mysine software
4 queue problem.

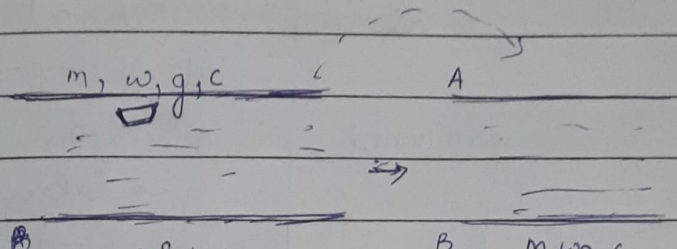
Sabyasachi
DATE / /

8-2-21

PROBLEM SOLVING:

1. Knowledge Representation
2. Converting problem in state space and then apply search algo.

E.g River crossing problem

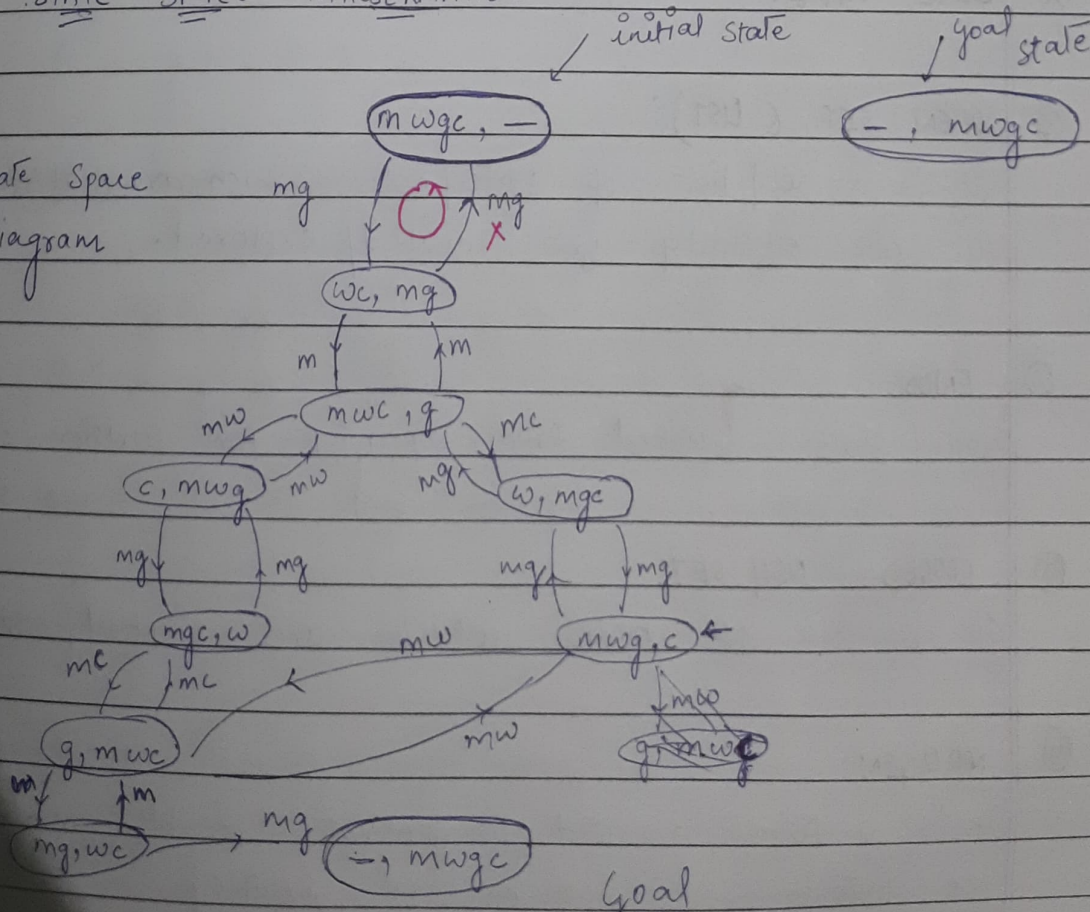


m = man
w = wolf
g = goat
c = cat

① any ~~two~~ max. two must be in boat
② w → g (not allowed), g - c (not allowed) pair.
③ man can row the boat.

STATE SPACE REPRESENTATION:

State Space Diagram

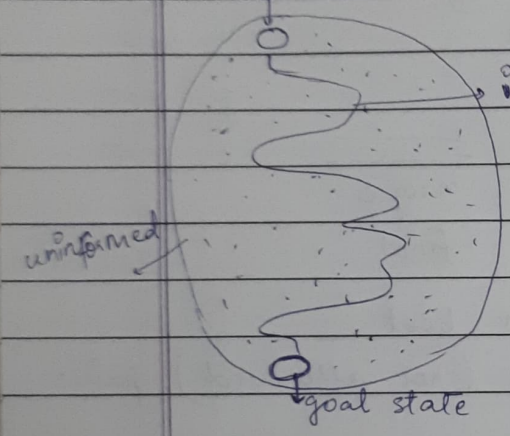


Difference b/w ai search & graph search.

Sabyasachi
DATE / /

AI SEARCH

- Search
  - Uninformed search (Blind search)
    - BFS
    - uniform cost search
    - DFS
    - Depth limited search
    - Iterative deepening search
  - Informed Search (Heuristic Search)
    - Hill climbing
    - Steepest Ascent Hill climbing
    - Best first search
    - A* algo.
    - AO* algo
    - Simulated Annealing.



SOME TERMS:

① OPEN SET (LIST):

It is a set/list of state/nodes which are already generated but not ~~already~~ yet visited/explored.

13/1/2021

② ENODE:

Node/state selected from openlist for further expansion

③ CLOSED LIST/SET:

It is set of nodes which are visited/expanded.

④ MOVGEN:

It is a function used to generate possible moves from the state.

MOVGen (S) = {A, B, C}

⑤ TESTGOA
It is
is goa

Basic
⇒ "gene

If openlist is empty then with unsuccess search.

Sabyasachi
DATE / /

① TESTGOAL (Boolean function)
It is a function which says that given state / node is goal or not.

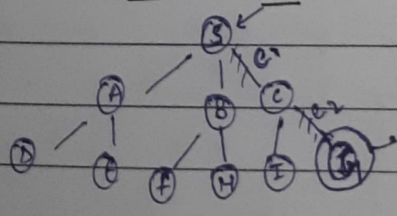
TestGoal (S) = No
Yes

Basic Algo.

⇒ "generate and Test" / unsystematic with successful search / random search

1. a Openlist = { start }
b. TestGoal (start) if yes exit, otherwise continue.
2. a. Enode = Select node 'N' from open list
b. closelist = { closelist ∪ {N} }
3p If openlist is empty then with unsuccessful search.
4. MoveGen (Enode)
5. a Apply Test goal to each node generated in step (4)
b If goal node is found then stop with succ. search. Otherwise continue.
6. openlist = Openlist ∪ { generated node from step 4 }
7. goto step 2.

8/1/2021 BREADTH FIRST SEARCH (BFS)



openlist = {S} → Queue
" = {A, B, C}
close list = {S, A}
{S, A, B, C}

* use queue as a data structure.

openlist
[A] B C
[B] C D E
[C] D E F H

path : S e1 C e2 G.

* Moving to nearest neighbour.

Sabyasachi
DATE / /

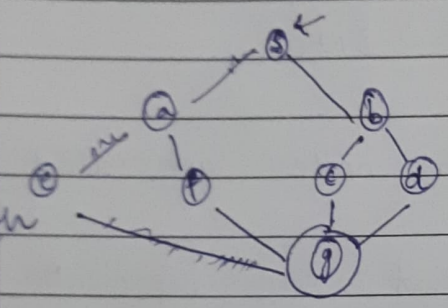
MEASURES OF SEARCHING ALGORITHMS:

1. Time Complexity
2. Space complexity
3. completeness
4. Optimality: (guarantee that we get optimal path)

COMPLETENESS

* Can this algorithm give guarantee that we can reach to goal state at finite ~~state~~ distance.

15/2/24

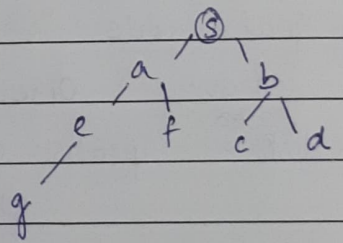


openlist = [s] - [a|b]

closelist = {} → {s} → {s,a}

[b|e|f] → [e|f|c|d] - [f|c|d|g]

{s,a,b} → {s,a,b,e}

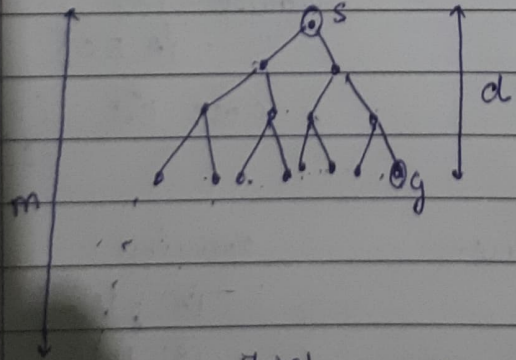


let, branching factor = 'b'

Distance of goal from start = 'd'

Max. depth of state space = 'm'

WORST CASE



Total no. nodes = $\{b^0 + b^1 + b^2 \dots b^d\}$

$$\Rightarrow \frac{1\,(b^{d+1}-1)}{(b-1)} = \frac{b^{d+1}-1}{b-1}$$

TIME COMPLEXITY

Normally we consider 'b' as constant but 'd' is always variable

Time Complexity: $O(b^{d+1})$ or $O(b^d)$

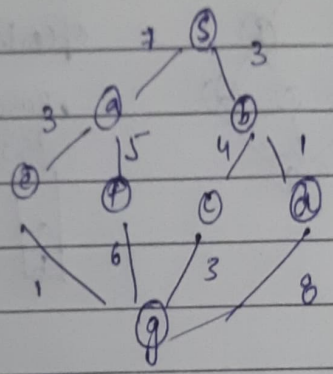
Sabyasachi
DATE / /

Time Complexity :) constant variable
[exponential Time complexity]

SPACE COMPLEXITY $\frac{b^{d+1}-1}{b-1}$

$O(b^{d+1})$ or $O(b^d)$

Uniform cost search

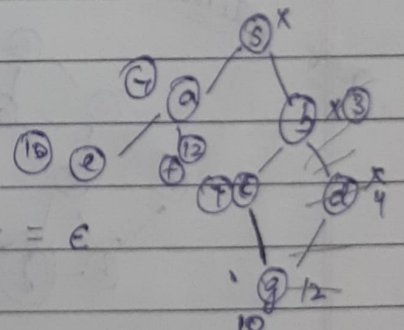


* open list is not FIFO queue rather it is priority queue.

open list <S> [3 b | 7 a] [d | a | c] [a | c | g] [c | e | f]

close list < > <S> <S,b> <S,b,d>

① Average branching factor : b
optimal path cost = c^*
let least cost edge of weight = ϵ



Then no. edges (path) b/w source and goal will be $\leq \frac{c^*}{\epsilon}$
(depth)

If $d = \frac{c^*}{\epsilon}$ (worst case)

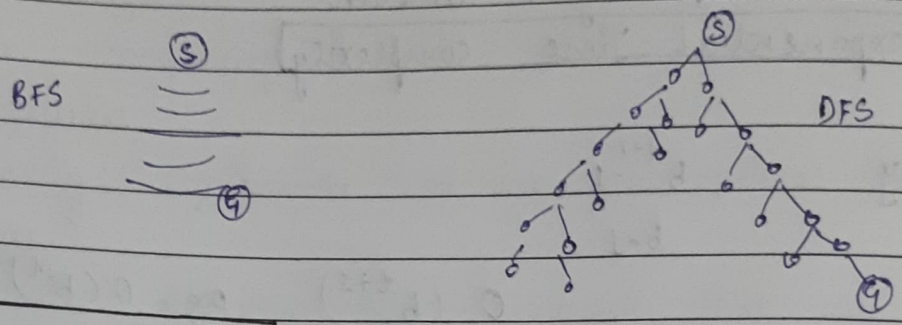
Time complexity = $O(b^{d+1}) \Rightarrow O(b^{c^*/\epsilon + 1})$

Space complexity = $O(b^{d+1}) \Rightarrow O(b^{c^*/\epsilon + 1})$

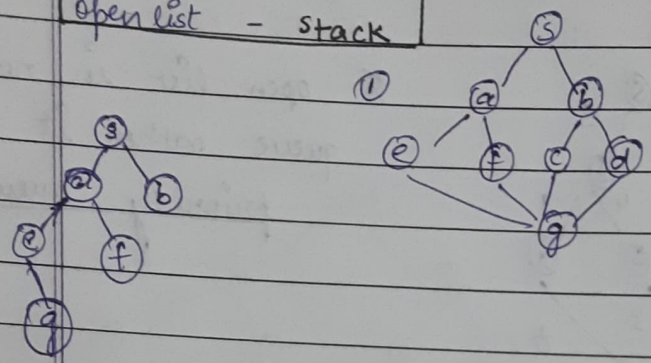
Sabyasachi
DATE / /

20/1/21

DEPTH FIRST SERCH:



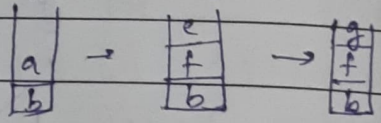
Open list - stack



open list = [s]

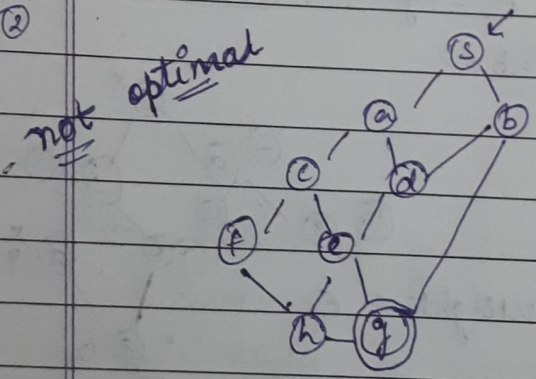
close list = { }

{s} {s, a}

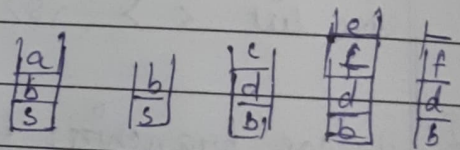


Que

(2)



not optimal



{s}

* not guarantee of completeness.
* not optimal.

Sabyasachi
DATE / /

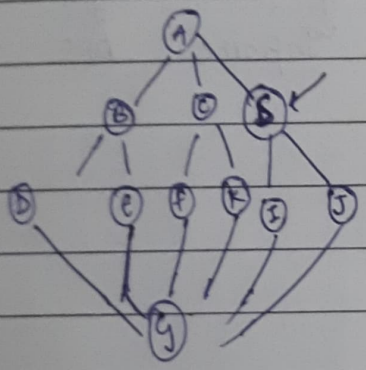
20/1/21 TIME / SPACE COMPLEXITY:

Initial

Let max depth = m
branch factor = b
goal is at depth = d

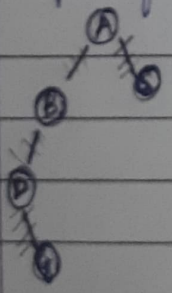
m — goal

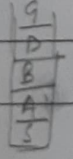
* Time complexity = $O(b^m)$
* Space complexity = $O(b \cdot m)$

Que: 

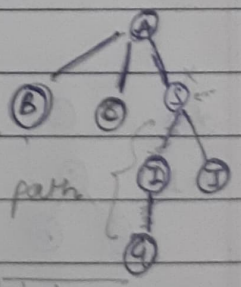
Draw Breadth first and Depth first spanning Tree.

Depth first





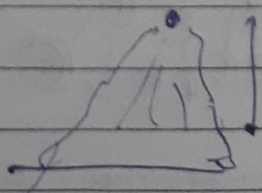
Breadth first Spanning tree



A | B
J | J | B | C

* DEPTH LIMITED SEARCH

It is DFS but limit of depth is set at the beginning to avoid infinite path / loop.
max depth = 'l'



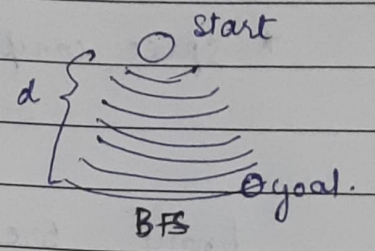
Sabyasachi
DATE / /

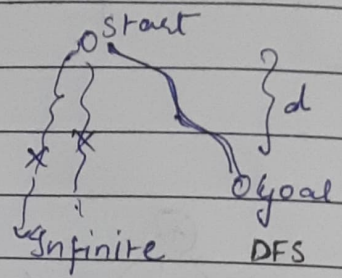
27/1/21

ITERATIVE DEEPENING:

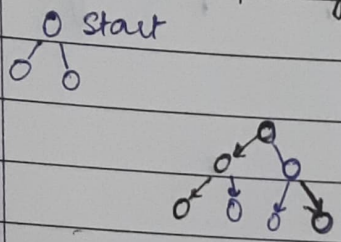
It is good mix of BFS and DFS.

GOAL
→ Complete like BFS
→ Optimal like BFS
→ Space complexity like DFS.

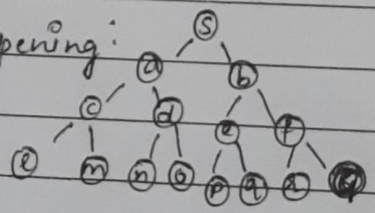




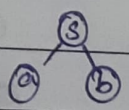
Iterative Deepening:

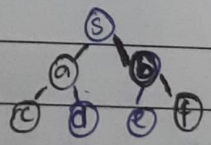


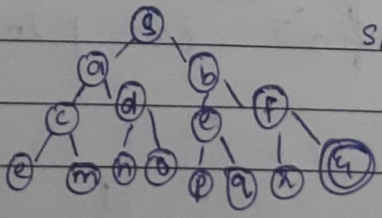
Apply iterative deepening:



① d=0 (S)

② d=1  S, a, b

③ d=2  S, a, b, c, d, b, e, f

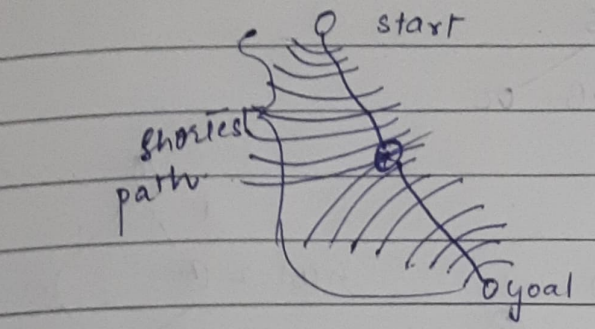
④ d=3  S, a, c, e, m, d, n, o, b, e, p, q, f, x, q

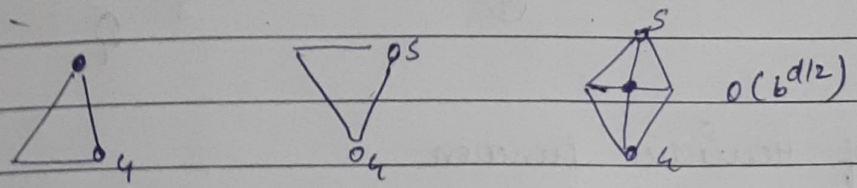
Time complexity = $O(b^d)$

Space complexity = $\Theta(b \cdot d)$

Sabyasachi
DATE / /

BIDIRECTIONAL SEARCH!



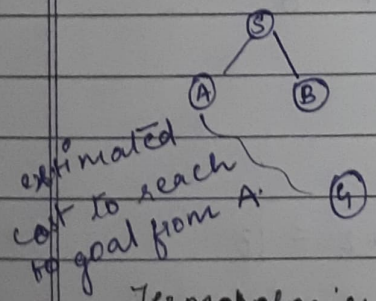


INFORMED SEARCH (HEURISTIC SEARCH)

Heuristic function:

h(node) or state = value

estimated value to reach to goal from this node.



Terminologies of Heuristic Functions.

① Admissible Heuristic: A heuristic is said to be admissible if it never overestimate the cost of reaching goal from any node.

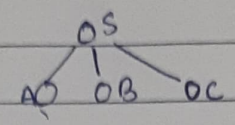
Let 'x' is a node.

$h^*(x)$ is actual cost to reach to goal from 'x'

If heuristic 'h' is admissible then $h(x) \leq h^*(x)$

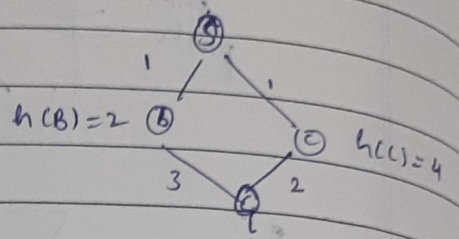
Sabyasachi
DATE / /

for $\forall x$



$h(A) \leq 10$

Actual cost = 10



Example of Heuristic function

① 8-puzzel problem

7	2	4
5		6
8	3	1

Start

→

	1	2
3	4	5
6	7	8

Goal

Heuristic Function : ① No. of misplaced tiles.

$h(S) = 1+1+1+1+1+1+1+1 = 8$

② Man Lathen Distance

$3+1+2+2+2+3+3+2 = 18$

$h(S) = 18$

→ Dominance of Heuristic

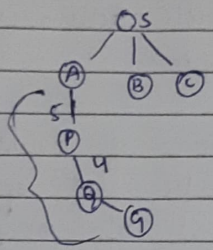
Let h_1 and h_2 are two admissible heuristic st.

$h_1(x) \geq h_2(x) \quad \forall x$

Then h_1 is called dominating heuristic.

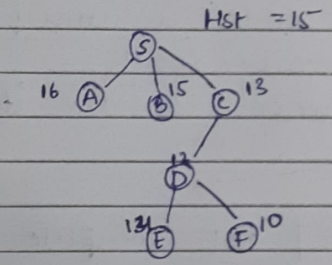
Date
Page No.

consistent heuristic : (always admissible heuristic but
= wise versa is not true),
More strictic heuristic.



Heuristic Search Algorithm:

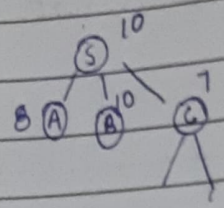
① HILL CLIMBING ALGORITHM:



Step 1: Calculate heuristic cost of start node.
If it is zero stop with succ. Search otherwise continue.
Step 2: Put start node in open list.
Step 3: select node from open list for expansion & remove from openlist.
Step 4: Generate one node at a time and calculate its heuristic cost.
Step 5: i.) If cost is zero stop with succ. search.
ii.) If cost is worse / equal than parent then discard it.
3.) If it is less then put it in open list.
Step 5: goto 3.

* If openlist is empty than stop with unsuccessful search.

2. STEEPEST ASCENT HILL CLIMBING:

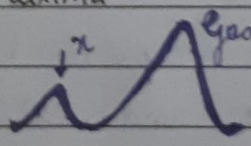


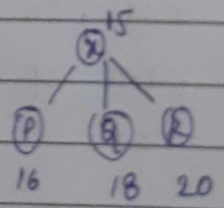
① Calculate h(s) if it is zero stop with Succ. Search otherwise set consider it Enode.

② Generate all options of Enode
Calculate heuristic cost of each option
i) If heuristic cost of any option is zero than stop with Succ. Search
ii) Select best (least cost) node & compair the cost with parent.
iii) If cost of best node is worse / equal to parent stop with unsucc. Search
iv) If cost is better consider it as Enode & goto 2.

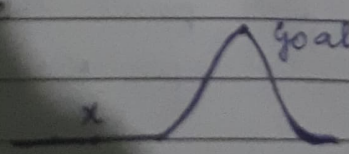
★ These algorithm will fail / not complete because.
1. Local Maxima.
2. Platue
3. Ridge

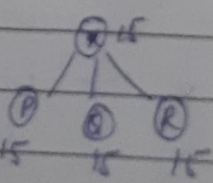
1. Local Maxima



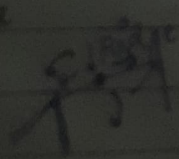


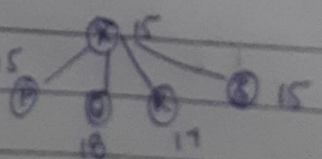
2. Platue





3. Ridge





Sabyasachi
DATE / /

3. Best First Search

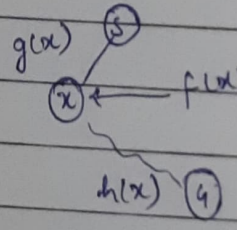
To calculate cost of any node 'x'.

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

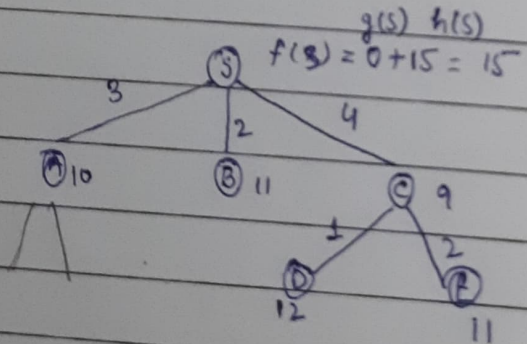
cost of node 'x'

$g(x)$: it is actual cost to reach to node 'x' from starting node.

$h(x)$: Heuristic cost of 'x' is estimated cost to reach to goal node from 'x'.



Ex:



f(A) = 3+7 = 10

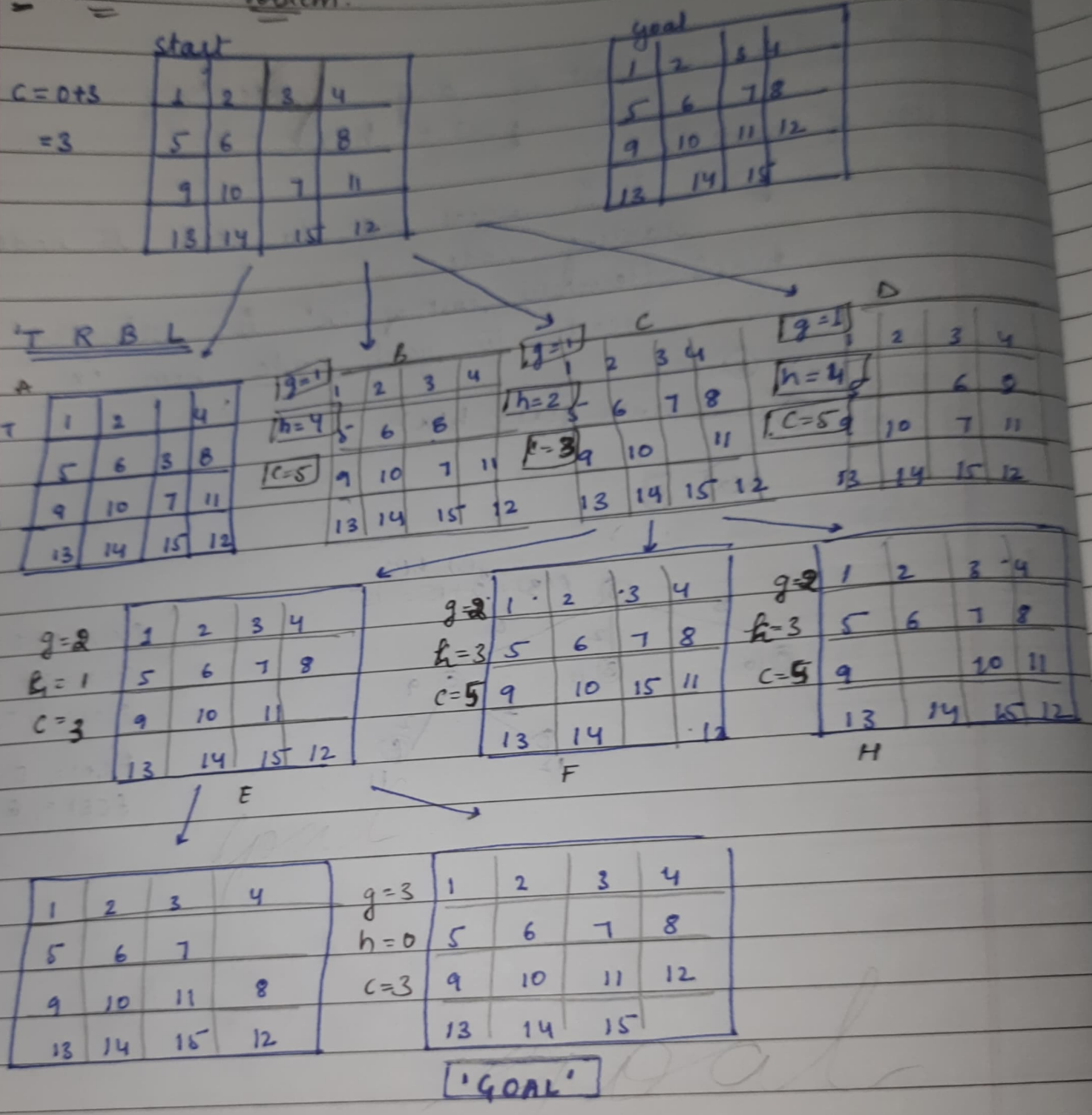
f(B) = 2+9 = 11

f(C) = 4+5 = 9

f(D) = 5+7 = 12

f(E) = 6+5 = 11

15 Puzzle Problem:



Sabyasachi
DATE / /

A* Algorithm

1. Check that starting node is goal? by calculating its heuristic cost, if heuristic cost is 0 that means start node is goal node stop with succ. search otherwise include start node in open list.

2. Select best node (least cost node) from open list for expansion, and place this node in closed list. ~~If open~~ If openlist is empty Stop with unsucc. search.
3. Generate all possible options (nodes) from e. nodes.

a.) Calculate total cost of each node by calculating 'g' cost & 'h' cost.

b. If 'h' cost of any node is zero stop with succ. search

c. Check that generated node is already in openlist.
If yes
i.) Compair the new cost with old one.
ii. If new cost is worse than old than no change.
iii.) New cost is better then old then update the new cost as cost of the node and change the parent link to new parent.

d. Check that generated node is already in closed list.
If yes.
i. Compare new cost with existing cost.
ii. If new cost is worst than no change.
iii If new cost is better
(α) Update the cost of node to new cost.
(β) change the parent link to new parent.

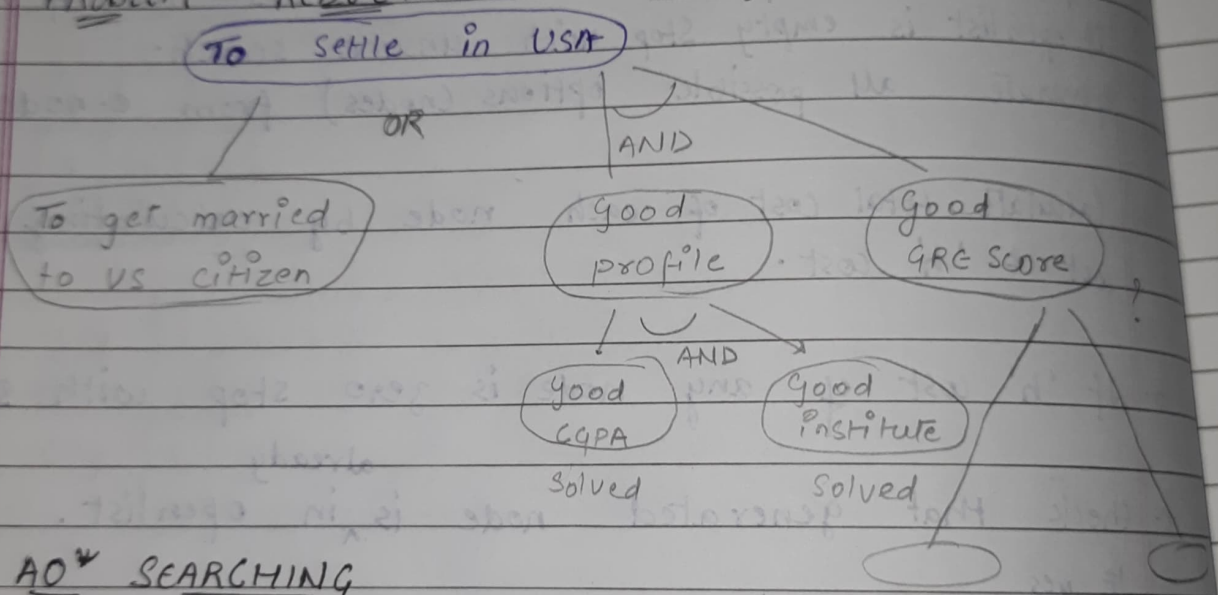
(d) Propogate its 'g' value to entire sub of this node.

(e) If node are neither in open list nor in closed list nor goal then place them in open list, and goto step (2).
(set their parent link.)

Propogate

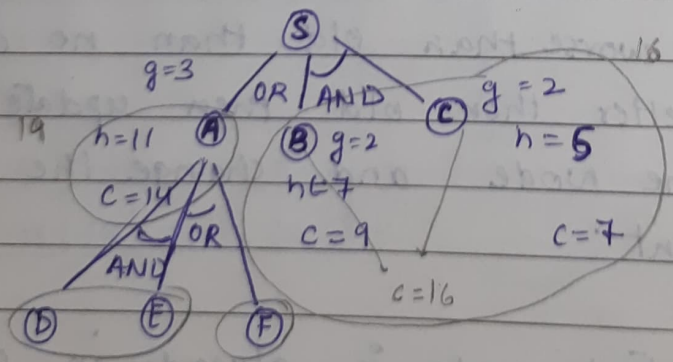
10/2/25

PROBLEM REDUCTION TECHNIQUE:

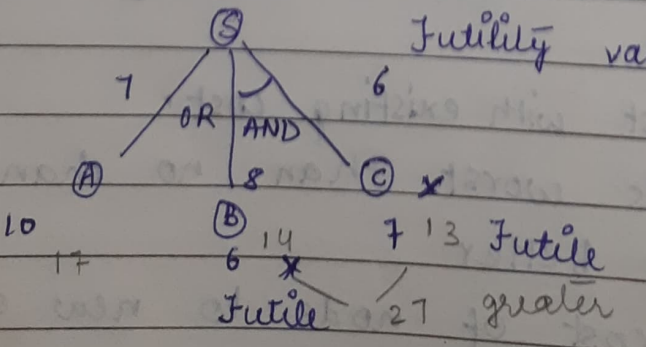


AO* SEARCHING

AND-OR GRAPH



⇒

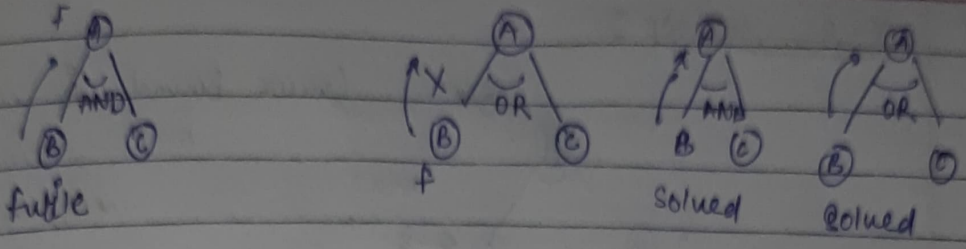


Futility value 25

greater than futility value

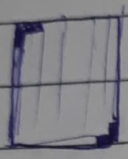
Sabyasachi
DATE / /

Propagate



10/2/21 GAME PLAYING

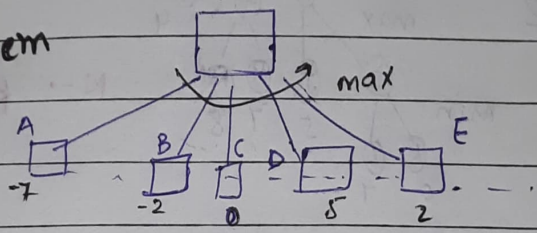
Two person game - One side is human and other side is computer program (AI prog.).



Board position state given to computer by human.

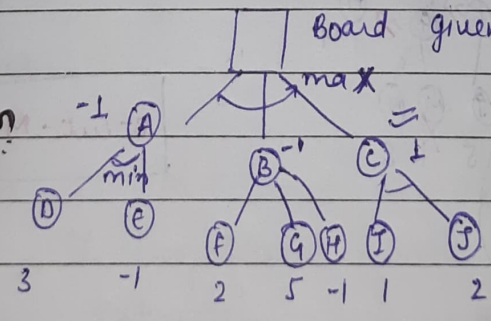
on an average 32 moves are possible [chess]

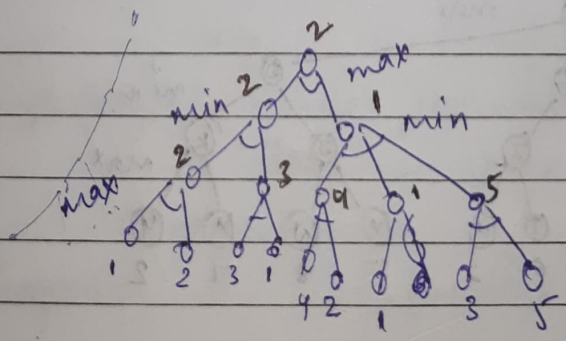
One ply system



* heuristic will give integer value ranges from -10 to +10
If value is "-ve" the board position is such that computer will lose the game.

Two ply system



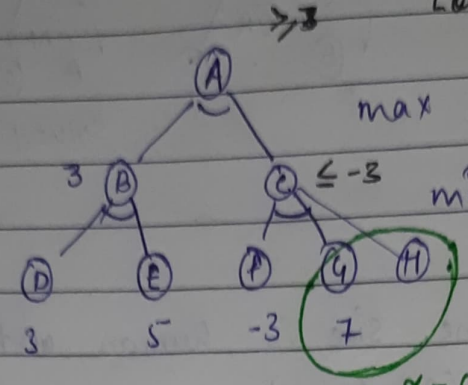


Max - Min Algorithm

12-02-2024

Sabyasachi
DATE / /

α-β cut off :

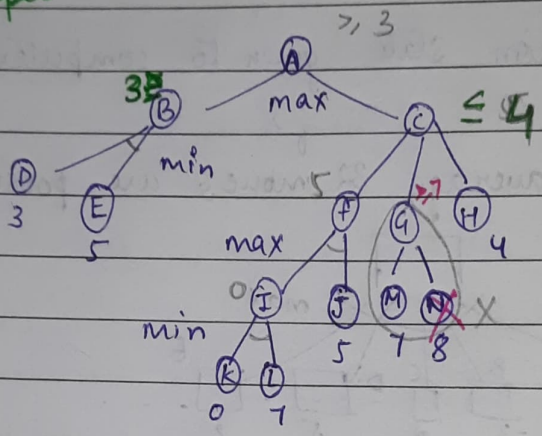


Lower bound is called max - min ply
α(A) = 3

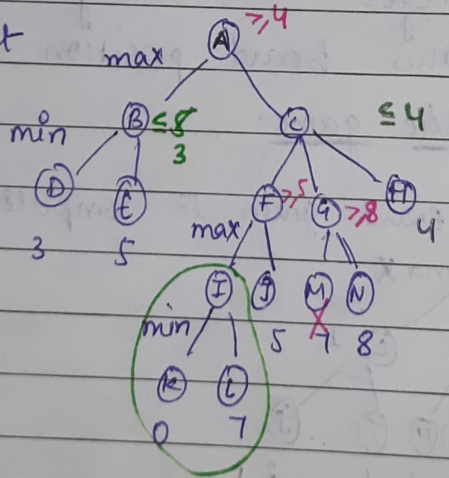
upper bound is called 'β value' β(C) = -3

max ply = α lower
min ply = β upper

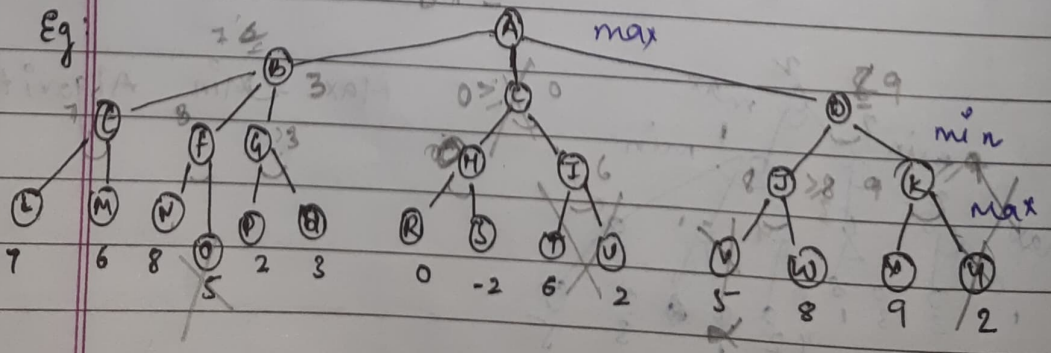
Eg.



right - left



Eg:



Sabyasachi
DATE / /

i. apply max min algo. and find next move. A → D →
ii. Apply L to R α-β cutoff
iii. apply R to L α-β cutoff

ii) β cut = 0, Y
α cut = , T, T U.

iii) α = V,
β =

Knowledge Representation

Propositional Logic: (Zero order logic)
In propositional logic, knowledge is represented in the form propositional logic expressions.
Any Expression. → operands
→ operators

The value of any expression is either True or False.

Operands: i.) T & F are the operands.
ii. Propositions are operands of prop. logic expression.

Predicate Proposition: → A simple statement whose value is true or false is represented in the form of preposition.

→ Proposition are like function

Example of

1.) Mahesh is father of Atul.

Proposition FATHER (Mahesh, Atul)

2.) x is father of y.

FATHER (x, y)

3.) Amit is Intelligent

I (Amit)

4.) I (x) : x is intelligent

5.) F(x): x is father

* Proposition are represented using capital letters.
In propositional logic expression we can also use 'function'

→ Functions are like propositions but they return non boolean values.

→ Function are ~~used~~ not operands of prop. logic exp rather these are used within proposition.

Examples of function

age (x): age of 'x'

father (x): Father of 'x'.

Ex: "age of 'x' is more than 30".

proposition. MORE (x, y) : 'x' is more than 'y'

MORE (age (x), 30)

Proposition ↳ function

* Normally function are written in Our case LHCs.

Sabyasachi
DATE / /

max (x, y)
sum (x, y)

Operators:

highest
1. ¬ or ~ "negation" } Unary operators
2. Conjunction ∧ (And)
3. Disjunction ∨ (OR)
4. Implication → } Binary operators
least 5. Equivalence ↔

parenthesis
2 + (3 * 4) They are neither operator nor operand

17-02-2021 Propositional logic expressions (zero order logic)
- Knowledge is represented in the form of expression
- In expression we have operand & operators
- operators are

- Operands
  - T or F
  - Propositions.
- Propositions: It is simple statement whose value is true or false and represented by english alphabet or word.

	Statements	Preposition
eg.	"It is raining"	R
	Car is open	O
	Car is wet	W

Ex:2 If it rains and car is in open, it will be wet.

$(R \wedge O)$: It is raining and car is open.

$R \wedge O \rightarrow W$

Ex:2 If it rains and car not in shed then car will be wet.

S: car is not in shed. } not got

$(R \wedge \neg S) \rightarrow W$

S: car is in shed,

$(R \wedge (\neg S)) \rightarrow W$

Ex:3 If it rains but I stay at home, I won't be wet.

R → It rains

H → stay home

W → I get wet.

$(R \wedge H) \rightarrow \neg W$

Ex:4 I will be wet if it rain.

$R \rightarrow W$

Ex:5 If it rains and the picnic is not cancled or I don't stay home, I will be wet.

P: Picnic is cancelled.

$(R \wedge (\neg P \vee \neg H)) \rightarrow W$

Sabyasachi
DATE / /

Ex:6 Whether or not picnic is cancelled, I am staying at home if it rains.

$\Rightarrow (P \vee \neg P) \wedge R \rightarrow H$ OR $R \rightarrow H$

↘ Equivalent.

Ex:7 Either it doesn't rain or I stay Home.

$\neg R \vee H$

19/2/21 LAWS OF PROPOSITIONAL:

1. COMMUTATIVE LAWS:

$A \vee B = B \vee A$

$A \wedge B = B \wedge A$

$A \leftrightarrow B = B \leftrightarrow A$

$A \rightarrow B$ not commutative

2. ASSOCIATIVE LAWS:

$A \vee (B \vee C) = (A \vee B) \vee C$

$A \wedge (B \wedge C) = (A \wedge B) \wedge C$

3. DISTRIBUTIVE LAWS:

$A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$

$A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$

4. Demorgon's Laws

$\neg(A \vee B) = \neg A \wedge \neg B$

$\neg(A \wedge B) = \neg A \vee \neg B$

5. LAW OF NEGATION

$\neg(\neg A) = A$

6. LAW OF EXCLUDED MIDDLE
$A \vee \neg A = T$

7. LAW OF CONTRA NEGATION CONTRADICTION
$A \wedge \neg A = F$

8. LAW OF IMPLICATION:
$A \rightarrow B = \neg A \vee B$

9. LAW OF EQUALITY
$A \leftrightarrow B = (A \rightarrow B) \wedge (B \rightarrow A)$

10. LAWS OF OR SIMPLIFICATION:
$A \vee A = A$
$A \vee T = T$
$A \vee F = A$
$A \vee (A \wedge B) = A$ ~~$(A \vee A) \vee (A \wedge B)$~~
~~$A \vee (A$~~

11. LAWS OF AND SIMPLIFICATION
$A \wedge A = A$
$A \wedge T = A$
$A \wedge F = F$
$A \wedge (A \vee B) = A$

Ex:1 $(\cancel{b \rightarrow c})\ (\neg b \rightarrow c) \wedge \neg(\neg b \rightarrow (c \vee d))) \rightarrow (\neg c \rightarrow d)$

law of implication ; law of negation

$\neg(\neg b \vee c) \wedge \neg(b \vee (c \vee d)) \rightarrow (\neg c \rightarrow d)$

~~$(\neg b \vee c) \wedge (\neg b \vee \neg(c \vee d)) \rightarrow (\neg c \rightarrow d)$~~

$(\neg b \vee c) \wedge (b \vee (c \vee d)) \vee (c \vee d)$

$(\neg b \vee c \wedge b) \wedge (\neg b \vee c \vee c \vee d) \vee (c \vee d)$

$(F \vee c) \wedge (\neg b \vee d \vee c) \vee (c \vee d)$

Sabyasachi
DATE / /

⇒ $(\neg(\neg b \vee c) \wedge \neg(b \vee c \vee d)) \rightarrow (c \vee d)$ law of negg
$((b \wedge \neg c) \wedge (\neg b \wedge c \wedge \neg d)) \rightarrow (c \vee d)$ law of Dem
$(b \wedge \neg c \wedge \neg b \wedge c \wedge \neg d) \rightarrow (c \vee d)$ law of impli
$F \rightarrow (c \vee d)$
$\neg F \vee (c \vee d)$
$T \vee c \vee d \Rightarrow T$

PROPERTIES OF LOGIC EXPRESSION:

1. Satisfiable: A logical Expression is said to be satisfiable if there exist some interpretation for which it is true.

   1. $a \vee b$ → satisfied
   2. $a \wedge b$ → satisfied
   3. $\neg a \wedge b \wedge a$ → not satisfied

2. Contradiction: A logical expression is said to be contradiction if it is false for every interpretation.

3. Valid / Tautology: A logical expression is said to be valid / tautology if it is "true" for all the interpritations.

4. Equivalence: Two logical expression are said to equivalent if they have same truth value under every interpretation.

   L_1: $a \rightarrow b$
   L_2: $\neg a \vee b$

Sabyasachi DATE / /

Find the expr. is satisfiable | valid | contingent

(1) $P \to Q \to \neg P$

$(\neg P \vee Q) \to \neg P$

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

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

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

T ⓣ $\wedge$ f or T $\not\equiv$ $(\neg Q \vee \neg P)$.

Satisfiable:

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

$A \vee \neg A \Rightarrow T$

Tautology | valid.

(3) $(P \vee Q) \to (P \wedge Q)$

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

$\neg A \vee A$

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

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

$(T \wedge \neg Q) \wedge (\neg P \wedge T)$

$\neg Q \wedge \neg P \Rightarrow$ Satisfiable

NORMALIZATION OF LOGIC EXPRESSION:

1. Conjunctive Normal formal (CNF)
2. Disjunctive Normal formal (DNF)

CNF: (Normal form):

1. Only negation, conjunction & disjunction operators are allowed.
2. Negation must be with individual operands.

Sabyasachi
DATE / /

CNF:

The expression is written in terms of AND of ORs.

Eg: $(a \vee b) \wedge (b \vee c) \wedge (c \vee d)$

Eg: $(\neg a \vee b) \wedge (\neg c \vee \neg d)$

Eg: $\neg (a \vee b)$ ✗ not in CNF

Eg: $(a \vee b) \wedge (\neg$ ~~not~~ in CNF

Eg: $a \wedge b$

Eg: $a \rightarrow b$ ✗ not in CNF

DNF OR of AND

Eg: $(a \wedge b) \vee (c \wedge d)$

Eg: $(a \wedge \neg b) \vee (\neg b \wedge c)$

3 $(a \wedge \neg b) \vee f$

4 $(a \vee b$

5 $\neg (a \vee b)$ ✗ not in DNF

6 $\neg a \rightarrow b$ ✗ not in DNF

Convert following in CNF.

1. $(P \rightarrow Q) \rightarrow R$

$(\neg P \vee Q) \rightarrow R$

$\neg (\neg P \vee Q) \vee R$

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

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

2. $P \vee (\neg P \wedge Q \wedge R)$

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

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

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

CLAUSE FORM:

A clause is logical expression that may have only negation and disjunction operator.

If negation is present then it must be with individual operand.

$\neg (a \vee b)$ ✗

$a \wedge b$ ✗

$\neg a \vee b$

$\neg a$

$a \vee b$

$a \rightarrow b$ ✗

INFERENCING TECHNIQUES (Reasoning Techniques) (RULES)

1. MODES PONEN

Given ① $A \rightarrow B$

② A

Infer : B

Statements:

If car is in open it

If it rains road will be wet

It is Raining

R: Raining W: wet road

① $R \rightarrow W$ ⇒ Rule

② R ⇒ fact

Infer : W

2. MODEL TOLLEN:

give : ① $A \rightarrow B$

② $\neg B$

Infer : $\neg A$

3. CHAIN RULE

give ① $A \rightarrow B$

② $B \rightarrow C$

Infer : $A \rightarrow C$

4. SIMPLIFICATION:

Given: $A \wedge B$

Infer: A

B

Sabyasachi
DATE / /

5. CONJUNCTION:

given: ① A
② B

Infer: $A \wedge B$

6. TRANSPOSITION:

Given: $A \rightarrow B$

Infer: $\neg B \rightarrow \neg A$