22/03/22

Artificial Intelligence & Machine Learning

UNIT - 1

* AI is defined as the study of systems that act in a way that to any observer would appear to be intelligent.
* AI involves using methods based on the intelligent behaviour of humans & other animals to solve complex problems.
* Abilities to achieve AI by a machine:
  i) Problem solving
  ii) Logical Reasoning
  iii) Generalized learning.

According to Aristotle a syllogism is defined as "a dispose in which discourse in which sudden certain things having being stated, something else follows of necessicity from their being said".

PROLOG → PROgramming in LOGic
LISP → LIST Programming

Rules

contains (X,Y) :- made from (X,Z), contains (Z,Y)

B :- A

__/__/____

1) i) tasty (cheese) → Cheese is tasty
ii) made from (cheese, milk)
Cheese is made from milk.
iii) contains (milk, calcium)
Milk contains calcium.

B :- A

X contains Y
X is made from Z & Z contains Y ⇒ X contains Y.
~~⇒ X cont~~

X ~~contains~~ & is made from Z & Z contains Y then X contains Y.

2) animals (mammals)
mammals (elephants)
animals (elephats)
→ Mammals are animals
Elephants are mammals
Elephants are animals.
→ Mammals are animals & elephants are mammals then elephants are animals.

animals (elephants) :- animals (mammals),
mammals (elephants)

animals ↑

__/__/____

3) has (mammals, legs)
is (dog, mammal)
has (cat, legs)

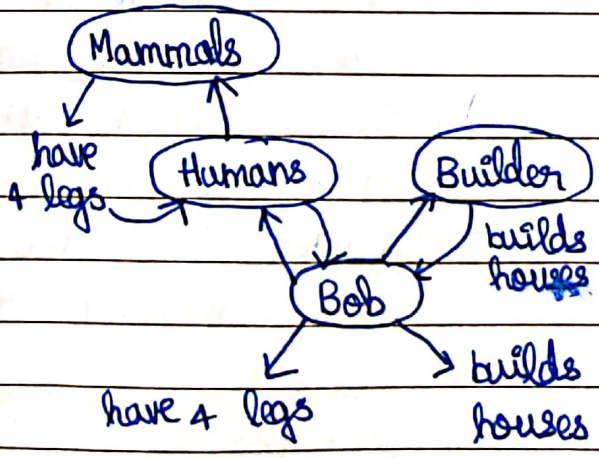
mammals — has
has (dog, legs) :- is (dog, mammals), has (mammal, legs)

★ Frames, slots, slot values

- Aggregation & association.
- * → Overriding symbol. (Add it before slot name to indicate that the slot value can be changed)
- We can express slot as a frame.

Frame Name	Slot	Slot value
No. of legs	Min	0
	Max	4

Humans are Mammals
Mammals have 4 legs
Bob is a human
Bob is a builder
Builder builds houses.



Admissibility : ...
c) Irrivocability : search method ... is not allowed

__/__/____

Procedures

Procedures are set of instructions associated with frames which are called on request.

Eg:

- Slot reader : is called when it wants to read the value of a slot.
- Slot writer : value to be inserted into a slot.
- instance constructor : creates new instance for a class.

→ WHEN NEEDED PROCEDURES

Demons

- It is a special type of procedure that is run automatically whenever a particular value changes or a particular event occurs.
- Some demons are called when a particular value is read or they are called automatically when the user of the system wants to know what value it plays in a particular slot. Such demons are called WHEN READ PROCEDURES.
- Some demons are automatically called when value of a slot is automatically changed. Such demons are called WHEN CHANGED PROCEDURES or WHEN WRITTEN PROCEDURES.

__/__/____

Implementation of Frames

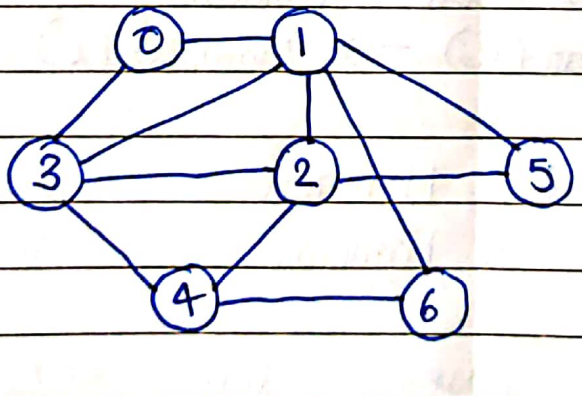
```
Function find_slot_value(S, F)
{       if F[S] == V
            then return V
        else if instance(F, F')
            then return find_slot_value(S, F')
        else if subclass(F, F1)
            then return find_slot_value(S, F1)
        else return fail
}
```

AI & ML Lab

Graph Traversals

DFS

- Depth First Search.
- Uses stack (FILO).



eg: 6 4 3 5 2 1 0

BFS

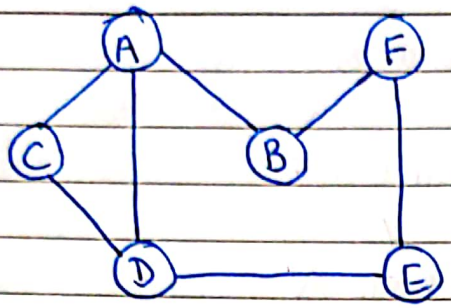
- Breadth First Search.
- Uses queue (FIFO)
- Level order traversal.

eg: 0 1 3 2 5 6 4

- Application : LinkedIn

__/__/____

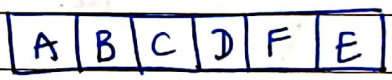
Q)



Taking A as root

DFS : C D E F B A

BFS : A B C D F E



Representational Adequacy

i) All Humans are Mammals

→ $\forall x$ Human$(x) \rightarrow$ Mammal(x)

ii) All Dogs are Mammals

→ $\forall x$ Dog$(x) \rightarrow$ Mammal

iii) Some Dogs have 3 legs

→ $\exists x$ Dog$(x) \rightarrow$ Legs$(x, 3)$

iv) Not all like both Mathematics & Science

→ $\neg[\forall x \quad x \rightarrow$ like$(x,$ Mathematics$) \wedge$ like$(x,$ Science$)]$

¬ → Negation sign
∧ → And

__/__/____

v) Some dogs chase cats & eat cheese
→ $\exists x$ Dogs(x) → chase(x, cats) ∧ eat(x, cheese)

vi) Some dogs chase cats & do not eat cheese
→ $\exists x$ Dogs(x) → chase(x, cats) ∧ ¬[eat(x, cheese)]

Search Space

- It is a representation of a set of all possible choices in a given problem.
- One or more choices can be the solu^n to the problem.

★ One method to find a solution is Heuristic approach. (But this method does not ensure an optimistic solu^n).

Robot, Block example → Imp.

Semantic Tree

Semantic net

root

1) One predecessor (parent node) & can have more than one successors (children)

leaf node

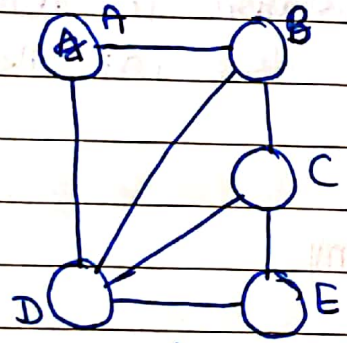
2) Root node which do not have predecessor in the starting pt.
3) The nodes do not have successors is called leaf nodes.

__/__/____

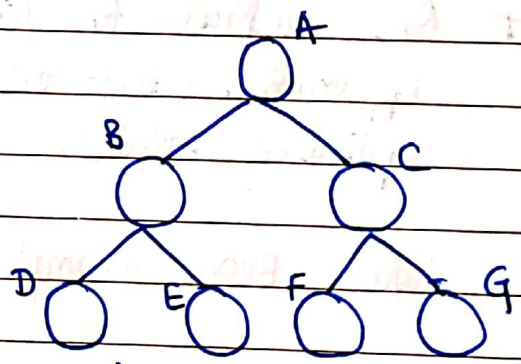
The leaf nodes which lead to a successful search are called goals.

4) Path is the root through the semantic tree. A path which leads from root node to the goal node is called complete path.
A path which leads from root node to the leaf node which are not goal nodes is called partial path.

5) An ancestor of a node is a node further up the tree in some path.
Descendant comes after a node in a path in a tree.

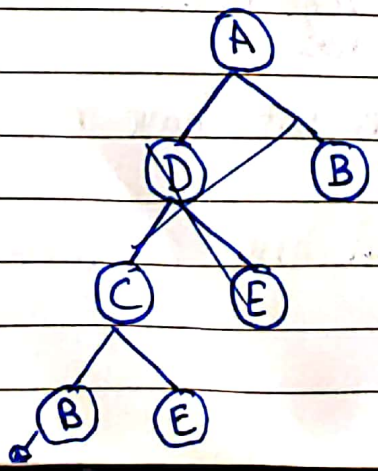


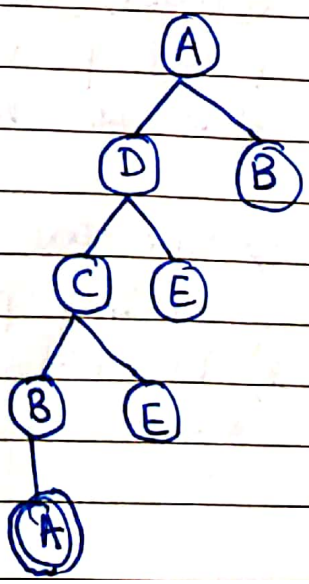
i) Semantic net



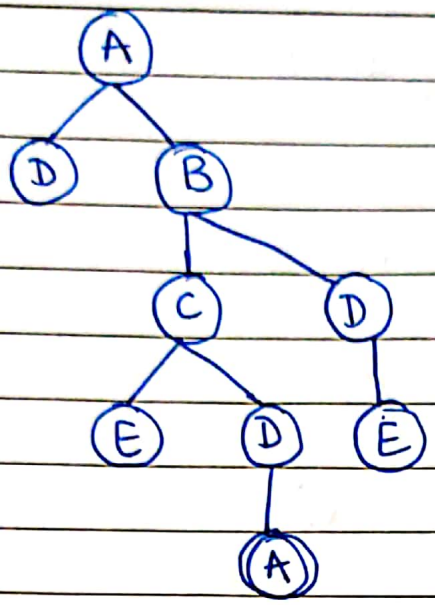
ii) Semantic tree

Search trees

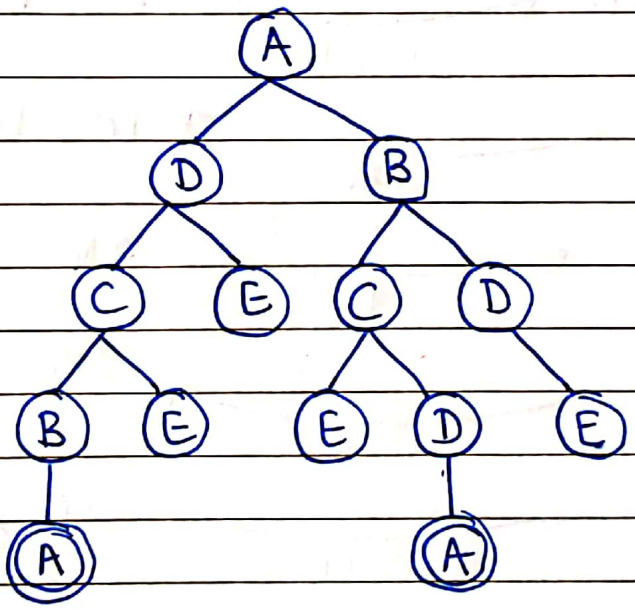




__/__/____



Final solution (All paths)



Missionaries & Cannibals

1) Move 1C to the other side
2) Move 2C " " " "
3) " 1M " " " "
4) 2M " " " "
5) " 1C & 1M " " "

operators

__/__/____

3C, 3M 1C → 0, 0 (2C)

1C, 3M ← 1C, 0 (1C)

3M → 1C (2C)

3M ← 2C (1C)

1C, 1M → 2C (2M)

1C, 1M ← 1C, 1M (1C, 1M)

2C → 1C 1M (2M)

2C ← 3M (1C)

1C → 3M (2C)

1C ← 1C, 3M (1C)

1C, 3M (2C)

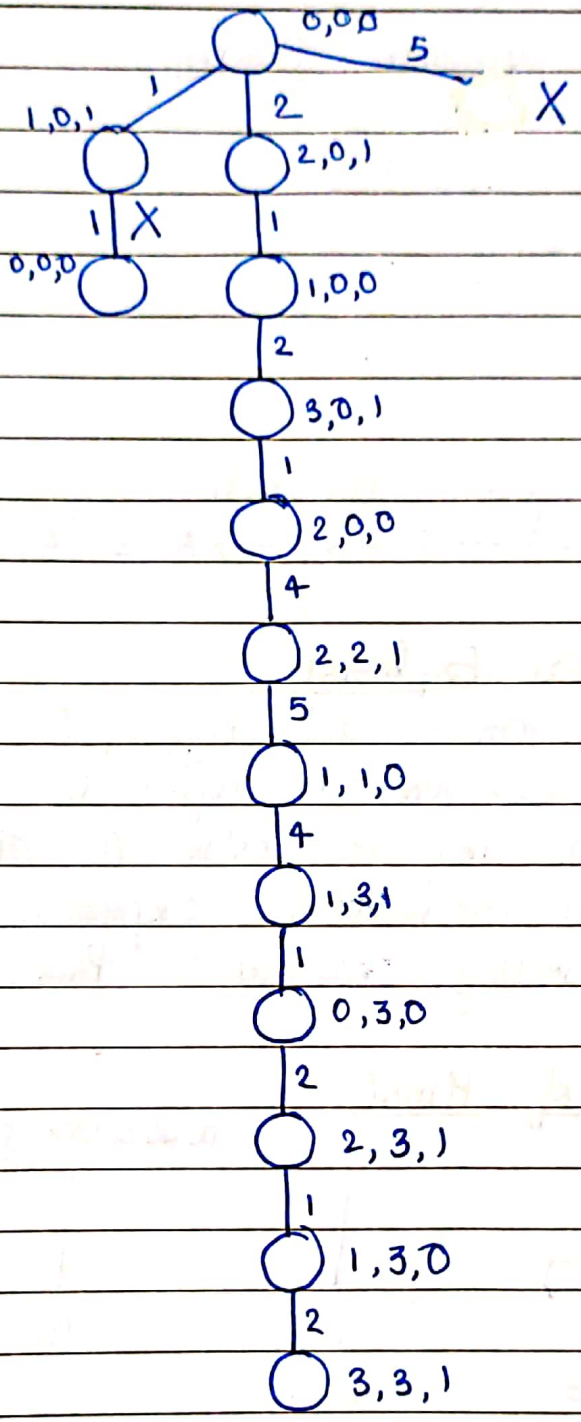
O , O , O

cannibals

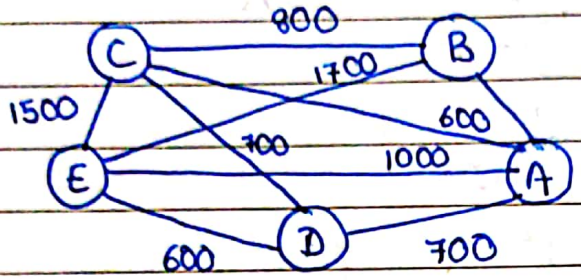
missionaries

cannoy

__/__/____



Travelling Salesman Problem



Nearest neighbor approach

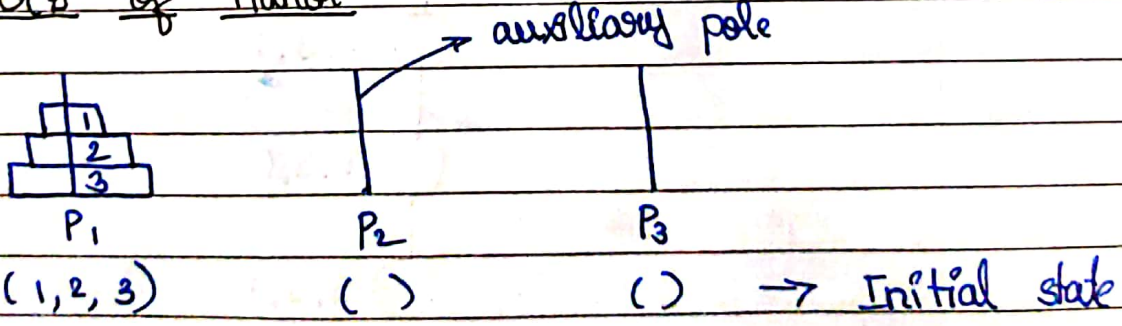
A → C → D → E → B → A = 4500

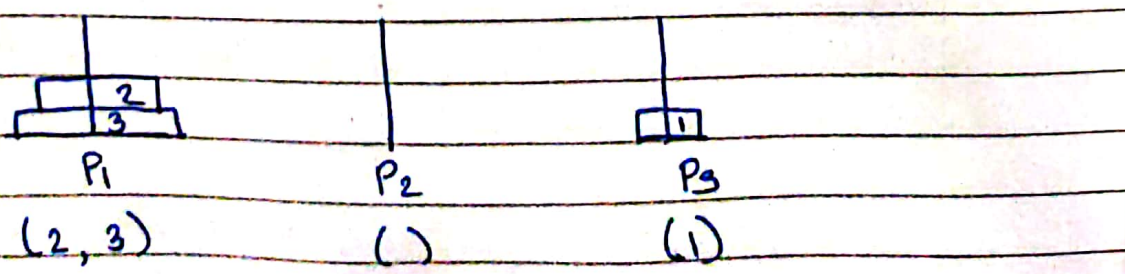
Combinatorial Explosion

As the problem gets bigger, it becomes more & more unreasonable to expect the computer to be pgm to be able to solve it. This problem is known as combinatorial explosion.

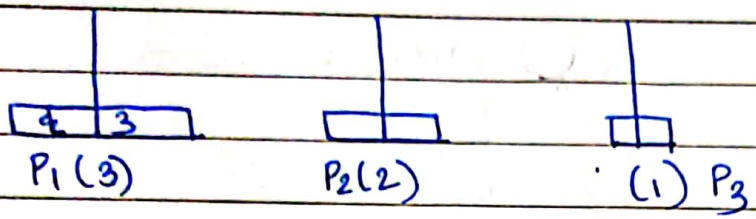
Eg: Travelling Salesman's Problem.

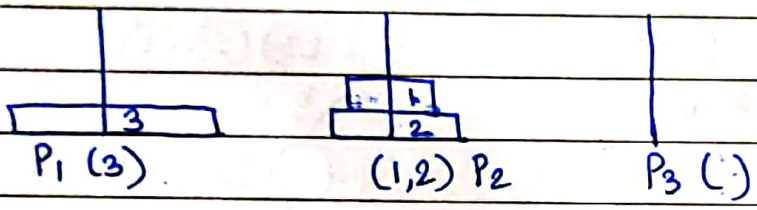
Towers of Hanoi

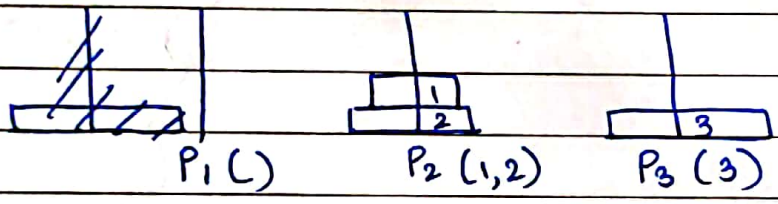


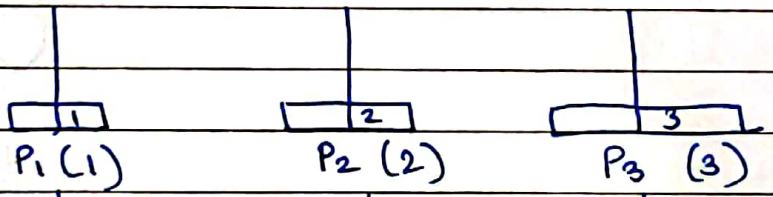


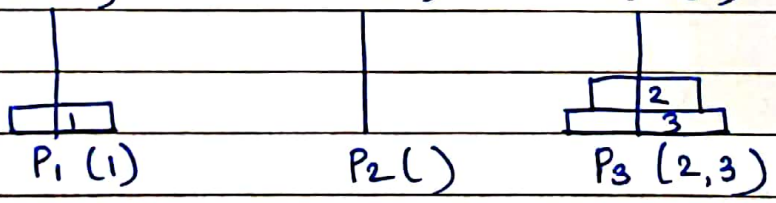
__/__/____

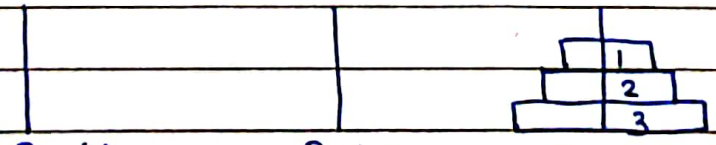










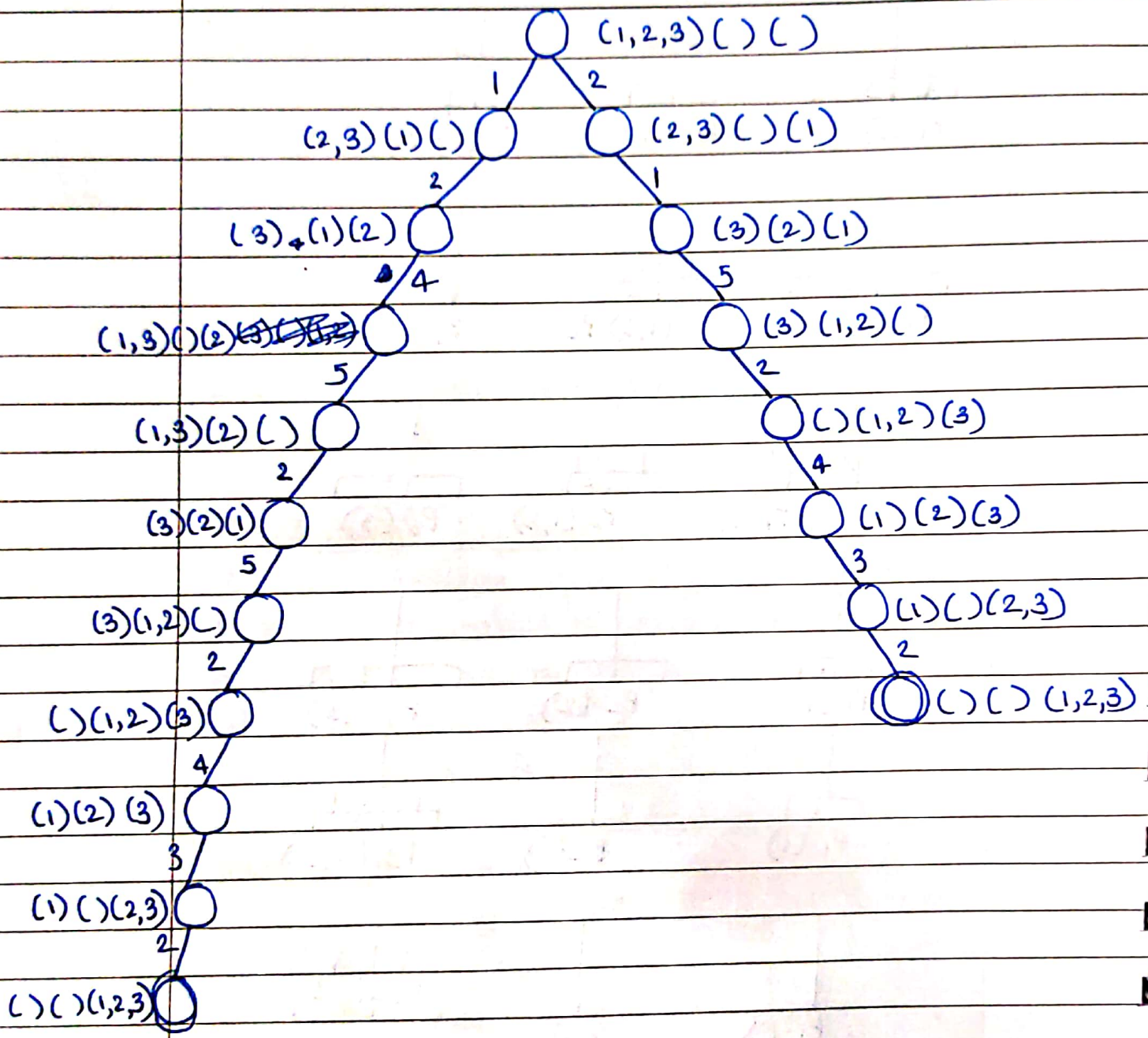


→ Final.
↳ Goal state

Operators

1) Move disc from P_1 to P_2
2) " " " P_1 " P_3
3) " " " P_2 " P_3
4) " " " P_2 " P_1
5) " " " P_3 " P_2
6) " " " P_3 " P_1

__/__/____

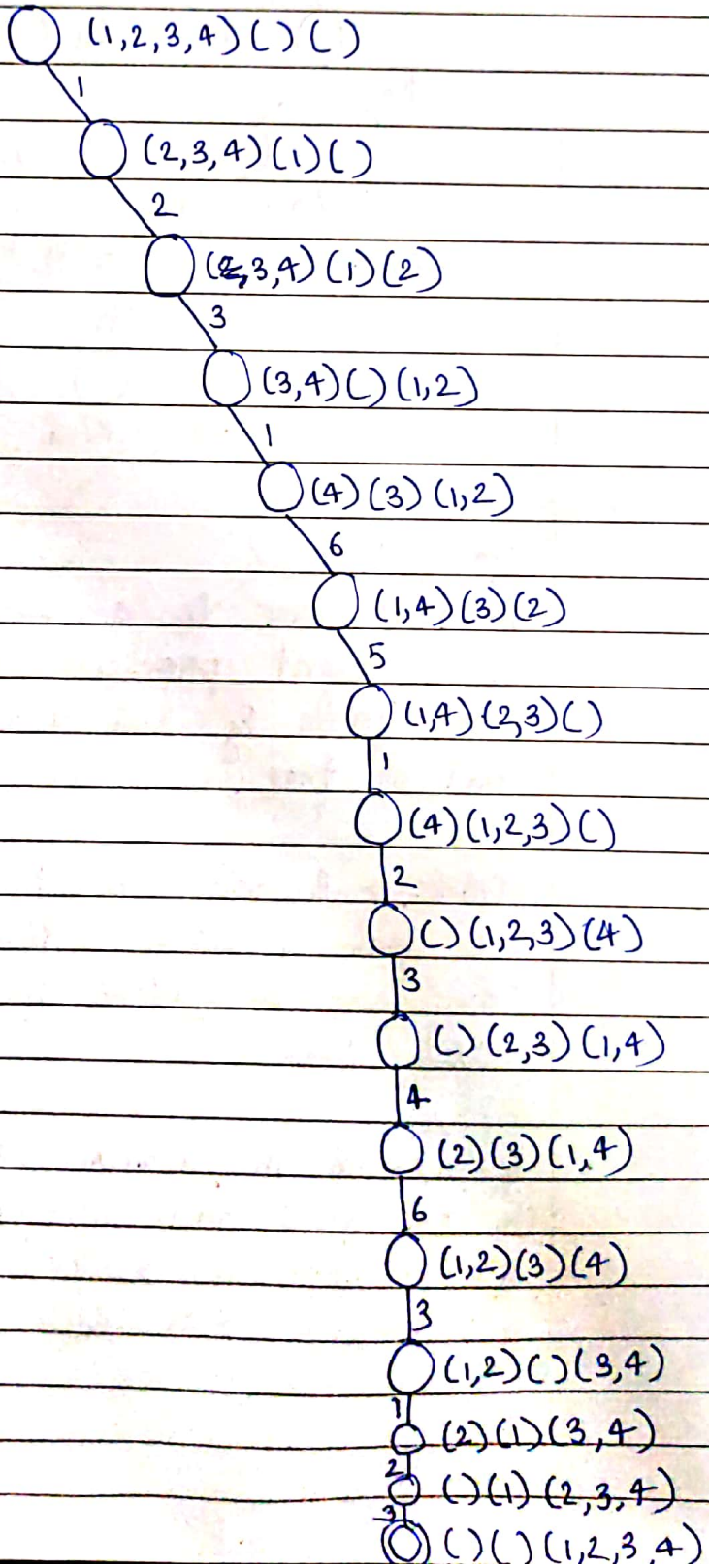


Problem reduction / Goal reduction

A complex problem can be solved by breaking it down into several smaller problems. If we solve all those smaller sub problems then we have solved the main problem.

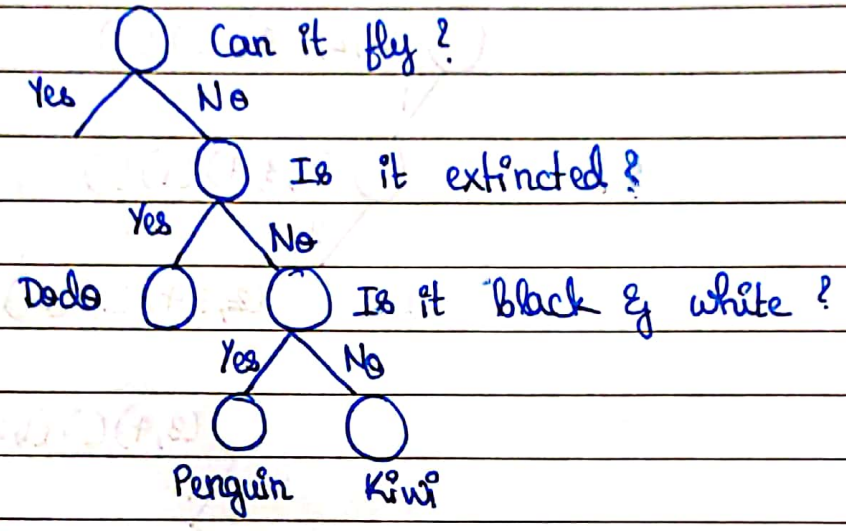
__/__/____

Where no. of discs are 4 :



__/__/____

Describe & Match



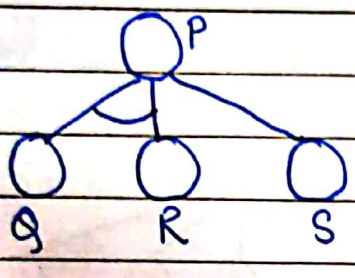
Goal tree

A goal tree is a form of semantic tree used to represent problems than can be broken down into goals & sub goals. These are also called And-or tree.

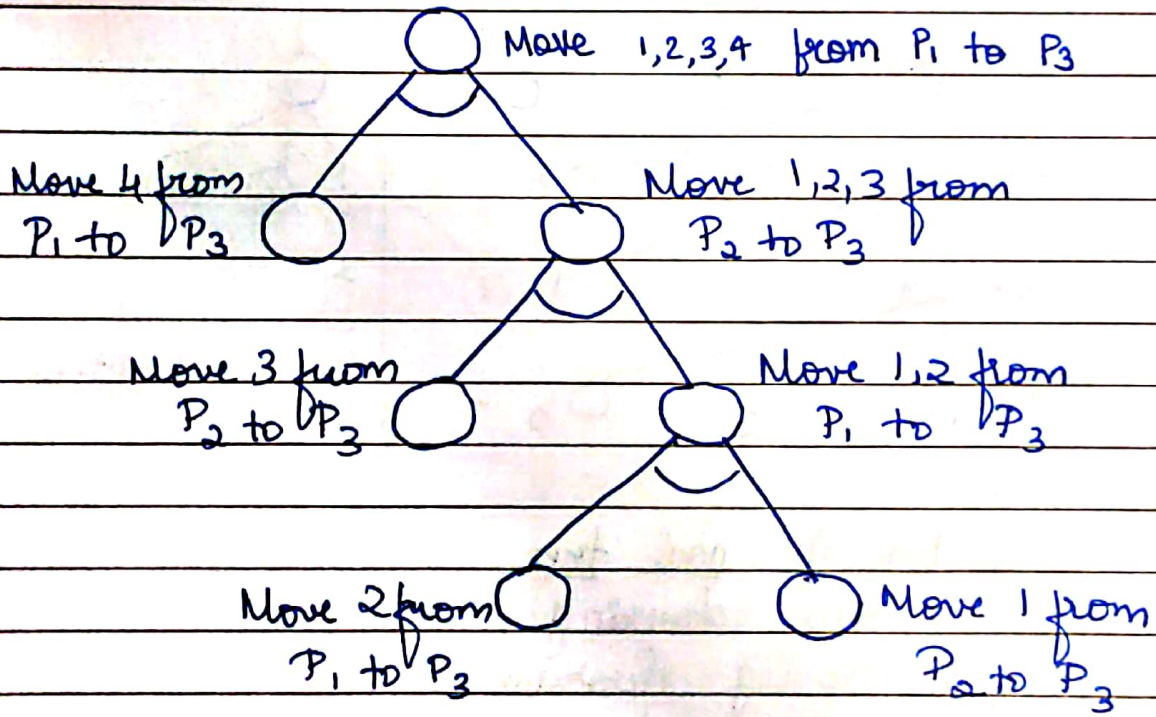
A solu" to the problem is referred as a goal & each individual step performed along the way to achieve the goal. is known as a sub goal.

* If a goal is achieved only by achieving all of its sub goals then such nodes are called and nodes.
* If a goal is achieved by solving only one of its subgoals then such nodes are called odd nodes.

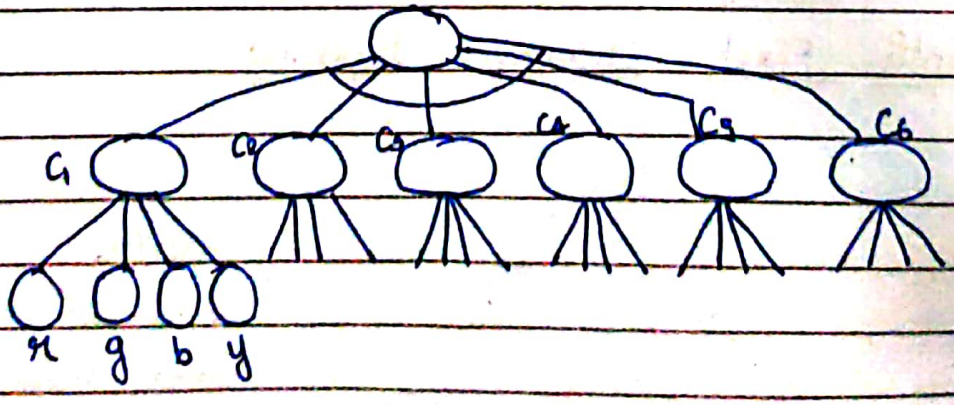
__/__/____



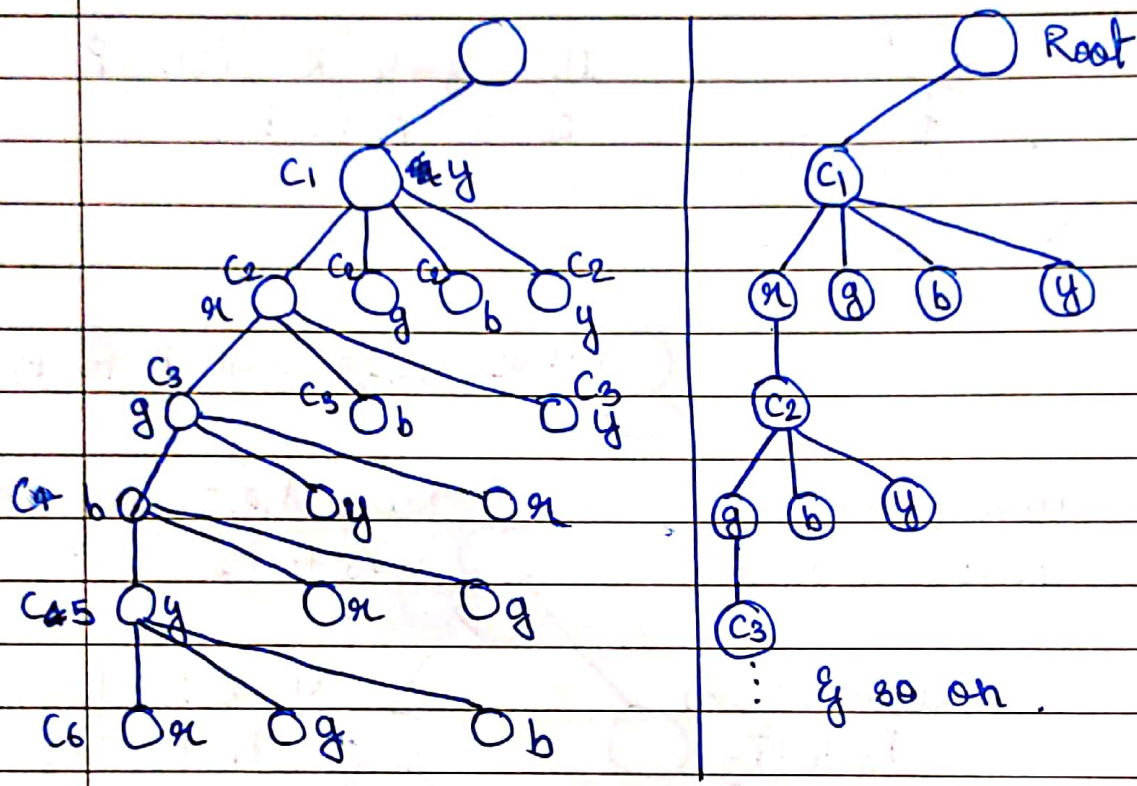
If Q & R then P
If S then P.



Map colouring



__/__/____

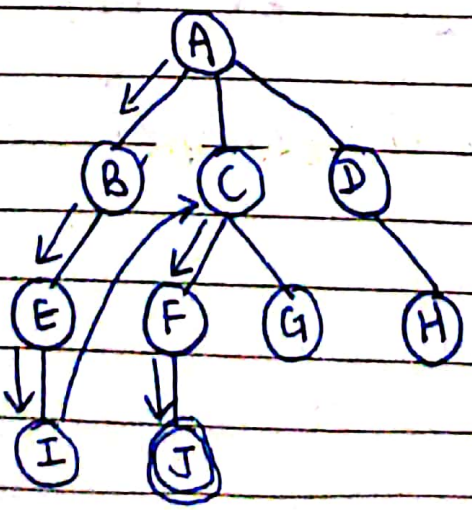


Uses of goal tree

i) Map colouring
ii) Proving theorems.

LAB

Depth First Iterative Depening



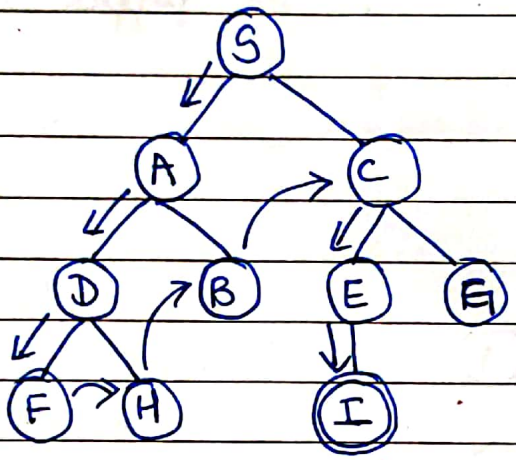
ABEE

Both DFS & BFS are used here.

__/__/____

d = 0 A
d = 1 A B C D
d = 2 A B E C F G D H
d = 3 A B E I C F J

②



d = 0 S
d = 1 S A C
d = 2 S A D B C E G
d = 3 S A D F H B C E I

Teamwork - 1

Initial stack
~~a~~ ['a', 'b', 'c']
Elements popped from stack
~~ε~~

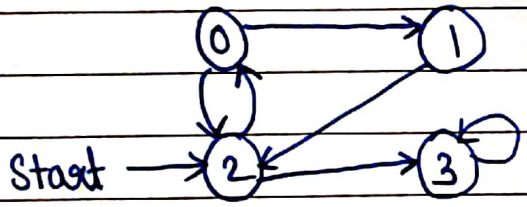
Stack

```
stack = []
stack.append('a')
stack.append('b')
stack.append('c')
print("Initial stack")
print(stack)
```

__/__/____

```
print("\n Elements popped from stack")
print(stack.pop())
print(stack.pop())
print(stack.pop())
print("Stack after elements are popped")
print(stack)
```

Graph



BFS

```
from collections import defaultdict

class Graph:
    def __init__(self):
        self.graph = defaultdict(list)

    def addEdge(self,u,v):
        self.graph[u].append(v)

    def BFS(self, s):
        visited = [False] * (max(self.graph)+1)
        queue = []
        queue.append(s)
        visited[s] = True
```

__/__/____

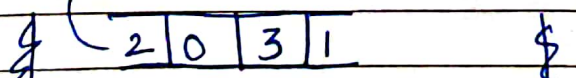
```
while queue:
    s = queue.pop(0)
    print(s, end=" ")
    for i in self.graph[s]:
        if visited[i] == False:
            queue.append(i)
            visited[i] = True
```

```
g = Graph()
g.addEdge(0,1)
g.addEdge(0,2)
g.addEdge(1,2)
g.addEdge(2,0)
g.addEdge(2,3)
g.addEdge(3,3)

print("Following is Breadth First Traversal (starting
 from vertex 2)")
g.BFS(2)
```

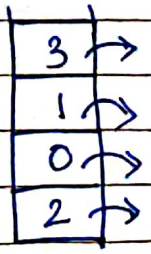
Output

BFS: 2 0 3 1



2 | 0 3 | 1

DFS:



2 0 1 3

__/__/____

DFS

```
from collections import defaultdict

class Graph:
    def __init__(self):
        self.graph = defaultdict(list)

    def addEdge(self, u, v):
        self.graph[u].append(v)

    def DFSUtil(self, v, visited):
        visited.add(v)
        print(v, end=' ')
        for neighbour in self.graph[v]:
            if neighbour not in visited:
                self.DFSUtil(neighbour, visited)

    def DFS(self, v):
        visited = set()
        self.DFSUtil(v, visited)

g = Graph()
g.addEdge(0,1)
g.addEdge(0,2)
g.addEdge(1,2)
g.addEdge(2,0)
g.addEdge(2,3)
g.addEdge(3,3)
```

__/__/____

print("Following is DFS from (starting from vertex 2)")
g.DFS(2).

Search Methodologies

Search is a method that can be used by computers to examine a problem space in order to find a goal.

Data driven Approach	Goal driven Approach
- Also called forward chaining.	- Also called backward chaining.
- Top down	- Bottom up.
- Goal state not known.	- Goal state known (start from goal & reach initial)
- Eg: Colouring of map	- Eg: Maze ptn (start from exit will be easier)

* Goal driven search is particularly useful in situations in which the goal can be clearly specified.
Eg: A theorem that is to be proved.

* Data driven search is most useful when the initial data is provided & it is not clear what the goal is.
Eg: Analyzing astronomical data.

No technique, simply search (Trial & error)

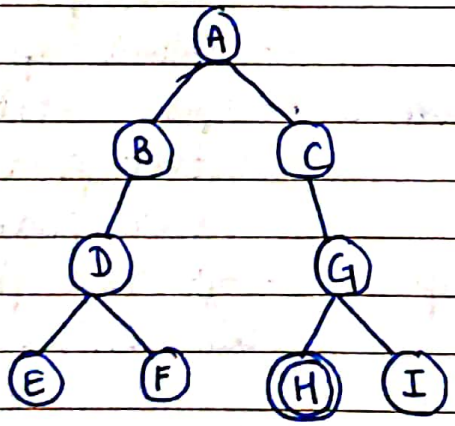
Generate & Test → Blind Search → Brute Force Search

Generator

3 properties

- It should be complete (all possible solns must be present & produced)
- It should be non redundant.
- It must be well informed. (it should not advice any soln that does not match the pbm st).

DFS & BFS



~~DFS → A B D E F C G H~~

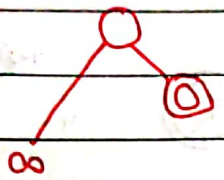
DFS : ~~E B D B H G C A~~ A B D E F C G H

Properties of Search Methods

1) Complexity - Time & Space
2) completeness
3) Optimality
4) Admissibility
5) Irrevocability - explore only one path (no backtracking)

__/__/____

BFS is considered as complete search method but DFS is ~~not~~ " " incomplete " " /tree.



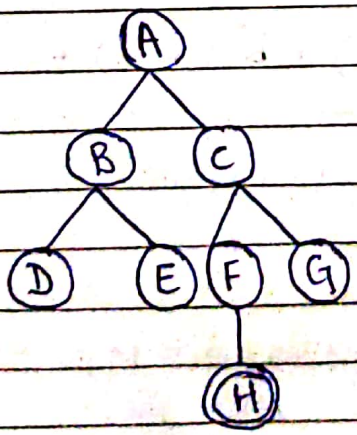
→ In this case DFS will never reach goal state.

BFS : Branch factor is the no. of children for a node.

The search method that finds optimal soln in quickest time is called admissible search time.

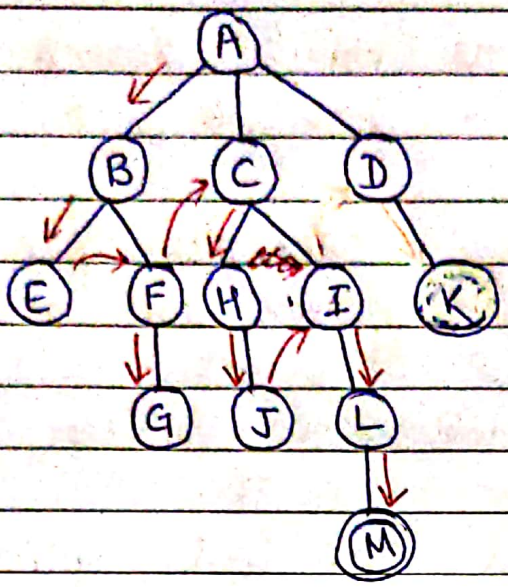
Search methods for which backtracking is allowed is called tentative methods.

DFID → Depth First Iterative Depening



Depth 0 → A
Depth 1 → ABC
Depth 2 → ABDECFG
Depth 3 → ABDECFH

__/__/____



Depth 0 → A
Depth 1 → ABCD
" 2 → ABEFCHIDK
" 3 → ABEFGCHJILDK
" 4 → ABEFGCHJILM

Depth d & branching factor b

$$1 + b + b^2 + b^3 + \dots + b^d = \frac{1 - b^{d+1}}{1 - b} \text{ nodes}$$

Eg: $d = 2, b = 2$:- $\frac{1 - 2^3}{1 - 2} = \frac{-7}{-1} = 7$ nodes

DFID 2

$$(d+1)1 + b(d) + b^2(d-1) + b^3(d-2) + \dots + b^d$$

↳ No. of times a node is visited.

In DFID, leaf nodes are visited only once, & other nodes will be explored multiple times.

same

DFS & BFS

$$1 + b + b^2 + \dots + b^d$$

For smaller trees DFID is inefficient.

__/__/____

Q) $d = 5$ & $b = 10$

→ DFS & BFS :

$$\text{Nodes} = \frac{1 - 10^{6}}{1 - 10} = \frac{1 - 1000000}{1 - 10} = \frac{999999}{9} = 111111$$

DFID

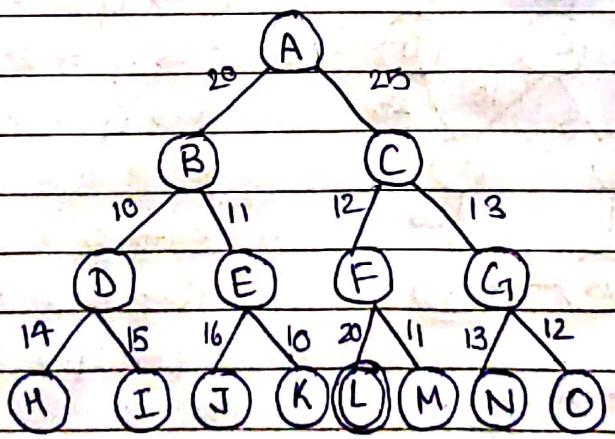
$$(5+1) + 10(5) + 10^2(4) + 10^3(3) + 10^4(2) + 10^5$$
$$= 6 + 50 + 400 + 3000 + 20000 + 100000$$
$$= 123456$$

Imp

Time complexity of DFID & BFS = $O(b^d)$

Space " " DFID & DFS = $O(bd)$

Best First Search



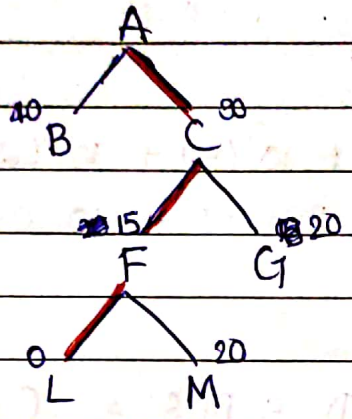
A → L → 50
B → L → 40
C → L → 30
D → L → 20
E → L → 15
F → L → 15

Heuristic

G → L → 20
L → L → 0
M → L → 20

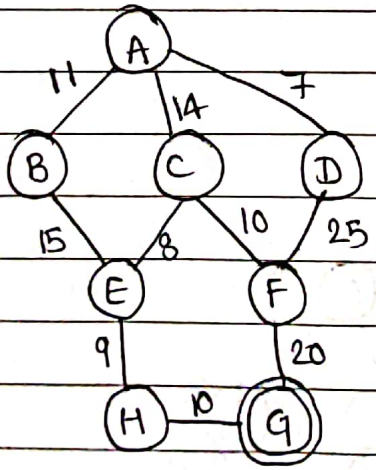
__/__/____

We need 2 queues (open queue, closed queue)
Priority ~~queue~~ based queue



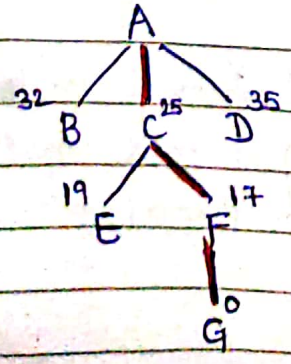
Open queue	Closed queue
~~ABC~~	A
CB	
~~CB~~	AC
FGB	AC
~~FGB~~	ACF
LMGB	
~~LMGB~~	ACFL
MGB	ACFL

Q)



A → G = 40
B → G = 32
C → G = 25
D → G = 35
F → G = 17
E → G = 19
H → G = 10
G → G = 0

~~CBA~~
CBDA



Open queue	Closed queue
A	
~~ABCD~~	A
~~CBD~~	AC
~~FEBD~~	ACF
~~GEBD~~	ACFG

__/__/____

Implementation of open queue

```
Function best()
{
  queue = [];
  state = root_node;
  while (true)
  { if is_goal (state)
      then return SUCCESS;
    else
    {
      add_to_front_queue (successor (state));
      sort (queue);
    }
   if queue == []
     then return FAILURE;
   state = queue[0];
   remove_from_front (queue);
  }
}
```

Tracing

State = A queue → B C D
sort → C B D

State = C queue → E F
sort → F E

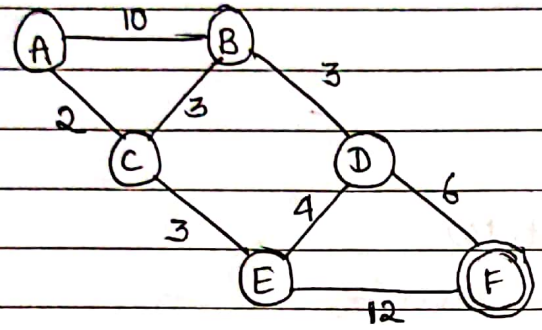
State = F queue → G
sort → G

State = G → Goal. return success

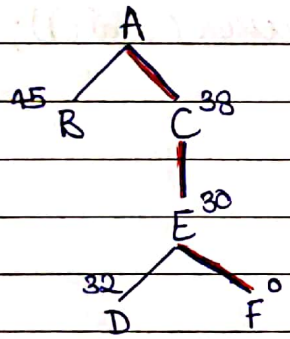
__/__/____

Text book pbm

Q)



A → F = 50
B → F = 45
C → F = 38
D → F = 32
E → F = 30
F → F = 0



State = A queue → BC
sort → CB

State = C queue → E
sort → E

State = E queue → DF
sort → FD

State = F ⟹ Goal return SUCCESS

The node with highest Heuristic value is the start node.

Advantages
Gives soln quicker

Disadvantage
Does not give best soln

TC of Best First Search < TC of BFS, DFS, DFID