L-3

State Space Search

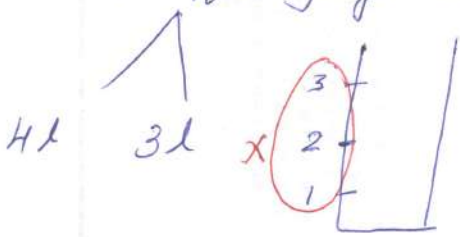
To solve an AI problem,

1. Define the problem
2. Analyze the problem
3. Represent the knowledge required to solve the problem.
4. Apply the best possible problem solving technique and find the solution.

We will define the problem using the idea of <u>state space</u>.

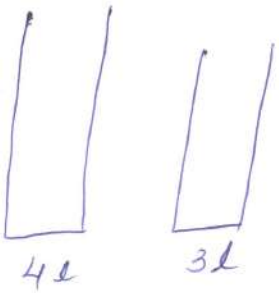
Water Jug Problem

- Two jugs w/o markers



- Pump – fill water into jugs

How do we fill exactly 2 litres of water into the 4l jug ?



(x, y) state

water in 4l jug — water in 3l jug

(3, 2)

1. Missionaries and cannibals problem
2. Towle of Hanoi
3. Monkey – banana problem

(0,0) – initial state $0 \le n \le 3$

(2, n) – final state $(n \ge 0, n \le 3)$ / goal state

(x, y) – $0 \le x \le 4$, $0 \le y \le 3$

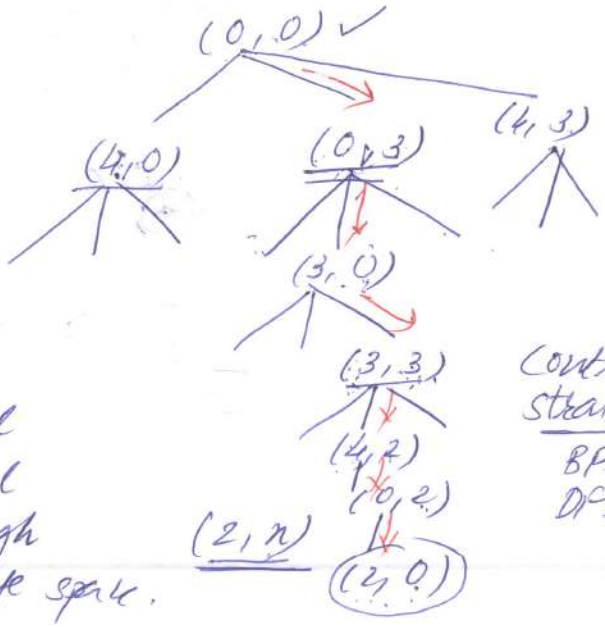
(1,0) (2,0) (3,0) (4,0) (0,1)

(1,1) (2,1) (3,1) (4,1) (0,2)

(1,2) (2,2) (3,2) (4,2) (0,3)

(1,3) (2,3) (3,3) (4,3) (

(0,0)



Rules :

LHS $\xrightarrow{op^n}$ RHS

current — Next

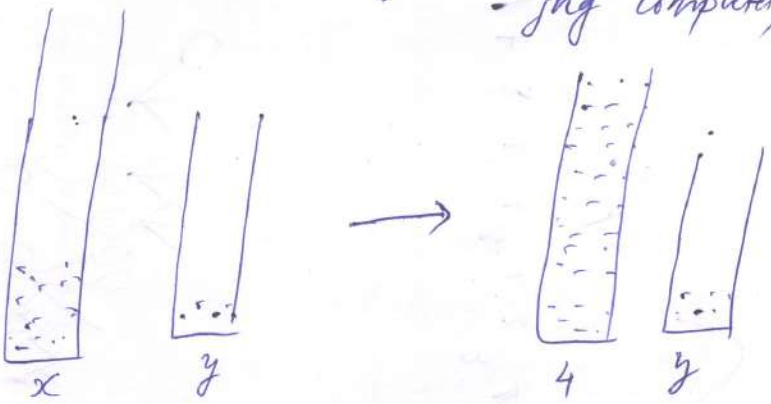
Path from initial state to the goal state. Passing through various states of the state space.

Control strategy

BFS

DFS

LHS		RHS	Operation
1. (x, y) if $x < 4$	$\longrightarrow$	$(4, y)$	Fill the first jug completely



2. (x, y) if $y < 3$	$\longrightarrow$	$(x, 3)$	Fill the second jug completely

L-4

State Space Search

Search Algorithms

- Depth-limited search
- Uniform cost
- Iterative deepening

uninformed search algo. ("blind")
- DFS, • BFS

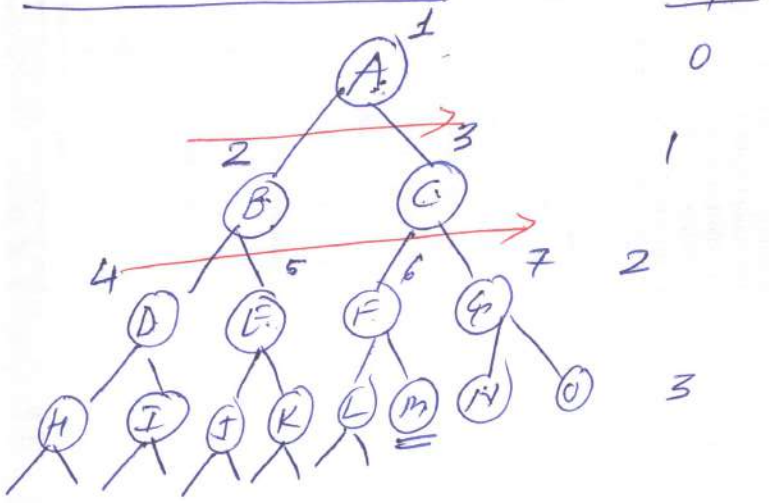
informed search algo/heuristics algo.

heuristics algo: information about "goodness" or "appropriateness" about each state.

goodness: how far/right the current-/given state is from the goal state.

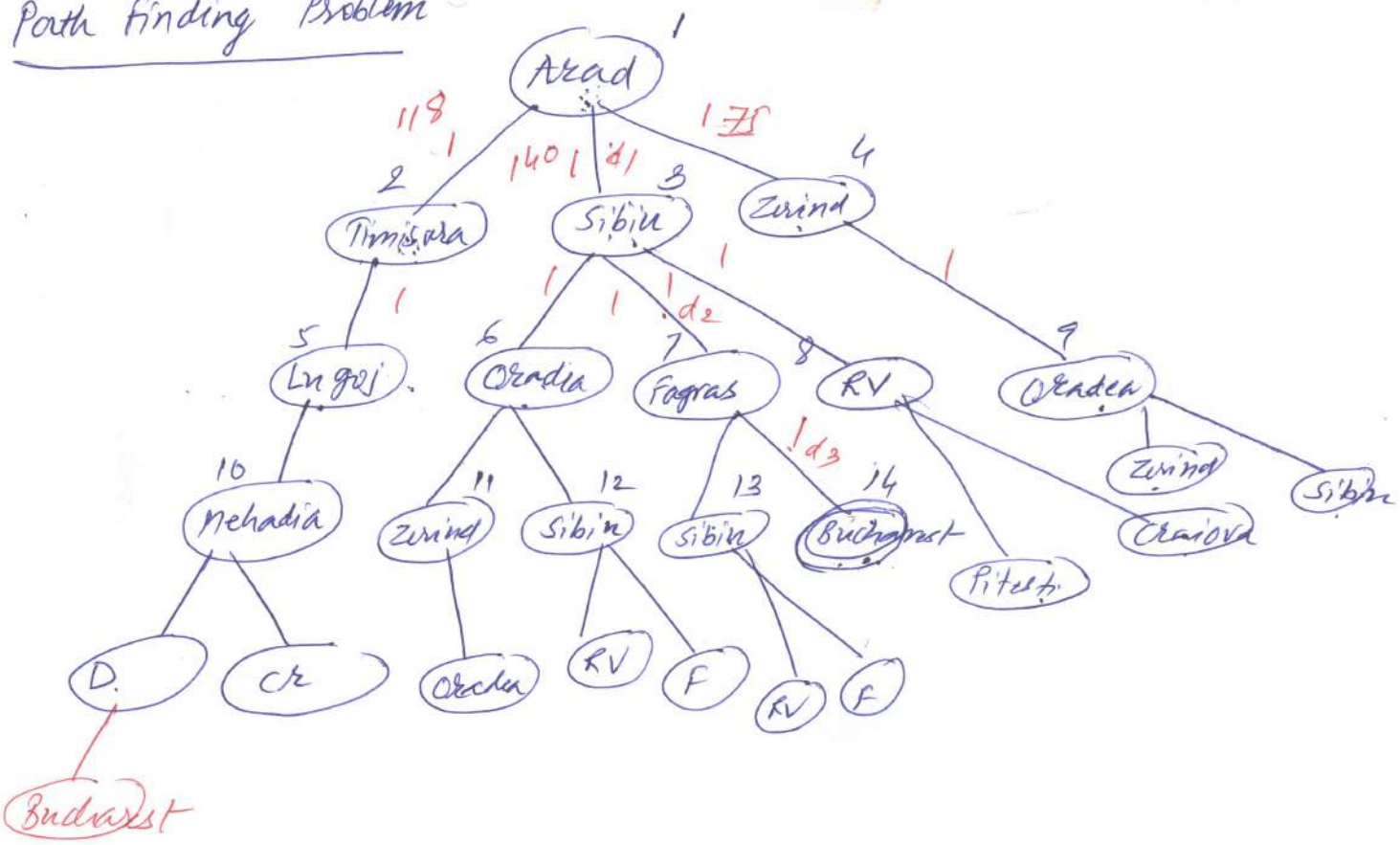
uninformed search: no info. about "goodness".

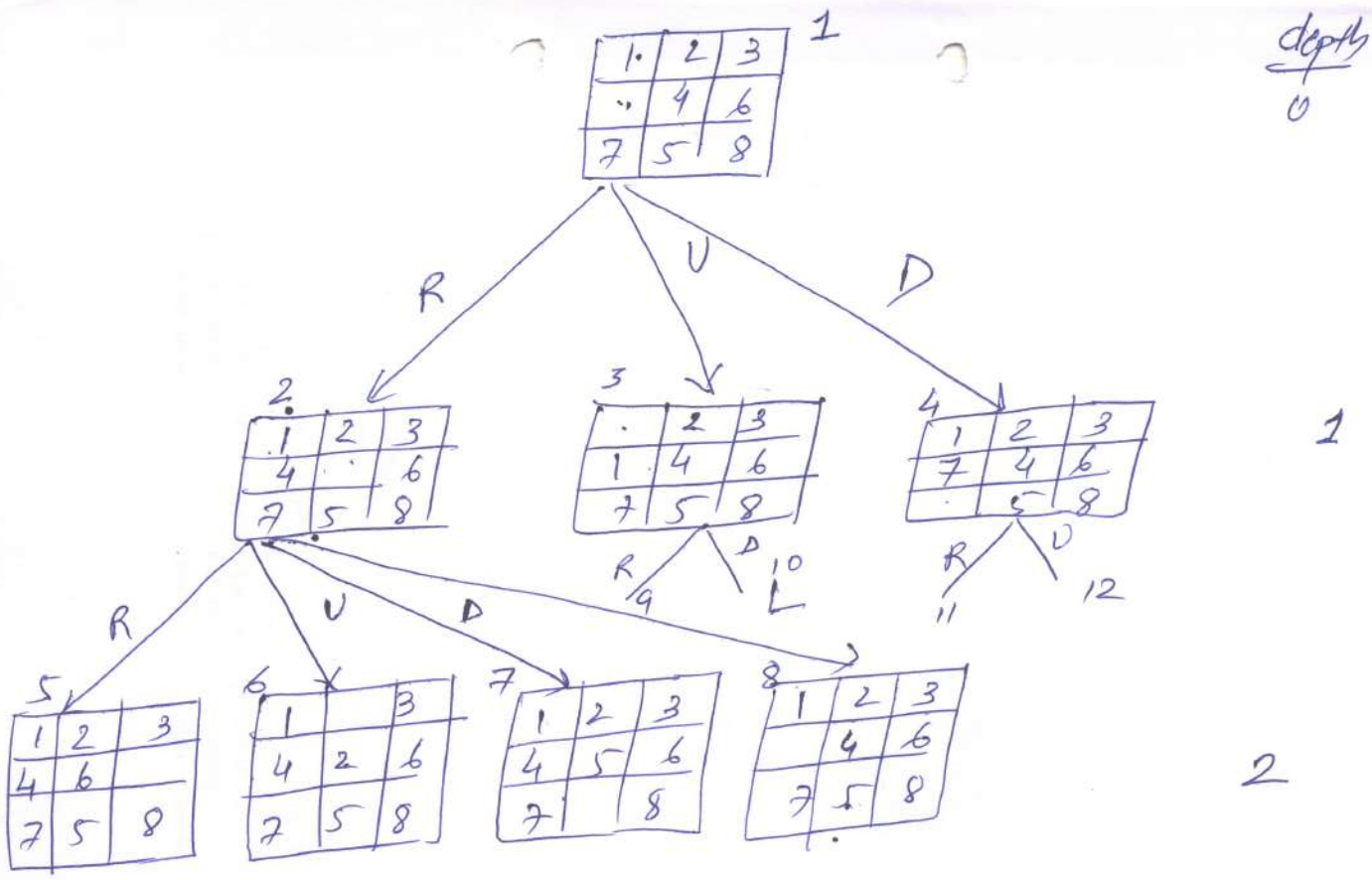
Breadth- First Search



assume that our solution state is (M)
Before generating children nodes of the given node, check if the given node is the same as the goal state. If yes, stop. Else generate the child nodes.

Path finding Problem




depth
0
1
R
U
D
2
3
4
1
R
U
D
R
D
10
L
9
R
U
11
12
5
6
7
8
2

Desirable Properties of Search Algo.

1. Completeness – the algo should be able to find a solution, if it exists.
2. Optimality – the algo is able to find the least cost solution.
3. Time Complexity – low/minimum
4. Space Complexity – low/minimum

BFS

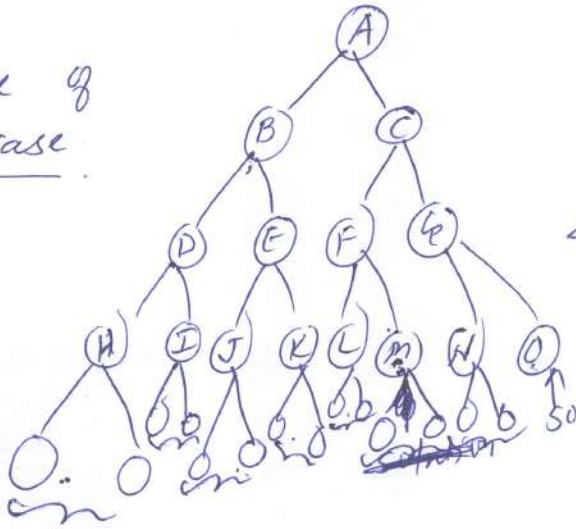
1. Complete
2. Not always optimal
3.
4.

L-5

b = branching factor = 2

d = depth at which the shallowest goal node is present = 3

Example of worst-case:



nearest depth	# nodes
0	$1 - 2^0$
1	$2 - 2^1$
2	$4 - 2^2$
3	$8 - 2^3$
4	$10 - ? \; 2^4 - 2 = 14$

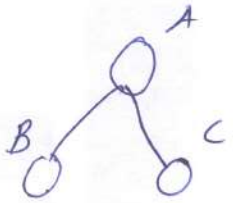
d+1

Space complexity $= 2^0 + 2^1 + 2^2 + 2^3 + \underline{\quad ? \quad}$

$= b^0 + b^1 + b^2 + b^3 \ldots b^{d+1} \; \underline{(b^{d+1} - b)}$

$= O(b^{d+1})$ – exponential

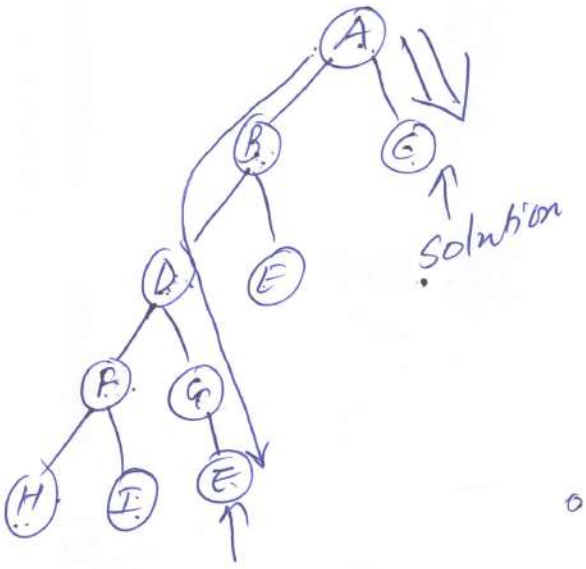
Time Complexity = no. of nodes processed by the algorithm
= Space Complexity



Best case : solution/goal node is at level 1

~~m — max. depth of the tree~~

Depth-first Search

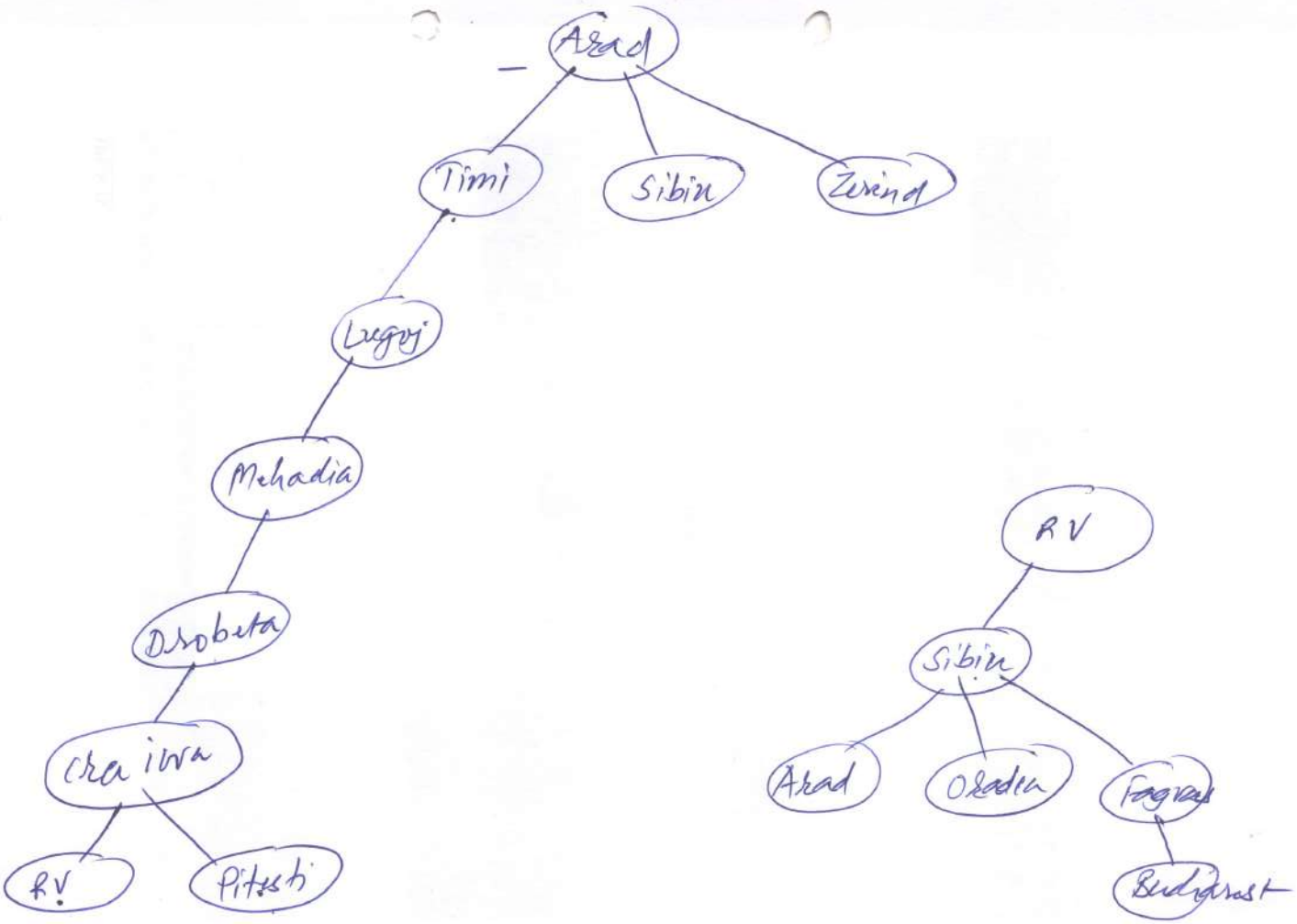


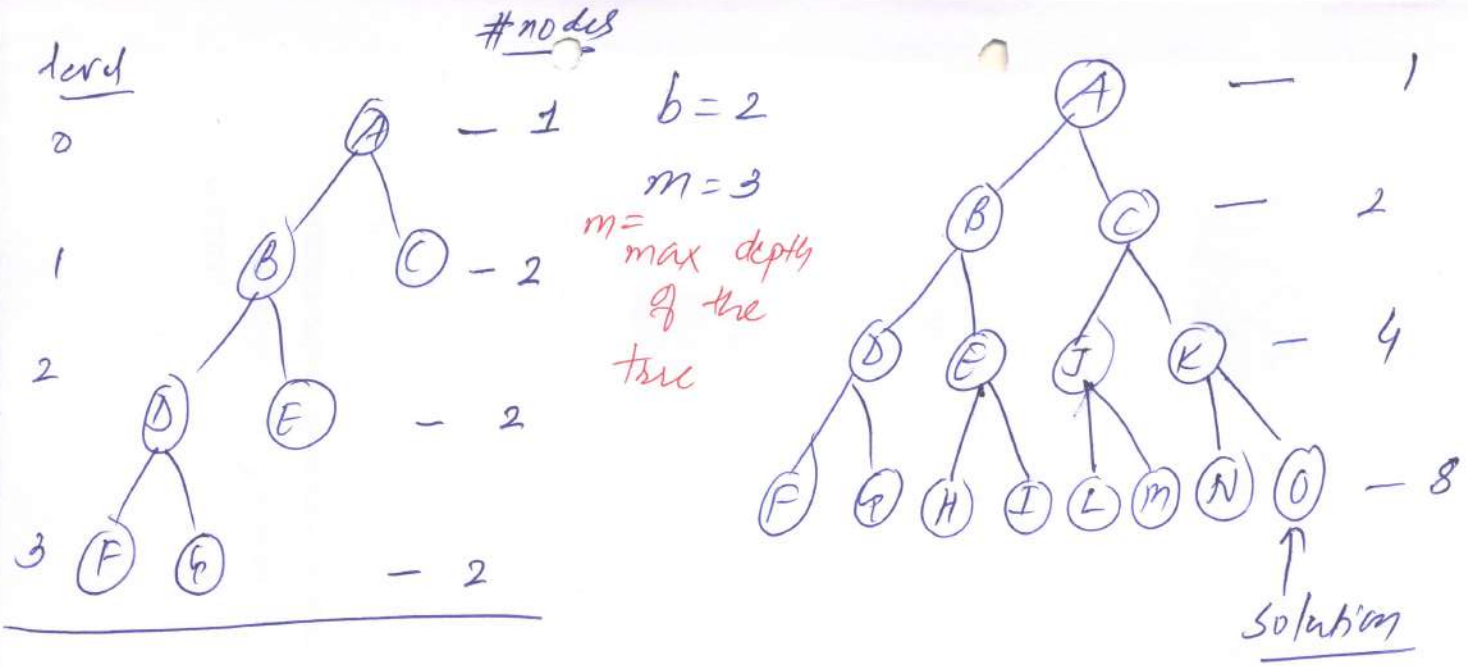
- Not complete
  - continuously generating new states, but not finding the goal state
  - cycle

Dead end

backtracking

- Not an optimal - DFS may not find the least cost solution, even if it exists.
- Space complexity - $O(bm)$
- Time complexity - $O(b^m)$


Arad
Timi
Sibiu
Zerind
Lugoj
Mehadia
Dobreta
Craiova
RV
Pitesti
RV
Sibiu
Arad
Oradea
Fagaras
Bucharest



Space Complexity = 1 + 2 + 2 + 2

= 7

= bm + 1

= O(bm)

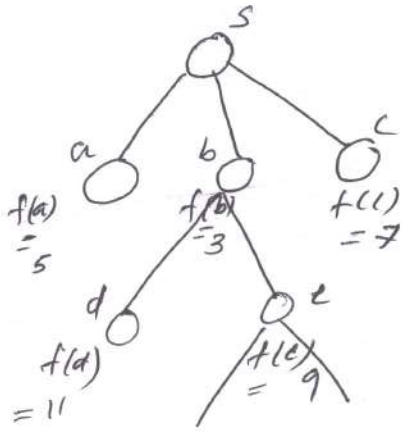
Time Complexity = $O(b^m)$

Heuristics Search Techniques

- Every node n is evaluated for $f(n)$.

$f(n)$ => estimated distance of node n from the goal state ($f(n) = h(n)$)

$f(n)$ → evaluation function



($f(b)$ is min of $f(a)$, $f(b)$ & $f(c)$. b is the most promising)

L-6

Search

Uninformed

- Systematically generating states & checking for goal states.

informed

- Use problem specific knowledge to decide which node should be explored next.
  - not given in the problem defn
- Explore the node which is the most promising.

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

$g(n)$ = cost to reach from initial state/node to the node n

$h(n)$ = estimated cost of reaching from node n to the goal node g.

Greedy Best-first-Search $\rightarrow f(n) = h(n)$ $(g(n) = 0)$

A* algo $\rightarrow f(n) = h(n) + g(n)$

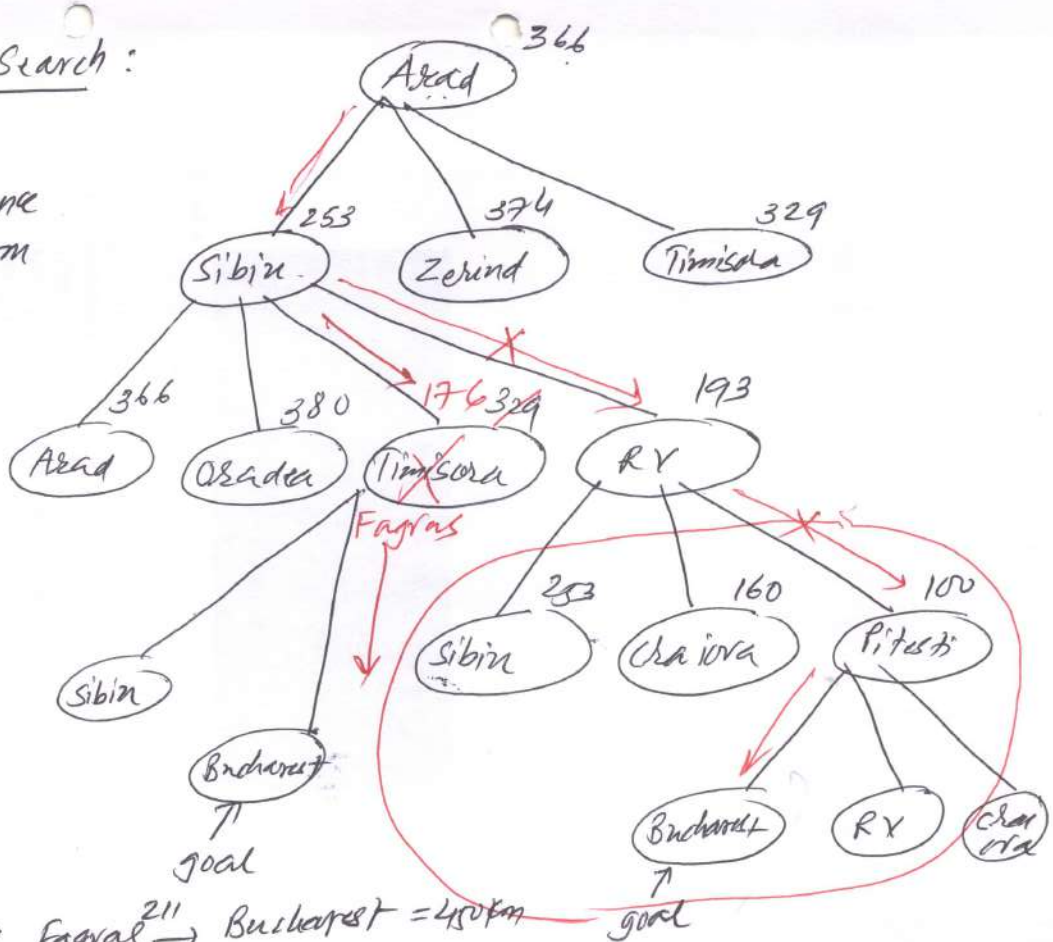
Greedy - Best - first - Search :

- $f(n) = h(n)$
  = estimated distance of node n from the goal node

- find $h(n)$ — experience

SLD - straight Line Distance

$f(\text{Sibiu}) = h(\text{Sibiu}) = 253$
$f(\text{Zerind}) = h(\text{Zerind}) = 374$



- Solution:
  Arad $\xrightarrow{140}$ Sibiu $\xrightarrow{99}$ Fagras $\xrightarrow{211}$ Bucharest = 450 km

- not optimal, not complete
  Arad $\xrightarrow{140}$ Sibiu $\xrightarrow{80}$ RV $\xrightarrow{97}$ Pitesti $\xrightarrow{101}$ Bucharest = 418 km

A* Algo :

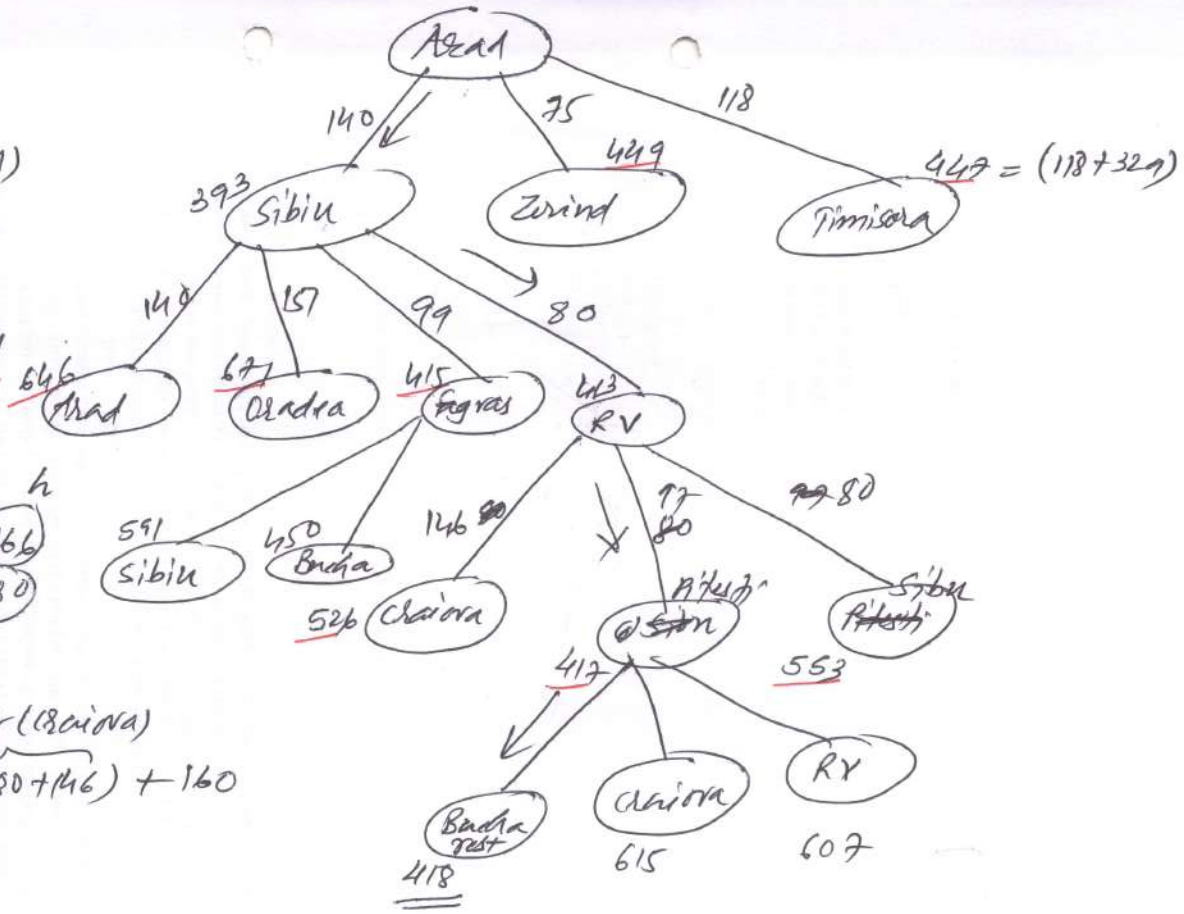
$f(n) = g(n) + h(n)$

$f(\text{Sibiu}) = 140 + 253$
$= 393$

$f(\text{Zerind}) = 75 + 374$
$= 449$

$f(\text{Arad}) = (280) + (366)$ (f, g, h)
$f(\text{Oradea}) = (291) + (380)$
$f(\text{RV}) = 413$

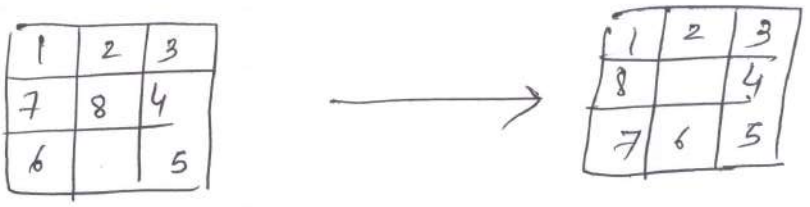
$f(\text{Craiova}) = \overbrace{(140+80+146)}^{g(\text{Craiova})} + 160$
$= 526$



- Optimal
- complete

Lect-7

8- puzzle problem



3 x 3 board
initial state

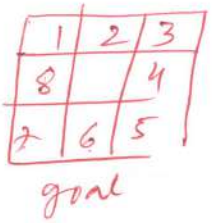
goal state

4 operations: Up, Down, left, Right
(available on blank)

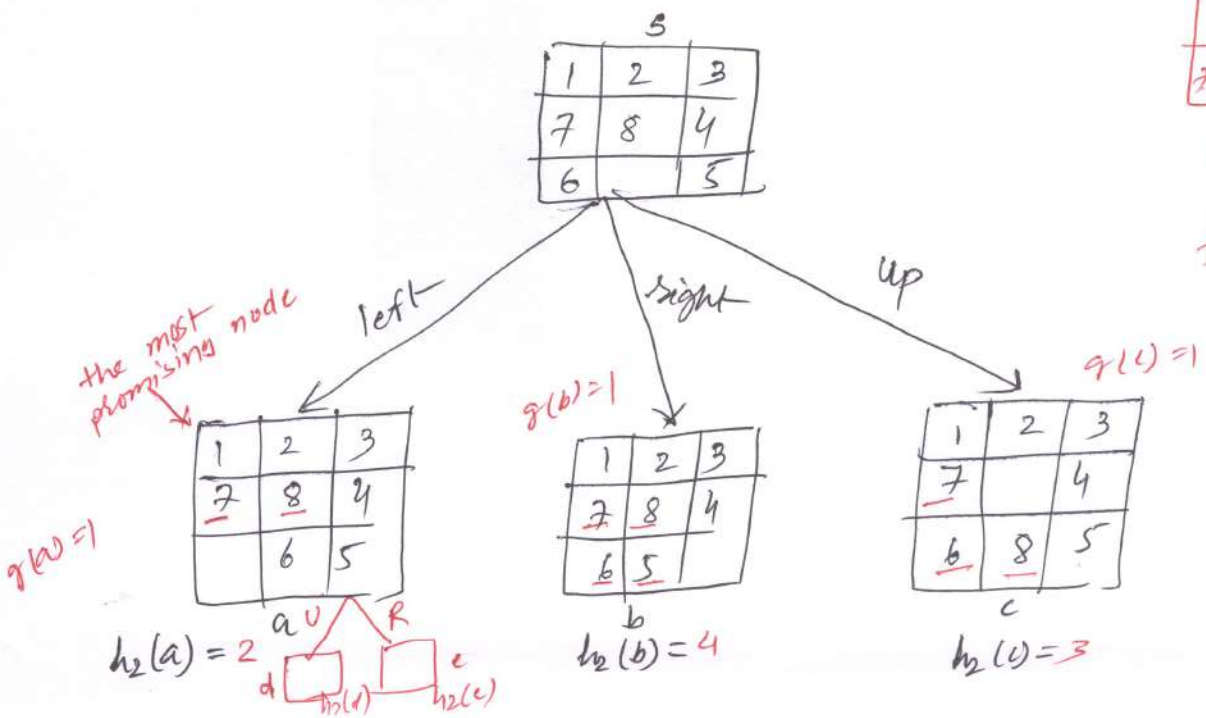
what should be "heuristics" ?

- One possible Heuristics is : no of tiles in correct position. (function h1)

The other Heuristics is : no of tiles in incorrect position. (function h2)

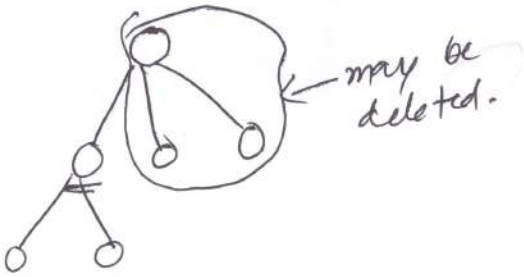


$f(n) = g(n) + h(n)$



✱ Localized Search Algorithms:

- Local Search : We generate childsen nodes of given node, Then pick the best/promising node and ~~explo~~ explore it. Delete the remaining nodes.



- Less memory

can not Local search algo.

Route-finding
job-scheduling problem

1. Hill climbing Algorithm

HILL_CLIMBING(problem) returns a state

current_state = initial state

loop do (children)

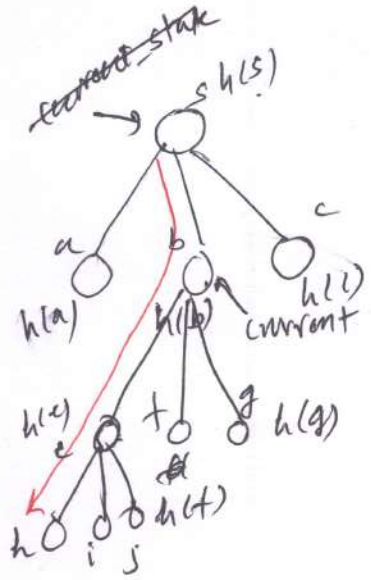
generate neighbours of the current node.

find the best_neighbor_node

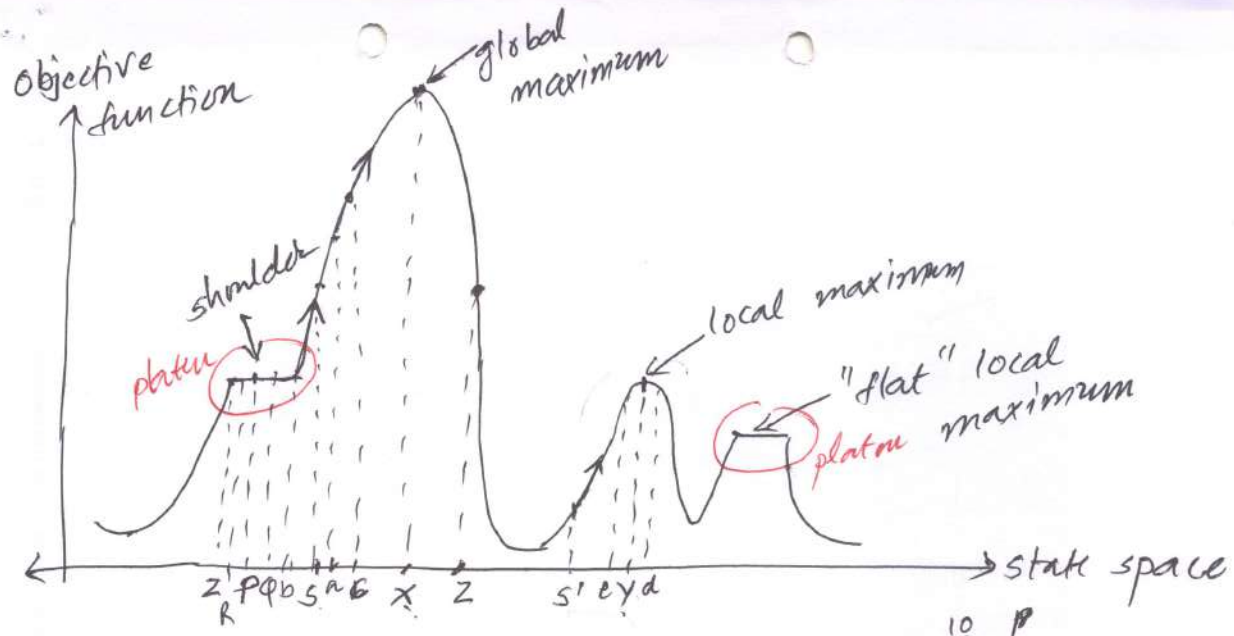
if (h(current) ≥ h(best_neighbor_node))

return current_state

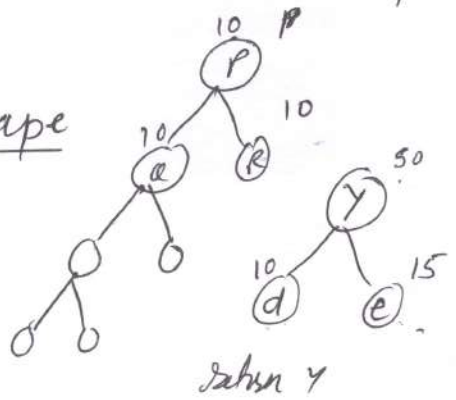
current_state = best_neighbor_node



"steepest ascent version"



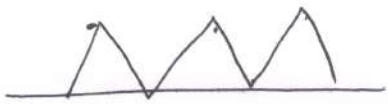
state space landscape



Subtree Y

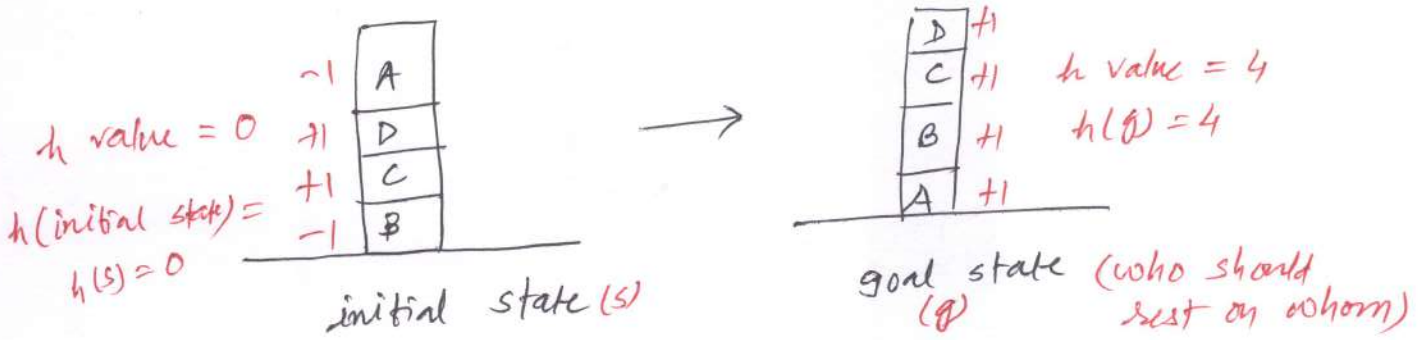
Limitations of Hill climbing :

1) Local maximum
2) Plateau – function is flat
3) Ridge



Example:

Blocks World game:



Apply steepest ascent - hill climbing.

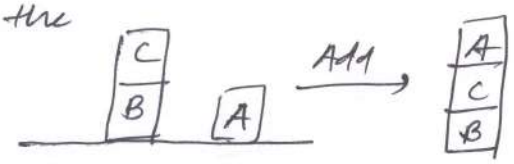
Heuristics function: +1 - to every block that is resting on correct ~~block~~ thing (either block or surface)

local heuristics

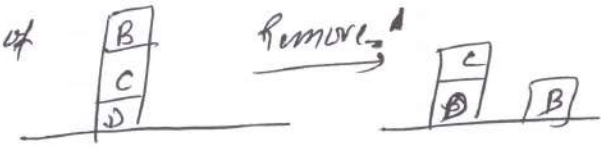
-1 - to every block that is resting on wrong thing.

Operations:

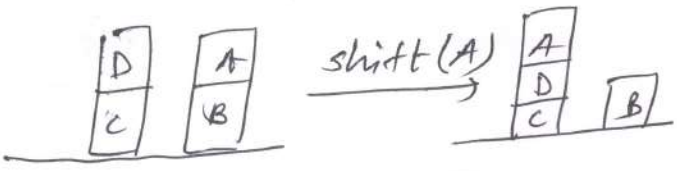
Add : Add a block from surface to the top of some structure

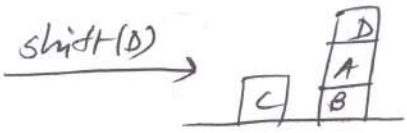


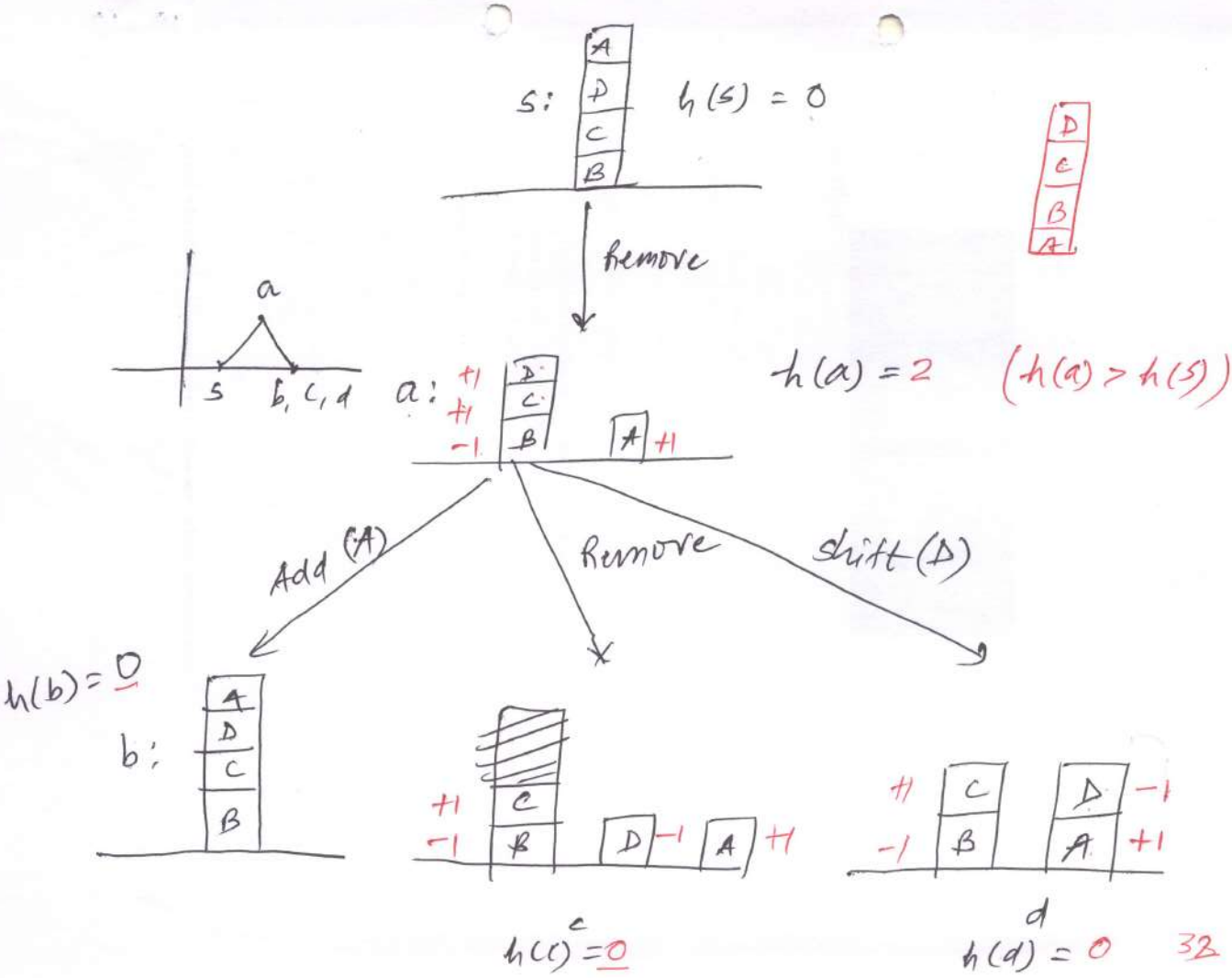
Remove : Remove a block from top of a structure & place on surface :



shift : Move a block from one structure to another structure.





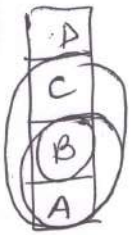

s:
A
D
C
B
h(s) = 0
Remove
a
s
b, c, d
a:
+1
+1
-1
D
C
B
A
+1
h(a) = 2
(h(a) > h(s))
Add (A)
Remove
Shift (D)
h(b) = 0
b:
A
D
C
B
+1
-1
C
B
D
-1
A
+1
c
h(c) = 0
+1
-1
C
B
D
-1
A
+1
d
h(d) = 0

AI Lect_9

Global Heuristics :

For each block that has correct support-structure :
+1 to every block in support-structure

For each block that has wrong support structure :
-1 to every block in support-structure



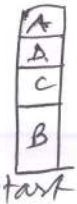
goal (g)

D : C +1, B +1, A +1

C : B +1, A +1

B : A +1

$h(g) = 6$



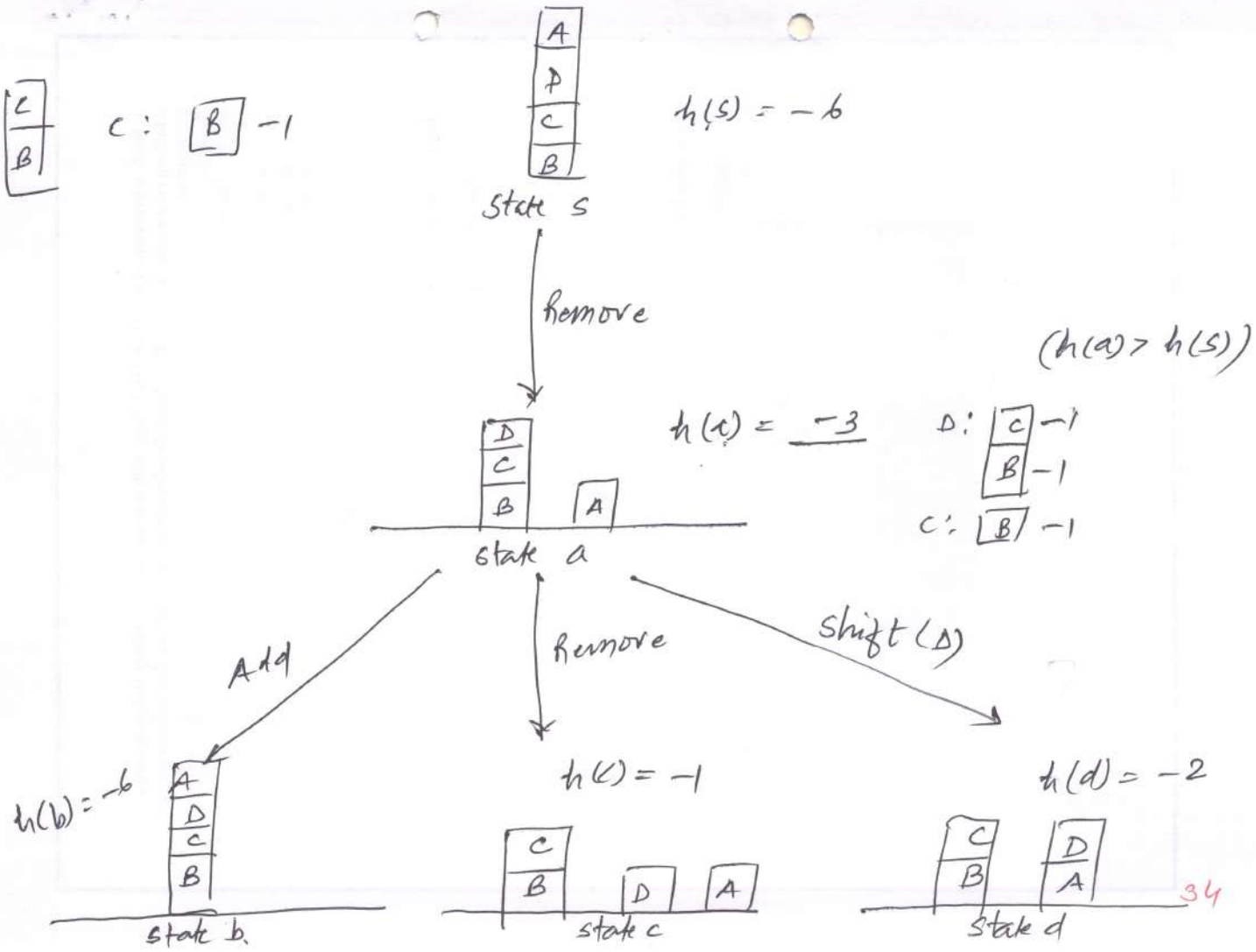
start state (s)

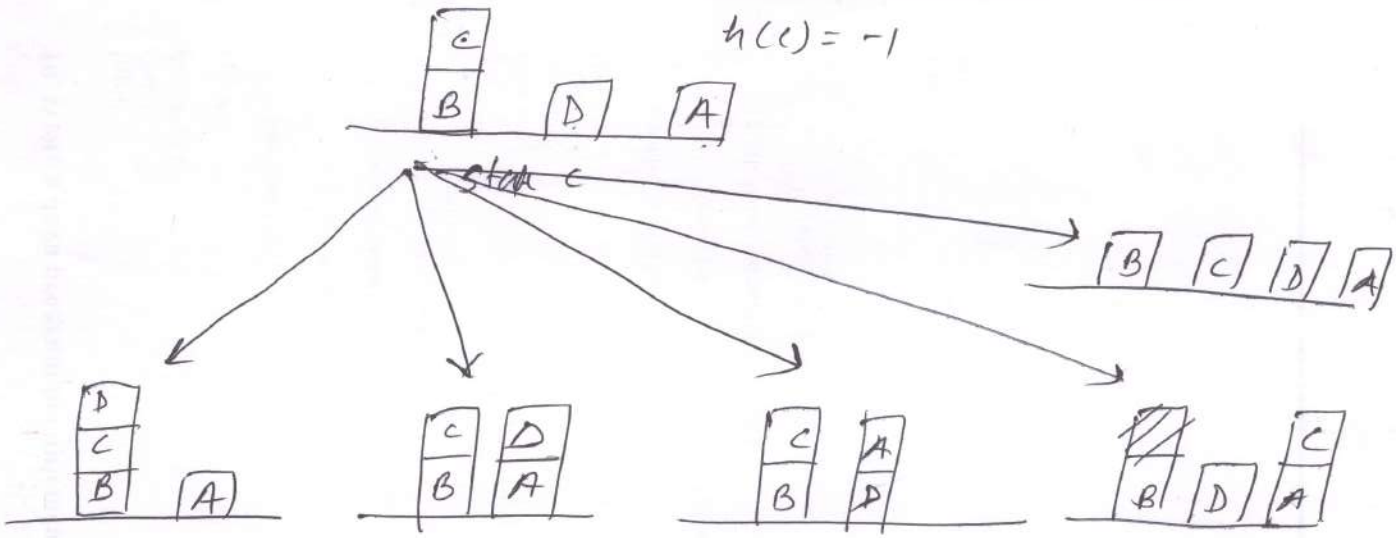
$h(s) = \underline{-6}$

A : D -1, C -1, B -1

D : C -1, B -1

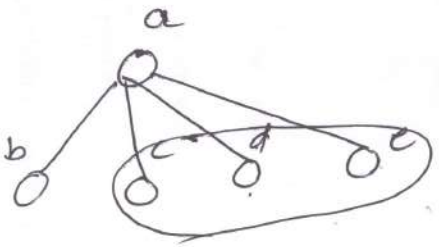
C : B -1


C
B
c : B -1
A
D
C
B
h(s) = -6
State s
Remove
(h(a) > h(s))
h(a) = -3
D
C
B
A
D: C -1
B -1
c: B -1
state a
Add
Remove
Shift (D)
h(b) = -6
A
D
C
B
h(c) = -1
C
B
D
A
h(d) = -2
C
B
D
A
state b
state c
state d


C
B
D
A
h(c) = -1
B C D A
D C B A
C B D A
C B A D
B D C A

stochastic hill climbing :

It selects at random any one child node which has higher heuristics value than the given node.



$h(b) < h(a)$

$h(c) > h(a)$

$h(d) > h(a)$

$h(e) > h(a)$

steepest ascent :

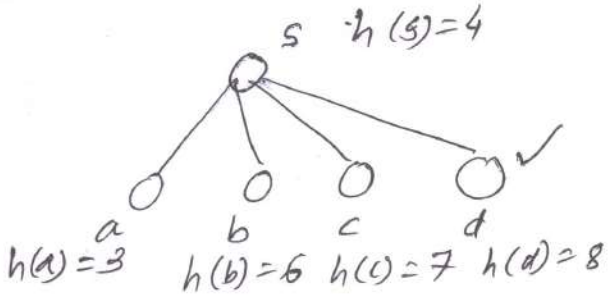
$\max(h(c), h(d), h(e))$

stochastic hill climbing :

select any one from b, c and d randomly.

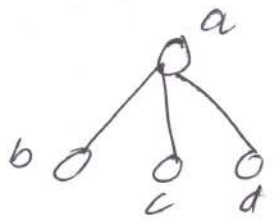
Simulated Annealing Algorithm

- Metallurgy — Annealing : heat the metal or glass at high temperature.
  Then cool it down gradually.



First choice Hill climbing :

Childsen nodes of given node are randomly generated one by one. The first node that is better than the cussent node is selected.



SIMULATED_ANNEALING (problem, schedule)

current_state = initial state

for t = 1 to ∞ // while(1)

T = schedule[t]

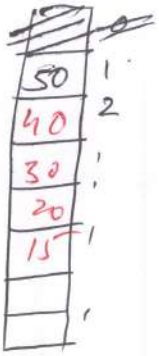
terminating condition → if T = 0 then return current_state

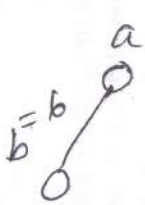
next_state = a randomly selected successor of current_state

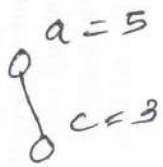
ΔE = h(next_state) − h(current_state)

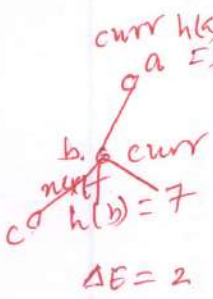
if $\Delta E > 0$ then current_state = next_state

else current_state = next_state with probability $e^{\Delta E/T}$.





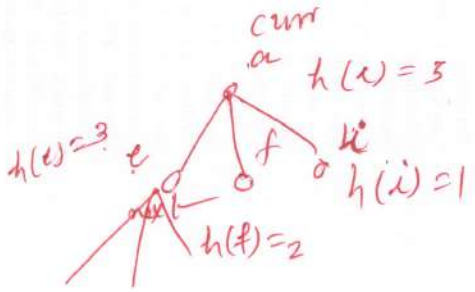




$\Delta E = 2$

$T = 50 \quad \Delta E = -2 \quad e^{-2/50} = \frac{1}{e^{2/50}}$

Lect-10



$\Delta E = -2$, $T = 10$

$e^{\Delta E/T} = 0.81$

$\Delta E = -2$, $T = 30$

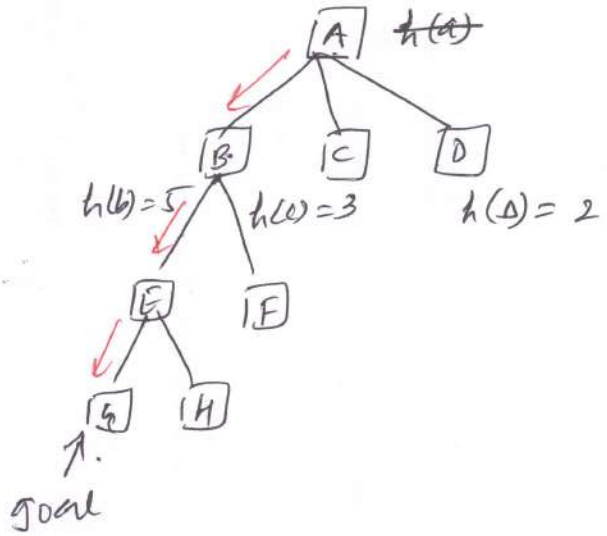
state e will become next state with prob. $e^{\Delta E/T}$

$e^{\Delta E/T} = e^{-2/30} = \boxed{0.93}$

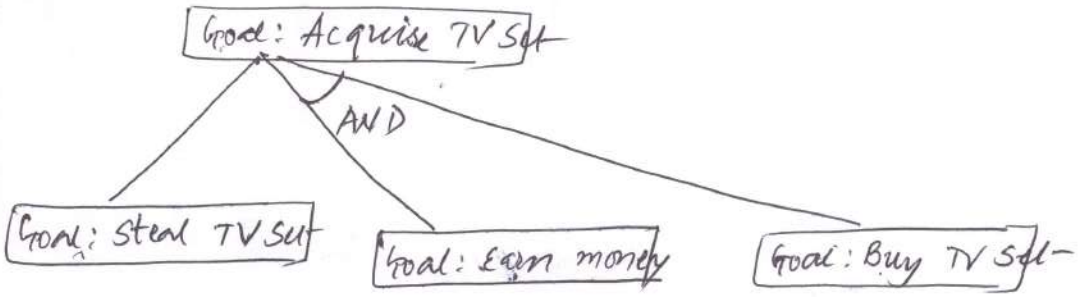
```
else
    r = random()  // generates a random no. bet^n 0 & 1.
    p = e^(ΔE/T)
    if (r ≤ p)
        curr_state = next_state
```

* AO* Algorithm

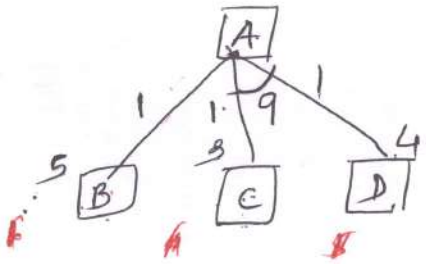
- Best-first-search / A* — OR graphs



AND-OR Graph

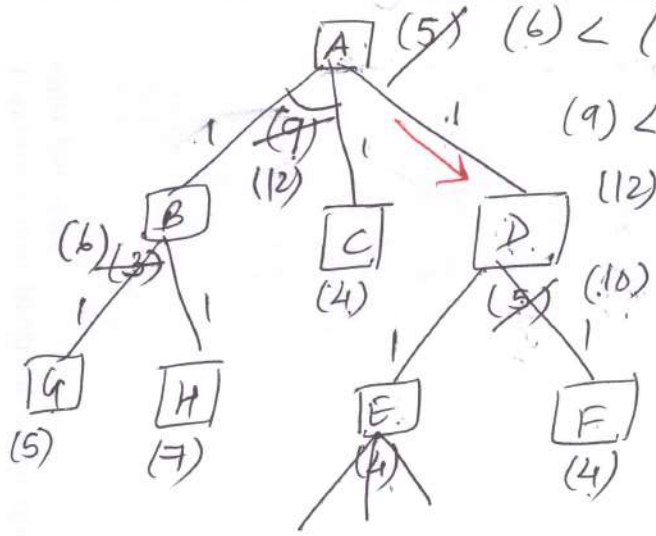


Acquire TV set = Steal TV set OR
(Earn money AND Buy TV set)



(smaller value means better.)

- A* / best-first-search algo. do not handle AND-OR graphs properly.
- To deal with AND-OR graphs, AO* algo is proposed.



(6) < (9) ⇒ (6) (Select D)

(9) < (10) ⇒ (9) (Select B, C)

(12) ≠ (11) ⇒ 11 (Select D)

~~So~~

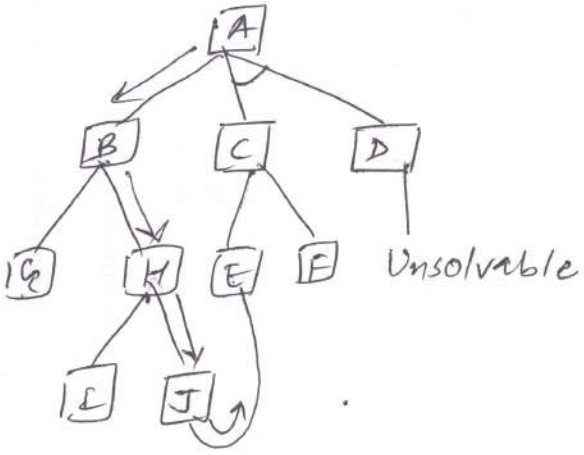
Lect-11

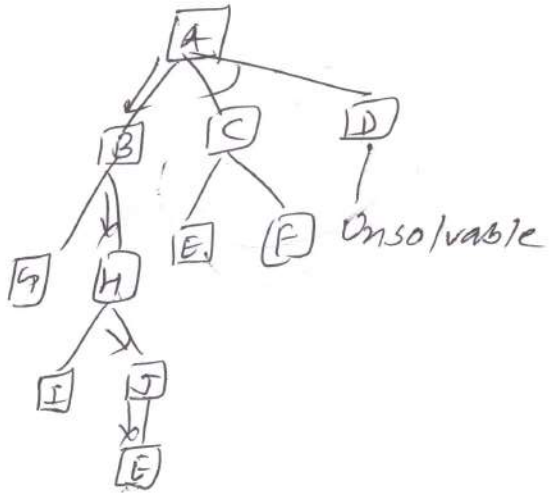
* Some important points about AO* algo :

(1) Sometimes longer path is taken

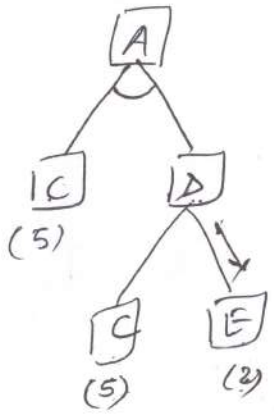
(2) It does not take into account interacting subgoals.

(1)





(2) Interacting Subgoals



node C comes twice
- To solve A, C has to be solved
- To solve A, we rely on E. But if we rely on C, without doing any effort D would be solved.

(Smaller the number, it is better)

The AO* algo solves C and E both. It does not take benifit of repeated subgoal.

Best-first search

A^*

AO^*

Hill climbing

Simulated Annealing

* Constraint Satisfaction Problems (CSP)
  - 8-queens problem
  - graph colouring
  - cryptarithmatic problem ✓

Cryptarithmatic Problem Examples:

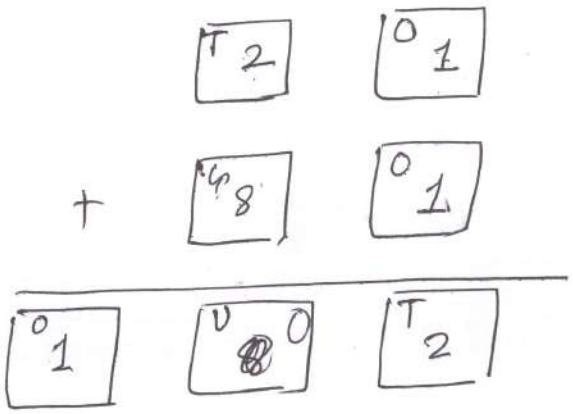
(1)

```
  T O
+ G O
-----
O U T
```

Assign a digit from 0 to 9 to each letter such that above formula is true.

Constrainsts:

- No two letters can be assigned the same digits.
- Digits should be from 0 to 9.
- Sum of two letters must be as given in formula.



T = 9

4 = 8

17

~~10: 11 to 19~~

OU : 11 to 19

11
12
13
14
15
16
17
18
~~19~~
10

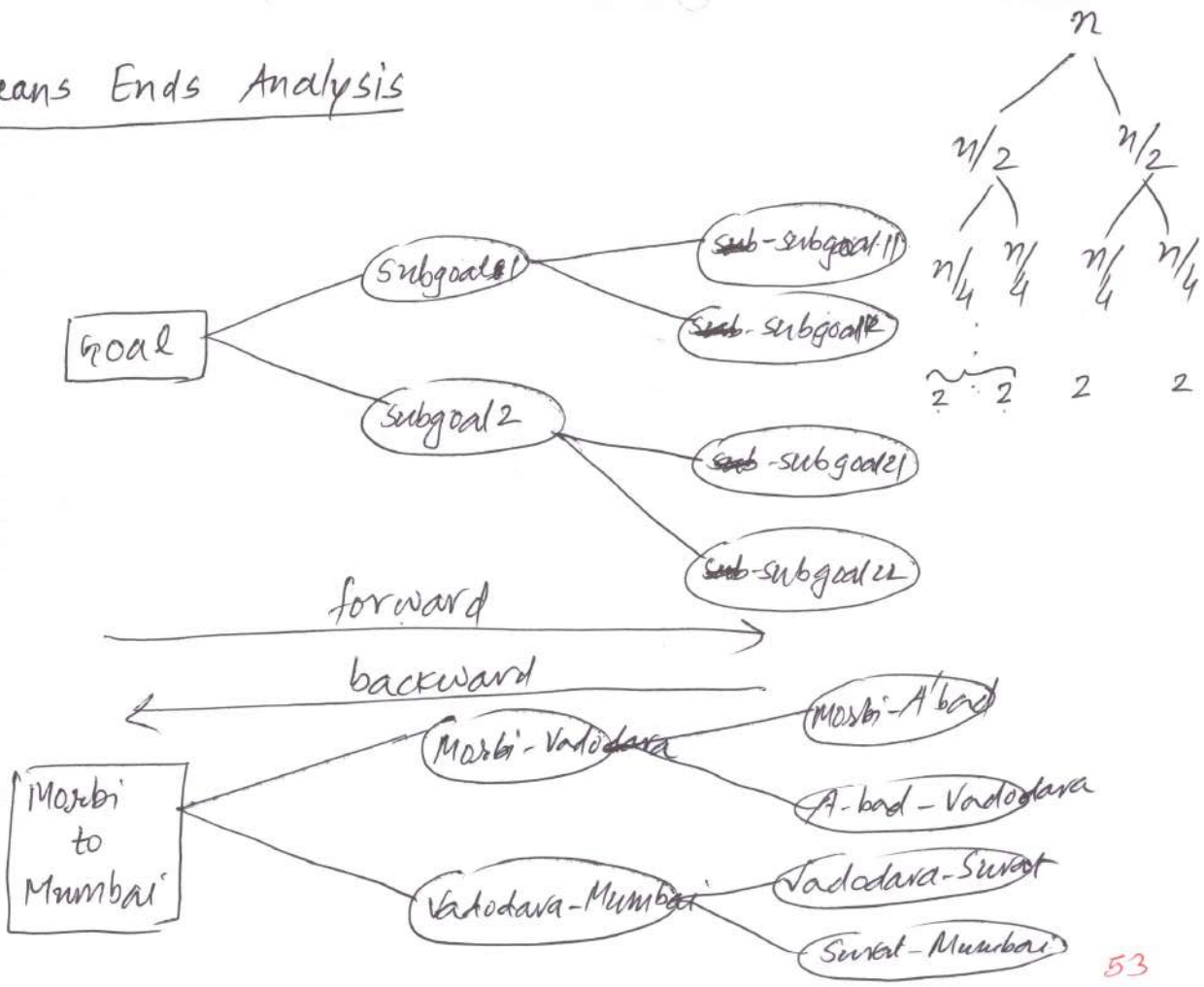
x4 = 7 → T + 4 = 2 + 7 = 9

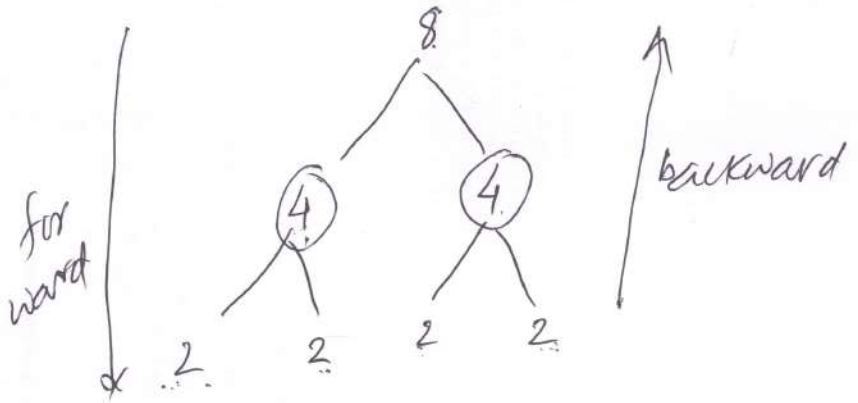
x (8) → T + 4 = 2 + 8 = 10

(9) → T + 4 = 2 + 9 = 11

49

* Means Ends Analysis





MEA
Algorithm (CURRENT, GOAL)

1) Compare CURRENT and GOAL states.
If there are differences, goto step 2
Else return success and stop.

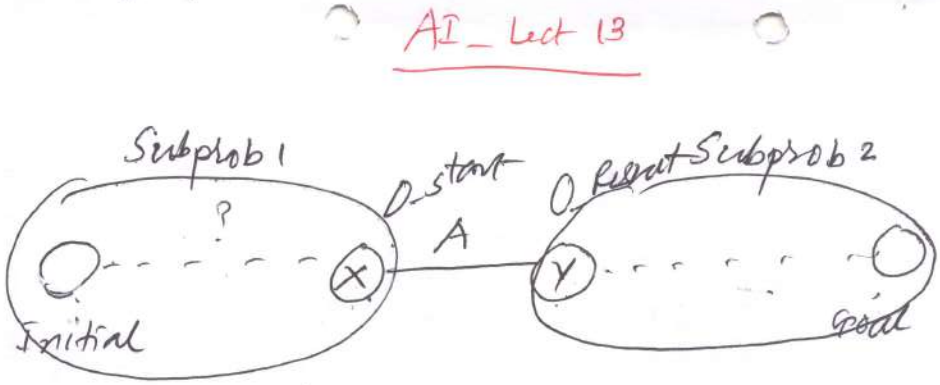
2) Select the most significant difference and reduce it by doing following steps.

a. Select an operator O which can reduce this difference. If there is no such operator, return failure.

b. O-start → a state in which Operator O's pre-conditions are satisfied.
O-Result → a state ~~in~~ which which result if Operator O is applied to CURRENT state.

c. First-part = MEA(CURRENT, O-~~result~~ START)
Second-part = MEA(O-Result, GOAL)
Result = First-part, O-~~result~~, Last-part.
Return Result

AI_ Lect 13



MEA (I, G)

MEA (I, O_start)

MEA (O_Result, Goal)

Differences between Initial & goal

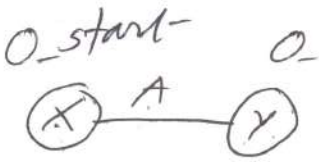
1

2

3 — operator A

4

5

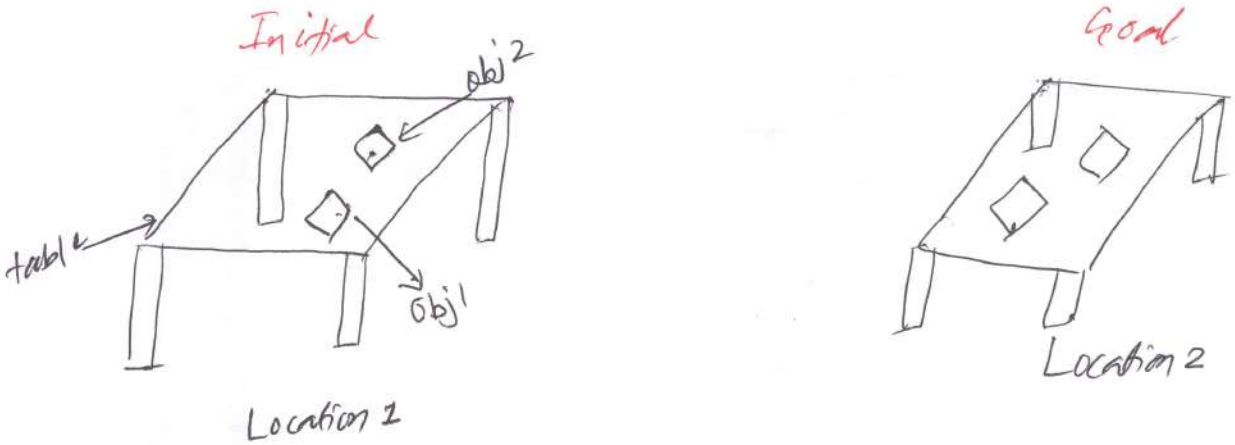


Differences between Initial & stak x

a

b — operator P

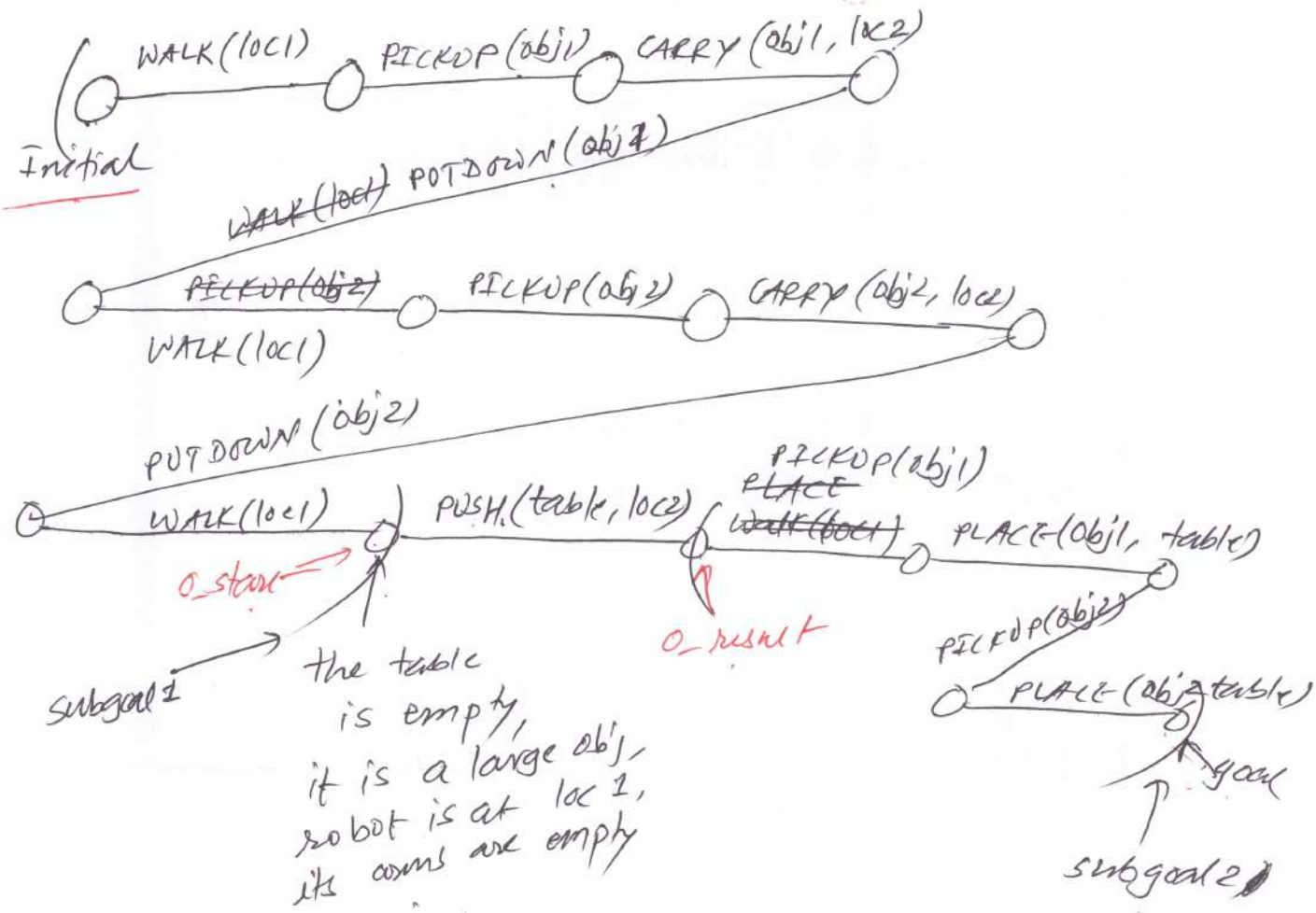
c



Robot

PUSH(obj, loc) – Robot will push object obj to location loc

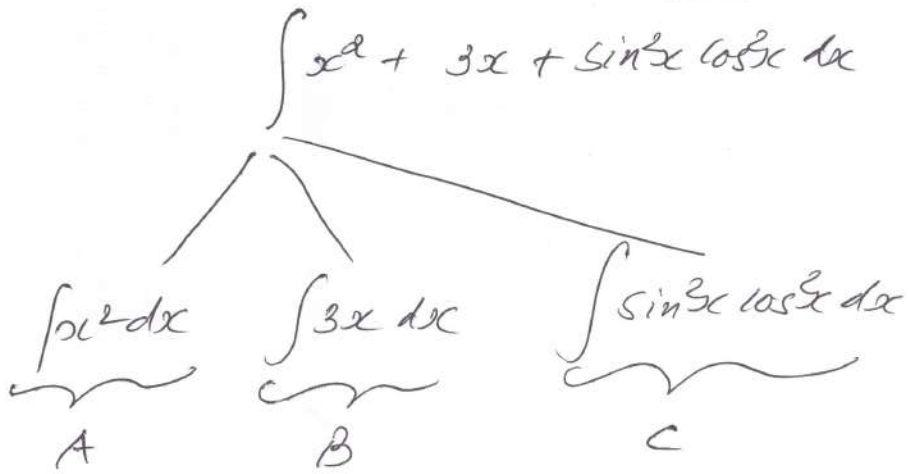
dest location


WALK(loc1)
PICKUP(obj1)
CARRY(obj1, loc2)
Initial
~~WALK(loc1)~~ PUTDOWN(obj1)
~~PICKUP(obj2)~~
PICKUP(obj2)
CARRY(obj2, loc2)
WALK(loc1)
PUTDOWN(obj2)
WALK(loc1)
PUSH(table, loc2)
PICKUP(obj1)
~~PLACE~~
~~WALK(loc1)~~
PLACE(obj1, table)
o_start
o_result
subgoal1
the table is empty, it is a large obj, robot is at loc 1, its arms are empty
PICKUP(obj2)
PLACE(obj2, table)
goal
subgoal2

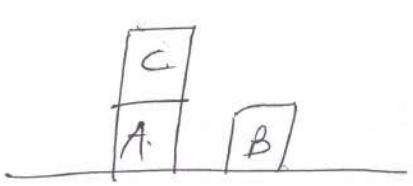
* characteristics of AI Problems:

1) Is the problem decomposable?

- To solve the main problem, is it possible to divide it into smaller problems and then combine their solutions. (↓ independent)

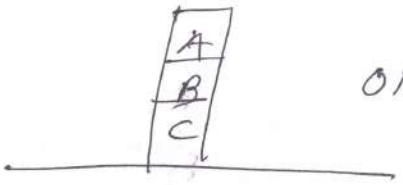


An example of decomposible problem



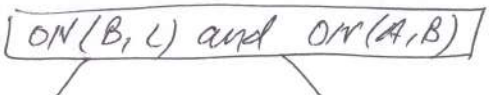
initial

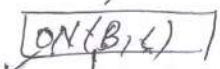
ON(C,A) and ~~ON~~ ONFLOOR(B)

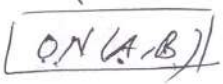


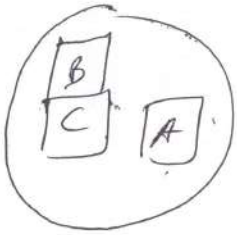
ON(B,C) and ON(A,B)

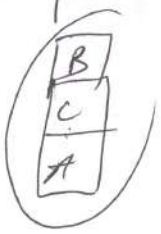
goal state

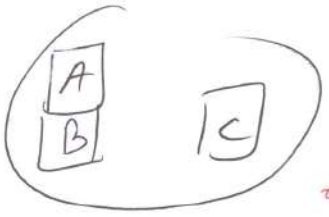




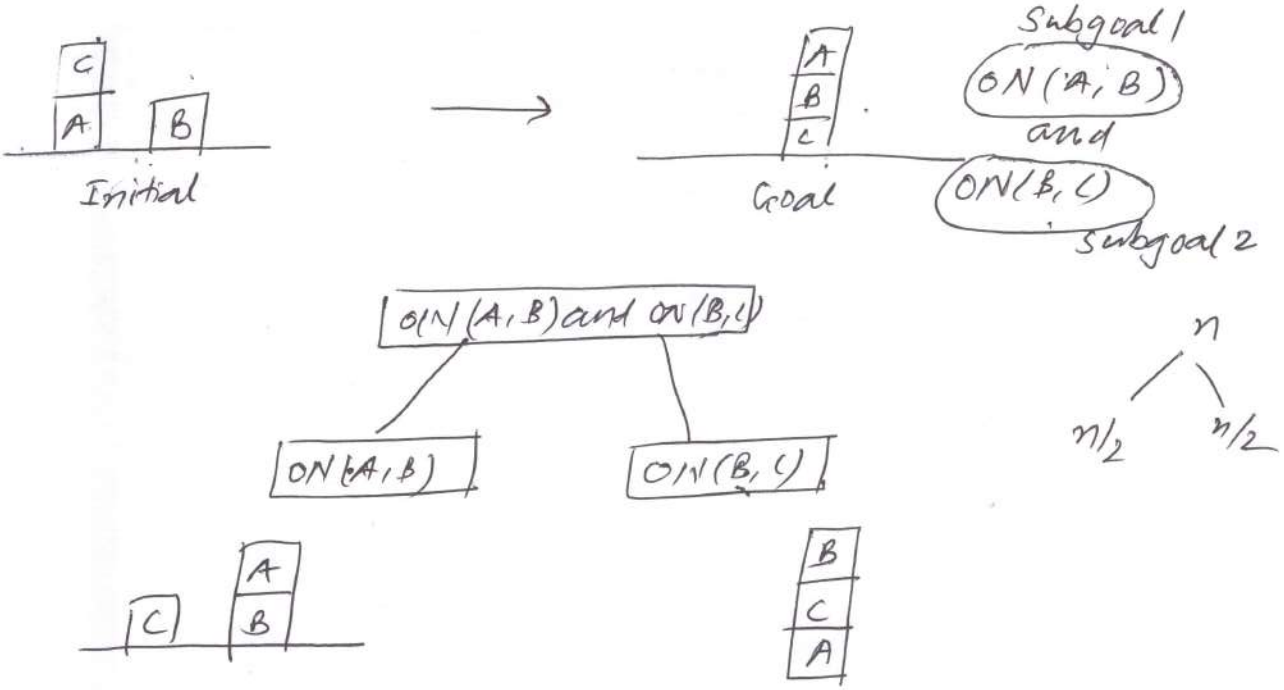








Lect_14


C
A
B
Initial
A
B
C
Goal
Subgoal 1
ON(A, B)
and
ON(B, C)
Subgoal 2
ON(A, B) and ON(B, C)
ON(A, B)
ON(B, C)
n
n/2
n/2
A
B
C
B
C
A

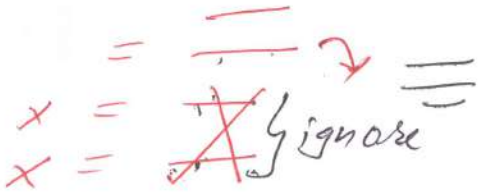
3. Is the Universe Predictable?

- Certain Outcome problems
  - Universe is predictable
  - blocks world problem

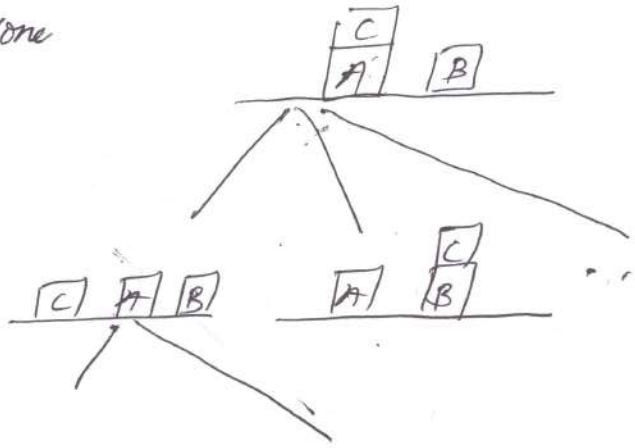
- Uncertain Outcome problems
  - Universe is not predictable
  - plan revision
  - chess

2. ~~Is~~ Can solution steps be ignored or Undone?

- Ignorable Problems – Proving a mathematical theorem
  - solution steps can be ignored



- Recoverable Problems
  - solution steps need to be undone
  - blocks world problem

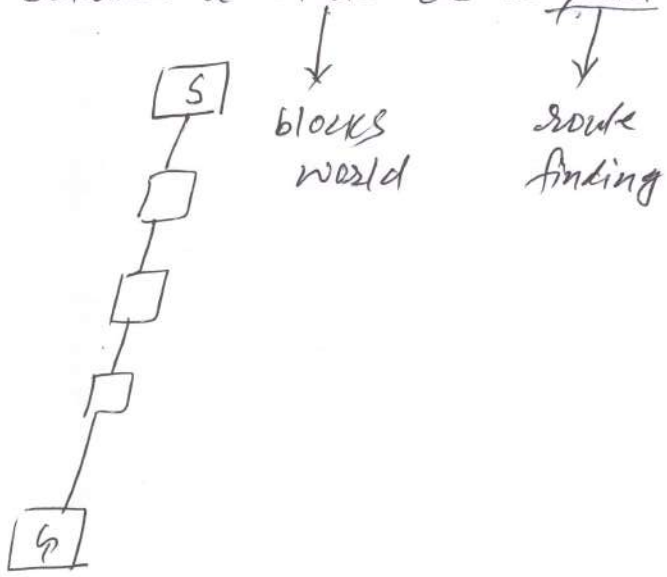


- Irrecoverable Problems
  - solution steps can not be ignored or undone.
  - game of chess

4. Is a good solution absolute or relative?

- Any path problem (absolute)
- best path problem (relative)

5. Is the solution a state or a path?



6. Role of knowledge

- Some problems which are solved efficiently if knowledge is used. → chess, cricket
- Some problems which can be solved not w/o using good amount of knowledge.
  - Knowledge is mandatory

7. Does ~~the~~ interaction ~~is~~ human is required ?

- Expert system for medical diagnosis. with