CS249 (Artificial Intelligence - I)

→ Book: RUSSELL & NORVIG (read from 1st chap → get motivated from this chap.

→ 54 students : 11 groups
1 - 5
⋮
36 - 40
⋮
51 - 54

Assignment - 1

11 weeks — 11 groups

will prepare Basic Slides of that week,

Basic Slides → Class Notes, Topics } Blue / Black white Background

(Include citations & contribution of every person)

Deadline : every week Saturday

Marks are given by students → avg. to 10

→ Quiz / Assignment — 20 - 30 %

MSE — 30 %

ESE — 50 % - 40 %

Mini Project
Real life App. of AI
(group of n=5)

Deadline — 1st week of April ← LAB MSE / ESE

AI :

→ Thinking / Perception / Action

Cognitive Perceptive

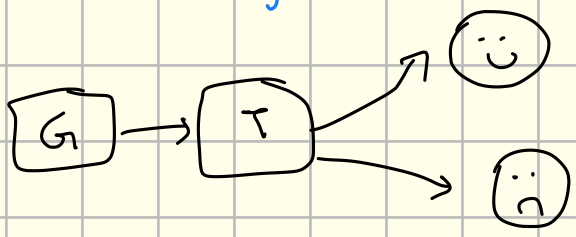
→ Model Targeted at thinking Perception and action to explain Past, Predict future, understand the subject, control the "world."

→  → which tree does this leaf belong to?

to answer this question we try to recall our Past experiences & try to answer the Q.

→ we will try to Perform some Representation that Supports our Q. of leaf matching to tree

→ Generate & Test:



→ generator shld'nt generate redundant

→ shld be informative

So that we have

→ Representation right

* SIMPLE ≠ TRIVIAL

→ SIMPLE: Can be Powerful

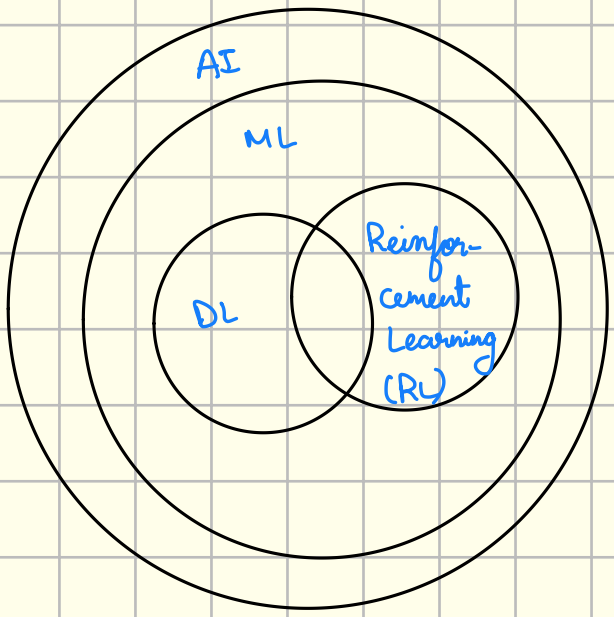
→ TRIVIAL: makes it sound its not only simple but has a

little worth

→ SIMPLE IDEAS may be Powerful

→ Don't always go for the Complicated one

→ Big Picture :



→ Solve the AI problems using REASONING

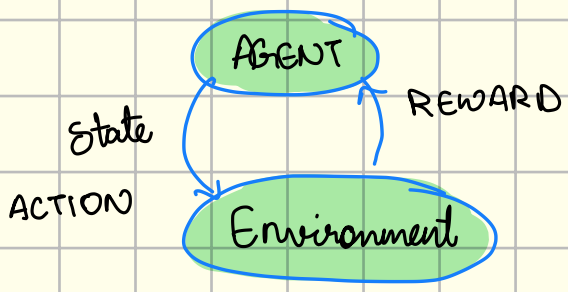
→ Algorithms Enabled by
Constraints Exposed AT
Representation that supports
Model targeted at Thinking / Perception / Action

→ Engineer → " Building tool kits "

IITAN
Computer Scientists → Computational Account of Intelligence
(understanding the Background that how they work)

→ actions are based on your Cognitive Perception & Observation

Eg: Autonomous Cars



"KNOWLEDGE IS POWER"

→ LAB : Python → W3SCHOOLS

↓

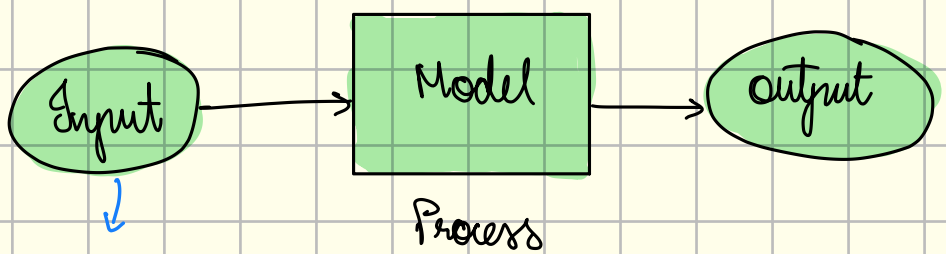
Frameworks (LIBRARIES)

4-5 memb. group

own Laptop

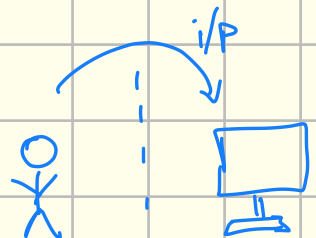
→ Prerequisite : ALGORITHMS

→ AI : Computational Modelling



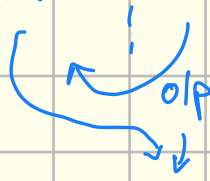
if the input is anything other than no.s ⇒ 1st convert them to no.s

Turing Test (1950)



→ The Computer is interrogated by a human in a teletype

→ IT PASSES, if the human can't tell if there is a comp. or a human at the other end.



if the human identifies that this o/p is given by human ⇒ Comp. Passes Turing test

→ The ability to solve Problem (computational Problem)

→ Search: Efficient TRIAL & ERROR

SPACE / TIME Complexity trade off

→ Use of Domain knowledge (Heuristic)

"a student is poor in Algo"

↓

it is decided on a BENCH MARK

Block file : insertion > deletion / updation

↓

which type of DS is used?

→ linked list

insertion O(1)

→ Popular Techniques : Linear Programming

Integer Programming

Dynamic Programming

Heuristic search

Evolutionary Algorithm

AGENT :

→ It may be a Computer Program but it should be capable of doing "more"

→ AI models created should be a Responsible AI model

KNOWLEDGE AND DEDUCTION:

→ How to store & retrive knowledge?

→ How to Interprete facts & rules and able to reduce?

→ The GAP between knowledge and realisation.

→ Logics of knowledge

(knowledge based systems : KBS

Expert systems : ES

Automated Theory Problems FORMAL verification)

The Ability to learn:

→ Can we learn to solve a Problem better?

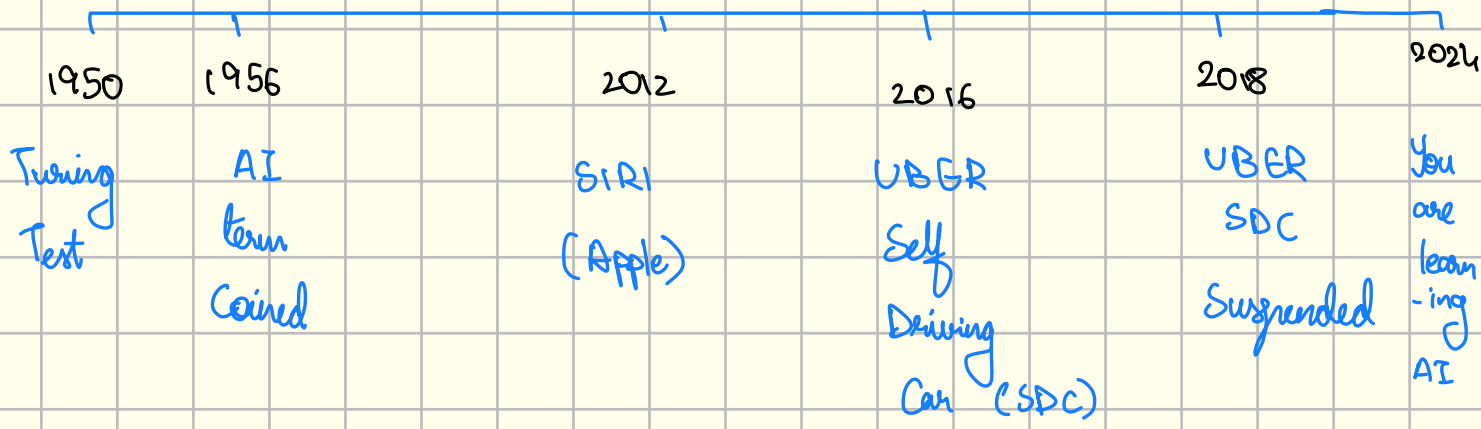
→ learning the answer.

→ learning the rule of GAME

→ learning to PLAN

→ Belief Network

→ Perseptron of Network N/w



What is AI?

→ Automated Problem solving

→ ML

→ Logic and Deduction

Human - Comp. Interaction

→ Comp. vision

→ NLP

→ Robotics

FUNDAMENTALS:

① The notion of expressing computation as an ALGORITHM

② GODEL'S INCOMPLETNESS Theorem (1931)

③ CHURCH - TURING THESIS (1936)

→ Turing machine is capable of computing any

computable Functions

→ This is accepted definition of computing

(4) 2 Notions of Intractability

→ NP Completeness

→ Reduction

(5) Problem solving by search

→ State space search

→ Prob. reduction ,

→ Game Playing

(6) logic and deduction

(1st order logic temporal logic)

→ Planning

→ Reasoning under uncertainity

→ Learning

→ additional topics

AGENT :

* An agent is just something that ACTS.

* All comp. programs do something but AGENT does something more.

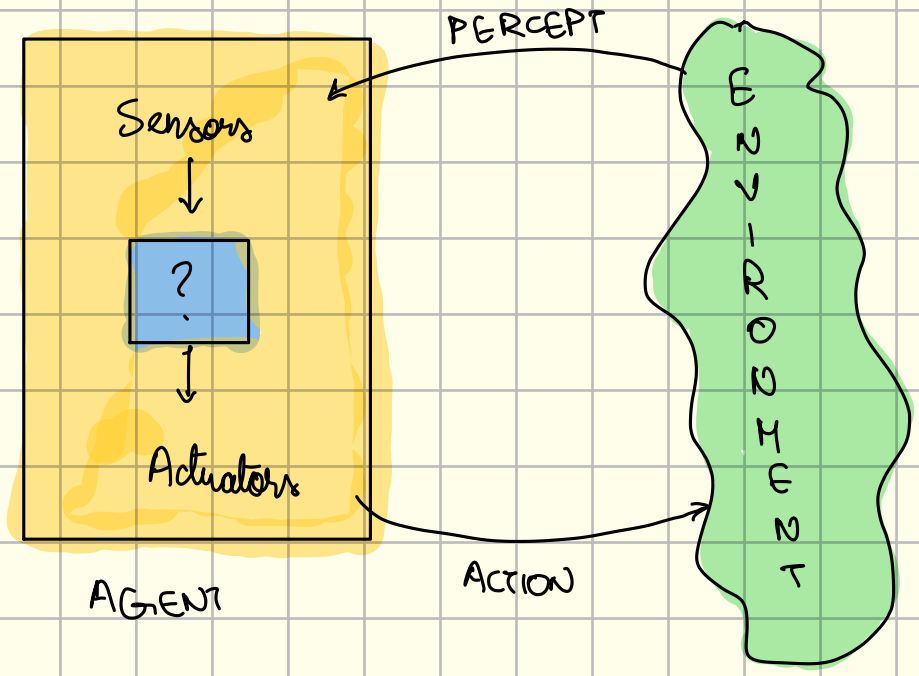
→ Operate autonomously

→ Perceive their Environment

→ Persist over a prolonged time-period

→ Adapt to change, create & Persue goals.

* A Rational agent is one that acts so as to achieve the best outcome or where there is Uncertainity the best expected outcome.



Agent type	Performance Measure	Environment	Actuator	Sensor

TAXI DRIVER	SAFE, LUXORY, COMFORT	Road Pedestrian	A, B, C, Steering	Speedometer, GPS, Comp. vision, rotative eye, camera
MEDICAL diagnostic System	health of the Patient	Hospital, Staff of Patient	Tests, Treatments	Questionaries
Interactive English Tutor	Students Score	Set of Students	Exercise	Smart Phone/ Key Board

→ Fully Observable | Partially Observable | Unobservable (No sensor)

→ Single Agent | Multi Agent (chess)

Cross word Puzzle

→ Deterministic v/s Stochastic Environment

↓ ↓

next action is determined by the current state

Next action can't be determined

Uncertain / Non-deterministic

→ Episodic v/s Sequential

agents experience is divided into atomic episodes.

→ many stats together make an environment

Eg: In a game 1st play : go through states → know the environment ↑ 1st Episode

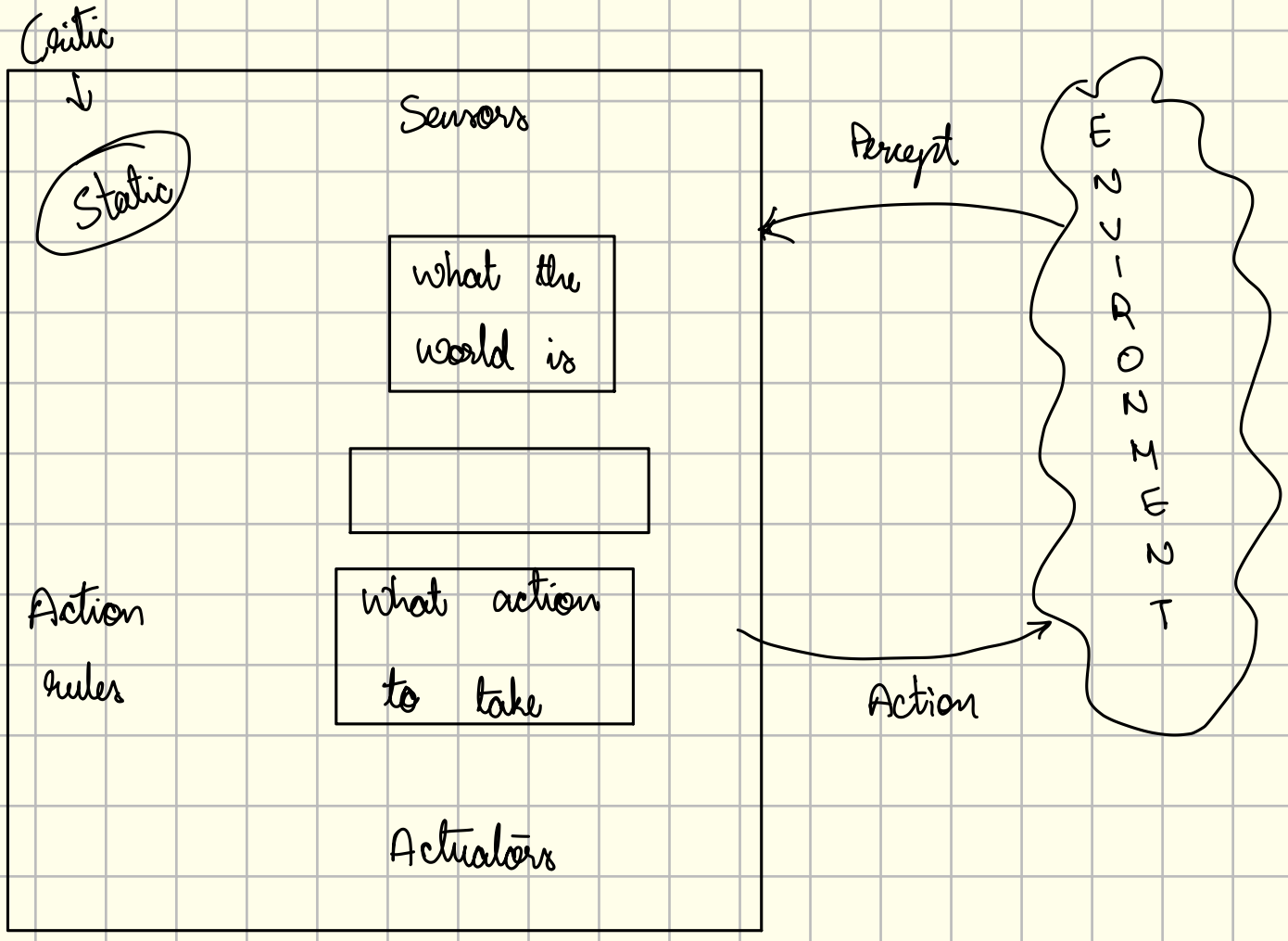
2nd Play : " " → " " ↓ 2nd Episode

→ Discreate | Continuous

→ known v/s Unknown

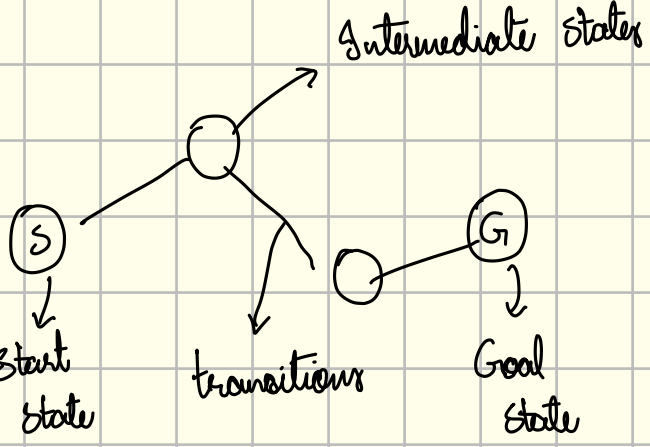
Discrete	Total Environment	Observation	Agent	Deterministic	Episode	Static
Discrete	Cross word	full	single	Deterministic	Sequential	static
"	Chess with	"	Multi	"	"	Semi
Continuous	Test driving	Partially	Multi	Stochastic	"	Dynamic

Image Analysing | Fully | Single | Deterministic | Episodic | Semi



SEARCH FRAMEWORKS

- → Uniformed / BLIND
- → INFORMED / HEURISTIC
- → Problem reduction
- → Game Tree Search
- → Others



BASIC SEARCH PROBLEMS:

→ Given $[S, s, O, G]$ where

S is the (implicitly specified) set of states (diff. betw. Implicit & Explicit)

$s \in S$ is the start state

O is the state transition operators

$G \subset S$ is the set of goals

→ To find a sequence of state transition leading from s to G

→ 15 Puzzle game.

Eg: 8 Puzzle

1	2	3
4	5	6
7	8	

G

6	7	2
5	1	4
8		3

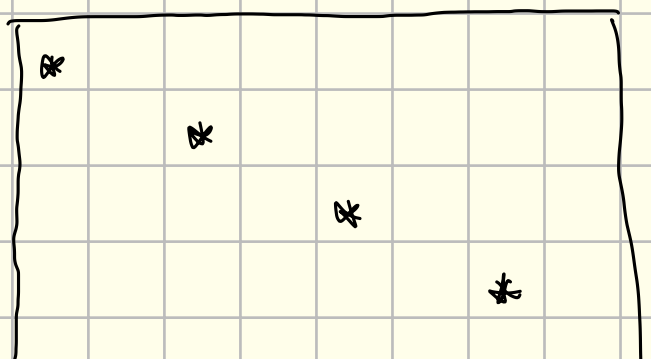
s

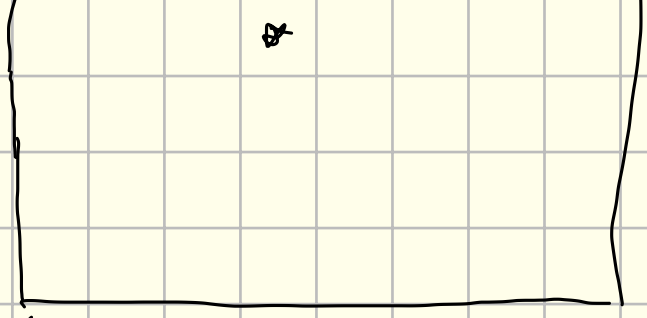
```
4 1 2
5   3
7 8 6
```

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

O : movement of blank tile in LRUD

→ 8 Queen Problem:





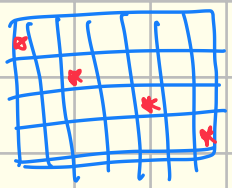
8 Queen Problem :

Placing 8 queens so that none attack each other

(i) A state is arrangement of 0-8 queens on board
operators add a queen to any sequence.

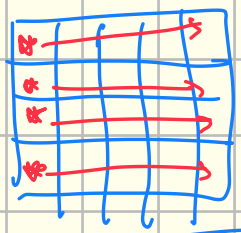
$(8\times 56) \rightarrow (8\times 56)^2 \rightarrow \dots$

(ii) A state is any arrangement of 0-8 queens with none-attacked operators place a queen in the left most empty Column.



$8 \rightarrow 8^2 \rightarrow 8^3 \dots$

(iii) A state is any arrangement of 8 queen operators
move an attacked queen to another square in the same row.



Each queen has 7 blank space to move

$7\times 8 \rightarrow (7\times 8)^2 \rightarrow (7\times 8)^3 \dots$

8 queens $\rightarrow$ $(7\times 8) \times (7\times 8) \times \dots$

available moves | 1st move | 2nd move

Missionaries & Cannibals Problem

M C

→ 3 M & 3 C are one side of a river, along with

a BOAT that can hold 1 or 2 people.

→ Find a way to get out to the other side without leaving a group of M outnumbered by C.

$S = \{ \langle x, y, z \rangle, \; x \in \{0,1,2,3\}, \; y \in \{0,1,2,3\}, \; z \in \{0,1\} \}$

$x \rightarrow$ no. of M on left

$y \rightarrow$ " " C on right

$z \rightarrow 0$: boat on left

1: boat on right

$S = \langle 3,3,0 \rangle$

$G = \{ \langle 0,0,1 \rangle, \langle 0,0,0 \rangle \}$

$\langle 0,0,0 \rangle$ → Infeasible State

$O : \{ \langle O_1, O_2 \rangle$

$O_1 \rightarrow$ #M on boat

$O_2 \rightarrow$ #C on boat

$\langle 0,0 \rangle$ ✗

$\langle 0,1 \rangle$

$\langle 1,0 \rangle$

$\langle 1,1 \rangle$

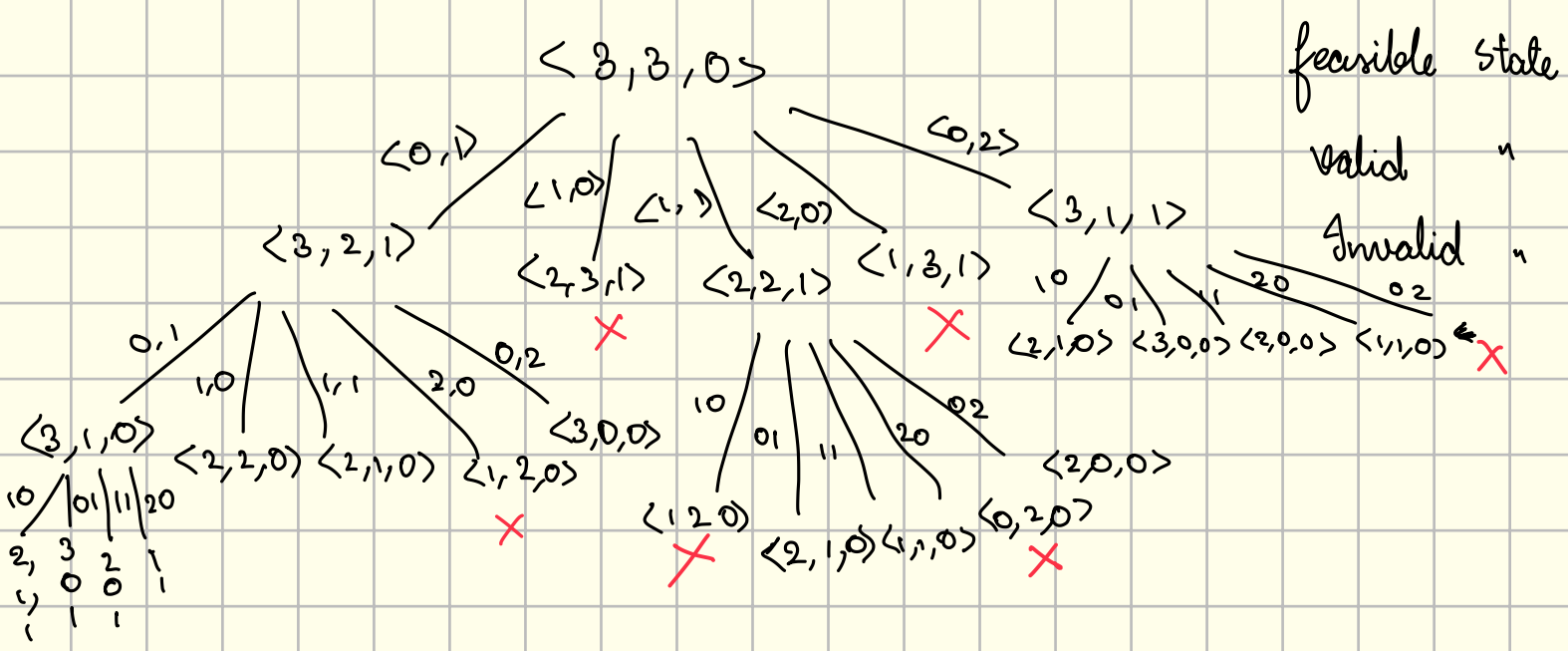
$\langle 2,0 \rangle$

$\langle 0,2 \rangle$

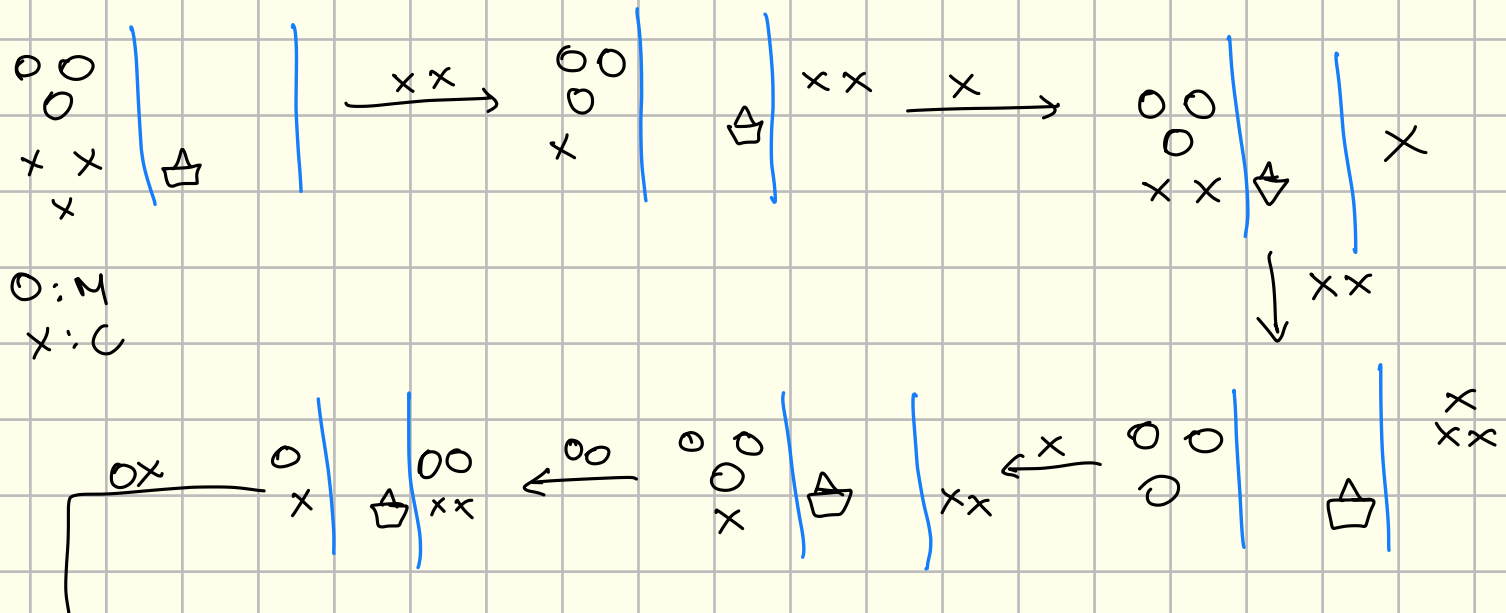
$$\begin{cases} O_1 \in \{0,1,2\} \\ O_2 \in \{0,1,2\} \\ 1 \leq O_1 + O_2 \leq 2 \end{cases}$$

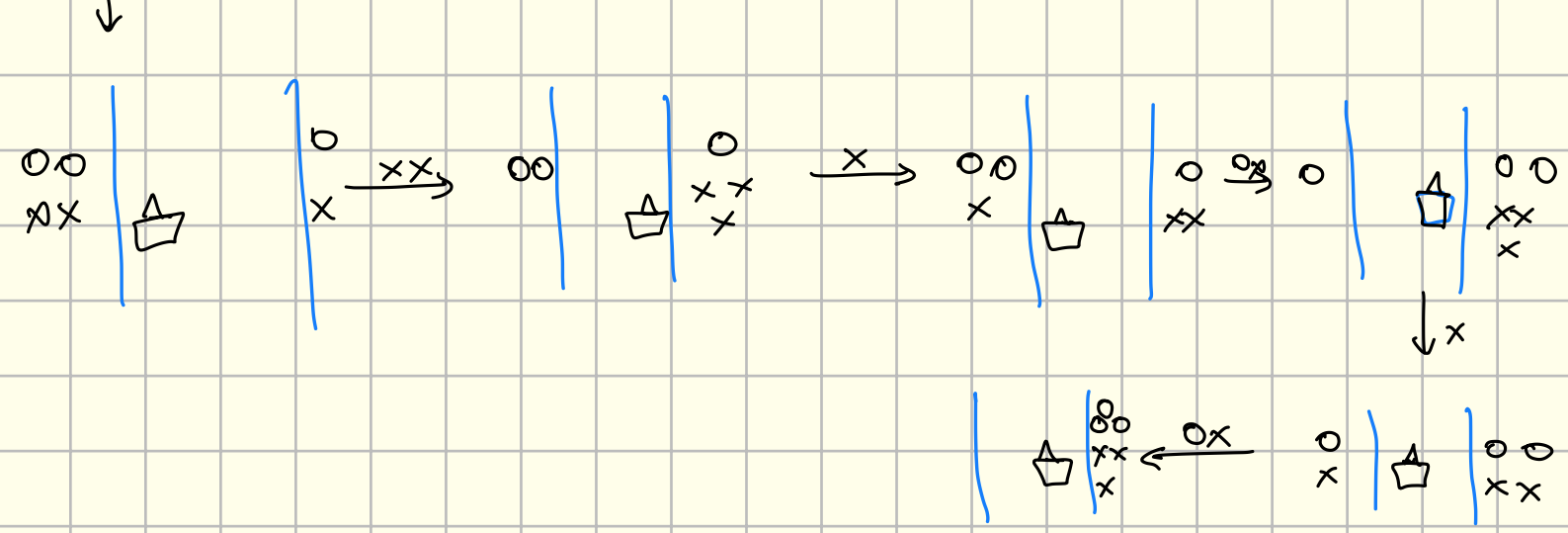
$O : \{ \langle O_1, O_2 \rangle : O_1 \in \{0,1,2\}, \; O_2 \in \{0,1,2\}, \; 1 \leq O_1 + O_2 \leq 2 \}$

→ draw the entire chart of how many min state transitions are required to reach goal state.

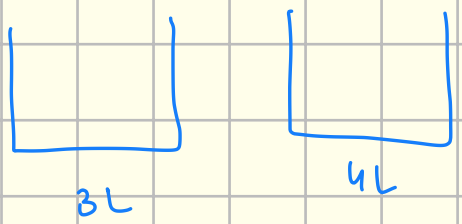


* Feasible State → Valid
  → Invalid (based on some constraint)
* Infeasible State (not reachable from s)
* Undefined State (Not matching as per the def.^{n} of the state.)





JUG / JAR PROBLEM:



Problem: measure 0, 1, 2 ..... 7L using 3L & 4L beakers

0 3 4 7

1L: 3 4 → 5L: 0 1
0 4 1 0
3 1 1 4

2L: 3 0 → 6L: 2 4
0 3
3 3
2 4

→ Similarly we can do it for 4L & 5L vessels.

Formulate this into an AI Problem:

→ S, S, O, G

S: $\{ \langle x,y \rangle$: x : amount of water in 1st vessel, $0 \le x \le 3$ $\}$
y : amount of water in 2nd vessel, $0 \le y \le 4$
$x, y \in \mathbb{Z}$

s : $\{\langle 0,0 \rangle\}$

O : $(0,y)$: making the 1st jar empty

$(x,0)$: making the 2nd jar empty

$(3,y)$: fill 1st jar

$(x,4)$: fill 2nd jar

$(x+k, y-k)$: transfer k lt. of water from 2nd jar to 1st jar

$k = \min(3-x, y)$

$(x-k, y+k)$: transfer k ltr. of water from 1st to 2nd jar

$k = \min(x, 4-y)$

G : $\{ (x,y) : 0 \le x \le 3,\ 0 \le y \le 4,\ x, y \in \mathbb{Z},\ x+y = T,\ 0 \le T \le 7,\ T \in \mathbb{Z} \}$

TRAVELLING SALESMAN PROBLEM :

→ Problem : there is a salesman he has to travell to diff. cities to sell his Products.

1) visit every city except source one of only once

2) start from source & come back to source.

3) Minimize Cost (Cost can be time / Money)

→ AI Problem Statement : s, S, O, G, C
↳ Cost.

s: source

G: find a path with Min. Path that visits every city exactly once & comes back to source

S: Multiple Paths available

O: the different transition avalde to go from one city to another

PDKHCBP

P K D B H C P

↳ 5! ⇒ diff. ways are possible.

C: Cost : can be money / time.

SEARCH ALGORITHMS : (BASIC)

① Initialize : set OPEN = {s}

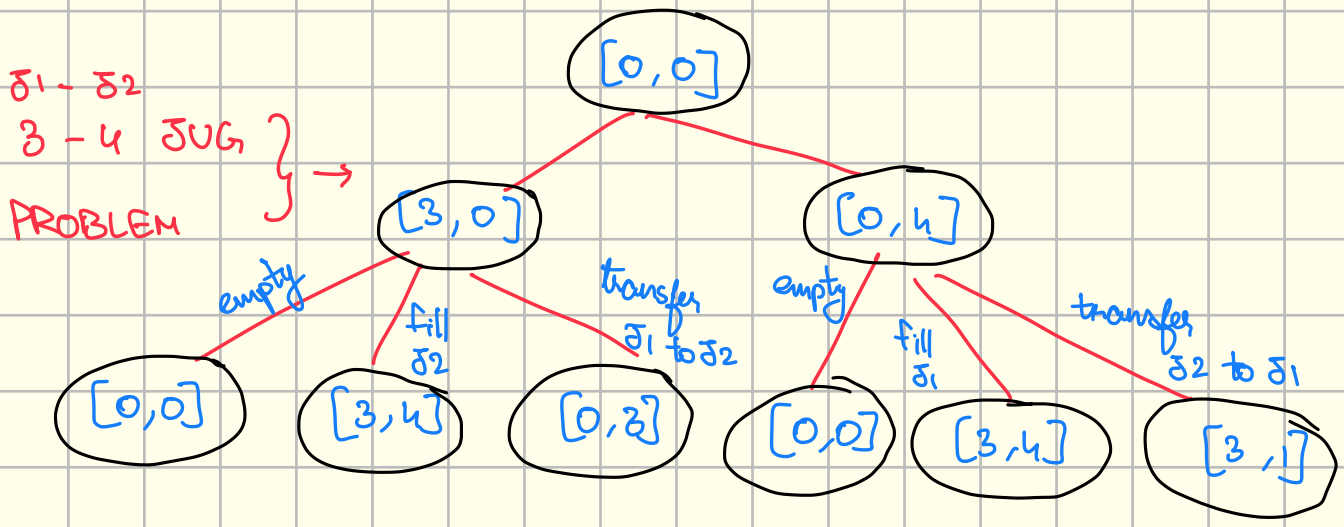
② Fail : if OPEN = {}, Terminate with failure

③ Select : Select a state n from OPEN

④ Terminate : if n ∈ G, terminate with success

⑤ Generate the successor / neighbours of n with 0 and insert them in OPEN.

⑥ LOOP : Go to step ②



BFS → Queue
DFS → Stack

SEARCH ALGO FOR STATE SPACE SEARCH :

→ we will maintain 2 lists, OPEN, CLOSED, for searching

$P_R = [S, O, S, G]$

Initialization : Insert s to OPEN, CLOSED = []

→ Termination with success : Remove node n from OPEN

if n ∈ G then terminate with success

add n in CLOSED

Selection : Apply all $o \in O$ on 'n' to generate all neighbours N of n $[N \subseteq S]$

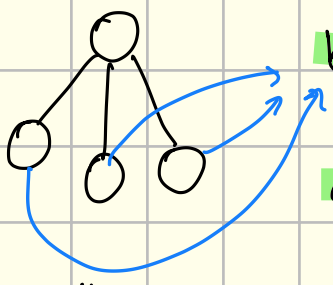
$\forall m \in N$ check if $m \notin$ (OPEN $\cup$ CLOSED)

Add m in OPEN

Loop

Termination with Failure : If OPEN is empty, terminate with Failure otherwise go for looping / next Iteration

→ Algorithm : Time Complexity | Space Complexity | Completeness | Optimality



b: branching factors

d: depth

b = 3

d = 1

Acyclic graphs ↓ trees

→ The Algo is generic and "blind" because it is not using any domain knowledge.

→ The Characteristics of the algo depends on what data structure we are using for OPEN

→ A graph is created however the graph G is an implicit graph. because all the vertices and edges are not known beforehand. (1st we decide the operation & based on that we get the vertices else we don't know ver. beforehand.)

→ we begin with start state, keep on building G_T till we reach goal state

BFS | DFS : time Complexity $O(b^d)$

b : branching factor
d : depth

Space Complexity :

DFS : $O(b*d)$ d : The max. depth of the graph from start state

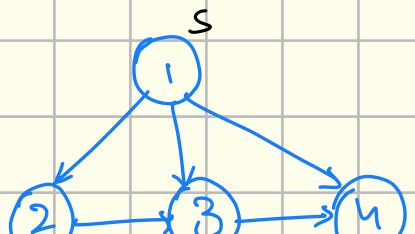
BFS : $O(b^d)$ d : min. level at which goal state can be formed

Completeness & Optimality :

DFS : Incomplete & Not Optimal

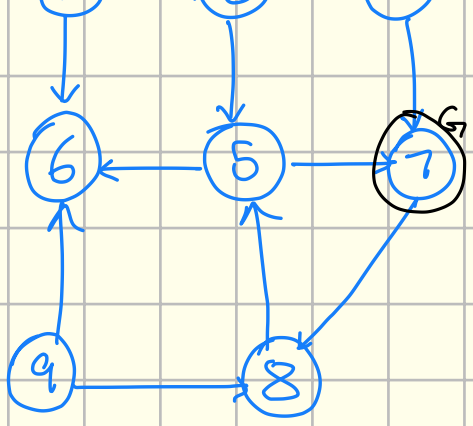
BFS : Optimal & Complete

Work :
AI RELATED RIDDLES



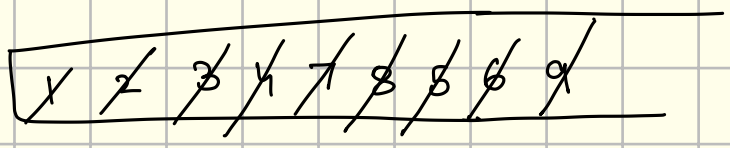
PERFORM DFS :

stack :



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

OPEN [stack]:

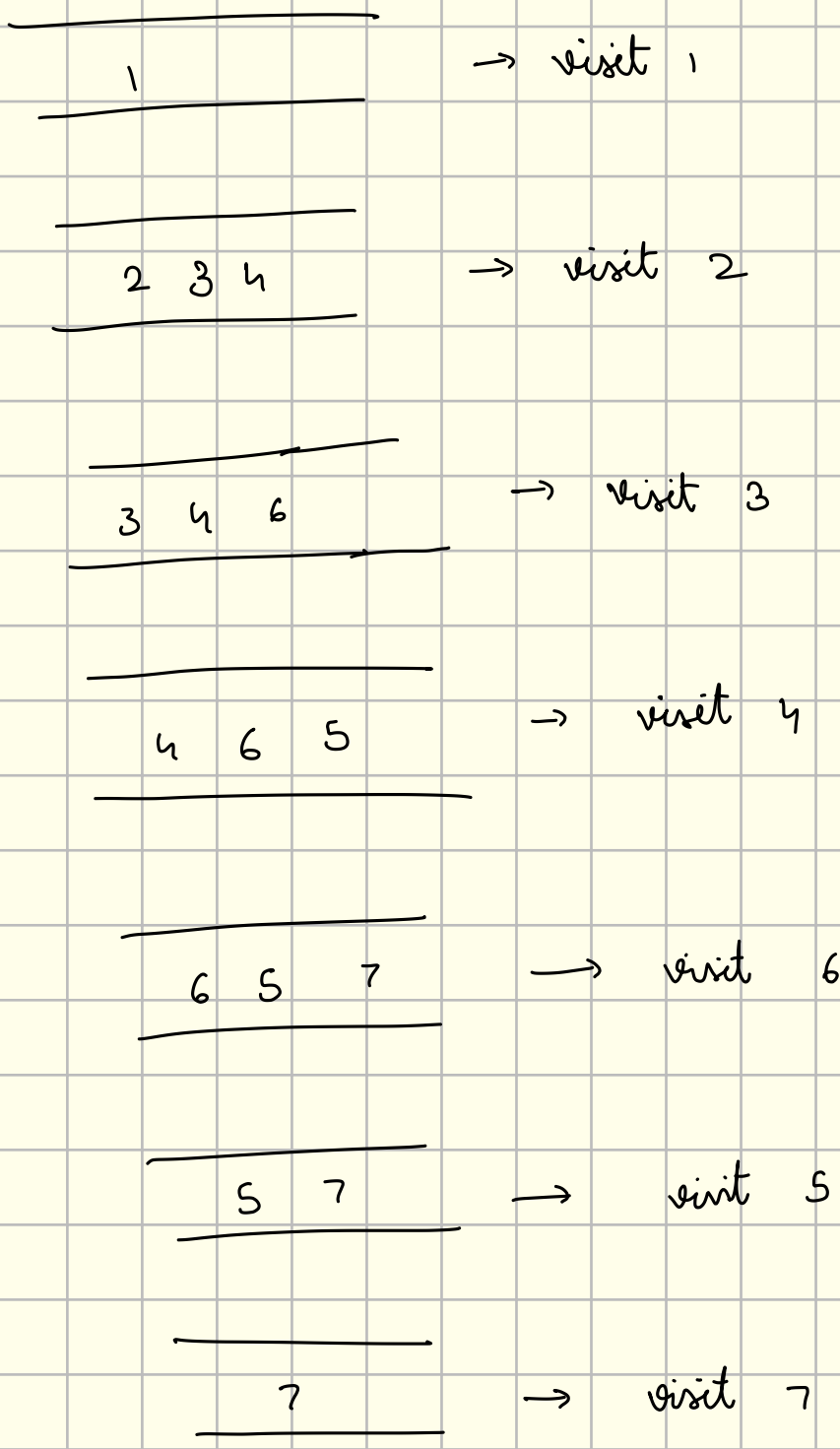


⇒ 1 4 7 8 5 6 3 2 9

OPEN [stack]:

1	→ visit 1
4 3 2	→ visit 2
4 3 6	→ visit 6
4, 3	→ visit 3
4, 5	→ visit 5
4, 7	→ visit 7
4	→ visit 4

PERFORM QUEUE:

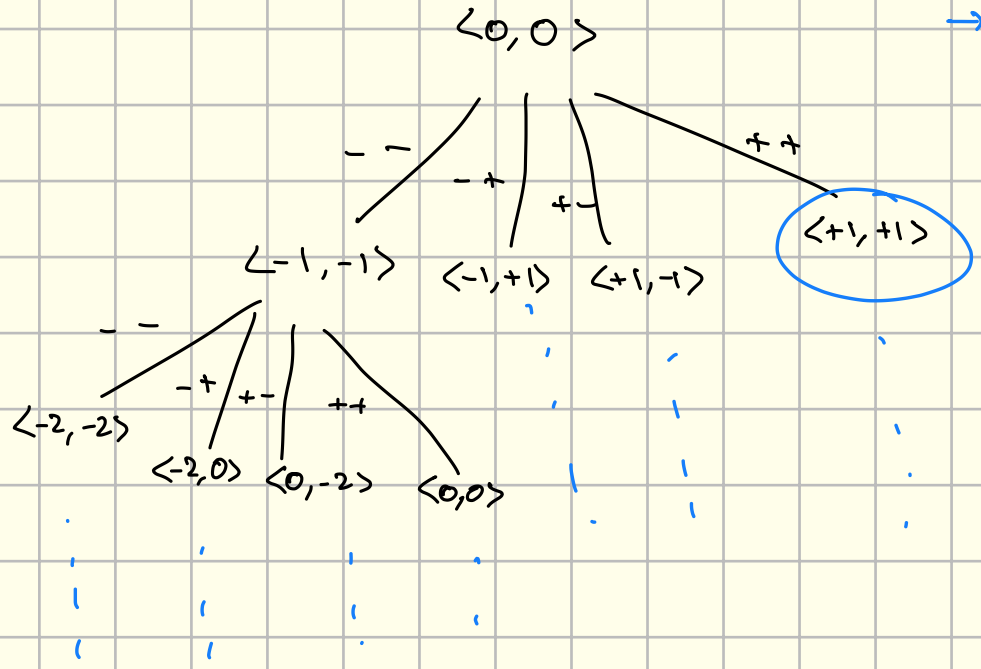


→ STATE SPACE may be infinite

b : <0,0>

G : <1,1>

O : <+,+>, <-,->, <+,->, <-,+>



→ If we apply DFS here we can't reach the goal state

∴ DFS is incomplete

COMPLETENESS: If start from S if there exist G then after completing the search we should be able to reach the goal state

→ diff. btw. A Goal & the Goal

BFS:

→ Time: b^d → $1 + b + b^2 + \dots + b^d \approx O(b^d)$ → d: depth of graph

→ Space: b^d

d: depth of goal state

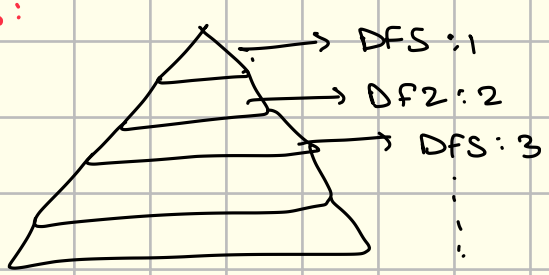
DFS:

→ Time: b^m

→ space: $b.m$ → m depth x b brothers → m: depth of the state space tree

	TIME	SPACE	COMPLETENESS	OPTIMALITY
BFS	$O(b^d)$	$O(b^d)$	✓	✓
DFS	$O(b^m)$	$O(b \cdot m)$	✗	✗
Depth limited DFS (DDFS)	$O(b^l)$ l: depth limit	$O(b \cdot l)$	✗	✗
Iterative Deepening DFS (IDFS)	$O(b^d)$	$O(b \cdot d)$	✓	✓
Bidirectional BFS (if Possible) (BBFS)	$O(b^{d/2})$	$O(b^{d/2})$	✓	✓

IDFS:



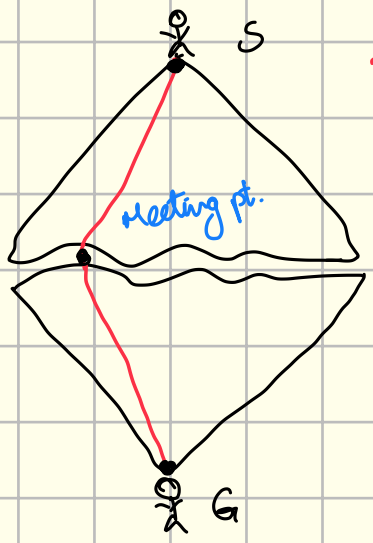
we keep changing the depth and apply the DFS.

∴ Space : $O(b \cdot m)$

Time : $O(b^m)$

→ It is Complete & Optimal

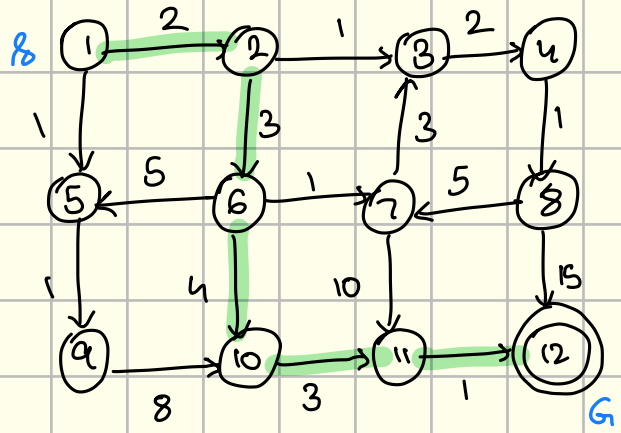
BIDIRECTIONAL :

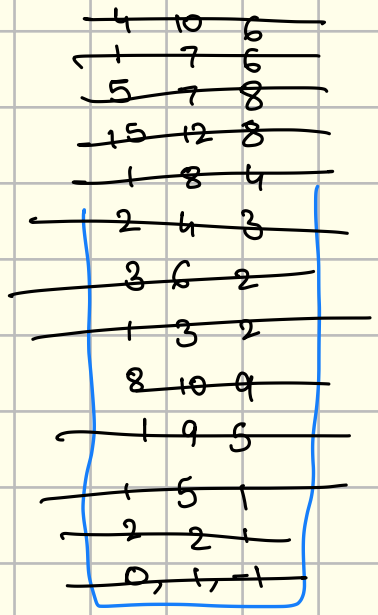


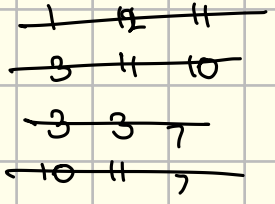
$O(b^{d/2}) + O(b^{d/2}) = O(b^{d/2})$

→ It may not always be possible to go from goal to s

→ An Algo is called COMPLETE, if it always returns a solution / Goal state, if there exists a goal state which is reachable from the start state.



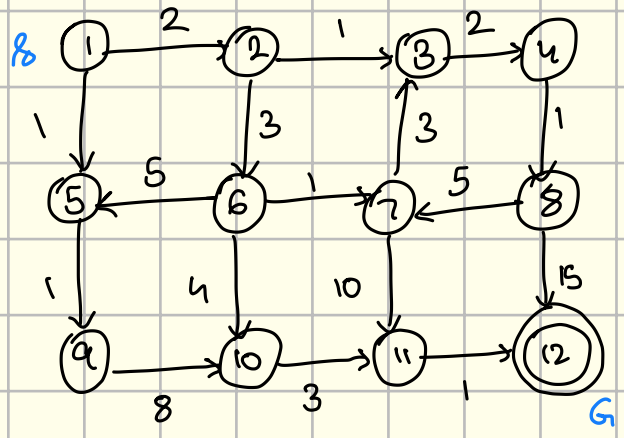




wt. node par

	1	2	3	4	5	6	7	8	9	10	11	12
vis →	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	1	1	1	1	1	1	1	1	1	1	1	1
dist →	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
	0	2	3	5	1	5	6	6	2	9	12	13

par →	-1	1	2	3	1	2	6	4	5	6	10	11



OPEN

$1^{(0)}$			
$2^{(2)}$	$5^{(1)}$		
$2^{(2)}$	$9^{(1)}$		
$9^{(2)}$	$3^{(3)}$	$6^{(5)}$	
$3^{(3)}$	$6^{(5)}$	$10^{(10)}$	
$6^{(5)}$	$10^{(10)}$	$4^{(5)}$	
~~$10^{(10)}$~~	$4^{(5)}$	$7^{(6)}$	$10^{(9)}$
$7^{(6)}$	$10^{(9)}$	$8^{(6)}$	
$10^{(9)}$	$8^{(6)}$	$11^{(16)}$	
$10^{(9)}$	$11^{(16)}$	$12^{(21)}$	
~~$11^{(16)}$~~	$12^{(21)}$	$11^{(12)}$	
$12^{(21)}$	$12^{(13)}$		

CLOSED :

$1^{(0)}$ $5^{(1)}$ $9^{(2)}$ $3^{(3)}$ $6^{(5)}$ $4^{(5)}$ $7^{(6)}$ $8^{(6)}$ $10^{(9)}$ $11^{(12)}$ $12^{(13)}$

HomeWork : UNIFORM COST SEARCH

FIRST SEARCH ALGORITHM :

1) Initialize : Set OPEN = {S}

2) FAIL : If OPEN = { } Terminate with failure.

3) SELECT : select a start, n, from OPEN

4) TERMINATE : If $n \in G$, terminate with success

5) EXPAND : Generate the successors of n using O and insert in OPEN

6) LOOP : Goto step 2

SAVING THE EXPLICIT SPACE:

1) CLOSED = { }

2) FAIL : X

3) SELECT : select a state n from OPEN and save n in closed

4) X

5) Expand : for each successor in m, insert m in OPEN only if

$m \notin [OPEN \cup CLOSED]$

SEARCH AND OPTIMIZATION:

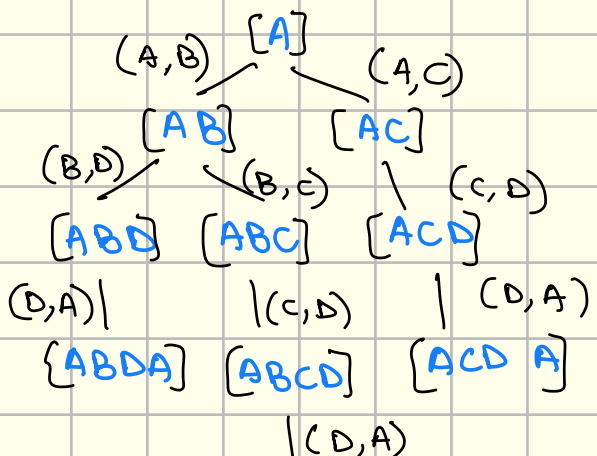
→ given : [s, S, O, G, C]

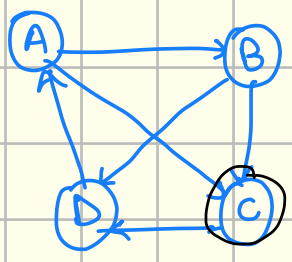
→ To Find :

(i) → A min^m cost sequence of transition to a goal state

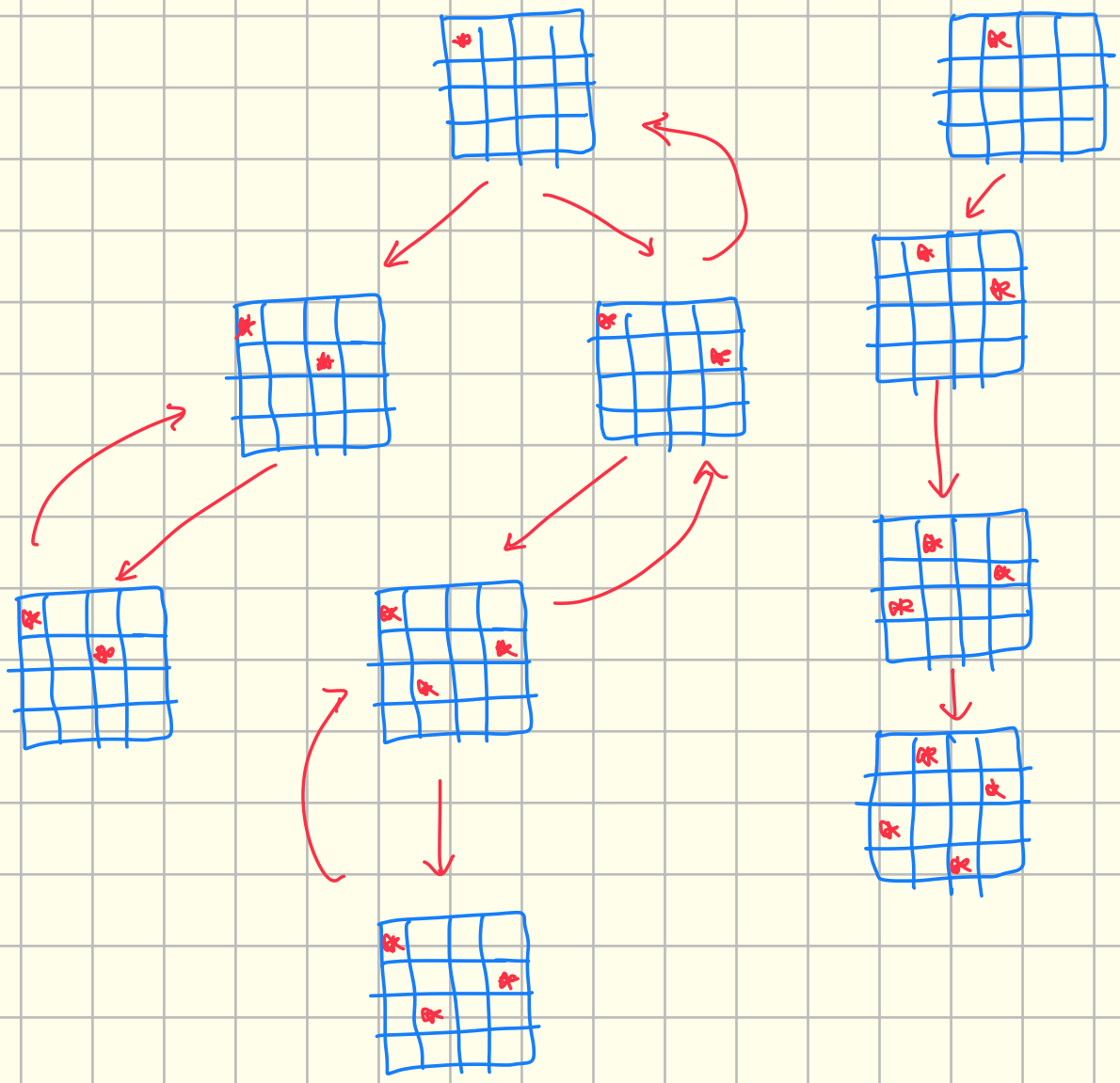
(4 Queen Problem) (ii) → A sequence of transitions to the min^m cost goal

(iii) → A min^m cost sequence of transition to a min^m cost





[A B C D A]



Home Work: Think about (iii)

→ UCS (Uniform Cost Search) assumes all operations have a cost

① Initialize : Set OPEN = {s}
CLOSED = { }, Set C(s) = 0

② FAIL : If OPEN = { }, Terminate & FAIL

③ Select : Select the min " cost state 'n'

from OPEN and solve 'n' in closed

④ Terminate : If $n \in G$, terminate with success

⑤ Expand : Generate the successors of n using O for each successor, 'm'.

If $m \notin [OPEN \cup CLOSED]$

set $c(m) = c(n) + c(n,m)$

If $m \in [OPEN \cup CLOSED]$

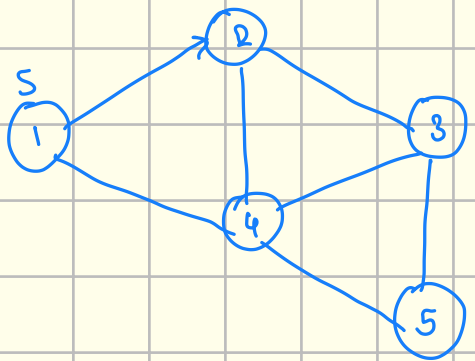
set $c(m) = \min\{c(m), c(n) + c(n,m)\}$

If c(m) has decreased and $m \in$ CLOSED, move it to OPEN

⑥ LOOP

Home Work: find diff. btw. ucs & Dijkstra → why the diff.?

UCS (diff. with Dijkstras)



G: {2, 3, 4, 5}

UCS :

u. cost = 0 (cost to reach v from s)

v. cost = 5 (cost to reach v from s)

↳ Cost (u, v)

Dijkstras → no -ve edges

ucs → allow -ve edges

→ n.Cost - The cost to reach $n \in S$ from s

Initialization : Open = [s] Closed : [] s.cost = 0

Iteration : Remove the lowest cost state n from OPEN.

CLOSED.append (n)

(use Heap / Priority Queue) → (O(1) to find min elem.)

Termination with Success : If $n \in G$ Then terminate with success.

(This termination cost is used only when all operator cost are +ve)

Selection : Apply all $o \in O$ on 'n' to obtain weight of n (say m)

$m \notin$ [OPEN ∪ CLOSED]

$m.cost = \underbrace{n.cost + cost(n,m)}_{B}$

If $m \in$ OPEN and $m.cost > B$

Revise the cost of m

$m.cost = B$

If $m \in$ CLOSED and $m.cost > B$

Revise cost of m

$m.cost = B$

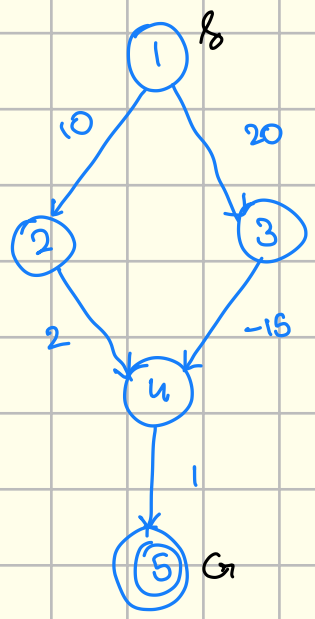
move m from CLOSED to OPEN

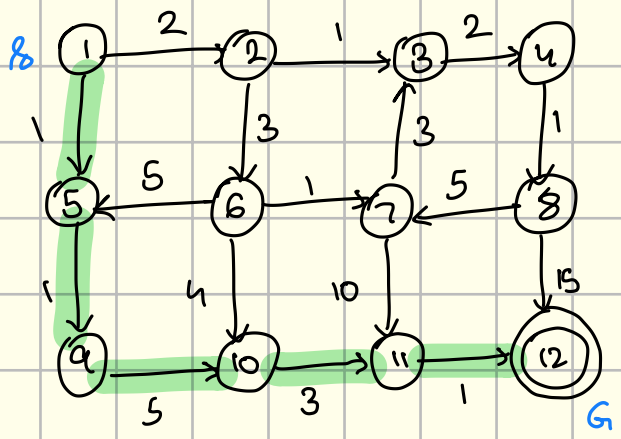
This cond.ⁿ will be encountered only when there is -ve weight

Terminate with Failure : If OPEN = [] , terminate with failure

Exhaustive Search : Terminate cond^n with graph having -ve edge

Example:

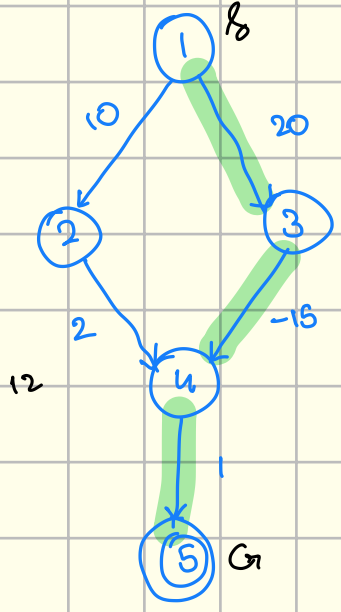




CLOSED:

$1^{(0)}$ $5^{(1)}$ $2^{(2)}$ $9^{(2)}$ $3^{(3)}$ $6^{(5)}$ $4^{(5)}$ $7^{(6)}$
$8^{(6)}$ $10^{(7)}$ $11^{(10)}$ $12^{(11)}$

OPEN:
$1^{(0)}$
$5^{(1)}$ $2^{(2)}$
$2^{(2)}$ $9^{(2)}$
$9^{(2)}$ $3^{(3)}$ $6^{(5)}$
$3^{(3)}$ $6^{(5)}$ $10^{(7)}$
$6^{(5)}$ $10^{(7)}$ $4^{(5)}$
$10^{(7)}$ $4^{(5)}$ $7^{(6)}$ ~~$10^{(9)}$~~
$10^{(7)}$ $7^{(6)}$ $8^{(6)}$
$10^{(7)}$ $8^{(6)}$ $11^{(16)}$
$10^{(7)}$ $11^{(16)}$ ~~$7^{(14)}$~~ $12^{(21)}$
~~$4^{(16)}$~~ $12^{(21)}$ $11^{(10)}$
~~$12^{(21)}$~~ $12^{(11)}$



CLOSED:

$1^{(0)}$ $2^{(10)}$ $4^{(12)}$ $15^{(13)}$ $3^{(20)}$

OPEN:

$1^{(0)}$
$2^{(10)}$ $3^{(20)}$
$3^{(20)}$ $4^{(12)}$
$3^{(20)}$ $15^{(13)}$
$3^{(20)}$
$4^{(15)}$
$5^{(16)}$

$1^{(0)}$

$1^{(0)}$

$2^{(10)}$ $3^{(20)}$

$3^{(20)}$ $4^{(22)}$

$4^{(22)}$

Theorem: The UCS algo applied on a problem having +ve operator cost will return optimal sol^n (Prove this)

UCS

Terminate with Success : If $n \in G$ then terminate with success

(This termination condition is used ONLY when all operators costs are +ve)

if $m \in$ CLOSED

Terminate with Failure : If OPEN = [], then terminate with failure

ONLY Termination condition for graph

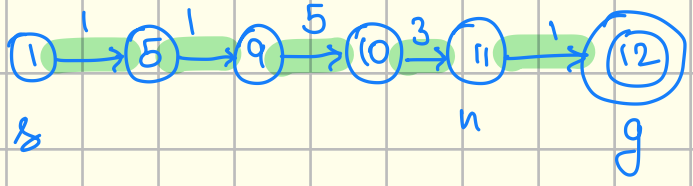
with -ve edge.

(This is being used when graph has -ve edge)

Theorem: (only for +ve operators cost)

(A) UCS applied on a prob. having +ve operator cost will return optimal solution

(B) **PROPOSITION:** At least one node in the path from s to the $n \in S$ (having optimal cost) should be in OPEN before n is being removed from OPEN



(C) There does not exist any other path from s to n having lesser cost than n.cost.

→ (i) At least one node belonging to the path having "OPTIMAL COST" from s to n should be in OPEN always before retiring n.

Proof: (i) optimal cost from

s to $n = C^*(n) = A+B$

In Path-1 : $A+B = C^*(n)$ — (1)

Assume : $\alpha < A$

$\alpha + B < C^*(n)$ ——— (2)

statement (2) contradicts statement (1)

(D) $g \in G$: closest optimal goal from s. (Corollary +ve Operator's Cost)

optimal cost of n from s

$C^*(n) \leq C^*(g)$; $\forall\, n \in S$ will be Expanded before g.

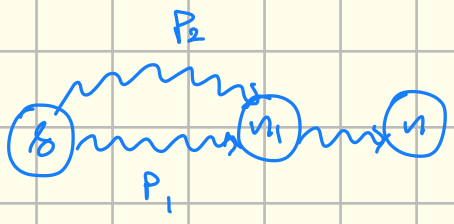
$C^*(10) = 7$

$C^*(9) = 2$

$C^*(12) = 11$

(2) If s to n → optimal

then s to n_1 → optimal



P is the optimal path from s to n through n_1 with cost $C^*(n)$

P_1 is the subpath of P from s to n_1

we need to proof P_1 is optimal subpath from s to n_1

Assume P_1 is not optimal path from s to n_1 with cost $C(n_1)$

let's $\exists$ P_2 which is optimal path from s to n_1

$\Rightarrow C^*(n_1) < C(n_1)$

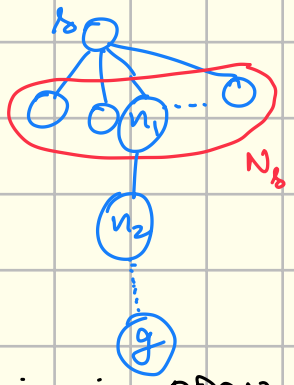
Path from s to n = optimal path from s to n_1 + path from n_1 to n

$= c^*(n_1) + P(n_1, n) < c^*(n)$

PROOF OF B:

Base Case : N_p : all the nodes reachable directly from s are in OPEN in the 1st step of Algo

→ Let, P be the optimal path from s to a node $n \in S$ consists of atleast one node in N_s



→ Lets assume, s to n consists of n_1 and n_2

n_1 is directly reachable from s, is in OPEN

→ Until n_1 is not being removed, (B) is true.

Where n_1 is expanded, n_2 will be visited

(i) n_2 is not in (OPEN ∪ CLOSED) ⇒ n_2 is brought to OPEN

(ii) $n_2 \in$ [OPEN ∪ CLOSED] ⇒ cost of n_2 will be revised through n_1 because of (C)

further (B) can be proved using Induction.

Proof (A): Can be proved by (B) of

Note: If the graph has unit weight / same weight use BFS instead of UCS

UCS :

→ If all operator costs costs are +ve, then the algo finds the min Cost sequence of transition to a goal.

→ No state comes back to OPEN from CLOSE (+ve)

→ If operator has unit / same cost then UCS is same as BFS

→ for -ve operator cost, it may come back to OPEN from CLOSED

BRANCH AND BOUND: (+ve Edge Cost) (Basic Version)

① Iterative: set OPEN = [s], CLOSED = [ ] set c(s) = 0

$c^* = \infty$

② Terminate: If OPEN = [ ], then return c^*

③ Select: select a state n from OPEN and save in CLOSED

④ New Bound Assignment: If $n \in G$ and $c(n) < c^*$, then set $c^* = c(n)$ and goto step 2

⑤ Expand: If $c(n) < c^*$ generate the successor of n

for each successor m:

if $m \notin$ [OPEN ∪ CLOSED]:

set $c(m) + c(n,m)$ as B and insert m in OPEN

if $m \in$ [OPEN ∪ CLOSED]:

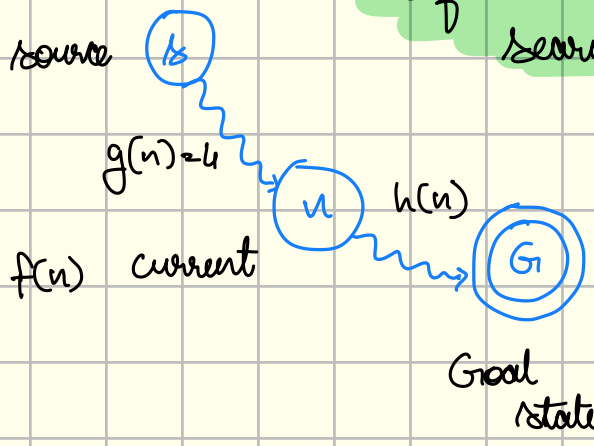
set $c(m) = \min(c(m), B)$

if $c(m)$ has decreased & $m \in$ CLOSED:

move to OPEN

⑥ Goto to step 2.

Informed search



$f(n) = g(n) + h(n)$

$g(n)$: Actual cost of n from s

$h(n)$: estimated cost of n to search G

→ Underestimate

→ Overestimate

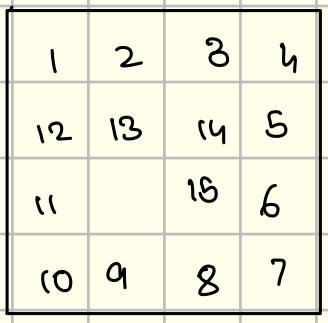
$h^*(n)$ = Optimal Cost to reach G from n

$h(n) \leq h^*(n)$ underestimate

$h(n) \geq h^*(n)$ Overestimate

$g(s) = 0$ (start)

$h(G) = 0$ (Goal)



G

'n'

$h(n) = 12$

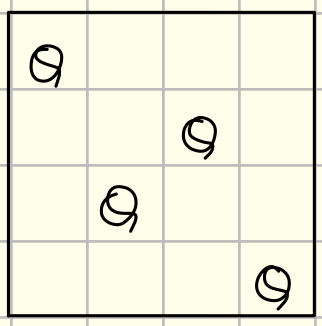
at least 12 tiles need to be moved

$h(n)$ (# tiles not in correct position)

for 8 Queen Problem what will be the heuristic ?

Read Russel Norvig

→ Heuristic



'n'

$h(n) = 2$

G: all Q's are not in attacking position

heuristic = cost of MST (underestimate)

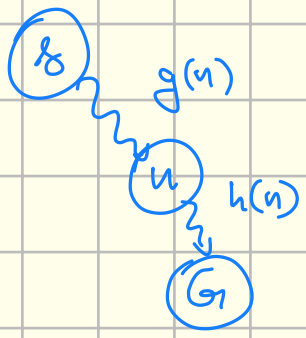
$2 \times \text{cost(MST)}$

$h > 2$ (overestimate)

pair of Q's in attacking position : $h(n)$

$g(n)$ = func^n of state and operator

$h(n)$ = func^n of only state (will not change)



$f(n) = g(n) + h(n)$

$f^*(G)$ } optimal value to reach G

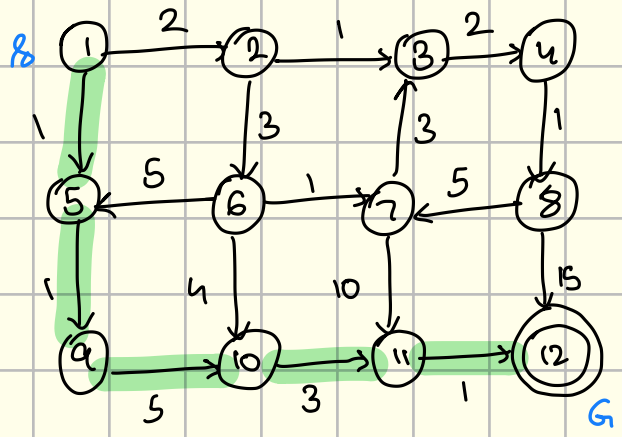
$g^*(G)$ }

INFORMEDNESS :

→ h_1 & h_2 are 2 UNDERESTIMATED HEURISTICS

→ h_1 is more informed than h_2, if $\forall\, n \in S$, $h_1(n) \geq h_2(n)$

→ $g(1) = 0$ ($\because g(\text{start}) = 0$)



→ $h(1) = 11$
$h(2) = 11$
$h(3) = 18$
$h(4) = 16$
$h(5) = 10$
$h(9) = 9$
$h(10) = 4$
$h(11) = 1$
$h(12) = 0$

$h(n)$: optimal path from n to G

→ If $\forall\, n$ $h(n) = h^*(n)$ then only the nodes in the optimal path will be expanded

↳ Retrieved from OPEN

$f(n) = g(n) + h(n)$

$f(1) = g(1) + h(1) = 0 + 11 = 11$

$f(2) = g(2) + h(2) = 2 + 11 = 13$

$f(3) = g(3) + h(3) = 3 + 18 = 21$

$f(5) = g(5) + h(6) = 1 + 10 = 11$

$f(9) = g(9) + h(9) = 2 + 9 = 11$

→ when the heuristic is UNDERESTIMATE i.e $\forall n, h(n) \leq h^*(n)$

Eg: $h_1(n_1) = 5$ $\quad h_1(n_2) = 6$

$h_2(n_1) = 7$ $\quad h_2(n_2) = 4$

→ Given K UNDERESTIMATED heuristic derive the best heuristic

$h_1, h_2, \ldots h_K \Rightarrow$ K underestimate heuristic

the best heuristic will be $\max(h_1, h_2 \ldots h_K)$ as it would be closest to the optimal heuristic value.

∴ BEST: $\forall n \quad \max(h_1, h_2, \ldots h_K)$

→ If $h(n) = 0$ then $f(n) = g(n) \Rightarrow$ algo. to be applied is UCS

A* Algorithm :

→ Initialization : OPEN = [s], CLOSED = [], s.cost = f(s) = h(s)

→ Iteration : Remove the lowest cost (based on f(n)) state n from OPEN

. . . . .

→ Selection : (m.cost = n.cost + cost(n,m)) ✗

(*) { f(m) = g(m) + h(m)
g(m) = g(n) + cost(n,m)

(m.cost > n.cost + cost(n,m)) ✗

(*) { g(m) > g(n) + cost(n,m)

If m ∉ [OPEN ∪ CLOSED]
g(m) = g(n) + cost(n,m)

If m ∈ [OPEN ∪ CLOSED]
g(m) = min(g(m), g(n) + cost(n,m))

If g(m) is ↓ & m ∈ CLOSED
put m in OPEN

→ VCS : UnInformed

A* : Informed

→ h(n) = 0 ⇒ UCS ⇒ Many Nodes will be Expanded

h(n) = h*(n) ⇒ min^m nodes will be expanded

→ A* uses heuristic search with underestimated heuristic

It always gives optimal solution

Q) which nodes are being expanded in UCS but not in A* ?

In UCS :

$$\forall n,\ g(n) \le g^*(G)$$

$$C(n) \le C^*(n)$$

In A* :

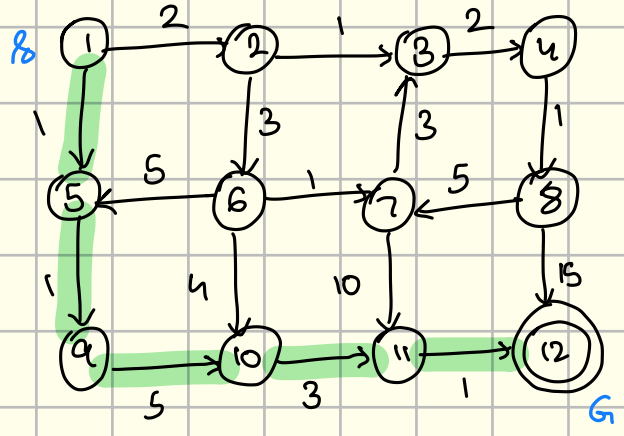
$$\forall n,\ f(n) \le \left(g^*(G) = f^*(G)\right)$$

$$g(n) + h(n)$$

$$g(n) < g^*(G) < f(n)$$

$$g(n) + h(n)$$

these n's will be expanded in ucs not in A^*



→ for node (2)

$g(n) = g(2) = 2$

$f(2) = 13$

$g^*(12) = 11$

$g(n) < g^*(G) < f(n)$

$2 < 11 < 13$

→ which nodes are being expanded in ucs but not in A^*

$$\forall n \quad g(n) < \left[\begin{matrix} g^*(G) \\ = f^*(G) \end{matrix} \right] < f(n)$$

→ In A^* ; take priority Q , based on $f(n)$

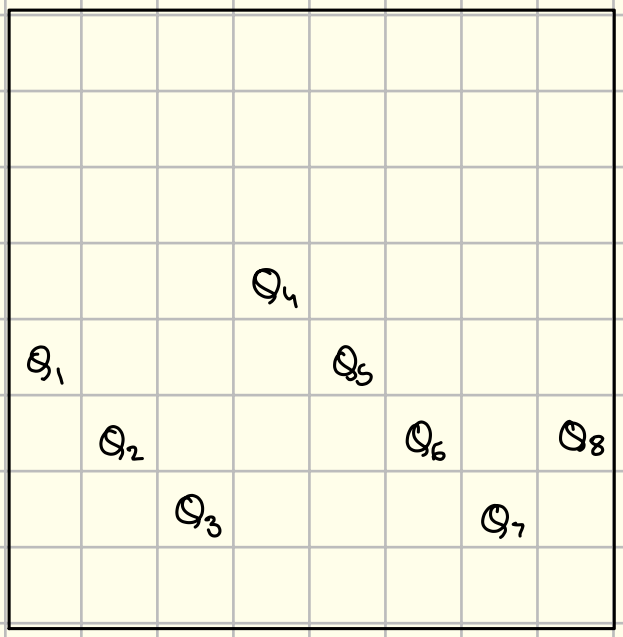
In ucs ; " " " " $g(n)$

LOCAL SERCH :

HILL CLIMBING :

→ for 8 Queen Problem:

$h(n)$: # pair of Q's in attacking position



$Q_1 \rightarrow (Q_2, Q_3, Q_5)$ = 3

$Q_2 \rightarrow (Q_3, Q_4, Q_6, Q_8)$ = 4

$Q_3 \rightarrow (Q_5, Q_7)$ = 2

$Q_4 \rightarrow (Q_5, Q_6, Q_7)$ = 3

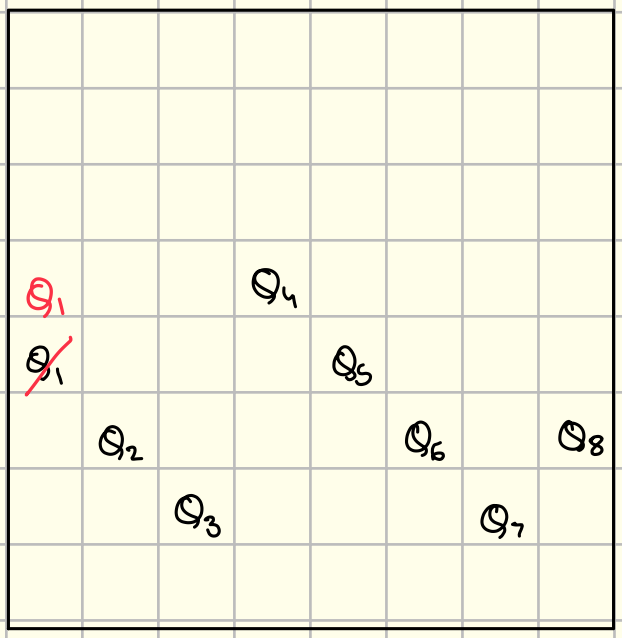
$Q_5 \rightarrow (Q_6, Q_7)$ = 2

$Q_6 \rightarrow (Q_7, Q_8)$ = 2

$Q_7 \rightarrow (Q_8)$ = 1

$Q_8 \rightarrow ()$ = 0

$h(n) = 17$



$Q_1 \rightarrow (Q_4)$ = 1

$Q_2 \rightarrow (Q_3, Q_4, Q_6, Q_8)$ = 4

$Q_3 \rightarrow (Q_5, Q_7)$ = 2

$Q_4 \rightarrow (Q_5, Q_6, Q_7)$ = 3

$Q_5 \rightarrow (Q_6, Q_7)$ = 2

$Q_6 \rightarrow (Q_7, Q_8)$ = 2

$Q_7 \rightarrow (Q_8)$ = 1

$Q_8 \rightarrow ()$ = 0

$h(n) = 15$

→ HC is a local search technique

→ 1st Choice Hill Climbing:

from the successor, choose 1st better i.e 4

→ Steepest Ascent Hill Climbing:

look all 56 neighbours, from these neighbours choose the

best i.e 12

→ Stochastic Hill Climbing:

look the better neighbour than (17),

from them Probabillistically choose better uphill move

(if maximization)

→ Random Restart Hill Climbing: (RRHC)

Multiple time, Run H.C (hill climbing) with random initial state each time

→ limitation of HC: may get stuck in local maximum / minimum

→ In H.C we are not seeing bad solution.

→ H.C is also called as a greedy local search algorithm as it always grabs a good neighbour without thinking about the next step.

→ Geometric distribution: $p_x(x) = (1-p)^{x-1} p$

$\text{Mean} = \frac{1}{p} \quad \Rightarrow p = \frac{1}{T}$

SIMULATED ANNEALING:

→ If we get a bad solution: choose that with some probability

→ $\Delta E = -\alpha$: -ve

As, $T \to 0$ prob: $e^{\frac{\Delta E}{T}} = e^{\frac{-\alpha}{T}} = \frac{1}{\infty} \approx 0$

Hill Climbing

Boils down to

⇒ Probability = 0 ⇒ Not taking bad solution

As, $T \to \infty$ Prob: $e^{\frac{\Delta E}{T}} = e^{\frac{-\alpha}{T}} = \frac{1}{1} \approx 1$

whatever solution we get we consider it

- If $\Delta E < 0$. ⇒ Always take worst state
- If $\Delta E > 0$, ⇒ better solution

(RANDOM WALK)

Few points on A*

1. A heuristic is called admissible if it always underestimates. ie, we always have $h(n) \leq f^*(n)$, where $f^*(n)$ denotes the minimum distance to a goal state from state n.

* For finite state spaces, A* always terminates

At any time before A* terminates

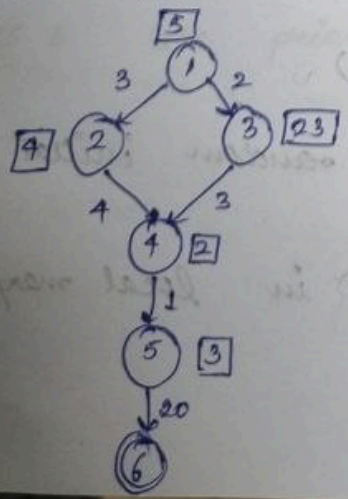
2. There exists in OPEN, a state n that is on an optimal path from s to a goal state with $f(n) \leq f^*(s)$

* If there is a path from s to a goal state A* terminate (even when the state space is infinite)

3. Algo A* is admissible, ie. if there is a path from s to a goal state, A* terminates by finding an optimal path.

* If A_1 and A_2 are 2 versions of A*, such that A_2 is more informed than A_1, then A_1 expands at least as many states as A_2

* If we given 2 or more admissible heuristic we can take their MAX to a stronger admissible heuristic



$f(n) = g(n) + h(n)$

$h(n)$ always underestimate heuristic

Beyeam A* steps

OPEN

$1^{(5)}$
$2^{7}, 3^{25}$
$3^{25}, 4^{9}$
$3^{25}, 5^{11}$
$3^{25}, 6^{28}$
$6^{28}, 4^{7}$
$6^{28}, 5^{9}$
6^{26}

CLOSED

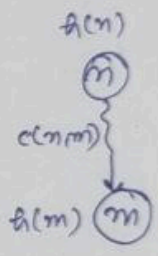
$1^{5}, 2^{7}, 4^{9}, 5^{11}$
$3^{25}, 4^{7}, 5^{9}, 6^{26}$

* A* heuristic, if monotone and admissible, then we don't need to transfer any NODE from CLOSED to OPEN, but if non-monotone then we may need to transfer.

consistent / Monotone

for every node n, $h(n) \leq c(n,m) + h(m)$

$h(n) - c(m,n) \leq h(m)$



* If the monotone restriction is satisfied, then A^* has already found an optimal path to the state it selects for expansion.
* If the monotone restriction is ~~not~~ satisfied, the f-value of states expanded by A^*, is non-decreasing.

Path Max

converts a non-monotone heuristic to monotone heuristic

① During generation of successor m of n

set $h'(m) = \max [h(m), h(n) - c(m,m)]$

and use $h'(m)$ at m

admissible (underestimate) → optimal

overestimate can still reach