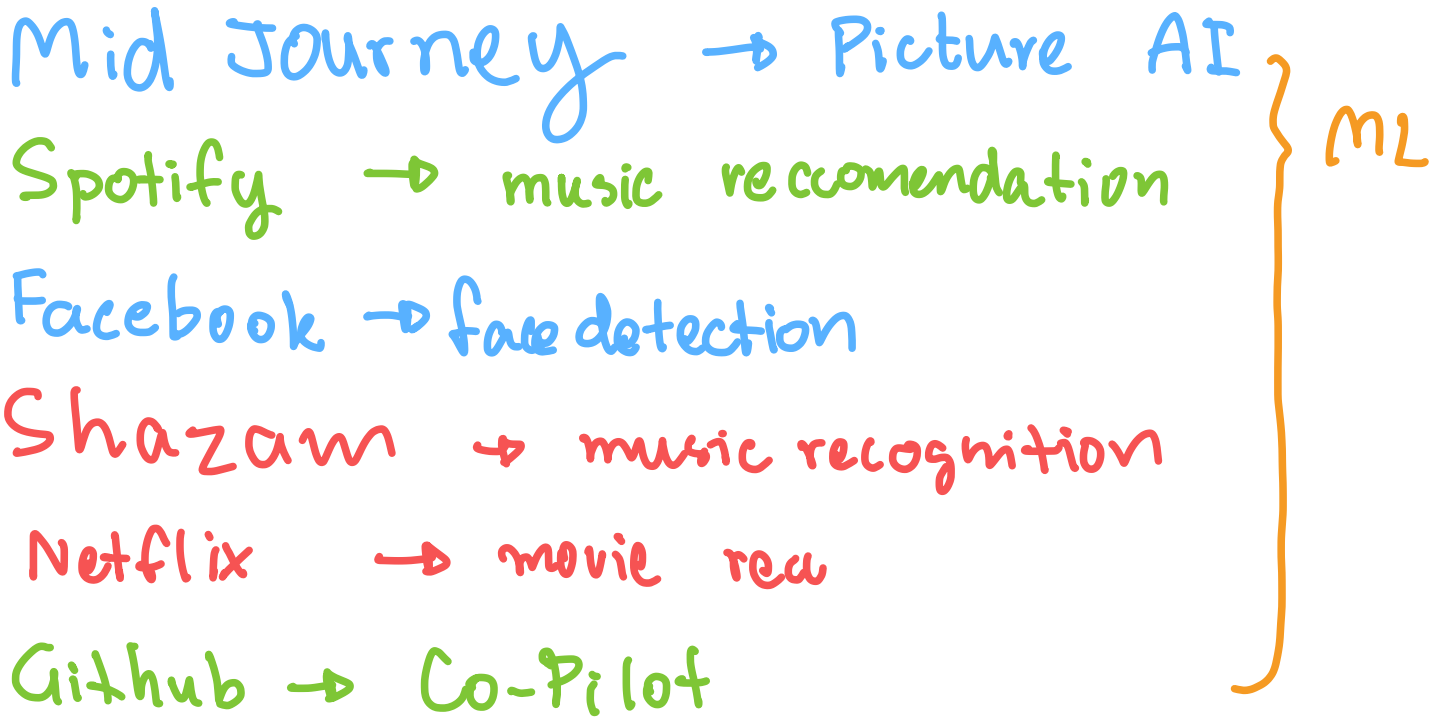
- AIMA by Stuart Russell and Peter Norvig
- UC Berkeley CS188 by Dan Klein and Pieter Abbeel (ai.berkeley.edu)



Most of the artworks from Berkeley CS188 course and the book.

1. Why study Artificial Intelligence?

Can you think of an application of AI?



“AI” is everywhere

Information Technology

Google

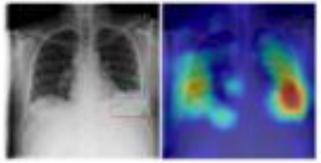
Face ID

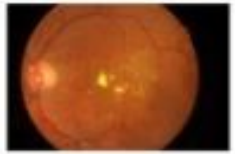
NETFLIX

Game-Playing

Healthcare

Diagnosis and Prescreening





Drug Discovery and Biochemistry

Exscientia

Surgery and other robots

“AI” is everywhere

Media


narrative science

yseop


F.E.A.R.

FIRST ENCOUNTER ASSAULT RECON

Business


Marketing

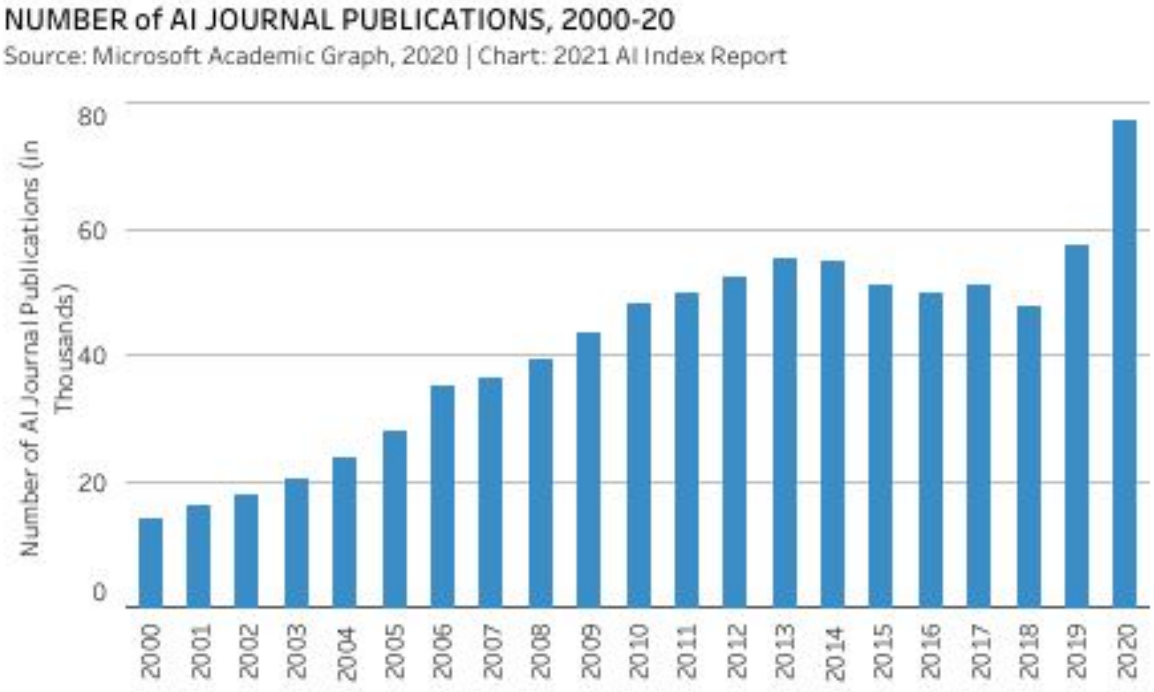

Human Resources


Finance


Manufacturing

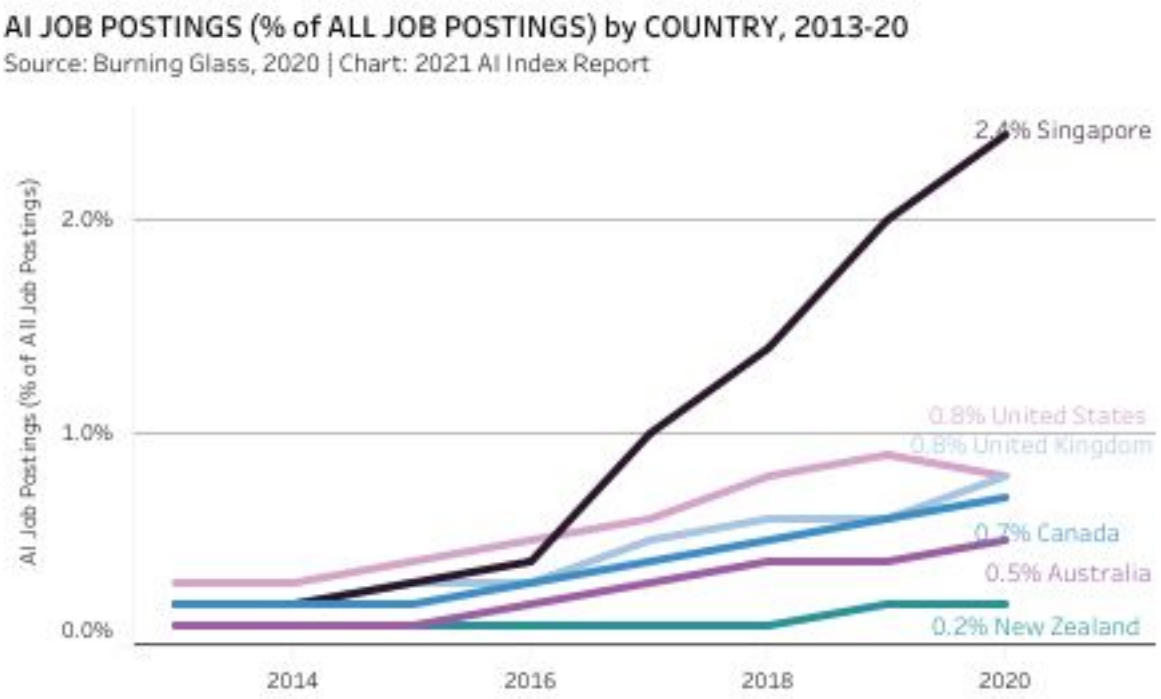
more...

Global AI Trend: Research.



https://aiindex.stanford.edu/ai-index-report-2021/

Global AI Trend: Job Postings.



https://aiindex.stanford.edu/ai-index-report-2021/

AI combines “cool stuff” from many disciplines.

Outside CS

- Philosophy
- Mathematics
- Psychology
- Cognitive science
- Economics
- Control theory
- Computer Engineering

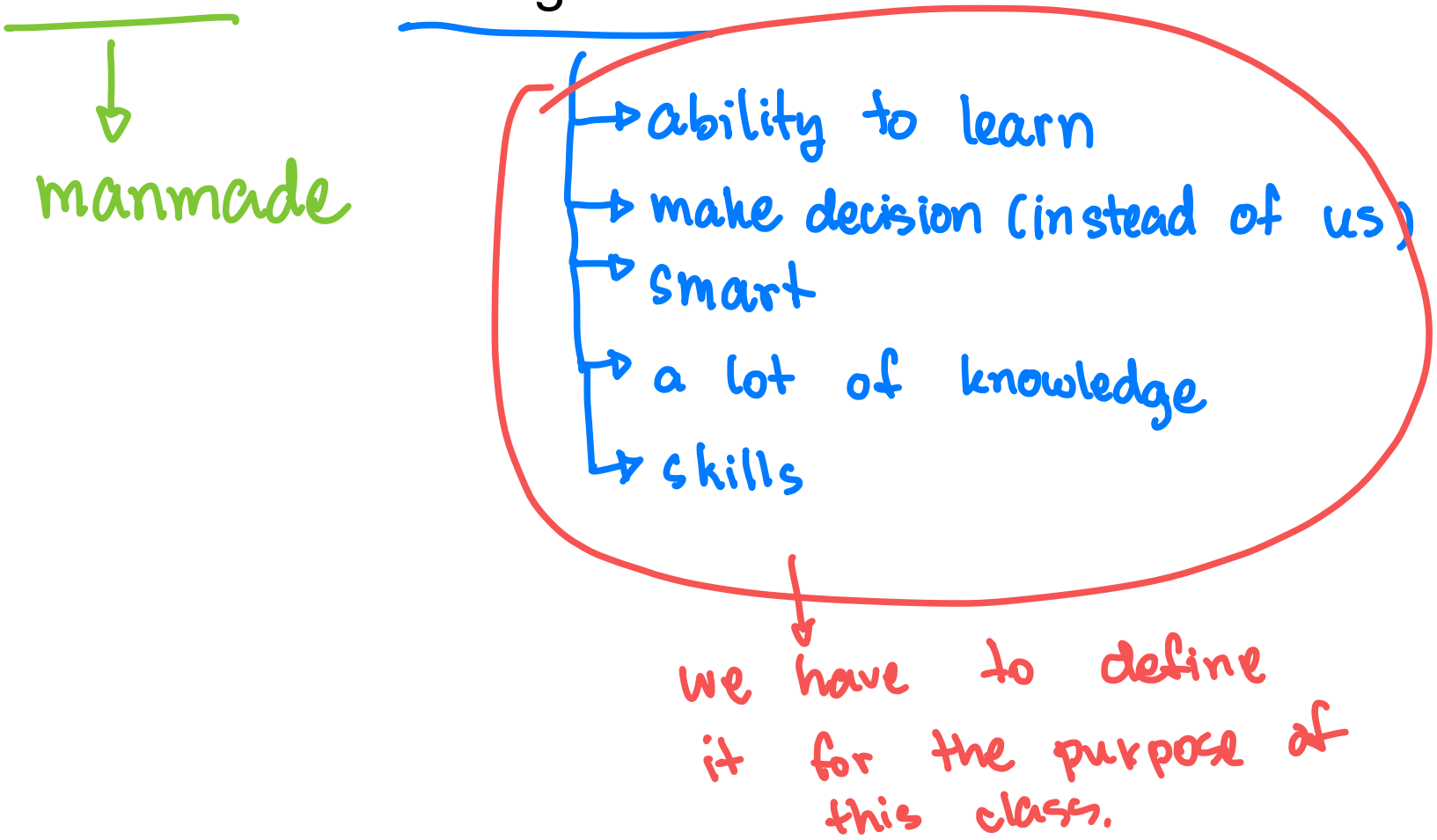
Within CS

- Algorithms
- Optimization
- Parallel Computing
- Data Management

Train our brain to be able to solve hard problems

2. What is AI?

“Artificial” and “Intelligence”



Artificial Intelligence aims to make a machine that ...

Thinking Humanly

think like a human.

- Cognitive science
- neuro sci

Thinking Rationally

think in the ~~"right"~~ "best" way

- logical thinking

* 2nd half.

Acting Humanly

act like human

- Turing test * *

* Acting Rationally *

make the "best" decision

- maximize some score.

Rational Decision and Utility

↳ "maximally achieving pre-defined goals"

"Being Rational"

↓

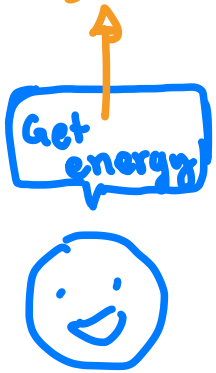
"act to maximize expected utility"

↓ expressed

Utility: calories | Utility of an outcome

↓

measurable quantity



100kcal

200kcal

Value Alignment Problem * Ethics

Walking Agents: https://www.youtube.com/watch?v=hx_bgoTF7bs
CoastRunner: https://www.youtube.com/watch?v=tlOlHko8ySg

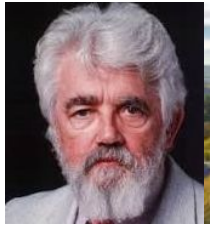
10 mins

Resume. 2:15 pm

A bit of history...

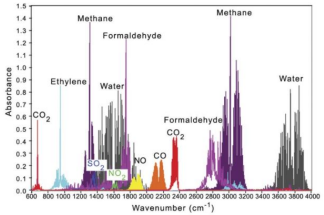
Artificial General Intelligence (AGI)

1956: The birth of “Artificial Intelligence”



Logic, Search algorithm.

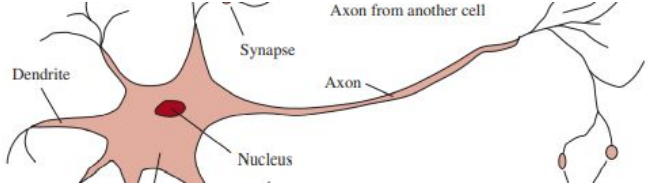
70s-80s: The expert systems domain specific intelligence



AI winter

In another timeline of AI…

1943: Artificial neural network models (McCulloch and Pitts)

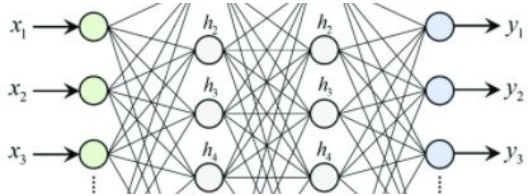


1969: Linear models could not solve XOR (Minsky and Papert)

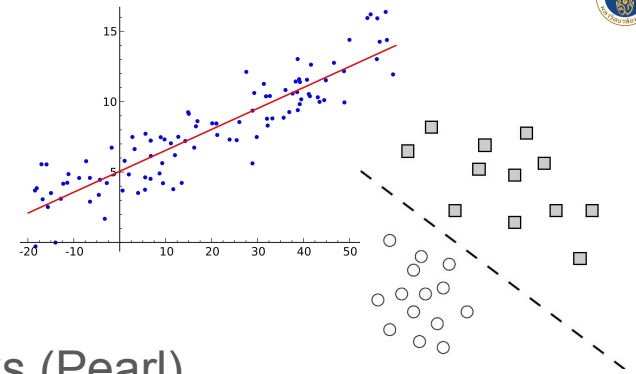
AI winter

1986: The rise of the backpropagation algorithm (Rumelhardt, Hinton, Williams)

Hinton



AI as statistics...



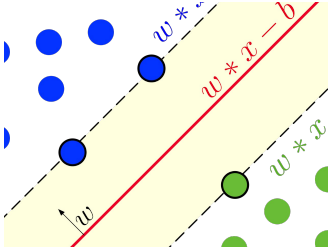
1801: Linear Regression (Gauss and Legendre)

1936: Linear Classifier (Fisher)

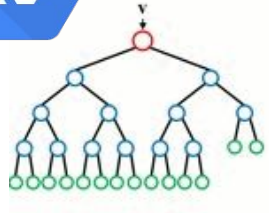
1985: Reasoning with uncertainty: Bayesian networks (Pearl)

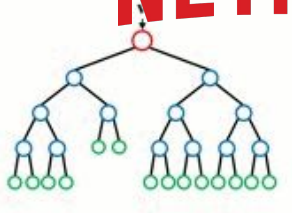
1995: Risk minimization: Support vector machines (Cortes and Vapnik)

2000s - present: Machine Learning



kaggle





NETFLIX

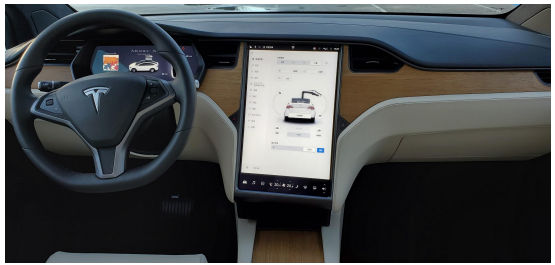
In another timeline of AI...

2010s - present: Deep Learning

Google

DeepMind

OpenAI



Facebook AI Research

3. Course Logistics

Instructors

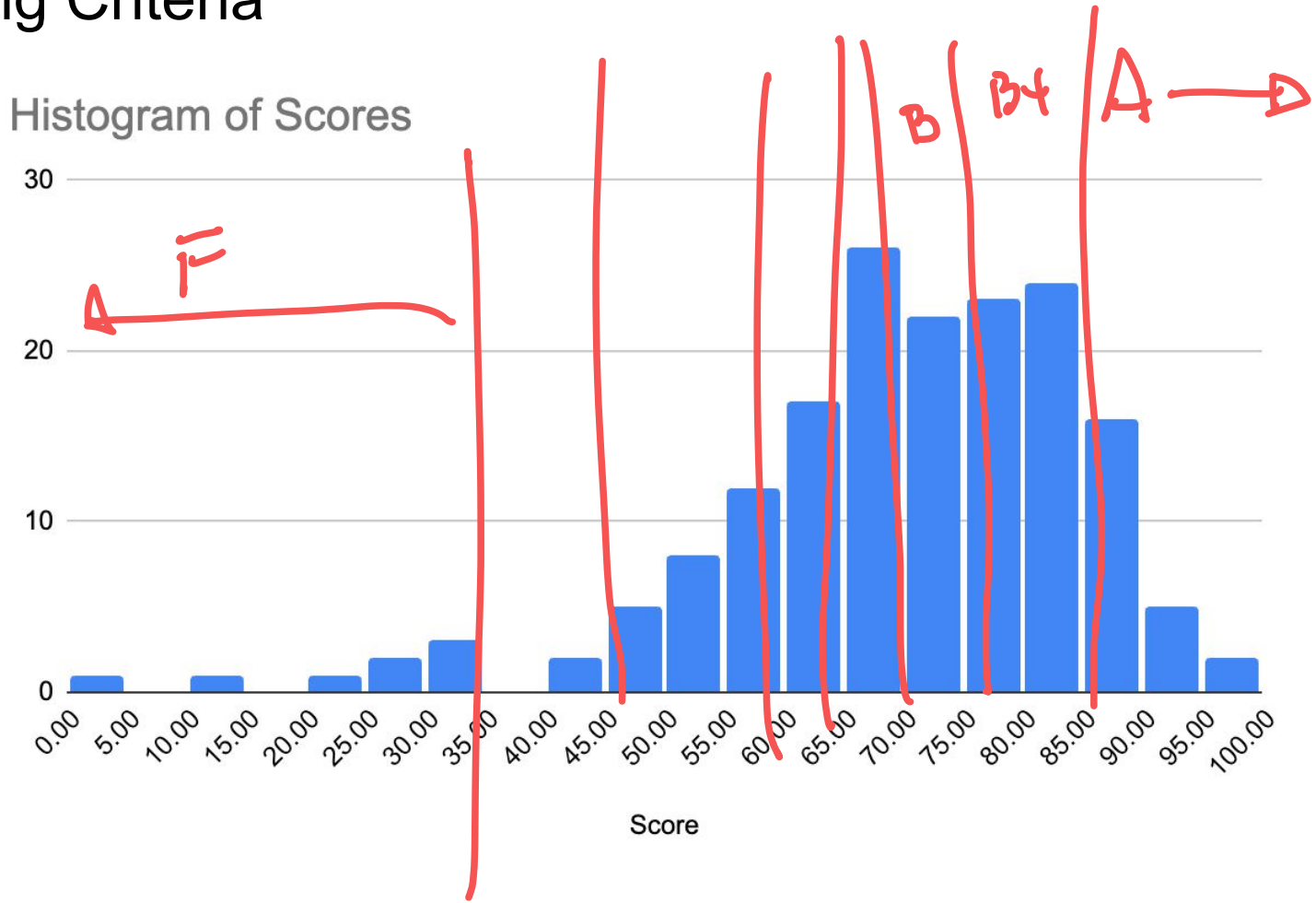
Teaching Assistant

- Dr. Thanapon Noraset (thanapon.nor@mahidol.edu)
- Prof. Dr. Peter Haddawy (peter.had@mahidol.ac.th)
- Asst. Prof. Dr. Apirak Hoonlor (apirak.hoo@mahidol.ac.th)
- Phaphontee Yamchote (phaphontee.yam@student.mahidol.ac.th)

Grading Criteria

Evaluation Methods	Weight (%)	Due
[illegible] Exercise	10	At the end of a topic
Examination	50	9, 17
Assignment	10	Every 2 or 3 weeks
Project I	10	8
Project II	10	14
Project III	10	17

Grading Criteria



Assignments and Project 1

Essay & Presentation

80

Programming

JS: Pathfinding

100

Essay & Presentation

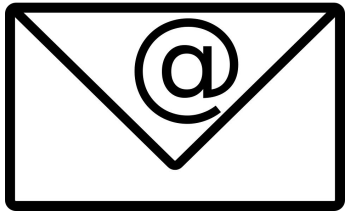
80

Programming

JS: Game playing

100

Need Help?



Ask in the class.

Books

- Artificial Intelligence: A Modern Approach by Stuart Russell and Peter Norvig
- Machine Learning by Tom Mitchell

Courses

- UC Berkeley CS188 Intro to AI
- Stanford CS221: AI: Principles and Techniques

Course Plan

1

Problem-solving by searching

(deterministic)

2

Probabilistic Reasoning

(indeterministic)

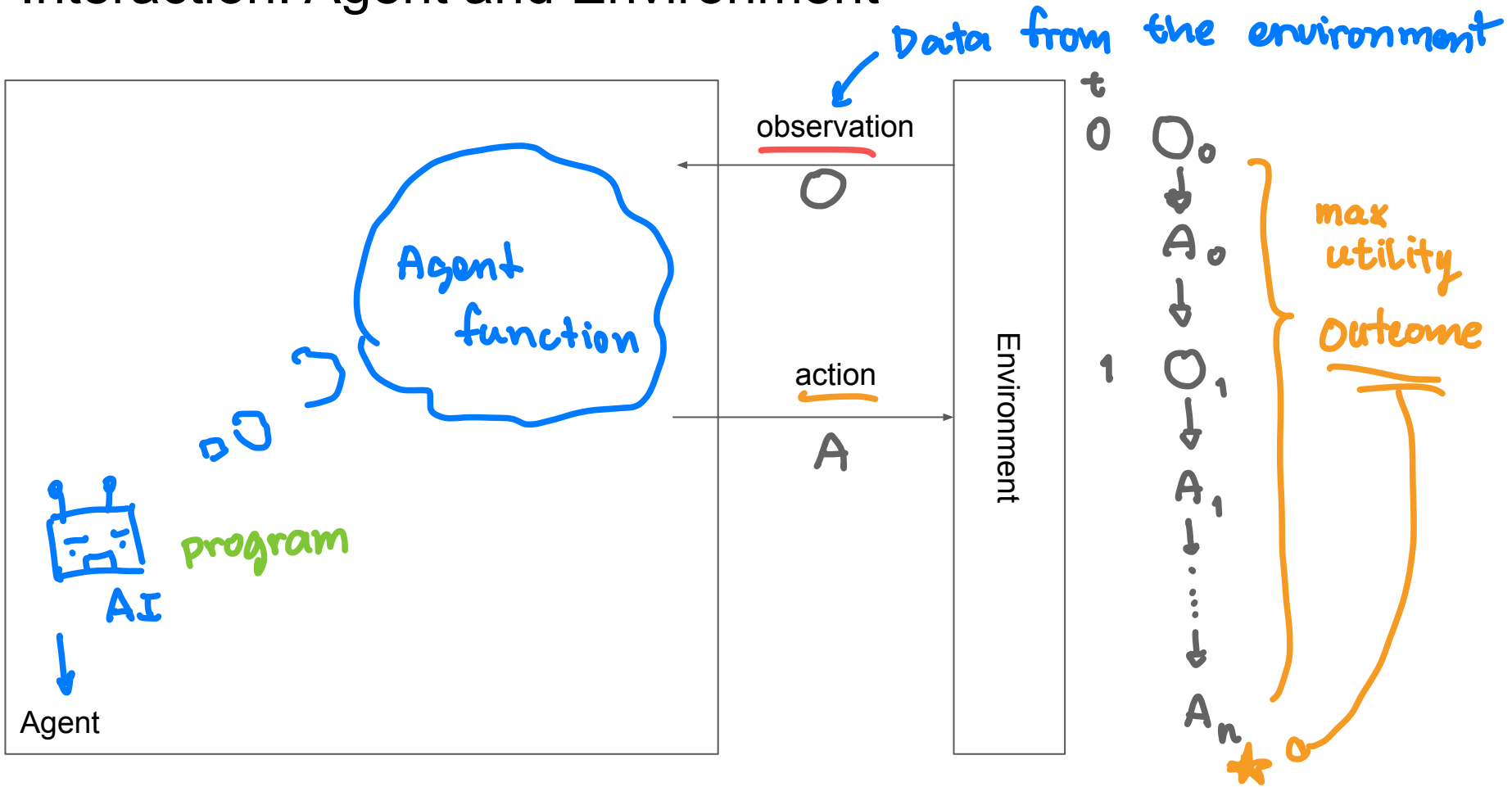
3

Advance Topics

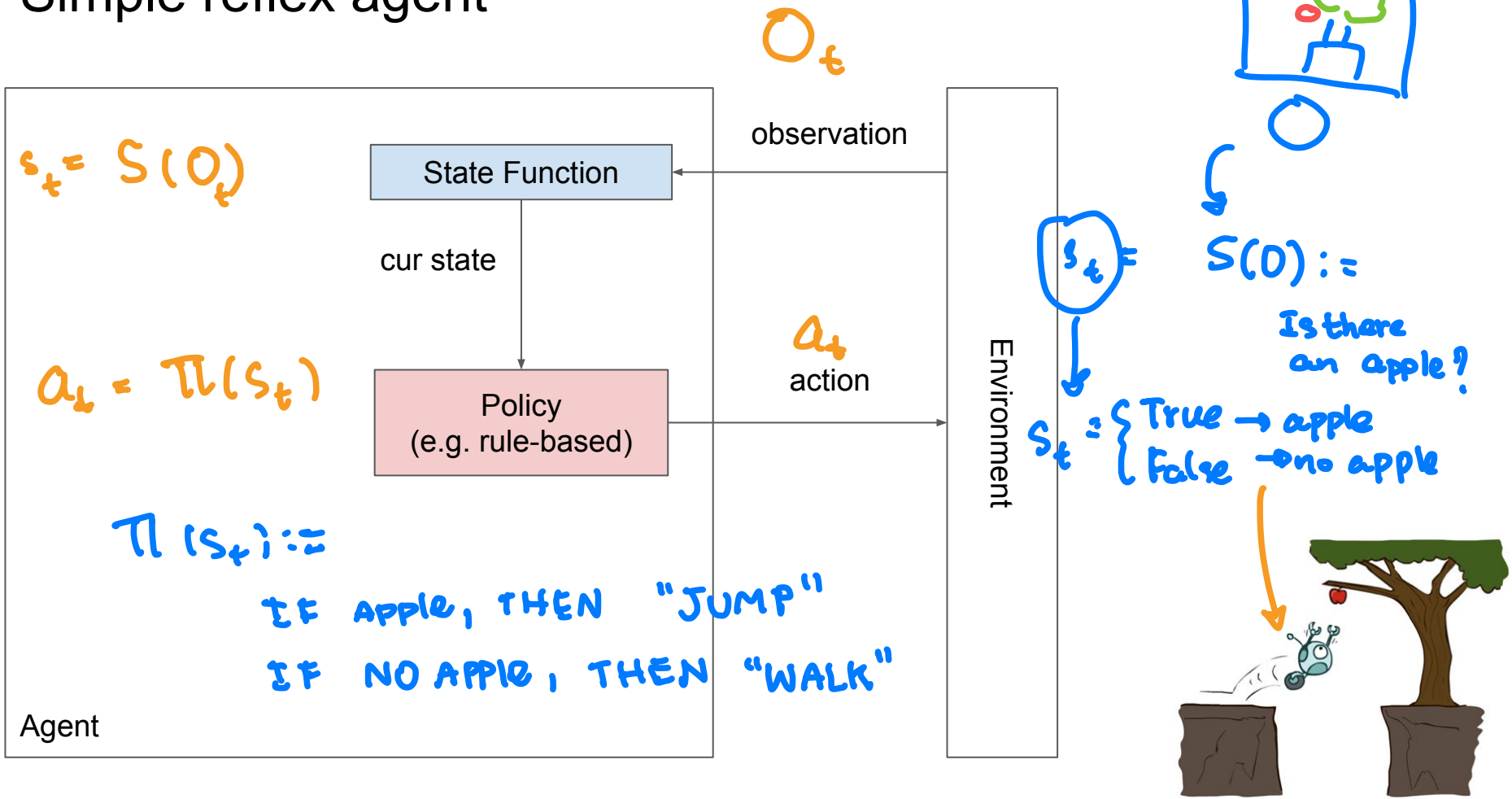
(CSP, ML)

4. Agent, Environment, and State

Interaction: Agent and Environment



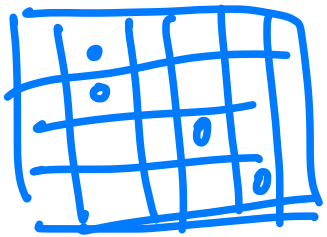
Simple reflex agent



Environment

* Fully Observable

one observation will get all data in the environment



chess

Partially Observable

1. require many observations to get full data.
2. hidden data.

} attemp to infer from what agent can observe.

* also deterministic

configuration, description

State: A representation of a system

char[][] board = new char[3][3];

Environment States S^e

- Data in the environment

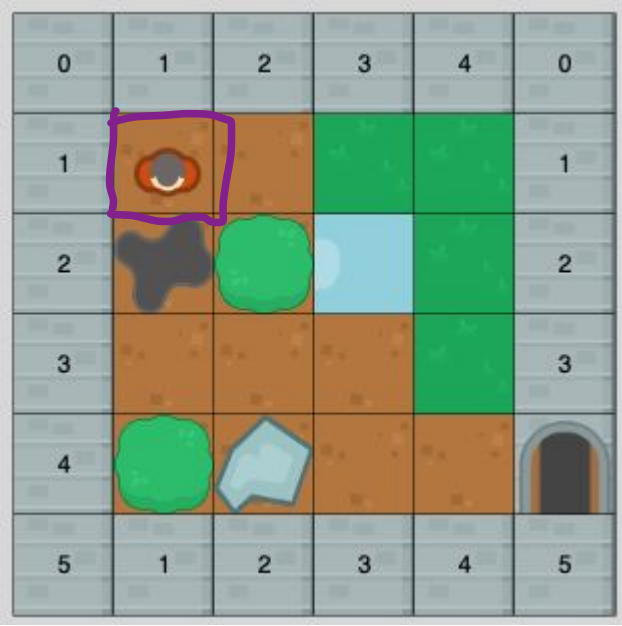
1. $S^e_t \neq O_t$
2. S^e_t usually has too many things

Agent States S^a

Info of the environment inside the agent.

1. only contain what necessary
2. sometimes, $O_t = S^a_t$

State: Examples

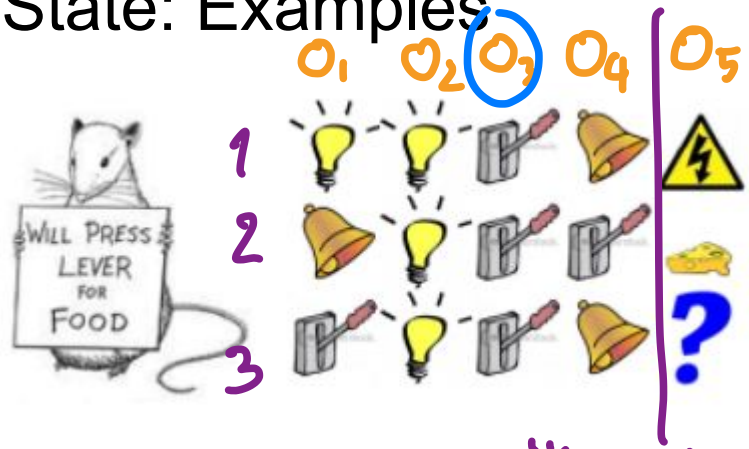


States.

- types of tiles
- cost (energy)
- distance to the goal

* - location, x and y

State: Examples



- 2 prev obs
  - ep1: [0 1 1 0 0]
  - ep2: [0 2 0 0 0]
  - ep3: [0 1 1 0 0]

$S_1^{(1)}$ O_3 $S_2^{(1)}$

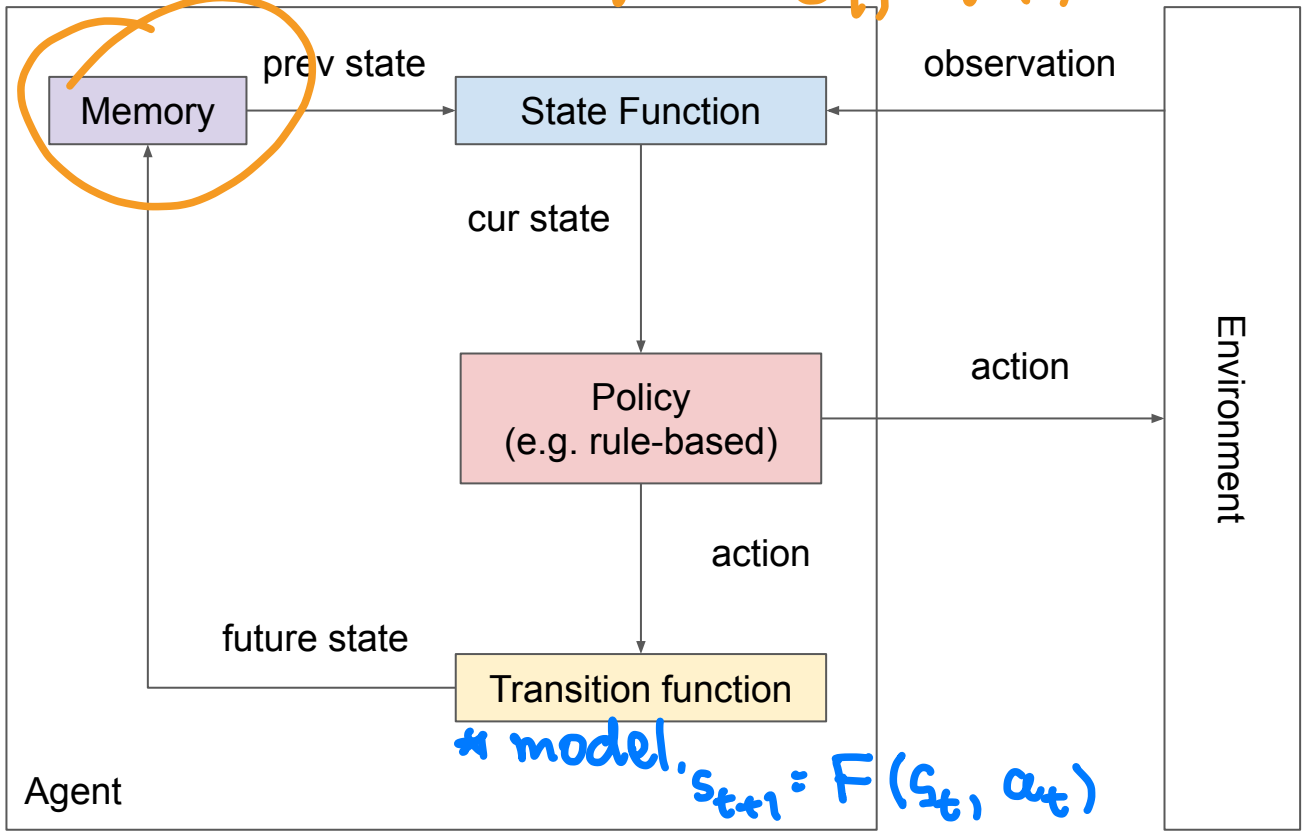
[2 0 0 0] + [0 1 0 0 0] - [1 0 0 0 0]

- 4 previous obs
  - ep1 [2 1 1 0 0]
  - ep2 [1 2 1 0 0]
  - ep3 [1 2 1 0 0]

memory

Model-based reflex agent

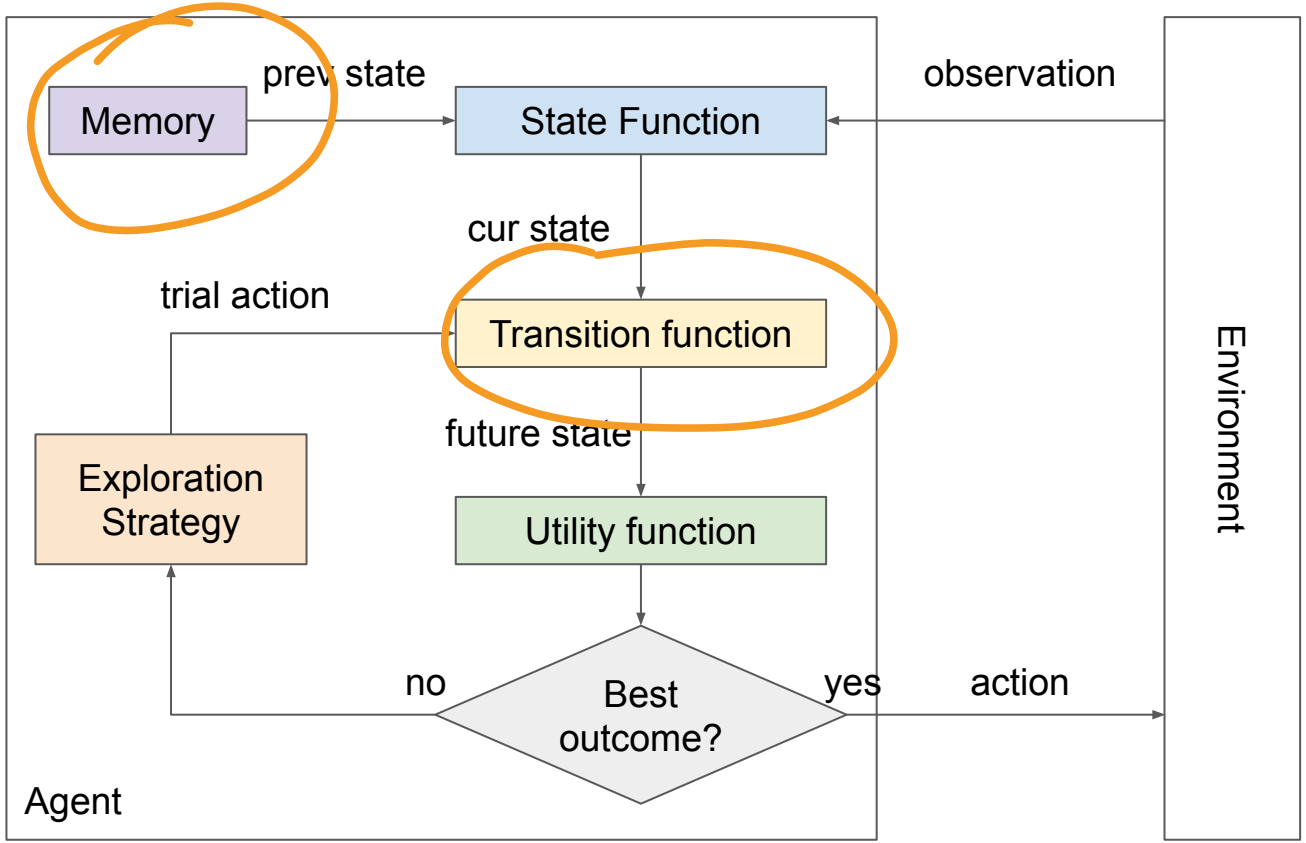
$s_t = S(o_t, s_{t-1})$

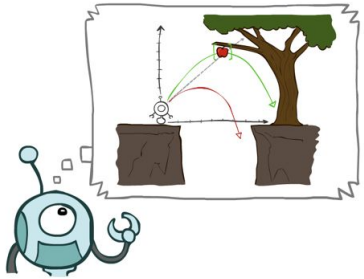


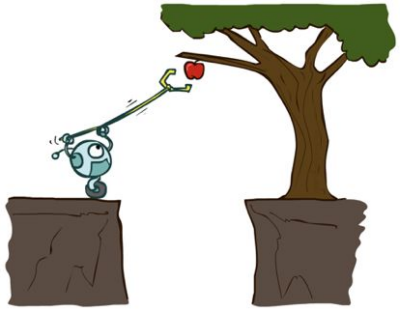
* model, $s_{t+1} = F(s_t, a_t)$

Rational Decisions

Model-based, utility-based agent







Summary

Further Readings

1. AIMA Chapter 1
2. AIMA Chapter 2
3. Get familiar with p5.js

Lecture 2

Thanapon Noraset
Faculty of ICT, Mahidol University

Adapted from AIMA by Stuart Russell and Peter Norvig, and UC Berkeley CS188 by Dan Klein and Pieter Abbeel (ai.berkeley.edu)

Feedback Summary

1. Increase speaking volume
2. Write a bit bigger
3. Model-based reflex agent
4. States

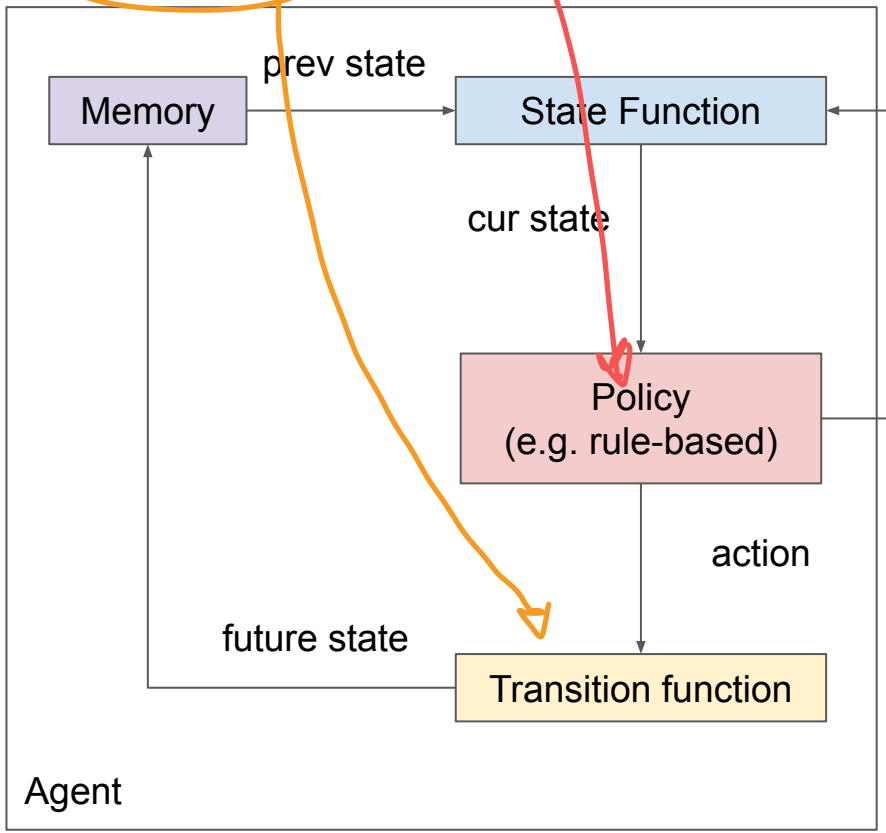
Announcements & Agenda

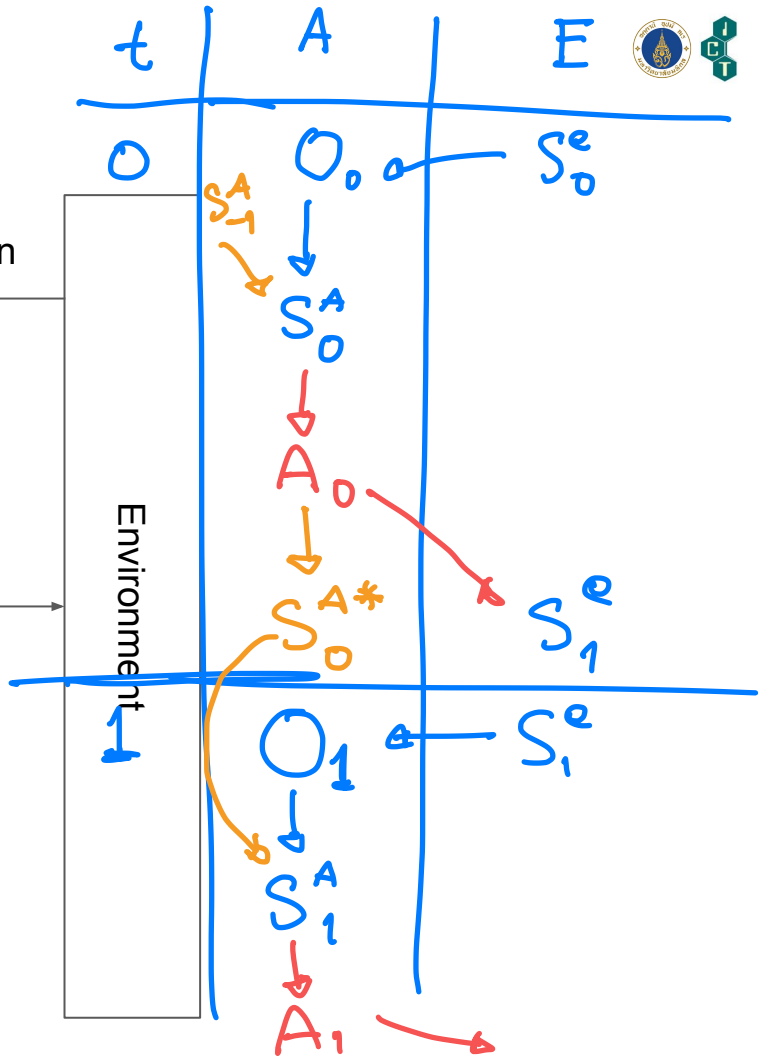
1. Feedback 2.
2. CheckPoint 2 due today
3. Assignment 1 will be announced on Fri

1. Agent, Environment, and State
2. Planning and State-space Exploration
3. Search Algorithms
4. Uninformed Search Strategies

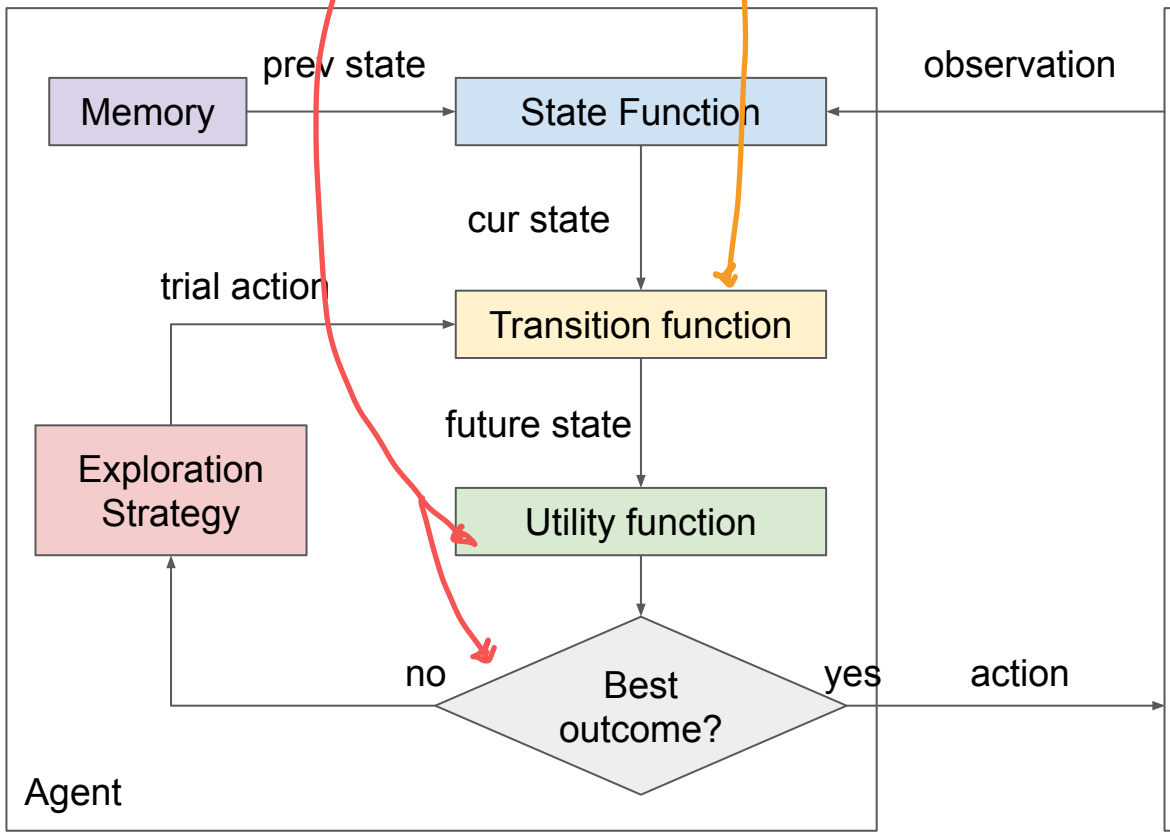
1. Agent, Environment, and State

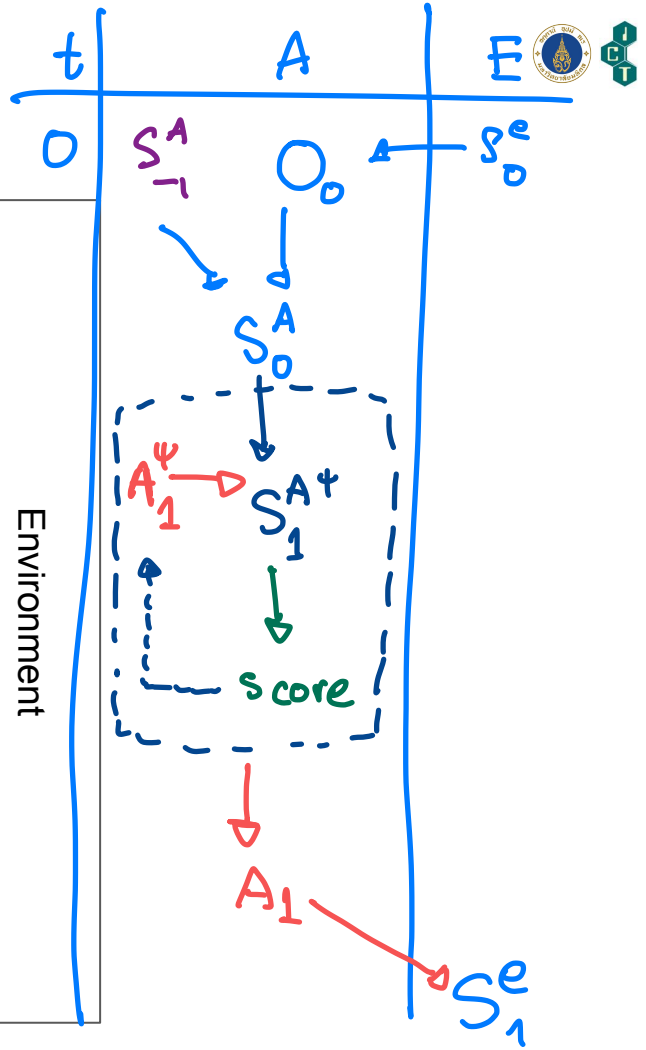
Model-based reflex agent



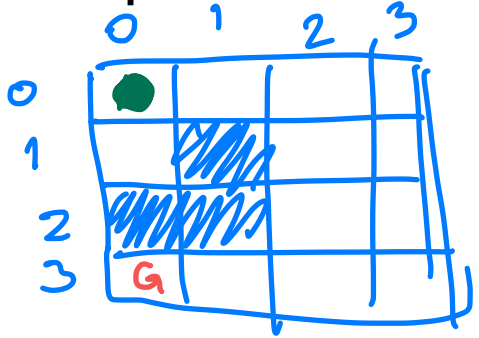


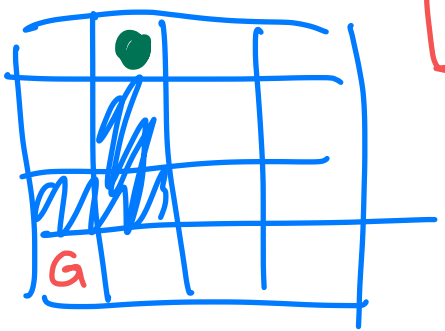
Utility-based, model-based Agent





Example: Pathfinding





$A_0 =$ 'Right'

Iter 1

A_0^{Ψ} = 'Down'

$S_0^A = (0,0)$

$S_1^{A\Psi} = (0,1)$

Score = 0

Iter 2

A_0^{Ψ} = 'Right'

$S_0^A = (0,0)$

$S_1^{A\Psi} = (1,0)$

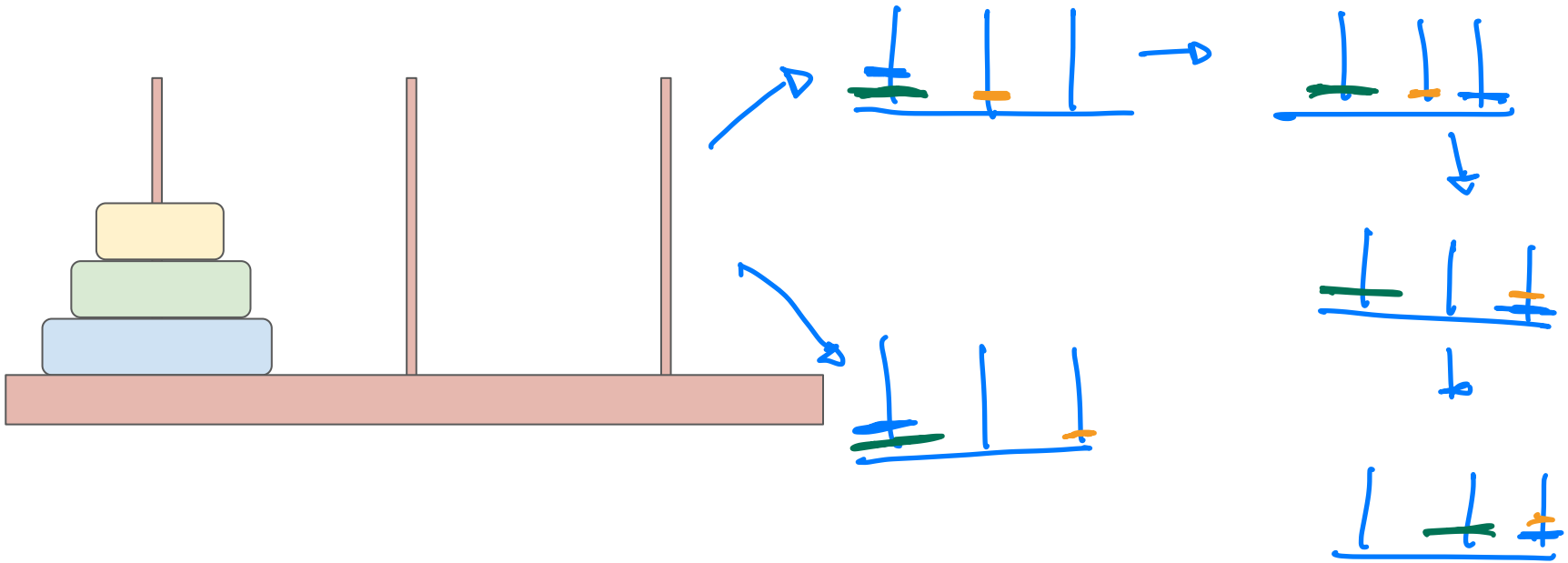
Score = 0

A_1^{Ψ} = 'Up'

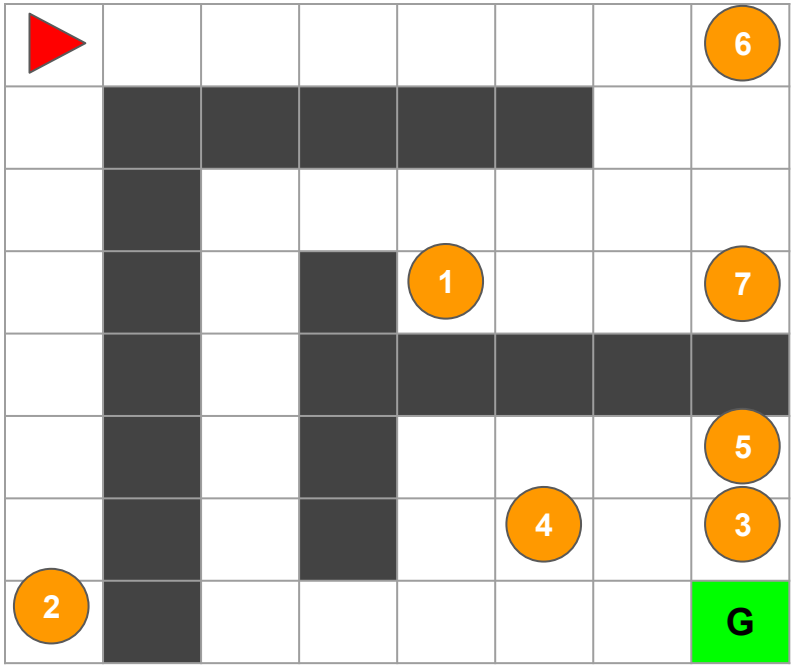
$S_1^A = (0,1)$

$S_2^{A\Psi} = (0,0)$

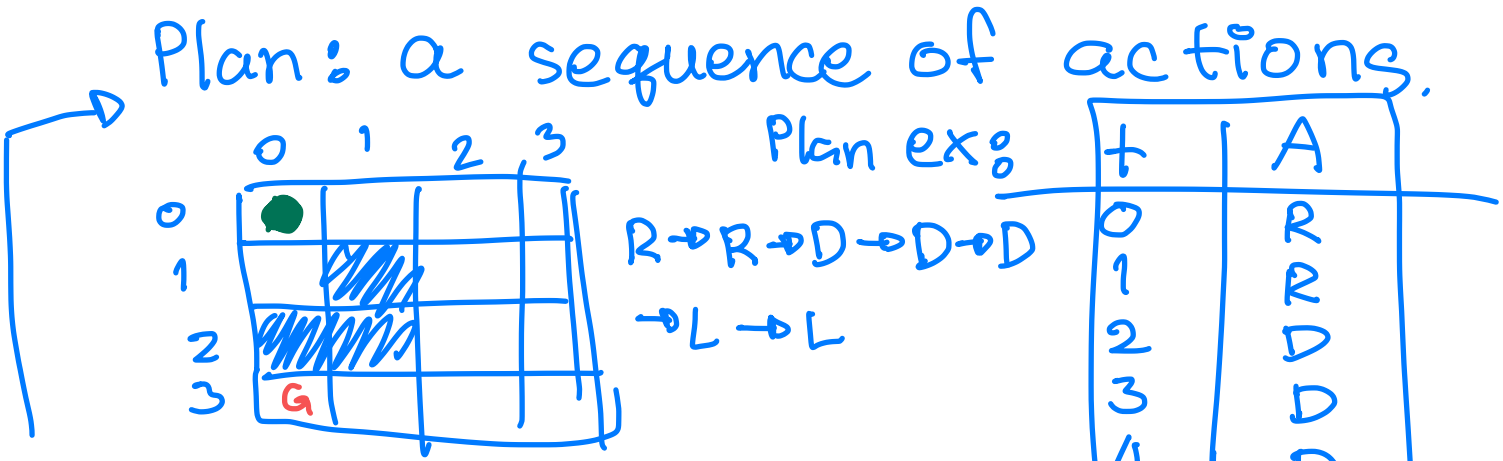
Example: Tower of Hanoi



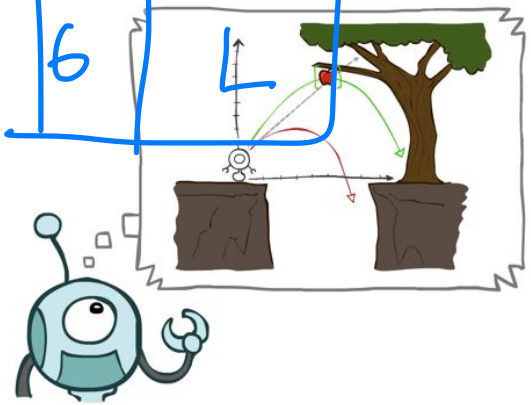
Example: Collect and Goal



Poll: Can you come up with a good order of the balls we should collect?



2. Planning and State-space Exploration



Resume 14:15

Planning: Assumptions and Objectives

1. Fully observable environment
2. Deterministic environment
3. We have a model (transition func)

To find a plan (a sequence of actions) that take an agent from an initial state to a goal state using minimum cost.

maximize utility

↳ utility = -1 * cost

Activity 1: Find a plan for the least energy path.

https://editor.p5js.org/thanapon.nor/full/0dOqnQ7Oo

1 time step

2:30 pm

On a piece of paper, please answer the following questions

1. What are the actions that the agent can output? actions = {u, d, l, r}
2. What are the differences among different types of tiles (areas)?
   Dirt, Grass, Water, Stone, Tree, Mud
   1 3 X X X 6
3. What are the plans and their energy requirements that you discovered?
   S → r → d → r → v ... : energy = 1000

at a time such as r, d, d, d, d: 11).

d, d, r, r, u, u, r, d, d, d, r = 32

r, r, d, d, d, r, r = 17

s, d, d, r, r, d, r, r: 15

enegy:

S>d>d>r>r>d>r

eft down right a b a b

energy:

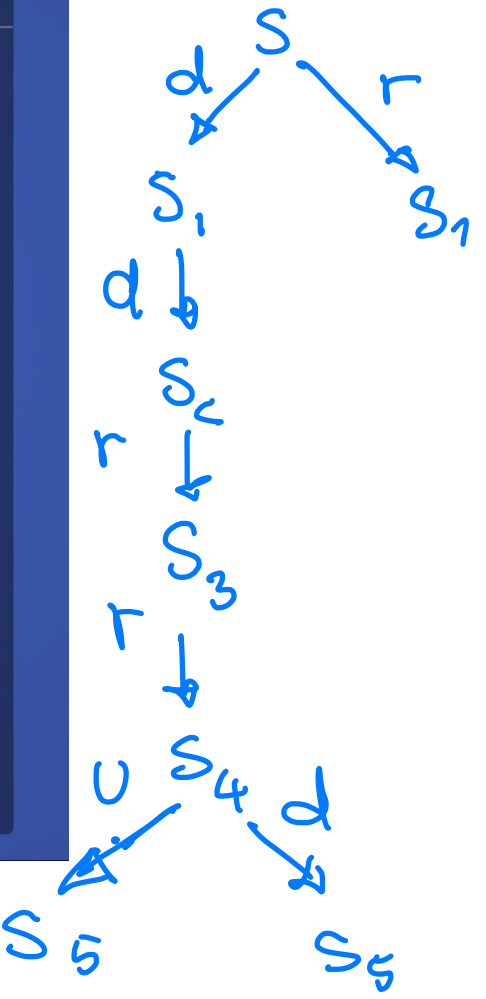
s, d, d, r, r, r, d, r: 17

r:20

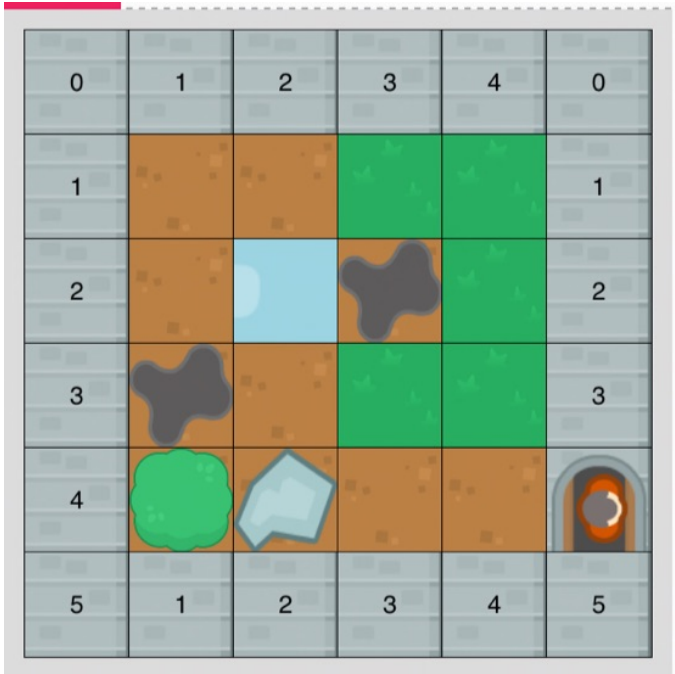
ddrruurrdddr:35

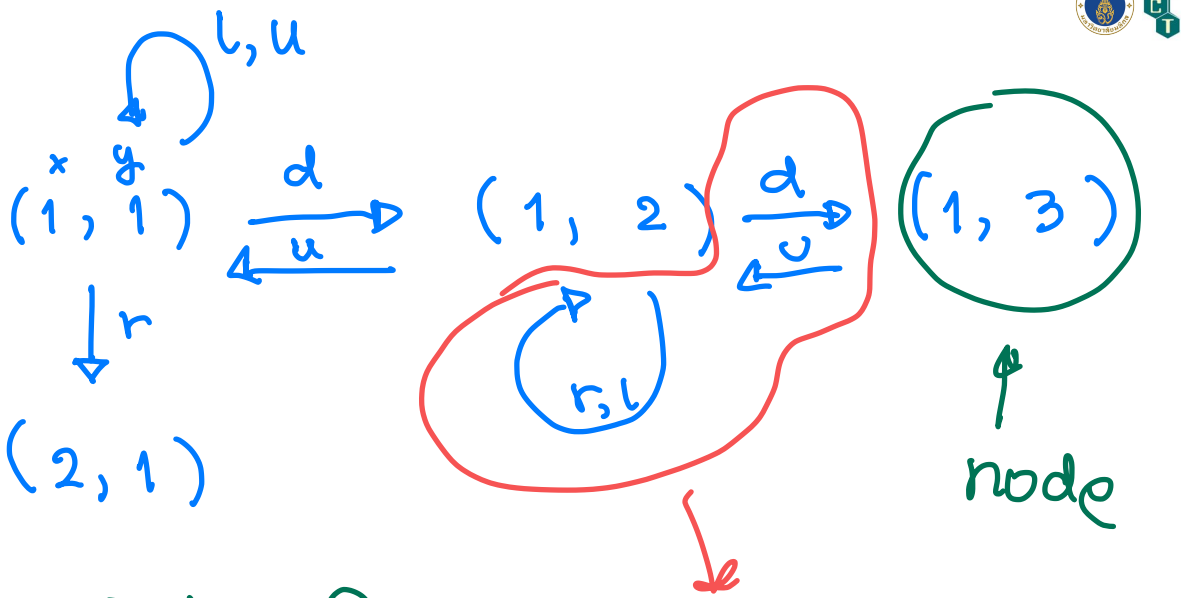
S, r, r, r, d, d, d, r: 16

d, d, r, r, d, r, r: 15



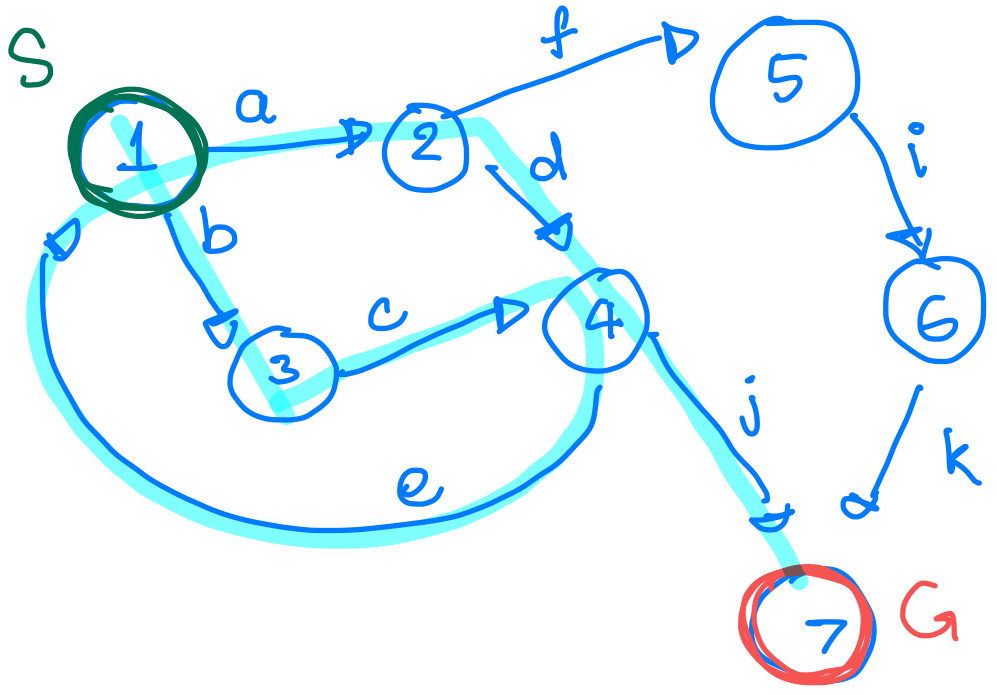
State Space





State Space is a graph

- nodes: State
- edges: Possible actions

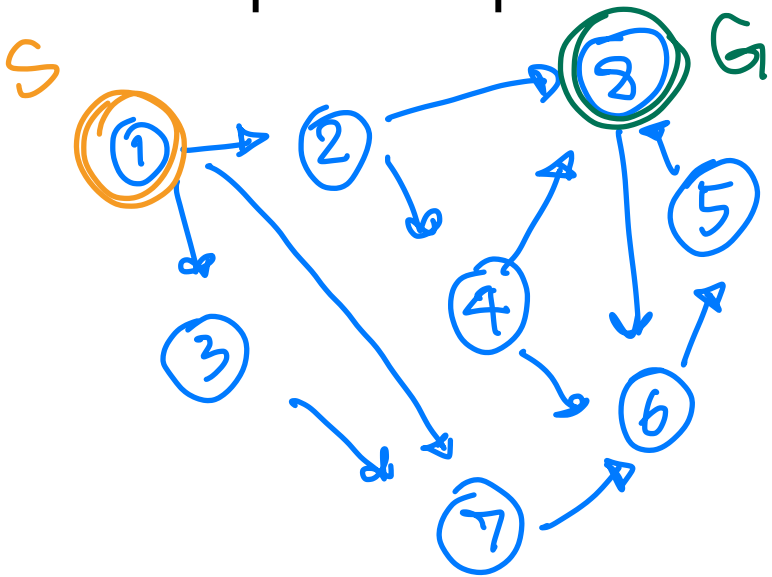


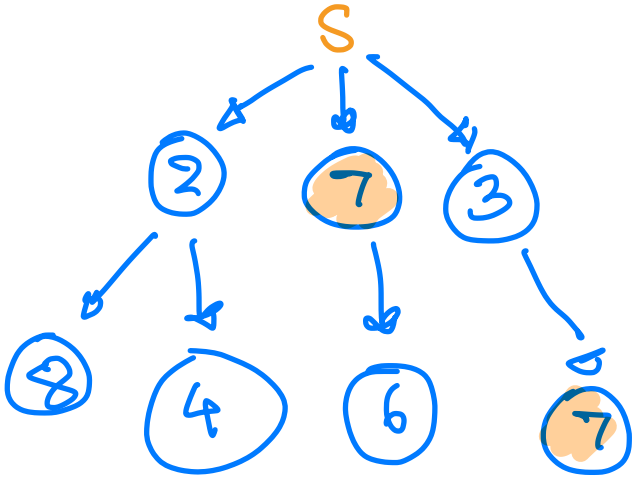
Abstract Example

Plan 1 : b → c → e → a → d → j

cost : 6 (steps)

State-Space Exploration





State-Space Exploration

: Tree

- nodes: current plan
- edges: Previous-Next

Activity 2: A systematic way to explore?

https://editor.p5js.org/thanapon.nor/full/T8Hk1cSlK

On a piece of paper, please answer the following questions:

4. During the exploration, did you find moving back useful?

5. What are the approaches that you use to explore?

1. Cost of actions

2. Exploration strategy

   ↳ transition function

3. Search Algorithms

Search Problem Formulation

1. Initial State
2. Possible actions
3. Transition function
4. Goal test function
5. Cost function

Search Problem

Formulate ← Problem

you

Agent

Search Algorithm → Plan

* run inside agent
* no action committed *

Example: Pathfinding

Step 1 Formulation

1. Initial state : $S_0 = (x, y) = (1, 1)$
2. Action space : $\{U, D, R, L\}$
3. Transition function

   Given input state $= (x, y)$

   if action == U, return $(x, y-1)$

   if action == D, return $(x, y+1)$

   * if the action is valid
4. Goal test funtion:

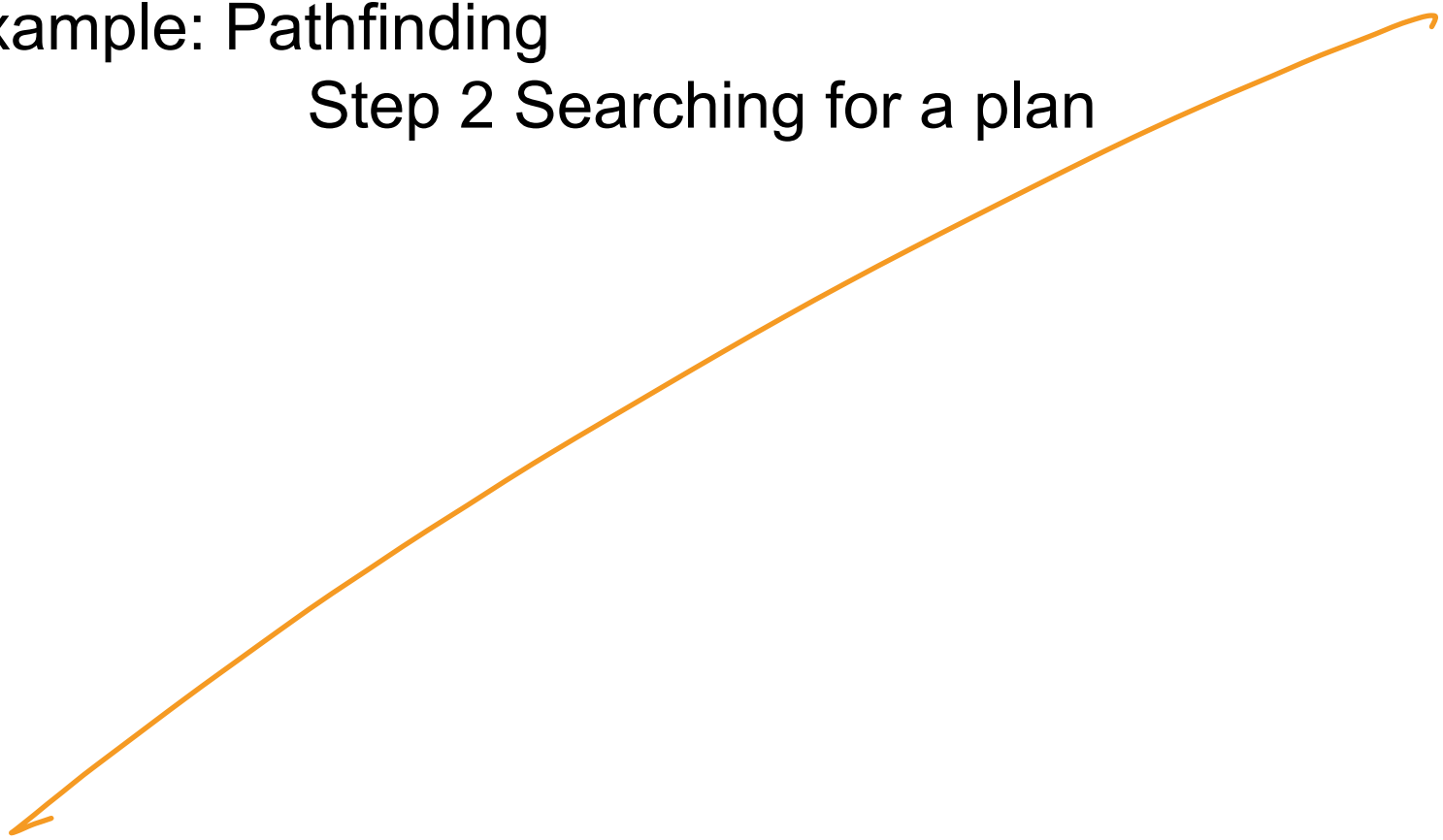
   $S == (5, 4)$
5. Cost function:

   if mud, return 6

   usually cost of each action

Example: Pathfinding

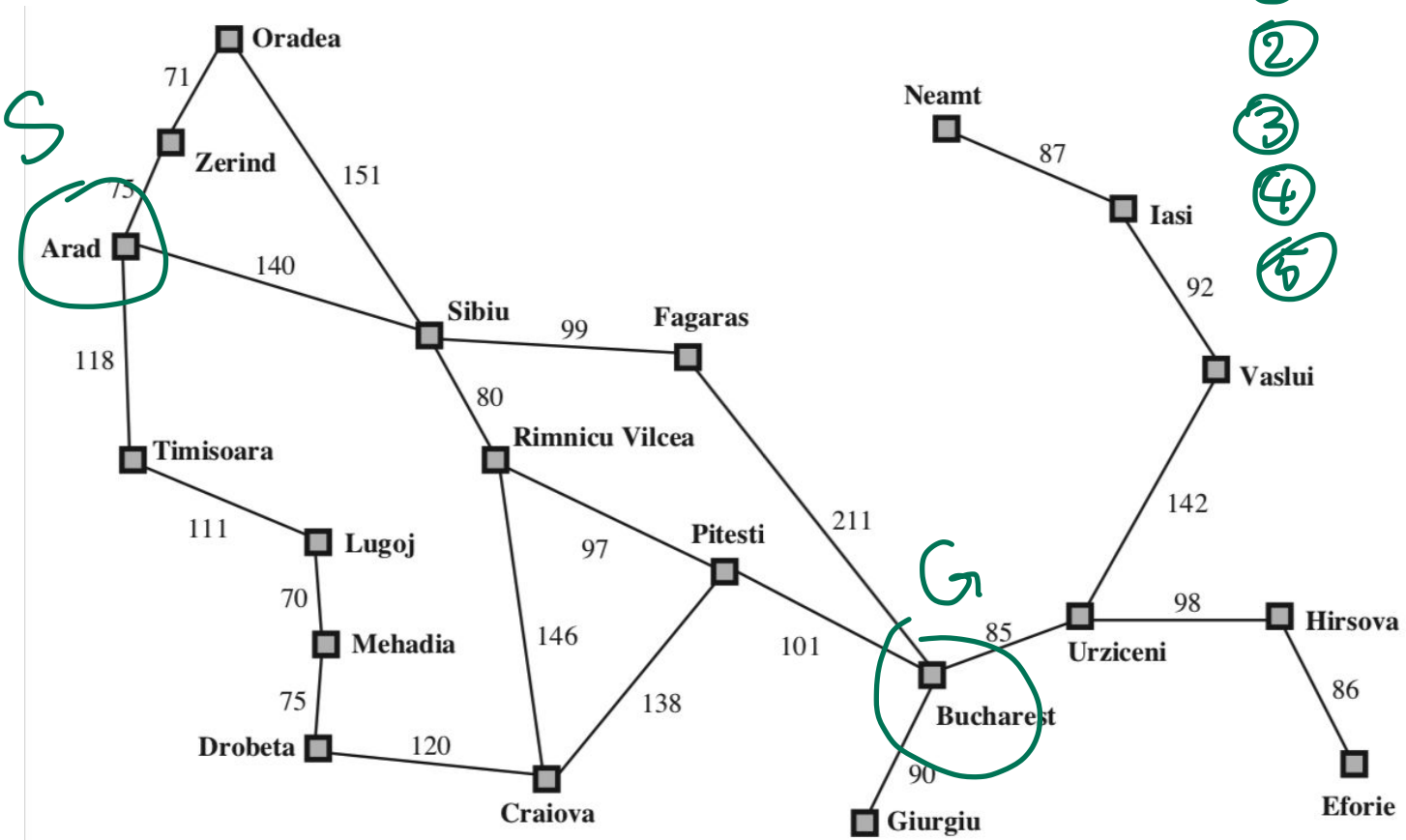
Step 2 Searching for a plan



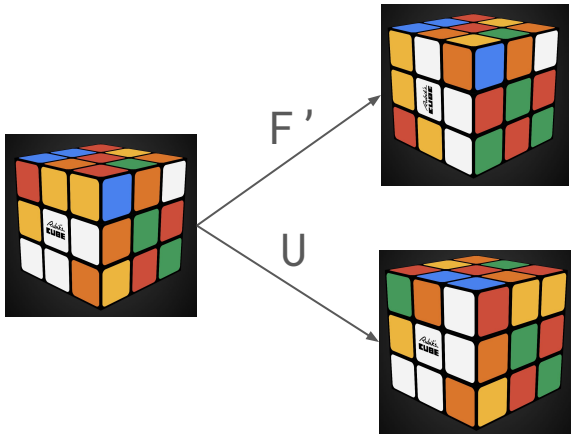
Example: Routing

Formulate this

1
2
3
4
5



Example: Rubik Cube



Formulate this

1

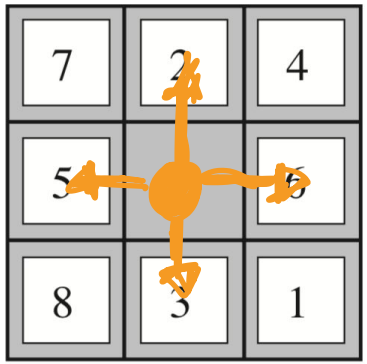
2

3

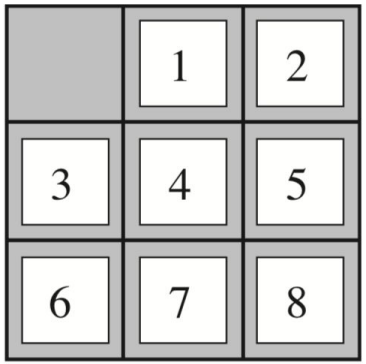
4

5

Example: 8-Puzzle



Start State



Goal State

Formulate this

1.
2.
3.
4.
5.

Graph Search Algorithm

Search

function GRAPH-SEARCH(*problem*) **returns** a solution, or failure

initialize the frontier using the initial state of *problem*

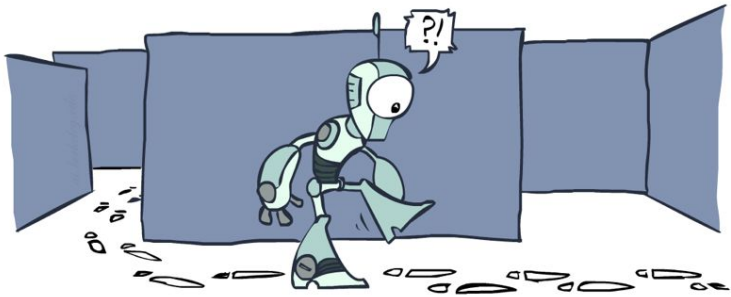
initialize the explored set to be empty

loop do

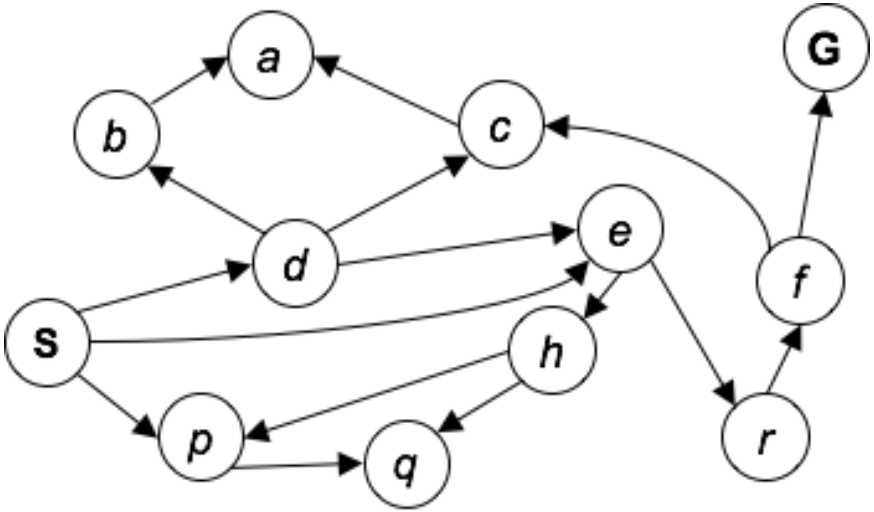
1. **if** the frontier is empty **then return** failure
2. choose a leaf node and remove it from the frontier
3. **if** the node contains a goal state **then return** the corresponding solution
4. ***add the node to the explored set***
5. expand the chosen node, adding the resulting nodes to the frontier
   only if not in the frontier or explored set

prevent moving back

possible options to explore.



Example: Abstract State-Space Graph



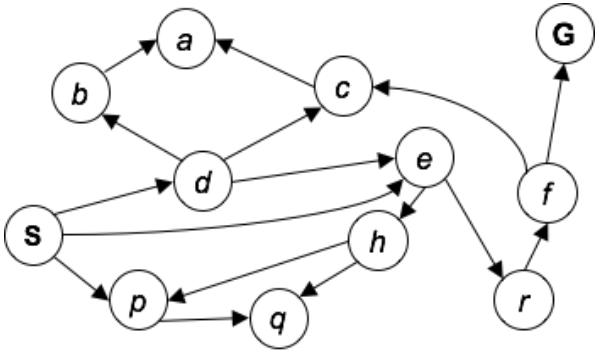
4. Uninformed-Search Strategies

Why we need a strategy?

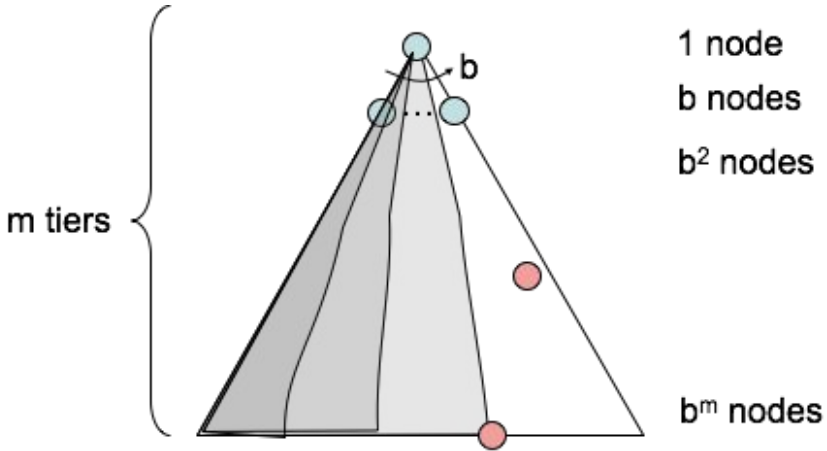
function GRAPH-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  initialize the explored set to be empty
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    add the node to the explored set
    expand the chosen node, adding the resulting nodes to the frontier
      only if not in the frontier or explored set

Strategy 1: Depth-First Search (DFS)

Frontier Search Tree Closed Set

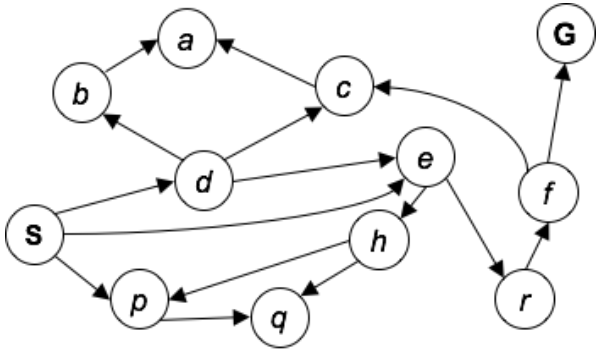


Depth-First Search (DFS) Evaluation

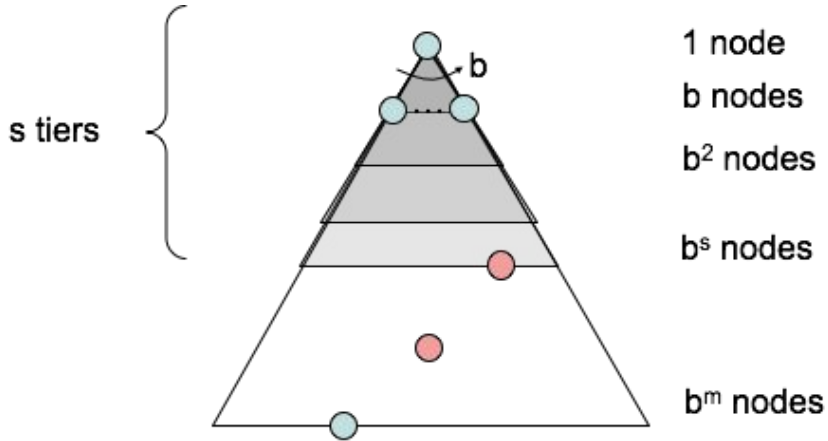


Strategy 2: Breadth-First Search (BFS)

Frontier Search Tree Closed Set



Breadth-First Search (BFS) Evaluation



Activity 3: DFS and BFS on our environment

https://editor.p5js.org/thanapon.nor/full/T8Hk1cSlK

Repeat the activity 2 again, but this time, use DFS and BFS strategy.

On a piece of paper,

6. Write the plans and their energy requirements that you discovered for DFS and BFS.

ITCS 451
section 2

Lecture 3

Thanapon Noraset
Faculty of ICT, Mahidol University

Adapted from AIMA by Stuart Russell and Peter Norvig, and UC Berkeley CS188 by Dan Klein and Pieter Abbeel (ai.berkeley.edu)

Feedback Week 2

1. Problem Formulation

2. Exploration

Announcements & Agenda

1. Checkpoint 3 on My Courses — Fri - Sun
2. Assignment 1 — now - Sun 4 Sep
3. Feedback 3
4. Office hours will be available soon. (TA is traveling)

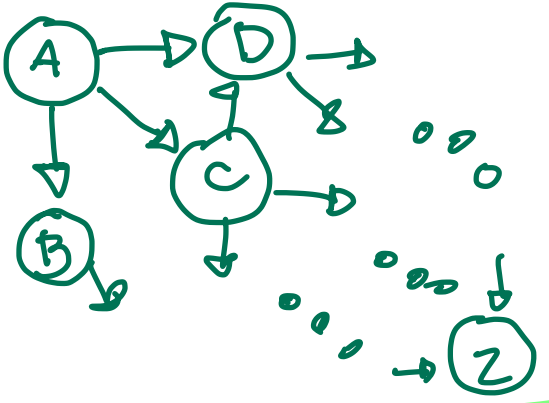
1. Uninformed Search Strategies
   a. Depth-First Search
   b. Breadth-First Search
   c. Iterative-Deepening Search
   d. Uniform-Cost Search
2. Informed Search Strategies
   a. Heuristic Function
   b. Greedy Search
   c. A* Search
   d. Admissibility and Consistency

Search Problem Formulation

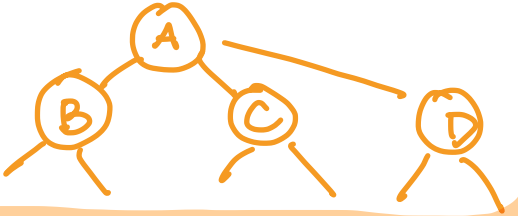
1. Initial state
2. Action Space
3. Transition func
4. Goal test func
5. Cost func

Search Algorithm

State space

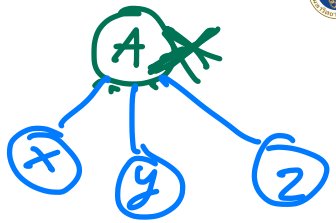


Search Tree



1. Uninformed-Search Strategies

Why we need a strategy?



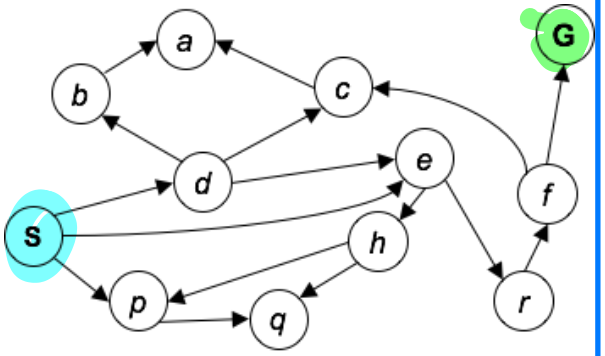
function GRAPH-SEARCH(*problem*) **returns** a solution, or failure
initialize the frontier using the initial state of *problem*
initialize the explored set to be empty
loop do
1. **if** the frontier is empty **then return** failure
2. choose a leaf node and remove it from the frontier
3. **if** the node contains a goal state **then return** the corresponding solution
4. ***add the node to the explored set***
5. expand the chosen node, adding the resulting nodes to the frontier
   only if not in the frontier or explored set

options to try

which one?

It depends on a search strategy

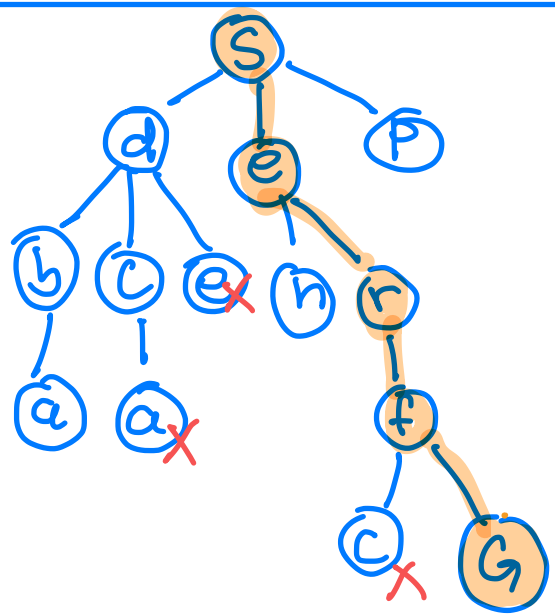
Strategy 1: Depth-First Search (DFS) "Highest-level"



Frontier

	Node	level
1	~~S~~	0
2	~~d~~	1
6	~~e~~	1
	p	1
3	~~b~~	2
5	~~c~~	2
4	~~a~~	3
	h	2
7	~~r~~	2
8	~~f~~	3
9	~~G~~ ✓✓✓	4

Search Tree



Explored ~~Closed~~ Set

State
S
d
b
a
c
e
r

Plan:

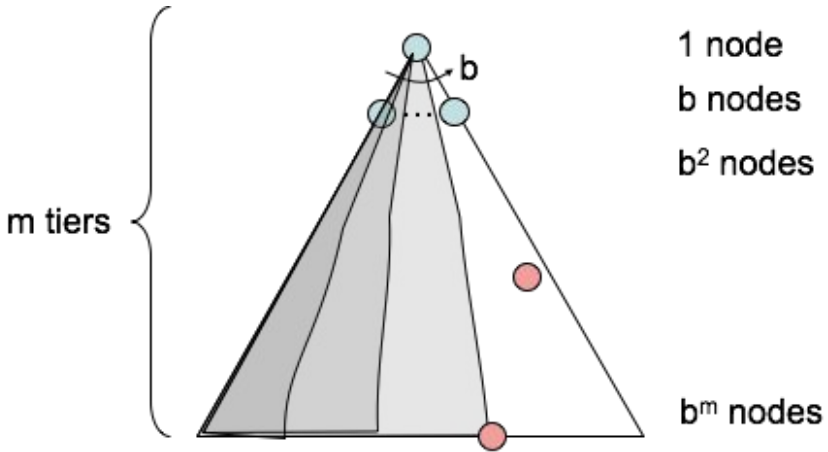
S → e → r → f → G

cost: 4 steps

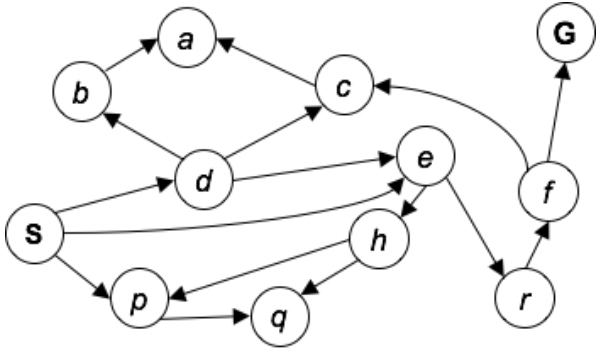
Depth-First Search (DFS) Evaluation

14 of 15

1. Completedness : always find a plan if there is one
   - m is finite, Yes
   - m is infinite, No
2. Optimality. : minimum cost plan
   - NO
3. Time complexity
   - $O(b^m)$
4. Space complexity
   - $O(bm)$



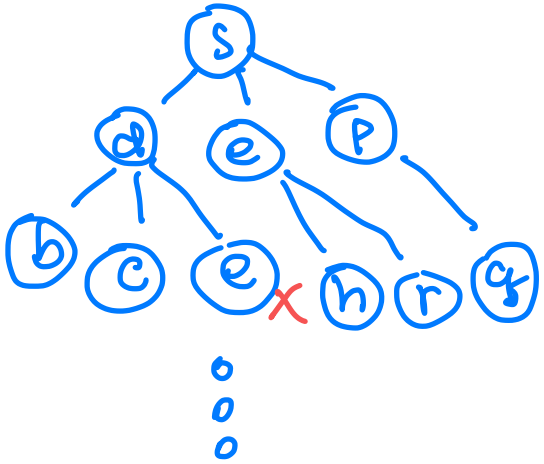
Strategy 2: Breadth-First Search (BFS) "Lowest-level."



Frontier

Node	level
~~S~~ (1)	0
~~d~~ (2)	1
~~e~~ (3)	1
~~p~~ (4)	1
b	2
c	2
h	2
r	2
q	2

Search Tree



Explored ~~Closed~~ Set

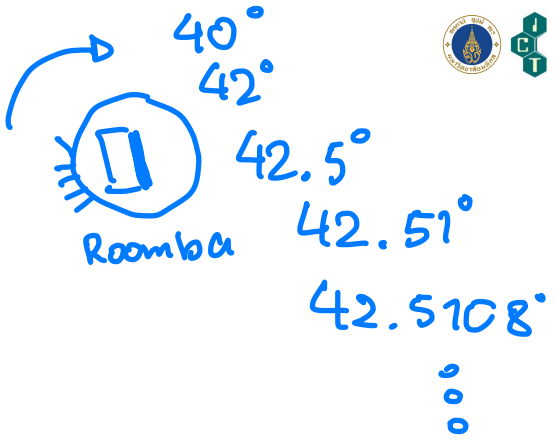
State
S
d
e
p

Plan:

S → e → r → f → G

Cost: 4 steps

Breadth-First Search (BFS) Evaluation

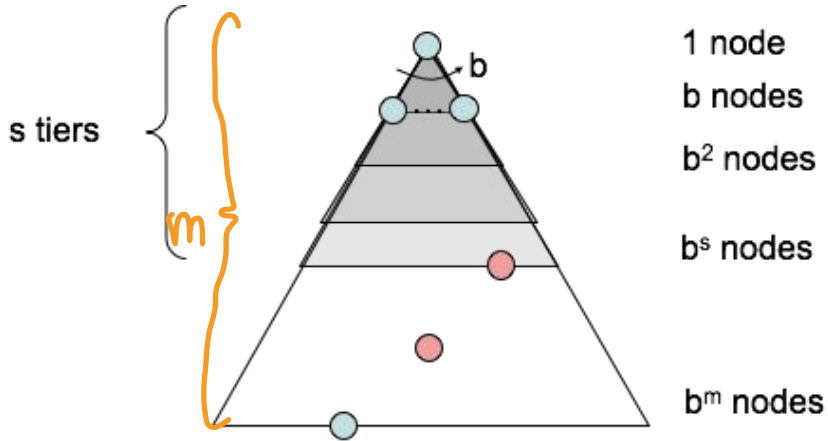


① Completedness
- b is finite, yes
- b is infinite, No

② Optimality
- yes * shortest (not always minimum cost)

③ Time complexity
- $O(b^s)$

④ Space complexity
- $O(b^s)$



Combining DFS and BFS

DFS

not optimal

memory : O(bm)

BFS

Optimal (shortest)

memory : $O(b^m)$

Iterative-Deepening Search

Strategy 3: Iterative-Deepening Search (IDS)

Strategy: Repeatedly run DFS but increasing the limit of depth each round

- Round 1: Run a DFS with depth limit 1, if no solution…
- Round 2: Run a DFS with depth limit 2, if no solution…
- Round 3: Run a DFS with depth limit 3, if no solution…
- …
- Round n: Run a DFS with depth limit n + step

Wasteful?

Strategy 3: Iterative-Deepening Search (IDS)

Depth limit

Frontier

Closed Set

Search Tree

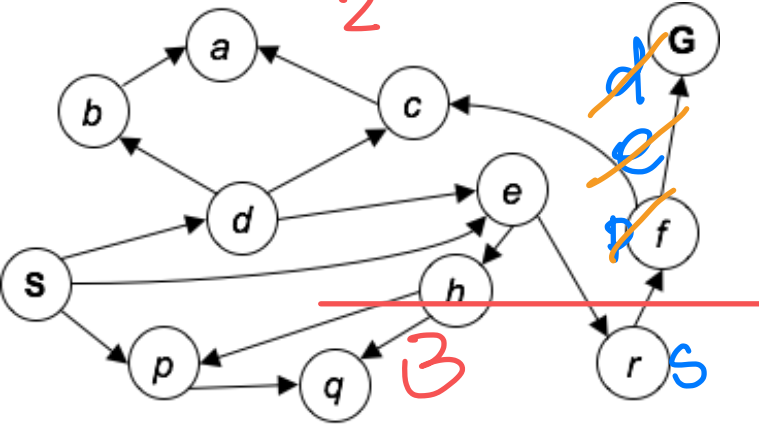
Explored set

Open Set

1

s 0

s



2

s 0

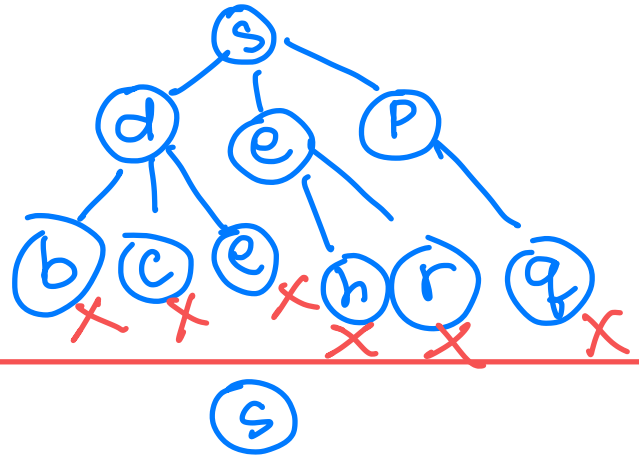
d 1

e 1

p 1

s 0

3



s

d

e

p

Iterative-Deepening Search (IDS) Evaluation

1. Completedness
   - BFS
2. Optimality
   - BFS ✓

3. Time complexity

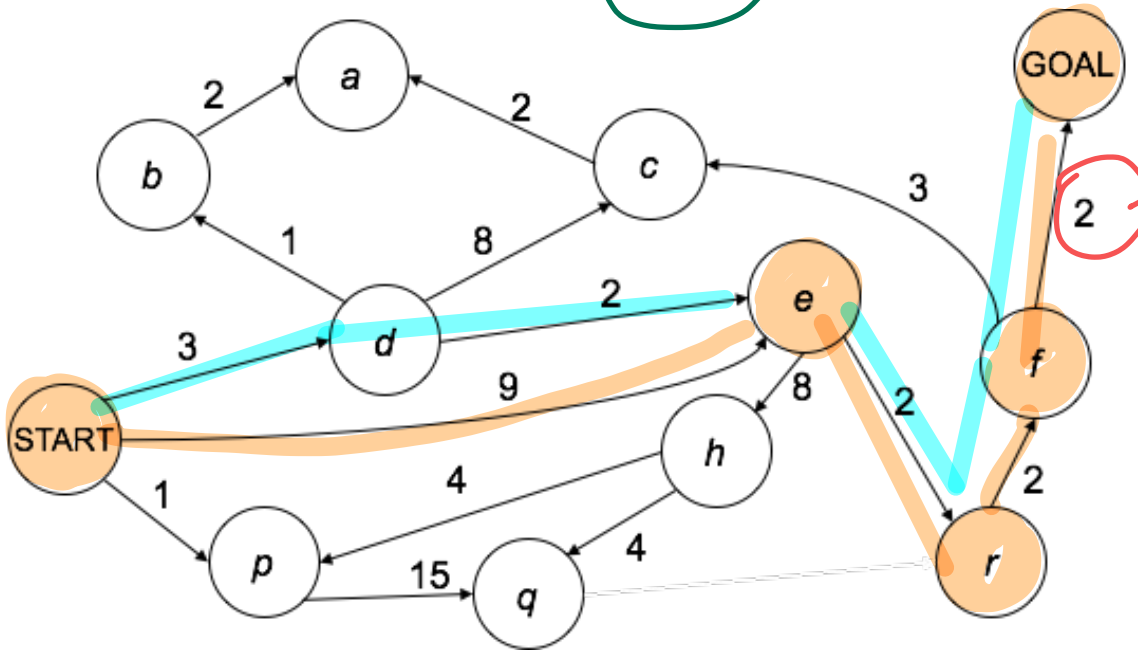
Homework

4. Memory complexity → DFS ✓

Path Cost Function: g(n)

node in a search tree

returns cost from start to n.



cost of the action

BFS Plan:

S → e → r → f → G

Cost: 15

Modified Graph Search Algorithm

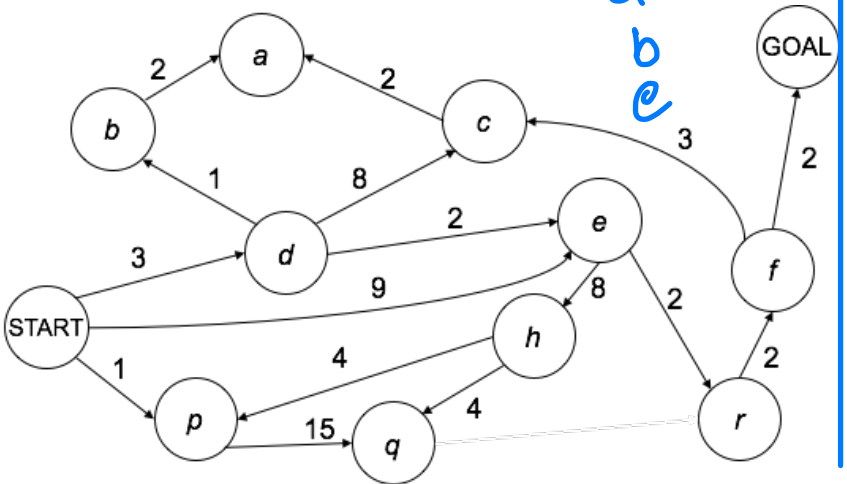
function GRAPH-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  initialize the explored set to be empty
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    add the node to the explored set
    expand the chosen node, adding the resulting nodes to the frontier
      ~~*only if not in the frontier or explored set*~~

① if in explored set, not add

② if in frontier, keep the better node.
that contains the same state.

Strategy 4: Uniform Cost Search (UCS)

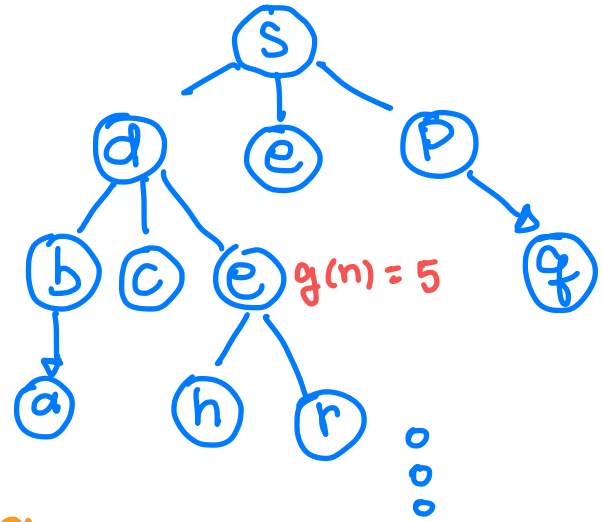
lowest g(n)



Explored ~~Closed~~ Set

state
S
p
d
b
e

Search Tree



Plan: S → d → e → r → f → G

Cost: 11

Frontier ~~Open Set~~

Node	g(n)
~~S~~	0
~~d~~	3
~~e~~	9
~~p~~	1
q	16
~~b~~	4
c	11
~~e~~	5
a	6
h	13
r	7

Uniform Cost Search (UCS) Evaluation

① Completedness : Yes

② Optimality : Yes (minimum cost)

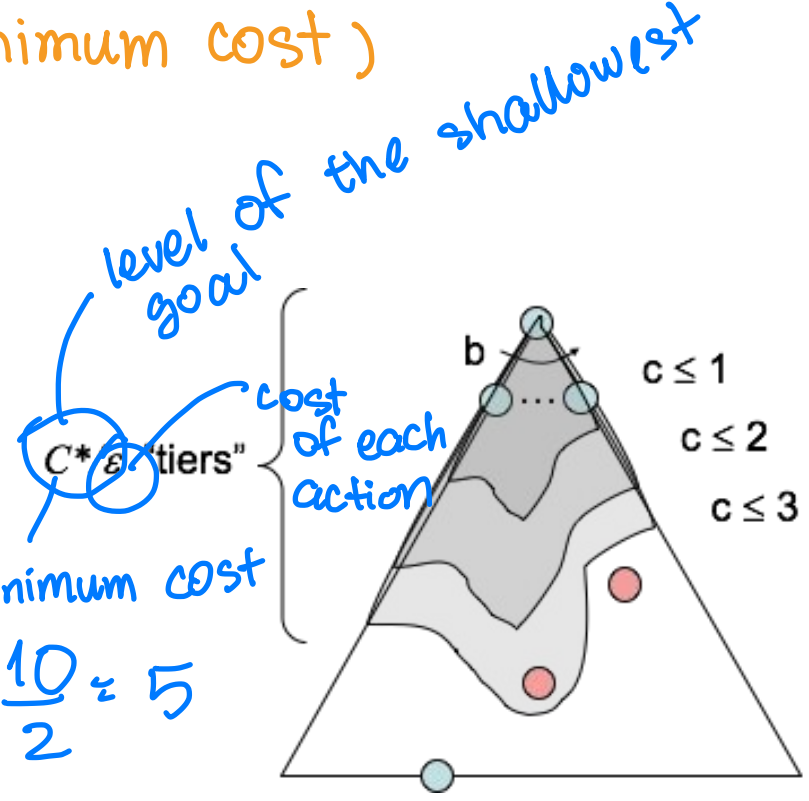
③ Time complexity : $O(b^{C^*/\varepsilon})$

④ Space complexity : $O(b^{C^*/\varepsilon})$

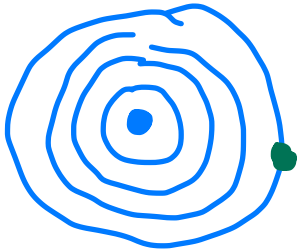
$C^* = 10$

$\varepsilon = 2$

levels $= \frac{10}{2} = 5$

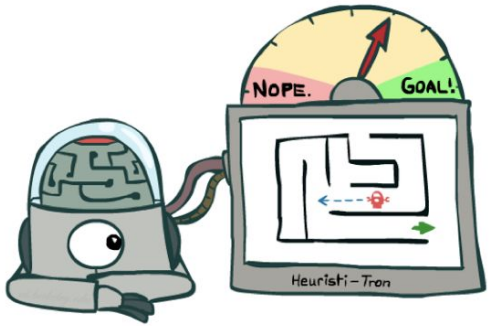


Uninformed Search



Informed Search

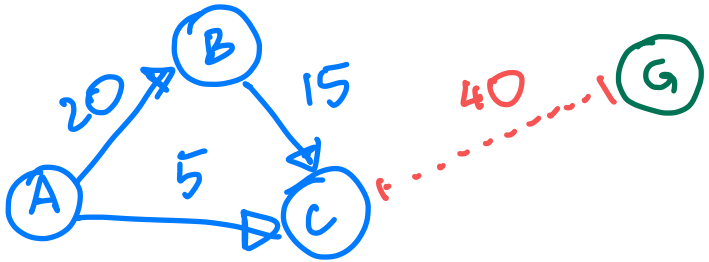
2. Informed Search Strategies



Heuristic Function: h(s)

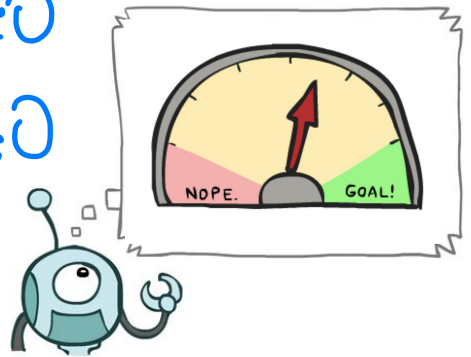
must be easy to compute.

estimate the cost from a state, s, to the goal state.

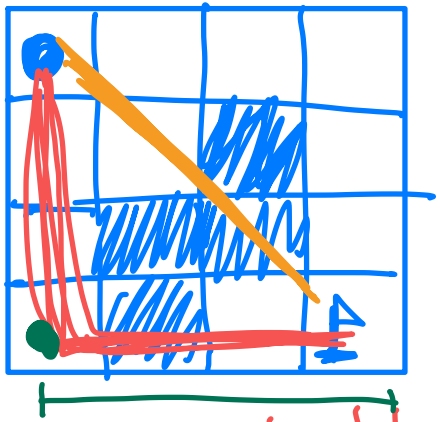


$g(A \to B \to C) = 35$, $h(C) = 40$

$g(A \to C) = 5$, $h(C) = 40$



Example: Heuristic Functions for Pathfinding



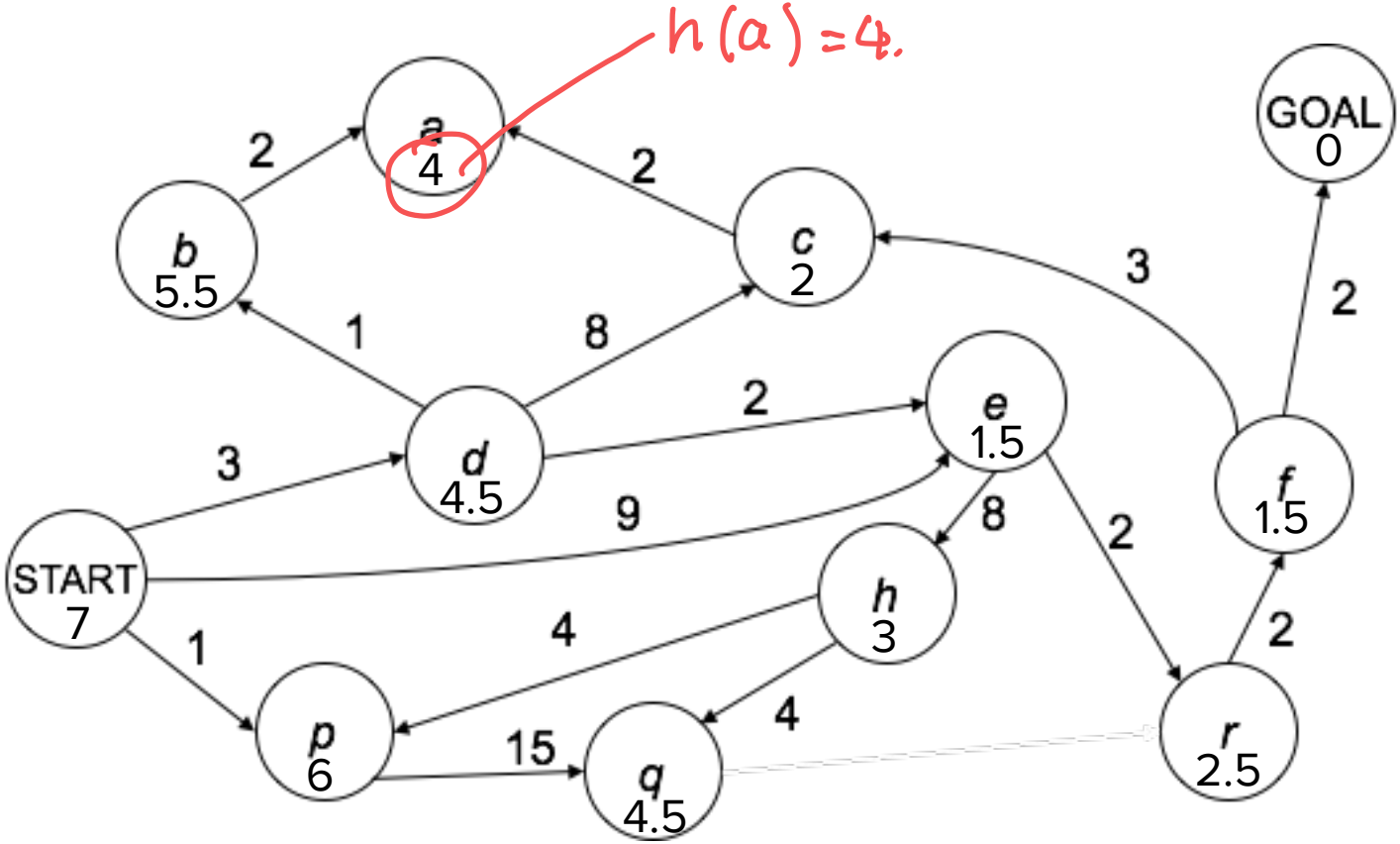
Euclidean distance

$$h(s) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

manhattan distance

$$h(s) = |x_1 - x_2| + |y_1 - y_2|$$

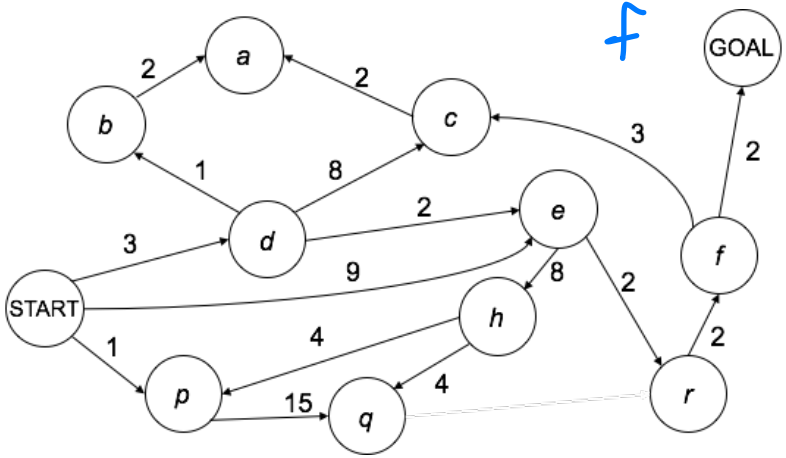
Example: Abstract State Space with Heuristic Values



Strategy 5: Greedy Best-First Search (GS)

Lowest h(s)

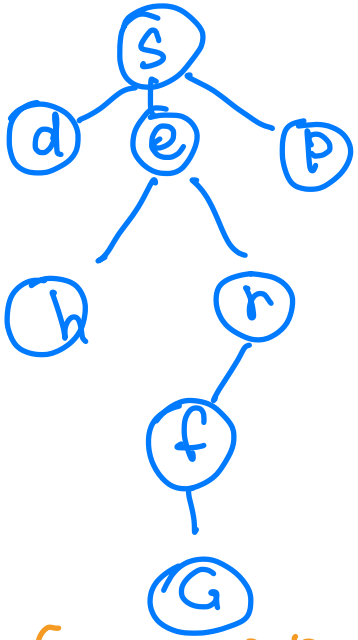
State	h(s)	State	h(s)
S	7	f	1.5
a	4	h	3
b	5.5	p	6
c	2	q	4.5
d	4.5	r	2.5
e	1.5	G	0



Explored ~~Closed~~ Set

State
S
e
r
f

Search Tree



Frontier ~~Open Set~~

	node	h(s)
1	~~S~~	7
	d	4.5
2	~~e~~	1.5
	p	6
	h	3
3	~~r~~	2.5
4	~~f~~	1.5
5	~~G~~ W	0

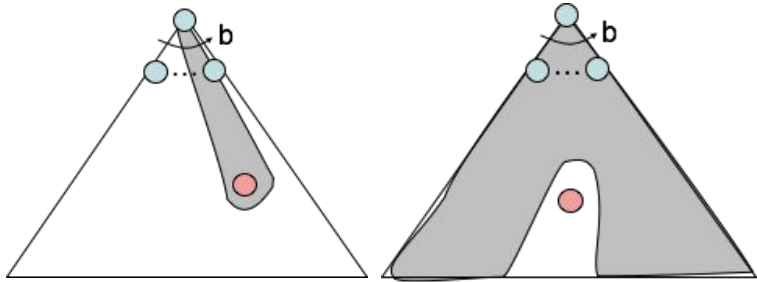
Plan: S → e → r → f → G

Cost 15

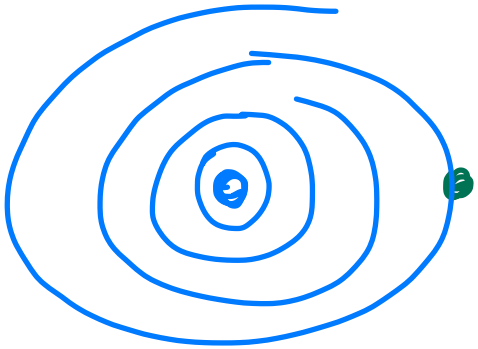
Greedy Best-First Search (GS) Evaluation

1. Completedness : NO * bad heuristic
2. Optimality : NO
3. Time Complexity : $O(b^m)$
4. Space Complexity : $O(b^m)$

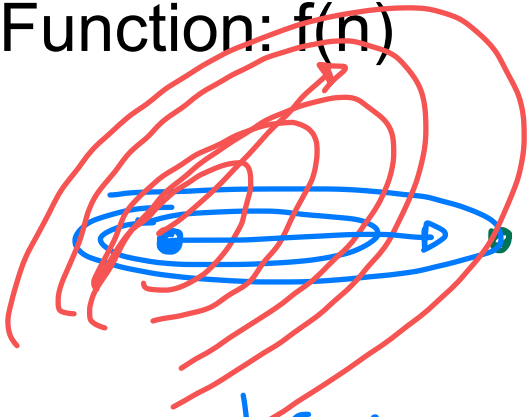
} usually better if we have good heuristic



Path Cost Function + Heuristic Function: f(n)



g(n)
(ucs)



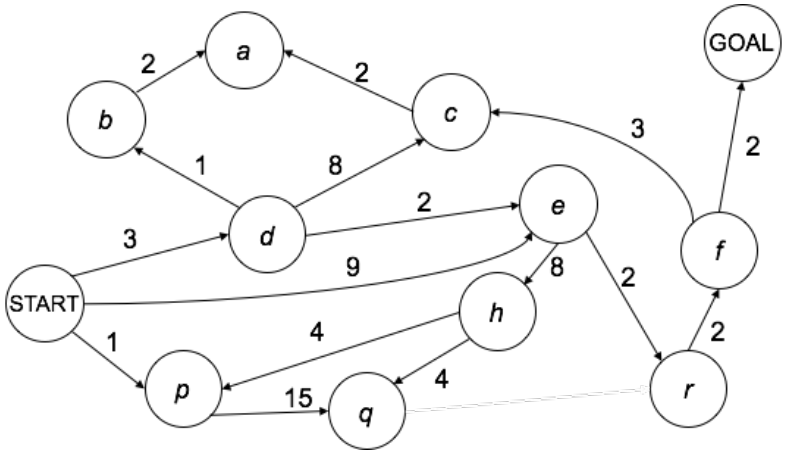
h(s)
(GS)

$$f(n) = g(n) + h(s \text{ of } n)$$

Strategy 6: A* Search

Lowest f(n)

State	h(s)	State	h(s)
S	7	f	1.5
a	4	h	3
b	5.5	p	6
c	2	q	4.5
d	4.5	r	2.5
e	1.5	G	0

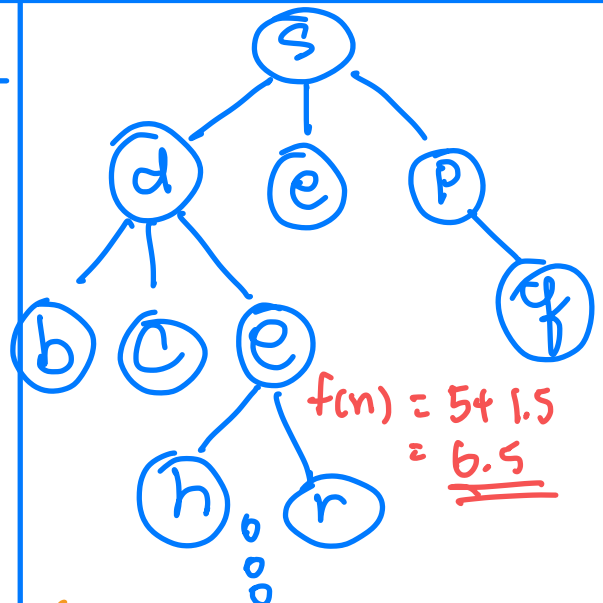


Explored ~~Closed~~ Set

state

S
p d e

Search Tree



Frontier ~~Open~~ Set

Node	f(n)
~~1 S~~	7
~~3 d~~	7.5
~~e~~	~~10.5~~
~~2 p~~	7
q	20.5
b	9.5
c	13
~~4 e~~	6.5
h	16
r	9.5

Plan S → d → e → r → f → G
Cost: 11

A* Evaluation

① Completedness : yes

② Optimality : yes

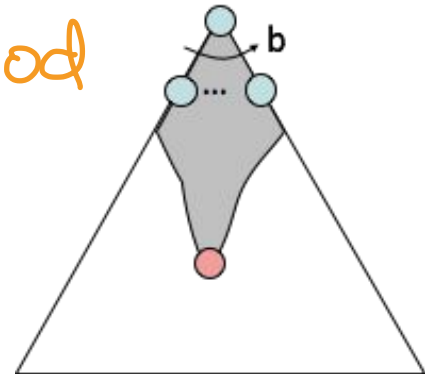
Conditions
- h(s) is admissible
- h(s) is consistent

⇨ next

③ Time Complexity: $O(b^{C^*/\epsilon})$

④ Space Complexity $O(b^{C^*/\epsilon})$

Usually better with a good h(s)

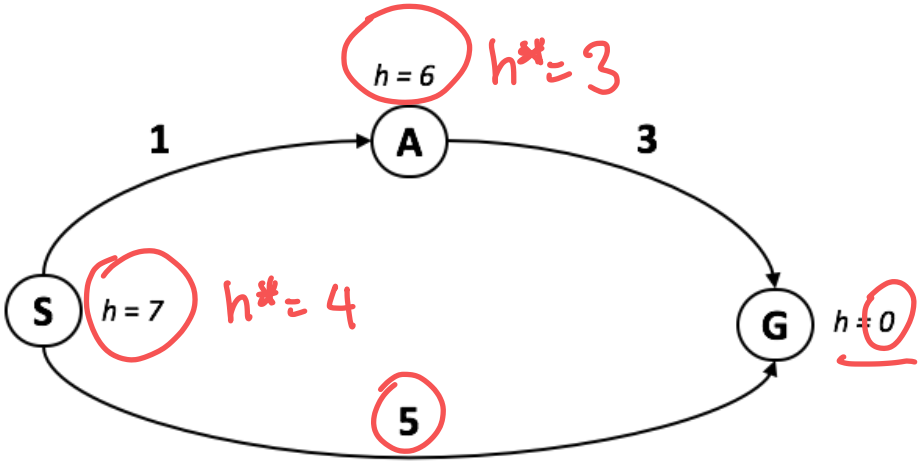


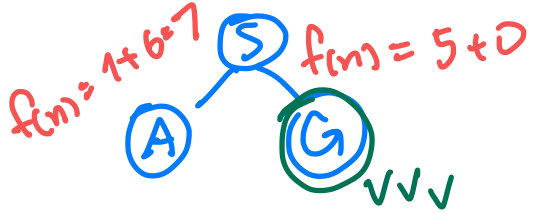
Feedback 3

- Heuristic functions
  - More detail today
- Evaluations of search strategies

	DFS	BFS	IDS	UCS	GS	A*
Completeness						
Optimality						
Time Complexity						
Space Complexity						

Optimality of A*: Admissibility





S → G , 5

Poll: Why do you think A* fails here?

h(s) is admissible, iff

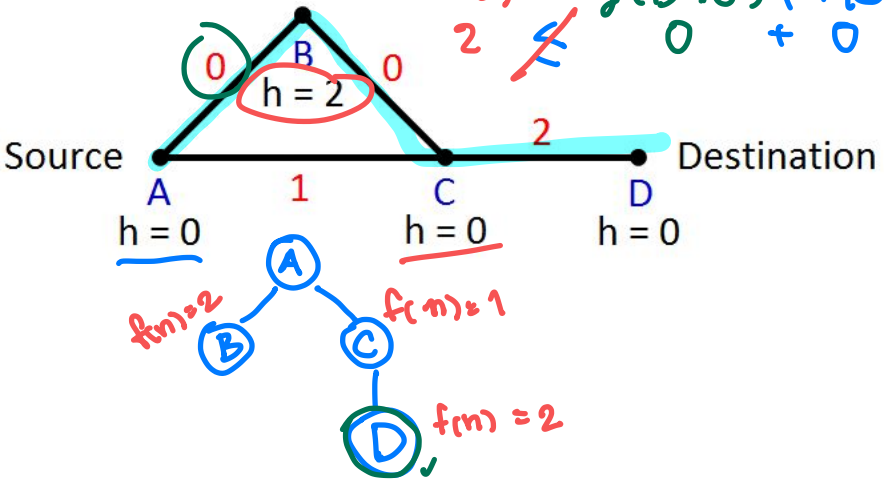
$$0 \leq h(s) \leq h^*(s)$$

for all S

The true minimum cost.

* heuristis must not overestimate the true cost.

Optimality of A*: Consistency



A → C → D, 3

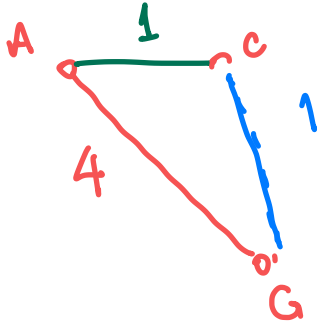
h(A) ≤ g(A→B) + h(B)

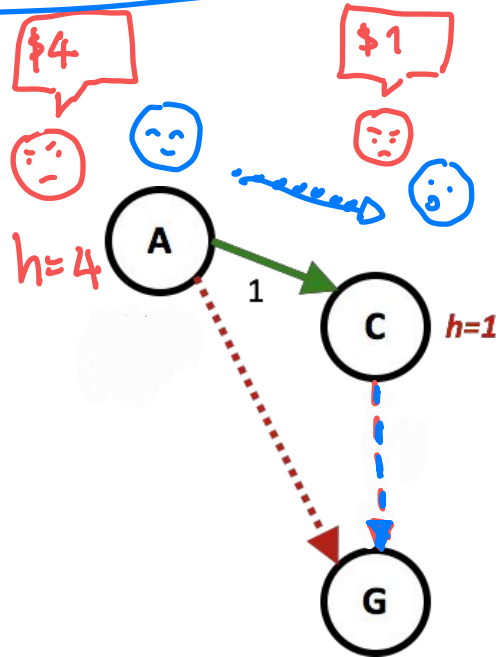
0 ≤ 0 + 2

h(s) is consistent, iff

$h(s) \leq g(s \to s') + h(s')$

for all s

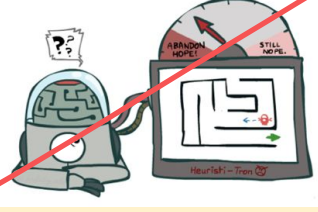


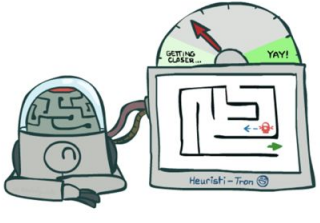


How to get admissible heuristic functions

Inadmissible (pessimistic) heuristics

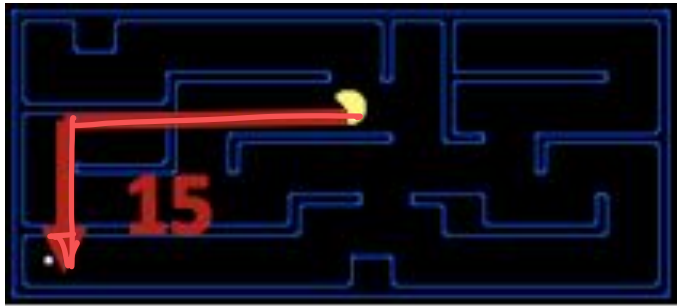
Admissible (optimistic) heuristics



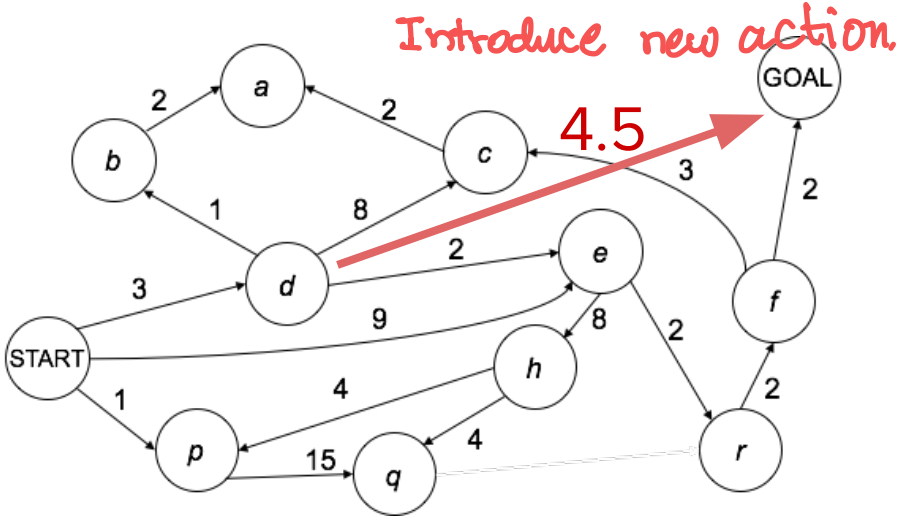


① Introduce new actions, ② Ignore rules

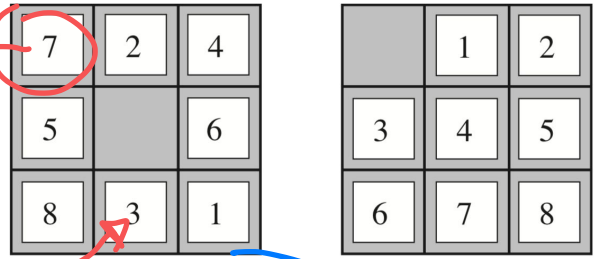
Example: Heuristic functions for pathfinding

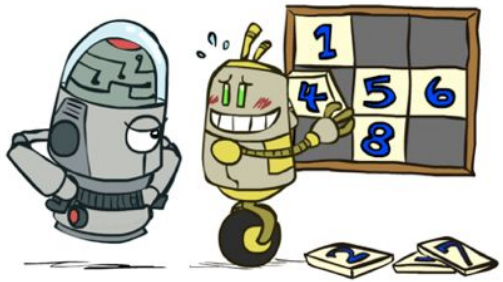


* ignore the wall.



Example: Heuristic functions for 8-puzzle (1)



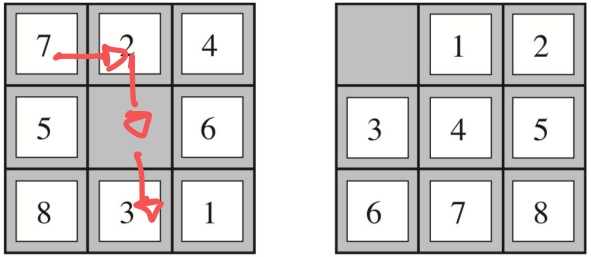


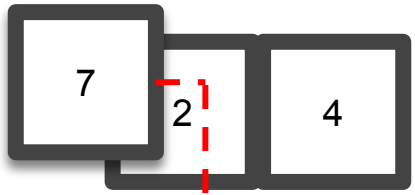
new action
* Take tile out and put anyway.

$h_1(S)$ = number incorrect tiles.

$h_1(\ ) = 8$

Example: Heuristic functions for 8-puzzle (2)





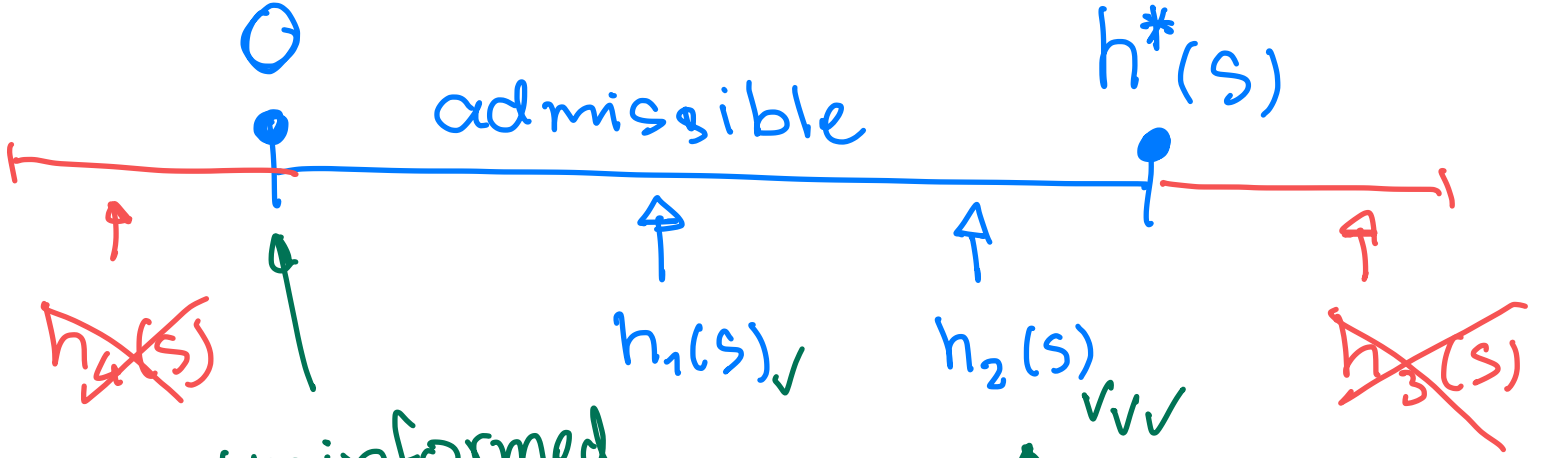
flyover
Hover

$h_2(\quad) = 3+1+2+2+3+2+3$

$= 16$

A ---- $h_2(S) = 16$ ---- $h_1(S) = 8$ ---- G

$h_5(s) = h_1(s)/5$



uninformed

*UCS

use less

time to search for a plan

$h_6(s) = h_2(s) + 2$

Example: Heuristic 1 vs Heuristic 2

	Search Cost (nodes generated)		
d	IDS	A* (h_1)	A* (h_2)
2	10	6	6
4	112	13	12
6	680	20	18
8	6384	39	25
10	47127	93	39
12	3644035	227	73
14	–	539	113
16	–	1301	211
18	–	3056	363
20	–	7276	676
22	–	18094	1219
24	–	39135	1641

Plan cost in step.

~ Iterations

Summary

Formulate Run Implement

Planning = Search Problem → Graph Search → A sequence of actions

Exploration Strategy: Choose the _____ nodes in the open set first

1 - DFS: Deepest → Lowest -1* level
2 - BFS: Shallowest → level
*3 - IDS runs DFS with depth limit several times
4 - UCS: Cheapest (path cost) → g(n)
5 - GS: Closest (heuristic) → h(s)
6 - A*: Lowest f-value → f(n)
  - Admissibility
  - Consistency

Priority → Priority Queue

2:20 pm

Lecture 4

Thanapon Noraset
Faculty of ICT, Mahidol University

Adapted from AIMA by Stuart Russell and Peter Norvig, and UC Berkeley CS188 by Dan Klein and Pieter Abbeel (ai.berkeley.edu)

Announcements & Agenda

1 Assignment 1 Sep 11

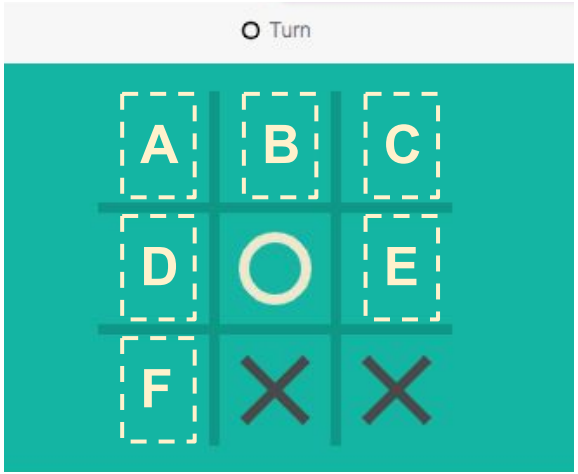
2. Checkpoint 4 Fri Sun

3. Feedback 4.

1. Competitive Environments
   a. Zero-sum games
   b. Settings of the environments
2. Adversarial Search Problem
   a. Optimal decisions
   b. Formulation
   c. Minimax Decision Algorithm
3. Improving Minimax Efficiency
   a. Alpha-Beta Pruning
   b. Depth-limited and Evaluation function

1. Competitive Environments

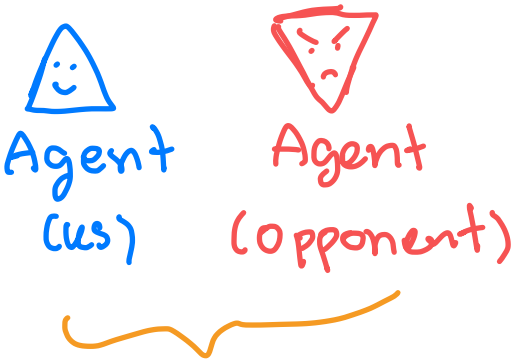
Which is the best “decision” for O?



① We can reason into the future in order to make the "best" move
* including what the opponent is going to do.

② unpredictable "sometimes".

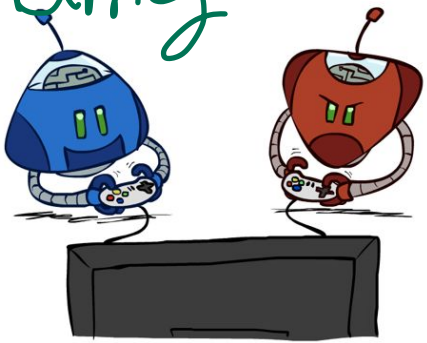
Multi-agent Competitive Environments



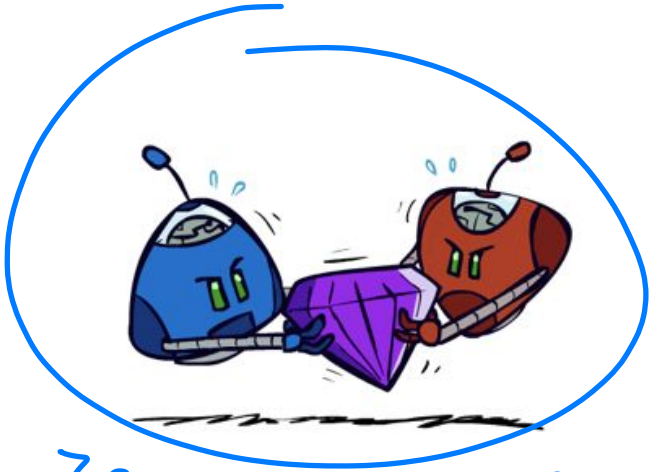
Simple form of competition

1. A fixed set of rules
2. well-defined ending

game → only zero-sum game



Zero-sum games



Zero-sum game

agents get opposite outcomes.

△ tries to maximize △ scores

▽ tries to minimize △ scores.

Different settings and Assumptions

Transition Function

$s' = T(s, a)$

Stochastic vs Deterministic

card game

tic-tac-toe

Number of Players

2 players vs more than 2

Information

Imperfect info

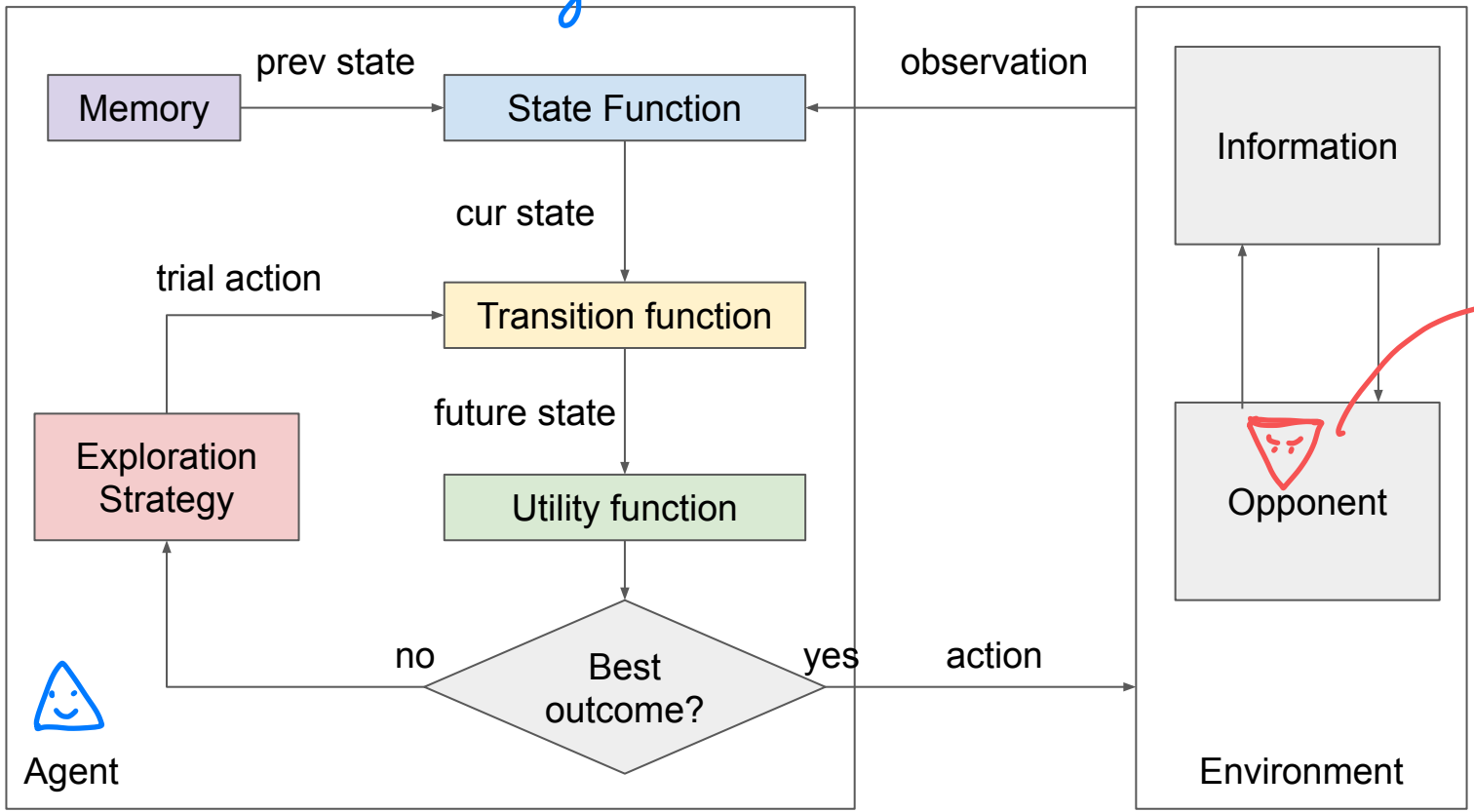
Perfect info

Competition

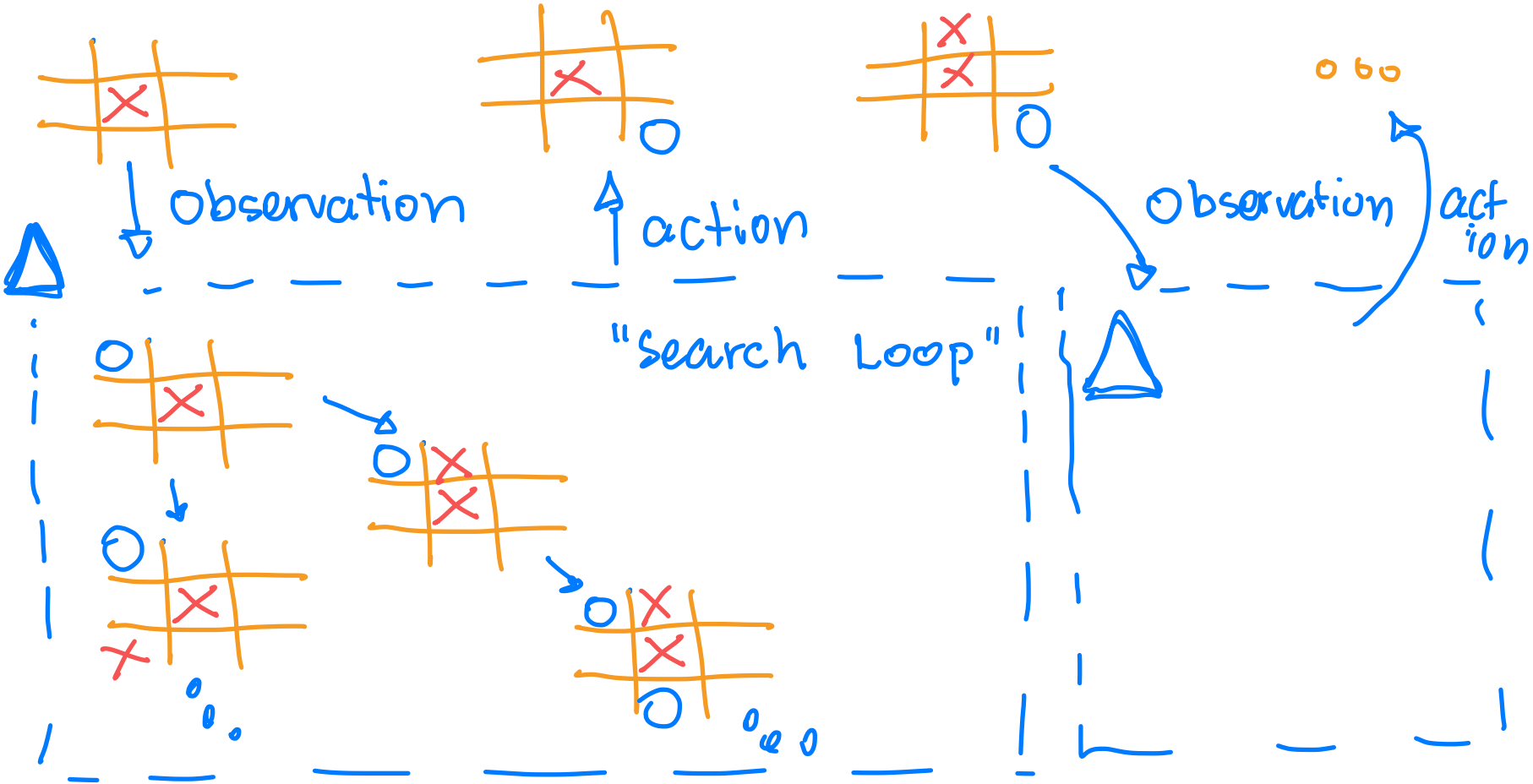
Zero-sum games

A Plan might not be valid for long...

we only search for one "best" move.



“Re-think” every turn



Optimal decision

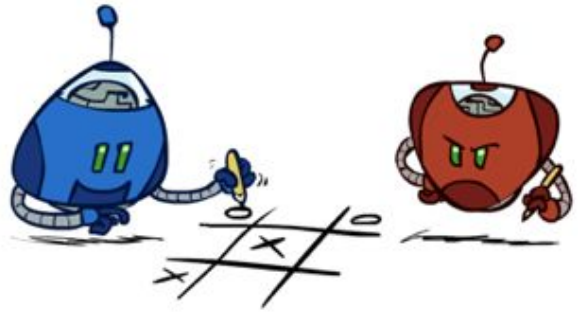
find the "best" move or action

2. Adversarial Search Problem

✓ "Formulation"

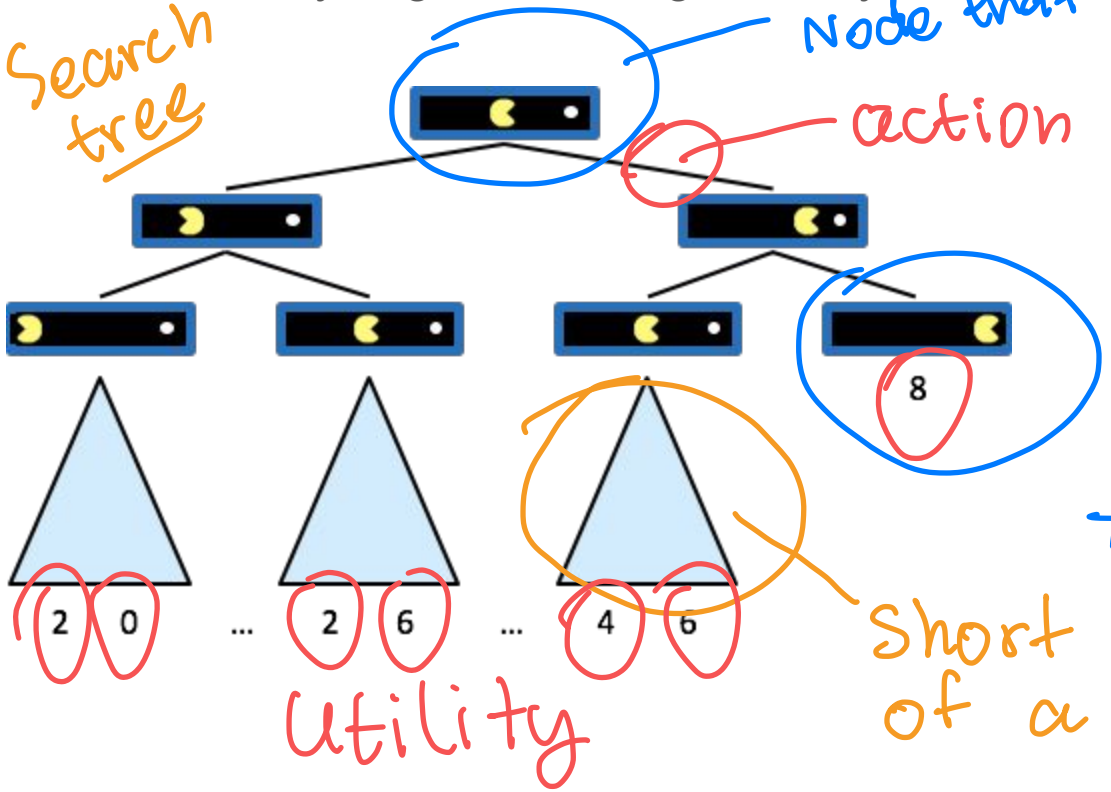
✓ "Algorithm".

▷ "Improving"



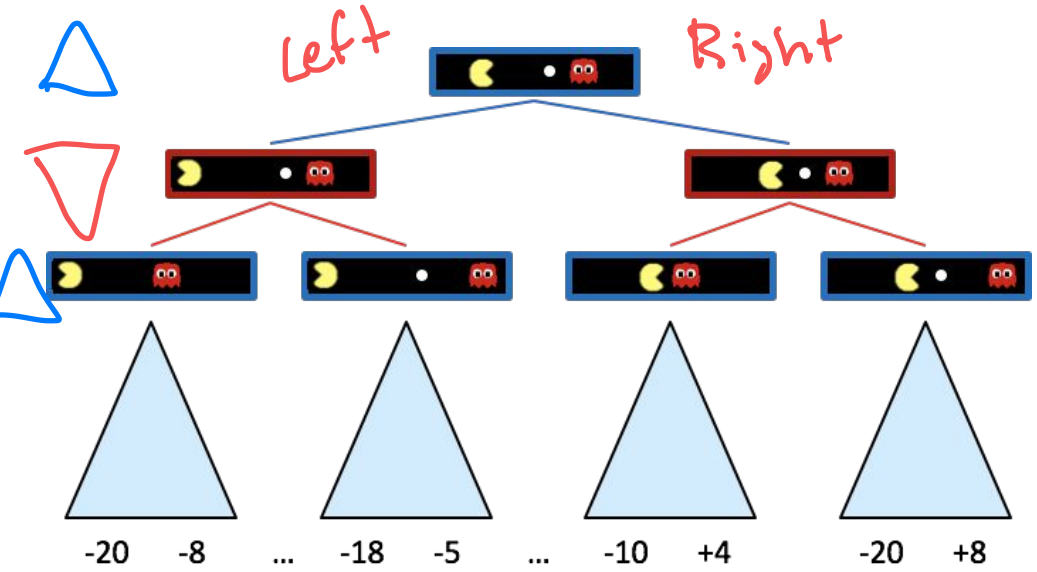
Optimal Decision: Single Agent

Poll: Would you go Left or Right?, Why?



Optimal Decision: Two Agents

Poll: Would you go Left or Right?, Why?



MAX Agent, MIN opponent

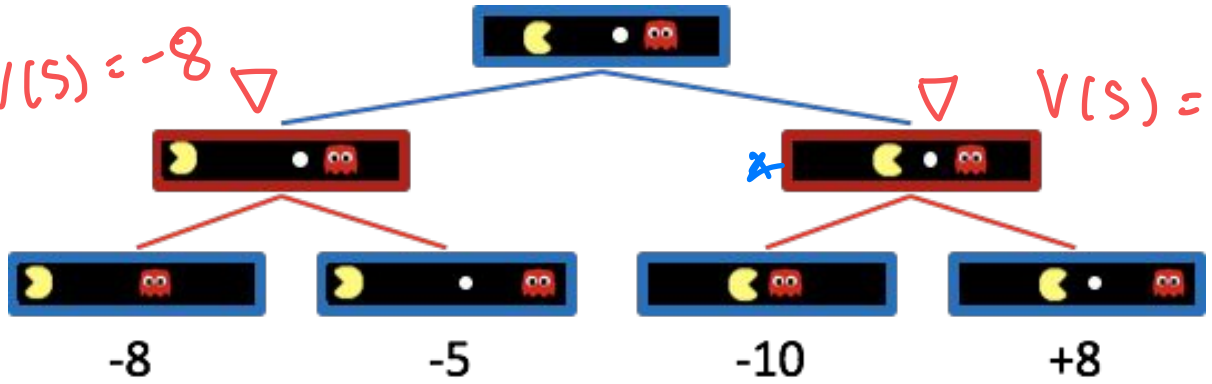
For non terminal state

$$\hat{V}(s) = \begin{cases} \Delta, \max V(s') \\ \nabla, \min V(s') \end{cases}$$

s' is a child of s

minimax value.

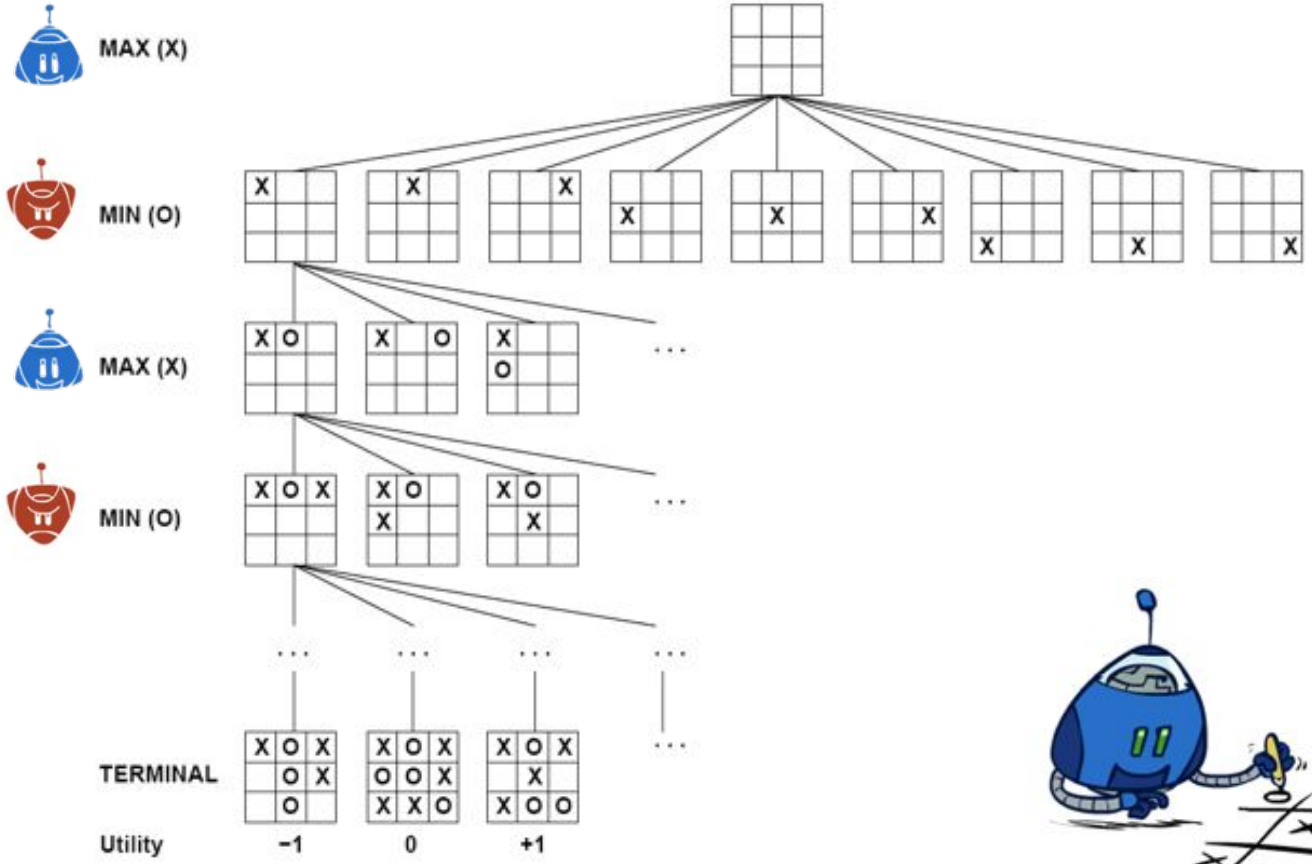
$V(s) = -8$ Δ

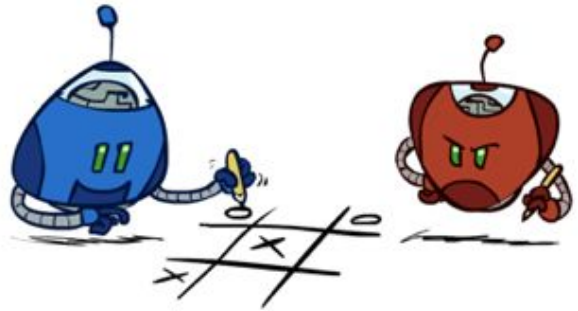


For terminal state

$$V(s) = \text{Utility}(s)$$

Example: Tic-Tac-Toe



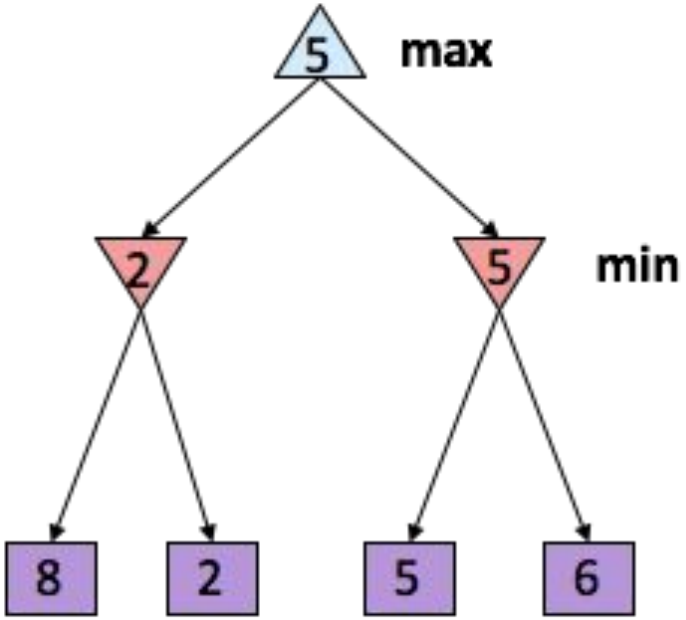


Example: Tic-Tac-Toe

Win: 1 point, Lose: -1 point, and Tie: 0 point

Utility.

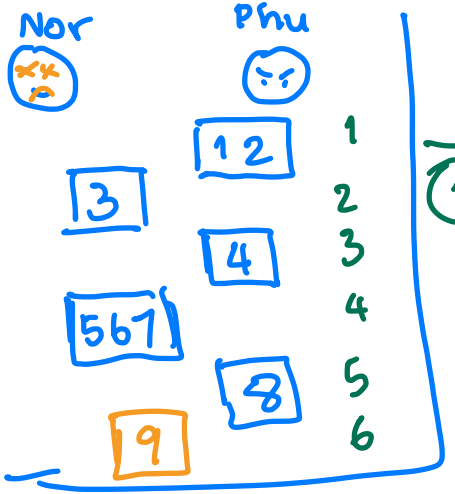
Minimax Values and Optimal Action



an action that leads to the highest-value child node.

Adversarial Search Formulation

1. Initial state (△'s turn)
2. Players △ ▽ } who is who?
3. Action space (a set of all valid moves)
4. Transition func
5. Terminal test isTerminal(s)
6. Utility Utility(terminal-state, player)



① Initial State

N, e.g. ① N = 0
② N = 2
③ N = 3

② Player. Phu ▽
Nor △

③ Action

$\{1, 2, 3 \mid a+N \leq 9\}$

④ Transition function

$N' = N + a$

⑤ Terminal test

$N == 9$

⑥ Utility

if player of the last state -1
else 1

The Minimax algorithm

First max node (initial state)

function MINIMAX-DECISION($state$) **returns** *an action*
  return $\arg\max_{a \in \text{ACTIONS}(s)}$ MIN-VALUE(RESULT($state, a$))

---

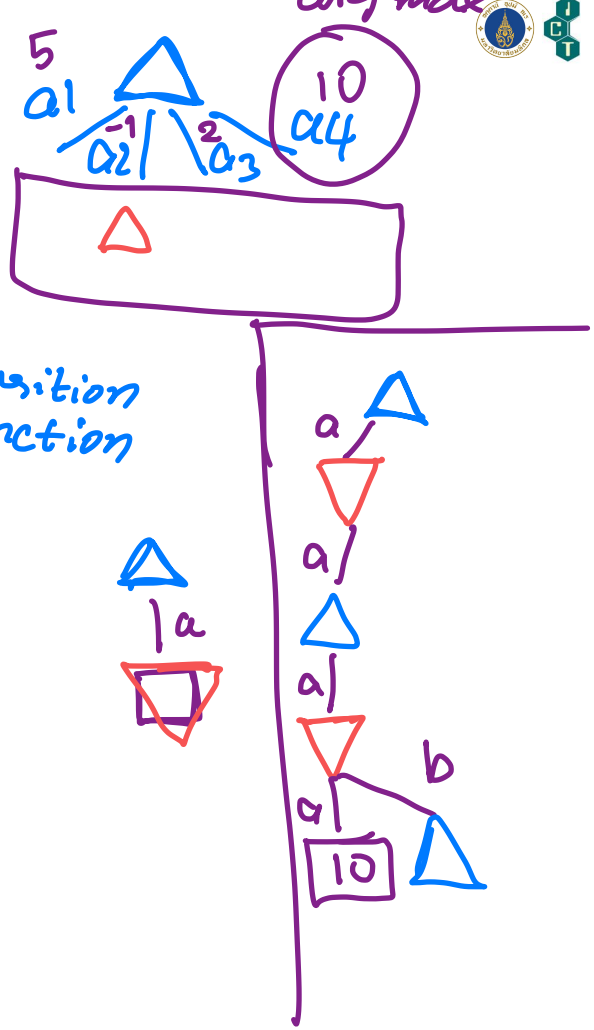
function MAX-VALUE($state$) **returns** *a utility value*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s, a)))$
  return v

10

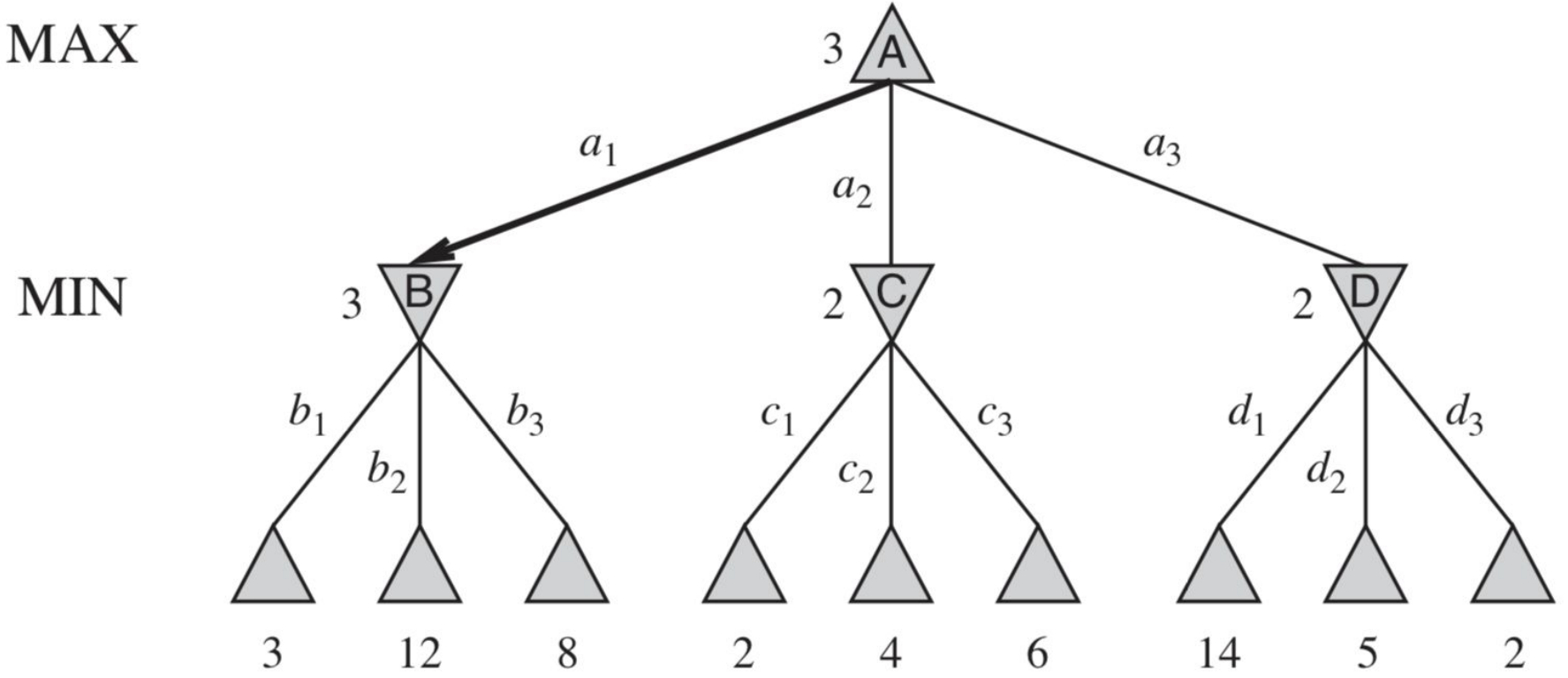
transition function

---

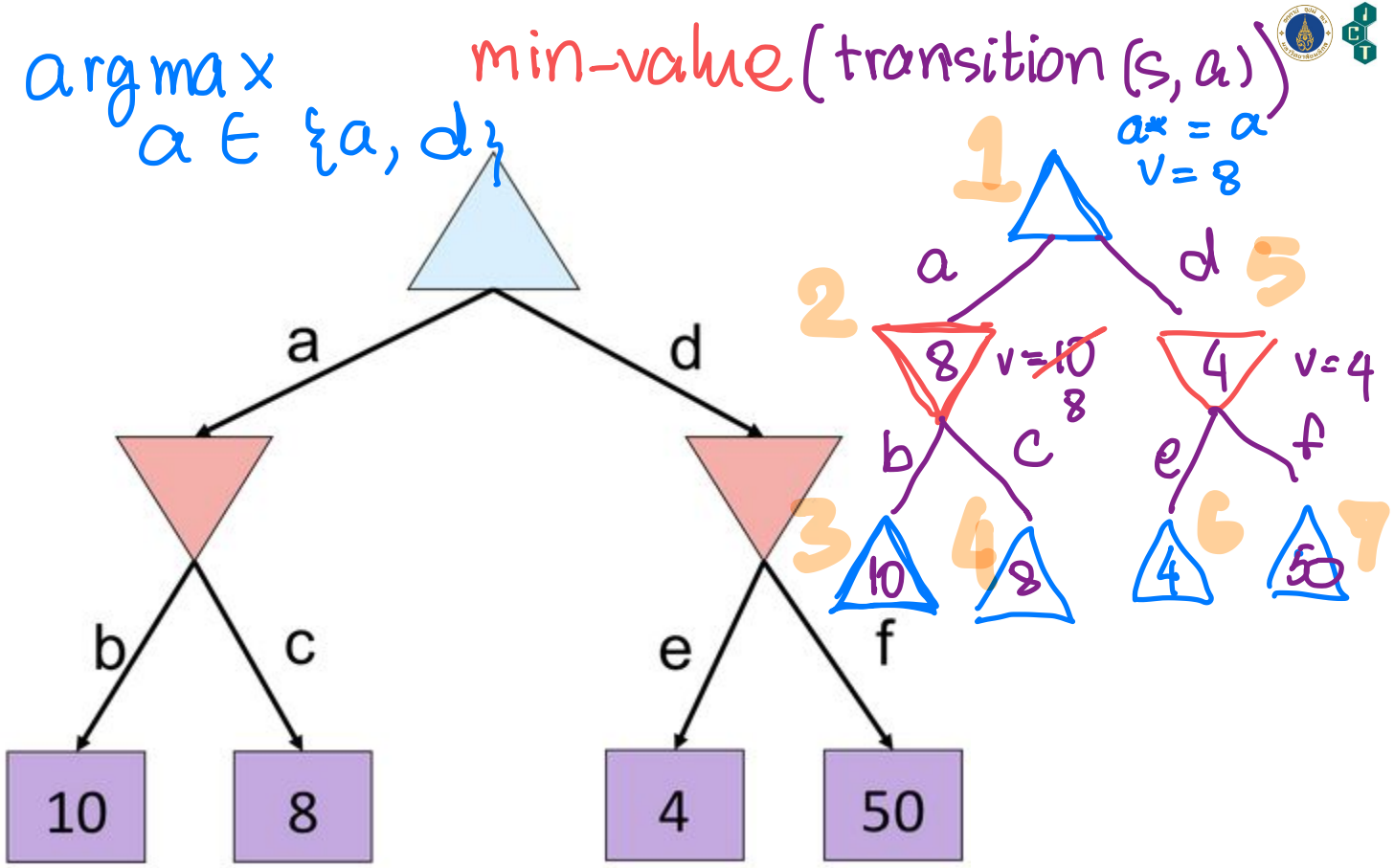
function MIN-VALUE($state$) **returns** *a utility value*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow \infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a)))$
  return v



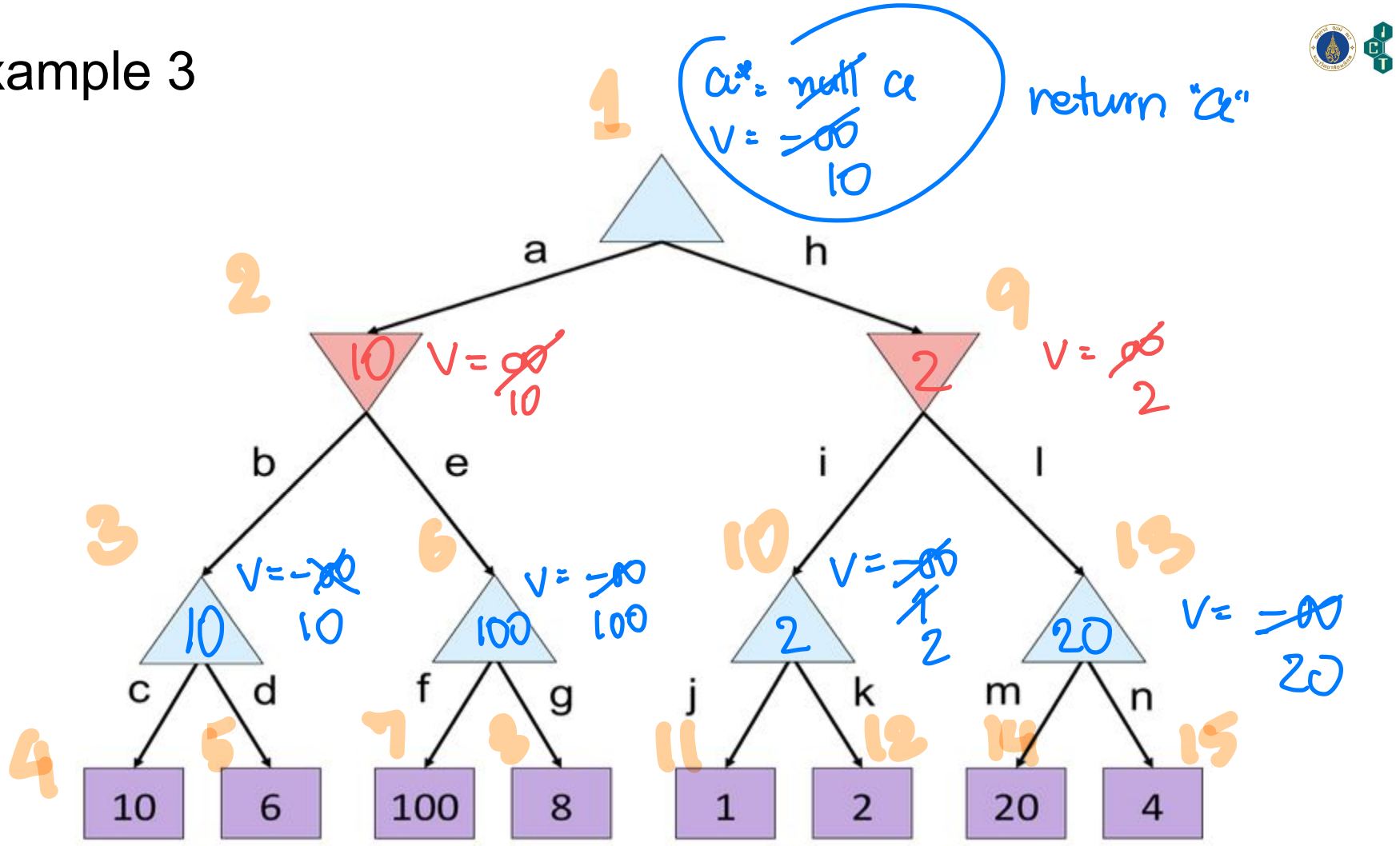
Example 1



Example 2


argmax
a ∈ {a, d}
min-value(transition(s, a))
a* = a
v = 8
a
d
b
c
e
f
10
8
4
50
1
2
3
4
5
6
7
8
v=10
8
4
v=4

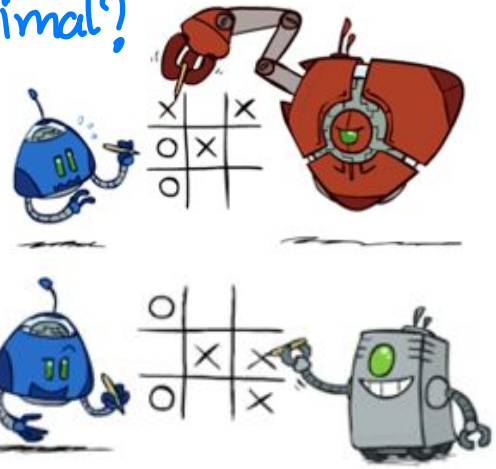
Example 3

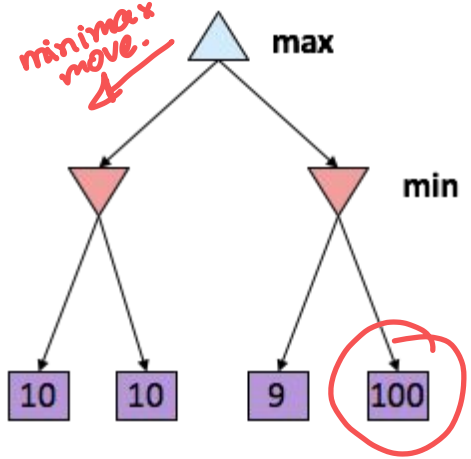

1
a*: null a
V = -∞
10
return "a"
a
h
2
10
V = -∞
10
9
2
V = -∞
2
b
e
i
l
3
10
V = -∞
10
6
100
V = -∞
100
10
2
V = -∞
2
13
20
V = -∞
20
c
d
f
g
j
k
m
n
4
5
7
8
11
12
14
15
10
6
100
8
1
2
20
4

Minimax Properties

1. is minimax search optimal?

↳ yes → only when the opponent is optimal.





2. Time Complexity

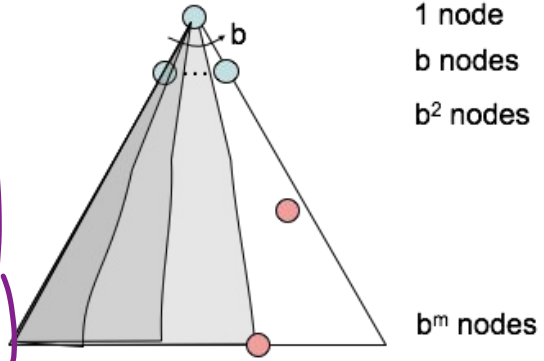
$O(b^m)$

3. Space Complexity

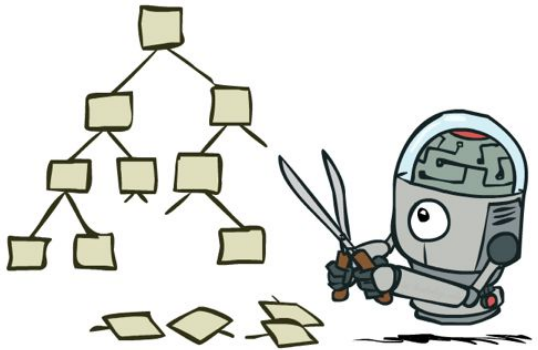
$O(bm)$

Chess
$b \approx 35$
$m \approx 100$

Go
$b \approx 250$
$m \approx 210$

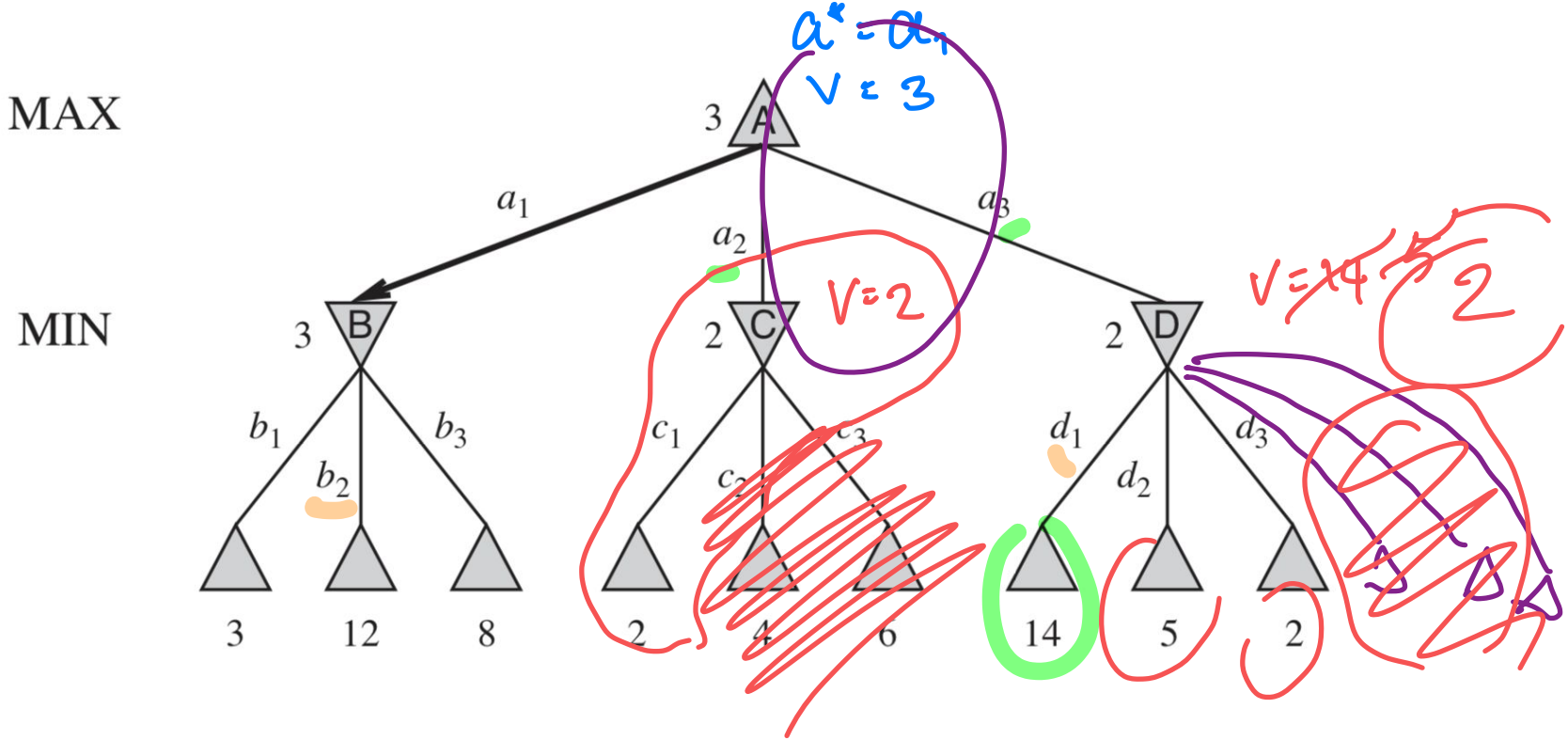


3. Improving Efficiency



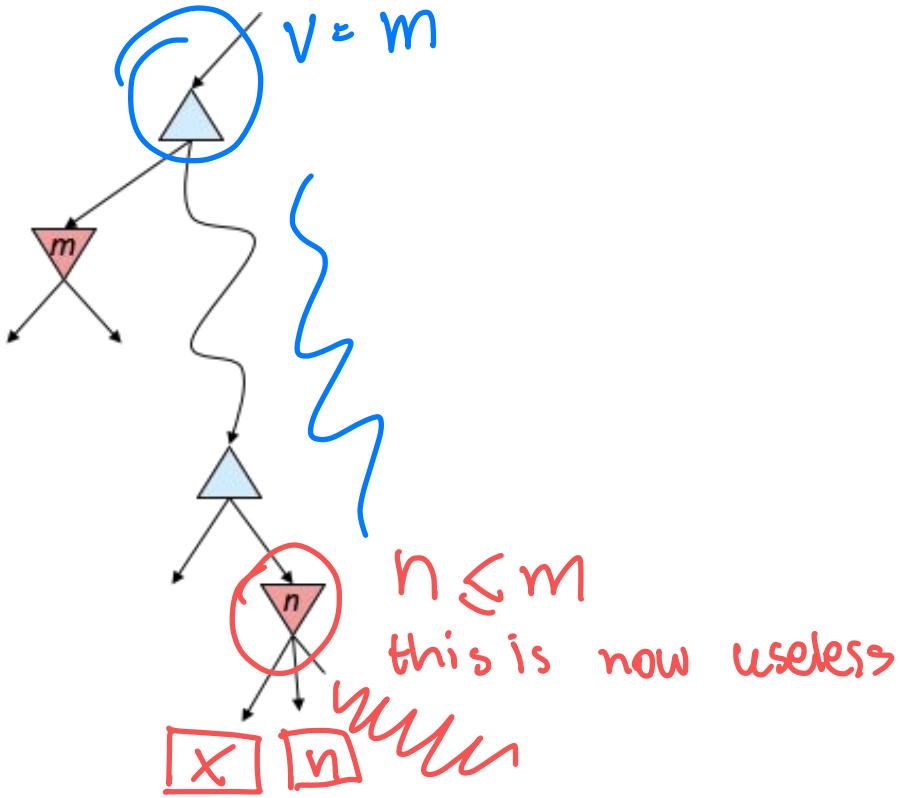
Minimax explores bad actions

Can you spot the unnecessary exploration?

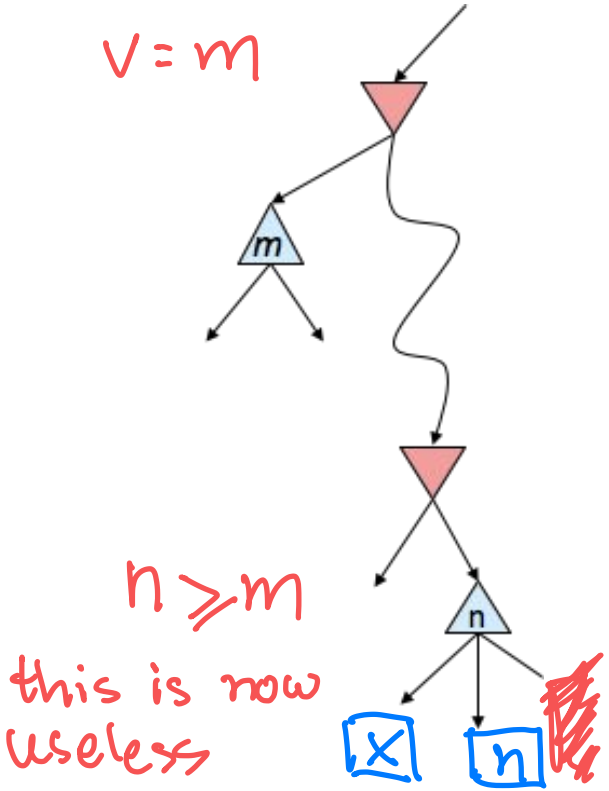


Alpha-Beta Pruning

Min case



Max case



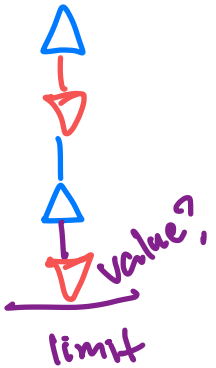
The Alpha-Beta Search algorithm (Modified Minimax)

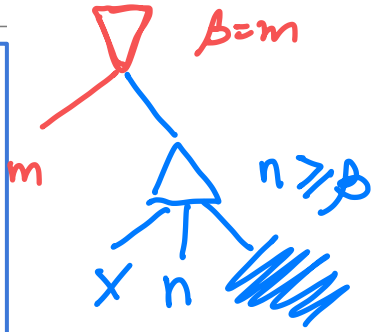
alpha beta

function ALPHA-BETA-SEARCH($state$) **returns** an action
  $v \leftarrow$ MAX-VALUE($state, -\infty, +\infty$)
  return the $action$ in ACTIONS($state$) with value v

---

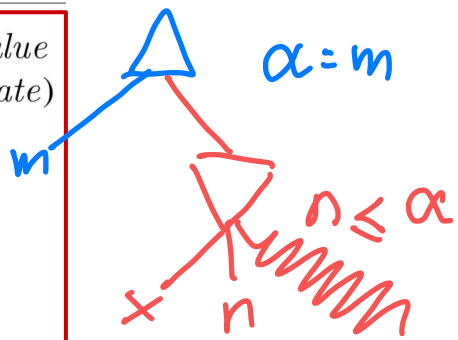
function MAX-VALUE($state, \alpha, \beta$) **returns** a $utility$ $value$
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  if depth-level ≥ limit then return ?
  $v \leftarrow -\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v



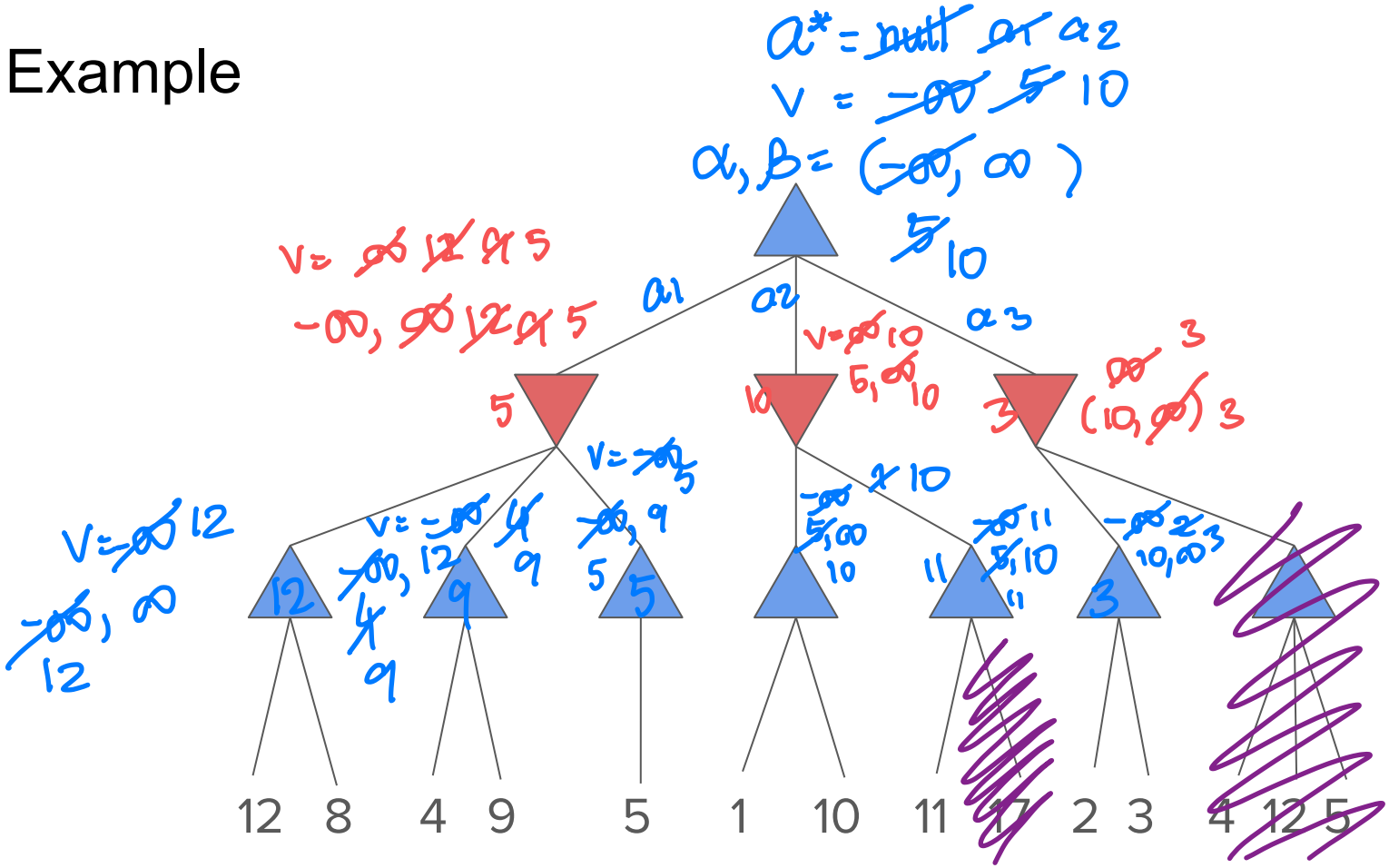


---

function MIN-VALUE($state, \alpha, \beta$) **returns** a $utility$ $value$
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v



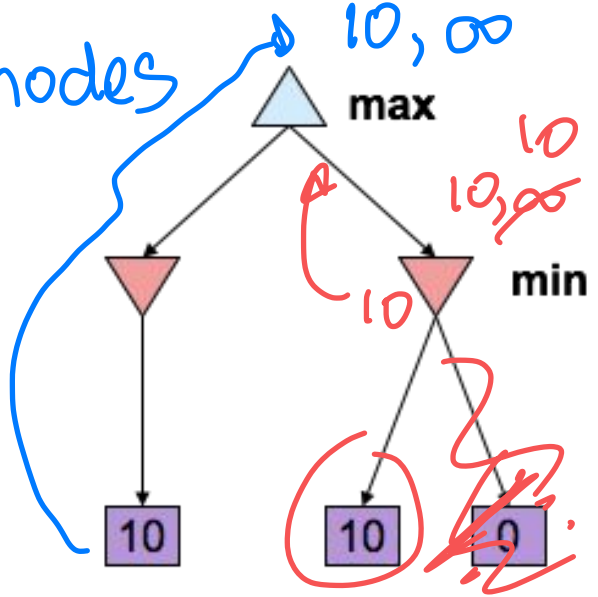
Example



Alpha-Beta Search Properties

1. Return the same action as minimax search.

2. The values of non-root nodes might not be the minimax values.

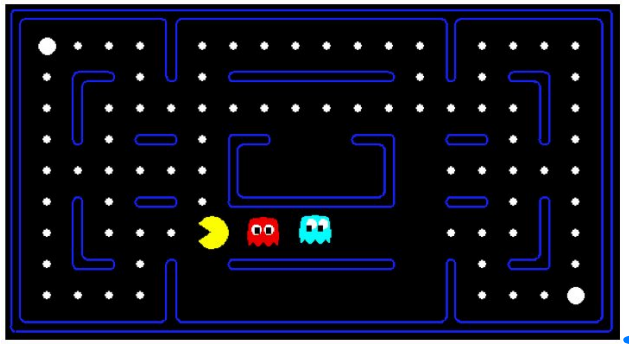


We need to be faster → Use heuristics!

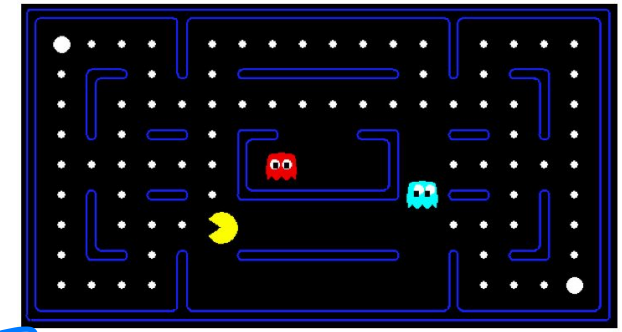
eval(B) > eval(A) > eval(C) = eval(D)

Poll: Sort the heuristic values
e.g. A > B > C > D

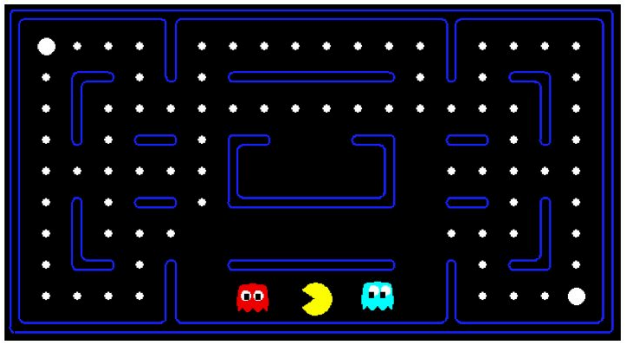
A



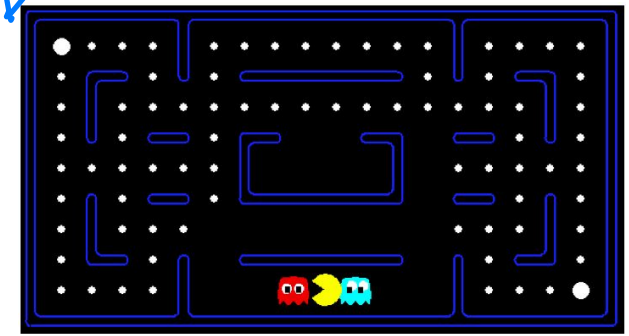
B



C

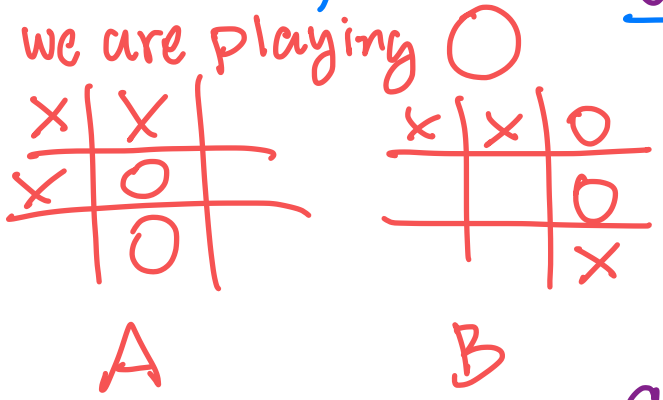


D



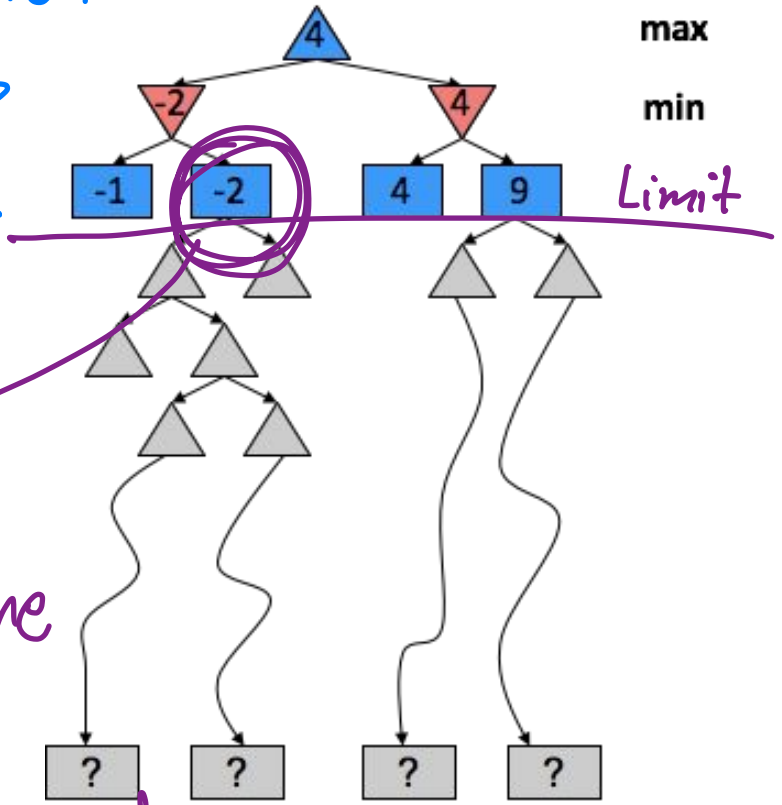
Depth-Limited Search + Evaluation Function

run minmax or alpha-beta search but if the depth-level reaches a limit, return eval(s) value.



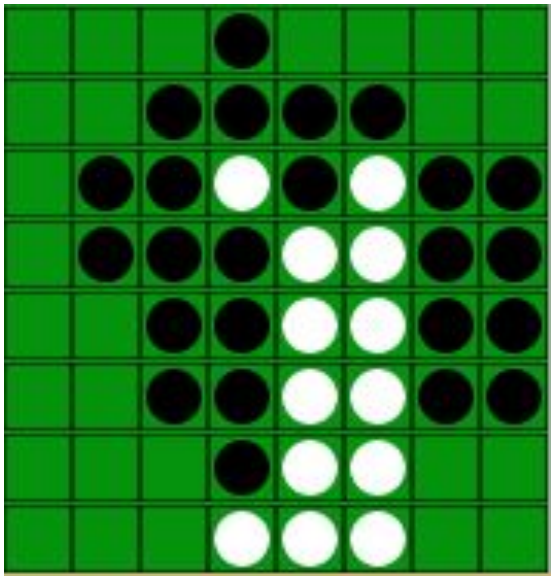
eval(A) < eval(B)

we need to approximate the utility of non-terminal nodes



Example: Othello (Reversi)

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$



- $f_1(s)$ = #white legal moves
- $f_2(s)$ = #white - #black
- ...

	a	b	c	d	e	f	g	h
1	100	-20	10	5	5	10	-20	100
2	-20	-50	-2	-2	-2	-2	-50	-20
3	10	-2	-1	-1	-1	-1	-2	10
4	5	-2	-1	-1	-1	-1	-2	5
5	5	-2	-1	-1	-1	-1	-2	5
6	10	-2	-1	-1	-1	-1	-2	10
7	-20	-50	-2	-2	-2	-2	-50	-20
8	100	-20	10	5	5	10	-20	100

https://play-othello.appspot.com/files/Othello.pdf

The Depth-Limited Alpha-Beta Search algorithm

function ALPHA-BETA-SEARCH($state$) **returns** an action
  $v \leftarrow$ MAX-VALUE($state, -\infty, +\infty$)
  return the $action$ in ACTIONS($state$) with value v

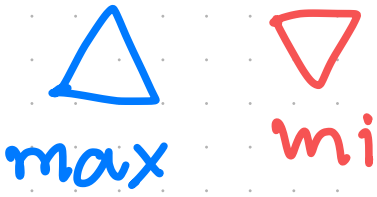
---

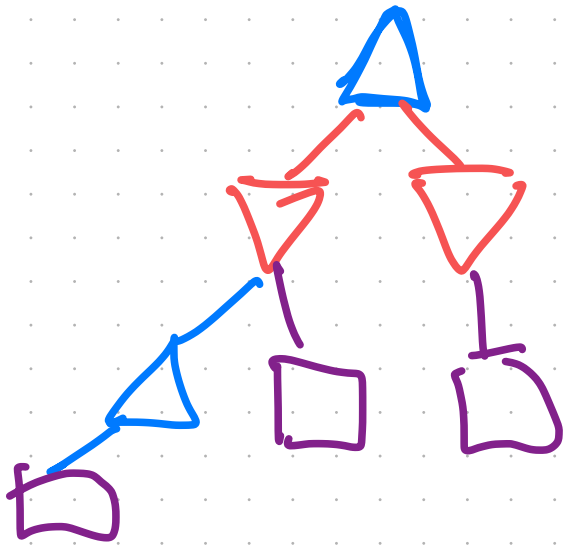
function MAX-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

---

function MIN-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v

Summary.

1. 



minimax value

$V(S)$ { max $V(S')$ / min $V(S')$

$V(S)$ = Utility(S) if S is terminal

2. Optimal action
   = a action that leads to an optimal value child node
3. Adversarial Search Problem → Formulation
4. minimax algorithm
5. Alpha - Beta pruning
6. Depth-limit + evaluation function.