Books: (1) Russel & Norvig.
(2) Rich & Knights.

Theory → modeling → algorithm → application ⇒ 4 things to implement an applicⁿ.

(1) Make a real life problem into a computational problem.
(Computational problem, e.g., graph). ⇒ modeling.

(2) & choose algorithm.

(3) Implement the algo.

» History of AI :-

1940s : first general purpose computer developed.

1950s : "Can machines think?" — Alan Turing, 1950.

How one can understand that something/somebody can think?
How you " " your friend thinks ?, if he/she would not be able to speak, write?

— Create a situation that they have to make a decision (by some indication like movement of hand, blinking eyes, moving legs etc.).

- If there is a person who only has brain, but does not have any motor to indicate or response, → no way to know whether he/she can think.

— So, communication with the world by ~~listening or~~ producing an output is critically important for us to judge whether there is any intelligence.

» 'Turing Test' is to figure out whether a machine thinks or not. → (a thought experiment).

→ During the Test, the human interrogator asks several questions to both players, Based on the answers, the interrogator attempts to determine which player is a computer and which player is a human respondent.

1956: A new field 'AI' was born in a ~~work~~ 2-months workshop of 10 man study on artificial intelligence, ~~in~~ at Dartmouth College in Hanover, New Hampshire by John McCarthy.

1964: Eliza - the chatbot ~~as~~ psychotherapist

1966: Shakey - general purpose mobile robot - (A* algo introduced that time).

» Milestone in the history of AI

1997: 'Deep Blue' (a chess playing expert system run on a unique purpose-built IBM supercomputer) defeated world champion chess player Garry Kasparov (human) by winning 2 games and drawing three in a six-game match.

1st generation of AI: Logic based AI / Symbolic AI (in combinatorial algorithms)

2nd generation of AI: Probabilistic AI

Current AI: Neural models.

AI: What is the nature of intelligent thought?

Intelligence: capacity ~~to~~ for learning, reasoning, understanding, and similar forms of mental activity.

Definitions:

① "[automation of] activities that we associate with human thinking, activities such as decision making, problem solving, learning..." (Bellman 1978)
↳ (systems that think like humans) (thought)

② "The study of mental faculties through the use of computational models" (Charniak & McDermott 1985)
↳ (systems that think rationally). (~~behavioral~~) (thought)

*③ "The study of how to make computers do things at which, at the moment, people are better" (Rich & Knight 1991). ↳ (systems that act like humans) (~~thought~~) (behavioral)

④ "The branch of computer science that is concerned with the automation of intelligent behavior" (Luger & Stubblefield 1993). ↳ (systems that act rationally) (behavioral)

» What major components were important for Turing Test?

- Natural language processing
- Knowledge representation
- Automated reasoning
- Machine learning.

additional for total Turing Test
- Computer vision
- Robotics

AI focuses on -

- Learning Process: gathers data, creates rules for transforming it into useful informa".
- Reasoning Process: Selects best algo
- Self-correction Process: fine tune algo. to offer most reliable result.

» Examples of AI

- Google map
- Face Detection & recognition.
- Chatbot
- Text Editor & Autocorrect.

Problem, Problem Spaces & Search

To solve a problem of building a system, we should take the following steps:

1. Define the problem accurately including detailed specifications and what constitutes a suitable solution.

2. Scrutinize the problem carefully, for some features may have a central affect on the chosen method of solution.

3. Segregate & represent the background knowledge needed in the solution of the problem.

4. Choose the best solving techniques for the problem to solve a solution.

» Problem solving is a process of generating solutions from observed data.

» A 'problem space' is an abstract space:

- It encompasses all valid states that can be generated by the application of any combina" of operators on any combination of objects.
- The problem space may contain one or more solutions. A solution is a combina" of operations & objects that achieve the goals.

» A 'search' refers to the search for a solution in a problem space.

- Search proceeds with different types of 'search control strategies'.
- DFS & BFS are 2 common search strategies.

» General Problem Solver (GPS) was a computer program created in 1957 by Simon & Newell to build a universal problem solver machine.

It can solve
- theorem proving.
- geometric problems
- chess playing.
- Tower of Hanoi.

It can not solve real-world problems.

» To build a system to solve a particular problem, we need to —

- Define the problem precisely — find input situations & final situations for an acceptable solution to the problem.
- Analyze the problem.
- Isolate & represent task knowledge necessary to solve the problem.
- Choose the best problem solving technique(s) & to apply to the particular problem.

» Problem Definitions:

A problem is defined by its 'elements' and their 'relations'. To provide a formal description of a problem, we need to

- Define a state space that contains all possible configurations of relevant objects, including some impossible ones.
- Specify one or more states that describe possible situations, from which the problem-solving process may start: 'initial state'.
- Specify one or more states that would be acceptable solution to the problem: 'goal state'.
- Specify set of rules that describe the actions (operators) available.

Search:

The problem can be solved by using the rules, in combination with an appropriate control strategy, to move through the problem space until a path from an initial state to a goal state is found. This process is known as 'Search'.

Thus:

- Search is fundamental to the problem-solving process
- Search is used when a more direct method is not known.
- Search provides framework into which more direct methods for solving subparts of a problem can be embedded.

- Problem space (represented by a directed graph; where nodes represent search state & paths represent the operators applied to change the state).

To simplify search algorithms, it is convenient to logically & programmatically represent a problem space as a tree.

⇒ A tree usually decreases the ~~time~~ complexity of a search at a cost.

⇒ "Any connected graph with no cycle is a tree".

Also we can say,

"The process of looking for a sequence of actions that reaches the goal is called search.

① Problem formulation :-

You will be given some an problem example problem. You formulate the problem by the following.

states, Initial state, Transition Model, Goal test, Path cost.

Ex1: 8-puzzle problem formulation

p-71 of Russel :- [90/1157]., 71., 80, 81).

Is there an AI algorithm that can solve all the problems? — from there the notion of search starts.

- ~~Note~~ Don't think about algorithm. Try to find out what is the input & o/p. If there is no clarity in that, the other person won't be able to ~~find~~ & answer the question.

← Can there be a uniform way to convert any computational problem to one input representation?

What is that representation?

Here comes the concept of state.

⇓

All information necessary to make a decision for the task at hand.

» Uninformed Search :-

We do not have idea about which is a better state, which is the worst, it is called uninformed search (Also called blind search).

All they can do is generate successors and distinguish ~~in~~ a goal state from ~~no~~ a non-goal state. All search strategies are distiguished by the order in which nodes are expanded.

» Strategies that know whether one non-goal is "more promising" than another are called informed search or heuristic search.

» BFS, DFS → Uninformed search.

Breadth first search.

» Best First Search Algorithm :-

(Informed or Heuristic search technique).

Algo:

1. Let OPEN be the priority queue containing initial state.

LOOP

If OPEN is empty
return 'failure'

NODE ← Remove-first (OPEN) [// Remove first node from OPEN //].

If NODE is a goal
return the path from initial to the NODE.

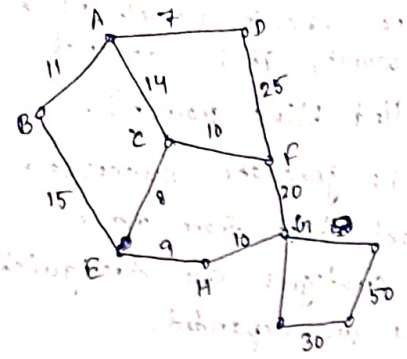
else
generate all successors of NODE and put the newly generated NODE into OPEN according to their f-values.

END LOOP.

start state = A.

Goal " = G.

We'll find out (or it will be given) straight line distance or Euclidean distance.



A → G = 40		E → G = 19	straight line distance.
B → G = 32		F → G = 17	
C → G = 25		H → G = 10	
D → G = 35		G → G = 0	

[Euclidian distance: $\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$.]

Straight line distance will be ~~calculated~~ considered as Heuristic value. Based on that, we'll proceed in Best first search.

first 'A' is in the priority queue named OPEN.

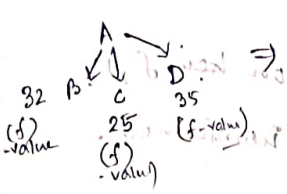
A.

Run the algo.

A is not the goal.

'f' value is the heuristic value.

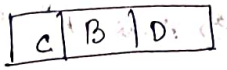
(*) The lesser the value of 'f', the higher is its priority.



⇒ If you go from A to B, then B will take A to G with which f value?

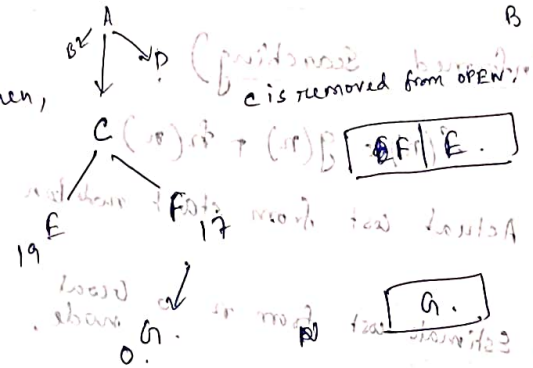
A is removed from OPEN.

In OPEN, now



[as cost of c is minimum among B, C & D, so, put c first in OPEN. Then B & then D].

Then,



c is removed from OPEN.

[A is in closed queue as A is already explored].

NOW G is NODE.

So, path is A → C → F → G. (Ans)

Let we would try breadth first search (uninformed) then, ~~it be~~ A would first explore B, C & D.

But, in heuristic (informed), ~~it~~ or A explores only C. So, ~~it the cost~~ it is the greedy method based on the heuristic value. Less time needed.

Best first search always gives a good soln. Answer, It may not give optimal solution. Much better than the uninformed method.

avg. case → polynomial time.
worst " → exponential. * (based on f).

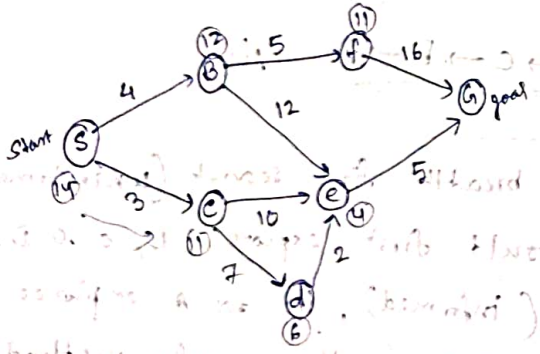
In best or avg case, best first is better than Uninformed.

If you would take care of the edge-cost as well as heuristic cost, there is more chance to get optimal sol^n. → A* algorithm takes care of that.

A* Algorithm (Informed Searching)

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

- $g(n)$: Actual cost from start node to n.
- $h(n)$: Estimate cost from n to Goal node.



S → G	⇒ 14.
B → G	⇒ 12
C → G	⇒ 11
d → G	⇒ 6.
e → G	⇒ 5
f → G	⇒ 11.

A* gives you optimal sol^n.
↓
Admissible search.

$f(S) = 0 + 14 = 14$.

$S \to B \Rightarrow 4 + 12 = 16$

$S \to C \Rightarrow 3 + 11 = 14$.

Here $14 < 16$.
So, we'll go $S \to C$.

$SC \to e \to 3 + 10 + 5 = 18$ [S→C ⇒ 3, C→e ⇒ 10, h(e) = 4]

$SC \to d \to 3 + 7 + 6 = 16$.

$16 < 18$.

So, path is

~~S C d →~~

Now, $S \to B \Rightarrow 16$ & $SC \to d$ is 16.

So, we'll explore both.

$SB \to f \Rightarrow 4 + 5 + 11 = 20$

$SB \to e \Rightarrow 4 + 12 + 4 = 20$.

$SCd \to e \Rightarrow \underbrace{3 + 7 + 2}_{g(n)} + \underbrace{4}_{h(n)} = 16$.

So, we'll consider,

$SCde \to G$

$\Rightarrow 3 + 7 + 2 + 5 + 0 = 17$.
(0 = h(G))

So, path is

$S \to C \to d \to e \to G$.

Time complexity of A*:

$$O(V + E) = O(b^d)$$

b: branch factor
d: depth.

Space complexity $\Rightarrow O(b^d)$.

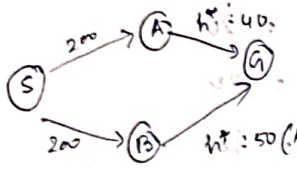
» How to make A* admissible:

$h(n) \leq h^*(n)$ - underestimation.

$h(n) \geq h^*(n)$ - overestimation.

$h(n)$: Heuristic value.

$h^*(n)$: Actual (original) value / optimal value.



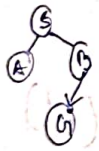
Case I: Overestimation

Let, $h(A) = 80$, $h(B) = 70$ $\Big\} > h^*$.

$f(A) = g(A) + h(A) = 200 + 80 = 280$.

$f(B) = 200 + 70 = 270$.

$f(B) < f(A)$.



$f(G) = g(G) + h(G) = 250 + 0 = 250$

So, path is, $S \to B \to G$.

Case II: Underestimation

Let, $h(A) = 30$ $(30 < 40)$

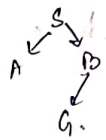
$h(B) = 20$ $(20 < 50)$.

$\therefore f(A) = g(A) + h(A) = 200 + 30 = 230$.

$f(B) = 200 + 20 = 220$.

$f(B) < f(A)$.

$\therefore f(G) = (200 + 50) + 0 = 250$.



Now, it won't stop here. because,

$f(A) < f(G)$.

So, again it will be calculated through A.

$\therefore f(G)$ (through A) is :

$(200 + 40) + 0 = 240$. $\therefore$ path is, $S \to A \to G$.

So, if we underestimate it, there is more chance to give optimal solution.

* If the heuristic function, h always underestimates the true cost (i.e, $h(n) \leq h^*(n)$), then A* is guaranteed to find an optimal solution.

* $h(n)$ is called admissible if $h(n) \leq h^*(n)$.

» Hill Climbing Algorithm. (Local Search Algo, greedy Approach, No backtrack)

1. Evaluate the initial state.
2. Loop until a soln is found or there are no operators left.
   - Select and apply a new operator.
   - Evaluate the new state.
     - if goal is reached, then quit.
     - if better than current state then it is the new current state.

It is simply a loop that continually moves in the direction of increasing value — i.e., uphill. It terminates when it reaches a "peak" where no neighbor has a higher value.

The algorithm does not maintain a search tree, so, the data structure for the current node need only record the state and the value of the objective function. Hill climbing does not look ahead beyond the immediate neighbors of the current state.

Algo: Hill Climbing

function Hill_climbing(problem) return a state that is local maximum

~~current ← Make_NODE(INITIAL_STATE(problem))~~

~~loop do~~

current ← MAKE-NODE(problem.INITIAL_STATE)

loop do

neighbor ← a highest-valued successor of current

if neighbor.VALUE ≤ current.VALUE then

return current.STATE

current ← neighbor.

8-Queens Problem & Hill climbing Algo

- Successor function:
  - move a single queen to another square in the same column. (8×7 = 56 successors)
- Example of a heuristic function h(n)
  - the number of pairs of queens that are attacking each other (directly or indirectly).
  - So, we want to minimize this.

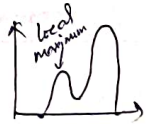
Page - 122, 123 of Russel.

Hill climbing is sometimes called greedy local search because it grabs a good neighbor state without thinking ahead about where to go next.

» Drawbacks of Hill Climbing :-

Hill climbing often gets stuck for the following reasons.

⇒ Local maxima :-



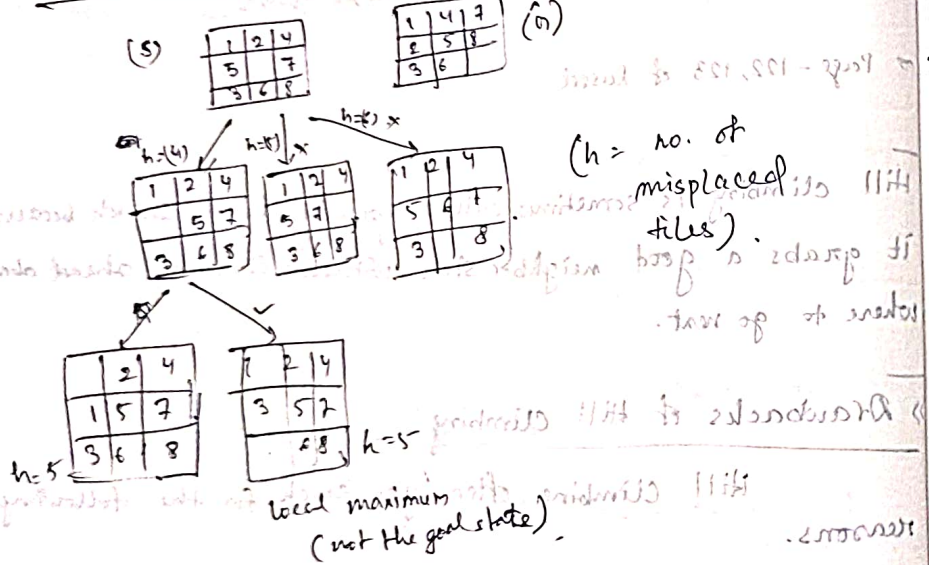
⇒ Plateau/flat maximum:

⇒ Ridge : ~~Ridges~~ result in a sequence of local maxima that is very difficult for greedy algorithms to navigate.

The success of hill climbing depends very much on the shape of the state-space land-scape: if there are few local maxima & plateaux, random restart hill climbing will find a good solution very quickly.

→ NP hard problems typically have an exponential number of local maxima to get stuck on.

8-puzzle with hill climbing



(h = no. of misplaced tiles).

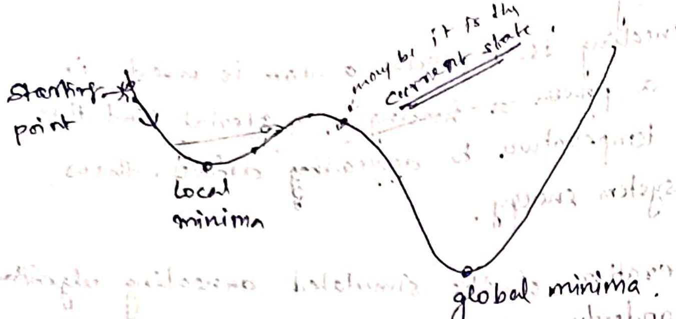
» Simulated Annealing :-

The term 'annealing' is a process of is used in metallurgy. It is a process of heating a material and then slowly lowering the temperature to decrease defects, thus minimizing the system energy.

— At each iteration of the simulated annealing algorithm a new point is randomly generated.

- The distance of the new point from the current point, or the extent of the search, is based on a probability distribution with a scale proportional to the temperature.
- The algorithm accepts all new points that lower the objective, but also, with a certain probability, points that raise the objective.
- By accepting points that raise the objective, the algorithm avoids being trapped in local minima, and is able to explore globally for more possible solutions.
- An annealing schedule is selected to systematically decrease the temperature as the algorithm proceeds.
- As the temperature decreases, the algorithm reduces the extent of its search to converge to a minimum.

- Simulated annealing(SA) is a global search optimization algorithm and it is modified version of hill climbing.
- SA starts (at high temperature)
- then gradually reduce the intensity (lower the temperature).
- SA is very useful where there are a lot of local minima.



SA escapes local optima by introducing controlled randomness in its search.

- Temperature t is the control variable.
- neighboring point will be chosen randomly instead of immediate neighbor.
- If the randomly selected point is less than the current state, then it is the next current state.
- Let bounce a ping pong ball that will touch at different points and ultimately reach the global minimum. (throw the ball so hard that it does not rest in the local minima)

- It picks random move instead of best move.
- If the move improves the result, then it accepts this random move, otherwise it accepts the move with some probability less than 1.

Based on annealing, the algorithm uses the equation

$e^{-\Delta D/T} > R(0,1)$,

- where ΔD is change of distance
- T is synthetic temperature (control variable) and
- R(0,1) is a random number in the interval [0,1].

$e^{-\Delta E/T} > R(0,1)$

ΔE : [(current_value) − (next_value)]

Algorithm

```
function Simulated-Annealing (problem, schedule) returns a solution state
    current ← problem. INITIAL
    for t = 1 to ∞ do          (we don't know the state space).
        T ← schedule (t)       [cooling rate can not be determined beforehand, should be emprically adjusted]
        if T = 0 then return current
        next ← a randomly selected successor of current
        ΔE ← VALUE (current) − VALUE (next)
        if ΔE > 0 then current ← next
        else current ← next only with probability e^{ΔE/T}
```

E denotes the objective function (also called energy).

~~If ΔE = E_next − E_current > 0, probability of accepting a state with a better objective function is always 1.~~

⇒ schedule (t) ⇒ we allow initially make large changes to T, but gradually reduce it.

⇒ how schedule (t) lowers T, may take experimenting.

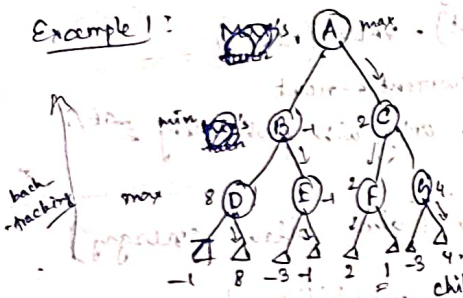
⇒ Once T is sufficiently low, or "cool enough", we finishes.

» Mini-Max Algorithm

Minimax is a decision-making strategy used in, game theory, particularly in 2-player games, where a player aims to minimize their potential maximum loss (Best move).

- Backtracking algorithm
- Best move strategy move
- Max will try to maximize its utility (Best move)
- Min will try to minimize utility (Worst move) at Max's side

Example 1:



Say, first is Max's turn, Max will choose the maximum value as between B & C.

Then, say Min's turn. Min will choose the min value between the two child nodes of the Max's chosen node..

Example 2:

⇒ Consider the following 2 player game tree in which the static scores are given from the first player's point of view.

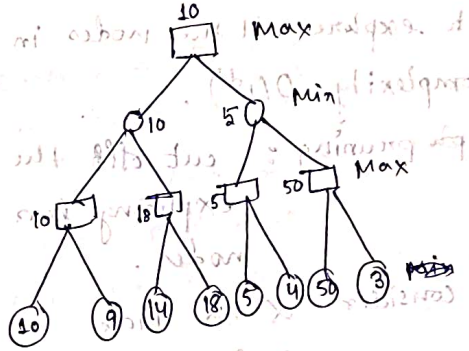
⇒ Apply the Mini Max search algorithm & compute the value of the root of the tree.

⇒ Also find the most convenient path for MAX Node.

Soln:
1. Generate the whole game trees to leaves
2. Apply the utility (payoff) function to leaves,
3. Use DFS for expanding the tree.

4. Back up values from leaves towards the root:
   - a Max node computes the maximum value from its child values.
   - a Min node " the minimum " " child values.

5. When value reaches the root: optimal move is determined.



Time taken by Mini-max is huge ($O(b^d)$). So it is not possible to use Mini-max for complex games like chess. So, we only compute upto certain depth & use the evaluation func^n to calculate the value of the board.

To overcome the drawback of Mini-max, we introduce alpha-beta pruning.

Alpha-Beta Pruning :-

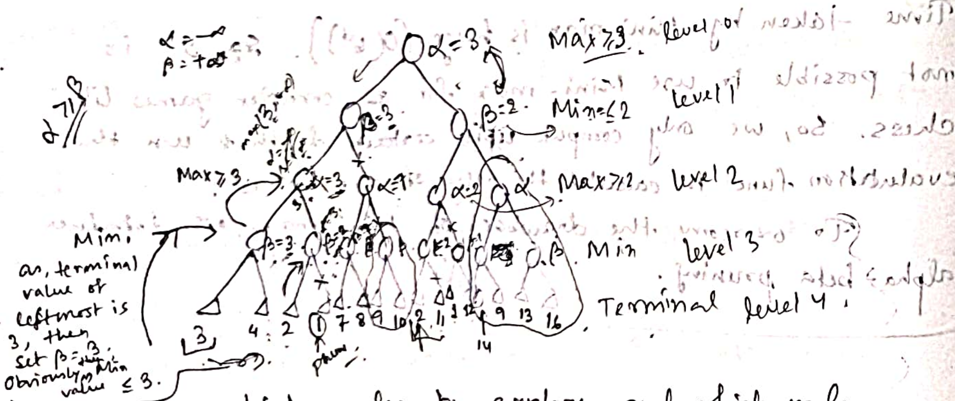
Advanced version of Minimax.

In Minimax → Time complexity = $O(b^d)$ [as we explore all nodes] ('b' denotes every node has 'b' no. of child, i.e. branch factor) (Game tree's depth = d)

⇒ We need to find out the best path for the MAX player (considered in the root node). For that, we use to explore all the nodes in minimax. So, huge time complexity $O(b^d)$.

In α-β pruning : cut off the search by exploring less no. of nodes.

⇒ In general, we consider α for Max, β for Min.



⇒ We'll see, which nodes to explore and which nodes may be ignored.

⇒ leftmost terminal = 3. Set β = 3. So, Min ≤ 3

⇒ Then go to next terminal node i.e., 4. As 4 > 3, so, no change in β.

⇒ Now, at the prev. level, set α = 3. So, Max value ≥ 3. (at level 2).

⇒ As, at level 4, we have 2, set β = 2 (at level 3). but at prev. level (level 2), α = 3.

⇒ level 1 : Min ≤ 3.

at level 1 : either value 3 or less. So, we do not consider the branches of α = 7 and so from level 2 onwards.

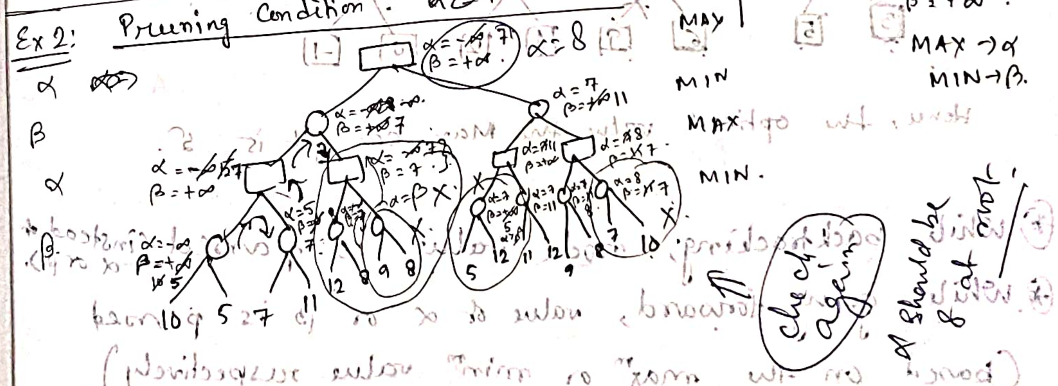
at level 2, either value 3 or greater. So, we do not consider β = 2 and so from level 3 onwards.

As soon as we get α = 3 in root, we now know that we atleast get one path through which if we go, we'll get α = 3. Now, see the right branches to check if we can get greater than 3 value at the root. Now, again start from the leftmost of rightside branches.

Best Case : $O(b^{d/2})$

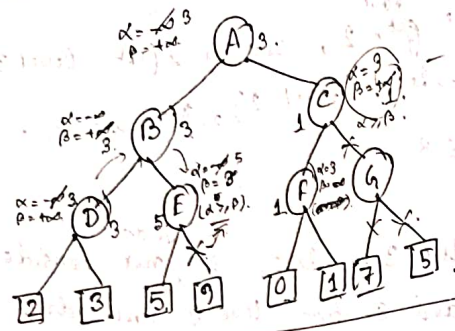
Pruning Condition : α ≥ β | initially α = -∞, β = +∞. MAX → α, MIN → β.

Ex 2:



>> ~~Constraint Satisfaction Problem~~ :⇒

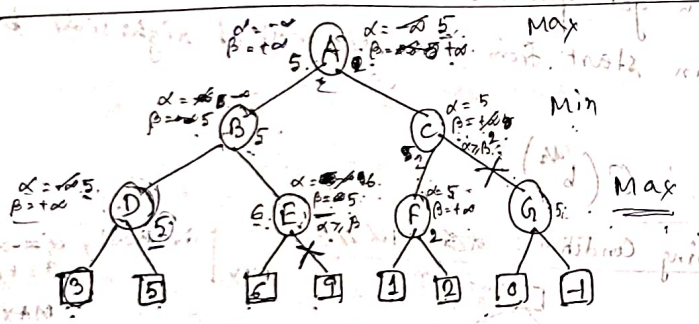
Ex 3:



Worst Case: When best move occurs at the right side of the tree and the αβ algo does not prune any leaves. ($O(b^d)$).

Best case: ⇒ when so many pruning happens and best move occurs at the left side of the tree.

Ex 4:



Hence, the optimal value the Max can get is 5.

* While backtracking, node value is passed (instead of α or β)
* While going forward, value of α or β is passed (based on the max^m or min^m value respectively).

Constraint Satisfaction Problem:

3 components: V, D, C.

⇒ V is set of variables $\{v_1, v_2, \ldots, v_n\}$

⇒ D is " " domains $\{D_1, D_2, \ldots, D_n\}$ one for each variable.

⇒ C " " " Constraints that specify allowable combination of values.

$C_i = (\text{scope}, \text{rel})$

Scope: set of variables that participate in constraint.

rel: relation that defines the values that variable can take.

Let, V_1 variable takes value from domain A.

V_2 " " " B.

Constraint C_1 says, $V_1 \neq V_2$.

$C_1 = ((V_1, V_2), V_1 \neq V_2)$ → for this example!

or, we can write

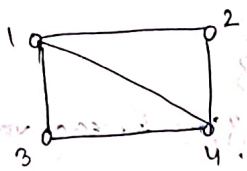
$C_1 = (V_1, V_2), (A, B)$.

Let A: {1, 2}

B: {2, 4}

then, $C_1 = \{(V_1, V_2), (1,2)(1,4), (2,4)\}$.

Example :- (intelligent Backtracking method is used)



Constraint graph.

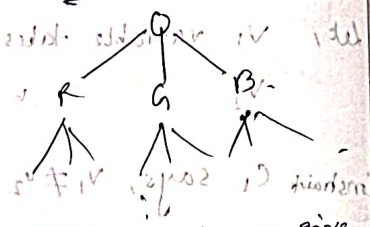
We shall color the graph. (graph coloring).

$V = \{1, 2, 3, 4\}$.

$D = \{Red, Green, Blue\}$.

$C = \{1 \neq 2, 1 \neq 3, 1 \neq 4, 2 \neq 4, 3 \neq 4\}$

search space



	1	2	3	4
Initial domain	R, G, B.	R, G, B	R, G, B	R, G, B.
1 = R	R.	GB	GB	GB.
2 = G	R	G	GB.	B.
3 = B.	R	G.	B	(B) empty.
3 = G.	R	G	G	B.

legal colors.

violating constraint (when we give 3 = B).

So, empty.

So, there is problem. We shall now backtrack to the place where conflict occurred.

↳ It this now satisfies the problem.

Let 3 = G.

Let, we have a graph,

Now, 2 colors & 6 nodes.

Search space is $2^6 = 64$.

Now, if we take heuristic value, then let heuristic function will be,

first color that node which has highest degree.

Let, middle node is fixed to R.

for the rest of the nodes, we have only one possibility i.e. green.

So, now that search space is 5.

That is,

If we take heuristic & try to solve it intelligently, then the search space is reduced a lot.

It is time efficient.

→ CSP : set of objects whose state must satisfy a number of constraints or limitations.

→ To solve a CSP, we need to define a state space and the notion of a solution. Each state in a CSP is defined by an assignment of values to some or all of the variables. $\{X_i = v_i, X_j = v_j, \ldots\}$. An assignment that does not violate any constraints is called a consistent or legal assignment.

→ A complete assignment is one in which every variable is assigned, and a solution to a CSP is a consistent, complete assignment.

→ A partial assignment is one that assigns values to only some of the variables.

Knowledge Representation

⇒ Intelligence requires knowledge.

⇒ Reasoning is processing of knowledge.

Knowledge Representation & Reasoning:

- Logic → Propositional logic / Predicate logic.
- Rules → if then.
- Semantic Net (Google graph).
- Frame (Slots (objects) and fillers (attributes))
- Scripts.

- How to store knowledge to a machine so that it can use the knowledge in future is called knowledge representation.

» Propositional Logic :⇒

- 'Atomic Sentences' can not be broken into other simpler sentences.
- There are two proposition symbols with fixed meanings: True & False.
- Complex sentences are constructed from simple sentences, using parenthesis and logical connectives.
- The five connectives (commonly used) are —
  1. ¬ (not) or negation
  - ∧ (conjunction)
  - ∨ (disjunction).
  - ⇒ implies.
  - ⇔ iff & only if.

- Syntax of propositional logic defines the allowable sentences.
- Semantics defines the rules for determining the truth of a sentence w.r.t a particular model.

False ⊨ True

Read as: False logically entails True. If all models that evaluate False to True also evaluate True to True.

A sentence α entails another sentence β if β is true in all worlds where α is true.

Inference is the process of deriving new sentences from old ones.

Modus Ponens / Modus Tollens $(P \wedge Q \vee R)$ =

Tautology

Contradiction

Connectives.

rules → Modus Ponens
Modus Tollens

forward Chaining

- forward deduction or forward reasoning method when using an inference engine.

- It is a form of reasoning which start with atomic sentences in the knowledge base and applies inference rules in the forward direction to extract more data until the goal is reached.

- Starts from known facts, triggers all rules whose premises are satisfied, and add their conclusion to the known facts.

- The process repeats until the problem is solved

Example:

"As per the law, it is a crime for an American to sell weapons to hostile nations. Country A, an enemy of America, has some missiles, and all the missiles were sold to it by Robert, who is an American citizen".

Prove that,

Robert is Criminal.

Ans: Conversion into FOL:-

1) American (P) ∧ weapon (Q) ∧ sells (P, Q, r) ∧ hostile (r) → Criminal (P) —①

2) ∃p Owns (A, p) ∧ Missile (p).
can be written as,

(T_1 is for Skolemn func".

Owns (A, T_1) —②

Missile (T_1) —③

3) ∀p Missiles (P) ∧ Owns (A, P) → Sells (Robert, P, A) —④

4) Missile (P) → Weapons (P) —⑤

5) Enemy (P, America) → Hostile (P) —⑥

6) Enemy (A, America) —⑦

7) American (Robert) —⑧

step 1: i) start with known facts.
ii) choose the sentences which do not have implications.

American (Robert)	Missile (T_1)	Owns (A, T_1)	Enemy (A, America)

step 2:

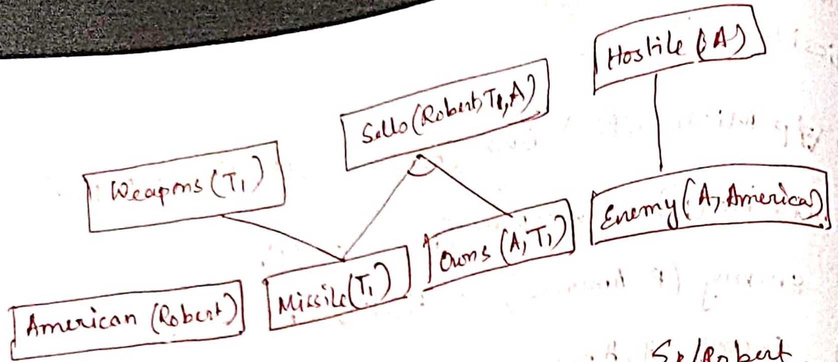
① does not satisfy premises; so, it will not be added in the first iteration.

② & ③ are already added in step 1.

④ satisfies with the substitution {p/T_1}, so, Sells (Robert, T_1, A) is added, which infers from the conjuns of ② & ③.

⑤ is infered from Missile (P) with the substitution Weapons (T_1).

⑥ is satisfied with substitution (p/A), so, Hostile (A) is added and which infers rule ⑦.

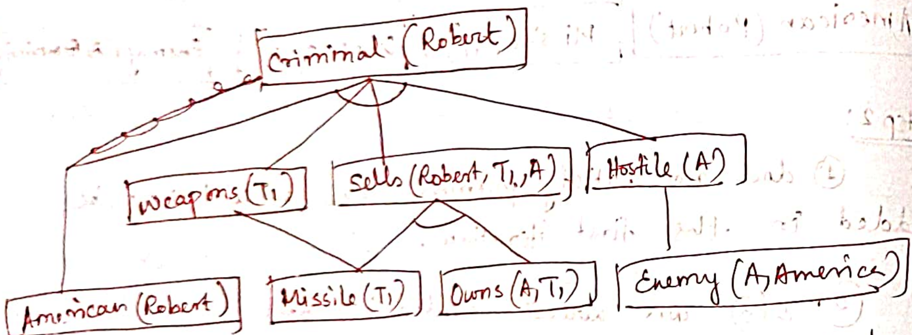


Step 3 :

① is satisfied with the substitution {p/Robert, q/T_1, r/A}.

So, we can add criminal (Robert) which infers all the available facts.

Hence we reached our goal statement.



Hence, it is proved that Robert is Criminal using fwd chaining approach.

» Backward chaining :-

– also known as backward deduction or backward reasoning when using an inference rule.

– starts with goal & works backward.

– chaining through rules to find known facts that support the goal.

– top-down approach.

Step 1 :

Criminal (Robert)

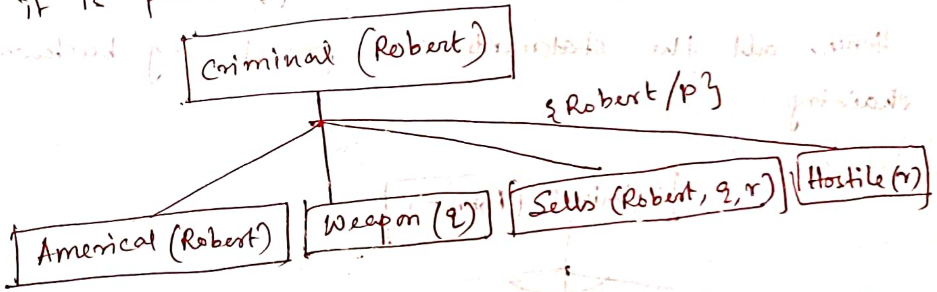
Step 2 :

We'll infer other facts from goal fact which satisfies the rules.

On ①, the goal predicate 'Criminal (Robert)' is present with substitute {Robert/p}.

So, we'll add all the conjunctive facts below the first level & will replace p with Robert.

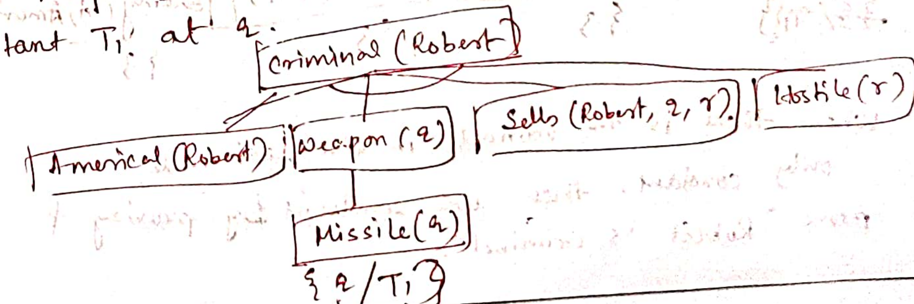
Here, we can see American (Robert) is a fact, so it is proved here.



Step 3 :

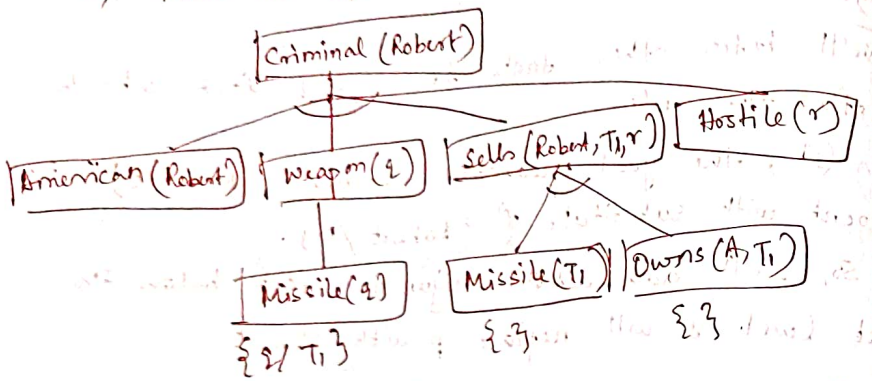
We'll extract further fact Missile (q) which infer from Weapon (q). (from ⑤).

Weapon (q) is also true with the substitution of a constant T_1 at q.



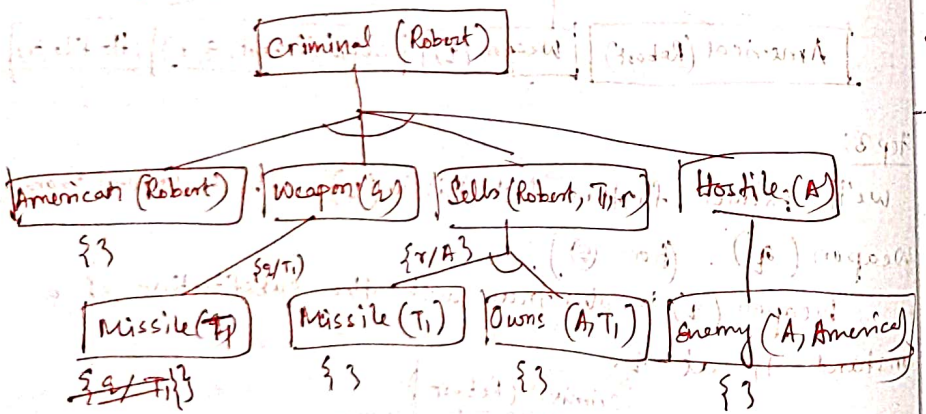
Step 4: we can infer facts Missile (T_1) & owns (A, T_1) from Sells (Robert, T_1, r) that satisfies (4), with the substitution of A in place of r.

So, these two statements are proved.



Step 5: we can infer the fact Enemy (A, America) from Hostile (A) which satisfies (6).

Here, all the statements are proved by backward chaining.



When there is no variable in the leaf node, only constant, then we conclude by proving & prove "Robert is Criminal".

Fwd Chaining

1) starts from known facts and applies inference rule to extract more data ~~unit~~ to reach goal.
2) bottom-up
3) Data-Driven
4) breadth-first search
5) tests for all possible rules
6) suitable for the planning, monitoring, control & interpretation
7) can generate infinite no. of possible conclusions
8. aimed for any conclusion

bckwd Chaining

1) starts from goal & works backward through inference rules to find the required facts that support goal
2) top-down
3) goal-driven
4) depth-first search
5) tests for few required rules.
6) suitable for diagnostic, prescription, debugging
7) generates finite no. of possible conclusion
8) aimed for required data.

Probabilistic reasoning

Uncertainty.

In ~~predc~~ FOL or propositional logics, we have learned knowledge representation with certainty. i.e., we know the predicates.

$A \rightarrow B$. if A is true then B is true.

But consider the situation where we are not sure about whether A is true or not. then, we cannot express this situation is uncertainty.

→ To represent uncertain knowledge, we need probabilistic reasoning.

» " " is a way of knowledge representation where we apply the concept of probability to indicate the uncertainty in knowledge.

probability theory + logic = Probabilistic Reasoning.

2 ways to solve problems with uncertain knowledge:

- Baye's rule
- Baysian statistics.

Conditional probability :-

probability of occurring an event when another event has already happened.

calculate event A when event B has already occurred.

i.e.,

⇒ The probability of A under the conditions B of B:

$$P(A|B) = \frac{P(A \wedge B)}{P(B)}$$

» Baye's theorem :-

If cancer corresponds to one's age, then by using Baye's theorem, we can determine the probability of cancer more accurately with the help of age.

$$P(A \cap B) = P(A|B)\,P(B).$$

or

$$P(A \cap B) = P(A|B).P(A).$$

Equating both the equations,

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

likelihood; in which we consider that hypothesis is true.

⇓

posterior, which we need to calculate.

(It is called probability of hypothesis A when we have occured an evidence B).

$P(A)$: prior probability / probability of hypothesis before considering the evidence.

$P(B)$: marginal / pure probability of an evidence.

We can write,

$$P(A_i|B) = \frac{P(A_i) * P(B|A_i)}{\sum_{i=1}^{k} P(A_i) * P(B|A_i)}$$

where, $A_1, A_2, \ldots, A_n$ is a set of mutually exclusive and exhaustive events.

Example: Q) What is the probability that a patient has disease meningitis with a stiff neck?

Given Data:

⇒ A doctor is a print out.

>> Bayesian Belief Network

Constraint Satisfaction Problem

We know that AI can be thought of as problem solving technique.

AI solves searching problem.

- Now we will jump from 'AI is search' to 'AI is representation'.

⇓

Specially when we have many problems to solve.

- All possible reasoning or inferences are possible only if we are allowed to look inside the state.
- Different problems have different problems or different representaⁿ. e.g. N-Queen problem with Q1 in col 1, Q2 is row2, col3, ... is a representaⁿ. However, N-puzzle can have diff. representaⁿ.

So, we cannot ~~do a gener~~ come up with a generalized solⁿ.

→ The goal of AI is to allow general purpose representation.

AI is thinking about in what language should the problem be given to the machine and in then using those properties of the language to reduce the amount of effort I need to do as a machine to solve the problem.

Standard Search Problem:-

'state' is a "blackbox" – any old data structure that supports goal test, eval, successor.

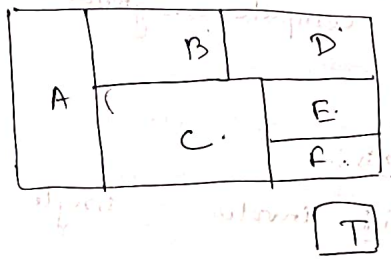
CSP: :-

'state' is not a black box. It is defined by variables X_i with values from domain D_i.

'goal ~~stat~~ test' is a set of constraints specifying allowable combinaⁿ of values for subsets of variables.

- CSP allows general-purpose algorithms with more power than standard search ~~proble~~ algorithms.

Map-Coloring Problem:-



7 variables:
A, B, C, D, E, f, T

3-colorings ~~problems~~ are possible.
{Red, Green, Blue}.

So, domain is: {Red, Green, Blue}.

Constraints: Adjacent regions must have different color.

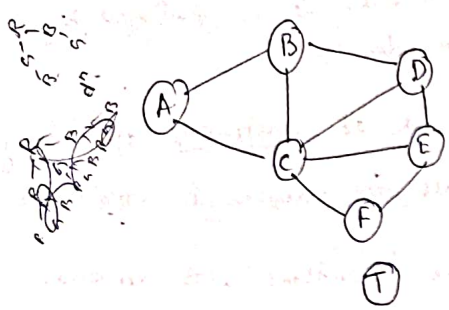
i.e., $A \neq B$.

$(A, B) \in \{(red, green), (red, blue), (green, blue), (blue, green)\}$

Now, ~~d~~ first represent it as Constraint Graph.

Binary CSP: each constraint relates at most 2 variables.

Constraint graph: nodes are variables, arcs show constraints.



There are two independent subproblem:
one is T
& another is rest of the graph.

Now, we consider each state variable is a node.

- If we have n variables & d domains, then there can have d^n complete assignments in discrete variables.

- Varieties of constraints:-

⇒ Unary constraints involve a single variable,
e.g.; A ≠ green

⇒ Binary constraints: involve pairs of variables.

A ≠ B.

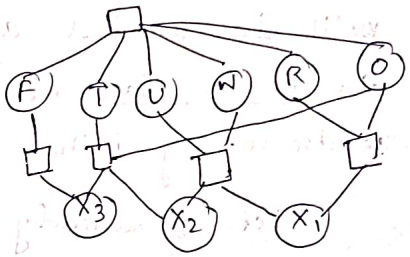
⇒ Higher-order constraints involve 3 or more variables, e.g. cryptarithmetic column constraints.

⇒ Preferences (soft constraints):-
e.g: 'red' is better than 'green'. often representable by cost for each variable assignment.
→ constrained optimiza^n problem.

□ : constraint

Example: Cryptarithmetic

$$\begin{array}{r} T\ W\ O \\ T\ W\ O \\ \hline F\ O\ U\ R \end{array}$$



Variables: F T U W R O X_1 X_2 X_3 [X_1, X_2, X_3: carry at 1st, 2nd & 3rd place resp.].

Domains: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}.

Constraints
- alldiff (F, T, U, W, R, O) i.e., all variables are different digits.

$O + O = R + 10 \cdot X_1$, etc,

$W + W + X_1 = U + 10 X_2$ etc.

Search Formulation using CSP:-

If I have to solve a problem using search, a state of the search space will be assignment to all the variables. If I say, 'assignment to all the variables', it is actually 'local search'.

When I usually do local search, I move in the solution space. But, when I am using systematic search, I typically do partially assignment space.

Here, we shall do systematic search where we'll do partial assignment spaces. Here, the initial state would be, 'no assignment whatsoever my initial state would be'.

Let, I already have assigned a L variables among n variables. So, I have (n-L) options where I can assign value.

For each variable, I can have d values, So, number of children will be;

$$(n-L).d$$

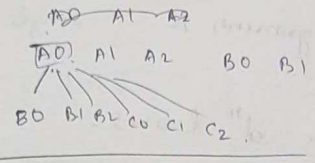
All the solutions are at depth n when all the variables are assigned.

So, how many leaves do I have?

At top level, # children : n.d
At 2nd " , # " : (n-1).d
At 3rd " , # " : (n-2).d

Total leaves : $n!\,d^n$

successor fucn is, assign a value to an unassigned variable that does not conflict with current assignment.



However, total assignments in my problem is d^n. So, the partial assignment goes wrong.

At some point, if I cannot assign a value, I backtrack. True for all CSPs.

Why do we have more states than possible total assignments? Because,

In search, each state represents the exact path we took to each state, ~~we know this~~ so we may assign x_1 true, x_2 false in a path. Also we may ~~a~~ x_2 false, x_1 true in a path. At the end of the day, these are same soln, but we are making diff. paths. Who cares, for the specific order.

So, what we can do, we can pick one variable first. Don't ~~too~~ assign all variables. At every point you think what is the best variable to assign, then I look all possible soln's.

This introduces Backtracking Search Algorithm.

⇒ Variable assignments are commutative here : i.e,

[A = red then B = green] same as [B = green then A = red]

- Only need to consider assignments to a single variable at each node
  ⇒ $b = d$ & there are d^n leaves.

(*) Depth First Search for CSPs with single variable assignments is called backtracking search.

(*) Backtracking search is the basic uninformed algorithm for CSPs

(*) It can solve n-queens for $n = 25$.

Now, what trick we can apply to make backtracking more efficient?

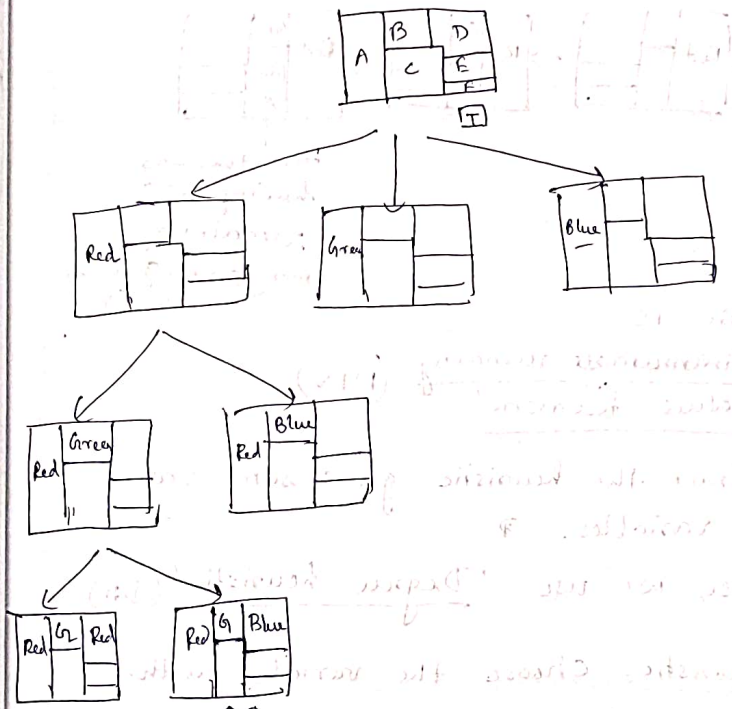
- The way you select the variable.
- " " " " " value for a variable.

So, improving backtracking efficiency involves

- which variable should be assigned ment?
- In what order should its value be tried
- Can we detect inevitable failure, early?
- Can we take advantage of problem structure?

SEND	TO	R TWO
MORE	GO	TWO
MONEY	OUT	FOUR

Backtracking example:-



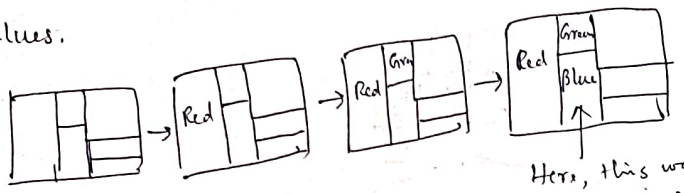
this one cannot lead to any solution. Because, C has no value remaining.

However, at this present moment of time, my algorithm does not know that there is no solution. So, what we did is, doing 'inference' in our brain that there will be a failure. Our brain can detect early. But the algo does not.

→ 'Search' is, exploring all possibility
→ 'inference' is that, we have proof that we should not go furd

because it will not lead to a solution, or if we should go fast because it will absolutely lead to a solution.

I - Which variable to assign?
⇒ Choose variable with the fewest remaining values.



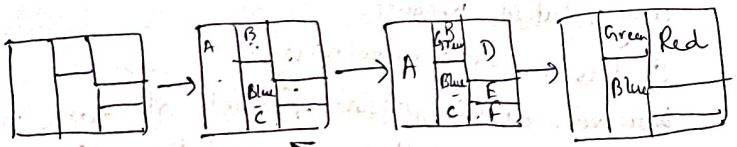
Here, this was having min remaining value among {red, green, blue}.

This heuristic is called 'minimum remaing values heuristic' (MRV)

→ However, sometimes the heuristic gives same value to multiple variables.

→ In that case, we use 'Degree heuristic' (DH)

→ Degree heuristic: Choose the variable with the most constraints on remaining variables.



here all 3 remaining variables have same value.
So, go for degree heuristic.

here C has maximum degree. So, assigning here first will make it simpler

[Check, if minimum remaining value is same for 2 states

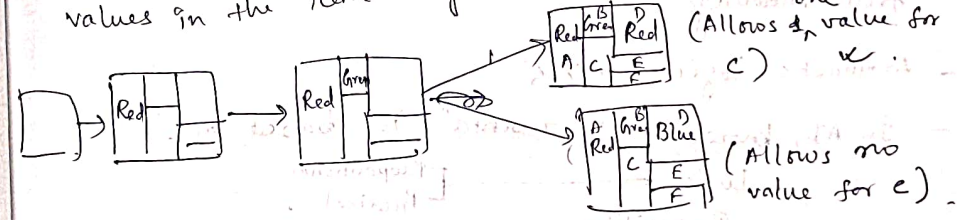
regions, then choose higher degree between two regions].
Here A had degree 0 with unassigned variable and D has degree 2. So, color D next.

[degree: # adjacent regions].

This is the variable ordering heuristic. It is not heuristic function of admissibility. It is just a heuristic.

⇒ Next Qsn is, which value to assign first?
Answer is, 'Least constraining value'.

» Least constraining value ⇒ (LCV).
Given a variable, choose the least constraining value: the one that rules out the fewest values in the remaining variables.



(Allows one value for c)

(Allows no value for c).

(looks like least constraing to other variables).

(**) Combining these two heuristics (MRV & LCV) can solve n-Queen for n = 1000.

» Constraint Propagation or Inference

I Forward Checking :⇒

Keep track of remaining legal values for unassigned variables.

Terminate search when any variable has no legal value.

PPt.

Arc consistency: (PPt).

Knowledge Representations

- It is one of the fundamental thing in AI.
- Represents knowledge about the world in a manner that facilitates inferencing (i.e., drawing conclusions) from knowledge.
- Arithmetic logic: $x \geq 5$
- In AI, knowledge representation is based on
  - Logic ← [Propositional, Predicate.
  - Probability
  - Logic & probability.
- Propositional logic

  gives in terms of True/False.

¬ Negation (Today is not raining)

∨ Disjunction (You can go or you can stay)

∧ Conjuction (You go and watch/stay).

→ if then (If there is rain then I won't go)

↔ iff (

atomic agent

Define objects

define relation between objects.

This will generate facts.

The relation is the predicate.

Person p is sitting in bench 25(a) ⇒ one fact

" p " not " " " 27 ⇒ another fact.

$(P \Rightarrow Q \;\&\; \neg P \Rightarrow Q)$

⇒ In the language of AI:

⇒ Atomic agent is state.

⇒ Propositional def^n are state variables.

⇒ A specific assignment to all propositional variables gives a state.

⇒ I am reducing the amount of effort it takes me to specify a problem by incorporating more & more structures in the real world representation.

→ Not only I am making facts that will be true or false, but also, I am modelling my degree of belief that,

I know it to be true, or false, but in the middle, if I don't know it is true or false, I may have preference whether it is most likely to be true or false. :- This is Epistemological Commitment.

Truth table

$P \to Q \Rightarrow \neg P \vee Q$.

P	Q	P → Q
T	T	T
T	F	F
F	T	T ⟹ "innocent until proven guilty".
F	F	T

bi-implication

$P \leftrightarrow Q$.

$\Rightarrow (P \to Q) \wedge (Q \to P)$.

Modus Ponen:

It is a deductive argument form that states if a conditional statement is true, then a subsequent statement is also true.

if $P \to Q$ is true
if P is true
then Q is true

$P \to Q, P \vdash Q$.

Modus tollens :-

$P \to Q$
$\neg Q$
$\therefore \neg P$.

(P → Q) If dog detects an intruder, dog will bark
(¬Q) The dog did not bark
(¬P) no intruder was detected by the dog.

Resolution in FOL

Resolution

Resolution is a theorem proving technique that proceeds by building refutation proofs, i.e., proofs by contradictions. It was invented by a Mathematician John Alan Robinson in the year 1965.

Resolution is used, if there are various statements are given, and we need to prove a conclusion of those statements. Unification is a key concept in proofs by resolutions. Resolution is a single inference rule which can efficiently operate on the **conjunctive normal form or clausal form**.

Clause: Disjunction of literals (an atomic sentence) is called a **clause**. It is also known as a unit clause.

Conjunctive Normal Form: A sentence represented as a conjunction of clauses is said to be **conjunctive normal form** or **CNF**.

> Note: To better understand this topic, firstly learns the FOL in AI.

The resolution inference rule:

The resolution rule for first-order logic is simply a lifted version of the propositional rule. Resolution can resolve two clauses if they contain complementary literals, which are assumed to be standardized apart so that they share no variables.

$$\frac{l_1 \vee \dots \vee l_k, \qquad m_1 \vee \dots \vee m_n}{\text{SUBST}(\theta, l_1 \vee \dots \vee l_{i-1} \vee l_{i+1} \vee \dots \vee l_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n)}$$

Where $\mathbf{l_i}$ and $\mathbf{m_j}$ are complementary literals.

This rule is also called the **binary resolution rule** because it only resolves exactly two literals.

Example:

We can resolve two clauses which are given below:

[Animal (g(x) V Loves (f(x), x)] and **[¬ Loves(a, b) V ¬Kills(a, b)]**

Where two complimentary literals are: **Loves (f(x), x) and ¬ Loves (a, b)**

These literals can be unified with unifier **θ= [a/f(x), and b/x]** , and it will generate a resolv clause:

[Animal (g(x) V ¬ Kills(f(x), x)].

Steps for Resolution:

1. Conversion of facts into first-order logic.
2. Convert FOL statements into CNF
3. Negate the statement which needs to prove (proof by contradiction)
4. Draw resolution graph (unification).

To better understand all the above steps, we will take an example in which we will apply resolution.

Example:

a. **John likes all kind of food.**

b. **Apple and vegetable are food**

c. **Anything anyone eats and not killed is food.**

d. **Anil eats peanuts and still alive**

e. **Harry eats everything that Anil eats.**
Prove by resolution that:

f. **John likes peanuts.**

Step-1: Conversion of Facts into FOL

In the first step we will convert all the given statements into its first order logic.

a. ∀x: food(x) → likes(John, x)

b. food(Apple) ∧ food(vegetables)

c. ∀x ∀y: eats(x, y) ∧ ¬ killed(x) → food(y)

d. eats (Anil, Peanuts) ∧ alive(Anil).

e. ∀x : eats(Anil, x) → eats(Harry, x)

f. ∀x: ¬ killed(x) → alive(x) } added predicates.

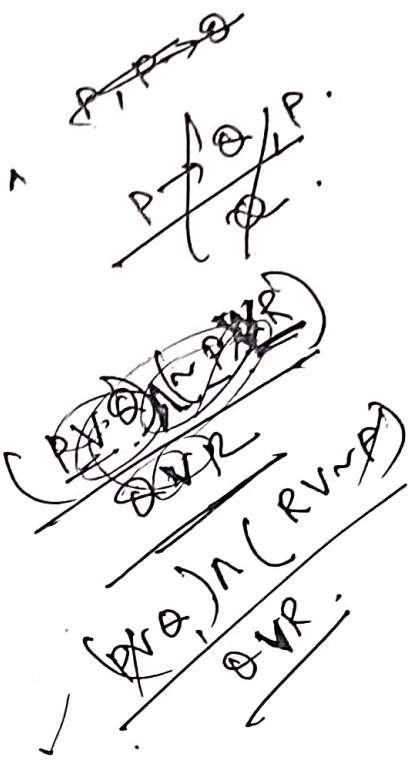
g. ∀x: alive(x) →¬ killed(x) }

h. likes(John, Peanuts)

Step-2: Conversion of FOL into CNF

In First order logic resolution, it is required to convert the FOL into CNF as CNF form makes easier for resolution proofs.

- **Eliminate all implication (→) and rewrite**
  a. ∀x ¬ food(x) V likes(John, x)
  b. food(Apple) Λ food(vegetables)
  c. ∀x ∀y ¬ [eats(x, y) Λ ¬ killed(x)] V food(y)
  d. eats (Anil, Peanuts) Λ alive(Anil)
  e. ∀x ¬ eats(Anil, x) V eats(Harry, x)
  f. ∀x¬ [¬ killed(x)] V alive(x)
  g. ∀x ¬ alive(x) V ¬ killed(x)
  h. likes(John, Peanuts).
- **Move negation (¬)inwards and rewrite**
  a. ∀x ¬ food(x) V likes(John, x)
  b. food(Apple) Λ food(vegetables)
  c. ∀x ∀y ¬ eats(x, y) V killed(x) V food(y)
  d. eats (Anil, Peanuts) Λ alive(Anil)
  e. ∀x ¬ eats(Anil, x) V eats(Harry, x)
  f. ∀x ¬killed(x)] V alive(x)
  g. ∀x ¬ alive(x) V ¬ killed(x)
  h. likes(John, Peanuts).
- **Rename variables or standardize variables**
  a. ∀x ¬ food(x) V likes(John, x)
  b. food(Apple) Λ food(vegetables)
  c. ∀y ∀z ¬ eats(y, z) V killed(y) V food(z)
  d. eats (Anil, Peanuts) Λ alive(Anil)
  e. ∀w¬ eats(Anil, w) V eats(Harry, w)
  f. ∀g ¬killed(g)] V alive(g)
  g. ∀k ¬ alive(k) V ¬ killed(k)



h. likes(John, Peanuts).

- **Eliminate existential instantiation quantifier by elimination.**
  In this step, we will eliminate existential quantifier ∃, and this process is known as **Skolemization**. But in this example problem since there is no existential quantifier so all the statements will remain same in this step.
- **Drop Universal quantifiers.**
  In this step we will drop all universal quantifier since all the statements are not implicitly quantified so we don't need it.

  a. ¬ food(x) V likes(John, x)

  b. food(Apple)

  c. food(vegetables)

  d. ¬ eats(y, z) V killed(y) V food(z)

  e. eats (Anil, Peanuts)

  f. alive(Anil)

  g. ¬ eats(Anil, w) V eats(Harry, w)

  h. killed(g) V alive(g)

  i. ¬ alive(k) V ¬ killed(k)

  j. likes(John, Peanuts).

> Note: Statements "food(Apple) ∧ food(vegetables)" and "eats (Anil, Peanuts) ∧ alive(Anil)" can be written in two separate statements.

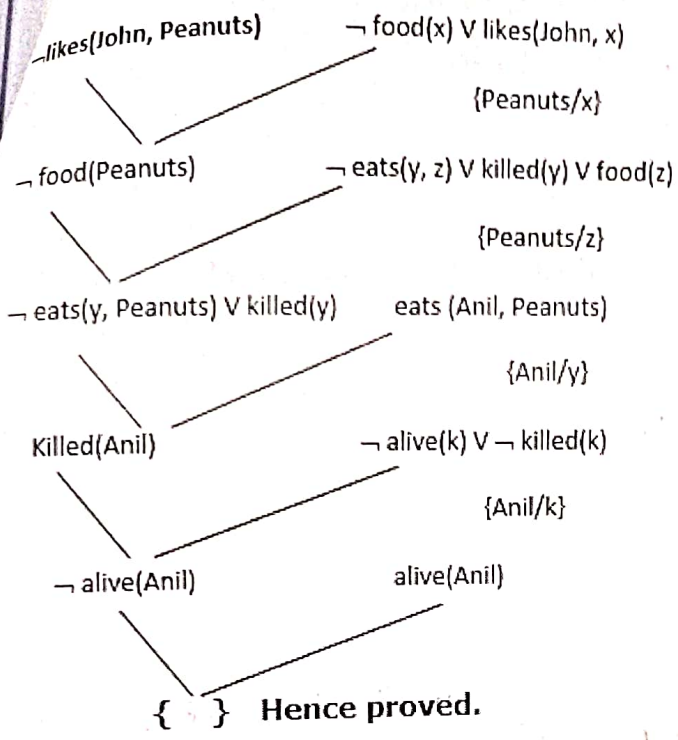
- **Distribute conjunction ∧ over disjunction ¬.**
  This step will not make any change in this problem.

Step-3: Negate the statement to be proved

In this statement, we will apply negation to the conclusion statements, which will be written as ¬likes(John, Peanuts)

Step-4: Draw Resolution graph:

Now in this step, we will solve the problem by resolution tree using substitution. For the above problem, it will be given as follows:



Hence the negation of the conclusion has been proved as a complete contradiction with the given set of statements.

Explanation of Resolution graph:

- In the first step of resolution graph, **¬likes(John, Peanuts)** , and **likes(John, x)** get resolved(canceled) by substitution of **{Peanuts/x}**, and we are left with **¬ food(Peanuts)**
- In the second step of the resolution graph, **¬ food(Peanuts)** , and **food(z)** get resolved (canceled) by substitution of **{ Peanuts/z}**, and we are left with **¬ eats(y, Peanuts) V killed(y)** .
- In the third step of the resolution graph, **¬ eats(y, Peanuts)** and **eats (Anil, Peanuts)** get resolved by substitution **{Anil/y}**, and we are left with **Killed(Anil)** .
- In the fourth step of the resolution graph, **Killed(Anil)** and **¬ killed(k)** get resolve by substitution **{Anil/k}**, and we are left with **¬ alive(Anil)** .
- In the last step of the resolution graph **¬ alive(Anil)** and **alive(Anil)** get resolved.

← Prev

Next →

Clausal form:

- A literal is either an atomic sentence or a negation of an atomic sentence

  $P, \neg P$.

- A clausal sentence is either a literal or a disjunction of literals.

  $P, \neg P, P \vee \neg Q$

- A clause is a set of literals.

  $\{P\}, \{\neg P\}, \{P, \neg Q\}$.

- Conversion to clausal form

  $P \rightarrow Q \Rightarrow \neg P \vee Q$.

  $P \leftrightarrow Q \Rightarrow (\neg P \vee Q) \wedge (\neg P \vee \neg Q)$] implications out.

→ Negations

$\neg\neg P \Rightarrow P$

$\neg (P \wedge Q) \Rightarrow \neg P \vee \neg Q$

$\neg (P \vee Q) \Rightarrow \neg P \wedge \neg Q$.

Distribution

$P \vee (Q \wedge R) \Rightarrow (P \vee Q) \wedge (P \vee R)$.

$(P \wedge Q) \vee R \Rightarrow (P \vee R) \wedge (Q \vee R)$

$P \vee (Q \vee \ldots \vee R) \Rightarrow (P \vee Q \vee \ldots \vee R)$

Operators out:

$P_1 \vee P_2 \vee \ldots \vee P_n \Rightarrow \{P_1, P_2, \ldots, P_n\}$

$P_1 \wedge \ldots \wedge P_n \Rightarrow P_1, \ldots, P_n$.

→ Resolution is a single rule of inference that can operate efficiently on a special form of sentences.

→ The special form is called conjunctive normal form (CNF) or clausal form.