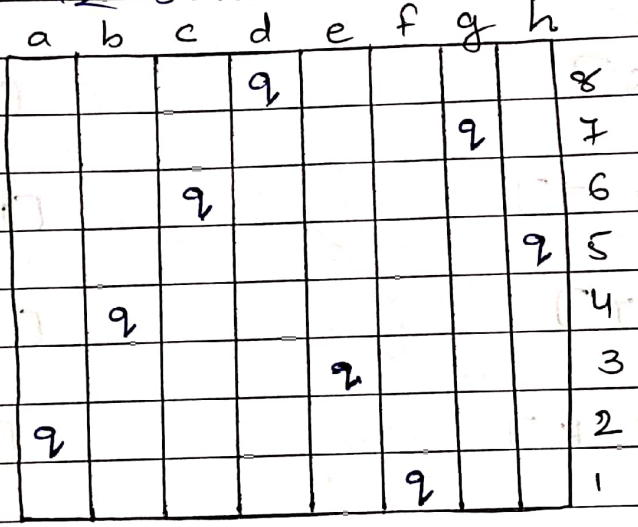
8-queen gama problem

For n queen nxn bo

For 8 queen
8 × 8 board.

→ The eight queens puzzle is the problem of placing eight chess queens on an 8×8 chess board so that no two queens threaten each other; thus, a solution requires that no two queens share the same row, column or diagonal.

→ There are 92 solutions.



Step-1:
If the remainder from dividing n by 6 is not 2 or 3 then the list is simply all even numbers followed by all odd numbers not greater than n.

Step-2:
Otherwise, write seperate lists of even and odd numbers
(2, 4, 6, 8 - 1, 3, 5, 7)

Step-3:
If the remainder is 2, swap 1 and 3 in odd list and move 5 to the end (3, 1, 7, 5)

Step-4:
If the remainder is 3, move 2 to the end of even list and 1, 3 to the end of odd list (4, 6, 8, 2 - 5, 7, 1, 3).

Step-5:
Append odd list to the even list and place queens in the rows given by these num. from left to right
$(a_2, b_4, c_6, d_8, e_3, f_1, g_7, h_5)$.

Farmer, Fox, Goat, Grass Problem

Condition:

(i) Goat and Grass cannot be together, otherwise it will eat the grass.

(ii) Fox catch goat alone then fox will eat the goat.

(iii) Farmer is the only driver of boat

(iv) Boat has capacity of two.

Notations:

Farmer → Fa Goat → G_1

Fox → Fo Grass → G_2

Start	River	
Fa, Fo, G_1, G_2		O
Fo, G_2	Fa, G_1 →	G_1
	← Fa	
Fo	Fa G_2 →	Fa G_1 G_2
Fa Fo G_1	← Fa G_1	G_2
G_1	Fa Fo →	Fa Fo G_2
	← Fa	
Fa G_1		Fo G_2
O	Fa G_1 →	Fa Fo G_1 G_2

Goal State.

3 Missionaries and 3 Cannibals Problems

Condition:

(i) On any side of river. No. of cannibals(C) should not be more than no. of missionaries (M). Otherwise (C) hunt (M).

(ii) Boat has capacity of two person. ~~Notation:~~

Notation:

for missionaries → M

For cannibals → C

Start		River	
Page	3M 3C		0
~~Step 1~~			
Step 1	3M 1C	2C →	2C
Step-2	3M 2C	← C	1C
Step-3	3M	2C →	3C
Step-4	3M 1C	← C	2C
Step-5	1M 1C	2M →	2M 2C
Step-6	2M 2C	← 1M 1C	1M 1C
Step-7	2C	2M →	3M 1C
Step-8	3C	← C	3M
Step-9	1C	2C →	3M 2C
Step-10	2C	← C	3M 1C
Step-11	0	2C →	3M 3C Goal State

Unit-3

Generators and Tester

Generator → generates possible soln

↓

two ways

(1) random → easy way to search

↓

it picks any solution ← doesn't gaurantee goal state

(2) systematic → completedness

↓

also known as DFS

Tester → test the fun^n or solution whether it is acceptable or not

↓

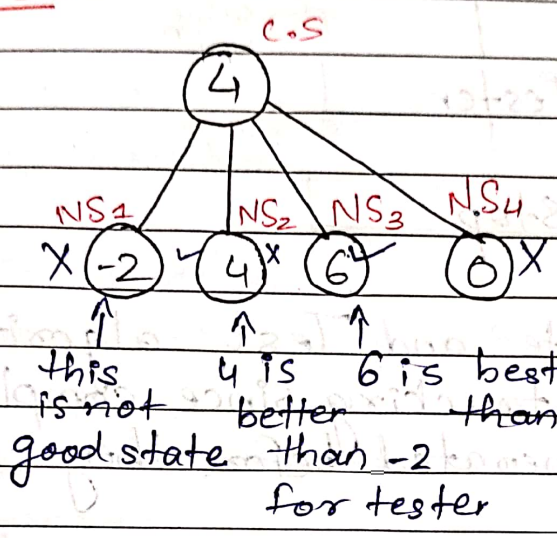
if not then again go to the generators

Tester will show we are on right path or not by saying yes or no.

→ from current state (c.s.) generator will generate the next state (n.s) then the tester will check whether the c.s is better or n.s. is better to each other.

→ I.S → C.S → N.S

example:



Now, 6 will become C.S.

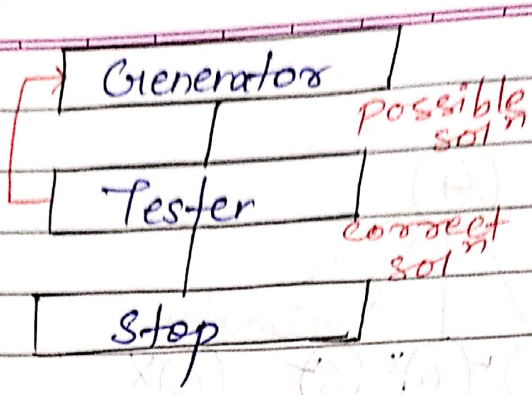
Generate and Test:

Generate and Test search algorithm is a very simple algorithm that gaurantees to find a solution if done systematically there exists a solution.

Generate - Test is applying method of searching goal state even at initial state. It consists of two function

1) Generator 2) Tester

Generator Fun^n generates a possible next node while the tester fun^n compares the next state with acceptable goal state and report with yes/no to indicate wheather the goal state is reached.



- The generate and Test algorithm is DFS with back tracking since complete solution must be generated before they can be tested.
- In its most systematic form it is simply an exhausting search of the problem space.
- Generate and Test, operates by generating solution randomly but then there is no gaurantee that a solution will ever be found. It is also known as British museum algorithm, a reference to a method for finding a object in a british museum by wondering randomly but some path are not considered because these seem unlikely to have to a solution. This evolution is performed by heuristic function.

Characteristics:

- It takes less time as next state are generated randomly.
- It takes lot of time if a systematic generater tester is use with DFS.
- DFS gauarantees that a solution would be found quickly.
- Random search didn't consider those paths which seen unlikely to have a solution.
- The evolution by tester function is performed using hueristic function.

Hill Climbing

→ In Hill Climbing if the test fun^{n} is augmented with a heuristic function that provides an estimate of how close a given state is a goal state.

→ Hill Climbing is often used when a good heuristic function is available for evaluating states but when no other useful knowledge is available.

e.g.

suppose you are unfamiliar city without a map and you want to get downtown. You simply aim for tall buildings.

Hill Climbing Technique:

1) Simple hill climbing.

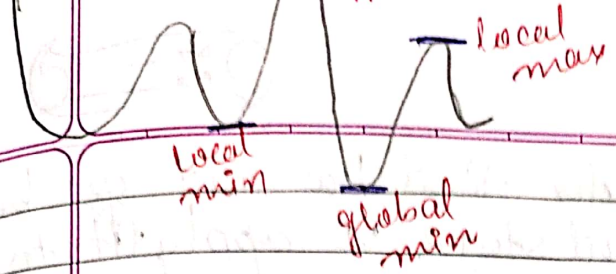
2) Steepest Ascent hill climbing.

Introduction of Hill-Climbing:

→ Hill Climbing algorithm is a local search algorithm which continuously moves in the direction of increasing value to find the peak of the mountain or best solution of the problem.

→ It terminates when it reaches a peak value where no neighbour has a higher value.

(The neighbour (next move) value should be best when we move ahead)



→ It is also called greedy local search as it only looks to its good imediate neighbour state and not beyond that.

→ Hill climbing is mostly use when a good heuristic is available and when no other useful knowledge is available.

$$Eva = T + G.H$$

↑ evaluation ↑ Tester ↑ good heuristic function

Features:

→ less time consuming
→ no back tracking
→ less optimal sol^n & sol^n is not gauranteed.

Algorithm: Simple Hill Climbing:

1) Evaluate the initial state. If it is also a goal state, then return it and quit. Otherwise, continue with the initial state as the current state.

2) Loop until a solution is found or until there are no new operators left to be applied in the current state:

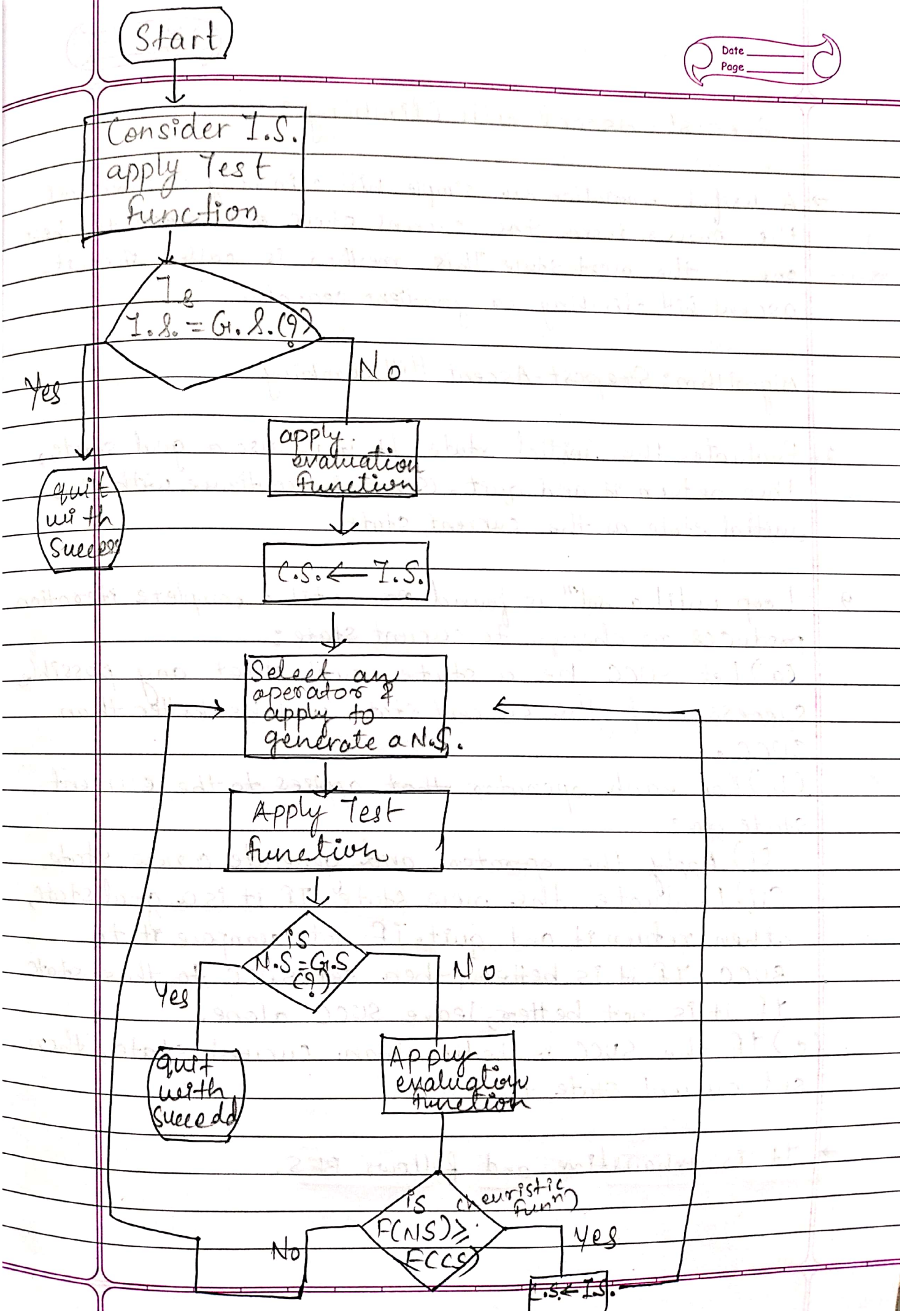
(a) Select an operator that has not yet been applied to the current state & apply it to produce a new state.

(b) Evaluate the new state

(i) If it is a goal state, then return it & quit.

(ii) If it is not a goal state but it is better than the current state, then make it the current state.

(iii) If it is not better than the current state, then continue in the loop.


Start
Consider I.S. apply Test function
Is I.S. = G.S. (?)
Yes
No
quit with Succes
apply evaluation function
C.S. ← I.S.
Select any operator & apply to generate a N.S.
Apply Test function
Is N.S = G.S (?)
Yes
No
quit with succedd
Apply evaluation function
Is (heuristic fun^n) F(NS) ≥ F(CS)
No
Yes
C.S. ← I.S.

Steepest-Ascent Hill Climbing:

→ A useful variation on simple hill climbing considers all the moves from the current state and selects the best one as the next state. This method is called steepest-ascent hill climbing or gradient search.

Algorithm: Steepest-Ascent Hill Climbing

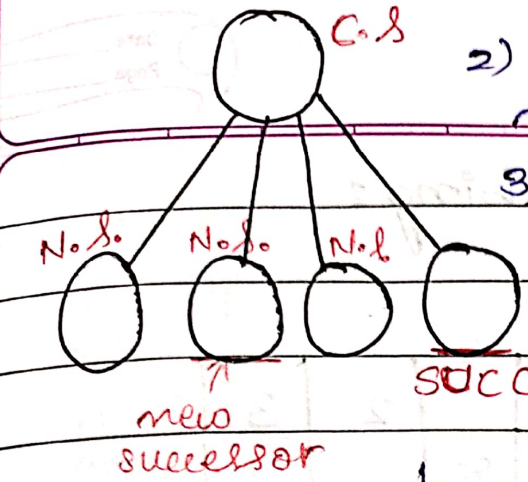
1. Evaluate the initial state. If it is also a goal state, then return it and quit. Otherwise, continue with the initial state as the current state.

2. Loop until a soln is found or until a complete iteration produces no change to current state:
   (a) Let SUCC be a state such that any possible successor of the current state will be better than SUCC.
   (b) For each operator that applies to the current state do:
      (i) Apply the operator and generate a new state.
      (ii) Evaluate the new state. If it is a goal state, then return it and quit. If not, compare it to SUCC. If it is better, then set SUCC to this state. If it is not better, leave SUCC alone.
   (c) If the SUCC is better than current state, then set current state to SUCC.

→ It is exhaustive and follows BFS.

1) In C.S we apply operator and the worst value (best move) is called successor.

2) Then the successor compare ~~with self~~ from next state.



3) Then we will check whether the N.S. have best value then successor.

→ If any N.S. is better than successor then that N.S will become new successor and that will compare with C.S.

→ And at last the new successor will become the C.S.

→ Iteration and current state change hona thatam the search thatam.

Example of Simple Hill Climbing:

I.S.

2	8	3
1	6	4
7	5	

→

G.S.

1	2	3
8		4
7	6	5

Here I.S. = C.S.

h(I.S.) = 6

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

Check F(CS) ≤ F(NS)

(8)

2	8	3
1	6	4
7		5

2	8	3
1	6	
7	5	4

(6)

(5)

2	8	3
1		4
7	6	5

(6)

2	8	3
1	6	4
	7	5

(6)

2	8	3
1	6	4
7	5	

C.S of heuristic is less then above C.S

(3)

2		3
1	8	4
7	6	5

(5)

2	8	3
	1	4
7	6	5

(5)

2	8	3
1	4	
7	6	5

(5)

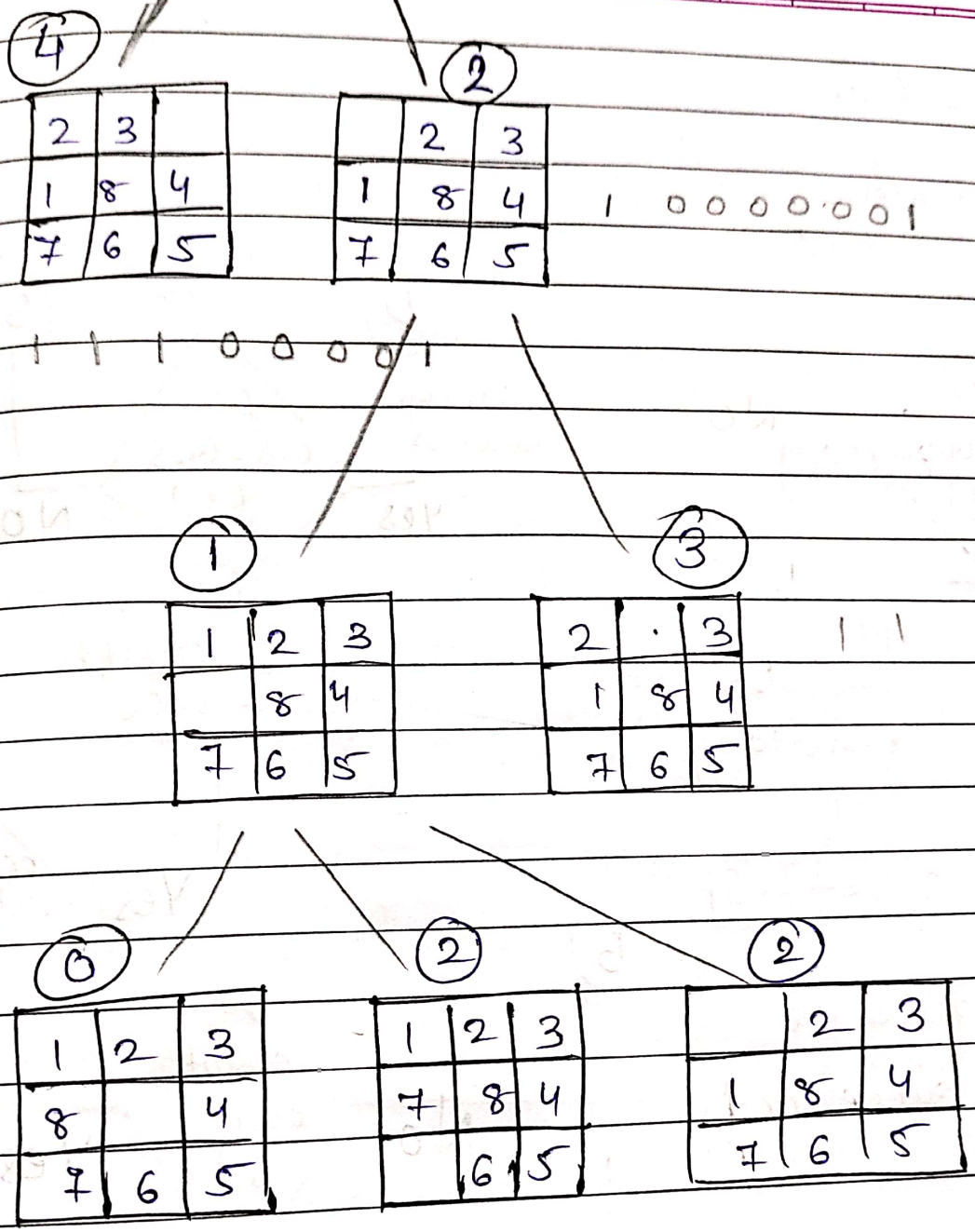
2	8	3
1	6	4
7		5

2 1 0 0 0 0 0 2

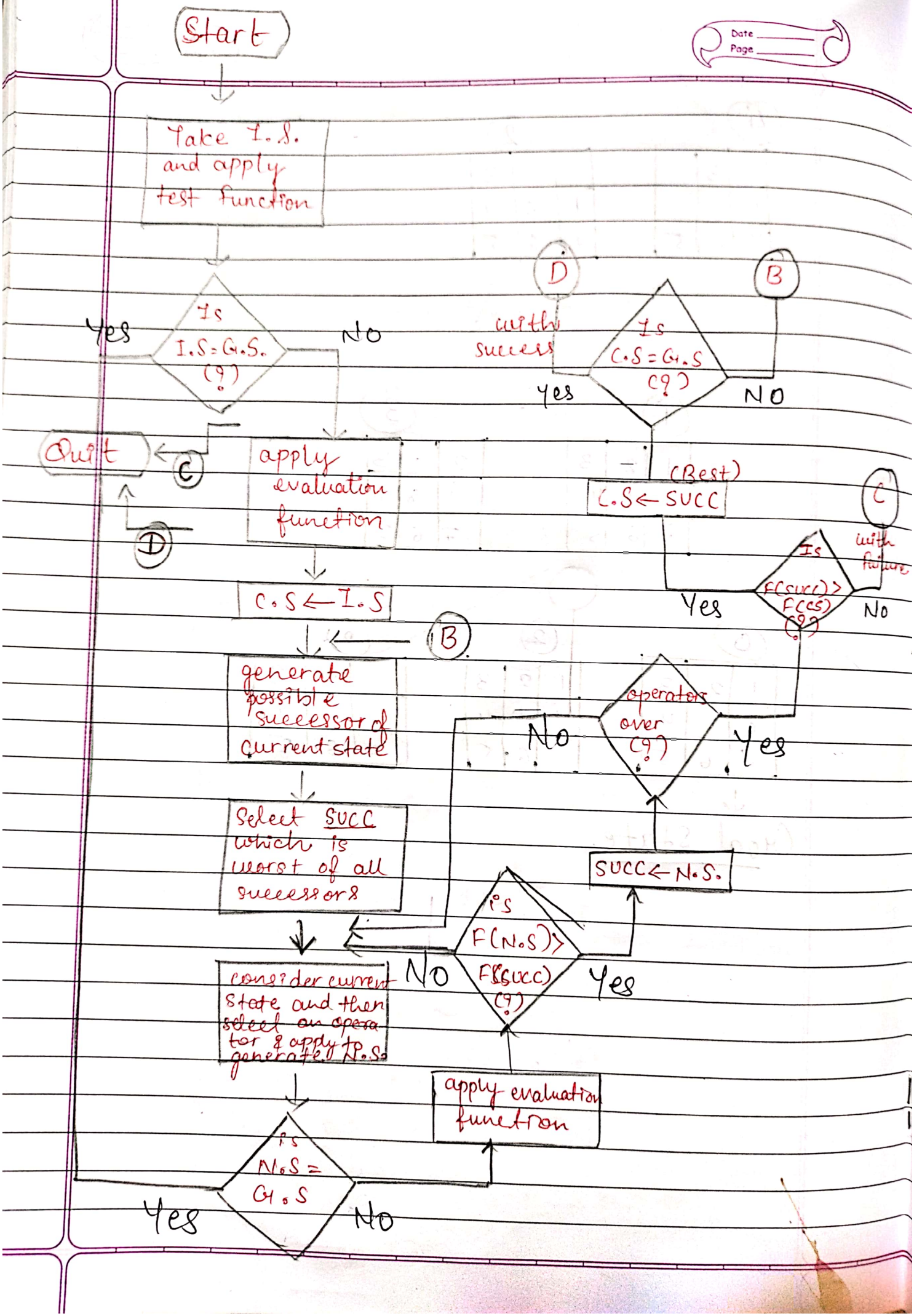
1 1 0 0 0 0 0 1

1 1 0 1 0 0 0 2

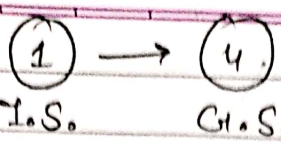
1 1 0 0 0 1 0 2


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

Goal State

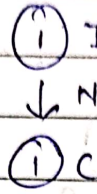

Start
Take I.S. and apply test function
Is I.S = G.S. (?)
Yes
No
Quit
C
D
apply evaluation function
C.S ← I.S
B
generate possible successor of current state
Select SUCC which is worst of all successors
consider current state and then select an operator & apply to C.S. generate N.S.
Is N.S = G.S
Yes
No
apply evaluation function
Is F(N.S) > F(SUCC) (?)
Yes
No
SUCC ← N.S.
operators over (?)
No
Yes
D
with success
Is C.S = G.S (?)
Yes
No
B
(Best)
C.S ← SUCC
C
with failure
Is F(SUCC) > F(CS) (?)
Yes
No

example:

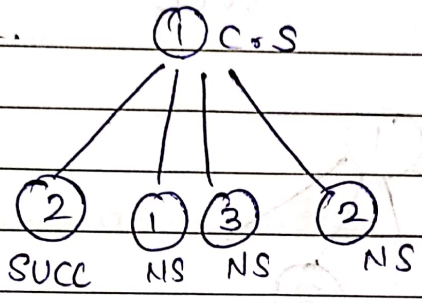


steepest ascent →

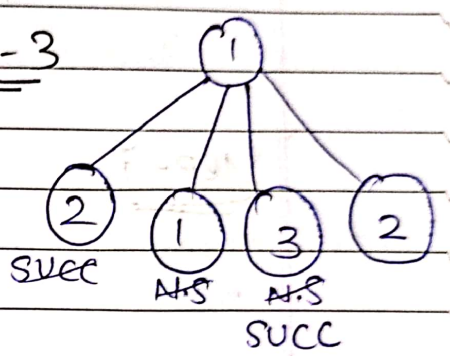
Step-1.



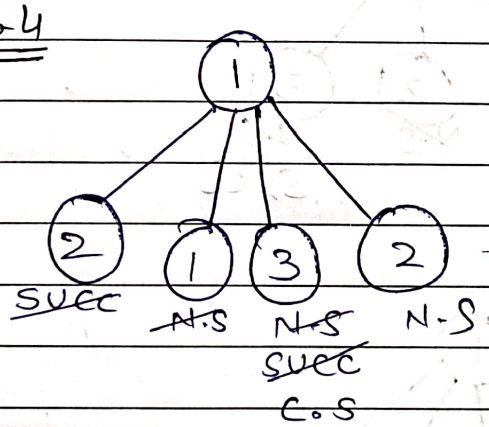
Step-2.



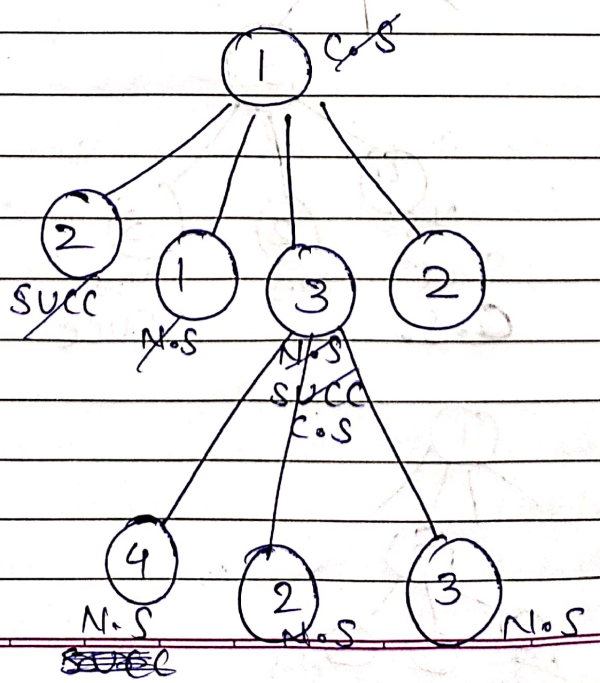
Step-3



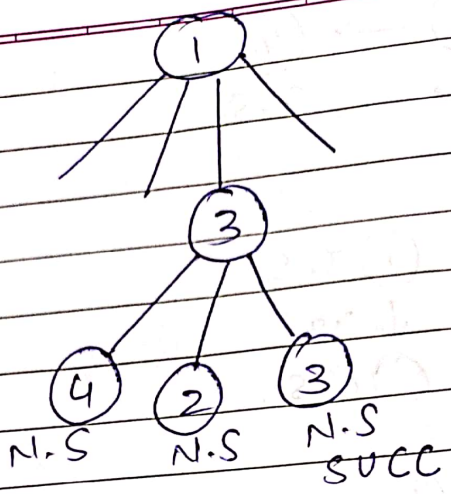
Step-4



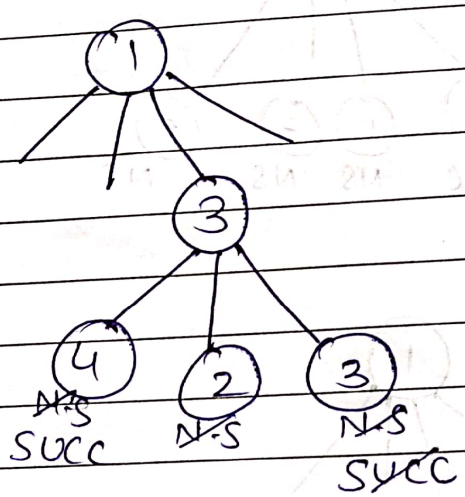
Step-5



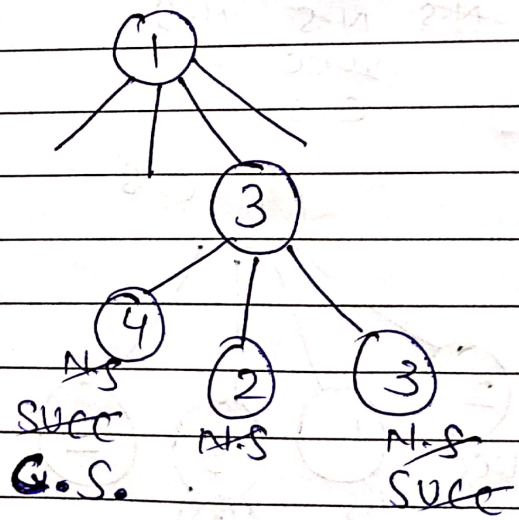
Step-6.



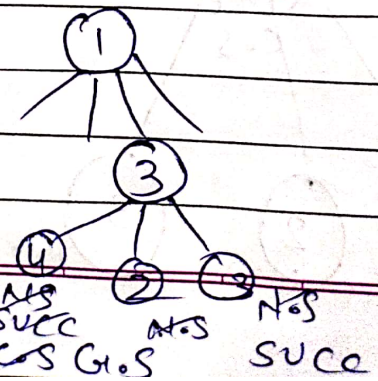
Step-7.



Step-8.



Step-9.



- It is a variation of simple Hill Climbing technique.

- It consider all the moves from the current state instead of moving to the first state i.e. better, move to the best possible state i.e. one move away.

- It donot just climb to a better state but climbing up the steepest slope.

- It has elements of the breadth first algorithm.

Note: Steepest ascent differs from the basic Hill Climbing algorithm by choosing the best successor rather than the first successor i.e. better.

Common Issue:

- It cannot backtrack to it parent node.

- If it gets track in the local maxima/minima, then nothing can help it to get out of that situation.

- A plateau is a flat area of the search space in which a whole set of neighboring states have the same value. On a plateau, it is not possible to determine the best direction in which to move by making local comparisons. (straigh area)

- A ridge is a special kind of local maximum. It is an area of the search space that is higher than surrounding areas and that itself has a slope (which one would like to climb). But the orientation of the high region, compared to the set of available moves and the directions in which they move, makes it impossible to traverse a ridge by single moves.

(1) A state that is better than all of its neighbours but not then some other states far away.

Soln: backtracking technique can be solution of local maxima or minima.

(2) plateau: A flat area of the search space in which all neighbouring states have the same value for that algorithm does not find any best direction to move.

Soln: The solution is to take big steps or very little step while searching to solve problem.

(3) ridge: A small small peaks:
The orientation of high region compare to the sets of available move make it impossible to climb up.

Soln: at the same time move in all directions and check which is better next state or which state is better.

Block world problem:

Points to remember:
→ Add one point for every block i.e. resting on the thing it is suppose to be resting on it.
→ Subtract one point for every block i.e. sitting on the wrong thing.

Rules
→ not more than three blocks on ground.
→ at a time one block can move.

G.S:

H
G
F
E
D
C
B
A

h(GS) = 8

I.S:

A	-1
H	+1
G	+1
F	+1
E	+1
D	+1
C	+1
B	-1

h(IS) = 4

+1	H
+1	G
+1	F
+1	E
+1	D
+1	C
-1	B

A	+1

h(NS) = 6

There are total 3 possibility for next state but all three are not better move.

So, we will quit the game with failure.

* → optimal solution

use in

A* Algorithm:

- A* is a corner stone name of many AI system.
- It is based on Best first Search algorithm.
- A* search finds the shortest path through a search space to a goal state using heuristic function.
- This technique finds minimal cost solutions to reach to a goal state.
- In A* the symbol * represent optimality
- A* requires heuristic function to evaluate the cost of path that passes through the particular state.

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

where,

$g(n)$ = actual cost to reach n.

$h'(n)$ = estimate of cost to reach goal from n.

$f'(n)$ = estimate of total cost along path thrugh n.

Best First Search: DFS & BFS

[most promising a model successor]

Algorithm:

1. Start with OPEN containing just the initial state
2. Until a goal is found or there are no nodes left on OPEN do:
   (a) Pick the best node on OPEN.
   (b) Generate its successors.
   (c) For each successor do:
      (i) If it has not been generated before, evaluate it, add it to OPEN, and record its parent.
      (ii) If it has been generated before, change the parent if this new path is better than the previous one. In that case, update the cost of getting to this node and to any successors that this node may already have.

OR graph → multiple path to solve problem but we want optimal
→ maintain list
→ easily track by parent node

$f' = g + h'$

where

g = cost of getting from the initial state to the current node.

h' = an estimate of the additional cost of getting from the current node to a goal state.

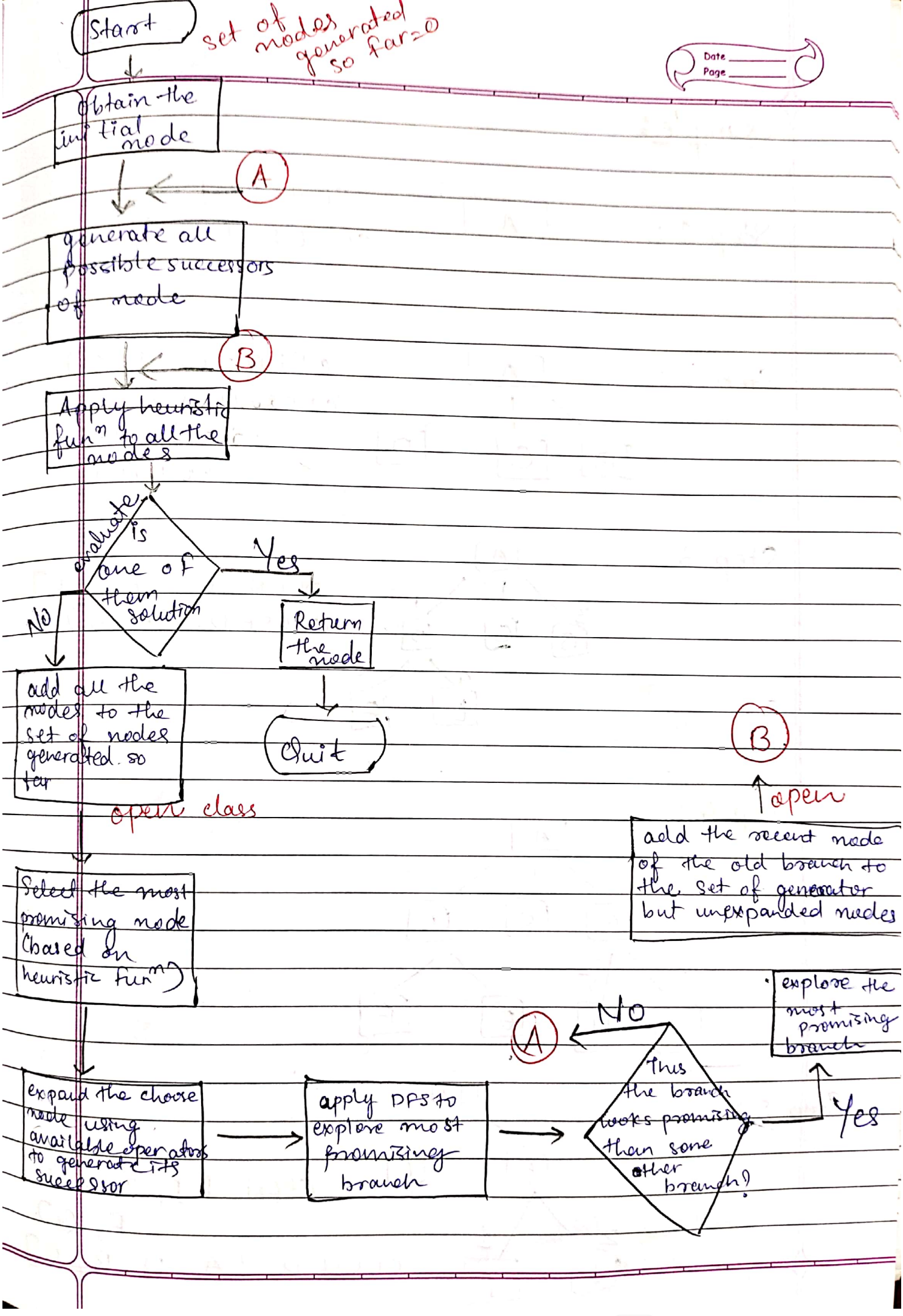
We need two list of nodes: list that through out problem we have to use

OPEN:

nodes that have been generated and have the heuristic function applied to them but which have not yet been examine.

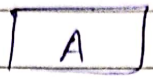
CLOSED:

nodes that have already been examine.


Start
set of nodes generated so far=0
Obtain the initial node
A
generate all possible successors of node
B
Apply heuristic fun^n to all the nodes
evaluate is one of them solution
Yes
NO
Return the node
Quit
add all the nodes to the set of nodes generated so far
open class
Select the most promising node (based on heuristic fun^n)
expand the choose node using available operators to generate its successor
apply DFS to explore most promising branch
A
NO
Thus the branch looks promising than some other branch?
Yes
explore the most promising branch
B
open
add the recent node of the old branch to the set of generator but unexpanded nodes

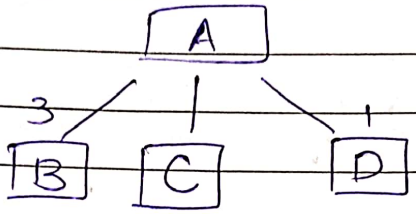
example:

Step 1:



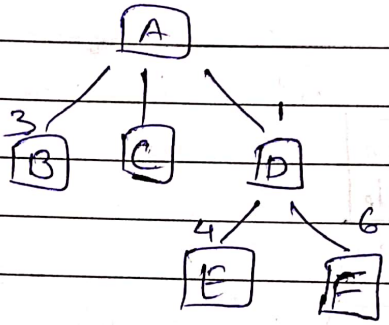
OPEN = [A]
CLOSED = [ ]

Step 2:



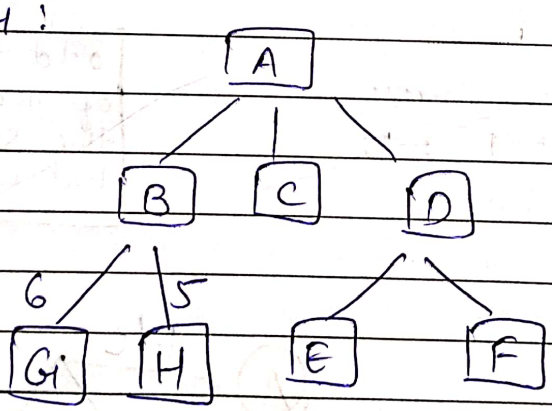
OPEN = [D B C]
CLOSED = [A]

Step 3:

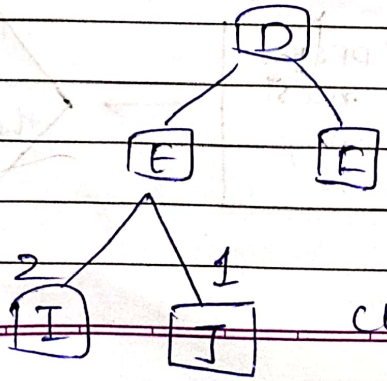


OPEN = [B E C F]
CLOSED = [A D]

Step 4:



Step 5:

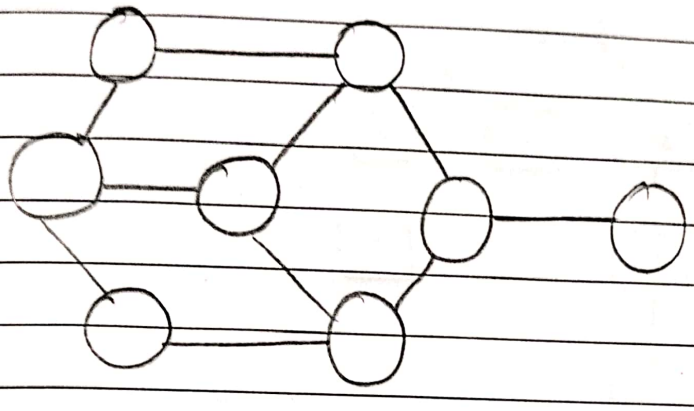


OPEN = [J, I, C, H, F, G]
CLOSED = [A, D, B, E]

A* Algorithm:

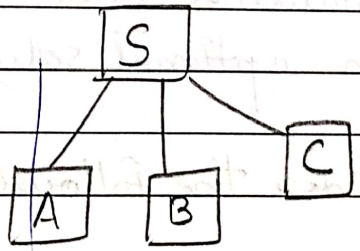
h(n) is the heuristic value (estimate value)
g(n) is the distance between two node lie (actual value)

example:

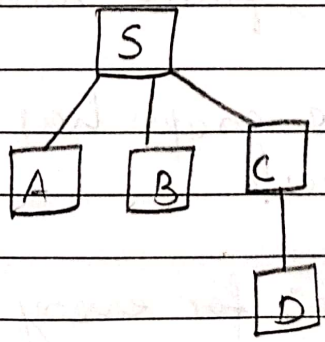


1. 

Queue
S

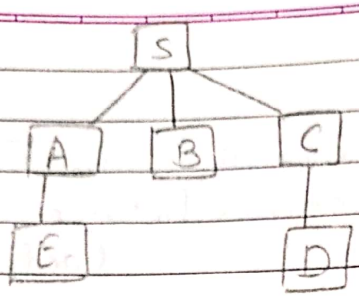
2. 

Queue
SC
SA
SB

3. 

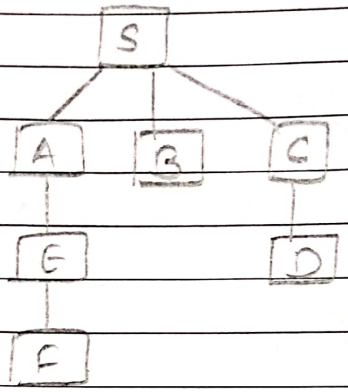
Queue
SA
[redacted]
[redacted]
SCD
SB

4.



Queue
SAE
SCD
SB

5.



A* search properties.

admissible→ The algorithm is A* is admissible, this means that provided a solution exist the 1st solution found by A* is an optimal solution

A* is admissible under the following conditions:

- In the state space graph
  1) every node has a finite number of successor.
  2) every arc in the graph has a cost greater than some $\epsilon > 0$.
  3) heuristic function: for every node small n, $h^*(n) \leq h(n)$.

Complete A^{**} is also complete under the above condition.

Knowledge Representation

Data
↓
Info
↓
Knowo
↓
Wisdom

1) Propositional / propositional logic

Characteristic

→ simple to deal
→ It is able to represent "some" real world facts (Dynamic)
→ always written in capital letter.

e.g. This is a man = MAN
The sun is rising = SUNRISING
Man is mortal = MORTALMAN

Advantages

→ It is easy to represent & understand as well
→ It require less memory.

Disadvantages

→ It is very hard to distinguish between name and characteristic.

e.g.
Virat Kohli is a crickter = CRICKTER
VIRAT KOHLI

→ Fails to capture the relationship of the real world facts.

Solution: To predicate logic of 1st order predicate logic - it can represent real world facts is the relationship between the entity and other thing.

e.g. The sun is rising

(Rising) (Sun)

↑ Predicate ↑ argument

e.g. Man is mortal → Mortal (man)

e.g. Virat Kohli is a cricketer.

→ cricketer (virat kohli)

e.g. Sun is rising in the east.

→ rising (sun, east)

e.g. Viral Kohli is left handed cricketer.

→ cricketer (virat kohli, left handed)

(1) Today is a sunny day = Today (sunny day)

(2) I like robotics = Likes (x, robotics)

(3) John is brother of Alex = brother (John, alex)

(4) John loves to play guitar = love-to-play (John, guitar)

(5) I scored 11 marks in AI = scored-ai (x, 11)

By using symbol → (Quantifiers). to quantify or calculate

- Universal (apply to everyone / everybody) symbol forall (∀)
- existantial symbol = ∃ (there exists)

other symbol:

- "→" → if then condition (implication)
- "∧" → and
- "∨" → or
- "¬" → not Inegative (not like)

1) All man are mortal =
$\forall x : man(x) \rightarrow mortal(x)$

2) All students Likes AI =
$\forall x : student(x) \wedge likes(x, ai)$
↓
represent facts not a condition

3) Ram likes to eat all kind of food.
$\forall x : food(x) \wedge likes\ to\ eat(ram, x)$

If all kind of food
then $\forall x : food(x) \wedge likes(ram, x)$

4) All yellow mushrooms are poisonous.
$\forall x$: mushroom $(x) \wedge$ colour $(x,$ yellow$) \rightarrow$ poisonous.

5) When it started raining all student carry umbrella.
$\forall x$: student $(x) \wedge$ start (raining) $\rightarrow$ carry $(x,$ umbrella).

6) Some purple mushrooms are poisonous.
$\exists x$: mushroom $(x) \wedge$ colour $(x,$ purple$) \rightarrow$ poisonous (x)

7) Not all students have AI book.
$\exists x$: student $(x) \wedge$ has $(x,$ ai_book)

8) Some students do not have AI book.
$\exists x$: student $(x) \wedge \neg$ has $(x,$ ai_book).

9) Mohan eats peanuts and goes to school.
eats (mohan, peanuts) $\wedge$ go to (mohan, school).
eats (mohan, peanuts) $\wedge$ go_to_school (mohan)

10) Venus is near sun but does not have tail
near (venus, sun) $\wedge \neg$ has (venus, tail)

11) Anything anyone watches in multiplex is a movie.
$\forall x : \forall y$: person $(x) \wedge$ thing $(y) \wedge$ watch_in_multiplex $(x, y) \rightarrow$ movie (y)

12) Alex watches terminator in multiplex

watches_in_multiplex(x, y)
alex, terminator

(1) Hardik Pandya is an all rounder cricketer who stays at Baroda.

→ cricketer(hardik pandya) ∧ all rounder(hardik Pandya) ∧ stay(hardik pandya, baroda)

(2) Everyone is loyal to someone.

→

(3) Laxman eats everything that ram eats

→ ∀x: food(x) ∧ eat(ram, x) → eat(lax, x)

(4) Anything anyone eats and is not killed by a food.

→ ∀x: ∀y: person(x) ∧ thing(y) ∧ eat(x, y) ∧ ¬ is_not_killed_by(x, y) → food(y)

(5) One who knows riding does not know skatting.

→ $\forall x$: person $(x) \wedge$ knows $(x$, riding$) \rightarrow \neg$ knows $(x$, skatting$)$

(1) Jack went to school and brought maggie in lunch box.

→ go to (Jack, school) $\wedge$ brought maggie (Jack, lunch box)

(2) While laughing sue fell in swimming pool and does not know swimming.

→ laughing (sue) $\wedge$ fell in swimming pool (sue) $:\wedge \neg$ knows (sue, swimming)

(3) All those who did not prepare for the test are absent in the class.

→ All x (person) $\wedge \neg$ prepare (x, test) $\rightarrow$ absent (x, class)

(4)

Story

(1) Ram was mighty a king.
→ king (Ram, mighty).

(2) Ram has a crown on his head.
→ has_crown (Ram, head)

(3) The crown has beautiful multicolour gems on it.
(multicolour, gem) ∧ (beautiful, gem) · (gem, on_a_crown)

(4) All courtiers are attracted towards ram's crown.
→ All (courtiers) ∀x : courtiers (x) ∧ attracted_towards (x, crown)

(5) Ram slept in night and next morning found his crown missing.
→ a : (slept, night) ∧ time (next_morning) ∧ ¬ find (

(6) Ram's instruct the chief minister to investigate
→ instruct (Ram, chief minister) ∧ chief_minister (investigate)

(7) The chief minister suspect three courtiers Rohan, Shyam and Mohan.
→ courtiers (Ram) ∧ courtiers (Shyam) ∧ courtiers (Mahan) ∧ suspect (chief minister, Ram) ∧ suspect (chief minister, Shyam) ∧ suspect (chief minister, Mohan)

Story Time

Date
Page

(1) If it croaks and eats flies it is a flag.
→ $\forall x$: frog $\wedge$ croaks $\wedge$ x : (eats, flies) → (x, frog).

(2) If it chirps and sings then it is canary
→ $\forall y$: canary : chirps $\wedge$ sings → (canary) : y.

(3) A frog is green in colour.
→ $\forall x$: frog → : (green, colour)

(4) Canary is yellow in colour.
→ x : canary → x : (yellow, colour)

(5) Fritz croaks and eats flies.
→ croaks (fritz) $\wedge$ fritz (eats, files).

Que: Find out what is colour of fritz.

Green.

$\exists$

colour (Fritz, green)

↑ 3 substitue.

frog (fritz)

↑ 1 substitue

croak (fritz, $\wedge$ eats (fritz, files)

↑ 5

NIL

(1) Steve only like easy courses.

→ $\forall x$: course $(x) \wedge$ easy $(x) \rightarrow$ likes (steve, x)

(2) Science course are hard.

→ $\forall x$: course $(x) \wedge$ department $(x,$ science$) \rightarrow$ hard (x)

(3) All courses in basket weavily department are easy

→ $\forall x$: course $(x) \wedge$ department $(x,$ bw$) \rightarrow$ easy (x)

(4) BK301 is a basket weaving course

→ course (BK301) $\wedge$ department (BK301, bw)

Que : Which course steve likes ?

likes (steve, BK301)

↑ (1, sub)

course (BK301) $\wedge$ easy (BK301)

↑ 4a

easy (BK301)

↑ (3, sub)

course (BK301) $\wedge$ dep (BK301, bw)

↑ 4 sentence

NIL

Story

(1) Marcus was a man.
→ man(marcus)

(2) Marcus was a pompeian.
→ pompeian(marcus)

(3) All pompeian was roman.
→ ∀x : pompeian → roman(x)

(4) Ceasar was a ruler.
→ ruler(ceasar)

(5) All romans was either loyal to ceasar or hated him.
→ ∀x : romans(x) → loyal_to ∨ (x) hate : ceasar

(6) Everyone is loyal to someone
→ ∀x : ∃ y : p(x) ∧ p(y) ∧ loyal_to(x,y)

(7) People only try to a assassinate rulers they are not loyal to.
→ ∀x : ∀y : assisnate(x,y) : ¬ loyal_to(x,y)

(8) marcus tried to assisnate ceasar.
→ assisnate : (marcus, ceasar).

(9) Newly added fact
All man are person
∀x : man(x) → person(x).

Que: Was marcus loyal to ceasar.

person(marcus) ∧ ruler(ceasar) ∧ assinate(marcus, ceasar) ⇒ loyal_to(marcus, ceasar)

↑ (7, sub)

person(marcus) ∧ rules(ceasar) ∧ assassinate(marcus, ceasar)

↑ (9, sub)

man(marcus) ∧ ruler(ceasar) ∧ assassinate(marcus, ceasar)

↑ 8

man(marcus) ∧ ruler(ceasar)

↑ 4

man(marcus)

↑ 1

NIL.

(1) John likes all kind of food.
→ $\forall x : Food(x) \rightarrow Likes(John, x)$

(2) Apple are food..
→ food (apple)

(3) Anything anyone eats and is not killed by a food.
→ $\forall x : \forall y : thing(x) \wedge person(y) \wedge eats(y, x) \wedge \neg killed\_by(y, x) \rightarrow food(x)$

(4) Bill eats peanuts and is still alive
→ eats(bill, peanuts) ∧ alive(bill).
4a) eat(bill, p), 4b) alive(bill)

(5) Sue eats everything that bill eats.
→ $\forall x : Food(x) \wedge eats(bill, x) \rightarrow eats(sue, x)$

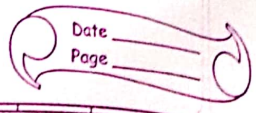
Q. Prove that John likes peanuts.

Newly added facts

(6) bill is a person
→ person(bill)

(7) Peanuts is a thing
→ thing(peanuts)

(8) All who are alive are not killed by anything
→ $\forall x : \forall y : person(x) \wedge thing(y) \wedge alive(x) \rightarrow \neg killed\_by(x, y)$



likes(john, peanut)

↑ (1, substitue)

food(p)
↳ peanut

↑ (3, substitue)

person(bill) ∧ thing(p) ∧ eats(b, p) ∧ ¬ killed_by(b, p)

↑ 4(a)

person(bill) ∧ thing(peanut) ∧ ¬ killed_by(bill, peanut)

↑ (7)

person(bill) ∧ ¬ killed_by(bill, p) :
↑(6) ¬killed_by(bill, p)

↑ (8, substitue)

person(bill) ∧ thing(peanut) ∧ eat(bill, peanut) ∧ alive(bill)

↑ (4)

person(bill) ∧ thing(peanut)

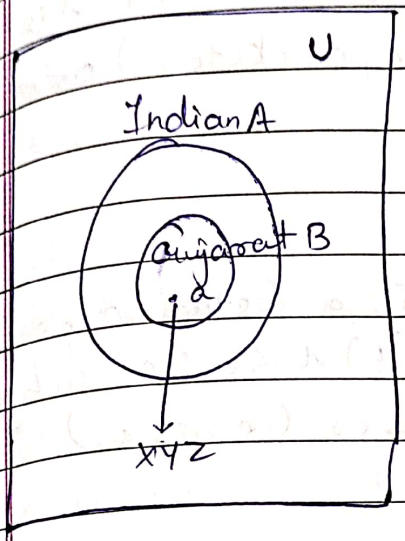
↑ (7)

person(bill)

↑ (6)

NIL

Concept Instance and B-a relationship



→ BCA

→ B Ba A

* It subset of class means = is a relationship

{ a instance B
a element of class

* It element of class means = instance

e.g.

① ∀x : cat(x) → animal(x).
Ba(cat, animal)

② animal(lucky)
instance(lucky, animal)

③ cat(lucky)
instance(lucky, cat)

④ Prasant B a singer
instance(Prasant, singer)

⑤ prasant is a classical singer.
instance(prasant, classical singer) ∧ Ba(classical singer, singer)

(6) Rohit sharma is a cricketer

instance (Rohit sharma, all rounder cricker) ∧ is a (all rounder, cricketer) ~~cricketer~~ cricketer

(7) I love A I.

instance (eric, student) ∧ is a (student, person) ∧ instance (ai, C.S subject) ∧ is a (cs subject, subject) ∧ loves (eric, ai).

(8) marcus was a man

instance (marcus, man) ∧ is a (man, person)

(9)

Declara

→ consist of
objects and
e.g. sun, e

→ It is also k
descriptive k

→ In declarat
knowing al
something.

→ It is expres
declarative.

→ e.g.
I like to

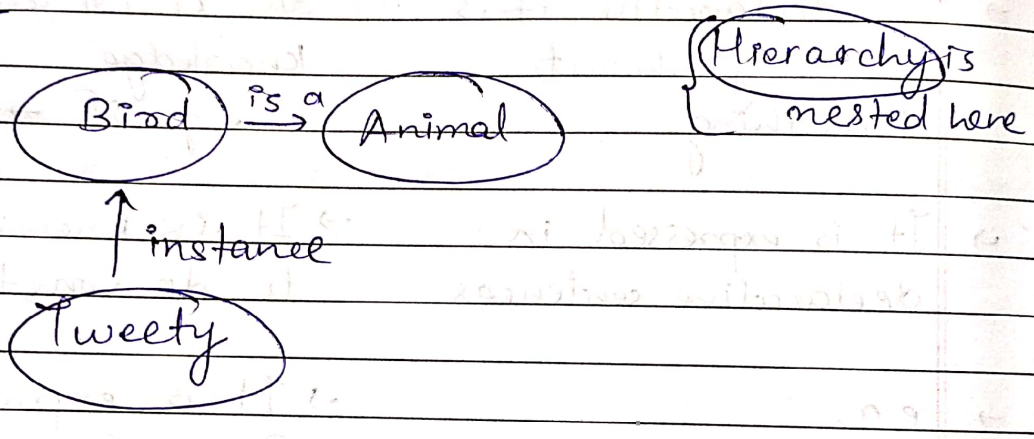
Declarative	Procedural
→ consist of concepts, objects and facts. e.g sun, earth.	→ consists of rules and strategies applied to available facts, objects and concepts. e.g if → then sun → star
→ It is also know as descriptive knowledge	
→ In declarative, it is knowing about something	→ also known as imperative knowledge e.g if → then
→ It is expressed in declarative sentences	→ It is knowing about how to do something
→ e.g I like to cooking	→ It is expressed in the form of procedural steps e.g How to cook a dish? All steps are required

Semantic Network:

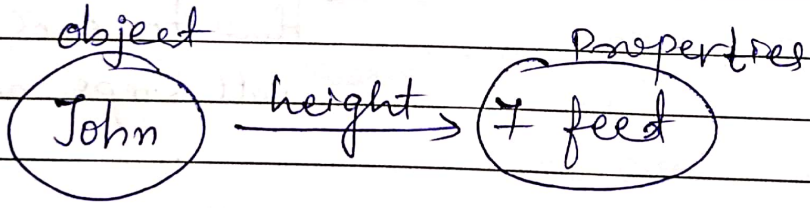
Def.n:

→ Information is represented as set of nodes. Connected to each other by a set of labled arcs which represents relationship among this nodes. This represents known as Semantic Network.

e.g



e.g Thon is 7' feet tall



e.g John is taller than Bill

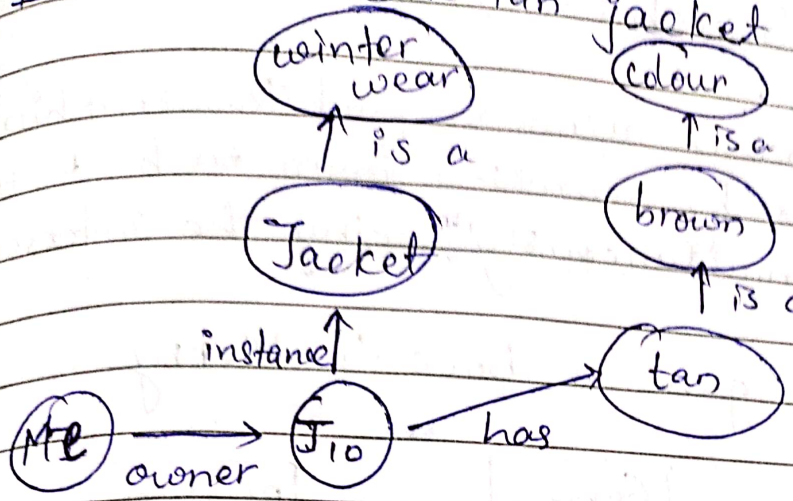
H_1 H_2

$H_1 > H_2$

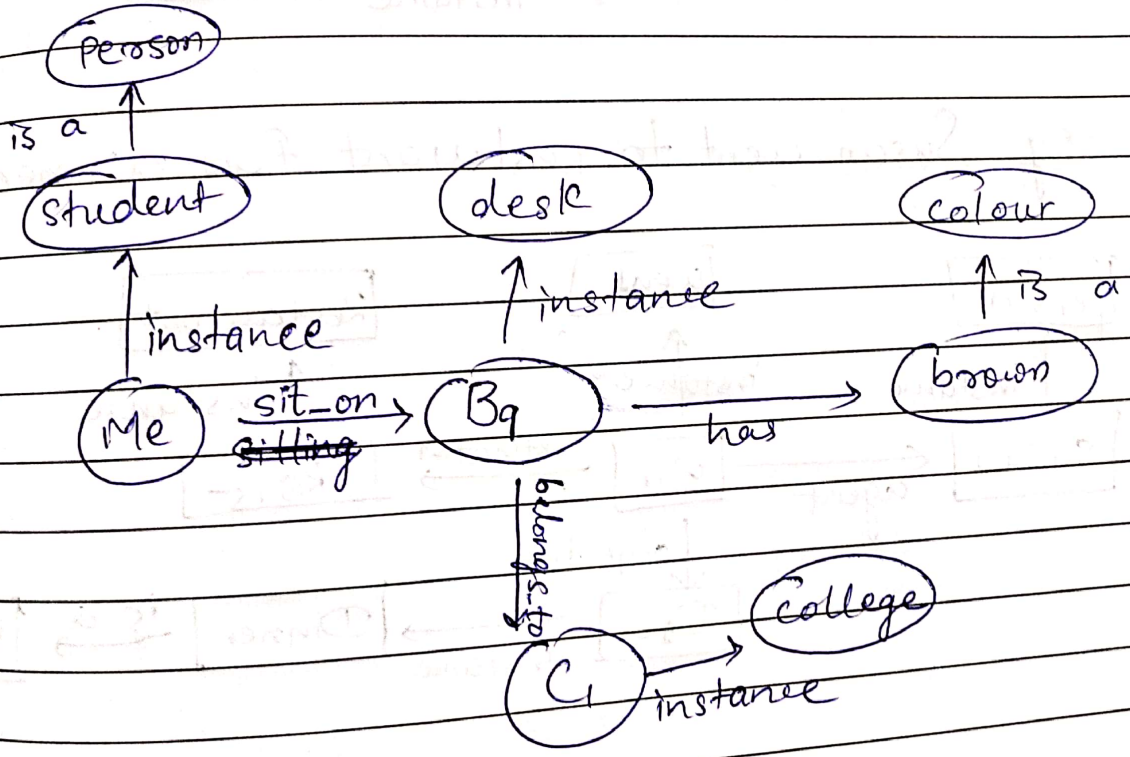
Bill ← John

h_2 h_1

I own a dark tan jacket



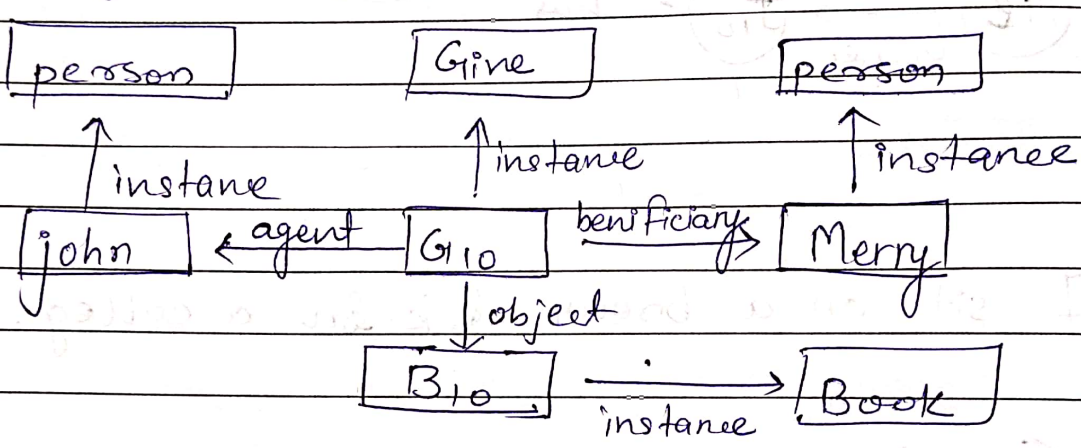
I sit on a brown desk in a college.



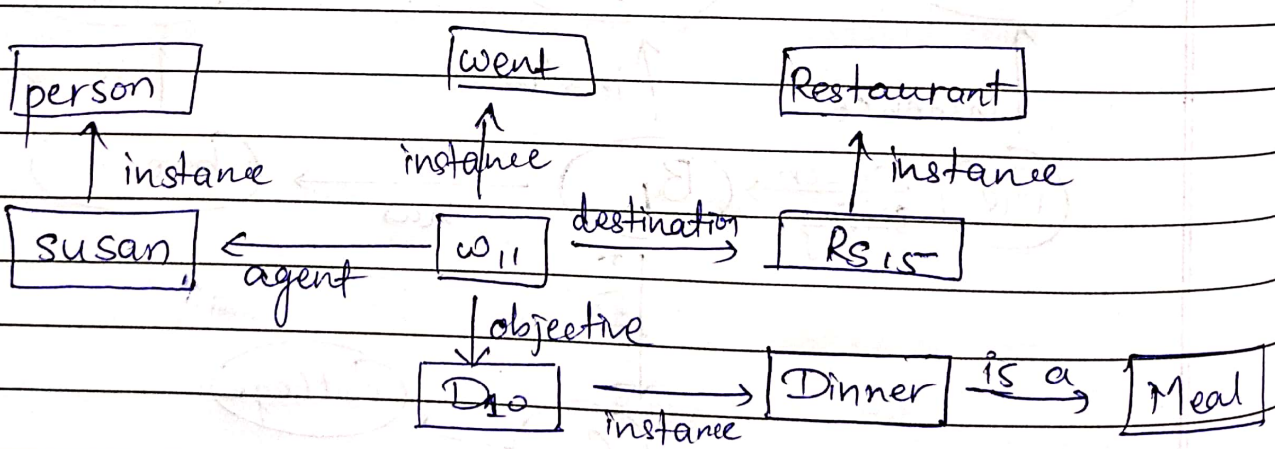
Intersection Search:

→ Intersection search allows to find Relationship among object by searching from each of the two nodes simultaneously seeing the intersection nodes.

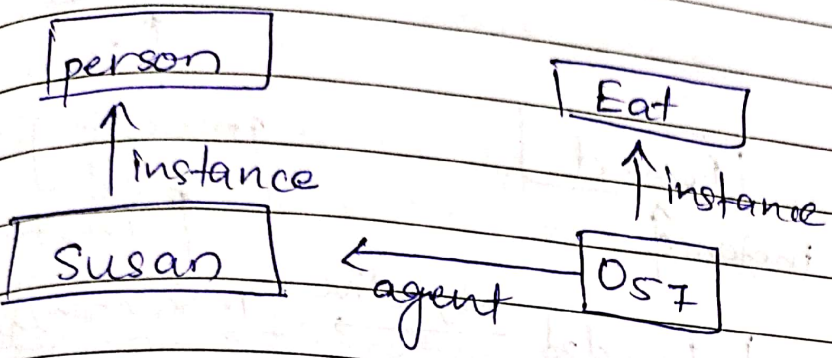
e.g. John gave a book to Merry.



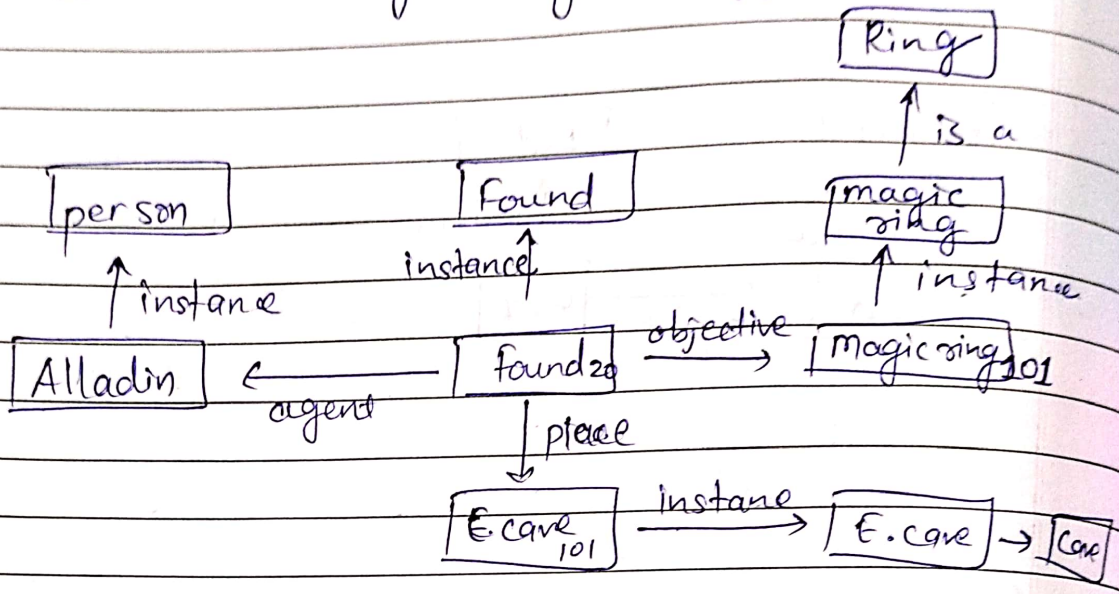
e.g. Susan went to resturant for dinner.



Susan orderdedmanchurian noodles.



Alladin found magic ring in egyptian cave.



the student give the exam. in main class

⇒ a s → nahi ayega

⇒ subset nahi banega

everyone love to eat noodles.

The → main set

every [quantifier] → subset class

a → subset

Partitioned Semantic Net

Drawbacks:

1. It is easy to visualize
2. Related knowledge can easily be ~~calculated~~ clustered using connected nodes.
3. Representation of this junction is difficult in Semantic Net.
4. Quantified expression are difficult to represent like all or some.

Advantages

→ Partition Network allow for expression to be quantified.

Partition Semantic Net

→ The basic idea behind Partitioned Semantic Net is to break the network into spaces which consists of loop of nodes and arcs considering each space as a node

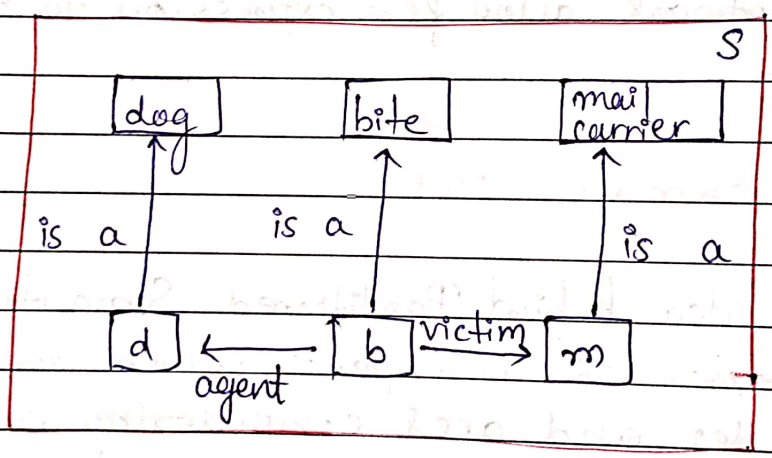
It is done as follows:

(1) Create a general statement GS.

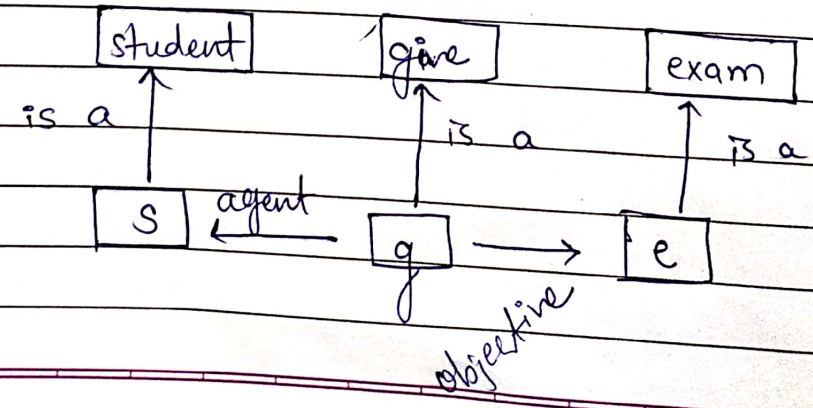
(2) Make node g an instance of GS.

(3) Every element will have two attributes
(1) Form: which states what relationship exists between g and GS.
(2) One or more ∀ and ∃ which represents quantification.

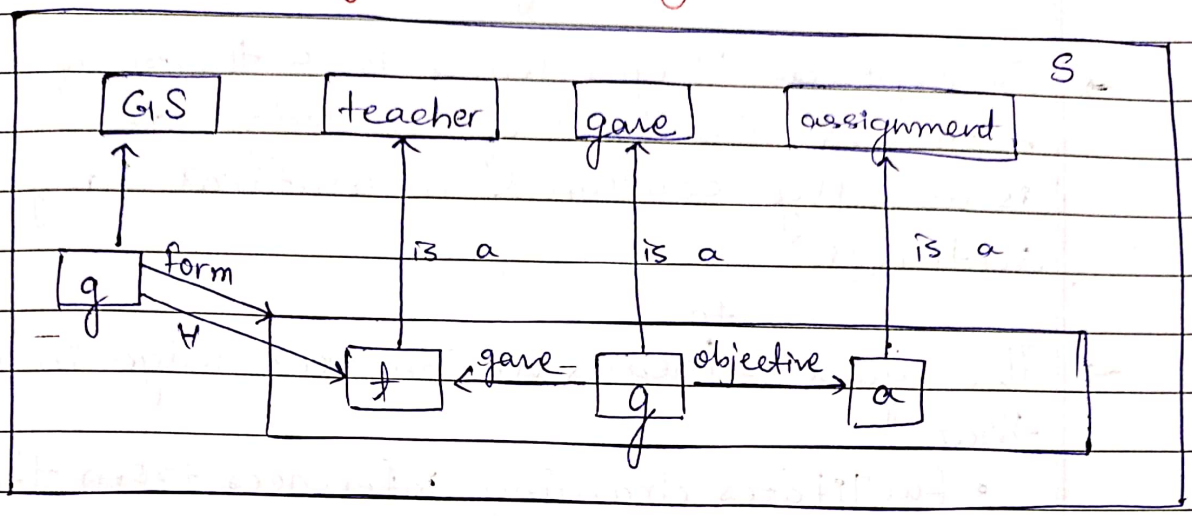
eg The dog bite the mail carriers.



eg The student give the exam.



e.g. Every teacher gave an assignment.



Conceptual Dependency [CD]

→ Conceptual Dependency is a theory of how to represent the kind of knowledge about events that is usually contained in natural language sentences.

→ The goal is to represent the knowledge in a way that
- facilitates drawing inferences from the sentences.
- Is independent of the language in which the sentences were originally stated

ATRANS → Transfer of an abstract relationship (e.g. give)
PTRANS → Transfer of the physical location of an object (e.g. go)
PROPEL → Application of physical force to an object (e.g. push)
MOVE → Movement of a body part by its owner (e.g. kick)
GRASP → Grasping of an object by an actor (e.g. clutch)
INGEST → Ingestion of an object by an animal (e.g. eat)
EXPEL → Expulsion of something from the body of an animal (e.g. cry)
MTRANS → Transfer of mental information (e.g. tell)
MBUILD → Building new information out of old (e.g. decide)
SPEAK → Production of sounds (e.g. say)
ATTEND → Focusing of a sense organ toward a stimulus (e.g. listen)

when purpose desire something then MTRANS

O → Object
P → past tense.
I → instrument

D → distance
Objective ~~means~~ purpose

Example:

(1) Ram ate apple.

Ram ⟺(P) INGEST ←(O)— Apple.

(2) Merry went to college yesterday.

Merry ⟺(P) PTRANS ←(D)— → College
yesterday ↘ Home

(3) Susan went to restaurant for dinner.

Susan ⟺(P) PTRANS ←(O)— dinner ←(O)— → restaurant
↘ office/home

(4) Susan ordered manchurian noodles.

Susan ⟺(P) MTRANS ←(O)— Manchurian noodles.

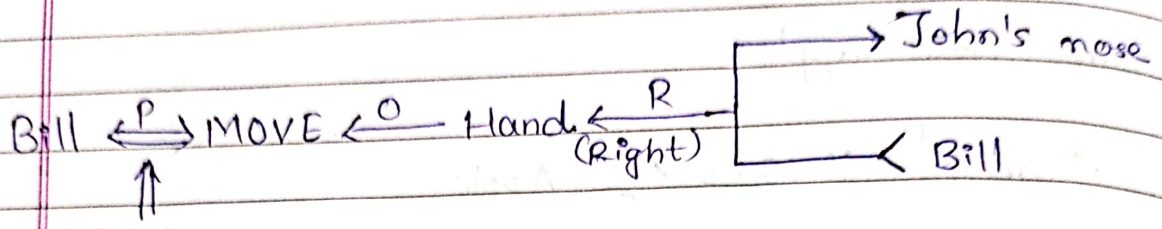
objective
eat

(5) Susan ate manchurian noodles with a fork.

Susan ⟺(P) INGEST ←(O)— manchurian noodles

Instrument (I)
fork

(6) Bill punched John with a broken nose.



3 Issue of knowledge representation:

(1) Important attributes.

(2) Relationship among attributes.

(3) Choosing granularity of representation

not fix

level of knowledge/info

(1) Important attributes

(1) instance [relationship to describe properties]

(2) Is a relationship.

e.g. cricketer is a : cricket league
batting avg : 0.362

sachin
instance : cricketer
bats : right
batting avg : 0.456
team : India
uniform colour : Blue

① Level of abstraction of inheritance.

② All sentences cannot be represented using instances and is a relationship.

③ It depends on level of abstraction you want to achieve in knowledge representation