4.02.2024

Iterative deepening A*

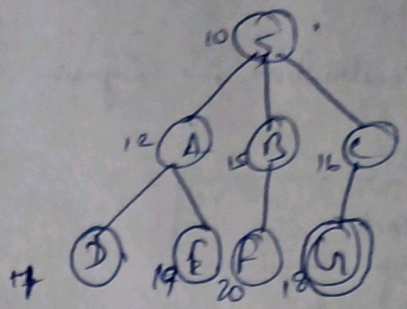
TH = f(start).

Do DFS and prune when f value of node is greater than TH.

i) If goal is found return.

ii) Update TH and TH = minimum f values of the nodes pruned.

Goto step 2.



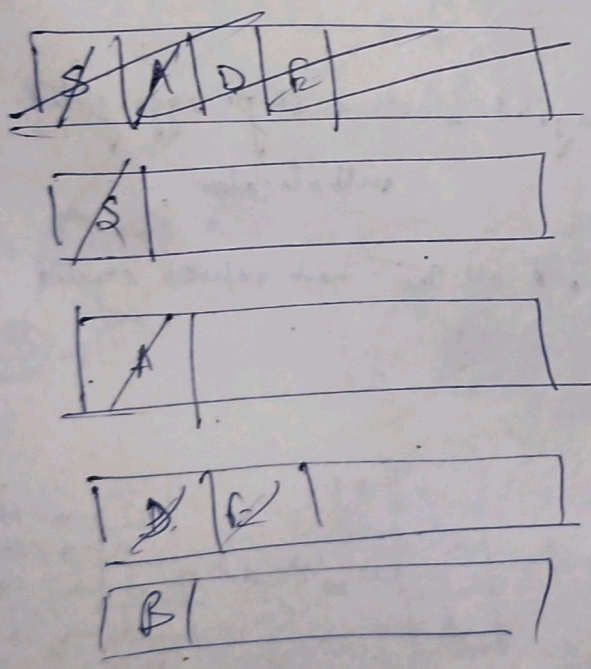
Worst case → All node f values are different.

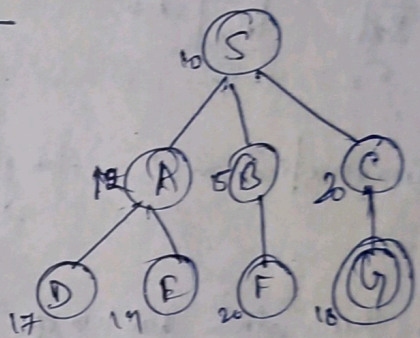
Time complexity = $\frac{k(k+1)}{2}$.

$= O(k^2)$.

Space complexity (Reduced).

Recursive Best First Search (RBFS)





Simplifying Memory-bounded A* algorithm (SMA*)

Local Search

28/2/24

Hill Climbing Search

Simple Hill Climbing Algorithm:

1. Evaluate the initial state. If initial state is goal then return.
   Otherwise, Consider initial state as current state.
2. Loop Until goal is found or no action left to be applied to current state.
   i. Apply an action that is not applied to Current state.
   (ii) Evaluate the new state.
   (a) If new state is goal then return.
   (b) If new state is better than current state then Consider it as a current state.
   (c) If it is not better, then go to step 2.

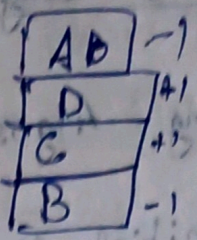
Diff. with Best first search:

(1) In hcs we have an OL. If we are stuck (no action) at a state, we can choose another path.

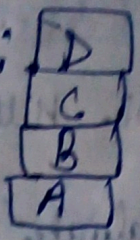
(2) Hill climbing compares the successor with current node, in BFS, all the successor nodes are compared and choosen.

E.g.

Initial state:



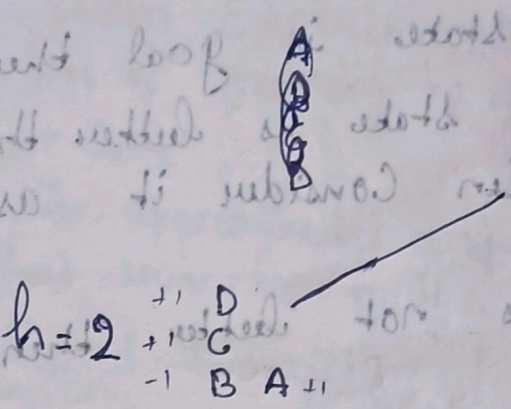
Goal:

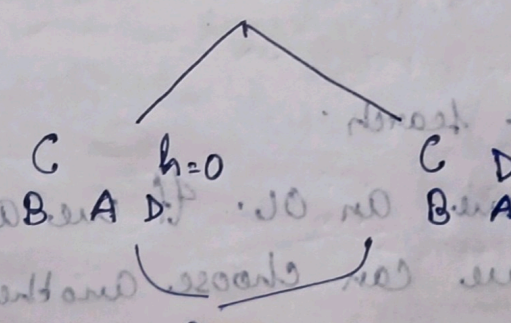


Possible heuristic:- If a block is resting on proper goblock then +1 point
else -1 Point.

h(Goal) = 4

h(Initial state) = 0





Both are not better than previous state i.e. Hill climbing stops here

Benefits:-

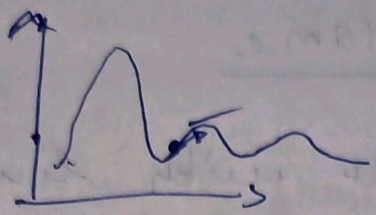
1. Simplicity
2. Less complexity.

Modified heuristic - All the blocks below it is correct then +1

Modified heuristic – All the blocks below it is correct → +1 else → -1

Problems of hill climbing ~~search~~ search:-

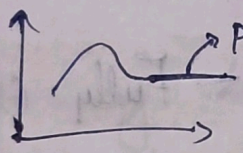
① Local Maxima.



Since it may not get most optimal goal. Does not guarantee optimal soln.

Soln:- We may backtrack and move to other areas of search space.

② Plateau.

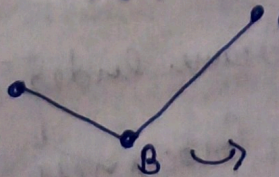


All the next ~~actions~~ states are equally better than current state.

Arbitrary choose action.

③ Ridge

There are better states but you cannot reach better state using a single action.



8

Solution:- Take two or more actions simultaneously to find better states.

5/3/24

Game

* Problems till now dealt with single agent

Multiagent Environment → Unpredictability

Agent
- competitive → *agent's action are against. *adversarial search.
- Cooperative

Game should be

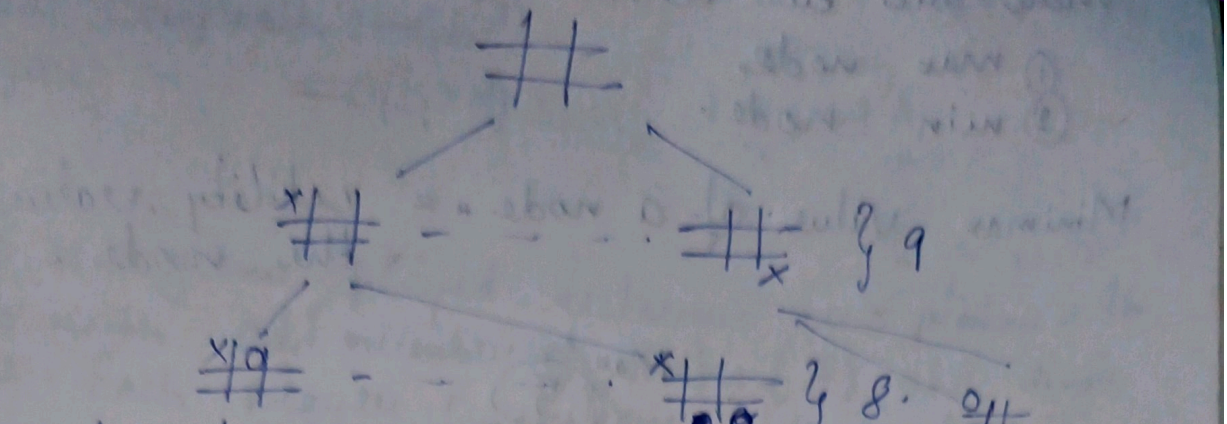
* 1. Deterministic
2. Perfect information — Fully Observable.
3. Zero sum — Only one can win the match, other loose
4. turn-taking — one after another

Zero sum → (+1) P1 wins.
(-1) P2 wins. & 0 sum.

Formal defn of Game.

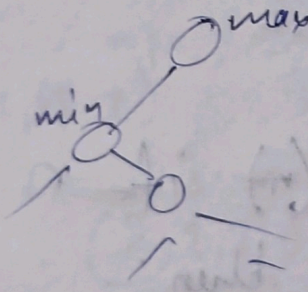
Initial State.
Successor function
Terminal Test — Game ended or not.
Utility function — Goodness of any state

Tic Tac Toe.



- Search adversary in game but not in state space
- Here a player doesn't want to just reach terminal pos., he also has to counter the opponent
- heuristic fn — how far is goal
  But in game, evaluation fn — goodness of a state.
- Others constraints — time etc.

Target — finding strategy to find winning pos. of Pl.



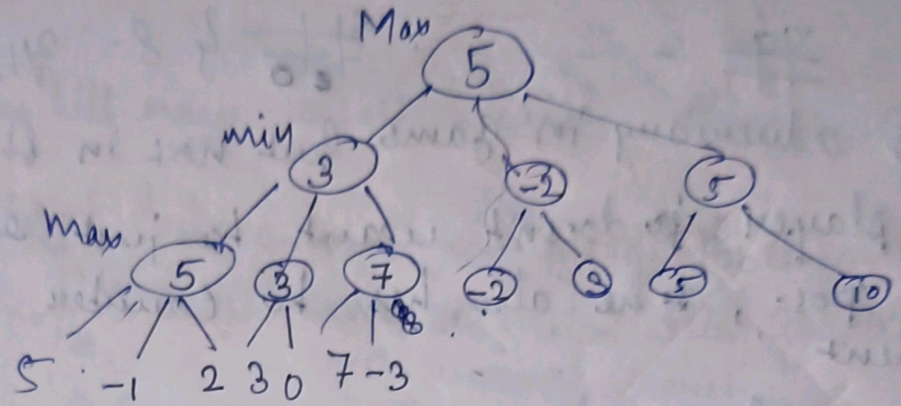
MiniMax Algorithm

Target — Try to reach the terminal position and depending on utility value of terminal node we want to propogate it towards root.

There are two kinds of nodes :-

1. max node
2. min node.

Minimax value of a node = utility value of the node.



Minimax value :-

1. If node is max — max value of all successors
2. If " " min — min " " " " "

Opponent tries to minimise utility value

Minimax Procedure.

~~Minimax(n) = Utility(n) if n is terminal~~

If n is terminal then

Minimax(n) = Utility(n)

If n is max node then

Minimax(n) = $\max_{s \in \text{successor}(n)}$ Minimax(s)

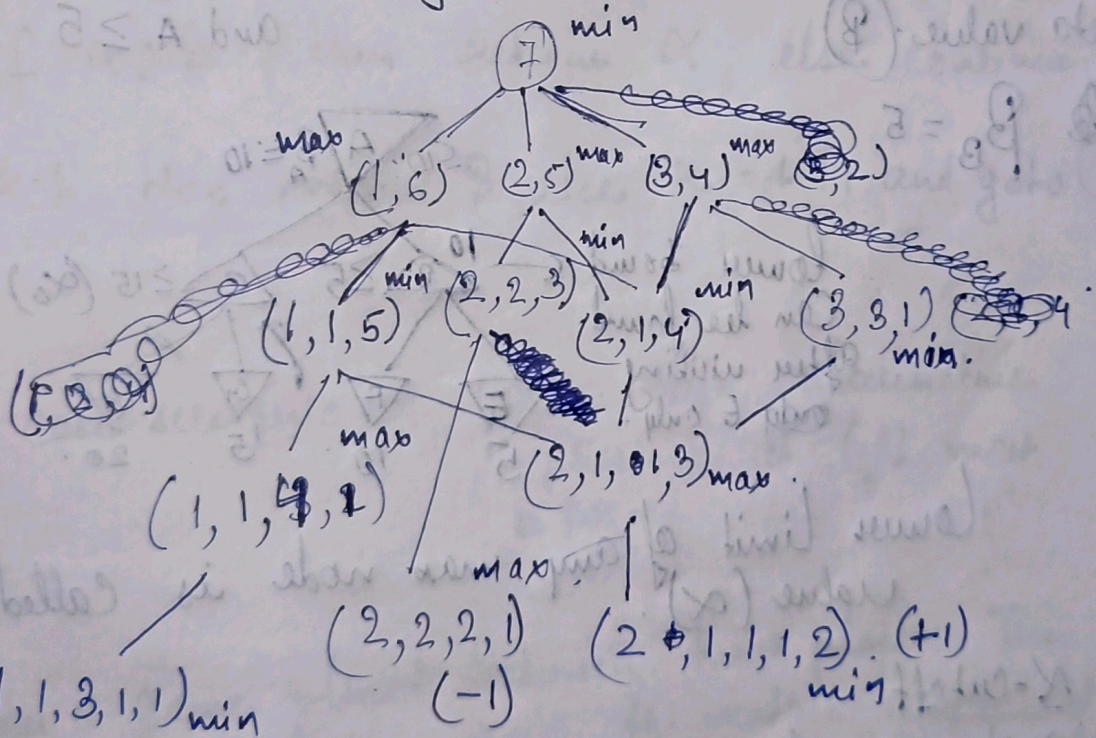
If n is min then

Minimax(n) = $\min_{s \in \text{successor}(n)}$ Minimax(s)

Minimax is just a DFS.

Time Complexity :- $O(b^d)$
Space " - $O(b \times d)$

NIM is a game played between two players. At start with odd number of sticks (7) at a single point. Each player partitions the single pile into two diff piles (>0) each of diff. size. The game comes to end pos. when no player can take a succesful move. The player who cannot give a succesful move first will lose the game. Draw the game. Min starts the game.

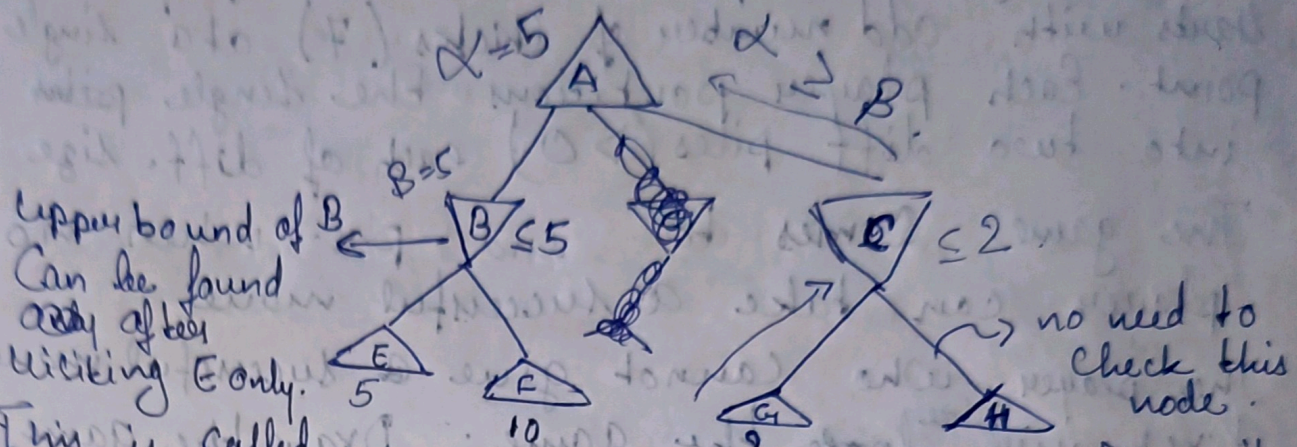


Winning for max = -1
Winning for min = +1

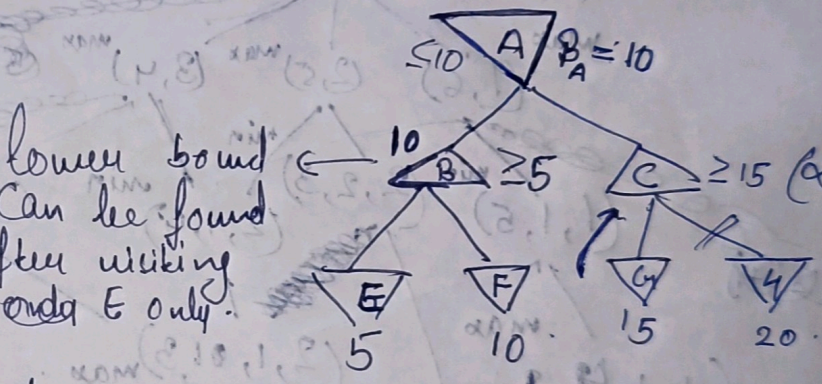
Pruning of Minimax Algorithm.

Alpha-beta Pruning.

max → △
min → ▽



$\beta_B = 5$



Lower limit of any max node is called alpha value (α).

α-Cutoff

① Search may be discontinued at a min node if β value of min node $\leq \alpha$ value of ancestor max node. → α Cutoff

β-cutoff

② Search may be discontinued at a max node if α value of this max node $\geq \beta$ value of ancestor min node.

AB(n, α, β)

1. If n = depth bound, then return AB(n) = Utility
otherwise, find the successor $n_1, n_2 \ldots n_b$ of n.
of n (in order).
And let k=1 and if n = MAX then go to step 2. and if n = MIN then go to step 2'.

(2) $\alpha = \max(\alpha, AB(n_k, \alpha, \beta))$ (2') $\beta = \min(\beta, AB(n_k, \alpha, \beta))$

(3) If $\alpha \geq \beta$ then return β else continue

(4) If k=b then return α, else k=k+1
And go to step (2).

(3') If $\alpha \geq \beta$ then return α else continue.

(4')

(5) If k=b then return β, else k = k+1 and goto (2')

Best case scenario :- When best alternative is present at left most node.

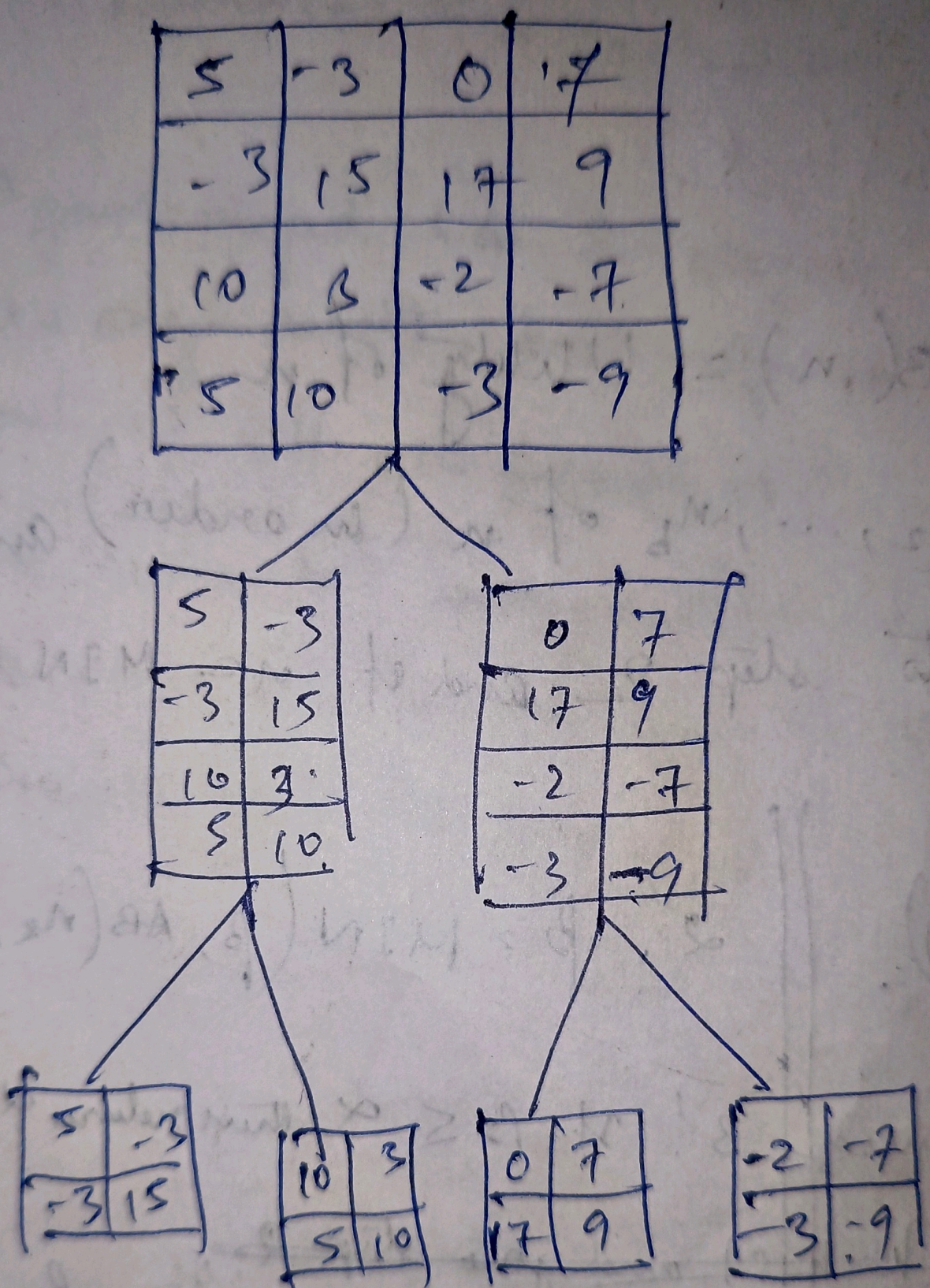
Consider a game played between two players. The Game starts with n×n sq. matrix (n is pow of 2)
Max starts the game by splitting the board in the middle. (vertically) and returns left or right part for min. Min in his turn split the board horizontally in the middle and returns top/bottom to Max.
After $\log_2(n)$ there is a single cell and

~~I cannot split~~
noone can split. Value of cell is P.
The outcome of game depends on value of P. (+P → max will get ≥ P.
-P → min " " ≥ P

How many minimum cell values to be known to max to take best moves.

~~Nilsson~~ Neilson — Reference.

Adversarial Search:


5 -3 0 7
-3 15 17 9
10 3 -2 -7
5 10 -3 -9
5 -3
-3 15
10 3
5 10
0 7
17 9
-2 -7
-3 -9
5 -3
-3 15
10 3
5 10
0 7
17 9
-2 -7
-3 -9

AI — 2

12/3/24.

Intelligent Agent.

1. Perceiving
2. Knowledge Representation.
3. Reasoning.
4. Acting

Knowledge Based Systems-

Knowledge representation as a collection of sentences — we can store knowledge as a coll. of sentences in a language in computer tractable form.

Logic can be used for representing knowledge

Logic.

Syntax — Symbols
Semantics — meaning (Structure)
Inference rule — used for generating new info.

Propositional Logic

declarative statements which tell the status of world.

Symbols — Propositional symbols including True & False.

Well Formed Formula:

E.g. — P : It is hot
Q : It is humid
R : it is raining
$(P \wedge Q) \rightarrow R$.

Semantics tries to find truth value of sentence

A sentence is satisfiable if it is true for any assignment

Propositional Logic

1. Truth Table.
2. Deductive (Proof) System
3. Resolution

$S_1\ S_2 - - S_M$ — M sentences
$x_1, x_2 - - x_n$ — n symbol
$KB = S_1 \wedge S_2 \cdots S_M$ — knowledge base

x_1	x_2	$\cdots$	x_n	S_1	S_2	$\cdots$	S_M	KB	α	KB → α

E.g. — If it is hot and if it is humid then it is raining. — S_1
If it is humid then it is hot — S_2
If it is humid — S_3.

P – hot
Q – humid
R → Raining

(1) $(P \wedge Q) \rightarrow R$ — S_1
(2) $Q \rightarrow P$
(3) Q.

Resolution.

P	Q	R	$P \wedge Q \rightarrow R$	$Q \rightarrow P$	Q	KB	R	KB→
F	F	F	T	T	F	F	F	T
F	F	T						
F	T	F						
F	T	T						
T	F	F						
T	F	T						
T	T	F						
T	T	T						

If the last column contains all true then the implication is correct.

KB → α

Proof System

Inference Rule

(*) Modus Ponens

(*) And introduction

(*) Or "

1. $P \wedge Q \rightarrow R$
2. $Q \rightarrow P$
3. Q.

Proof it is raining.

∴ Q is True.

∴ (Q → P) is True.

P

Q → P.

∴ Q is True.

(1.) Q

(2) Q → P.

(3.) P (Modus Ponens).

(4.) P ∧ Q (AND introduction).

(5.) (P ∧ Q) → R

(6.) R (Modus Ponens).

Predicate Calculus.

→ "Everyone is younger than his/her father"

Person (x), represent father via a function.

∀x Younger (x, father (x))

∀x (Person (x) ⟶ Younger (x, fatherof (x))

→ "None earn more than the president"

Person (x) , President (y).

∀x,y (Person (x) ∧ President (y) ⟹ ¬ Earns more (x, y)).

Income of x → i(x).

Gt( ) → Greater.

∀ (x, y) (Person (x) ∧ president (y) → ¬ Gt (i(x), i(y)

→ You can fool someone all the time.

(Person (x) ∧ Time (t) ⟶ Can fool (x, t))

∃x ∀t (Person (x) ∧ Time (t) ⟶ Can fool (x, t))

"All the packages of room no. 27 are smaller than any package in room no. 20."

P(x) ∧ P(y) ⟹ Smaller (In (x,

Pack

~~Package(x) ∧ Package(y) ∧~~ In(

$\forall (x,y)\,(P(x) \wedge P(y) \wedge In(x,27) \wedge In(x,20) \rightarrow Smaller(x,y))$

"Noone likes everyone."

$\forall x \exists y \neg Likes(x,y)$.

$\exists x \exists x \neg Likes(x,y)$.

Inference Rule:

Modus Ponens.
Modus ——.
And Introduction
" Elimination.
Or Introduction.
Double negation
Chaining.

Another 4 rules:-

1. Universal Elimination
2. Existential "
3. " Introduction.
4. GMP. (Generalized Modus Ponens)

(∀)
Universal Elimination

↳ replace every instance occurrence of variable with its particular constant instance

$\forall x \; Eat(Rita, x)$

$Eat(Rita, Icecream)$.

Existential Elimination
(∃).

↳ we can replace variable with some instance.

Existential Introduction.

$P(G) \rightarrow \exists x \; P(x)$

Generalized Modus Ponens.

$P(G), Q(G)$.

$\forall x (P(x) \wedge Q(x) \rightarrow R(x))$.

↙
elimination $P(G) \wedge Q(G)$.

$\Rightarrow (P(G) \wedge Q(G) \rightarrow R(G))$

$\Rightarrow R(G)$.

(a) Every IT student is genius.
(b) Raju is a IT student.
(c) Raju is Genius. → Proof.

→ (a) $\forall x \; I(x) \rightarrow G(x)$.
(b) $I(Raju)$.
Proof: $G(Raju)$.

Using Universal Elimⁿ.

I(Raju) → Gl(Raju) ← (II)

I(Raju). ← (III)

Using Modus Ponens (II & III).

Gl(Raju).

Proof

Unification.

$\forall x_1, x_2 \ldots x_n \; (Y_1 \vee Y_2 \vee \ldots Y_m)$.

$P(x, y), x) \vee Q(A, x)$

$\neg P(B, A) \vee R(C)$

All the appearance of a variable should be replaced by same constant.

$P(B, A) \vee Q(A, A)$ → $x = A$, $y = B$.

Substitution.

$P(x, f(y), B) \xrightarrow[y \to b]{x \to a} P(a, f(b), B)$

Replace with variable.

$\delta = t_1/x_1, t_2/x_2 \ldots t_n/x_n$

δ is a substitution which consists of pairs t/x where t is a term and x is a

variable. where any variable x is replaced by term t.

$P(x, f(y), B)$ $S = a/x, b/y \rightarrow P(a, f(b), B)$

$= S = A/x, C/y \rightarrow P(A, f(C), B)$

$P(x, f(y), B)$ $S = \{A/x, C/y\}$

↓ Substitution

ω (clause)

$WS = P(A, f(C), B)$.

Combination of Substitutions.

$S_1 = g(x, y)/z$

$S_2 = A/x, B/y, C/w, D/z$

$S_1 S_2$: Apply S_2 to every appearance of term in S_1,

If a variable is not in S_1 then include it in S_1 from S_2.

$S_1 S_2 = \{g(A, B)/z, A/x, B/y, C/w\}$.

Consider $S_1 = f(y)/x$, $S_2 = A/y$

$S_1 S_2 = f(A)/x, A/y$.

(*). $P(x, y) S_1 S_2 = P(f(A), A)$.

$P(x, y) S_1 = P(f(y), y)$.

$P(f(y), y) S_2 = P(f(A), A)$. ∎

(*) Prev. example shows substitution is associative.

Commutative?

$$P(x,y)\,S_2,S_1 \neq P(x,y)\,S_1 S_2$$

Unification is a process where a set of sentences can be combined to form a single clause.

$\{w\}$ S

↓

set of sentences

If $w_1 S = w_2 S = w_3 S = \cdots w_n S$

then S is called unifier.

$w_1 = P(x, f(y), B)$

$w_2 = P(x, f(B), B)$ ← Can we unify.

$S = A/x, B/y$

$w_1 S = P(A, f(B), B) = w_2 S = P(A, f(B), B)$

$\Rightarrow w_1 S = w_2 S$

$\therefore S$ is a unifier.

Most General Unifier (MGU).

WS = $\forall g$ S' → Another unifier.
↳ MGU

Finding out MGU.

① Sentence → Least structured form.

P(x, y) → (P, x, y)

P(x, f(y)) → (P, x, (f, y))

② Disagreement set.

P B (f A D)

↕

P x (f A y)

Apply B for x ; then D for y.

P B (f A D)

∴ MGU = B/x, D/y.

P (f(x), x)

P (z, z)

Here using same strategy

P(f(x), x) will convert to P(f(x), f(x))

And this will never stop.

∴ No MGU.

Converting to normal form.

$\exists x \forall y \left(\forall z \; P(f(x), y, z) \rightarrow \exists x \, Q(x, y) \wedge \exists v \, R(y, v) \right)$

$A \rightarrow B$

$\neg A \vee B$.

$\exists x \forall y \exists z \; \neg P(f(x), y, z) \vee \exists x \, Q(x, y) \wedge \exists v \, R(y, v)$

Removal of existential quantifier.

Skolemization:

Skolem ~~constant function~~ constant.

$\exists z \, P(z) \rightarrow P(A)$ ↳ If $\exists$ is out of scope of $\forall$

Every person has some height.

$\forall x \exists y \, (H(x, y))$

If within scope of universal quantifier.

$\forall x \, H(x, h(x))$ ← Skolem function to remove existential quantifier.

$\forall z \forall x \exists y (H(x, y, z))$

↓

$\forall z \forall x \, H(x, h(x, z), z)$.

$\exists x \forall y$

$\forall y \; \neg P(f(A), y, h(y)) \wedge Q(A, g(y)) \wedge R(y, B)$

Resolution

1. All the packages of room no. 27 are smaller than any of the package of room no. 28. ~~Consider packs~~
2. Consider package A and package B are two packages.
3. Package A may be in room no. 27 or ~~path~~ room no. 28.
4. Package B is in room no. ~~27~~ 27
5. ~~Packg~~ Package B is not smaller than package A.

(a) Proove Package A is in room 27.

(b) $S(x,y)$: x is smaller than y

(c) $P(x)$: x is a package

(1) $\forall x \forall y \; P(x) \wedge P(y) \wedge I(x,27) \wedge I(y,28) \rightarrow S(x,y)$

In

$\forall x \forall y \; (\neg P(x) \vee \neg P(y) \vee \neg I(x,27) \vee \neg I(y,28) \vee S(x,y))$

(2) (i) $P(A)$ (ii) $P(B)$

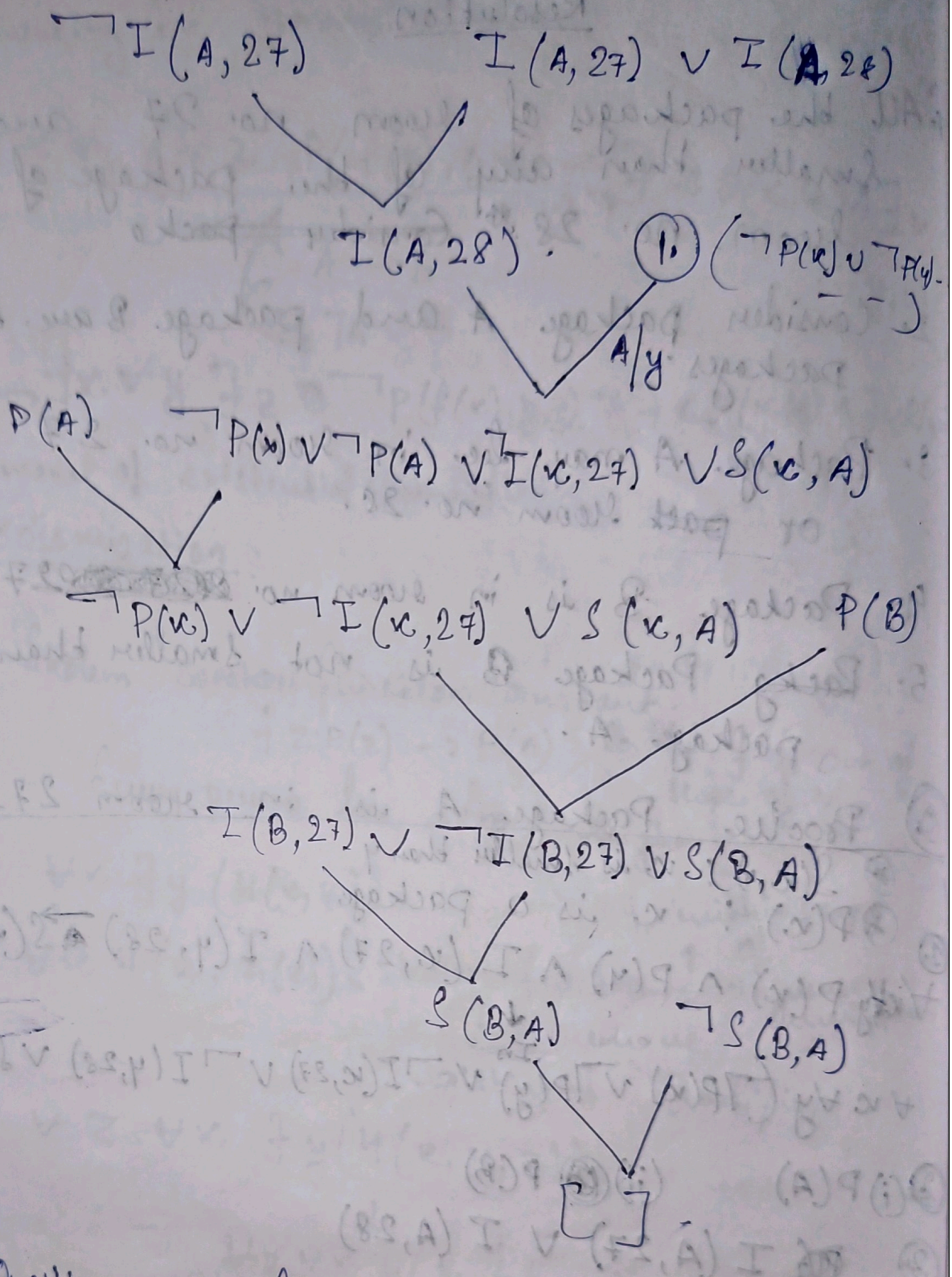
(3) $I(A,27) \vee I(A,28)$

(4) $I(B,27)$

(5) $\neg S(B,A)$

(6) $\neg I(A,27)$.

↳ Take negation of whatever we need to proove in CNF form.



Question may be asked.

like ? in which room is A.

We can write negation of question and answer.

like $\neg I(A,u) \vee Ans(u)$

↓

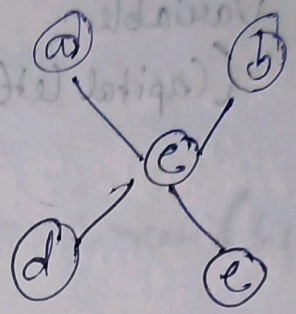
Ques in negation form.

(*) For theorem proving generate a null set.
(*) For answer finding get a clause like Ans(x7)

PROLOG

Programming in Logic

Parent (a, c)
Parent (b, c)
:
:
} Clauses. (Fact)



Question Clause.

?- Parent (a, c) — Yes.

?- Parent (a, x)

Variable in capital
Constant in small

Fact — everything that's true.

Offspring (X, Y) :- Parent (Y, X)

X — Conclusion
:- — Inverse
Y — Condition.
} (Rule.)

Clause
- Fact
- Rule
- Question.

* Relationships among object
* recursively

Data Objects

- Simple Object
  - Constant (Small letters)
  - Variable (Capital letters)
- Structure
  Collection of simple objects. Or structure which are combined through a funda.

  E.g. date(1, May, 2023).

  date → funda

  1 May 2023

$P_1 = P(1, 2)$

$P_2 = P(3, 4)$

line(Point$_1$, Point$_2$).

List

L = [H|T]

head → tail

L = [a, b, c, d]

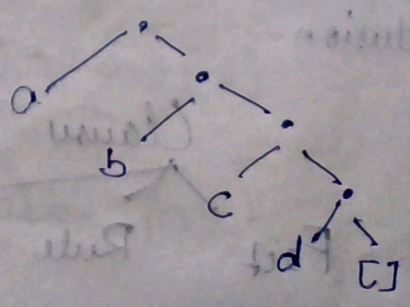
(H|T)

a → head

[b, c, d] → tail

↓

funda



* Member relation ship :-

member(X, [X|T]).
member(X, [Y|T]) :- member(X, T).

* Concatenation of two list.

concat([], L, L).
concat([X|L1], L2, [X|L3]) :- concat(L1, L2, L3)

* Insert operation at beginning

insert(X, T, [X|T]).

* Delete Operation -

delete(X, [X|T], T)
delete(X, [Y|T], [Y|T']) :- delete(X, T, T')

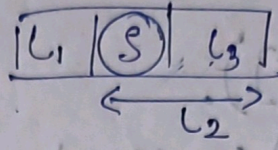
* Insert at any position.

insert(X, L1, L2) :- delete(X, L2, L1)

* Sublist :-

(*) Sublist.

sublist(S, L)
concat(L1, L2, L), concat(S, L3, L2).



(*) Check given list is odd length or even.

evenlength([])
evenlength([X|T]) :- oddlength(T)
oddlength([X|T]) :- evenlength(T)

(*) ~~Factor~~ length of a given list.

length([], 0)
length([H|T], L) :- length(T, L1), L is L1+1

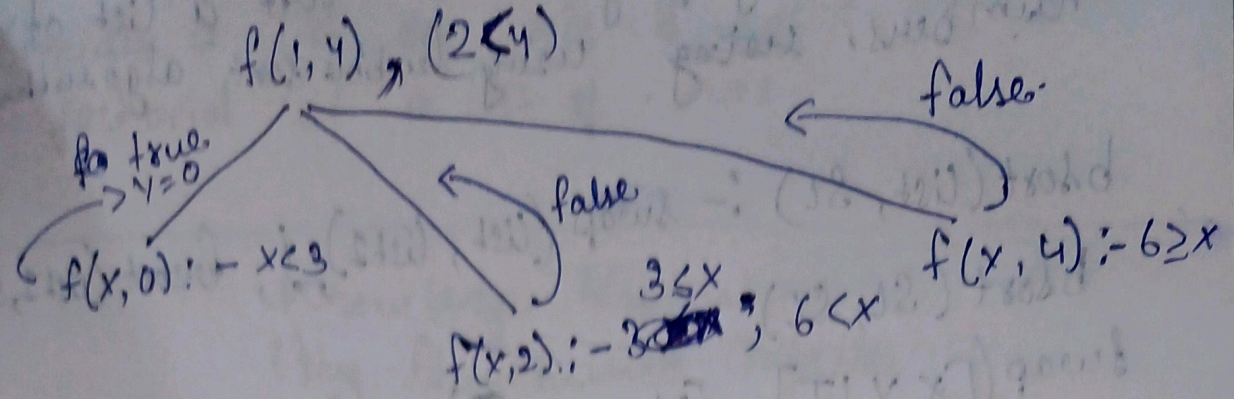
(*) factorial of a number.

fact(0,1).
fact(N, F) :- N>0, N1 is N-1, fact(N1, F1), F is N*F1.

(2)

f(x,0) :- x<3
f(x,2) :- 3≤x ; x<6.
f(x,y) :- 6≥x

?- f(1,Y), 2<Y → false
(Y=0, false.)



Prolog backtracks and checks all the node.

∴ Sometimes backtracking may be uncontrolled

Controlled backtracking

Cut (!)

f(x,0) :- x<3, !

f(x,2) :- 3≤x; x<6, !

f(x,4) :- 6≥x

Ravi likes all the animal except snakes.

① Ravi likes all the animal -

likes(ravi, x) :- animal(x)

likes(ravi,x) :- snake(x), !, fail. → to make true statement false after cut

likes(ravi, x) :- animal(x).

Prolog by Ivan Bradpro?

Write a prolog program to sort a list of numbers. ~~string~~ using bubble sort algorithm.

```
bsort(list, SL) :- swap(list, list1), !, bsort(list1, SL)
bsort(SL, SL).
swap([X, Y|T], [Y, X|T]) :- Y<X.
swap([Z|R], [Z, R']) :- swap(R, R')
```

Insertion Sort (Exam — insertion sort & merge sort)

```
insort([], [])
insort([X|T], SL) :- insort(T, T'),
                     insert(X, T', SL)
```

~~insert(X, [Y|T], [Y|sorted1]) :- X>Y, insert(X, T, sorted1)~~

If x is ~~less than~~ greater than Y, insert x at tail.

```
insert(X, [Y|sorted], [Y|sorted1]) :- X>Y,
                                      insert(X, sorted, sorted1), !.
insert(X, sorted, [X|sorted])
```

↳ if x is smaller than head then insert x at head.

↓ Cut

Consider a list on nos. & find out factorial of a list of nos.

Knowledge Representation Using Structured Objects.

Previous – Logic based Storage of Knowledge.
↳ formal logic

Structured Objects –
Knowledge based formalisms.

① Semantic Networks.

Relationship
is a
instance
part of
has
functional

Explain → Draw.

② is-a hierarchy

Non binary relations.
↳ turning relationships into objects.
↳ we may have to add some new objects and create instances.

Any class information is inherited by its members & subclass.

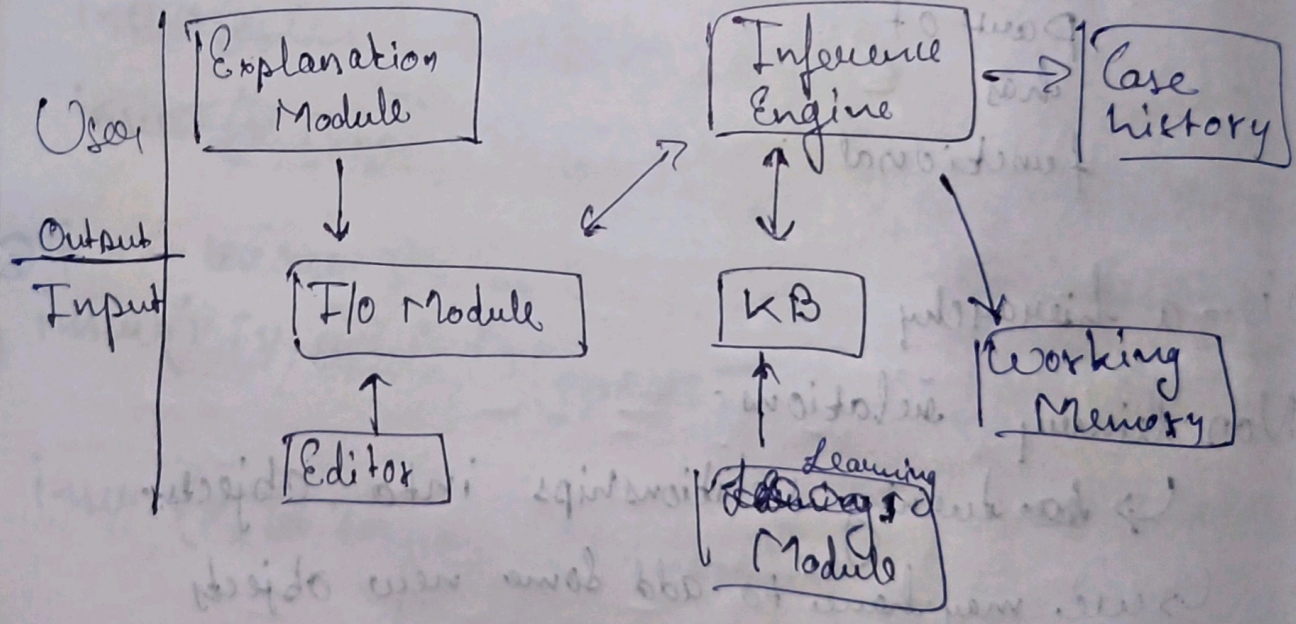
(2) Frame ~~by~~ by Minsky

Expert System

(1) Expert systems work with knowledge rather than data.

(2) knowledge ~~ba~~ base is different than Control program of the system.

(3) Expert system when returns answer, has some explanation behind the answer.



KB :- Knowledge Base.

Match — Matching info from wm with LB.
Select
Execute

— Peterson — Reference

Game theory A*, searching & — 3 ques.
Pred. Calc, Expert sys — 2 ques
~~A*, Lean~~
Prolog — 1 ques.

Class Test — A* to Pred. Calculus.