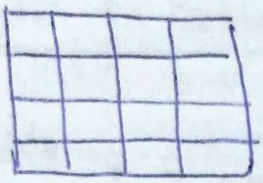
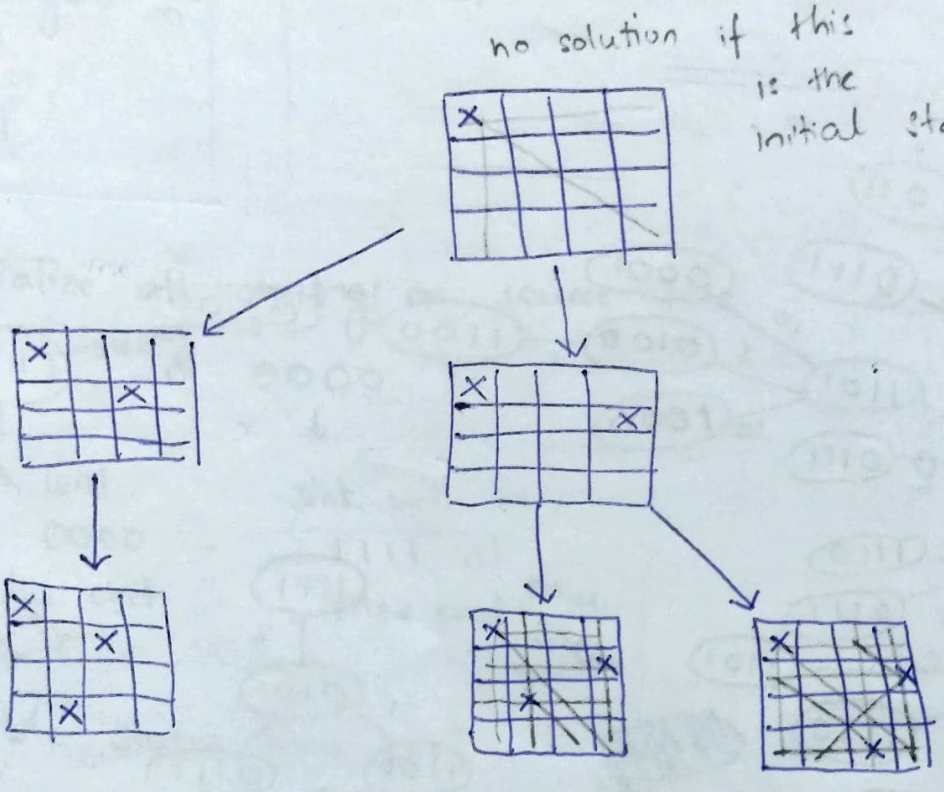
4 Queens Problem

q can move ↑, ↓, ←, →, diagonally



4 queens.

place 4 q such that they won't attack each other.



★ For informed search, the h(n) could be # of tiles where we cannot place another queen.

15.12.2022

2 Player Games

adversiaral search

Maximizer : tries to maximizes his chance to win

Minimizer : tries to minimize the chance of opponent (maximizer) to win

Game tree : level 1 for player 1, level 2 for player 2, and so on.

⇓ maximizer

⇓ minimizer

Q. Tower of blocks

2 players

Moves: remove either 1 block or remove 2 blocks

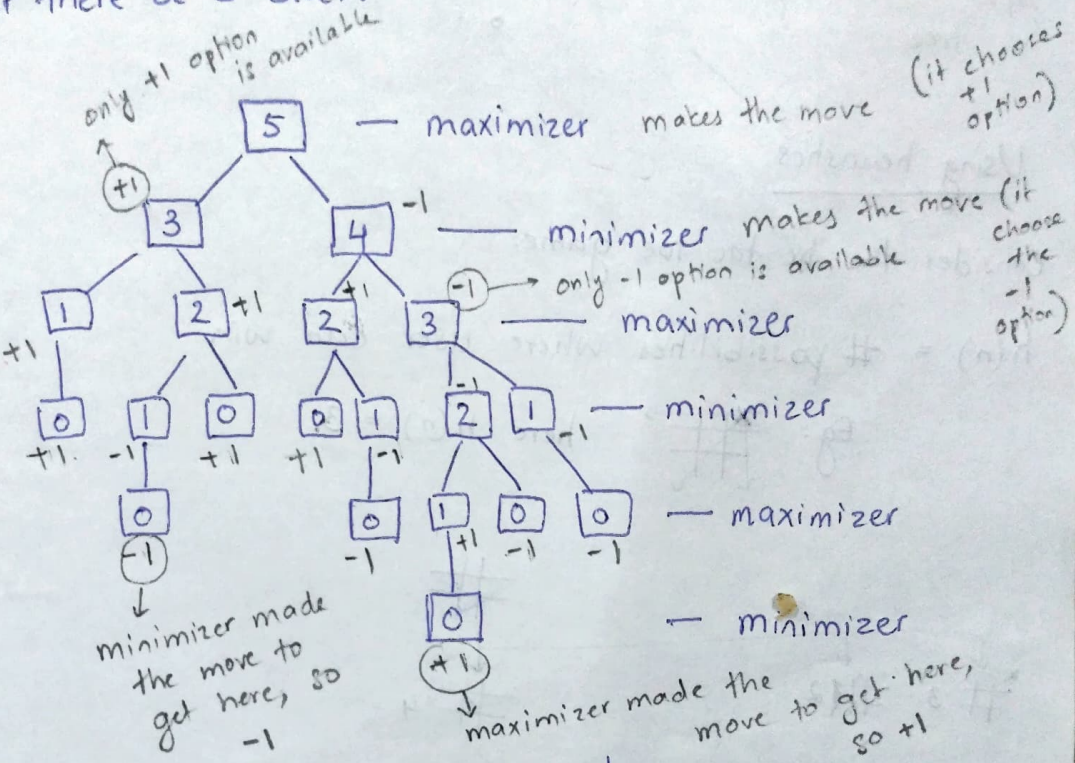
goal: be the 1st to remove all blocks (make tower empty)

initial state: let there be 5 blocks.

[This is the game tree]

⇓

called Min - Max Search



possibilities or moves ↑, complexity ↑

★ For every state, we should give a utility value.

+ve = good move

-ve = bad move

The leaf nodes represent the goal, i.e. no more moves, so we can consider it like a failure.

utility = -1 for leaf nodes.

~~All 1 in maxi~~

Consider: +1 = maximizer is winning

-1 = minimizer is winning

If leaf Maximizer leaf nodes will be given -1 and minimizer leaf nodes will be given +1.

-1 → any -ve utility value

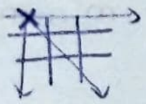
+1 → any +ve utility value

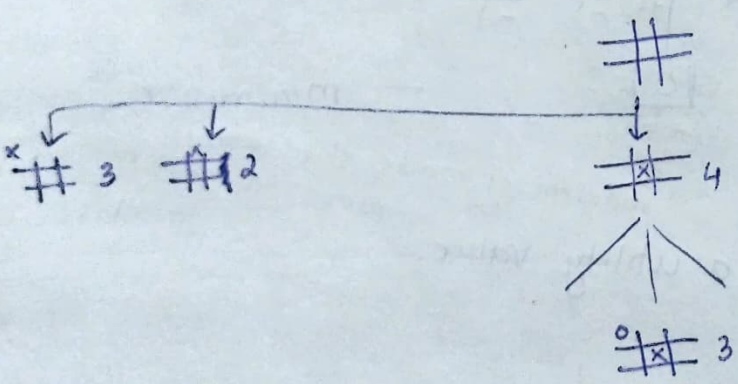
Implementation:
backtrack using DFS after constructing the tree

Using heuristics

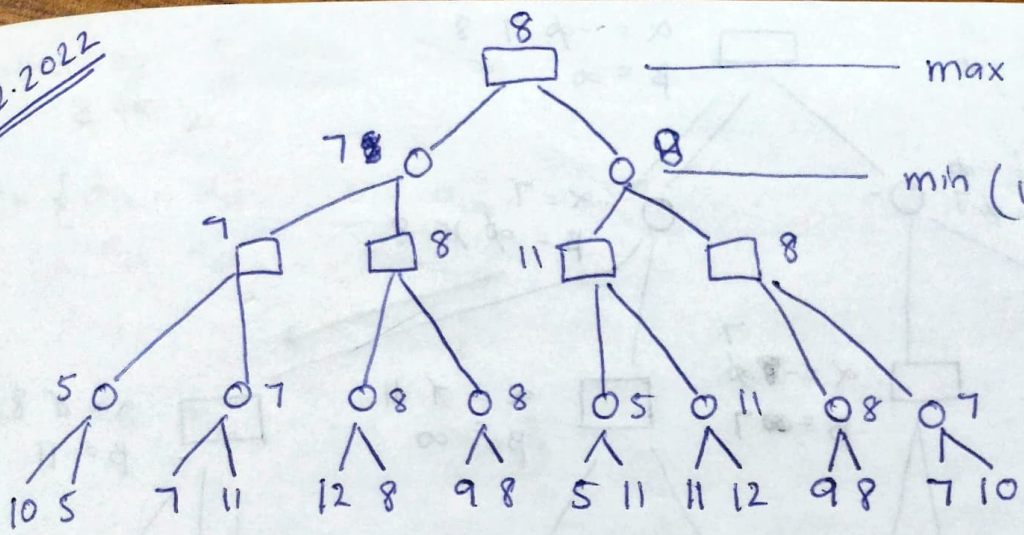
Consider the tic tac toe game.

h(n) = # possibilities where user can win

Eg:  here h(n) = 3

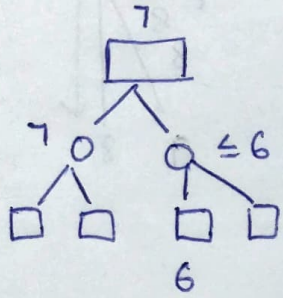


17.12.2022



Without exploring the path, we can find the max utility.

Eg:



I.e. we are pruning the game tree.

α-β Pruning Technique

α is for the max nodes, initial value = $-\infty$

β is for the min nodes, initial value = $+\infty$

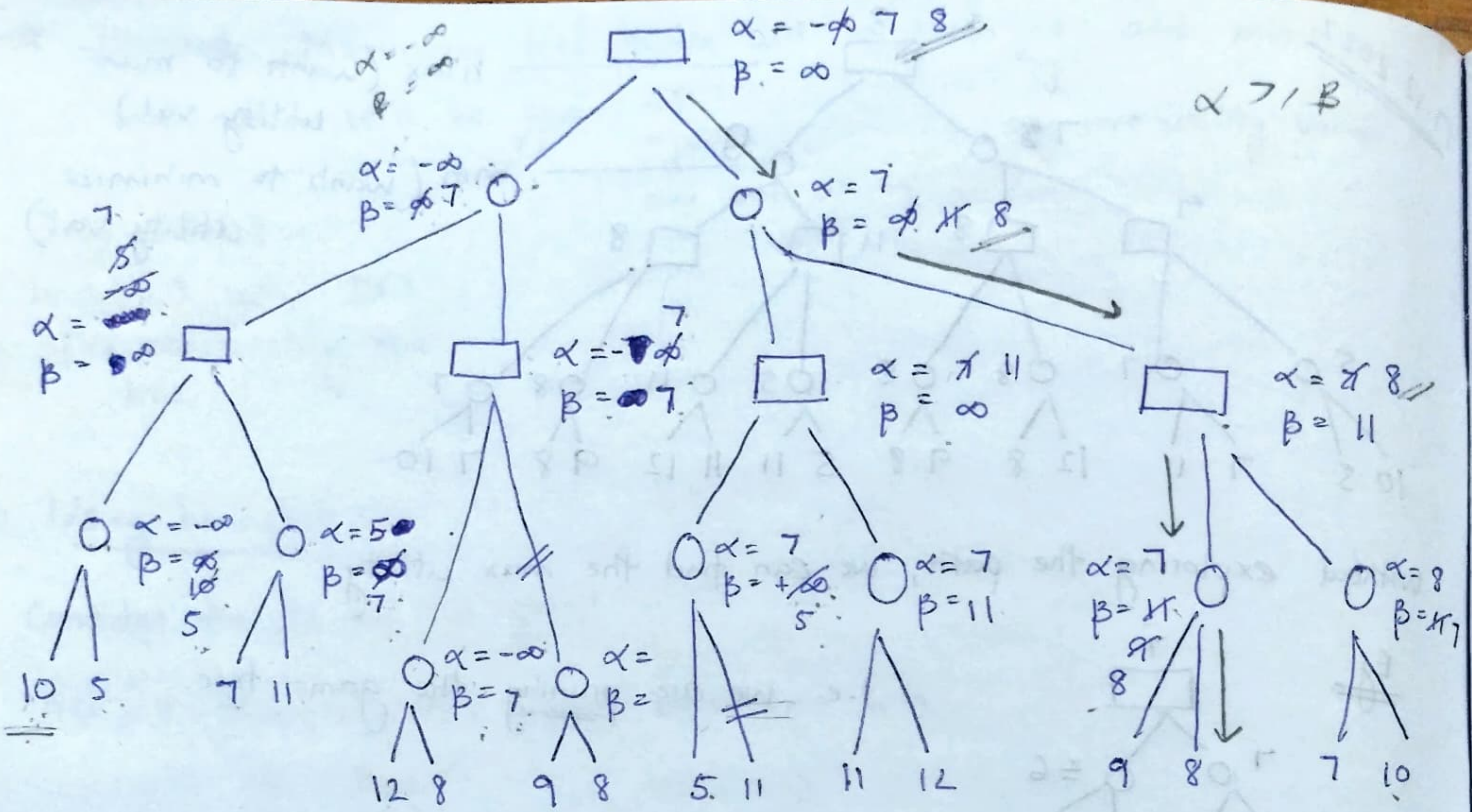
When $\boxed{\alpha \geq \beta}$, prune (the other branch) $\Longrightarrow$  means pruned

- min nodes change β, not α
  max nodes change α, not β.

$\alpha =$
$\beta =$

$\alpha \geq \beta$

- We perform DFS
- To get the path:
  At max node, choose the node that gave the α value.
  At min node, chose the node that gave the β value



20.12.2022

Constraint Satisfaction Problem (CSP)

$V = \{R_1, R_2, R_3, R_4, R_5, R_6\}$ ⇒ variables

$D = \{Red, Green, Blue\}$ ⇒ domain

$C = \{(scope, relation)\}$ pair ⇒ constraints

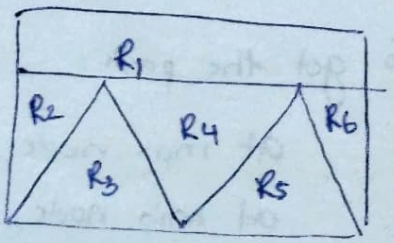
$= \{(R_1, R_2, R_3, R_4, R_5, R_6)\ (R_1 \neq R_2, R_3, R_4, R_5, R_6)$

$(R_2, R_3)\ (R_2 \neq R_3)$

$(R_3, R_4)\ (R_3 \neq R_4)$

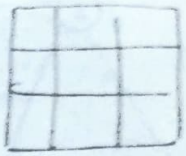
$(R_4, R_5)\ (R_4 \neq R_5)$

$(R_5, R_6)\ (R_5 \neq R_6)\}$



map colouring problem

Another eg: Sudoku



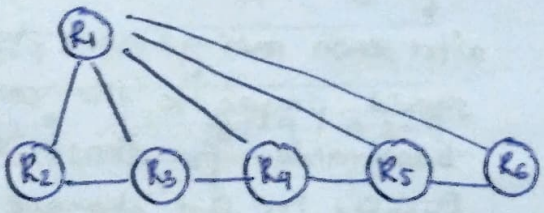
$V = \{ a_{11}, a_{12}, \ldots a_{19}$
$a_{21}, a_{22}, \ldots a_{29}$
$\vdots$
$a_{91}, a_{92}, \ldots a_{99} \}$

$D = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$

$C =$

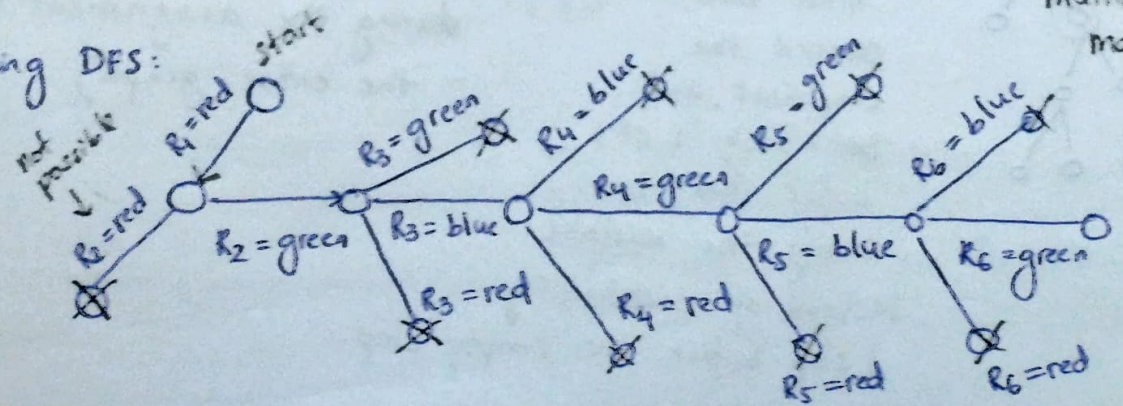
Other examples: crossword, N-Queens, cryptarithmetic

How to solve map colouring? reduce it to a graph problem.

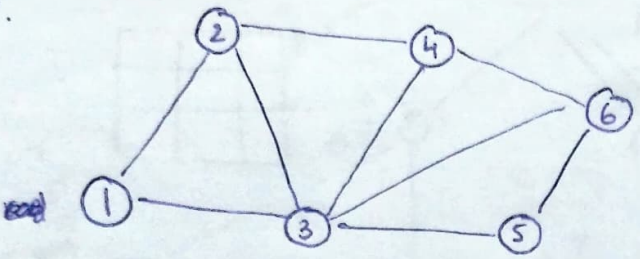


For all possible solutions, perform complete DFS multiple sols may exist

→ using DFS:



Q.



Forward Checking

We remove the values from the D of each node based on the assignment at each step.

	1	2	3	4	5	6
initial	RGB	RGB	RGB	RGB	RGB	RGB
1 = R	R	GB	GB	RGB	RGB	RGB
4 = G	R	B	B	G	RGB	RGB
2 = B	R	B	—	G	RGB	RGB

cannot proceed → backtrack

↓

Adv: less exploration (removed some E)

Dis: we may identify wrong assignments at a very later point

↳ Sol: constraint propagation or IB

★ Constraint propagation

for each assignment, check if the previous assignments still hold

i.e. change the domain values of the affected variables and check if they are consistent. with the con

↓

after each assignment, we should propagate the constraints backwards for those variables whose D got changed during the assignment (not the entire graph)

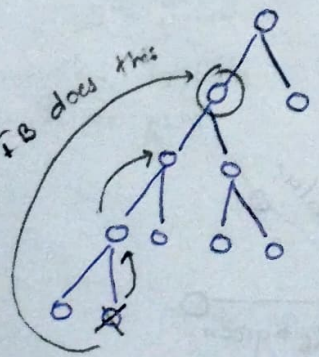
★ Intelligent backtracking (IB)

Searching

↓

find the move that caused the emptiness and backtrack 1 step before that to rectify the mistake.

Instead of backtracking 1 step before the empty step.



Cryptarithmic Puzzles

$$\begin{array}{r} TO \\ GO \\ + \\ \hline OUT \end{array}$$

$V = \{T, O, G, U\}$

$D = \{0-9\}$

$C = \{(T\times10 + O) + (G\times10 + O) = (O\times100 + U\times10 + T)\}$

$O = 1$.

$$\Rightarrow \begin{array}{r} T\ 1 \\ G\ 1 \\ + \\ \hline 1\ U\ T \end{array} \xrightarrow{T=2} \begin{array}{r} 2\ 1 \\ G\ 1 \\ + \\ \hline 1\ U\ 2 \end{array}$$

or $2 + G = 10 + U$

$2 + 9 = 10 + 1$ ✗

$2 + 8 = 10 + 0$ ✓

$\Rightarrow G = 8,\ U = 0$

∴ $O = 1$

$T = 2$

$G = 8$

$U = 0$

Using FC:

	T	O	G	U
	0-9	0-9	0-9	0-9
O=1	0,2-9	1	0,2-9	0,2-9
T=2	2	1	0,3-9	0,3-9

Q.

```
   8 1
   E A T 9
 9 2 1
 T H A T 9 +
-----------
 A P P L E
 1 0 0 3 8
```

	T	E	A	H	P	L	
	0-9	0-9	0-9	0-9	0-9	0.9	
A=1	0,2-9	0,2-9	1	0,2-9	0,2-9	0,2-9	
L=2	~~0,3-9~~	~~0,3-9~~	~~1~~	~~0,3-9~~	~~0,3-9~~	~~2~~	→ may cause backtracking ↓ consider T instead
T=9	9	0,3-8	1	0,3-8	0,3-8	0,3-8	
E=8	9	8	1	0,3-7	0,3-7	0,3-7	
L=3	9	8	1	0,4-7	0,4-7	3	
P=0	9	8	1	4,7	0	3	
H=2	9	8	1	2	0	3	

Q.

```
  C4=1 C3=1 C2=0 C1=0 C0=0
     1  9  3  4
     S  O  M  E
  9T I5 3M E4 +
-------------
 S P E N T
 1 0 4 6 8
```

~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ ~~9~~

∴ S=1, T=8, P=0, E=4, O=9, I=5, M=3, N=6, $C_4=1$, $C_3=1$, $C_2=0$

$1 + T + C_3 = 10 + P \Rightarrow T=8 \quad C_3=1$

$\Rightarrow E=4$

$O + I = 10 + 14 = 14$, if $C_2=0 \Rightarrow O=9, I=5, C_2=0$

$2M = N \Rightarrow M=3, N=6$

3.1.2023

Planning — total
— partial

We plan the sequence of actions to reach the goal state from initial state.

Almost similar to problem searching if env:
- static
- fully observable
- deterministic

(state space search is applicable only if the env satisfies these properties)

1. Representation of the world
2. Initial state rep using conjunction of +ve literals
3. Goal state "
4. Actions

} to represent a planning problem, we need.

↓

actions represented in stripe format

We want to find a sequence of actions to go from initial to goal.

★ Causal dependency: between the actions should be specified

$a_1 \longrightarrow a_2$, from set of actions $a_1 \cdots a_n$

a_2 is possible only if a_1 has already happened.

Cargo Transportation Problem

★ FOL can be used to represent the world

Suppose there are 3 airports, 5 cargos, 2 flights

Eg: ~~Airport(A_1)~~

Initial:

$Airport(A_1) \wedge Airport(A_2) \wedge flight(F_1) \wedge flight(F_2) \wedge cargo(C_1) \wedge cargo(C_2) \wedge cargo(C_3) \wedge At(C_1, A_1) \wedge At(C_2, A_1) \wedge At(C_3, A_1)$ ★ $\wedge At(F_1, A_1) \wedge At(F_2, A_2)$

Goal:

$At(C_1, A_2) \wedge At(C_2, A_2) \wedge At(C_3, A_2)$

initial and precond should be complete

* Actions can be represented using a set of preconditions and postconditions

preconditions → using a set of +ve literals

postconditions → using a set of +ve and -ve literals

if a literal is not mentioned in initial or precond, then its negation is true

Actions:

1. Load (C_1, A_1)

$\forall A_1$

preconditions: $At(C_1, A_1) \wedge At(F_1, A_1) \wedge cargo(C_1) \wedge flight(F_1) \wedge Airport(A_1)$

post conditions:

$\sim At(C_1, A_1) \wedge In(C_1, F_1)$

we don't need to define this in the initial (considered as $\sim In(C_1, F_1)$ automatically in initial)

2. $Fly(F_1, A_1, A_2)$

preconditions:

$Flight(F_1) \wedge At(F_1, A_1) \wedge Airport(A_1)$

postconditions:

$\sim At(F_1, A_1) \wedge At(F_1, A_2)$

3. Unload (C_1, F_1)

mandatory to define in preconditions

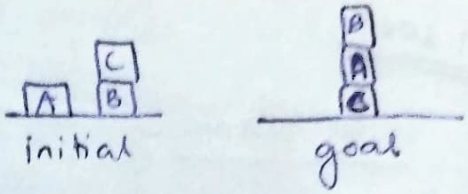
preconditions:

$In(C_1, F_1) \wedge At(F_1, A_2) \wedge cargo(C_1) \wedge flight(F_1) \wedge Airport(A_2)$

postconditions:

$\sim In(C_1, F_1) \wedge At(C_1, A_2)$ ~~$\wedge cargo(C_1) \wedge flight(F_1) \wedge Airport(A_2)$~~

Q. Represent the blocks world problem.



Initial :

on (A, table) ∧ on (B, table) ∧ free (A) ∧ free (C)
∧ block (A) ∧ block (B) ∧ block (C) ∧ on (C, B)
∧ free (robot)

Goal:

on (C, table) ∧ on (A, C) ∧ on (B, A) ∧ free (B)

Actions :

1. pick (C)

precond: free (C) ∧ free (robot) ∧ block (C) ∧
$\exists_A$ [on (C, A) ∨ on (C, table)]

postcond:
~free (robot) ∧ In (C, robot) ∧ free (B)

pick (B)
pre: hand (free) ∧
free (B) ∧
$\exists_A$ [on (B, A) ∨
on (B, table)]
post: hand (B) ∧
free (A)

2. place (C, A)

precond: In (C, robot) ∧ free (A) ∧ block (C) ∧ block (A)

postcond:
~In (C, robot) ∧
~~right (A, robot)~~ free (robot) ∧ ~free (A) ∧ on (C, A)

3. place (C, table)

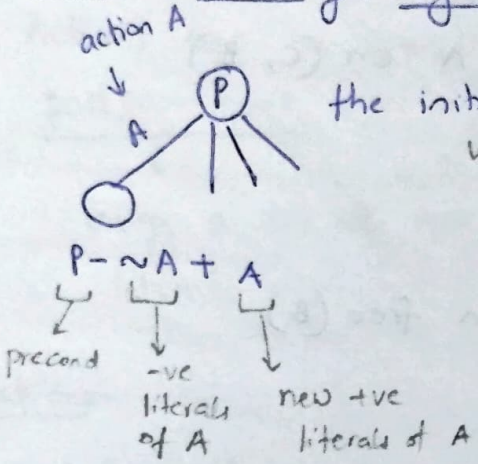
precond: In (C, robot) ∧ block (C)

postcond: ~In (C, robot) ∧ free (robot) ∧ on (C, table)

5.1 2023

Total planning

1. Forward Searching / Progression



the initial state contains a set of literals : p which are the preconditions

Eg : buy (isbn)

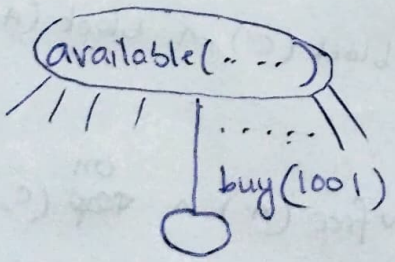
precond : available (isbn)

postcond : owns isbn

~available (isbn)

Consider that there are 5 lakh. books, we want 1001 i.e. buy(1001)

Then :



* Disadvantage : ~~goal oriented~~, so many branches

2. Backward Searching / Regression (goal focused search)

initial state is goal state, and aim is to find the initial state.

Adv: branching factor is limited

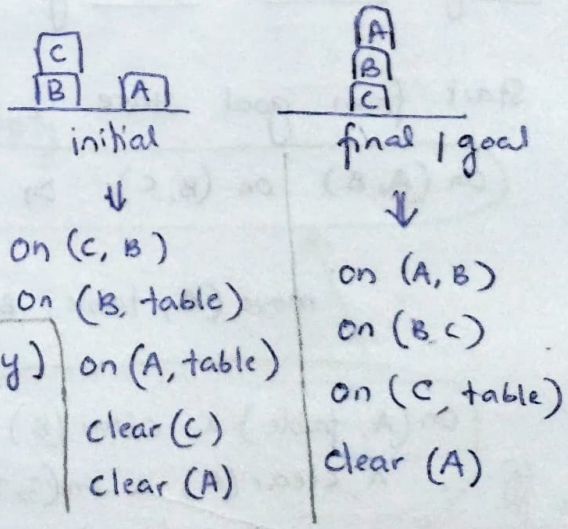
we can apply heuristics

Every Action has:
- A - Action
- P - Precondition
- E - Effect List (-ve list, +ve list)

- Given: goal state G with set of literals L
- Find the action whose postcondition contains L ⇒ action we are going to do
- ~~Negation of~~ Find the state whose preconditions are there in the cur state - negate list ~~literals~~
  - ↳ to verify: its negations should be present in the postconditions of the action (or prev state?)

Continue until we reach

Q. Blocks world problem.



Actions:

1. move (b, x, y)

   Pre: clear (b) ∧ on (b, x) ∧ clear (y)

   blocks (b, x, y)

   ~~pre~~

   effect: (negate list) ~clear (y)

   (positive list) clear (x)

   on (b, y)

   } ~clear (y) ∧ clear (x) ∧ on (b, y)

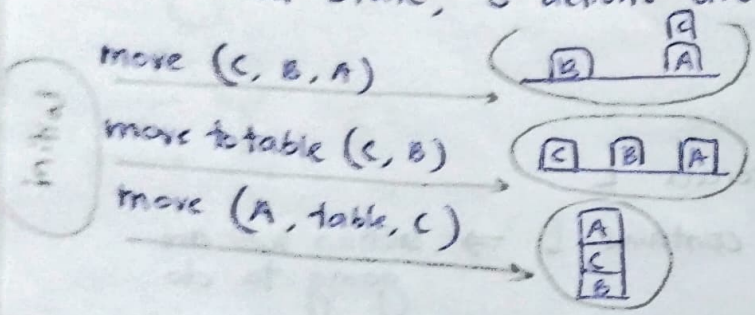
2 movetotable (b, x):

Pre: clear (b) ∧ on (b, x) ∧ blocks (b, x)

Effects: clear (x)

Using Forward searching:

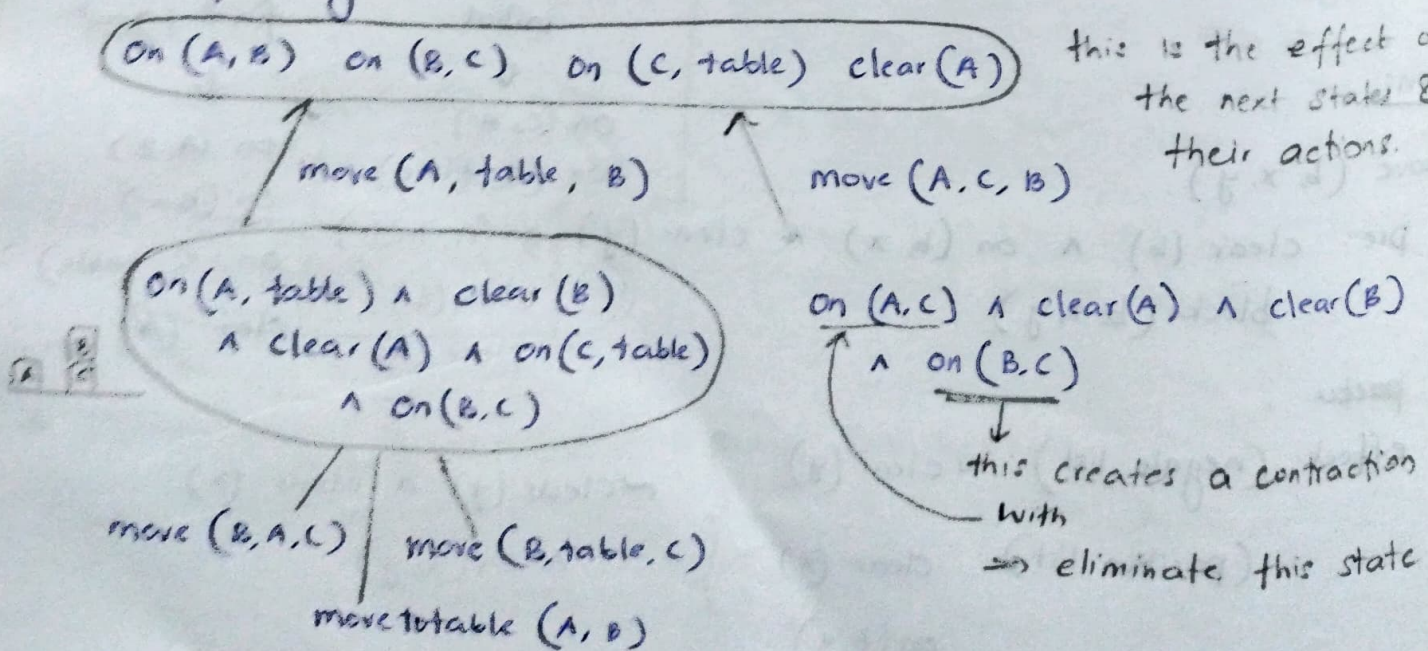
From initial state, 3 actions are possible:



From the preconditions of each state, find the possible actions for each state.

Using Backward searching:

Start from goal state



find the actions that led to the current possible states until we reach the initial state.

17.1.2023

Goal Stack Planning backward search for plan

we have: stack, initial, goal, actions

start from goal → push into stack

push the preconditions.

Eg: Blocks world problem

pickup (x) : from table

pre: on (x, table), clear (x), arm empty

effect: ~~not~~ ~arm empty, holding (x)

stack (x, y) :

pre: holding (x), clear (y)

effect: on (x, y), ~holding (x), arm empty

putdown (x) : on to the table

pre: holding (x)

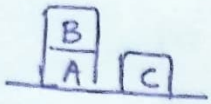
effect: on (x, table)

Unstack (x, y) :

pre: on (x, y), clear (x), arm empty

effect: clear (y), holding (x), ~arm empty

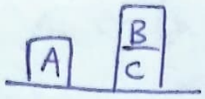
initial:



↓

on (A, table) ✓
on (C, table) ✓
on (B, A)
clear (B) ✓
clear (C)
arm empty ✓

goal:



↓

on (A, table) ✓
on (C, table) ✓
on (B, C)
clear (B) ✓
clear (A)
arm empty ✓

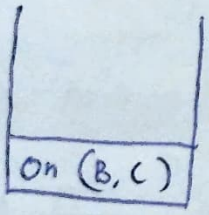
1. Find (goal - initial)
   ↳ push into stack
2. until not empty
   1) pop
   2) satisfied → nothing
   3) not → find action whose effect contains it
   4) push its pre into stack

if 2 actions choose 1 but later come back to it ↳ if not isn't possible

if all its precond are satisfied, we can perform that action ↳ update the current state

Find the diff b/w initial and goal.

on (B, C) is not in initial, add to stack.



pop

check if it's satisfied → is it in the goal state? or contains on (B, C) ←

if not, then check which action has effect on (B, C)

and push its precond

↓

pickup, stack, putdown, unstack