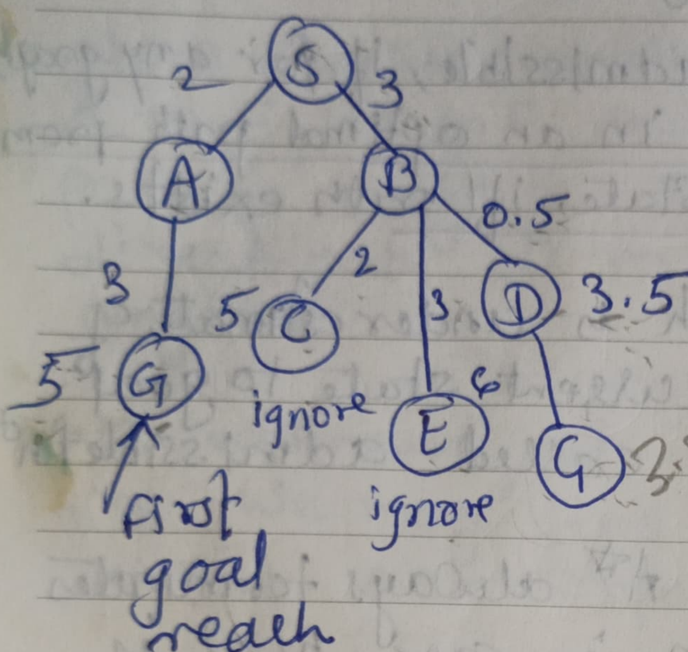
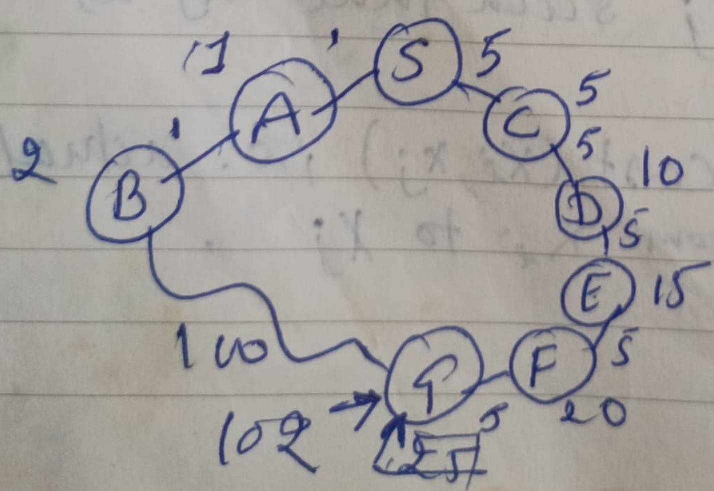
The Branch & Bound Principal.



- Use any (complete) search method to find a path.
- Remove all partial path that have an accumulate cost larger or equal than the found path
- continue search for the next path.
- ~~Ignore~~ Iterate
- Change the termination condition:
  → Only terminate when a path to a goal node has become the BESTPATH



Optimal uniform cost version.

1. QUEUE ← Path only containing the root;

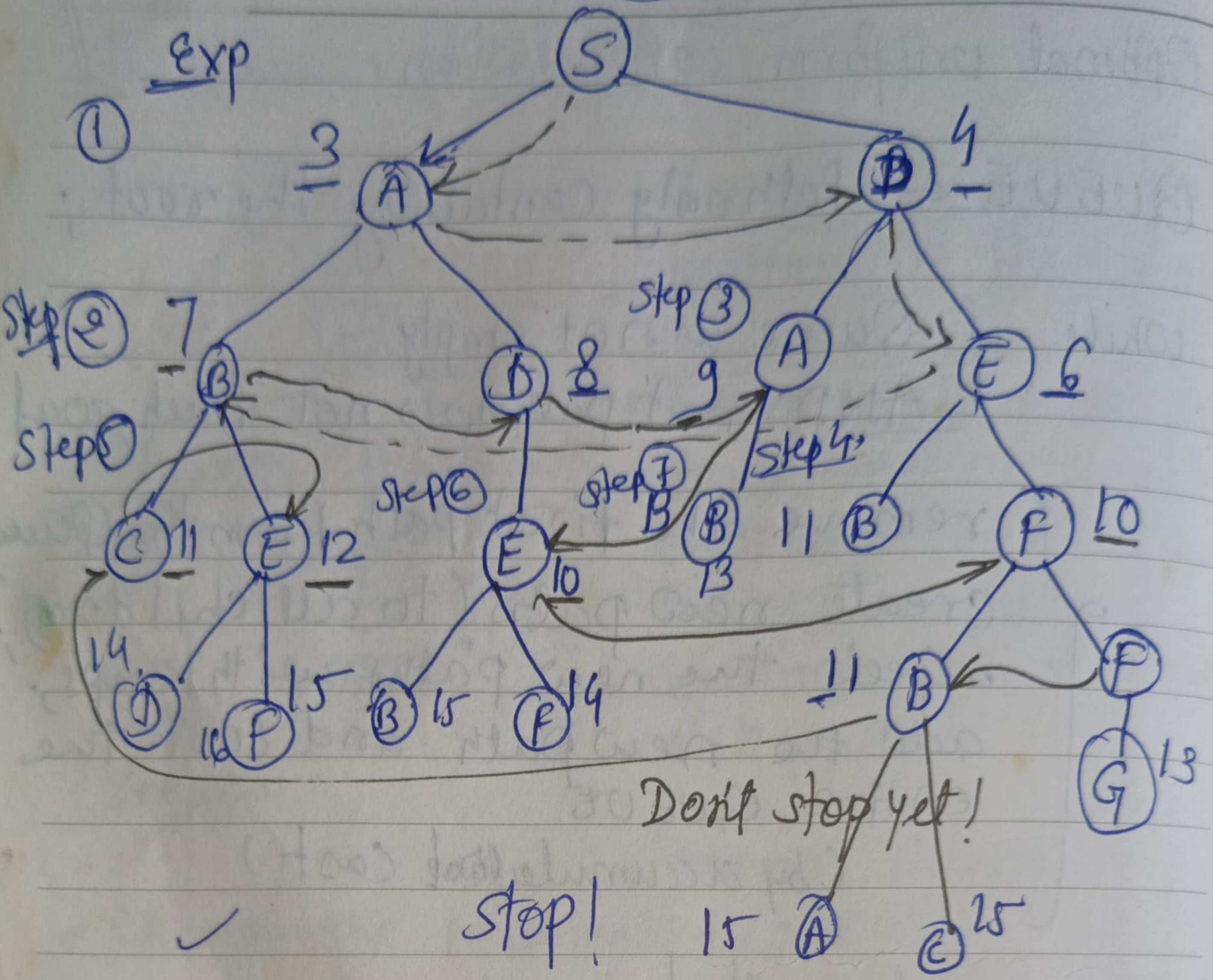
2. while { Queue is not empty
AND first path does not reach goal

{ remove the first path from the Queue;
create new paths (to all children);
reject the new paths with loops;
add the new path and sort the
entire QUEUE
(by accumulated cost)

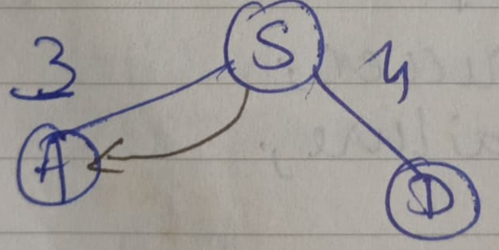
3. If goal reached
THEN success;
Else failure;

Exp

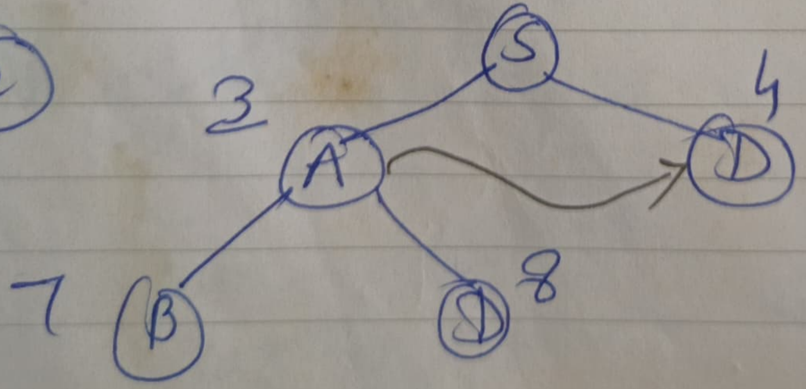
1


S
3
A
4
B
Step 2
7
B
D
8
Step 3
9
A
E
6
Step 5
Step 6
Step 7
B
Step 4
C
11
E
12
E
10
B
13
11
B
F
10
14
D
15
F
16
B
15
F
14
11
B
F
G
13
Don't stop yet!
Stop!
15
A
C
25

1


3
S
4
A
D
8

2

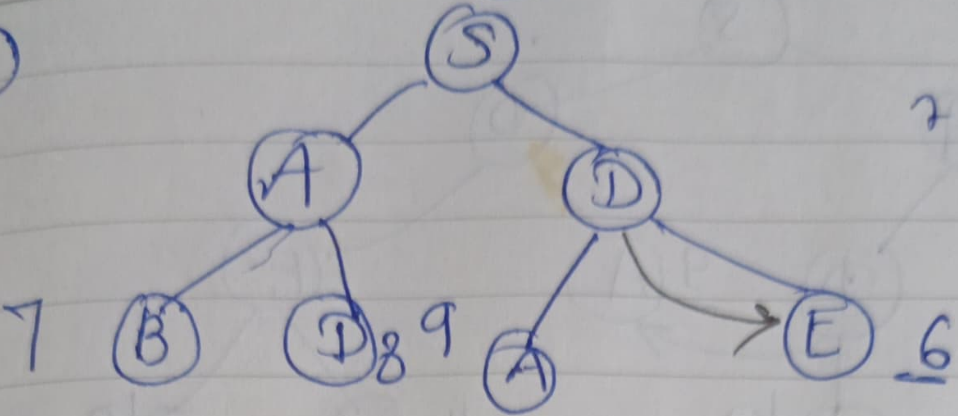

S
3
4
A
D
7
B
D
8

A D B D

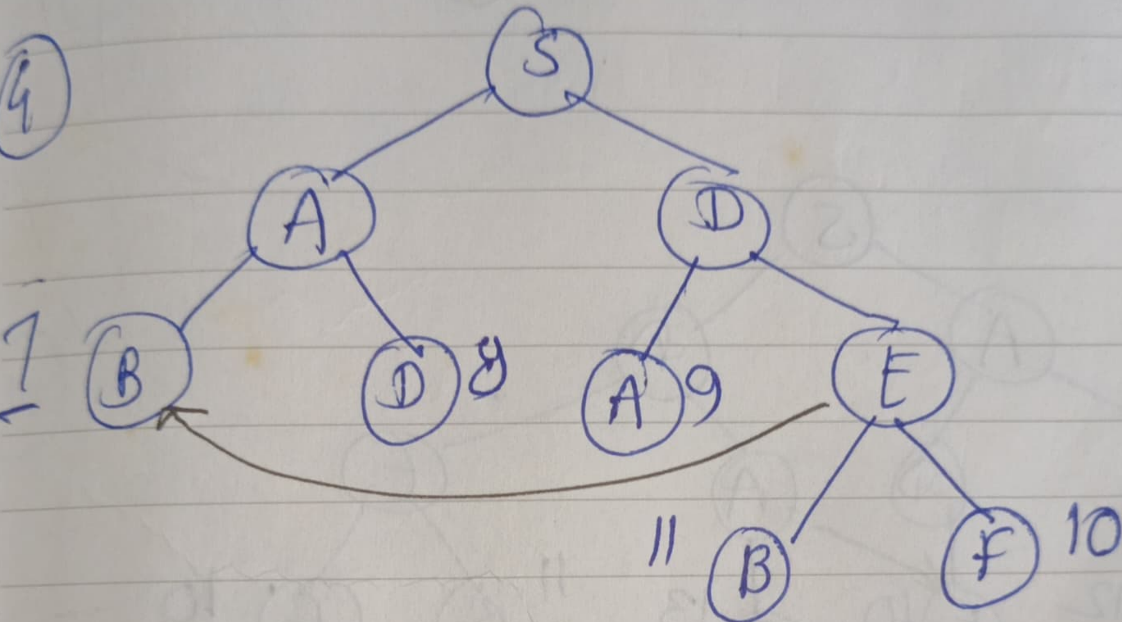
79

$^7B\ \overset{8}{D}\ A^9\ E^6$

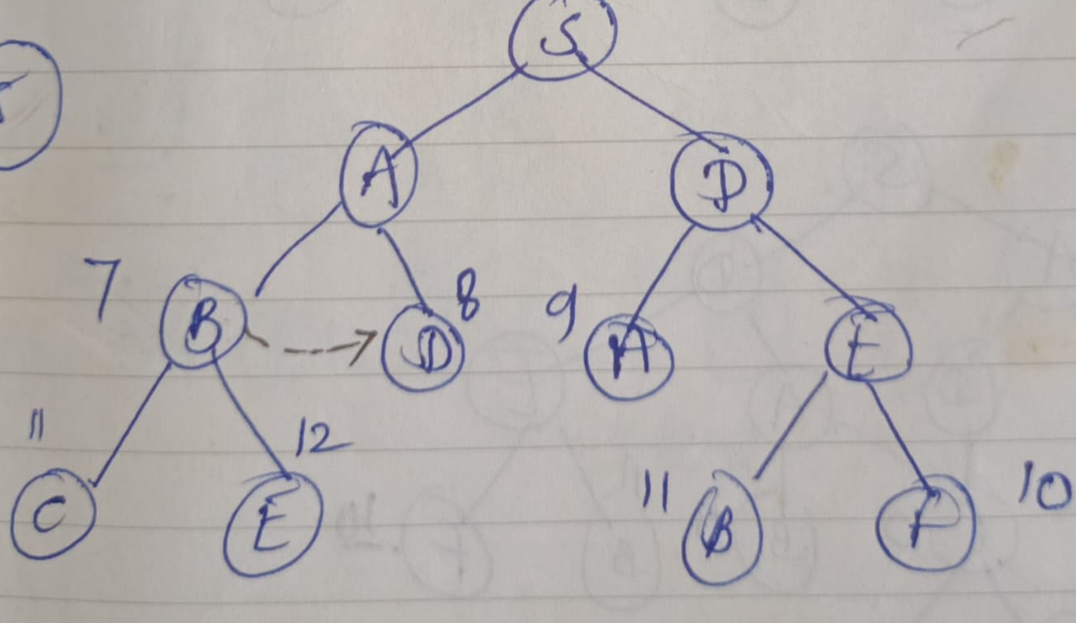
3



4

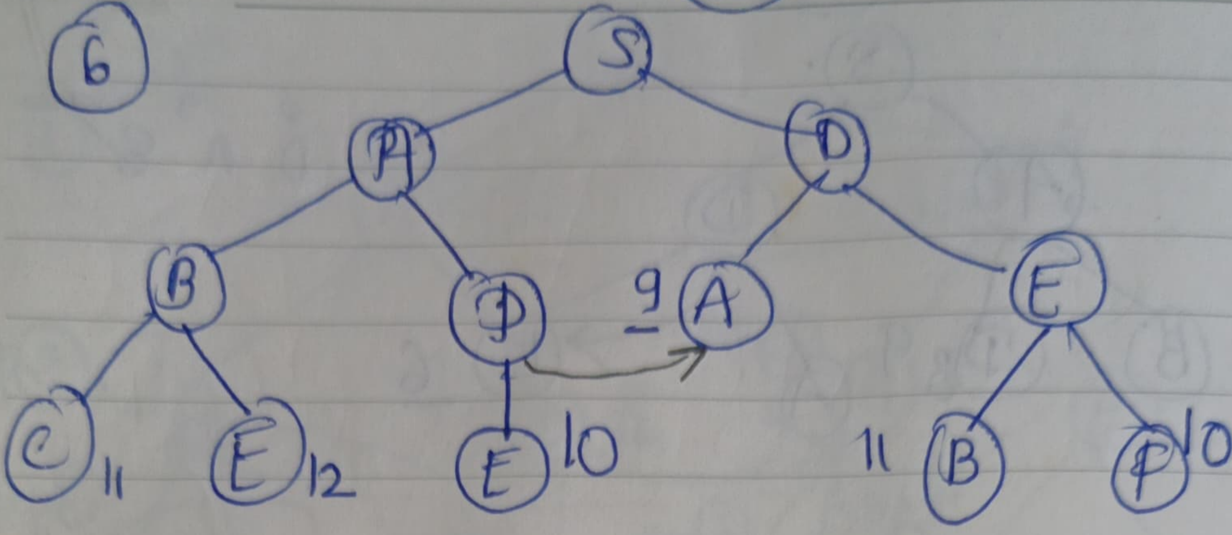


5



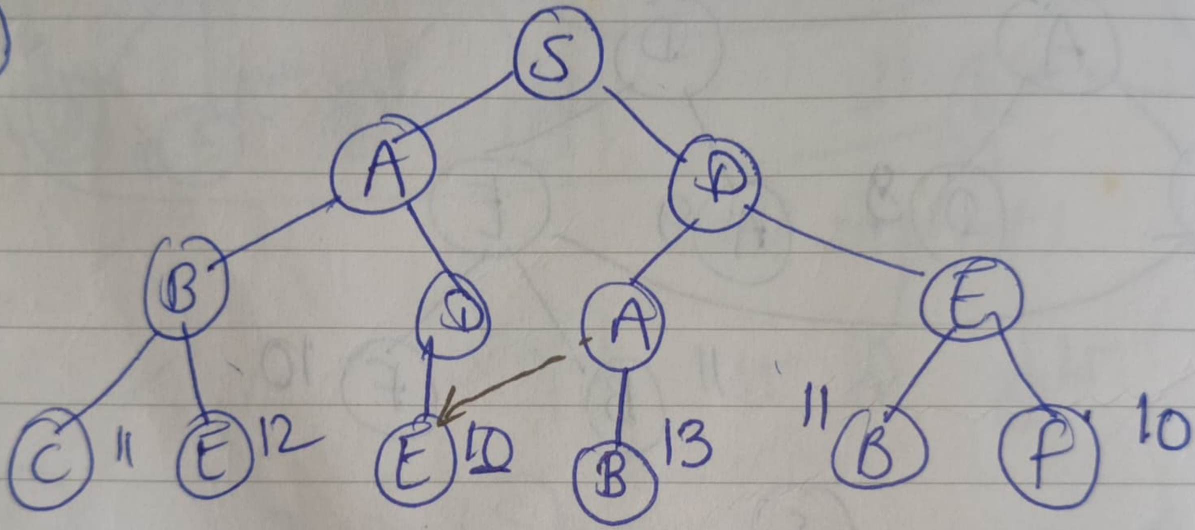
80

6



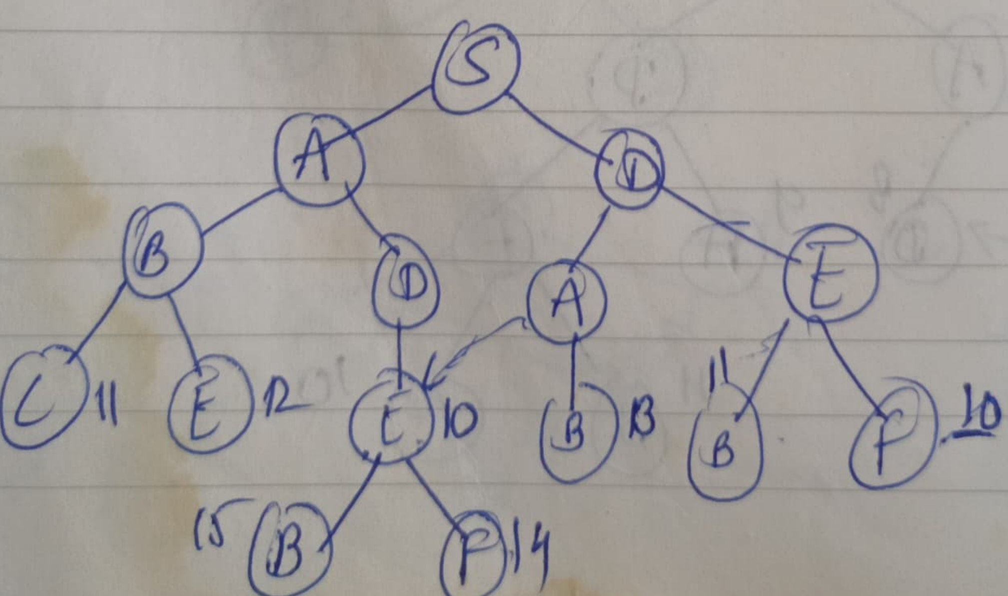
S
A
D
B
D
9
A
E
C
11
E
12
E
10
11
B
F
10

7



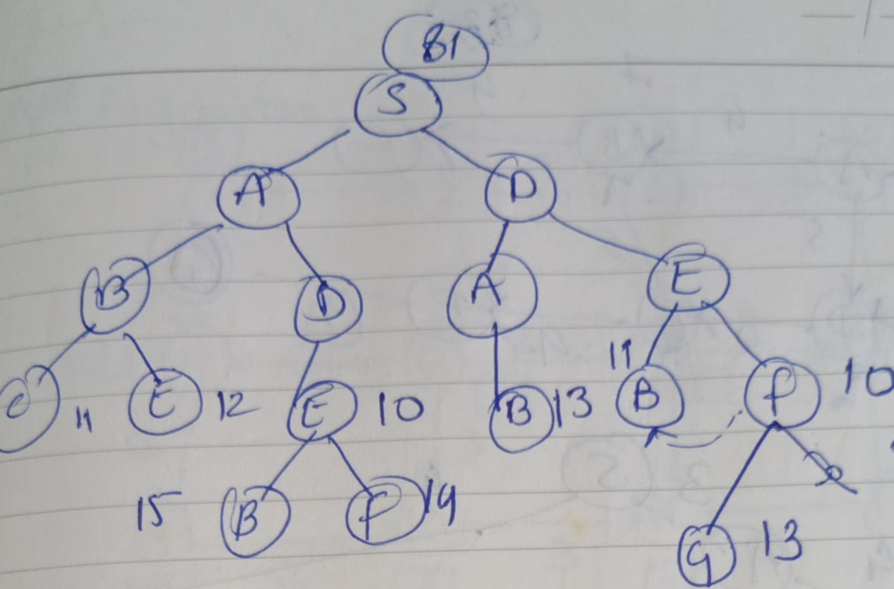
S
A
D
B
D
A
E
C
11
E
12
E
10
B
13
11
B
F
10

8



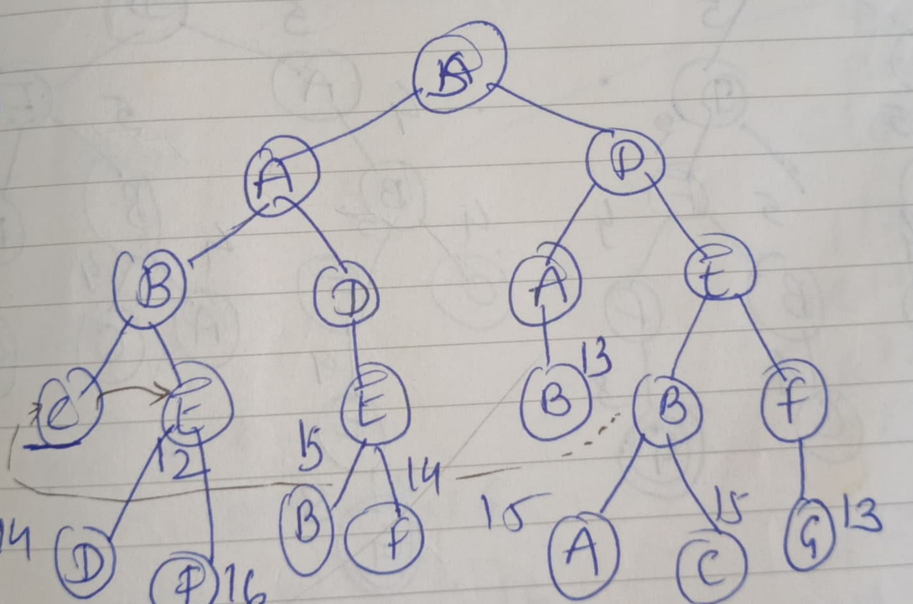
S
A
D
B
D
A
E
C
11
E
12
E
10
B
13
11
B
F
10
15
B
F
14

9



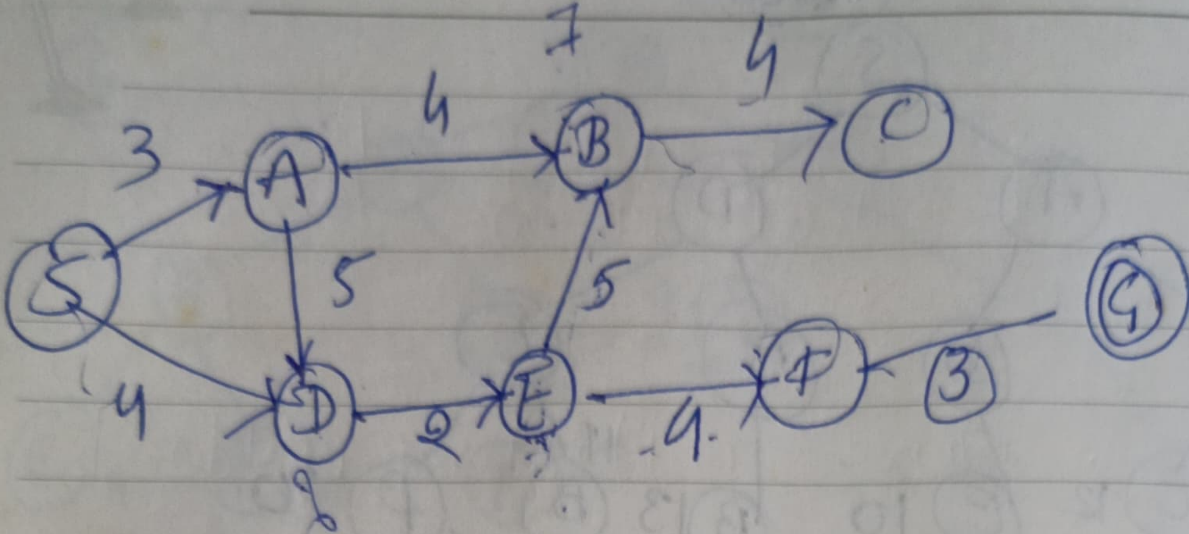
Don't stop yet!

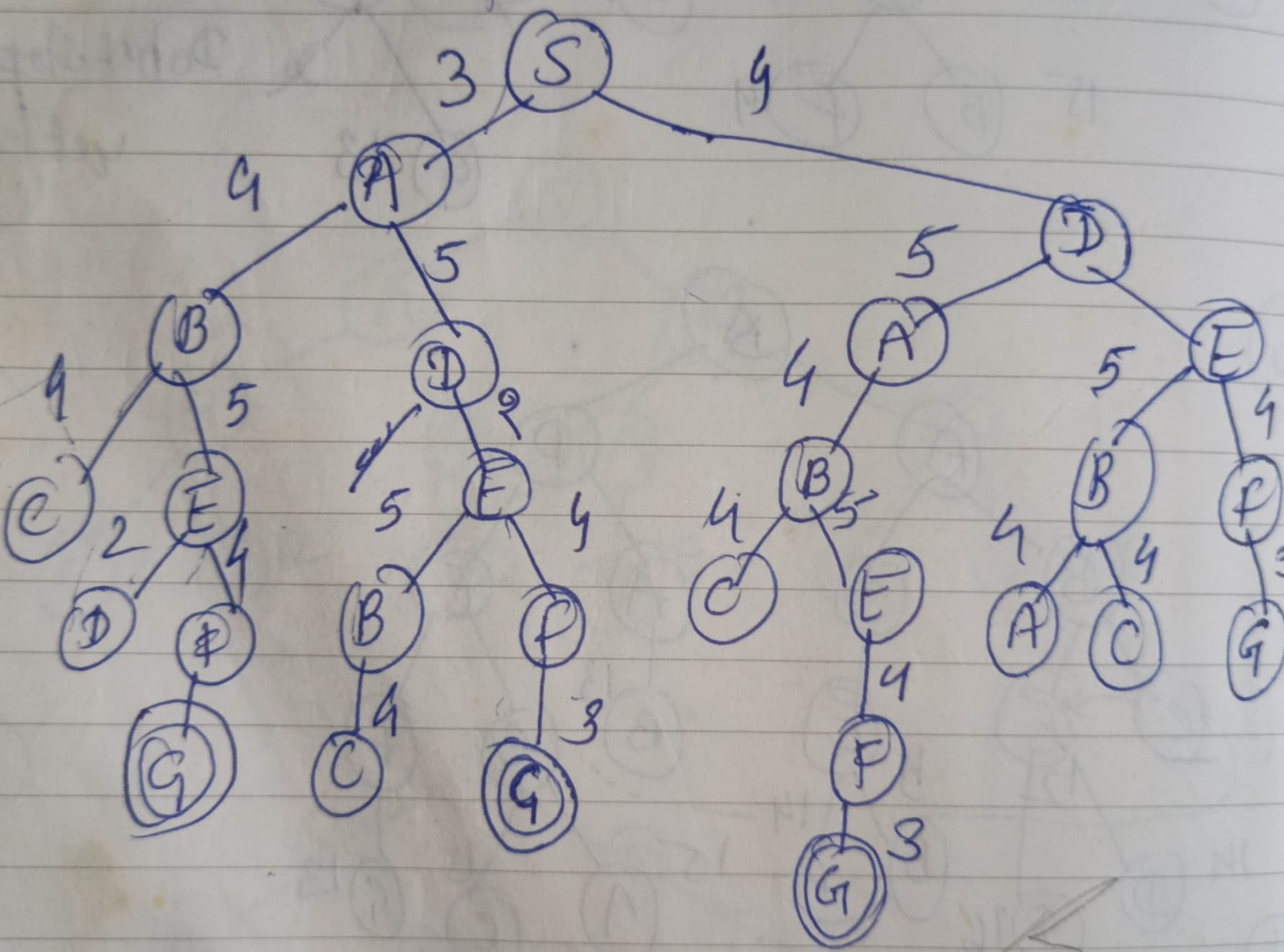
10



11

stop.

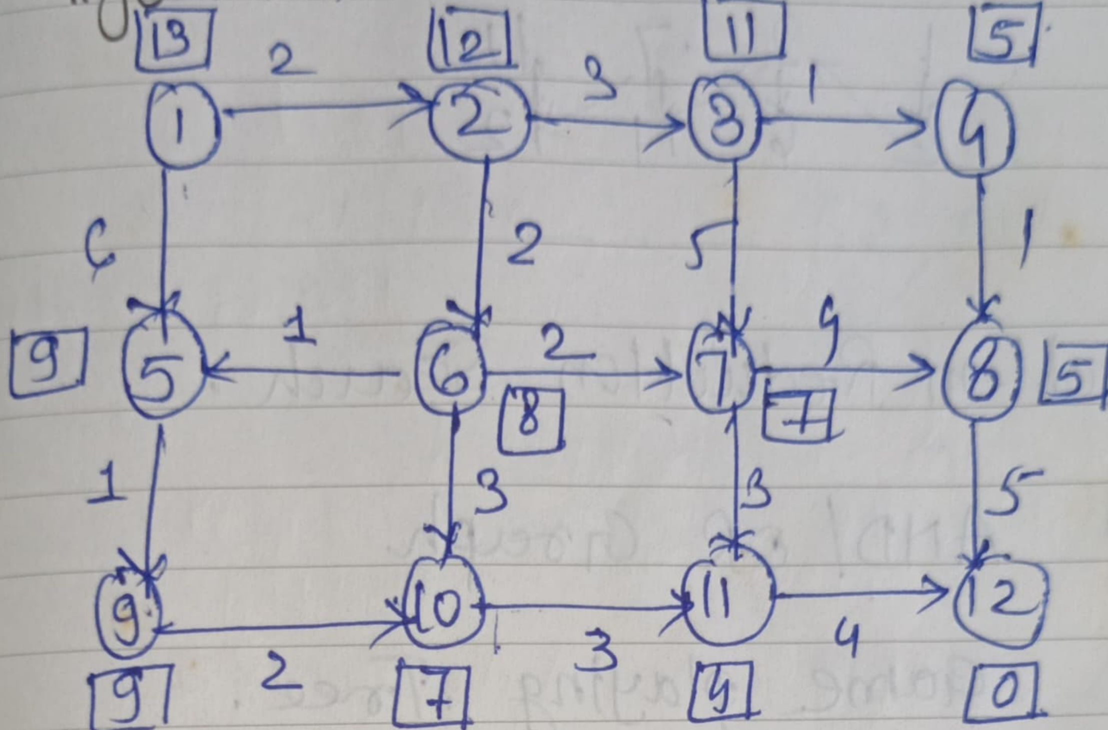

S
A
B
C
D
E
F
G
3
4
5
7
2


S
A
B
C
D
E
F
G

83

A* Algorithm.

Exp



becs goal node has eastimation cost 0

Open

~~1~~[13]

~~2~~[14], 5[15].

~~6~~[12] 5[15] 3[16]

~~7~~[13] 10[14] 5[15] 3[16]

~~11~~[13], 8[15] 10[14] 5[15] 3[16]

12[13], 8[15], 10[14] 5[15] 3[16]

Closed

1[13]

2[14]

6[12]

7[13]

11[13]

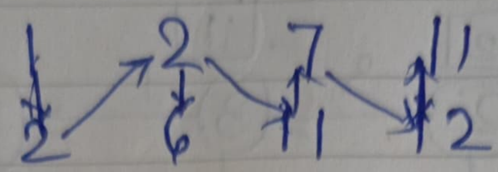
12[13]

stop

goal. & Success

Good.

28/7/10

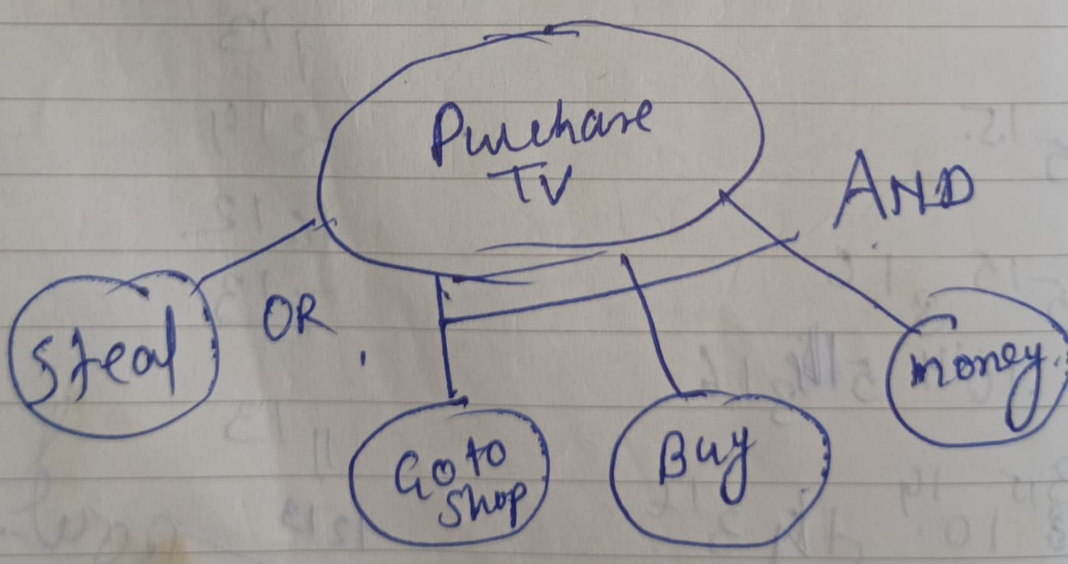


Problem Reduction Search.

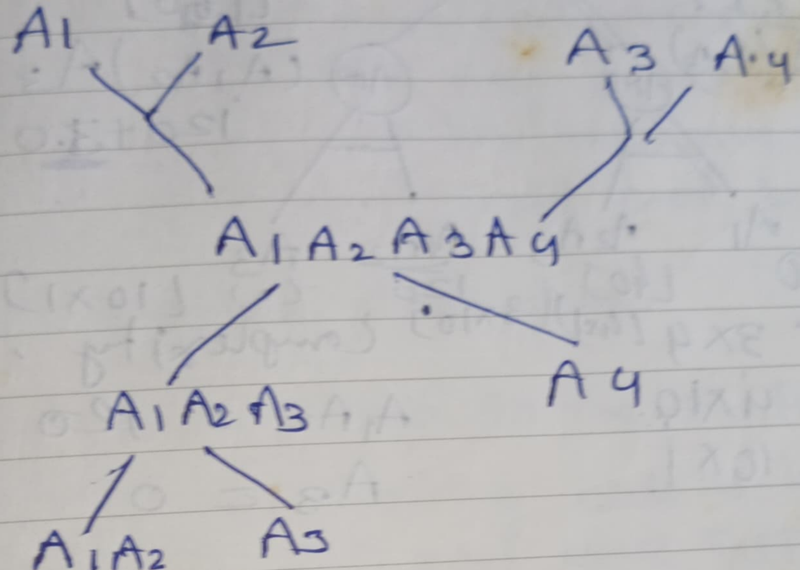
① AND/OR Graph

② Game playing Tree.

AND OR Graph.



$A_1, A_2, A_3 \ldots\ldots A_n$



$A_1 = n \times m$ | $A_1 = 3 \times 4$

$A_2 = m \times k$ | $A_2 = 4 \times 3$

complexity $A_1 A_2 = n \times m \times k$ $\quad = 4 \times 3 \times 3$.

$[A_1 A_2] = n \times k$

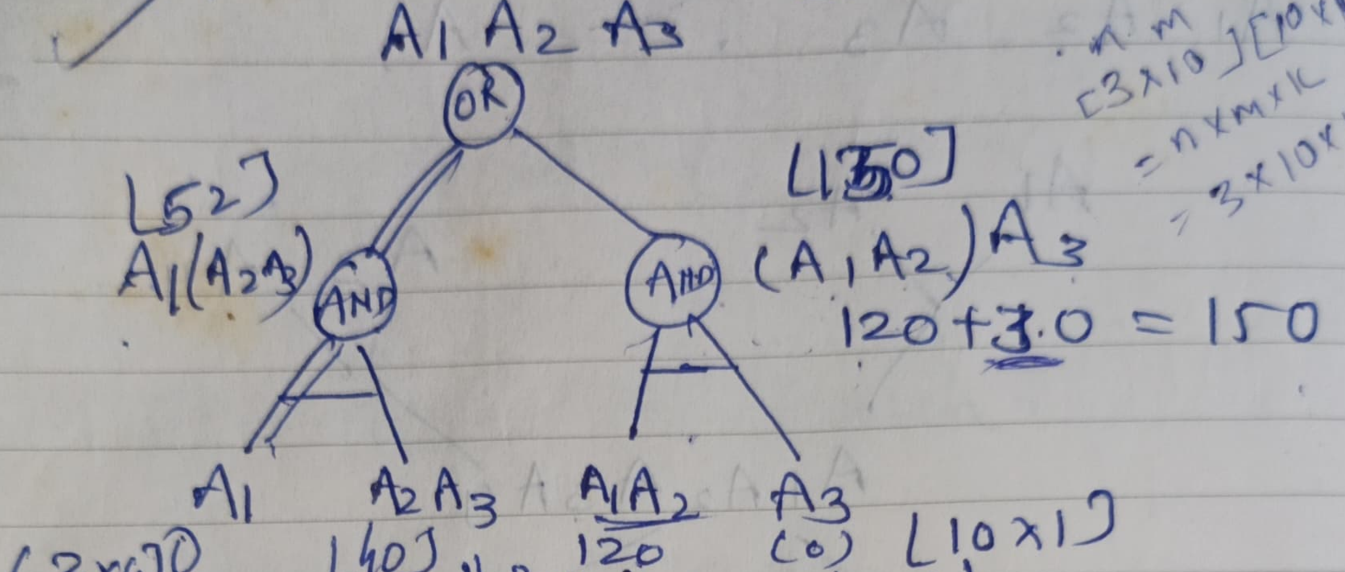
$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 1 \\ 1 & 2 & 3 \\ 3 & 2 & 4 \end{bmatrix}$$

120 (86)

$n \times m$ $[3\times10][10\times1]$

$= n\times m\times k$

$= 3\times10\times1 = 30$



Complexity.

$A_1 = 3\times4$ $\qquad$ $A_1A_2 = 120$

$A_2 = 4\times10$

$A_3 = 10\times1$ $\qquad$ $A_3 = 0$

✓ AND ORGRAPH:- AND-OR graph (or tree) is useful for representing the solution of problem that can be solved by decomposing them into a set of smaller problems, all of which must then be solved.

- One AND arc may point to any no of successor nodes, all of which must be solved in order for the arc to point to a solution

Problem Reduction

- FUTILITY is chosen to correspond to a threshold such that any solution with a cost above it is too expensive to be practical, even if it could ever be found.

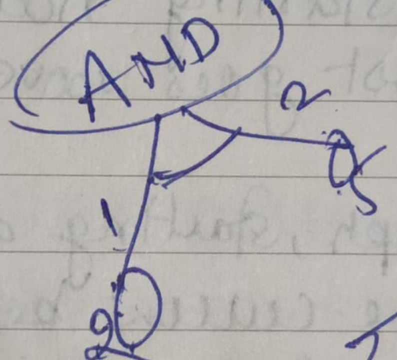
Algorith : Problem Reduction

1. Initialize the graph to the starting node.
2. Loop until the starting node is labeled SOLVED or until its cost goes above FUTILITY:

a. Traverse the graph, starting at the initial node and following the current best path. and accumulated the set of nodes that are on that path and have not yet been expanded or labeled as solved.

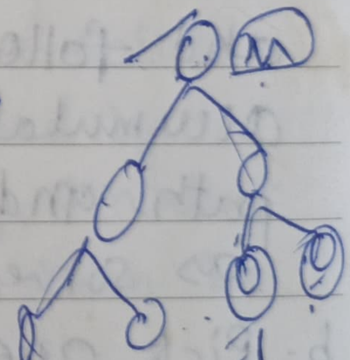
b. Pick one of these nodes and expand it. If there are no successors, assign FUTILITY as the value of this node. Otherwise, add its successor to the graph and for each of them compute f'. If f' of any node is 0, mark that node be solved.

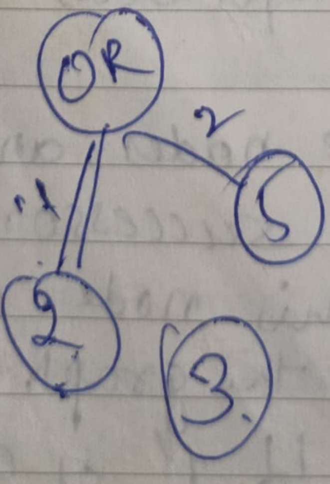
c. Change the f' eastimate of the newly expanded node to reflect the new information provided by its succesors. Propagate this change backward through the graph. This propogation of revised cost eastimates backup the tree were examined. But now expanded nodes must be reexamined so that the best current path can be selected



$\Sigma\ (2+5)+(1+2)$

$\underline{\underline{10}}$





AO* Algo

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

AND/OR

~~Initialize~~ Initialize

1. set $G^* = \{s\}$, $f(s) = h(s)$
2. If $s \in T$, label 's' as solved
   Terminate: if s is solved then terminate.
3. Select a non-terminal leaf node n from the marked SUB TREE.
4. Expand: Make explicit the successors of n
   for each new successor m:
   set $f(m) = h(m)$
   if m is terminal, label m solved.
5. Cost Revision: Call Cost-revise(n).
6. LOOP: Goto step 2.

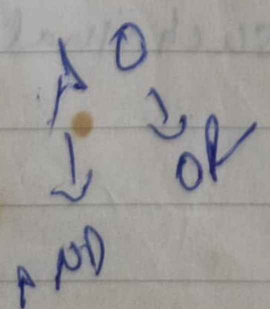
Cost-revise (n)

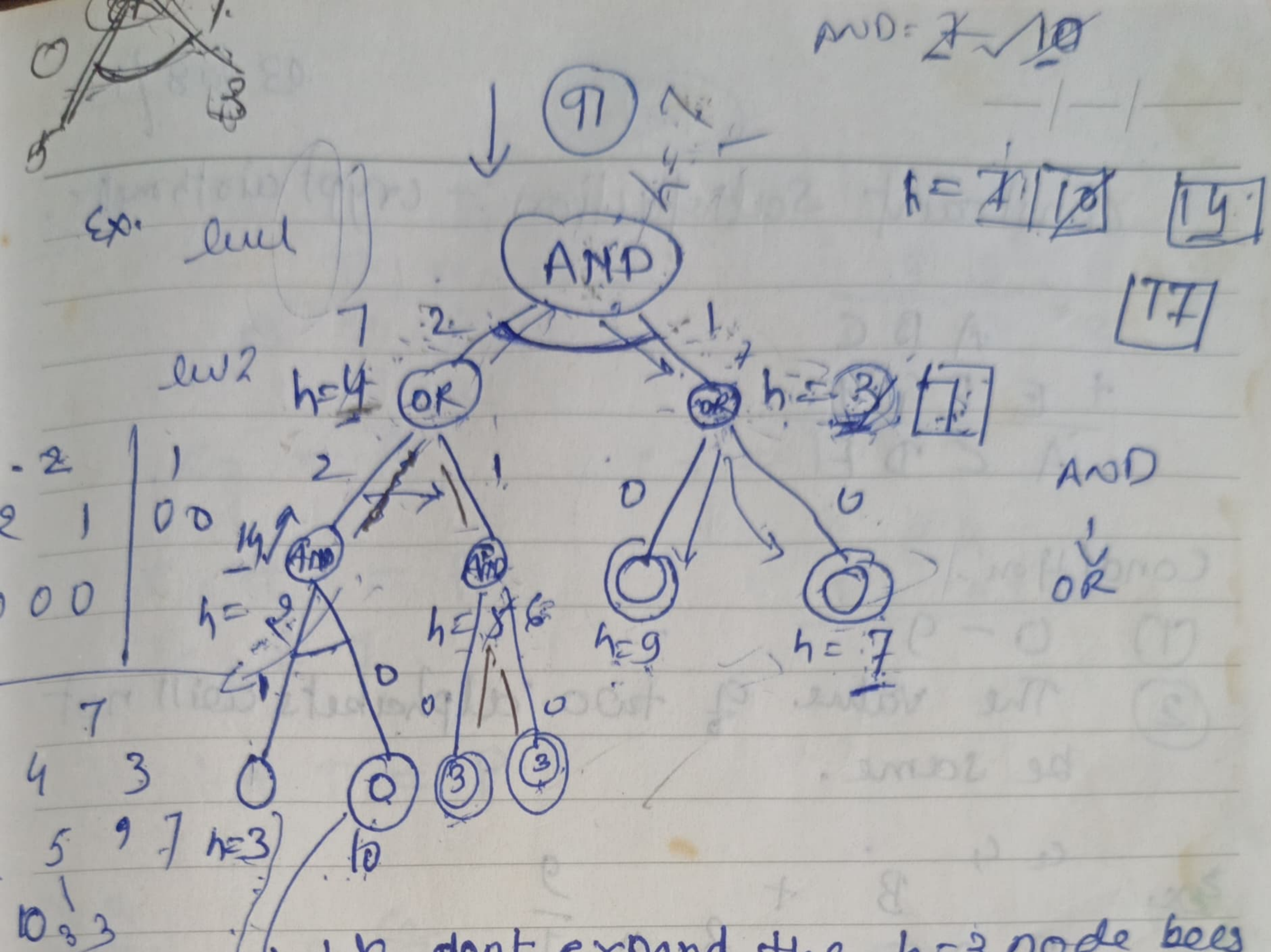
1. create $Z = \{n\}$
2. If $Z = \{\}$ return.
3. Select a node m from Z such that m has no descendents in 'Z'

4. If m is an AND node with successors $r_1, r_2, \ldots, r_k$.
set $f(m) = \Sigma \, [f(r_i) + c(m, r_i)]$
mark the edge to each successor of m.
if each successor is labelled SOLVED then label m as SOLVED.

5. If m is an OR node with successors $r_1, r_2, \ldots r_k$
set $f(m) = \min \{ f(r_i) + c(m, r_i) \}$
mark the edge to best successors of m.
If the marked successor is labeled SOLVED, Label m as solved.

6. If the cost or label of m has (changed then insert those parents of m) changed then insert those parents of m into Z for which m is a marked successor.





We dont expand the h=3 node bcoz the value of h is always greater than 10.

Cryptarithmetic Problems :- (CPs) They are the typical constraint satisfaction problems which aim to find a substitution sequence of digits for letters so that the resulting sum is arithmetically correct.

(92) 03/08/10

Constraint Satisfaction :- crypt airthmetic.

$$\begin{array}{r} A\ B\ C \\ +\ E\ F\ G\ H \\ \hline A\ C\ D\ E \end{array}$$

Condition.

(1) 0 - 9

(2) The value of two alphabets will not be same.

Ex.

$$\begin{array}{r} c_2\ c_1\quad \\ B \\ F\ B\ I \\ \hline B\ S\ S \end{array} \quad + \quad \begin{array}{r} 9 \\ 8\ 9\ 1 \\ \hline 9\ 0\ 0 \end{array}$$

$c_2 + F = B$

Put $c_2 = 0$

$F = B$ it is not possible.

$c_2 = 1$

step 2

$c_1 + B = S$ $\quad$ $1 + B = S$.

$c_1 = 0$ $\quad$ put $B = 9$

$B = S$.

$\boxed{c_1 = 1}$

Ex 2.

	C_2	C_1	E	A	T
+		T	H	A	T
A	P	P	L	E	

	8	1	9	
9	2	1	9	
1	0	0	3	8

$C_2 = A$, $C_2 = 1$.

$C_1 + T = P$

$C_1 = 1$.

$1 + T = P$.

E A T
T H A T

Ex 3.

	C_2	C_1	
		U	S
+		A	S
	A	L	L

	8	5
	1	5
1	0	0

$C_2 = A$.

$A = 1$

$C_1 + U + A = L$

$C_1 = 1$

$U = 8$ or 9.

$0 + 9 + 1 = 10$

$1 + 8 + 1 = 10$

$1 + 9 + 1 \quad 11$

$A = 1$
$C_1 = 0$
$U = 8$
$C = 1$
$U = 8$ or 9

$C_1 = 0$:

$A_1 = 1$
$U = 9$
$t = 0$
~~but~~
$S = 4$
$S + S < 10$.

$C_1 = 1$:

$A = 1$
$C_1 = 1$
$U = 8$
$C_2 = 1$
$t = 0$
$S = 5$
$S + S \geq 9$

DAYS	9 8 1
TOO	7 6 6
SHORT	1 0 6 7

SHE	9 2 5
THE	2 5
BEST	1 5 ~~8~~ 0

C_2 C_1

DAYS
TOO
SHORT.

$C_2 + D = S$.

$C = 1$

D A Y S

9 8 7 1
6 5 5
1 0 5 2 6

(95)

Ex ① CROSS
ROADS
DANGER

② SHE
THE
BEST

③ SEND
MORE
MONEY

④ DAYS
TOO
SHORT.

c_2 c_1
CROSS
ROADS
DANGER

$E + D = N$
$E + 1 = 01$

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

$c_1 + c + R = A$
$c_1 = 0$
$c + R = A$

$c_1 = 1$
$c_1 + c + R = A$
$c_1 = 0$
$c + R = A$

$c_2 = D$

9851
766
10617

DAYS
TOO
SHORT

9 8 5 1
~~7~~ ~~6~~ 6
1 0 6 ~~1~~ ~~7~~

Logic

Logic is the understanding of human intelligence

1. Propositional logic
2. Predicate logic.

Propositional logic.

Fact: 1. Atomic sentence P, Q, R, S

2. Compound or complex
   ↳ connectives
   1. $\sim$ (not)
   2. $\wedge$ (And) $P \wedge Q$, $P \vee Q$
   3. $\vee$ (Or)
   4. $\rightarrow$ (conditional) $P \rightarrow Q$.
   5. $\leftrightarrow$

$P \leftrightarrow Q = (P \rightarrow Q) \wedge (Q \rightarrow P)$

Syntax - way of writing
Symmetric - meaning

Only give the true/false value.
not give the conclusion.

Limitation

① Ram is a man.
② Shyam is a man.
③ All man are mortal.

Predicate logic :-

Same connectives as ($\sim, \wedge, \vee, \rightarrow, \leftrightarrow$)
variable P, Q, R, S.
Quantifiers

$\exists$ – Existential (for some)
$\forall$ – universal (for all)

function f(x) term → x.
terms :- constant, variable, function
Predicate (term)
Predicate (function)
but vise versa is not possible.

Condition :- If the term generate the value then it will be function.
& If only you get true/false then it will be predicate.

function eg. — age (Ram) = 26 etc.
Predicate — doctor (Mohan) – True/false.

Some Equivalance Rule :-

(1) Idempatency $P \vee P = P$

$P \wedge P = P$

(2) Associativity :

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

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

(3) Commutativity

$P \vee Q = Q \vee P$

$P \wedge Q = Q \wedge P$

$P \leftrightarrow Q = Q \leftrightarrow P$

(4) Distributivity.

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

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

(5) De Morgan's Law.

$$\sim(P \vee Q) = \sim P \wedge \sim Q$$
$$\sim(P \wedge Q) = \sim P \vee \sim Q$$

(6) Conditional elimination.

$$P \rightarrow Q = \sim P \vee Q.$$

(7) Bi-Conditional elimination.

$$P \leftrightarrow Q = (P \rightarrow Q) \wedge (Q \rightarrow P)$$

Infrence Rule for Propositional Logic.

(1) Modus Ponens

$$\frac{\begin{array}{c} P \text{ is sentence} \\ P \rightarrow Q \end{array}}{Q}$$

eg. Joy is Father

$$\frac{\text{Joy is father} \rightarrow \text{Joe has child.}}{\text{Joy has child.}}$$

Chain Rule.

$$\frac{\begin{array}{c} P \to Q \\ Q \to R \end{array}}{P \to R}$$

P → Programmer likes LISP
Q → Programmer hates COBOL
R → Programmer likes Recursion

Programmer likes LISP → Prog. likes Recursion

① If the unicorn is mythical then it is immortal.
But if it is not mythical then it is a mortal and mammal,

② If unicorn is either immortal or a mammal, then it is horned,