Artifical Intelligence

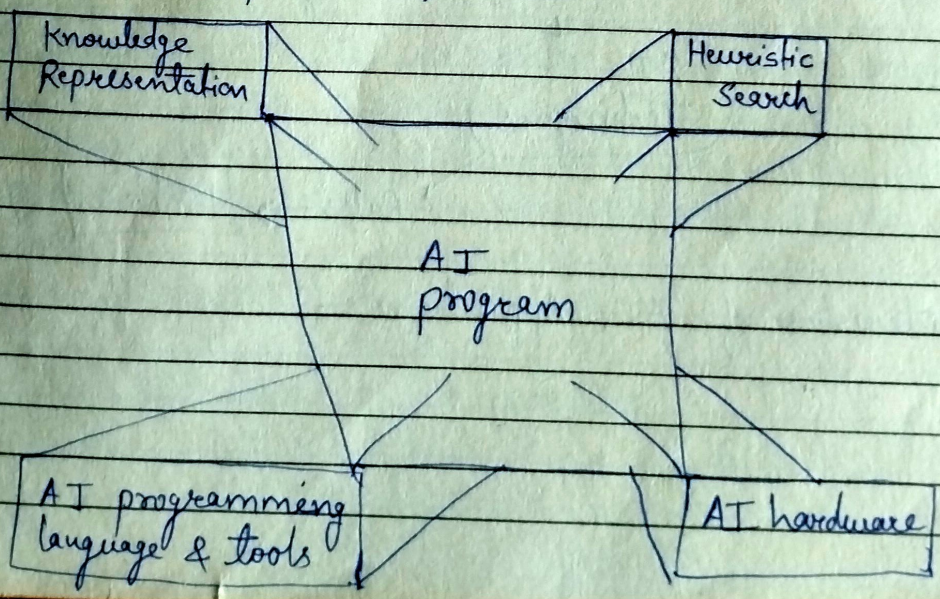
Date: Page:

- **AI** → Artifical Intelligence is usually defined as Science of making computers do things that require intelligence done by human. AI is a branch of science which deals with helping machines find solutions to complex problems in a more human like fashion.

- **Application of AI**

* Game Playing → AI can be used for gaming purpose. The AI machines can play strategic games like chess.
* Speech recognition
* Natural language processing → is used to make the conversation sound as human and personal as possible.
* Computer Vision
* Expert System
* Heuristic classification
* Handwriting recognition
* Intelligent Robot

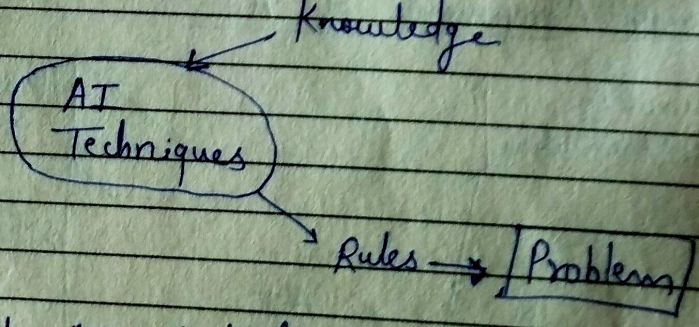
- **Major Components of AI** –



AI Techniques → It is a method that exploides knowledge that should be represented in such a way that

→ knowledge captures generalisation
→ Understandable by people.
→ Easily modifiable to correct.
→ Can be used in many situations.
→ Can reduce its volume.

• Parts of AI techniques
→ knowledge representation — Used to capture the knowledge about real w
→ Search Algorithm — Finding / Searching solution the problem



• Characteristics of AI Applications —

LISP

(List Processing)

John McCarthy invented LISP in 1958-59. shortly after the development of FORTRAN. It was implemented by Steve Russell on an IBM 704 computer.
It is particularly suitable for AI programs as it processes symbolic information effectively.

- Features of LISP

→ It uses iterative design methodology and easy extensibility
→ It commonly allows updating the program dynamically.
→ It is machine independent.
→ It provides advanced OOP.
→ It provides wide ranging data types like objects structure, list, vectors, adjustable arrays, hash table and symbols.
→ It is expression based.
→ It provides a complete i/o library.

- Expression → LISP expression are called Symbolic expre. are s-expression. The s-exp. are composed of three valid objects, atoms, list and string. LISP program run either on an intepreter or as compiled code.

* program : (+ 7 9)
(write (+ 7 9))

LISP uses prifix notation
a * (b + c) / d (60 * 9/5) + 32
(/ (* a (+ b c) d) (write (+ (* (/ 9 5) 60) 32)

60 * /59
* /6059 + 32

Date: Page:

program 3
(write line "Hello world")

• Basic building blocks in LISP (4mk)

LISP program are written using 3 basic building blocks atom, list, string

* atom → Atom is a number or string of contiguous characters. It includes numbers and special character.
e.g name, 12308907, *hello*, Block#221, abc123

* list → A list is a sequence of atoms and/or other list enclosed in paraentheses ( ).
e.g. (i am a list)
(a (a b c) d e f g h)
(father tom (susan bill joe)
(sun mon tue wed thu fri sat)

* string → A string is a group of characters enclosed in double quotation mark (" ").
e.g. "I am a string".
"a ba c e # $ %"

Adding
• Comments → The semicolon (;) symbol is used for indicating a comment line
e.g. ; tell them whereabout

Date : Page :

Some Notable points –

- The basic numeric operation in LISP are +, −, *, /
- LISP represents a function call f(x) as (fx)
  e.g. Cos(45) → (cos45)
- LISP expressions are case insensitive

LISP data types – LISP data type can be categorized as

- Scalar types – number, character, symbol.
- Data structure – list, vector, bit-vector, string

Any variable can take any lisp object as its value unless you have declared at explicitely.

Although it is not necessary to specify data type for LISP variable however it helps in certain loops, expressions, in method declaration exp. and some other situation.

Data types are arranged into a hierarchy. A data type is a set of LISP object and many objects may belong to one such a set.

The type p predicate is used for finding whether an object belong to a specific type.

The type-of function returns the data type of given objects.

program
```
(setq x 10)
(setq y 34.564)
(setq ch nil)
(setq n 134.78)
```

```
(setq bg 11.0e+4)
(setq x 124/2)
(print x)
(print y)
(print ch)
(print n)
(print bg)
(print x)
```

Output:
```
10
34.564
nil
134.78
11.0e+4
124/2
```

⊙ LIST Manipulating function —

Car → It takes a LIST as arguement and return its first element.

Cdr → It takes a LIST as arguement and returns a list without the first element.

Cons → It takes two arg. and an element and a list and return a list with the element inserted at first place.

list → It takes any no. of arg and returns a list with the arguement as member element of the list.

Append → It merges two or more list into one.

last → It takes a list and returns list containing last element.

member → It takes two arg. of which the second must be a list, if the first arg. is member of II arg

terpri – terminate printing

(write (car (a b c d e f)))

Date: Page:

and then it returns the remainder of the list begining with the first argument.

reverse → It takes a list and returns a list with the top element in reverse order.

```
(write (car (a b c d e f)))
(terpri)
(write (cdr (a b c d e f)))
(terpri)
(write (cons a (b c)))
(terpri)
(write (list a (b c) (e f)))
(terpri)
(write (append (b c) (e f) (p q) ( ) (g)))
(terpri)
(write (last (a b c d (e f))))
(terpri)
(write (reverse (a b c d (e f))))
```

o/p –

```
A
B C D E F
(A B C)
(A (B C) (E F))
(B C E F P Q G)
(E F)
((E F) D C B A)
```

Date : Page :

- Predicates: Predicates are fun" that test their argument for specific conditions and returns nil if the cond" is false or some known nil value if the condition is true.

(atom name)	lessp	arrayp
equal	numberp	
evenp	Symbolp	
oddp	integerp	
zerop	floatp	
null.	complexp	
listp	characterp	
greaterp	stringp	

e.g. (write (atom 'abcd)); o/p → T
(terpri) // for new line
(write (equal a b)) o/p → NIL
(terpri)
(write (evenp 10)) o/p → T
(terpri)
(write (evenp 7)) o/p → NIL
(terpri)
(write (oddp 7))
(terpri)
(write (zerop 0.000001))

- Conditional statements → i) if

```
(setq val 50)
(if (= val 50)
    (format t "equal to 50"))
(terpri)
```

```
(if (> val 75)
    (format t "greater than 50"))
(if (< val 150)
    (format t "less than 150"))
```

ii) Cond statement → Syntax (Cond (condition1 statement1)
(condition2 statement2)
:
(conditionn statement n))

```
(setq val 500)
Cond ((> val 200)
      (format t "greater than 200")
      ((form < val 200)
      (format t "less than 200")))
```

iii) when → (when (condition) (statement)

```
(setq num 50)
(when (= num 50)
  (format t "equal to 50"))
```

iv) Case → Syntax (case (Key-value)
((Key1) (statement1 ... statement n)
((Key2) (statement1 ... statement n)
:
((Keyn) (statement1 ... statement n)

e.g.

```
(setq val 2)
(case val)
(1 (format t "you selected 1")
(2 (format t "you selected 2")
(3 (format t "you selected 3")))))
```

* Loops in LISP →

1. Loop → Syntax – (loop (s.expression))

program –

```
(setq a 10)
(loop
   (setq a (+ a 1))
   (write a)
   (terpri)
   (when (> a 17)(return a)))
```

2. Loop for →

Syntax (loop for loop variable in <a list>
do (action)
)

program loop for loop-variable from values to value
do (action))

e.g. (loop for x in (tom dick harry)
do (format t "~S" x)
)

o/p → TOM DICK HARRY

ii) (loop for a from 10 to 20
do (print a)
)

o/p → 10 11 12 20

iii) loop for x from 1 to 20
if (evenp x)
do (print x)
(terpri)
)

o/p
2 12
4 14
6 16
8 18
10 20

o/p → 2 4 6 8 10 12 14 16 18 20

~ → To print new line

3. do → Syntax - (do ((variable1 value1 updated value1)
(variable2 value2 updated value2)
⋮
)
(test return_value)
(s-expression)
)

e.g. (do ((x 0 (+ 2 x))
(y 20 (- y 2)))
((= x y) (- x y))
(format t "~% x=~d y=~d" x y)
)

o/p → x=0 y=20
x=2 y=18
x=4 y=16
⋮
x=8 y=12

4. do-times → Syntax - (do-times (n 11)
e.g. (print n) (print (* n n))
)

o/p → 0 0
1 1
2 4
3 9
⋮
11 121

5. ~~loop~~ dolist → e.g. (dolist (n (1 2 3 4 5 6 7 8 9)
(format t "~% number : ~d
(terpri) Square: ~d" n (* n n))
)

o/p → number : 1 Squar : 1
number : 2 Square : 4
⋮ ⋮
number : 9 Square : 81

Date: Page:

* Property list :- A property list consists of entries where every entry consists of key called an indicator and value called a property.

For e.g. let us have a person object. We would like this person object to have properties like name, height, weight, address etc.

When a symbol is created its property list is initially empty. Properties are created using get with setf form.

program — (write (setf (get 'book title) '(A I))
(terpri)
(write (setf (get 'book 'author) '(Rich and Knight))
(terpri)
(write (setf (get 'book 'publisher) '(Tata McGraw
)))

- get →

Various property list fun^n allows us to assign properties as well as retrieve, replace or remove the prop. of

- get function → Syntax

get symbol indicator

e.g. (get 'book 'title)

get search the plist for an indicator equivalent to indicator. If found value is returned or else the default is returned.

- Setf fun^n → Syntax — setf((get function property/value)

Setf is used with getting to create a new property value pair.

(1)

- Symbol - Plist

Symbol

- remprop →

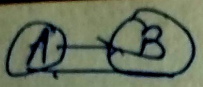
remprop
the indi

- getf → Synt

getf Sear
indicator
then the
default is

- remf →

remf
given indicat



Date : Page :

- Symbol-Plist → Syntax - [Symbol-Plist Symbol]

Symbol-Plist allows to see all the properties of a Symbol

- remprop → Syntax - [remprop Symbol indicator]

remprop fun^n is used to remove the prop. equivalent to the indicator.

- getf → Syntax - [getf place indicator]

getf searches the property list stored in place for an indicator equivalent to an indicator. If 1 is found then the corresponding value is returned otherwise the default is returned.

- remf → [remf place indicator]

remf removes the property equivalent to the given indicator from the given place.

Unit - II

Problem Solving

Date: Page:

- Problem Solving in AI → Problem solving particularly in AI may be characterised as a systematic search through a range of possible action in order to reach some predefined goal or solution.

A [Given initial situation] → What can I do to move from A to B? → B [Desired Goal situation]

That is problem solving is a process of designing and carrying out a set of steps to reach a goal.

- Production System → A production system is a tool used in AI and specially within the applied AI domain known as expert systems. It is a good way to describe the operations that can be performed in a search for a solution of a problem. Production consists of two parts: A precondition (IF statements) and an action (THEN). If a production precondition matches the current state then the production is trigger. If production action is executed it is said to have fired. The productions are rules of the form A → B where LHS is known as condition and the RHS is known as action.

A → B
↓ ↓
Condition Action

- State Space Searching :→ A set of all possible states.
- Control Strategy → Control strategy by which we come to know which rule is to be applied next during the process of reaching for a solution to a prob

(4,y) x<4 0<x+y≤4, y>0
(x,3) y<3 0<x+y≤3, (
(0,y) x>0 0<x+y≤4, y≥
(x,0) y>0

Control Strategy
- It should cau
- It should be

* Water - Jug P

Set of rules

1. Fill 4 gal
2. Fill 3 gal
3. Empty 4
4. Empty 3 g
5. Pour water
6. Pour water
7. Pour all wa
8. Pour all

4gal	x	jug
	0	
	0	
	3	
	3	
	4	
	0	
	2	

(4,y) $x<4$ $0<x+y\geq 4$, $y>0$ $(4, y-(4-x))$
(x,3) $y<3$ $0<x+y\leq 3$, $(x-(3-y)$
(0,y) $x>0$ $0<x+y\leq 4$, $y\geq 0$
(x,0) $y>0$

Date: Page:

Control Strategy requirements
- It should cause motion.
- It should be systematic.

* Water - Jug Problems:

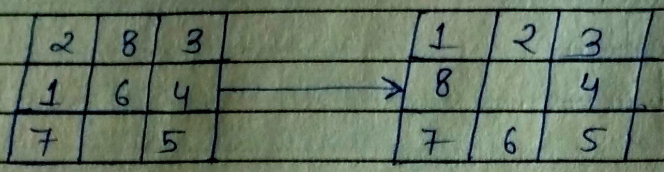
Set of rules

1. Fill 4 gal jug $(x, y) \xrightarrow{x<4} (4, y)$
2. Fill 3 gal jug $(x, y) \xrightarrow{y<3} (x, 3)$
3. Empty 4 gal jug $(x, y) \xrightarrow{x>0} (0, y)$
4. Empty 3 gal jug $(x, y) \xrightarrow{y>0} (x, 0)$
5. Pour water from 3 gal jug to fill 4 gal jug
   $(x, y) \longrightarrow (4, y-(4-x))$
   $0<x+y\geq 4$ and $y>0$
6. Pour water from 4 gal jug to fill 3 gal jug
   $(x, y) \longrightarrow (x-(3-y), 3)$
   $0<x+y\geq 3$ and $x>0$
7. Pour all water from 3 gal jug into 4 gal jug
   $(x, y) \longrightarrow (x+y, 0)$
   $0<x+y\leq 4$ and $y\geq 0$
8. Pour all water from 4 gal jug into 3 gal jug
   $(x, y) \longrightarrow (0, x+y)$
   $0<x+y\leq 3$ and $x\geq 0$

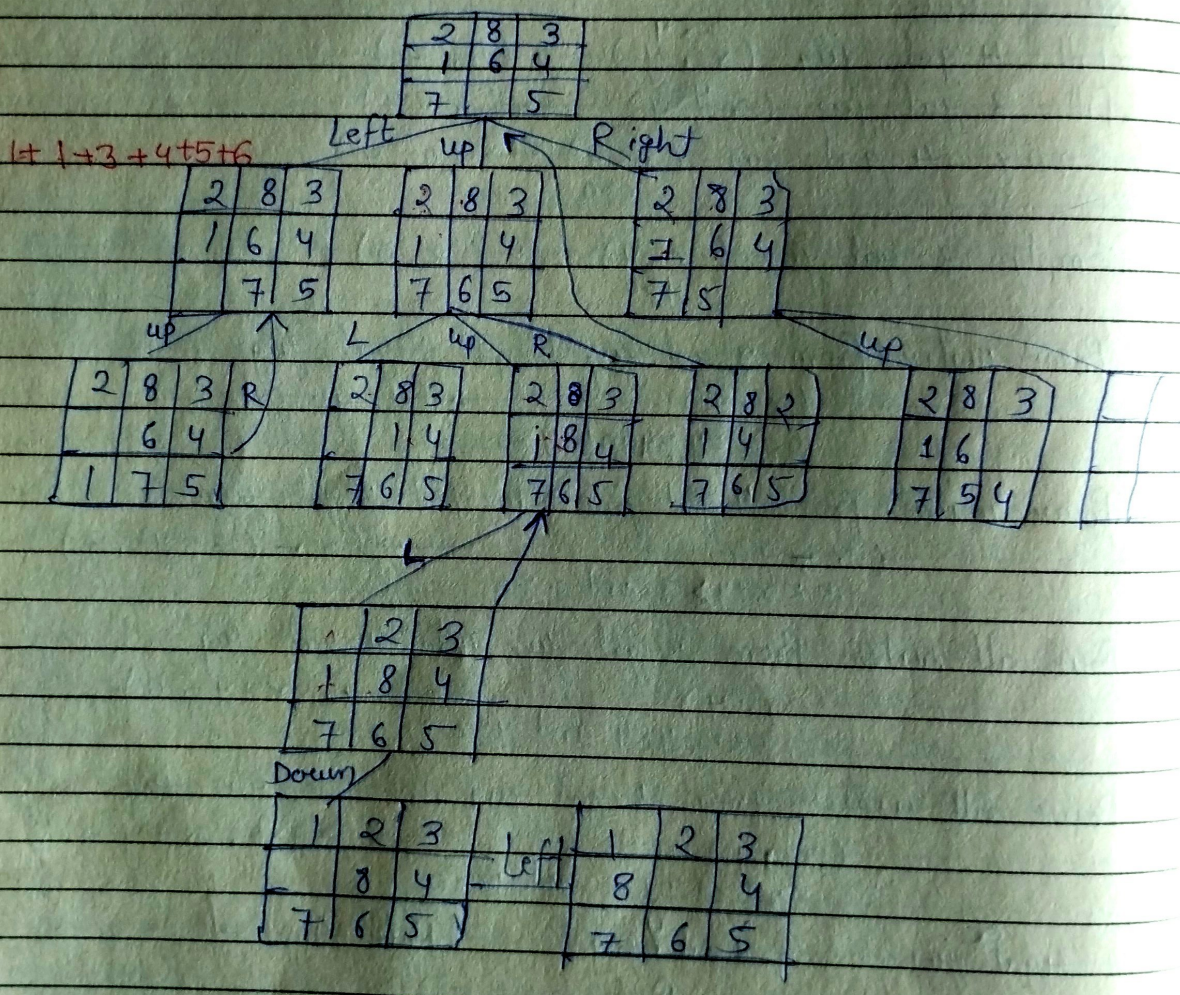
4gal x jug	3gal y jug	Rule Applied
0	0	
0	3	2
3	0	7
3	3	2
4	2	5
0	2	3
2	0	7

Date : Page :

• 8- Puzzle Problem



Initial Goal



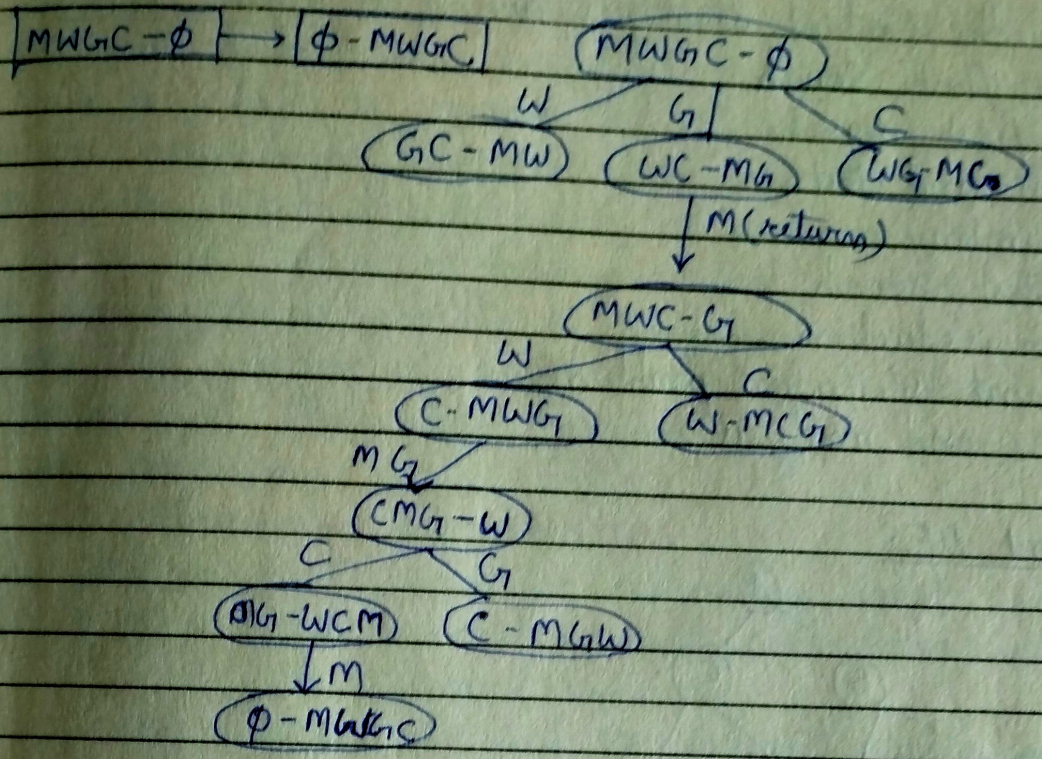
Man- M
Wolf - W
Cabbage - C
Goat - G

MWGC -

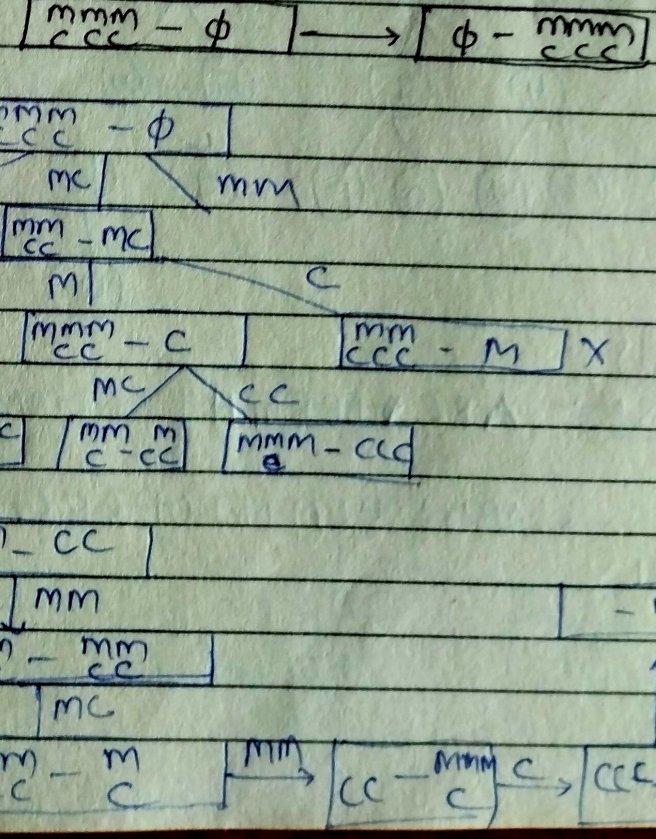
• 3 miss
3 Cannib

Man - M
Wolf - W
Cabbage - C
Goat - G

Date: Page:



- 3 missionaries
3 cannibals

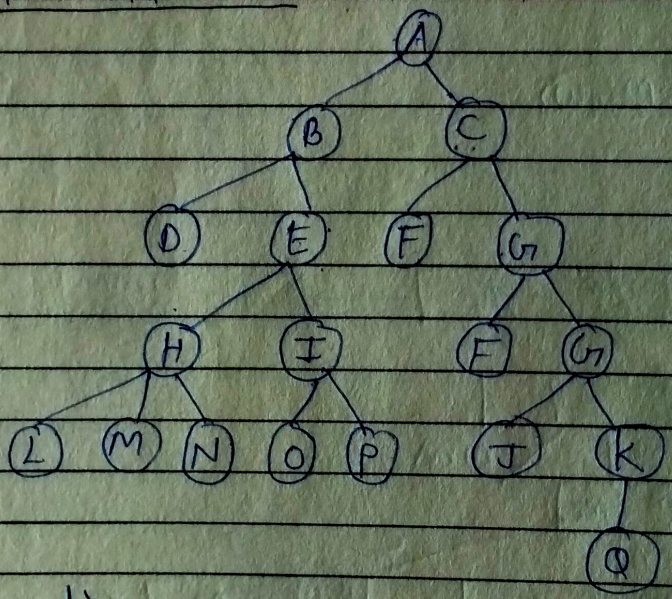


Some of the Control strategies are

1. Breadth first Search
2. Depth first Search
3. Generate and test
4. Hill Climbing
5. Best First Search
6. Problem reduction
7. Constraint Satisfaction problem
8. Mean end analysis

- BFS and DFS

ABDEHLMNIOPCEGFGJKQ



(level)

BFS → A B C D E F G H I F G L M N O P J K Q

DFS → A B D E H L M N I O P C E G F G J K Q
(RL)

Date: Page:

• **Heuristic Search** → In order to solve a problem with large search space some knowledge that are domain specific must be added to improve the efficiency of search. Heuristic means an advice that is often effective but we can not guarantee that it will work in each and every case.

It is a technique that improve the efficiency of search process possibly by ~~sacrifying~~ sacrificing to completeness. The purpose of heuristic fun^n is to guide the search process in the most profitable path among all that are possible.

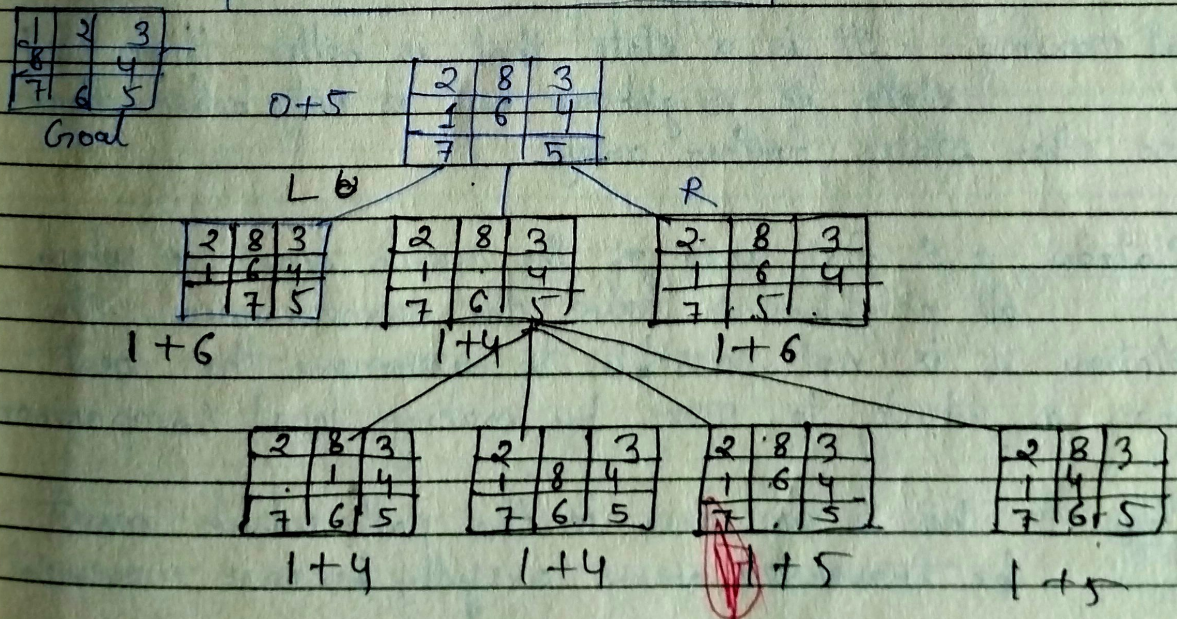
Heuristic function

$f(n)$ = fun^n used for estimating least cost

$g(n)$ = This estimates least cost from initial state to current node n.

$h(n)$ = This is used to estimate least cost from current node to goal state / node.

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

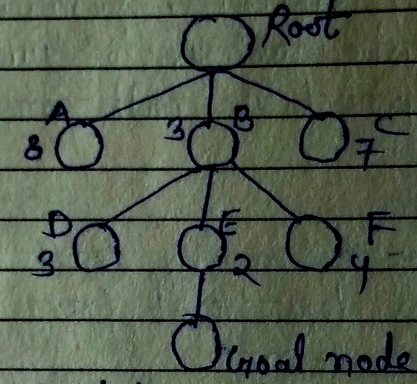


Date : Page :

Local Search Algorithm

- **Hill climbing** :- It is also called optimization algo, uses simple heuristic function, the amount of distance node is from the goal state.

In fact there is practically no difference between hill climbing and depth first search except that the children of the node that has been updated.



* Problems with hill climbing -
  i) local maxima / maximum
  ii) Plateau
  iii) Ridge

i) local maxima → It is a state that is better than all ~~state~~ its neighbour but is not better than some other states further away.

ii) Plateau → A flat area of the search space in which all neighbours have the same value. On a plateau it is not possible to determine the best direction in which to move by making local comparis

iii) Ridge → This is an area in the path which must be traversed very carefully because moveme

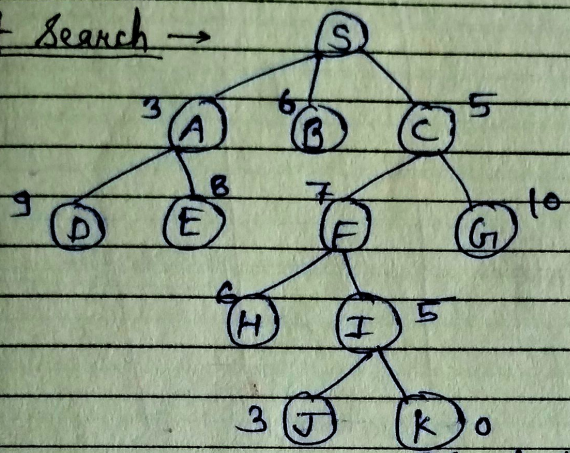
in any
or res

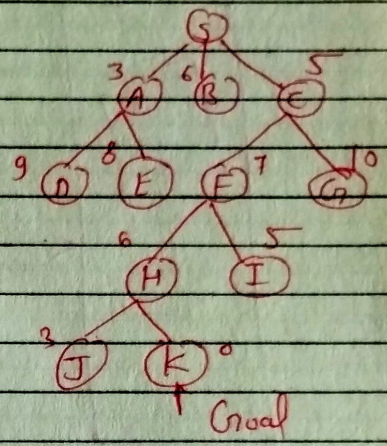
• Best Fir

step No.
1
2
3
4
5
6

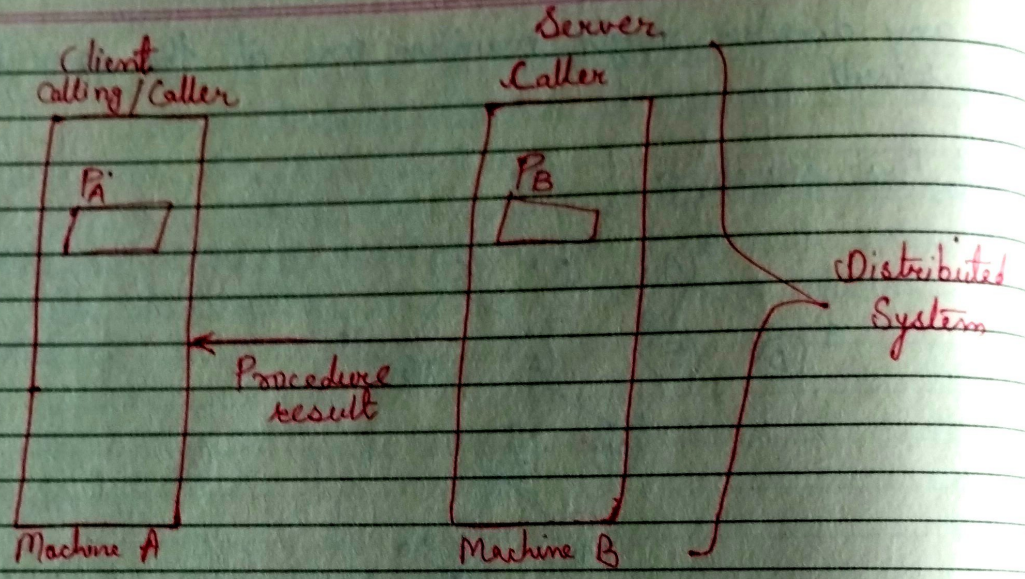
in any direction might maintain one at the same level. as result.

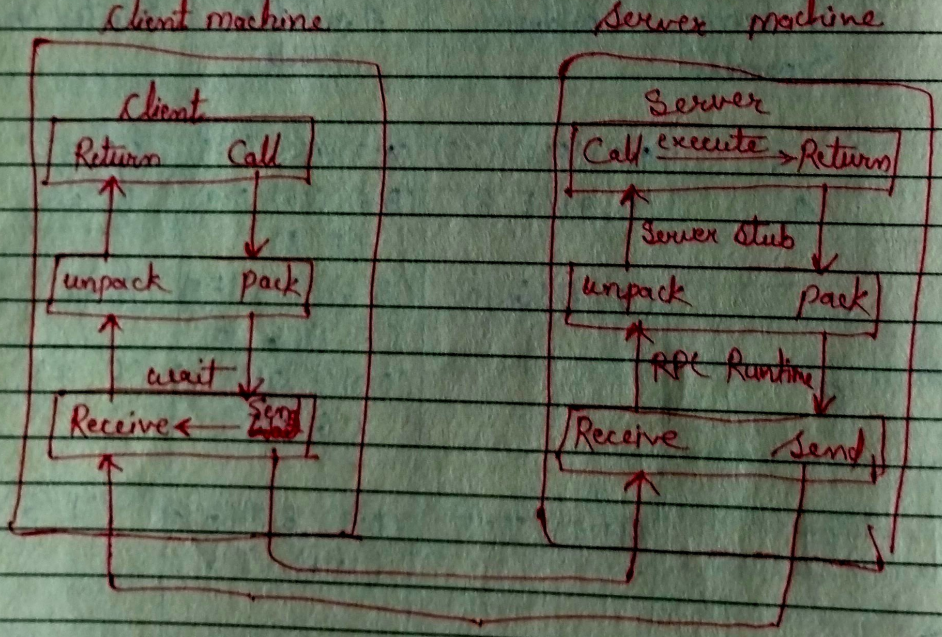
• Best First Search →





Step No.	Node being Expanded	Childrem	Available Node	Selected Node
1	S	(A:3)(B:6)(C:5)	(A:3)(B:6)(C:5)	(A:3)
2	A	(D:9)(E:8)	(B:6)(C:5)(D:9) (E:8)	(C:5)
3	C	(F:7)(G:10)	(B:6)(D:9)(E:8) (F:7)(G:10)	(B:6)
4	B	—	(D:9)(E:8)(F:7) (G:10)	(F:7)
5	F	(H:6)(I:5)	(D:9)(E:8)(G:10) (H:6)(I:5)	(I:5)
6	I	(J:3)(K:0)	(D:9)(E:8)(H:6) (G:10)(J:3)(K:0)	(K:0) ↑Goal Search Stop


Client
calling / Caller
P_A
Procedure result
Machine A
Server
Caller
P_B
Distributed System
Machine B

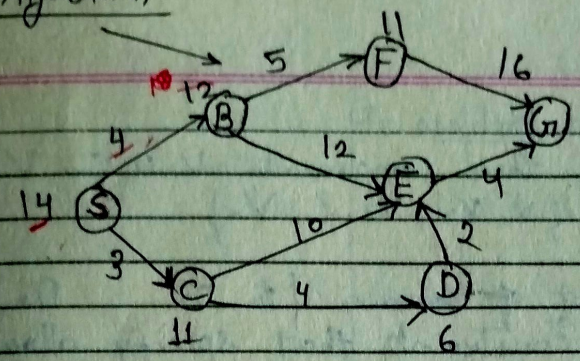

Client machine
Client
Return
Call
unpack
pack
wait
Receive
Send
Server machine
Server
Call
execute
Return
Server Stub
unpack
pack
RPC Runtime
Receive
Send

Distributed Systems

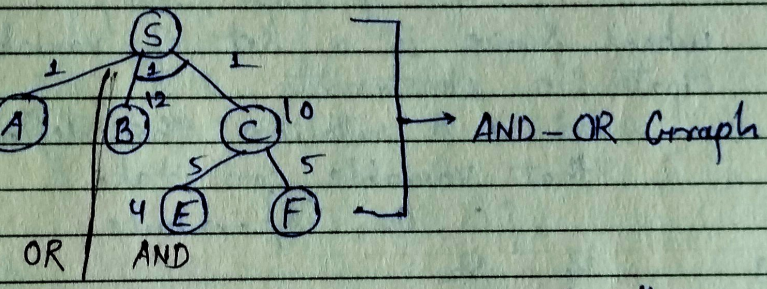
A* Algorithm

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

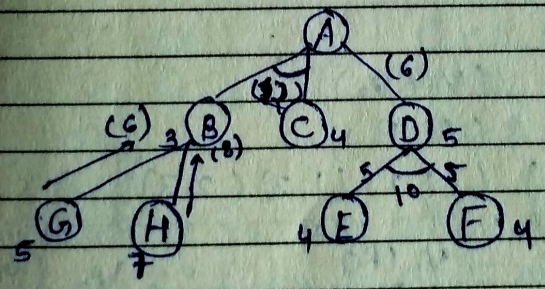
S → 14+4 = 18
14+3 = 17
S →

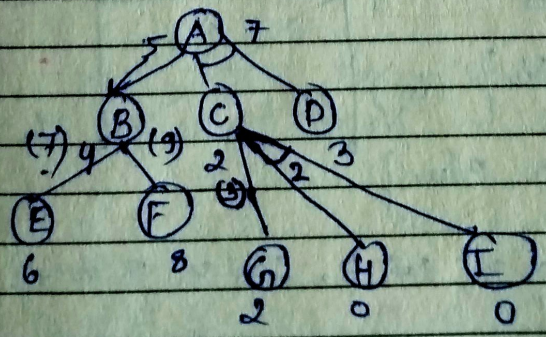


- **AO* Algorithm →**



* AO* Algorithm gives a optimal solution.





constraint

- Constant Satisfaction Problem —

→ CSP consists of three components - V, D, C. V is a set of variable $\{V_1, V_2, V_3 \ldots V_n\}$.

D = is a set of domain $\{D_1, D_2 \ldots D_n\}$

C = is a set of constraint that specify allowable combination of values.

$C_i = \{ \text{Scope}, \text{rel} \}$

where Scope is a set of variables that participates in constraints.

→ rel is a relation that defines the values that variable can take.

$\{C_1, C_2, C_3\}$ → rule

e.g.

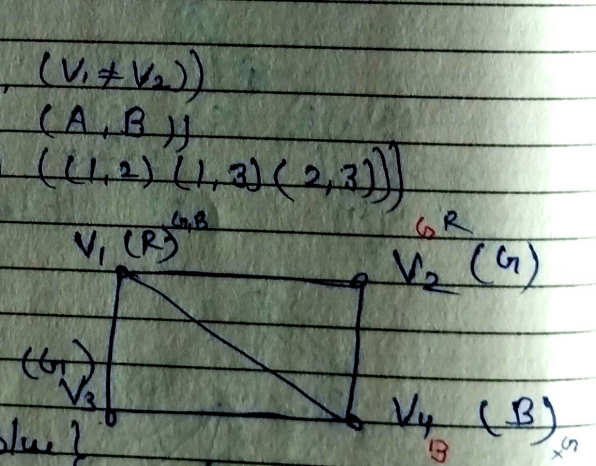
V_1	V_2
A	B
(1,2)	(2,3)

$C_1 = ((V_1, V_2), (V_1 \neq V_2))$

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

$C_1 = ((V_1, V_2) \ ((1,2)(1,3)(2,3)))$

* Algorithm for CSP



$V = \{V_1, V_2, V_3, V_4\}$

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

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

. V is a

D₂}
allowable

that part-

the values

	1	2	3	4	
	R, G, B	R, G, B	R, G, B	R, G, B	
	1=R	G, B	G, B	G, B	
	R	G	G, B	B	
	R	G	B G	B	Solution not found

G	R, B	R, B	R, B	B	R, G	R, G	R, G
G	R	R, B	B	B	R	R, G	G
G	R	R	B	B	R	R	G

- Cypt Arithmetic Problem –

T O	O-1	2 1
G O	T-2	8 1
OUT	G-8	1 0 2

Constraints → No two letters have same value.
→ Sum of digit must be so as shown in problem.
→ Digits than can be assigned to a alphabet must be b/w 0–9.
2, 3, 4, 5, 6, 7, 8

SEND	m-1	9 5 6 7
MORE	S-9	1 0 8 5
MONEY	O-0	
		1 0 6 5 2

E+O=N 6

5 4
8 8
.3 2

E + 0 = N
E N
0 R
N E

2, 3, 4, 5, 6, 7, 8
6
7
3

8
7
5

S E N D
9 5 6 7
M 1 0 0 R 8 E 5
M 1 0 0 6 N 5 E 2 4

S - 9
E - 5
N - 6
M - 1
R - 8
D - 7
O - 0
Y - 2

C R O S S
R O A D S
D A N G E R

D - 1
C - 9

C
8,9

9 6 2 3 3
6 2 5 1 3
1 8 5 8 7 4 6

R 5+5
6+6
7+7
8+8
9+9
R-4
S-8
R-6
E-0
C-9
A-5

6 + 0 = A
O + A = N

C + R
9 + 6 = 15

C R O S S
R O A D S
D A N G E R

S - 3
C - 9
R - 6
D - 1
A - 5
N - 8
G - 7
E - 4

A N G
U P S
C U P
C
U
K
P
P
X

2 ← P

Qua 2 D O N A L D
7 G E R A L D
R O B E R T

D + D → T
3 + 3 → 6

O N
0 B

Que 1
5 0

O + O → O
5 + 5 → 0
N + R → B
D + D → T
A → 2
N + R → B

5 O N A L D
G E R A L D
R O B E R T

F O R T Y
9 7 8 6
T E N
T E N
S I X T Y
3 1 4 8 6

R - 8
D - 7
O - 0
Y - 2

5 + 5
6 + 6
7 + 7
8 + 8
9 + 9
R - 4
S - 8
R - 6
E - 0
C - 9
A - 5

Que 2 DONALD + GERALD = ROBERT

D + D → T
3 + 3 → 6

A + A → E = 0
5 + 5 → 10
D + D → 10

A = 5 N = 2
D = 1 B = 8
R = 6 L = 3 G = 5

Que 1 A → 5, E → 0

D + D → T
D + G → R
D + G → R
2 D + D → T 4
1, 3, 6, 7, 8, 9
L + L → R
3 + 3 → 6

D → 5
E → 9
O →

T → 0
R → 7
E → 9
B → 3

FORTY + TEN + TEN = SIXTY

Que UPS + CUP = PUC

Ans 421

UNIT III

Que How to provide knowledge ? Data

Data that already present in the system or the data given itself can be converted into knowledge because data and knowledge are same. The only difference exist in terms of representation. The data can be converted into knowledge if we chang its representation, for which we have following technique for knowledge representation techniques.

1. Propositional logic
2. Predicate logic
3. Rules facts
4. Frames
5. Scripts
6. Semantic Nets
7. Conceptual Dependencies
8. Blackboard architecture

- Reasoning → Once the system have knowledge the second task is to provide it reasoning technique which includes
  1. Modes Ponen
  2. Chain Rule
  3. Resolution
  4. Refutation
  5. Backward chaining

* Knowledge technique

- Propositional technique (zero order logic)
  0 - False - F
  1 - True - T

1. T (True), F (False) are propositional logic formula.
2. A, B, C, D etc. (representation of various sentences) are propositional logic formulb.
3. Step 1 and 2 combine with following operators are also propositional logic formula

a. ¬ (negation)
b. ∧ (Conjuction)
c. ∨ (disjunction)
d. → (implication)
e. ↔ (equivalence)

a. ¬

T	F
F	T

b. ∧

T	T	T
T	F	F
F	T	F
F	F	F

d. →

T	T	T
T	F	F
F	T	T
F	F	T

e. ↔

T	T	T
T	F	F
F	T	F
F	F	T

- Rules or laws in propositional logic —

1. Commutative law
   i) (A∨B) = (B∨B)
   ii) A∧B = B∧A
   iii) A↔B = B↔A

- Associative law
  i) A∨(B∨C) = (A∨B)∨C
  ii) (A∧(B∧C) = (A∧B)∧C

- Distributive law
  i) A∨(B∧C) = (A∨B)∧(A∨C)
  ii) A∧(B∨C) = (A∧B) ∨(A∧C)

• Demorgan's law

1. $\neg(A \vee B) = \neg A \wedge \neg B$
2. $\neg(A \wedge B) = \neg A \vee \neg B$

• Law of Negation

$$\neg(\neg A) = A$$

• Law of excluded middle

$$A \vee \neg A = T$$

• Law of Contradiction $A \wedge \neg A = F$

• Law of implication

$$A \rightarrow B = \neg A \vee B$$

There are two methods of writing propositional logic formula

1. Conjuctive normal form (AND of OR's)

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

2. Disjuctive normal form (OR of AND's)

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

Any formula is said to be in CNF or DNF when it can only have three operators $\wedge$ $\vee$ and $\neg$. and $\neg$ is attached to single item.

$\neg(A \vee B) \wedge (C \vee D)$ is not CNF because $\neg$ is attached to group though it contains only conj^n, disj^n and $\neg$.

Limitation of propositional logic

→ In propositional logic we can write only limited set of english sentences but it is not powerful enough to represent general sentences.

e.g. P.L. is not work in following e.g.

1. All eng student are intelligent.
2. Amit is an intelligent eng. Student.

Que Is Amit

• Predicate Logic

operators and
are also used
following term

a. PREDICATES

return
→ Predic

→ x is

b. Functions

a predicate
e.g.

* c. Quatifiers
log
1. Univ
Sen
every

2. Existe

some

Que Is Amit Intelligent ?

Date : Page :

• Predicate Logic (First order) → Predicate logic is a superset of propositional logic such that all the operators and rules associated with propositional logic are also used in predicate logic predicate logic has following terms as operand in the formula.

a. PREDICATES — represent a sentences, whose values, ~~rule~~ will be true or False. Predicate should return in capital letters

→ Predicate logic can be written as

x is Greater than y

GREATER (x, y).

→ x is father of y.

FATHER (x, y)

b. Functions — They are also a part of predicate logic formula. Normally they are used within a predicate and always return non-boolean value.

e.g. age (x)

*c. Qualifiers → following extra o/p are used in predicate logic formula.

1. Universal qualifier (∀) → It is used when sentences contains words like for All, always everyone, everything etc.

2. Existential qualifier (∃) → It is used when sentences contain word like there exist, someone, sometimes, few, some etc.

1. Apple is a fruit.
FRUIT(Apple)

2. Sky is Blue e.g. BLUE(sky)

3. Deb is not tall
→TAL(Deb)

4. Ram is a teacher. Ans TEACHER(Ram)

5. Ram is a father of atul.
FATHER(ram, atul)

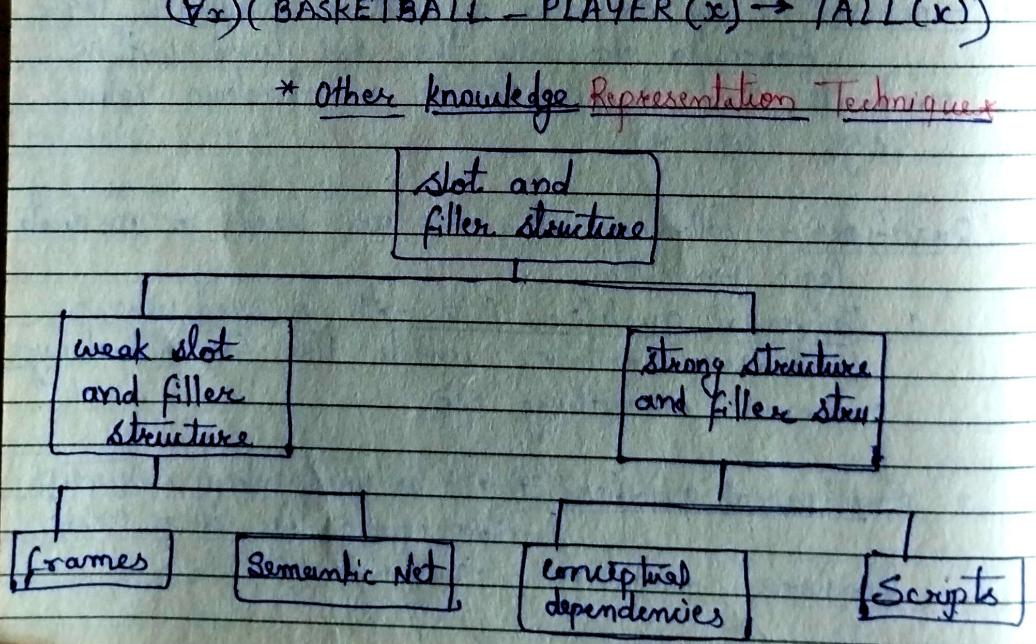
6. Ram is a father of atul and husband of gita
(FATHER(Ram, Atul) ∧ HUSBAND(Ram, Geeta))

7. father of John is married to tina's mother.
MARRIED(FATHER(John), MOTHER(Tina))

8. Fatima loves flower.
~~Flower~~(~~Fatima~~) Loves(fatima, Flower)

9. All basketball players are tall.
$(\forall x)$(BASKETBALL-PLAYER(x) → TALL(x))

* Other knowledge Representation Techniques



* weak slot → In weak slot and filler structure one can have many ways to represent same knowledge. There is no standardization of rules to represent knowledge.

• Frames → A frame is a collection of attributes (usually called slots) and associated values (and possibly contains on values) that describes some entity in the world.

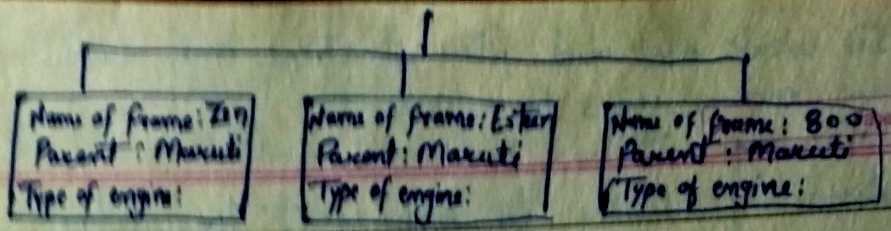
A frame provides a method of combining declaration and procedures within a single knowledge representation environment. The fundamental organising principles of frame system are the pacakageing of both data and procedure into common knowledge structure.

* Structure of a frame → Frame structure is mainly divided into four basic components –

1. Name of the frame.
2. Parent of the frame
3. The slots (attributes) and their values.
4. Attached predicates for different slot (optional)

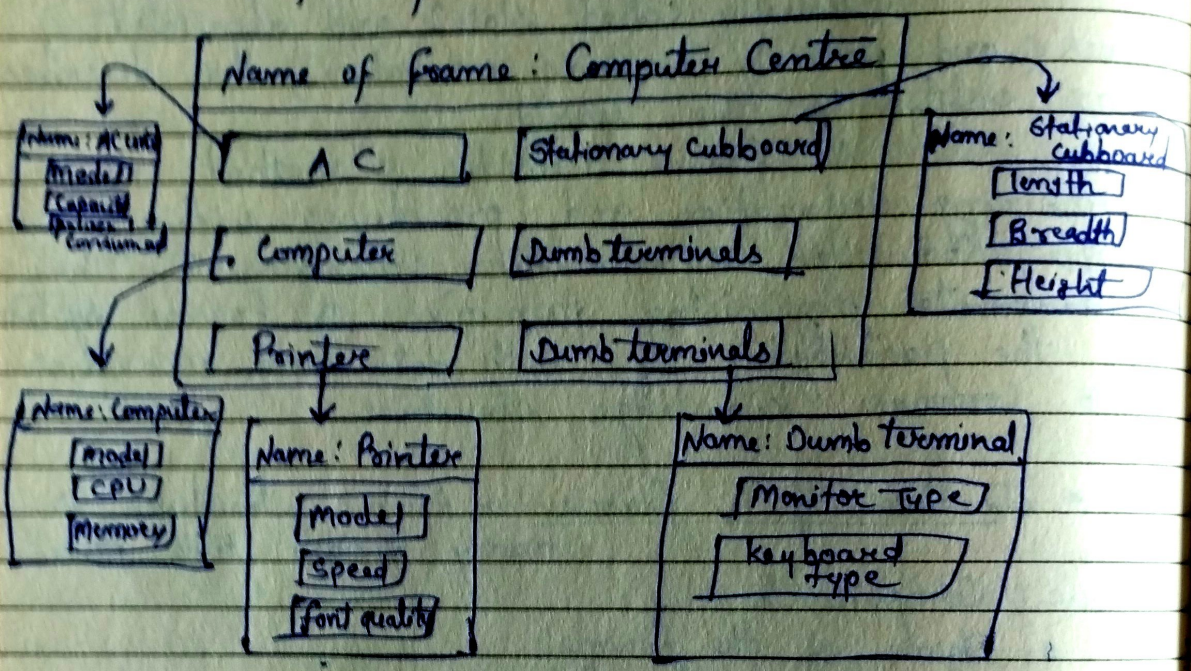
Name of frame	:	Car
Parent	:	Automobile
Wheels	:	Four
Steering	:	One
Head light	:	Two
⋮		⋮

Name of frame	:	Maruti
Parent	:	Car
Suspension	:	

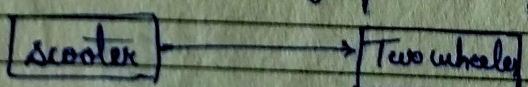


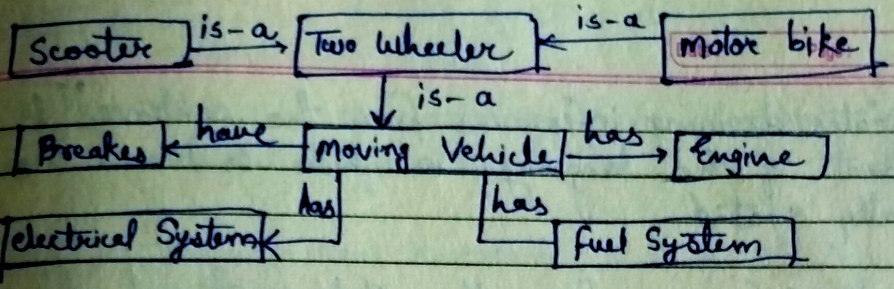
e.g.

Frame of Computer Centre



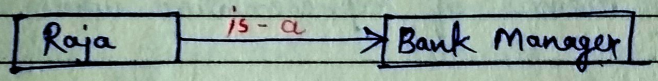
* Semantic Net In Semantic Nets information is represented as a set of nodes connected to each other by set of labelled arc which shows relationship (like is-a, has-a.....) among the nodes. Nodes represent entities, attributes, states or events and arc represents relationship between the nodes, labels on the arc specify type of relationship exist between the nodes. It can be viewed as graphical representation of knowledge.



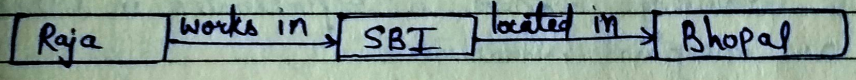


Que Give Semantic Net representation following Sentences:

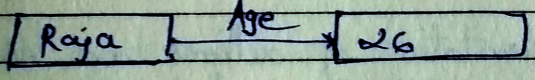
- Raja is a bank manager.
- Raja works in SBI located in Bhopal.
- Raja is 26 year old.
- Raja has blue eyes.
- Raja is taller than piyush.



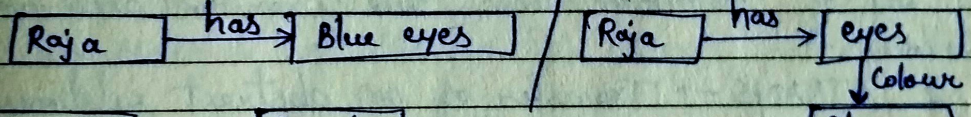
2)



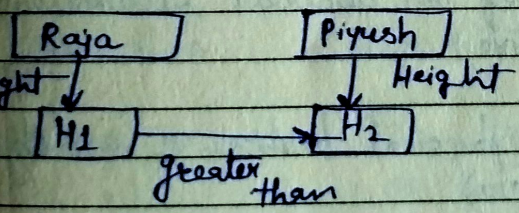
3)



4)



5)



* Conceptual dependency → Conceptual dependency is a theory of how to represent the kind of knowledge about events that is usually contained in natural language sentences. The main aim for the development of Conceptual dependency as a KRT are given below

- Facilitates drawing inferences from the sentences [is in-dependent of the language in which sentences were orginally stated
- To make inferences from the statement and also to identify conditions in which two sentences can have similare meaning
- To provide facilities for the systems to take part in dialogues and ans. que.
- To provide a means of representation which are lang. independent.

knowledge is represented in CD by elements called as conceptual structures.

In CD representation of actions are built from a set of primitive acts. These primitives acts are as follows:

1. ATRANS → Transfer of an abstract relationship. (e.g. Give)
2. PTRANS → Transfer of physical location of an object (e.g. Go)
3. MTRANS → Transfer of Mental information (e.g. tell)
4. PROPEL → Application of physical force to an object (e.g. Push)
5. MOVE → Movement of a body part by its owner. (e.g. kick)

6. GRASP → Grasping of an object by an actor. (e.g. clutch)
7. INGEST → Ingestion of an object by an animal. (e.g. eat)
8. EXPEL → Expulsion of something from the body of an animal. (e.g. cry).
9. MBUILD → Building new information out of old. (e.g. decide)
10. SPEAK → Production of sound (e.g. Say)
11. ATTEND → Focusing of a sense Organ toward a stimulus (e.g. listen)

A second set of CD building blocks is the set of allowable dependencies among the conceptualisation describe in a sentence. There are 6 primitive conceptual categories from which dependency structures can be build. These are.

1. PP → Object PiP (Picture Procedure)
2. ACTS (Action / Action done by an actor)
3. AA → Modifier of Action (action aider)
4. PA → Modifier of PP (picture aider)

Que John Rain. John $\overset{P}{\Longleftrightarrow}$ PTRANS PP $\Longleftrightarrow$ ACT

John is Tall John $\Longleftrightarrow$ height (> average)
PP $\Longleftrightarrow$ PA

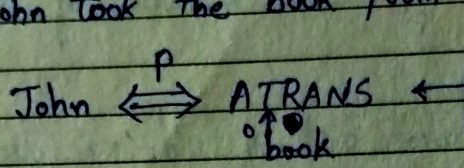
John is a doctor John $\Longleftrightarrow$ doctor (> average)
PP $\Longleftrightarrow$ PA

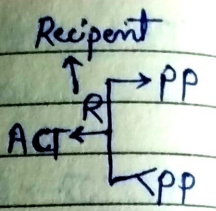
A nice boy Nice → Boy
PA → PP

John pushed the cart. → object case relation.

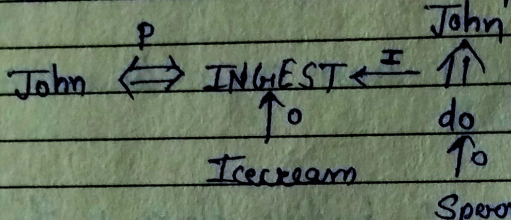
John ⟺ (P) PROPEL ← (O) Cart PP → ACT

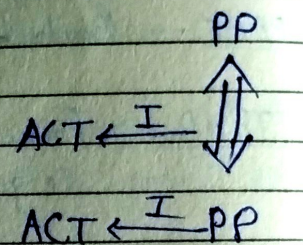
- John took the book from the mary



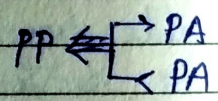


- John ate icecream with a spoon





- The Plants grew. plants → Size > x, Size = x



Yesterday

- John ran yesterday. John ⟺ (P) PTRANS

Scripts

The script is a structure that describe sequence of event in particular context. A script consist of set of slots some information may be associated with each slot about what kind of values it may contain as well as default value to be used if no other information is available.

Components —

1. Entry condition → Condition that must in general be satisfied before the events describe the script can occur.

Date : Page :

2. Result – Condition that will in general be true after the event describe in the Script have occured.

3. Props – Slots representing object that are involved in the events describe in the scripts.

4. Roles – Slots representing people who are involved in the events describe in the scripts.

5. Tracks – The specific variation on a more general pattern that is represented by this particular Script different tracks of the same Script will be share to many but not all components.

6. Scenes – The actual sequence of an event that occur. The event are represented in Conceptual dependency formalism.

Ex. Going to hostel
Going to bank for withdrawal
" movie (theatre)

Unit IV | Natural Language Processing (NLP)

NLP is one of the most interesting application of AI. In NLP we develop programmes that can understand natural language.

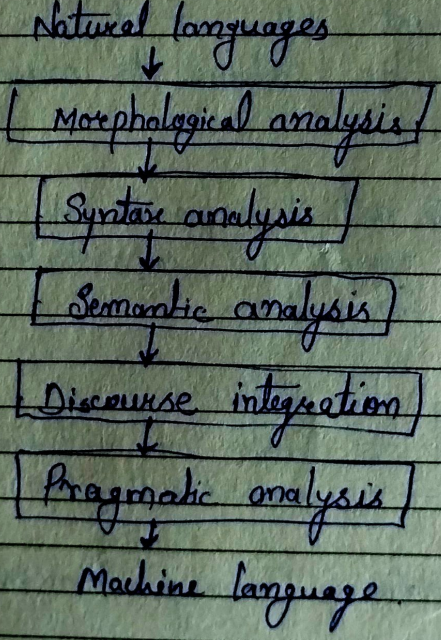
NLP Unit IV

Date : Page :

In NLP we use natural language for instructing computer to perform a task and naturally we have to use a translator that can convert natural language into machine language.

Natural language → NLP → Machine language

Working → Its working is very similar to that of Compiler like first it checks the validity of words then it checks its syntax of sentences. It works in 5 phases.



Morphological Analysis → It assumes program as a collection of words. It divide the paragraph into word and check their validity for which it uses hash tables. for

for e.g. "tghy" → Although it contain all valid alphabets but still this ~~word~~ word is not valid.

Syntax Analysis → It assume paragraph as a collection of sentences. It divide paragraph into sentences and check their grammer (Syntax).
for eg. → "Table this is"
Although it contain all valid word but grammer is wrong.

Semantic Analysis → Its purpose is to check whether sentences of given paragraph are meaningfull or not. for e.g "this table is eating" — Although it contains all valid words, grammer is right but still it is not meaningful.

Discourse integration → In NL meaning of individual sentence is based on sentences that preceeds it. for ex. — Suresh wants it.
— He loves flowers.
— Rahul went there.
In above sentences the value of it, he, there must be related to sentence that preceeds them. Discourse Integration place the value of such words from previous sentences.

Pragmatic Analysis → Its purpose is to get actual meaning of the sentences those are not clear like "Comeback if you will not well".

Five rules for simple subset of english grammer are given below —

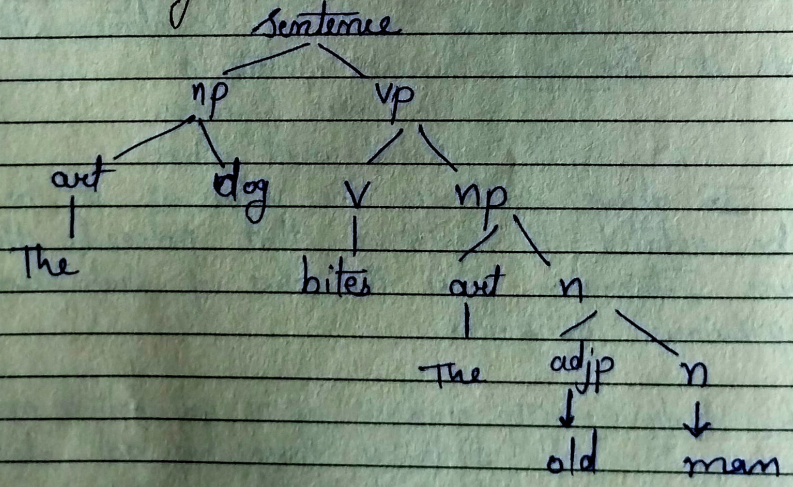
i) Sentence → np vp
ii) np → n
iii) np → art n
iv) vp → v
v) vp → v np

v → verb
np → noun phrase
vp → verb phrase
art → article

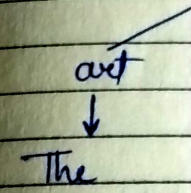
Que1. Add rules for adjective and adverb to the above grammer.

n → adjp n
adjp → adj
adjp → adj. np.
where adjp stand for adjective phrase.

Que2 Give a phrase tree for the sentences
The dog bites the oldman.



Que3 The boy



Game playing require much need is the of winning or

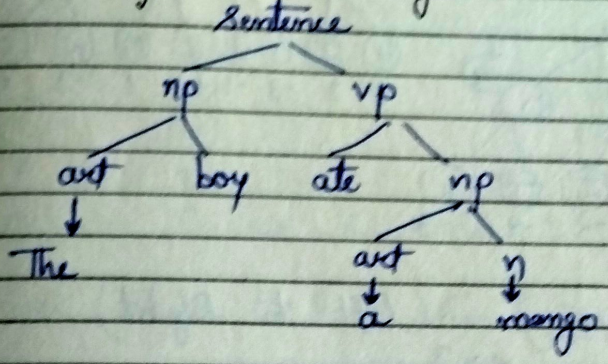
→ Searching for this. So searchi we need

Algorithms

minimax al

max →
min →

Ques The boy ate a mango.



Game playing

Game playing is a domain of AI. Games don't require much knowledge, the only knowledge we need is the rule, legal moves, and the condition of winning or loosing the game.

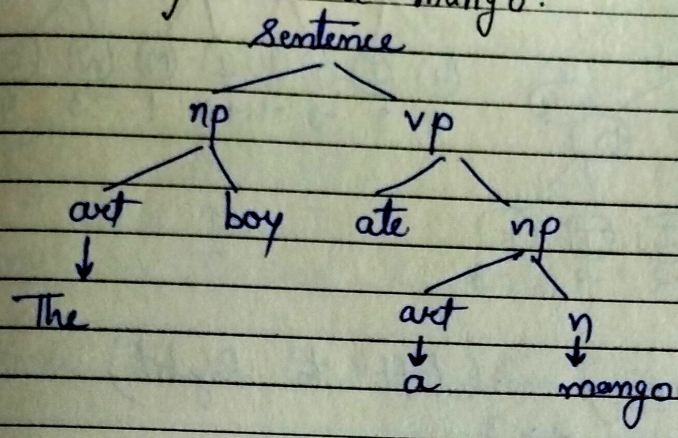
→ Searching techniques like BFS are not accurate for this as the branching factor is very high, So searching will take a lot of time. So we need another search procedure.

Algorithms : 1) Minimax algo.
2) alpha beta pruning / cut off

- minimax algo. / Search

max → max will try to maximize the utility.
min → try to minimize utility.

Ques The boy ate a mango.



Game Playing

Game playing is a domain of AI. Games don't require much knowledge, the only knowledge we need is the rule, legal moves, and the condition of winning or loosing the game.

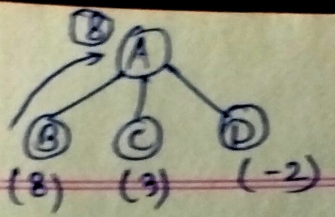
→ Searching techniques like BFS are not accurate for this. as the branching factor is very high, So searching will take a lot of time. So we need another search procedure.

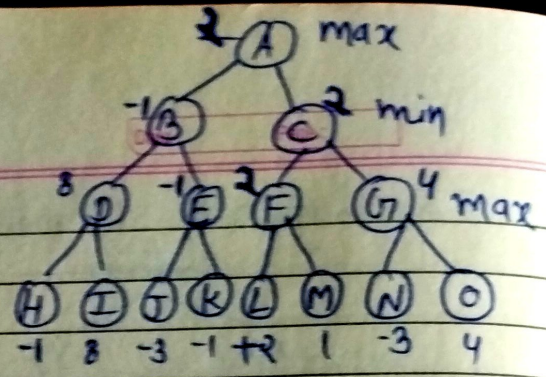
Algorithms : 1) Minimax algo.
2) alpha beta pruning / cut off

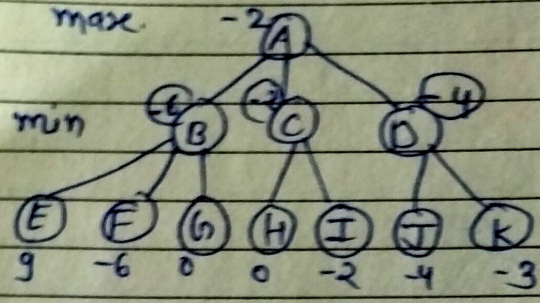
• minimax algo. / Search

max → max will try to maximize the utility.
min → try to minimize utility.

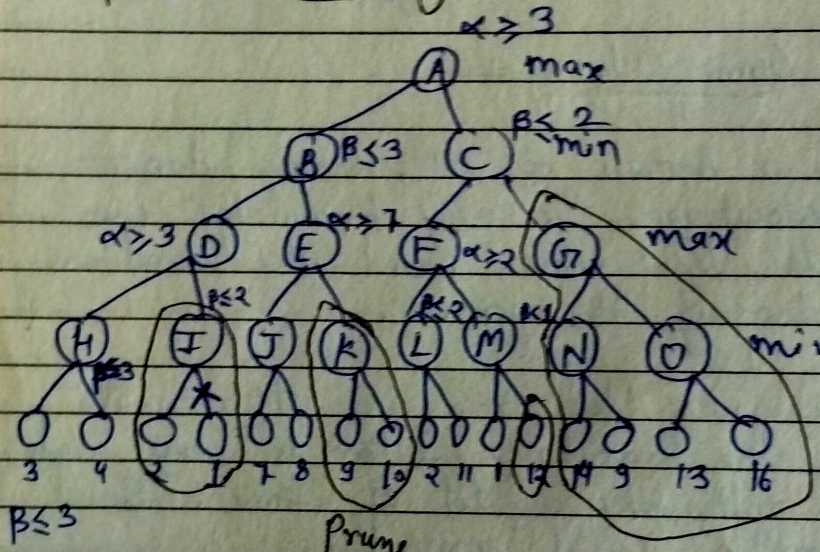
eg.

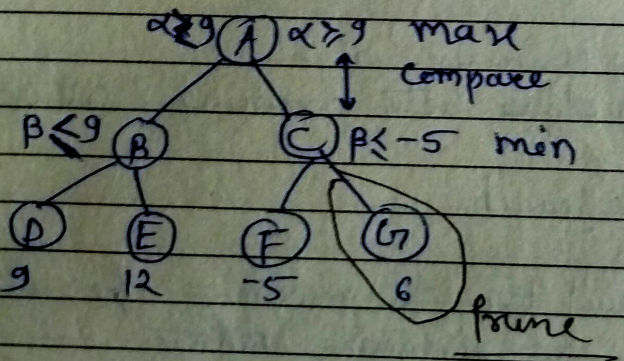


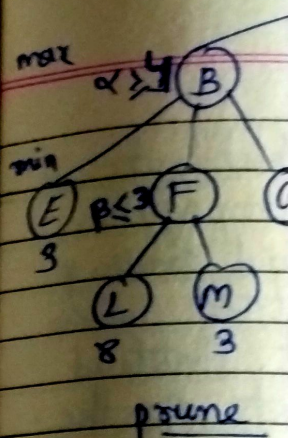




• Alpha Beta Pruning — (left to Right)







prune

Hence

max

min

β≤

max

α≥7

L

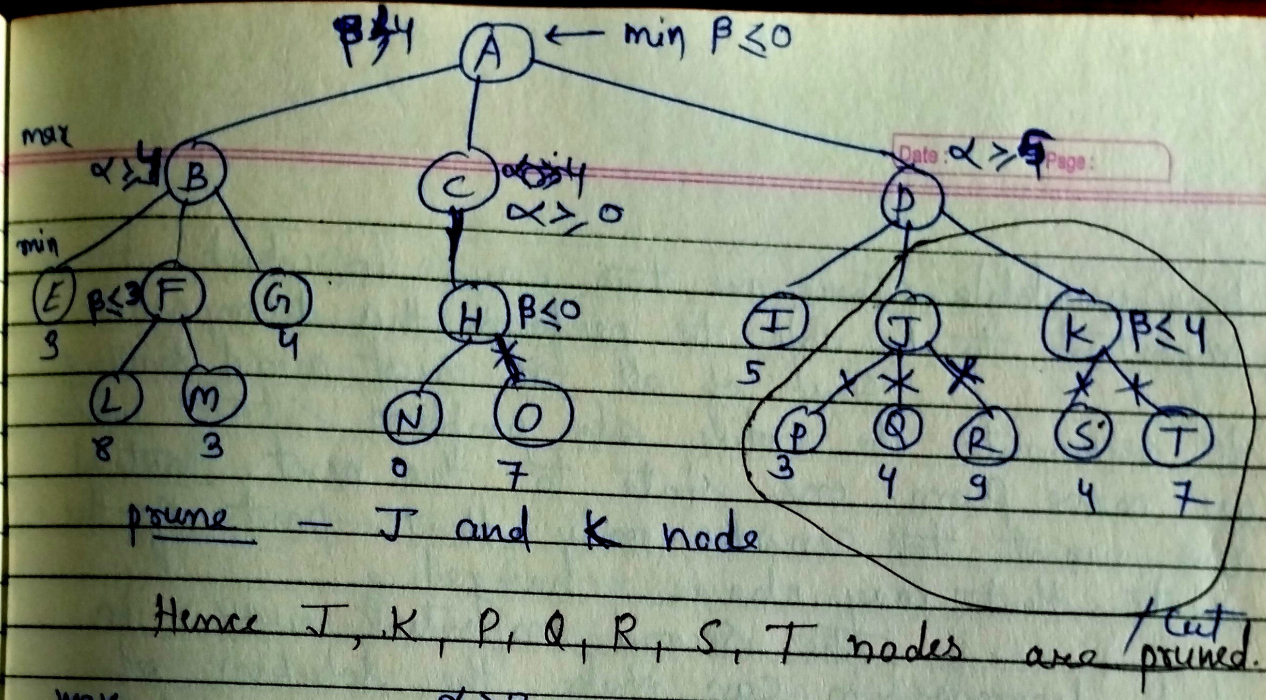
M

7

6

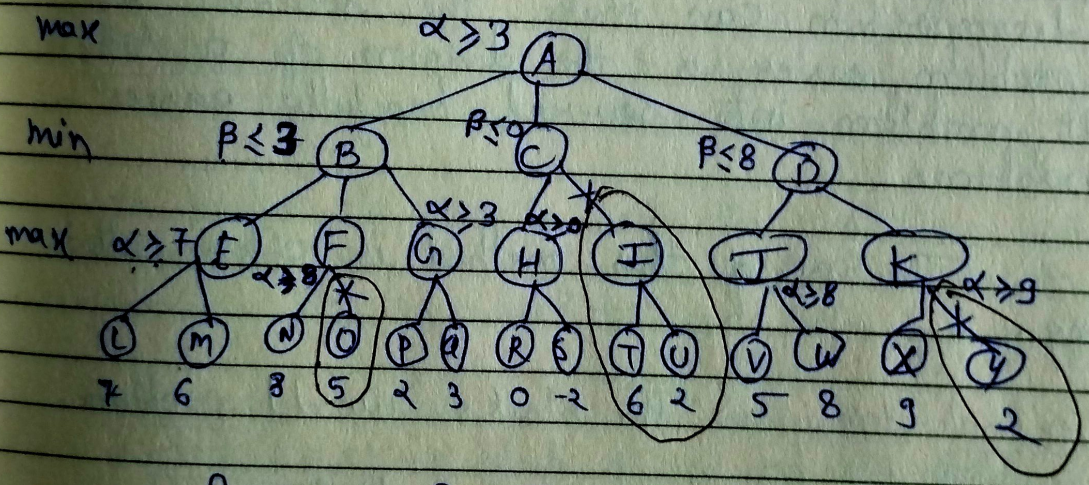
Pru

• For more
to be ab
separately
at the en
Unless we
of the stat
becomes to



prune – J and K node

Hence J, K, P, Q, R, S, T nodes are pruned.



Prune – O, I, T, U, Y

Planning

- For more complicated problem domains it becomes important to be able to work on small peace of problem separately and then to combine the partial solution at the end into a complete problem solution. Unless we can do this the no. of combination of the state of the components of a problem becomes to huge to handle in the amount of

time available. There are two ways in which it is important to be able to perform this decomposition.

First of all we must avoid having to recompute the entire ~~state~~ problem state when we move from one state to the next. Instead we want to consider only that part of the state that may have changed.

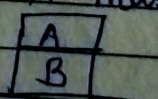
The second important way in which decomposition can make the solution of hard problem easier is the division of a single difficult problem into several, hopefully easier sub problems.

The use of this method is often called planning.

An example domain → The blocks world problem.

There is a flat surface on which blocks can be placed. There are no. of squares blocks are all the same size. They can be stacked one upon another. There is a robot arm that can manipulate the blocks. The actions it can perform include:

- UNSTACK (A, B) — pickup block A from its current position on block B. The arm must be empty and block A must have no blocks on top of it.



Date : Page :

- STACK (A, B) – Place block A on block B. The arm must already be holding A and the surface of B must be clear.

- PICKUP (A) → Pickup block A from the table and hold it. The arm must be empty and there must be nothing on top of block.
- PUTDOWN (A) → Put block A down on the table. The arm must have been holding block A

In order to specify both the conditions under which an operation may be performed and the result of performing it, we need to use following predicates -

- ON (A, B) → block A is on block B.
- ONTABLE (A) → block A is on the table.
- CLEAR (A) → There is nothing on top of block A.
- HOLDING (A) → The arm is holding block A.
- ARMEMPTY → The arm is holding nothing.

Components of planning system –

- In problem solving system based on the elementary techniques it is necessary to perform each of the following fun^{n} —

- choose the best rule to apply next based on

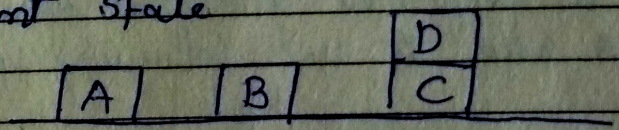
Date : Page :

the best available heuristic ~~function~~ information.

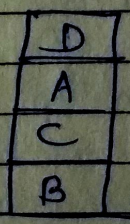
- Apply the choosen rule to compute the new problem state that arises from its application.
- Detect when a solution has been found.
- Detect dead ~~ends~~ ends so that they can be abandoned and the systems effort directed in more fruitful directions.

Block World Problem –

Current state



Goal state



Soln – The fun" used to define the current states are –

1. ONTABLE(A)
2. CLEAR(A)
3. ONTABLE(B)
4. CLEAR(B)
5. ONTABLE(C)
6. ON(D, C)
7. CLEAR(D)
   ARMEMPTY

Date : Page :

The functions (actions) that robot performs to solve this problem are

1. UNSTACK (D, C)
2. ~~PICKUP (~~
2. PUTDOWN (D)
3. PICKUP (C)
4. STACK (C, B)
5. PICKUP (A)
6. STACK (A, C)
7. PICKUP (D)
8. STACK (D, A)

* Condition of goal state

Rule —
- ONTABLE (B)
- ON (C, B)
- ON (A, C)
- ON (D, A)
- CLEAR (D)

ARMEMPTY