26/7/17

Artificial Intelligence

What is intelligence?

→ Is it the ability to reason?

→ Is it the ability to acquire knowledge and apply knowledge?

→ Is it the ability to perceive and manipulate the things in the physical world?

All are part of Intelligence

It eludes precise abstract definition

Artificial Intelligence
True Artificial Intelligence
Super Artificial Intelligence

Artificial Intelligence was coined by John ~~McCarthy~~ McCarthy in 1956 (mathematics professor)

The concept developed over 756 BC before.

It is an amalgam of various knowledge, representation and knowledge manipulation techniques

AI problem vs Simple problem — Discover

27/7/17

Artificial Intelligence Problem

No precise method, model and algorithm available to solve the problem. That are characterized by imprecision and uncertainity.

Probability
↑
Uncertain
Imprecise
↓
Fuzzy logic

→ How to solve them?

Lacks common sense
M.I.

* Information are stored in symbolic form
* Search using symbolic logic

→ Traditional AI methods → Implementation of these methods are based on symbolic logic.

Computer as good as manipulating symbols equivalent to that of manipulating numeric values

1/8/17 How to solve AI problems?

4 parts / steps

Symbolism
manipulation of symbols

* Define the problem precisely.
* Analyze the problem.
* Represent and isolate knowledge that are required to solve the problem.
* Apply appropriate problem solving technique to solve the problem

Example Playing chess

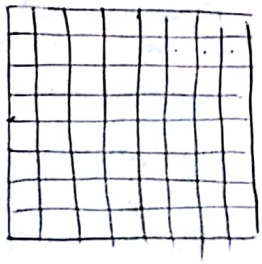
Aim: To build a program that could play chess

① First we shold specify: the starting position of the chess board.
② The rules that define legal moves
③ The board that represent the win for one player or other.

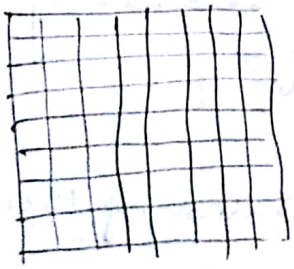
Goal: Not only play a legal chess game but also win the game if possible

32 symbols

current position



8×8



final position after applying specify rule

$$\frac{10^{120} \times 10^{-9}}{60 \times 60 \times 24 \times 365}$$

Problem State space search

① Space
② Time
③ Enough free

2/8/17

State: A situation in the world described by some facts is a state.

In physics: a state is a set of measurable physical parameters fully describing some parts of the universe.

In computer Sc. A process state is a set of values currently in register, buffer and stack. So a state is a snapshot of a process in time.

Diff. these two:

Continuous	Discrete
Instaneous	Non Instaneous.

State includes everything important about a situation in one bundle i.e., everything necessary to reason about it

In AI it usually means a list of facts (there may be lot of facts and often we want only the facts ~~that~~ required to reason about states)

methods to problem solving
a compress these facts.

Tower of Hanoi

Water Jug problem

There are given two jugs ~~of 4 liters~~ a 4 l and a 3 l. Neither has any measuring markers on it. There is a pump that can be used to fill the jug with water.

How can you get exactly 2 liter of water in 4 lt jug.

0,1,2,3,4 jug. → 0,1,2,3

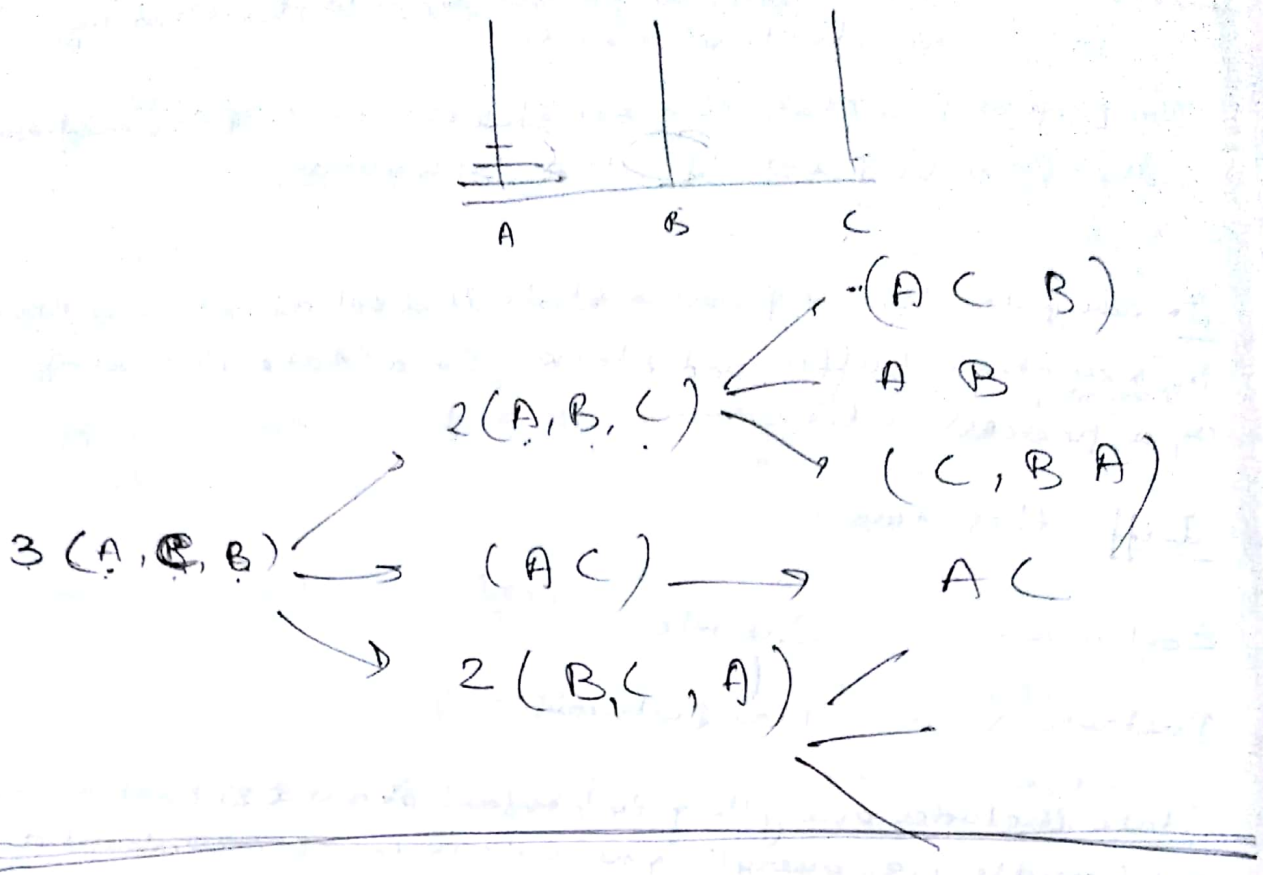
(X, Y)

X → water in 4 lt.

Y → water in 3 lt

(A, B, C) → Stack

A → initial, B → final, C → Intermidiate.



3/8/17 **Rules for water Jug Problem**

Initial State : (0,0)

Final state : (2,0)

~~(x,y) → (, y+1~~

(0,0)

Rules: (1) (x,y) → (4,y) Fill 4lt jug
if x<4. (Pre condition)

(2) (x,y) → (x,3) Fill 3lt jug
if y<3 (precarditia

~~(3) (4,y) → (4,3) Fill 3lt jug~~

(4) (x,y) → (0,y) Empty 4lt jug
if x>0

(5) (x,y) → (x,0) Emty 3lt jug
if y>0

~~(x, y)~~ ⟶ ~~(1, 3)~~

(x, y) ⟶ (y, 0)

~~if y = 0~~
~~x = 0~~

* (x, y) ⟶ ($\overset{4}{\cancel{x+y}}$, [y − (4−x)]) Pour water from 3lt jug into 4 lt jug until the jug is full
if (x+y) ≥ 4 and (y > 0)

* (x, y) ⟶ (x − (3−y), 3) Pour water from 4lt jug into 3lt jug until the jug is full
if (x+y) ≥ 3 and (x > 0)

H.W Complete set of rules for WJP

8/8/17 **Rules for water Jug Problem**

Initial State : (0,0)
Final State : (2,0)

(x, y)
↙ water in 4lt jug
↘ water in 3lt jug

x = 0, 1, 2, 3, 4
y = 0, 1, 2, 3

Solution:
(0,0) ⟶ Initial
(0,3)
(3,0)
(3,3)
(4,2)
(0,2)
(2,0) ⟶ final

(0,0)
↙ ↓ ↘

State Space : S = D × M
D ∈ {0, 1, 2, 3, 4}
M = {0, 1, 2, 3}
S = (x, y)
x = water in 4lt jug
y = water in 3lt jug

① (x, y) ⟶ (0, y) Empty 4lt jug Operation Rule
if x > 0

② (x, y) ⟶ (x, 0) Empty 3lt jug
if y > 0

③ (x, y) ⟶ (4, y) Fill the 4lt jug
if x < 4

④ (x, y) ⟶ (x, 3) Fill the 3lt jug
if y < 3

P.T.O

(4) $(x, y) \longrightarrow (4, y-(4-x))$
if $x+y \geq 4$ and $y > 0$

Transfer water from 3lt do 4lt jug until 4lt jug is full

(5) $(x, y) \longrightarrow (x-(3-y), 3)$
if $x+y \geq 3$ and $x > 0$

Transfer water from 4lt to 3lt jug until 3lt jug is full

(6) $(0, y) \longrightarrow (y, 0)$
if $y > 0$

Empty 3lt jug into 4lt jug

(7) $(x, 0) \longrightarrow (0, x)$
if $x > 0$ and $x \leq 3$

Empty 4lt jug into 3lt jug.

~~(8) $(x, y) \longrightarrow (x+y, 0)$~~
~~if $x+y \leq 4$~~

8/8/17

Analysing AI Problems

w.r.t. Seven AI Problems characteristics

1. Is the problem decomposable into independent subproblems or not.

Decomposable ← into independent subproblems

Solve the problem computing the expression

$$\int (x^2 + 3x)\, dx$$

$$= \int x^2\, dx + \int 3x\, dx + C$$

$$= \frac{x^3}{3} + \frac{3x^2}{2} + C$$

Rules ←

CLEAR(X) → ON(X, TABLE)
Block X has nothing on it

Operation : Pickup X and put in on the table

CLEAR(X) and clear(Y).
→ ON(X, Y)
put X on Y

Block world

Start ? Goal

Not Decomposable

Subproblem are not independent.

2. Can Solution steps be ignored or undone?

P1: Mathematical Theorem Proving	P2: 8 puzzle problem *	P3: Playing chess
Aim:  Initial → Final		Planning Process
Ignorable: Solution steps can be ignored	Recoverable: Solution steps can be undone	Irrecoverable: Solution steps can not be undone

* The 8 puzzle is a square tray in which are placed eight small tiles the remaining ninth square is uncovered. Each tile has a number on it. A tile that is adjacent to the black space can be slided into that space. The game consists of a starting position and a specific goal position.

Requires simple control structure that never backtracks. Easy to implement	Recoverable problems can be solved by slightly more complicated control strategy that sometimes makes mistakes. Backtracking is necessary to recover from mistakes Implemented using stack	can be solved by a system that expends a great deal of effort for making each decision Since the decision must be final

9/8/17 3. Is the problem Universe Predictable?

Everytime we make a move we know exactly what will happen. So it is possible to plan an entire sequence of moves and be confident what will be the resulting states.

Bridge game (Playing cards)

Uncertain Outcome
Probability and Plan revision
There is no feedback from environment

8 puzzle game

Certain outcome
No feedback from environment
~~Probability & Plan revision~~
~~would feedback from environment~~

We use planning to avoid having do undo actual move. Backtracking is necessary during the process

4. Is a good solution Absolute or Relative?

Ex:
Consider the following facts on a database

(1) Marcus was a man
(2) Marcus was a pompeian
(3) Marcus was born in 40AD
(4) All men are ~~mortal~~ mortal
(5) All Pompeian died when the volcano erupted in 79 A.D
(6) No mortal lives longer than 150 years
(7) It is now 2017 AD

(8) Marcus was a pompeian (2)
(9) All Pompeian died when the volcano erupted in (5)
79 A.D
(10) It is now 2017 AD (7)

Marcus died in 79AD.

(X) Marcus is dead

---

(8) Marcus was a man
(4) All men are mortal
] Marcus was mortal

(8) Marcus was born in 40AD
(3) It is now 2017 AD
] Marcus age is 1977

(6) No mortal lives longer than 150 yr

(X) Marcus is dead

This problem solution is good and absolute since once we arrived ~~the~~ at the solution we need not to check for other way of solving the problem and find ~~of~~ other solution.

Traveling Salesperson Problem: is Relative

6. What is the role of Knowledge?

Newspaper understanding

Knowledge requirement is multiple and vast (not unlimited) (many domains)

Playing chess

Limited Knowledge

Ex: ✓ Rules determine the legal moves
✓ Control strategy mechanism
✓ Good strategies and tactics to constrain search and speed up execution.

7. Does the task took interaction with the person?

Solitary

For which the computer is given a problem description and produce an answer with no intermediate communication and no demand for an explanation of the reasoning process

Conversational

In which there is intermediate communication between a person and the computer either to provide additional assistance to the computer or to provide additional information to the user or both

10/8/17

5. Is a Good Solution a State or a Path?

The bank President ate a dish of pasta salad with a fork

bank → bank of a river ✗
bank → financial institution ✓

dish → container ✗
dish → item of food ✓

Water Jug Problem

Ex: Analyze Tower of Hanoi and 8 puzzle w.r.t 7 A.I characteristics.

(**) **8 puzzle game**

(1) Is the problem decomposable into independent subproblems?

No, because to move a block to correct position we need to follow specific sequence of steps that are not independent of one another

(2) Can the solution steps be ignored or undone?

The steps can be recoverable. Hence the solution can be undone using backtracking

(3) Is the problem universe predictable?

~~No, there is vast no. of states. that is 8! which is 40320. So it is difficult to to predict the problem universe certain Outcome~~

Yes, everytime we make a move we know exactly what will happen. So it is possible to plan an entire sequence of moves and be confident what will happen next.

(4) Is a good solution is Absolute or Relative?

In case of 8 puzzle game solution is Absolute because once we reach the final solution state we need not check for other possible way of solving the same problem.

(5) Is a good solution state or path?

Here in this case the solution is a path. Because we need to follow a specific path and ~~plan to~~ make plan to avoid mistakes as the ~~solution~~ problem universe is predictable.

(6) What is the role of Knowledge?

Knowledge required to solve the problem is limited. We only need to know:

1. How to move a block
2. What block we can move on the empty space.

(7) Does the task took interaction with the person?

No, to solve a puzzle game it doesn't need to be interacted with a person. So it is solitary

(*) Tower of Hanoi

(1) Is the problem decomposable into independent subproblems?

~~No, because~~ It is not decomposable because to move n-discs from A → C through B we need to first:

1. Move (n-1) disc to A → B
2. Then to move the nth disc (largest) to C
3. Then to move the (n-1) disc to (B → C)

This process must be followed recursively in proper sequence that is given so ~~Hence~~ the steps are not independent and hence not decomposable.

(2) Can the solution steps be ignored or undone?

The solution steps can undone and they are recoverable. Steps can be undone using backtracking.

(3) Is the problem universe predictable?

Yes, the problem universe is predictable. Since when we make a move we know exactly what will happen next. We can plan the entire sequence of moves and avoid making mistakes.

(4) Is a good solution is Absolute or Relative

In tower of Hanoi problem ~~steps~~, solution is ~~Relative~~ Absolute because once we get to the solution we need not find other way of solving the problem.

(5) Is a good solution is state or path?

In this problem, a proper sequence must be ~~followed~~ followed to reach the goal state. We have to make plan to avoid mistakes. Tower of Hanoi can be solved by following the given recursive sequence

(1) Move $(n-1)$ disc from $A \to B$
(2) Move n^{th} disc (largest) $A \to C$
(3) Move the $(n-1)$ disc from $B \to C$

where A is the initial tower and C is the final

So the good solution is a path to the find state

(6) ~~Role~~ what is the Role of Knowledge?

Knowledge required is very limited. We only need to ~~note~~ know:

① ~~How~~ to move a disc from one tower to another

② We can't keep a ~~big~~ larger disc onto a smaller disc

③ We can use any of tower to move a disc.

④ We can't keep a disc anywhere else expect the towers.

(7) Does the task took interaction with the person?

No it does not need interaction with the person. Once the input has been given (that is the no. of disc) It can carry the task all alone and therefore it is solitary.

16/8/17 **The Missionaries and Cannibals Problem**

Three missionaries and three cannibals find themselves on side of a river. They have agreed that they would all like to get to the other side. But the missionaries are not sure what else the cannibals have to agreed to. So the missionaries want to manage the trip across the river in such a way that the number of missionaries on ~~the~~ either side of the river is ~~one~~ never less than the number of cannibals who are on the same side. The only boat available holds only two people at a time. How can every one gets across the river without missionaries risking being eaten.

(X, Y) → No. of cannibals of on the same side (0, 1, 2, 3)

↓ No. of Missionaries

(0, 1, 2, 3)

Initial state (0, 0)

Final state (3, 3)

(X, Y, Z) → Side of the ~~where~~ boat (0 → Start) (1 → Dest)

↓ No. of missionaries (0, 1, 2, 3)

↓ No. of cannibals (0, 1, 2, 3)

$(m_1, m_2, m_2, c_1, c_2, c_3, b_s, b_d)$

We don't bother about individuals.

① (X, Y, Z) → (0, 2, 1) ~~X-2, Y-2, Z=0~~

② (X, Y, Z) → (X-2, Y, 1) ~~X-2, Y-2~~

③ (X, Y, Z) → (X-1

(X, Y) night

(3, 3) →

↓

(0, 0)

(3, 3)

(3, 1)

~~(3, 2)~~

~~(3, 0)~~

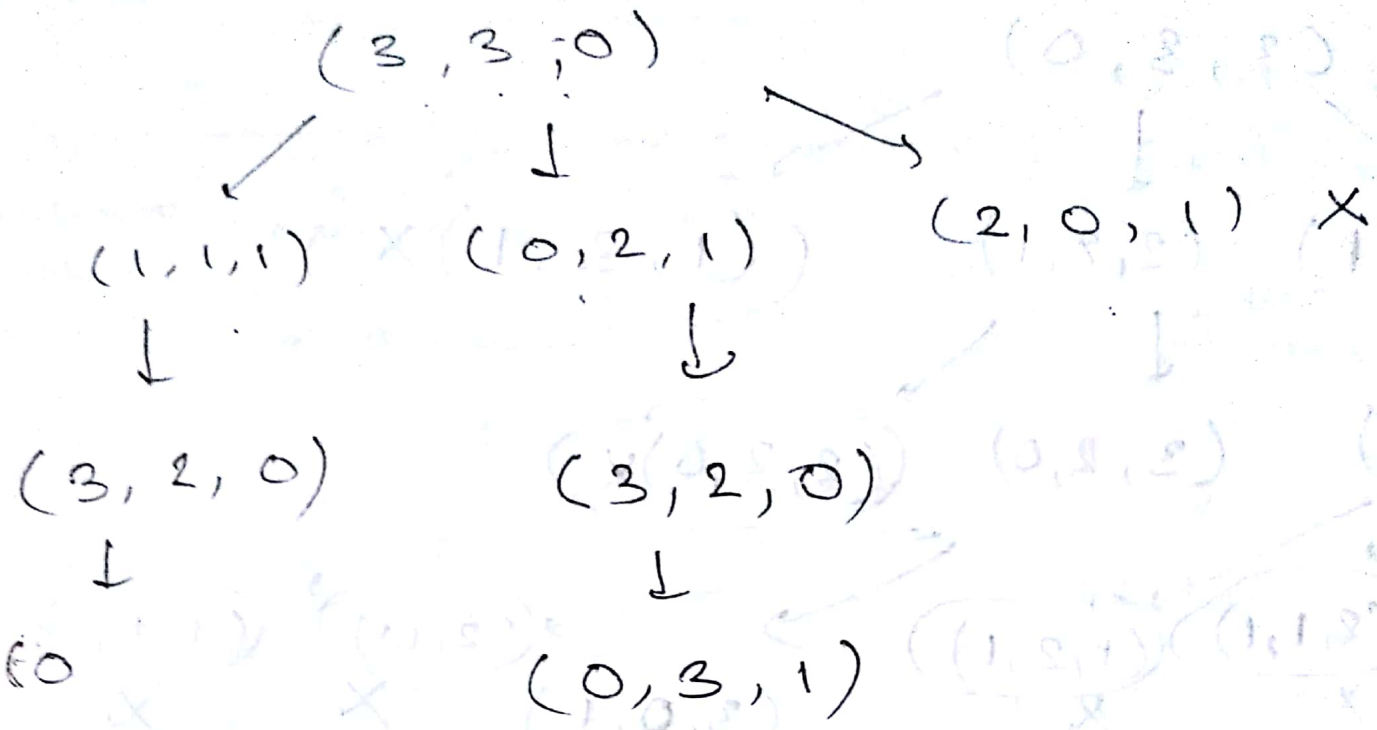
(3, 2)

(3, 0)

(3, 1)

(1, 1)

(2, 2)



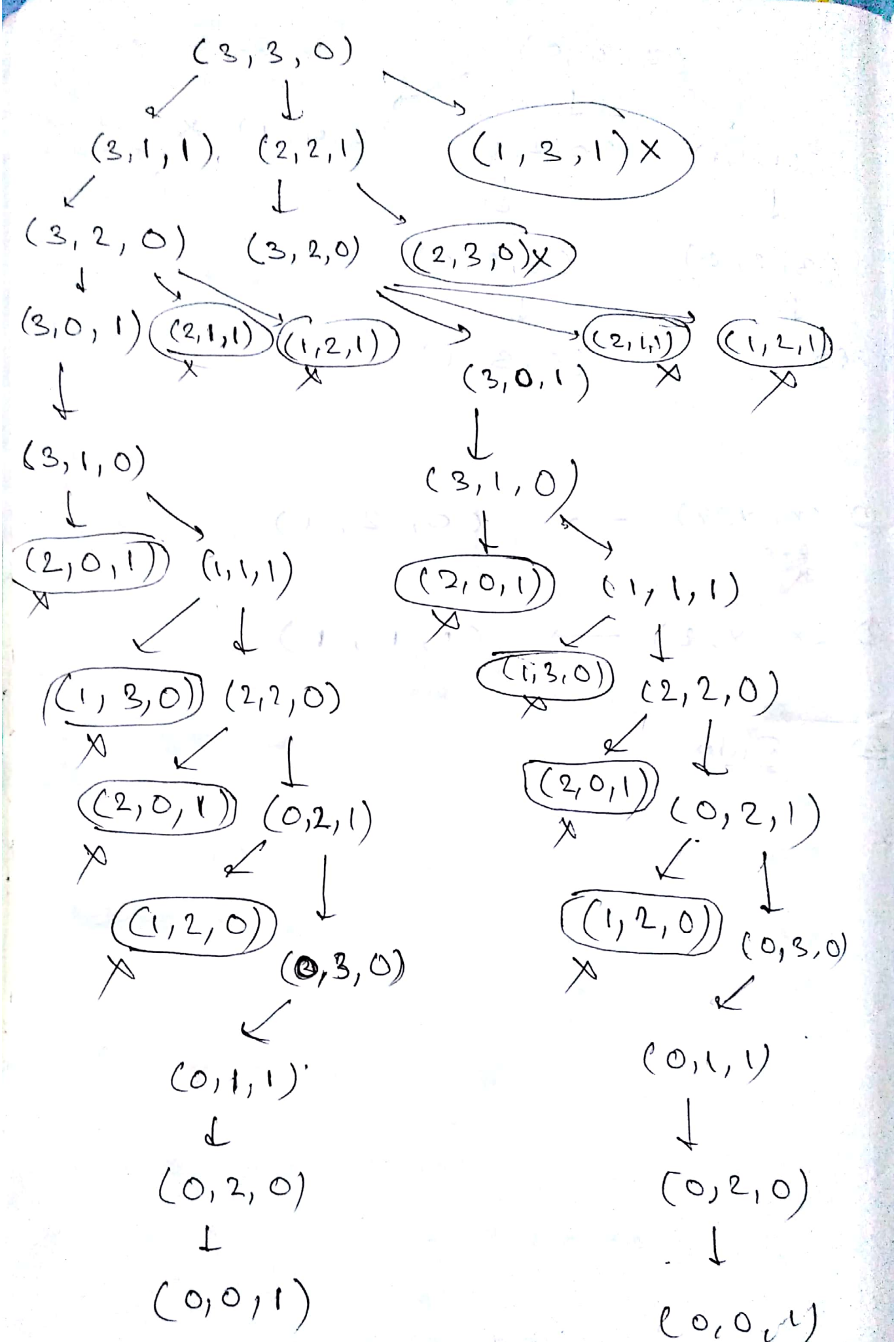
① $(x, y, z) \longrightarrow (0, 2, 1)$

$z = 0$

x

② $(x, y, z) \longrightarrow (1, 1, 1)$

③ Rules



Missionaries Cannibals problem :

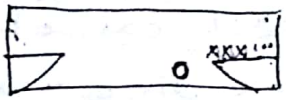
(X, Y, Z) → (Side on which the boat is residing) 0 : right, 1 : Left

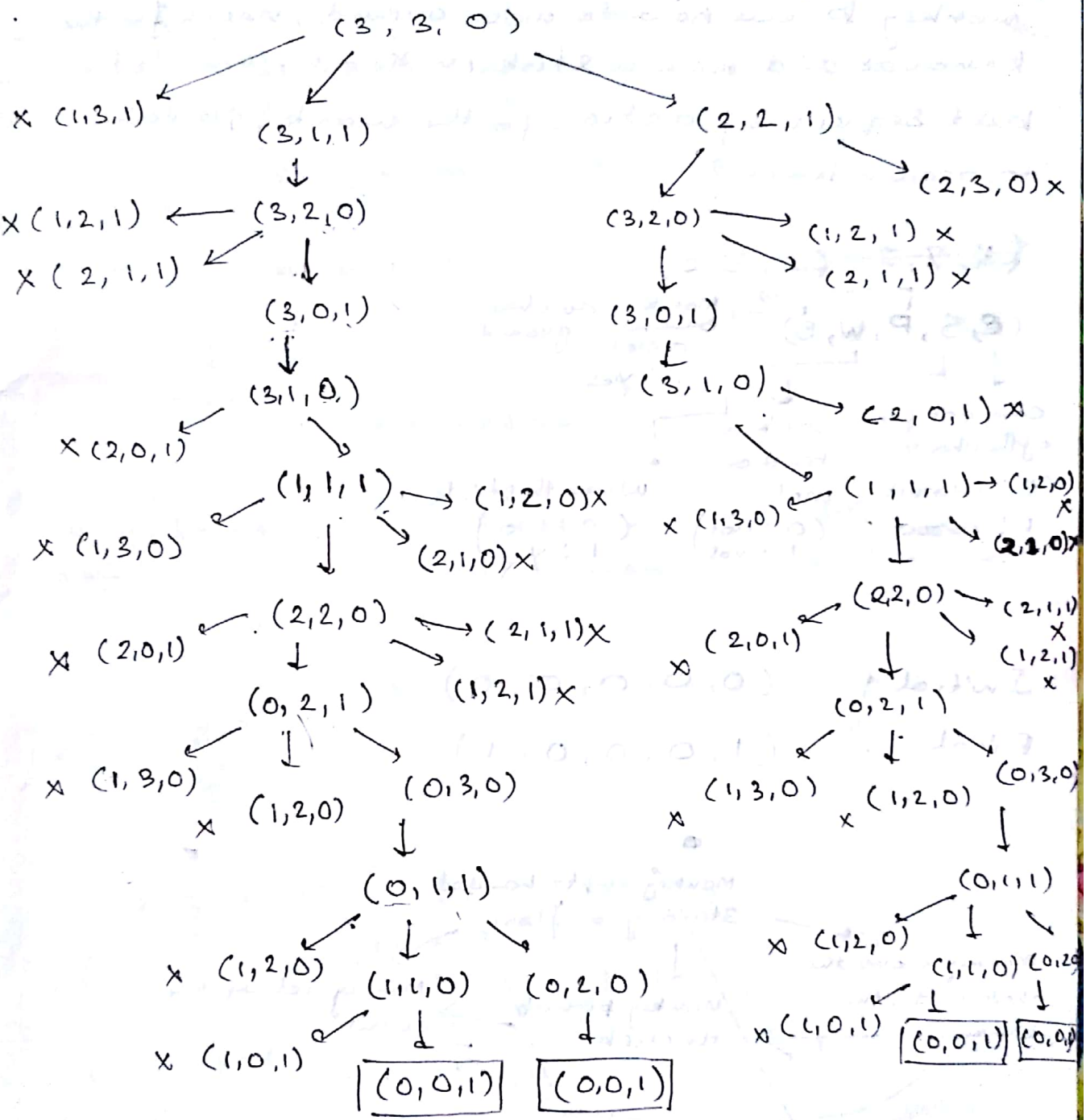
X → No. of Missionaries (0, 1, 2, 3) on rt. side

Y → No. of cannibals (0, 1, 2, 3) on rt. side

Initial State : (3, 3, 0)

Final State : (0, 0, 1)





Rules

1. Move 1 cannibal through the boat
2. Move 2 cannibal through the boat
3. Move 1 missionary through the boat
4. Move 2 missionary through the boat
5. Move 1 missionary & 1 cannibal through the boat

21/8/17

A hungry monkey finds himself in a room in which a bunch of bananas is hanging from the ceiling. The monkey unfortunately cannot reach the bananas, however in the room there are also a chair and a stick. The monkey knows how to move around, reach for the bananas and wave a stick in the air. What is the best sequence of actions for the monkey to take to acquire lunch?

~~(X, Y, ?)~~ (A, B, C)

(C, S, P, W, B)

Monkey on chair
Banana on ground
(0 : No
1 : Yes)

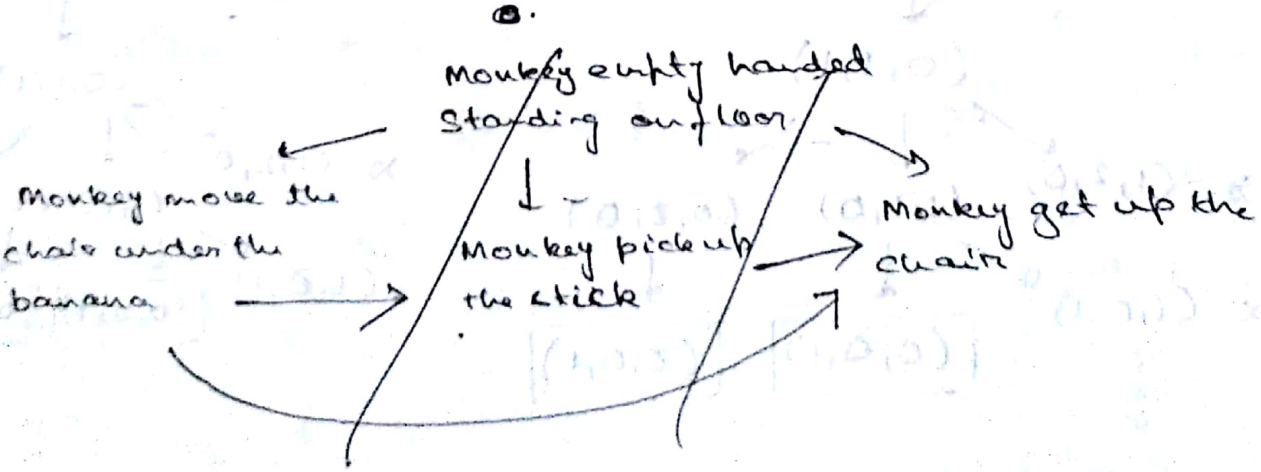
C: connect pos. of the chair
(0 : incorrect
1 : correct

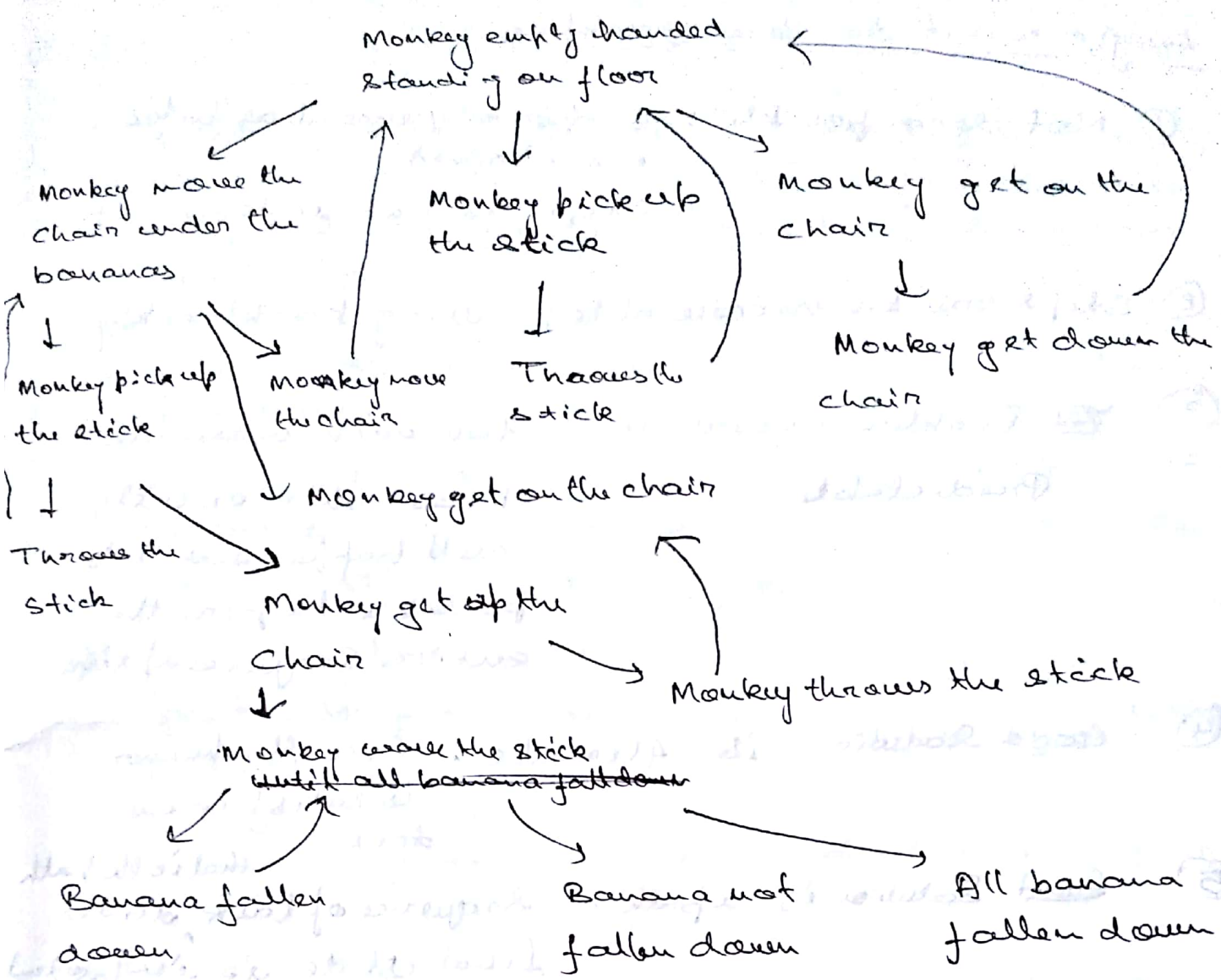
S: Stick in hand or not
(0 : not
1 : yes)

W: Wave the stick
(0 : No
1 : Yes)

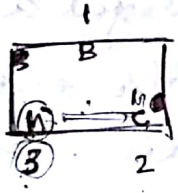
Initial : (0, 0, 0, 0, 0)

Final : (1, 0, 0, 0, 1)





MC HB



(1, 2, 3, 0, 0) $\xrightarrow{GO(2)}$ (1, 2, 2, 0, 0)

(1, 2, 2, 0, 0) $\xrightarrow{MOVE(1)}$ (1, 1, 1, 0, 0)

(1, 1, 1, 0, 0) $\xrightarrow{GO\text{-}UP}$ (1, 1, 1, 1, 0)

(1, 1, 1, 1, 0) $\xrightarrow{GRAB}$ (1, 1, 1, 1, 1)

Analyze w.r.t to Water problem

① Not decomposable : Proper sequence has to be maintained.
Steps are not independent

② Steps can be recoverable : using backtracking

③ Problem universe is : For each move we
Predictable Know what exactly will happen and it is possible to plan the entire sequence of steps

④ Good Solution is Absolute : Once the target is satisfied we done.

⑤ Solution is a path : Sequence of task to the final state is important that is the path

⑥ Knowledge requirement : Only legal actions
is limited and the outcomes of the action should be known

⑦ The problem is solitary : It doesn't need interaction with a person to perform any task.

22/8/17

State Space Search

It is a way of solving the problem by starting with initial state, using a set of rules to move from one state to another state and attempting to end with one of a set of final states.

This corresponds to the way of solving problems in two important ways:

1. It allows for a formal definition of a problem as need to convert some given situation into desired situation using a set of permissible rules
2. It permits us to define the process of solving a particular problem as a combination known techniques and search.

Necessary things to provide Formal definition description of AI Problems

1. Define problem state space: contains all possible configuration of relevant objects
2. Specify one or more initial states within the state space that describe possible situation from which problem solving process may start. These are called initial states
3. Specify one or more states that would be acceptable as solution to the problem. These states are called goal states
4. Specify a set of rules that describe the actions available

H.W — What are the unstated assumptions are present in the informal problem description?

- How general should the rules can be?
- How much work required to solve the problem should be pre computed.

Production System

Consists of:

1. Set of rules
2. One or more data / Knowledge base
3. Control strategy ⟷ It will always cause the motion. It must be systematic.
4. Rules apply

State Space

$S: \quad D \times D \times D \times D \times M \times M \times M \times M$

$D = (0, 1, 2, 3)$

$M = (0, 1)$

Wow stick (0,1) — Have Banana (0,1)

$S: \quad (B, C, S, M, AS, WS, MC, HB)$

- B: Pos. of Banana
- C: Pos. of chair
- S: Pos. of Stick
- M: Pos. of Monkey
- AS: Acquire Stick (0,1)
- WS: Monkey on chair (0,1)

Initial State: $(0, 1, 2, 3, 0, 0, 0, 0)$

Final State: $(0, 0, 0, 0, x, 1, x, 1)$

Rules: GO(Y) | Monkey goes to position Y (with or without stick)

$(B, C, S, M, AS, WS, MC, HB) \xrightarrow{GO(Y)} (B, C, S, Y, AS, WS, MC, HB)$

if $Y \neq M \quad Y \in \{0, 1, 2, 3\}$

MOVE(X, Y) | Monkey moves item X to position Y

$(B, C, S, M, AS, WS, MC, HB) \xrightarrow{MOVE(chair, Y)} (B, Y, S, Y, AS, WS, MC, HB)$

if $C \neq Y$ and $Y \in \{0, 1, 2, 3\}$ and $C = M$

$(B, C, S, M, AS, WS, MC, HB) \xrightarrow{MOVE(stick, Y)} (B, C, Y, Y, AS, WS, MC, HB)$

~~MOVE (X, Y)~~ if $S \neq Y$ and $Y \in \{0, 1, 2, 3\}$ and $S = M$

GRAB | ~~Monkey grabs the item X~~ ~~GRAB~~

~~$X \in \{Stick, Banana\}$~~ Monkey grabs the banana and eat it

~~$(B, C, S, M, AS, WS, MC, HB) \xrightarrow{GRAB(stick)} (B, C, S, M, 1, WS, MC, HB)$~~

~~if $AS = 0$, $M = S$~~

$(B, C, S, M, AS, WS, MC, HB) \xrightarrow{GRAB(Banana)} (B, C, S, M, AS, WS, MC, 1)$

if $HB = 0$, $M = B$, $WS = 1$

~~RELEASE THROW (X)~~ | ~~Monkey throws the item X~~ release

~~$X \in \{STICK\}$~~

~~$(B, C, S, M, AS, WS, MC, HB) \xrightarrow{RELEASE(stick)} (B, C, S, M, 0, W)$.~~

INTERACT_STICK | Monkey ~~throws~~ grabs or releases the stick

$(B, C, S, M, AS, WS, MC, HB) \xrightarrow{Interact\_stick} (B, C, S, M, \overline{AS}, WS, MC, HB)$

if $M = S$

~~Interact_chair GO UP~~ | Monkey goes up on down the chair

$(B, C, S, M, AS, WS, MC, HB) \xrightarrow{Interact\_chair} (B, C, S, M, AS, WS, \overline{MC}, HB)$

if $C = M$

WAVE. | Monkey waves the stick

$(B, S, S, M, AS, WS, MC, HB) \xrightarrow{WAVE} (B, C, S, M, AS, [redacted], MC, HB)$

if ~~B, C, S~~, $B = C = S = M$, $AS = 1$, $MC = 1$

Solution:

Initial: (0,1,2,3,0,0,0,0)

↓ GO(1)

(0,1,2,1,0,0,0,0)

↓ MOVE(chair, 0)

(0,0,2,0,0,0,0,0)

↓ GO(2)

(0,0,2,2,0,0,0,0)

↓ INTERACT-STICK

(0,0,2,2,1,0,0,0)

↓ ~~GO(0)~~ MOVE(stick, 0)

(0,0,0,0,1,0,0,0)

↓ INTERACT-CHAIR

(0,0,0,0,1,0,1,0)

↓ WAVE

(0,0,0,0,1,1,1,0)

↓ GRAB

(0,0,0,0,1,1,1,1)

24/8/17

Searching ~~Problems~~ Methods

Search Process: can be view as a traversal through directed graph in which node represent the problem state and each ~~edge~~ arc represent the relationship between the node it connects.

The search process must find a path through directed graph starting at an initial state and ending with one or more final state.

Five important issues that are related to all search process:

- Direction in which to conduct the search

The Objective of a search process is to discover a path through a problem space from an initial state to a goal state.

There are two direction in which a search process could process.

1. Forward from start state.
2. Backward from the goal state.

Depending upon topology of the problem space, it may be significantly more efficient to conduct search in one direction rather than the other. There are three factors on which the direction of the search depends.

(i) Are more possible start state or goal state

It is desirable to move from smaller set of states to a larger set of states.

(ii) In which direction the branching factor is higher?

It is better to proceed in the direction of lower branching factor

(iii) Will the program be asked to justify its reasoning process to a user?

- It is important to proceed in the direction that corresponds more closely with the way the user will think.

28/8/18 **Bidirectional Search Strategy**—

Simultaneous search forward from start states and backward from the goal state until two paths meets somewhere in between.

The bidirectional search may be ineffective when two searches may pass each other resulting in more work than it would have been taken for one of them.

- **Topology of the search process**

Problem tree VS Problem Graph

* Search process is the application of a set of production rules

Problem tree: A simple way to implement any search strategy is to perform as a tree traversal. Each of the tree node is expanded by applying the production rules to generate a set of succesor nodes. and each of which in turn be expanded to continue the process until a node that represents a solution is found.

Problem Graph: A tree search procedure can be converted to a graph search procedure by modifying action performed at time when a node is generated

i) Examine the set of nodes that have been created so far to see if the new node is already exists

ii) If it doesn't exist, simply add it to the graph just as for tree

iii) If it does already exist then perform the following three things:

① Set the node that is being expanded to point to the already existing node corresponds to its successor rather than to the new one. The new one can be simply thrown away

② If you keep track of the best path to each node then check to see if the new path is better or worse than old one. If the new path is worse then "do nothing"

else if better than
- Record the new path as correct path to get to the node and propagate corresponding changes down through successors as necessary

20/08/17

How each node of the search process be represented?

Problem of Knowledge Representation
1. How individual objects / facts can be represented.
2. How these representations can be efficiently combined to form the complete representation of the problem state.
3. How can one represent the sequence of problem state that arise out of a search process efficiently?

→ Particularly important in the context of a search process

- Selecting the applicable rules: (Matching & indexing)

The clever search involves chosing from among the rules that can be applied at a particular point the ones that are most likely lead to a search.
Extracting rules that can be applied from the collection of rules requires some kind of matching between the current state and pre-conditions of the rules.

- Using heuristic function to guide the search

What is heuristic?
A heuristic is defined as a technique that aids in the discovery of solution to a problem even though there is no gurantee that it will never lead to a wrong direction.

11/9/17

Depth First Search

DFS: Pursue a single branch of a problem until it yield to a solution or until some pre-specified depth has been reached. Only then go back and explore other branches.
Alternative node at the same level are ignored completely as long as there is a hope of reaching the goal using the original choice

Algorithm :

The node at the top the tree. Node of the no moment

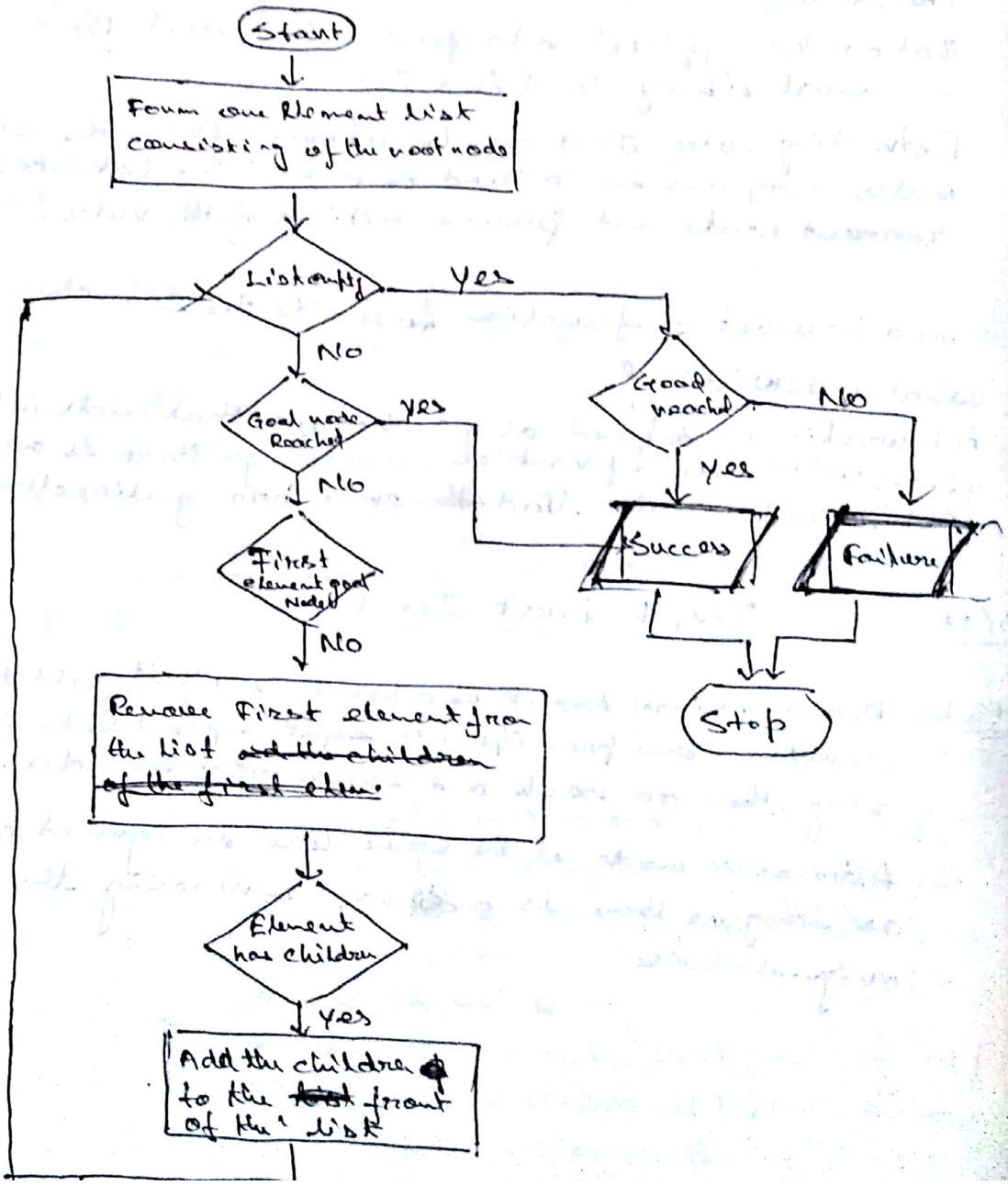
Step 1: Form a one element list consisting of the root node.

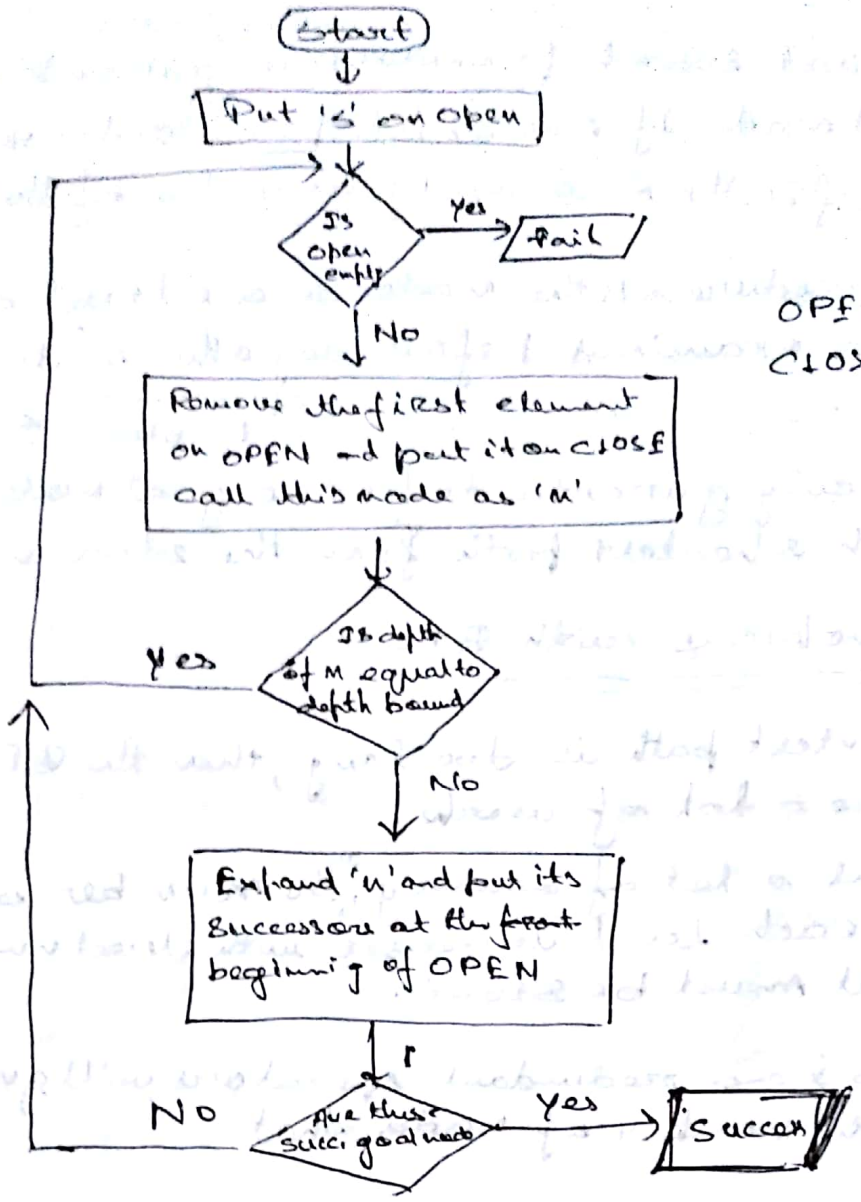
Step 2: Until the list becomes empty or the goal node has been reached, repeat the following :

(2a) If the first element is goal node do nothing

(2b) If the first element is not the goal node then remove first element from the list and add the children of the first element if any to the front of list.

Step 3: If the goal node has been reached announce SUCCESS else the FAILURE otherwise.





OPEN } are queues
CLOSE }

DFS flow chart.

Breadth First Search

Algorithm:

Step 1: Form a one element list consisting of the root node

Step 2: Until the list becomes empty or the goal node has been reached, repeat the following

(2a) If the first element is goal node do nothing

(2b) If the first element is not the goal node then remove first element from the list and add the children of the first element if any to the rear/back of list

Step 3: If the goal node has been reached announce SUCCESS else FAILURE

otherwise

A breadth first search procedure is guranteed to find a goal node if one exist provided that there are a finite number of branches of the tree.

In this procedure all the nodes on one level of the tree are examined before any other node in the next level.

optimization Problem ←

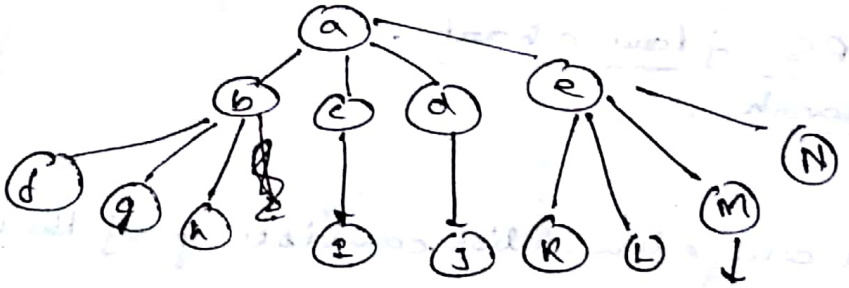
It is not only gurantee to find a goal node but the one with shortest path from the start node

Major Problems with BFS

① If the shortest path is too long, then the BFS requires do a lot of work.

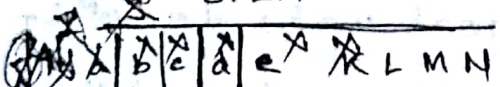
② It required a lot of memory, as the number of nodes at each level increases with level number and they all must be stored.

③ If irrelevant and redundant operators will greatly increases the number of nodes that



D.F.S

OPEN



N=a N=c N=d N=e

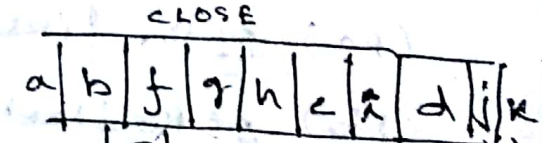
N=b N=i N=j N=k

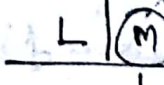
N=f N=l

N=g N=m

N=h

CLOSE





Goal Node reached

14/9/17

Hill climbing

It is a variant of the depth first search in which the feedback from the test procedure is used to help the search to move in a specific direction

Algorithm

Step 1: Form one element queue consisting of root node

Step 2: Repeat the following until the goal node has been found or until the queue becomes empty.

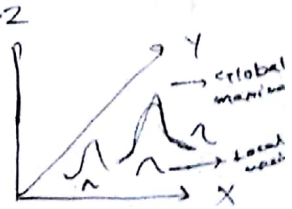
(2a) If the first element of the queue is a goal node then do nothing

(2b) If the first element is not the goal node then
(i) Remove the first element from the queue
(ii) Sort the first element children if any by the estimated remaining distance to goal node and add the node to the front of queue

(3) If the goal node has been found then announce SUCCESS otherwise FAILURE

```
Function hill()
    queue = [ ]
    State = root_node
    while true
    {
        if is_goal(&state)
            then return success
        else
        {
            &out (succession(&state))
            add to front of queue (succession(&state))
        }
        if queue == [ ]
            then return failure
        State = queue[0]
        remove first ~~first~~ item/element from queue
    }
```

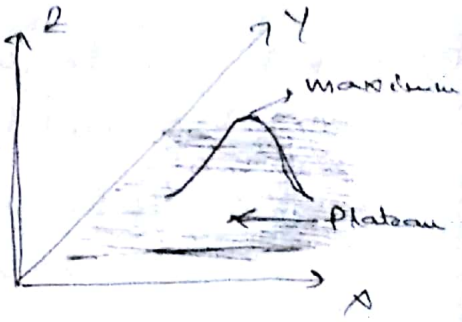
Hill climbing can be fooled by
(1) Foothill
(2) Plateau
(3) Ridge



Foothill

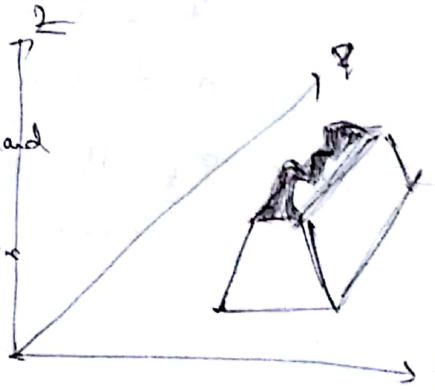
Foothills are often called local maxima by mathematicians
Local maxima attracts the hill climbing procedure like magnets
Local maximum is a part of search space that appears to be preferable to the parts around it, but which is infact just a foothill of a larger hill

Plateau



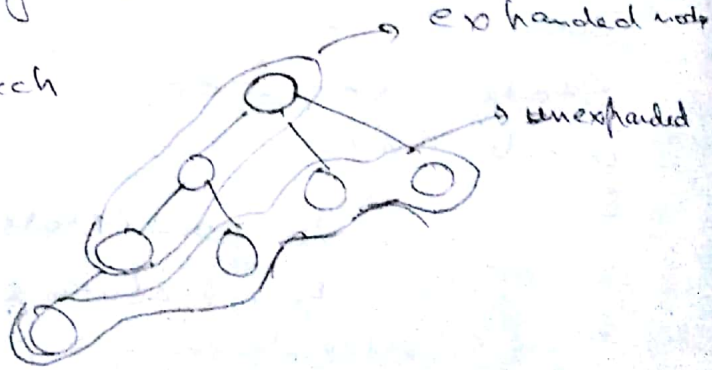
A plateau is a region in a search space where all the values of neighbourhoods are of the same. In this case although there may be suitable maximum value somewhere nearby, but there is no indication from local terrain of which direction to go to find it.

Ridge



A ridge is long thin region of highland with lower land on either side. The hill climbing algorithm would determine that any point on the top of ridge is maximum because hill falls away in all four direction is very narrow one that leads to top of ridge on

BFS
DFS
→ Best First Search



Best First Search :

Algorithm:

Step 1 : Form a one element list consisting of the root node

Step 2 : Until the list becomes empty or the goal node has been reached, repeat the following :

~~Step 3~~ (2a) If the first element is goal node do nothing

(2b) If the first element is not the goal node then remove first element from the list and add the children of the first element if any to the queue and sort the queue w.r.t estimated remaining remaining dist. from the goal node

Step 3 : If the goal node has been reached announce success otherwise failure.

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

```
Function_Best_Search()
    queue=[]
    State = root_node
    while true
    {
        if is_goal (State)
            then return SUCCESS
        else
        {
            add to the queue ( successor (State))
            sort ( State) acc. to remaining dist. to the goal nod
        }
        if (queue == [])
        return Failure
        State = queue[0]
        remove first_item from (queue)
    }
```

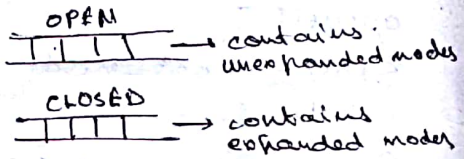
A* Algorithm (Best First Algorithm) + Implementation Detail

A more flexible way to use heuristic function is use some criteria to order all of the nodes on OPEN at every step.

In order to apply such an ordering procedure we need some measure by which to evaluate the promise of a node such a measure is called evaluation function $f(n)$

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

The value of the evaluation function $f(n)$ is an estimated cost path from start node S to the node 'n' PLUS an estimate of the cost of minimal cost path from the node 'n' to the goal node $f(n)$ is an estimate of cost of minimal cost path constrained to go through 'n'.



A*_Algorithm

Step 1: Put the start node 'S' on a list called OPEN
compute $f(s)$
Set CLOSED to empty list

Step 2: If OPEN is empty then
Exit with failure
Else
continue to the next step

Step 3: Pick up the node on OPEN with the lowest f value and call it the BEST_NODE
Remove the BEST NODE from OPEN and put it on CLOSED

Step 4: If the BEST NODE is the goal node then
"Exit with SUCCESS and the solution path is obtained by tracing back through the pointers
Else
Continue to the next step

Step 5: Expand the BESTNODE by generating its successors. For each of its successor repeat the following

(5a) Set SUCCESSOR points back to BESTNODE
[The backward pointers are useful to trace back the path once the goal node is found]

(5b) compute g(SUCCESSOR) = g(BESTNODE) + cost of getting the SUCCESSOR from the BESTNODE

(5c) See if the successor is same as any node on 'OPEN' if so call that node as 'OLD' and add it to the list of BESTNODE's successors [Simply throw away the current successor]

See whether it is cheaper to get OLD via its current parent or SUCCESSOR via BESTNODE

if OLD is cheaper
"Do Nothing"
else
SUCCESSOR is cheaper

(i) Reset the OLD's parent link to point to BESTNODE

(ii) Record the new cheaper path by updating g(OLD) and f(OLD) values

(5d) See if the SUCCESSOR was not in OPEN ~~any node~~. See if the successor is same as any node on 'CLOSED' if so call that Node as 'OLD' and add it to the list of 'BESTNODES' successors. (Simply throw away the current node)

See ~~weather~~ whether it is cheaper to get OLD via its current parent or successor via BESTNODE

If old path is cheaper then ~~computing~~
do nothing
else
(i) reset the parent link of OLD to point to BESTNODE and update the g(OLD) & f(OLD) value
(ii) we must propagate the newly found information to the successor of the OLD

For this we must perform DFS starting from the OLD change each node's g value when it is visited and terminated each branch when either a node with no children is encountered or node to which an equivalent or better path has already been found is reached.

5(e) :→ if successor was not already on either OPEN or CLOSED. Put it on OPEN and direct the point from it back to BESTNODE
Also compute

$$f(\text{SUCCESSOR}) = g(\text{SUCCESSOR}) + h(\text{successor})$$

(5f) :→ Go to Step 2

Properties of Searching methods

1. Complexity
2. Completeness
3. Optimality
4. Admissibility
5. Irrevocability

Complexity : It is useful to describe how efficient a search method is, overtime and space (memory)

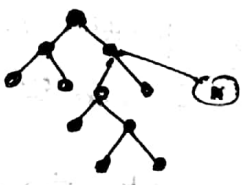
* **Time complexity** : of a search method is related to the length of time that the method would take to find a goal state

* **Space complexity** : is related to the amount of memory that the search method needs to use

DFS vs BFS

DFS → Not complete

BFS → complete



$O(b^d)$

d = depth

b = branching factor

Completeness : A search method is described as being complete if it is guaranteed to find a goal state if one exist.

18/10/17

Optimality: A search method is optimal only if it is guranteed to find the least solution that exist.

↳ it will find path to a goal state. that involves taking the least number of steps

Admissibility: A search method is admissible if it is guranteed to find the best solution in quickest possible time.

Irrevocability

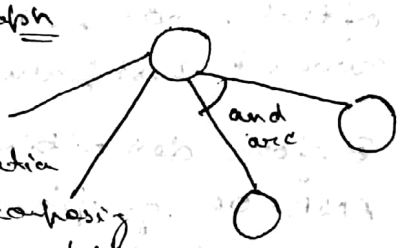
Search method that do not use backtracking and which therefore reason just one path are described as irrevocable

Tentative

Search methods that use backtracking are described as tentative

Problem Reduction Approach

OR Graph **AND-OR Graph**



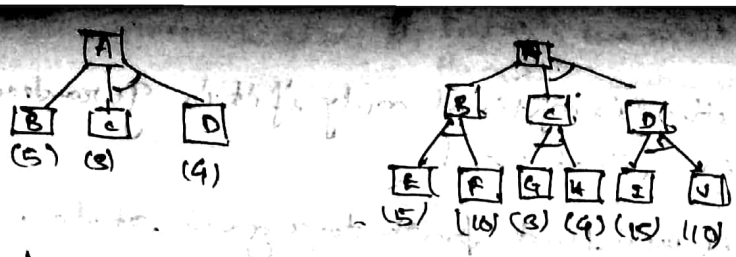
The AND-OR Graph structure

It is very useful in representing the solution to the problem that can be solved by decomposing them into a set of smaller sub-problems, all of which then must be solved.

This decomposition or reduction generates arcs that are called AND arcs. One AND arc may point any number of successor nodes, all of which must be solved in order for the arc to a solution

Just as in OR graph several arc may merge from a single node indicating varities of way that the problem can be solved

The best first search is not adequate for searching AND-OR graph



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

Assumptions

(1) The number at each node represents f value. Here we could consider only $h(n)$ and ignore $g(n)$

(2) Every Operation has uniform cost for simplicity, ie, each arc with single successor has cost of 1

Each AND arc with multiple successors has a cost of 1 for each of its components

Here the problem is that the choice of which node to expand next, must depend not only on the 'f' value of that node but also whether that node is the part of current best path from the initial node.

Before describing an algorithm for searching AND-OR graph, we need to exploit a value called as FUTILITY

↓

This is the threshold value, when the estimated cost of solution becomes greater than the value of FUTILITY, then we abandon the search

Futility corresponds to threshold, any solution with cost above it is too expensive to be practical

11/10/17

Problem Reduction Algorithm

Step 1: Initialize the graph only with the start node ← Initial node

Step 2: Repeat the following steps until the initial node is labelled as 'solved' or its cost goes above 'FUTILITY'

Step 2(a) Traverse the graph starting from the initial node by following the current best path and accumulate the nodes that are on the path & have not yet been expanded or labelled as solved.

Step 2b: Pick up one of the accumulated node in previous step and expand it.

(i) If there are no successors then assign 'Futility' as the value of the node

Else

Add its successors to the graph and compute f value for each of them.

If f value of any node is 0 (Zero) then label that node as 'Solved'

Step 2c: Change the f estimate of the newly expanded node to reflect the new information provided by its successors and propagate this change backward through the graph

If any node contains a successor whose descendents are all solved then label the node itself as 'SOLVED'

At each node that is visited while going up the graph, decide which of its successors is the promising and mark it as the part of the current best path. This may cause the current best path to change

AO* Algorithm

Assumptions:

- AO* algorithm uses a single structure 'graph'
- Each node in the 'graph' will point both down to its immediate successor and upto its immediate predecessor
- Each node in the 'graph' will also be associated with it an 'h' value: an estimated cost of the path from itself to set of solution nodes

Algorithm

Step 1: Let the GRAPH initially consist of the node representing the initial node and call this initial node as INIT and compute h(INIT)

Step 2: Until the INIT is labelled as SOLVED or until INIT's h value becomes greater than FUTILITY. Repeat the following

Step 2(a): Trace the labelled arc from INIT AO* and select for expansion one of the as yet unexpanded nodes that occurs on this path. Call this selected node as NODE

Step 2(b): Generate the successors of NODE.

2b1: If there are none then

Assign FUTILITY as 'h' value of the 'NODE'

↓

(This is equivalent to say that the NODE is unsolvable)

2b2 If there are successors of 'NODE then

for each successors that is not also an ancestor of the NODE do the following

2b2.1 Add the successor to the ~~graph~~ GRAPH

2b2.2 If the successor is the terminal node then

- Label it Solved
- Assign its 'h' value to 0 zero

2b2.3 If the successor is not the terminal node then ~~compute its 'h' value~~

- compute its 'h' value

Step 2(c): Propagate the newly discovered information up in the graph the by doing the following

2c1: Let 'S' be the set of nodes that have labelled as SOLVED or whose h values h values have been changed, so need to have h values propagate to their parents

2c1.1 Initialize S to NODE and until S becomes empty repeat the following

2.c.1.1.1 If possible select from S a node none of whose descendants in the GRAPH occurs in S. If there is no such node then select any node call this node as CURRENT and remove it from S

↓

CURRENT

2c.1.1.2 Compute the cost of each arc emerging from ~~connect~~ CURRENT. Assign the minimum of cost just computed for arcs emerging from it as its new h value

↓

CURRENT

2.C113 : Mark the best path out of CURRENT by marking the arc that had minimum cost which was computed in the previous step

2.C114 : Mark CURRENT 'SOLVED' if all of the nodes connected to it through newly labeled / marked arc have been labelled as SOLVED

2.C115 : If (CURRENT has been SOLVED) or (if the cost of CURRENT was just changed (h value) then

CURRENT's new status must be propagated back up the GRAPH so add all of its ancestor of CURRENT to 'S'

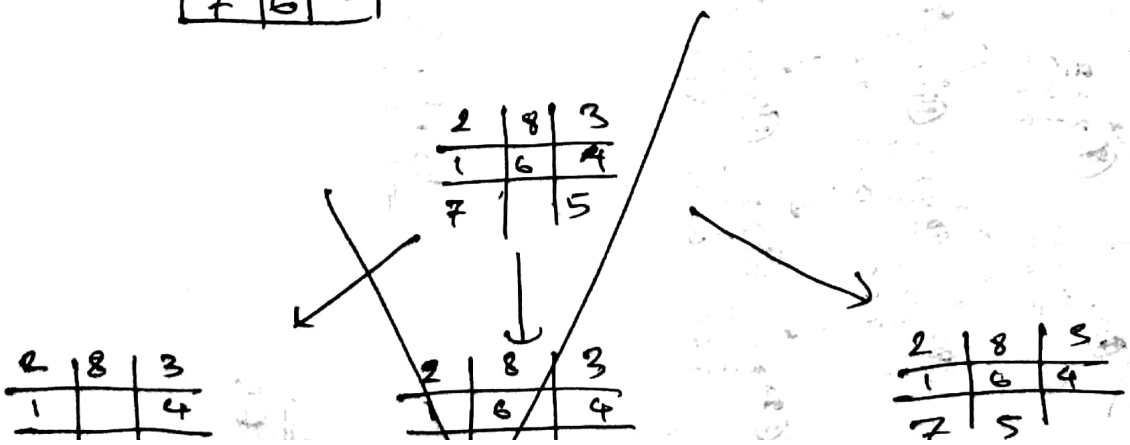
Initial State

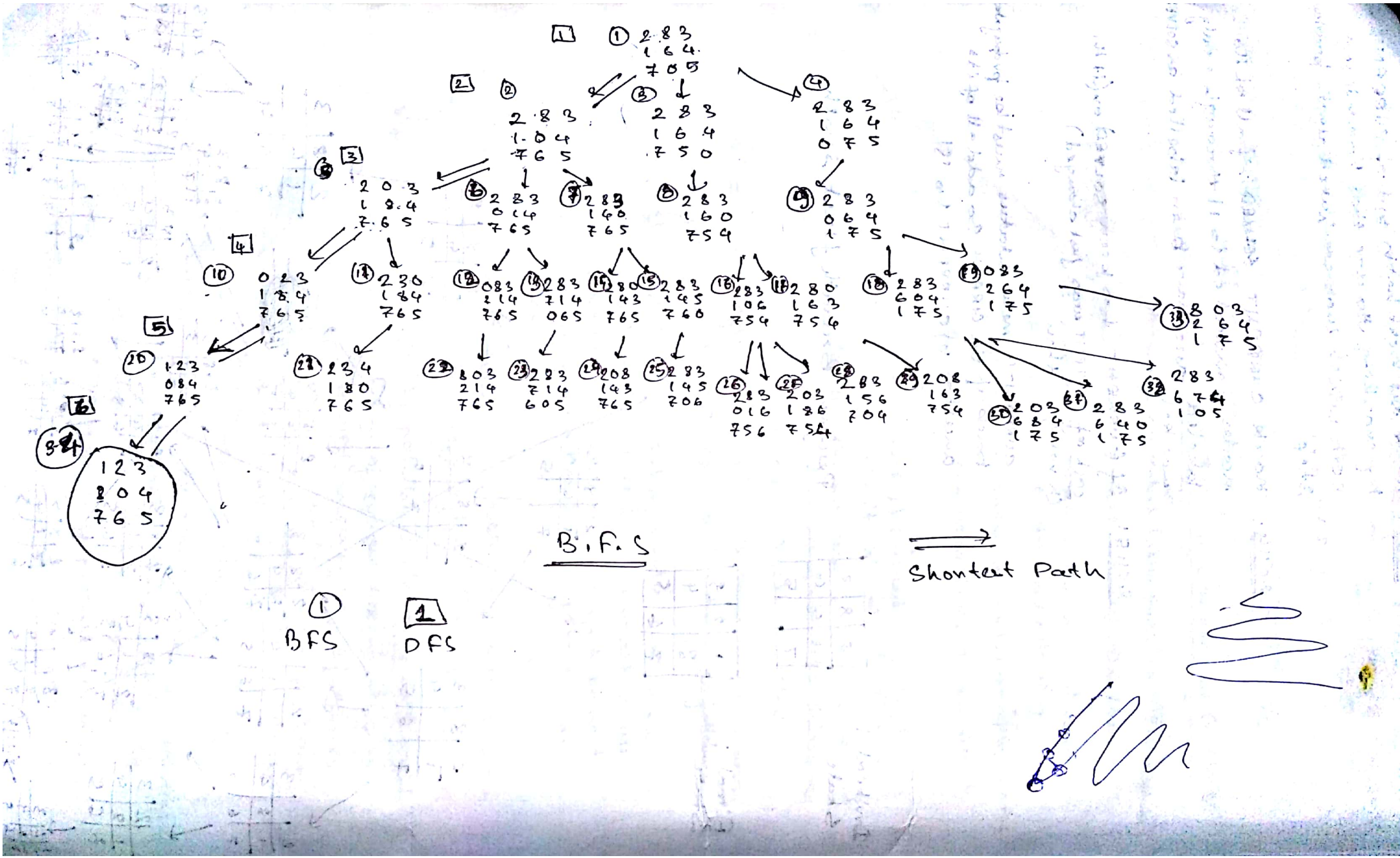
2	8	3
1	6	4
7		5

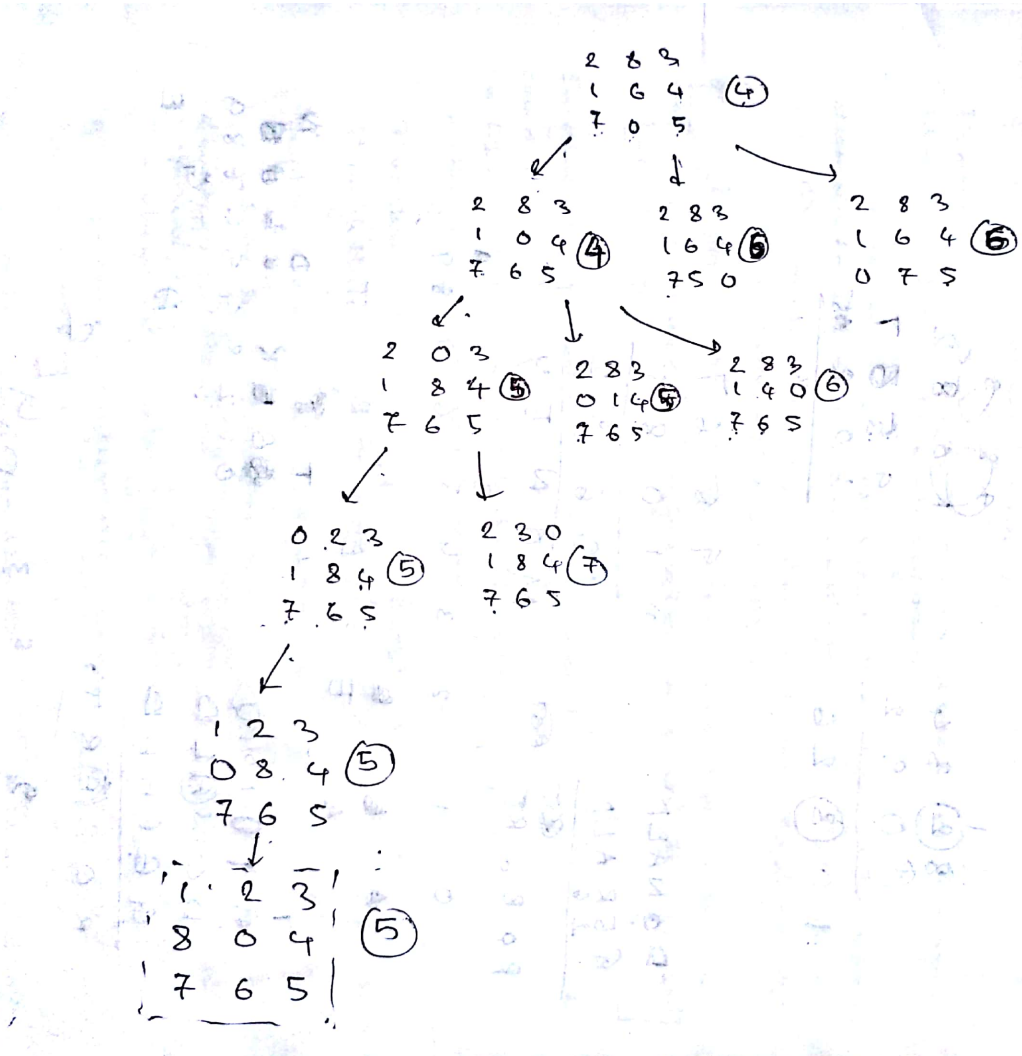
Perform Breadth first Search

Final State

1	2	3
8		4
7	6	5







SEND
+ MORE
MONEY

DONALD
+ GERALD
ROBERT

CROSS
+ ROADS
DANGER

R S

O	M	Y			E	N	D	S	R
0	1	2	3	4	5	6	7	8	9

E B = N − 1

S − 8/9

E N

8 2 3 4
1 0 9 2
10 3 26

1 0 8 2

9 5 6 7
1 0 8 5

DONALD
GERALD
ROBERT

9 5 6 7
1 0 8 5
10 6 5 2
MONEY

0	1	2	3	4	5	6	7	8	9
	A	E	L	D		R	T		

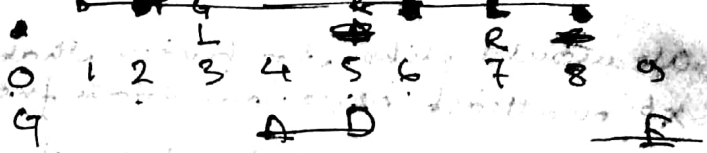
DONALD
GERALD
ROBERT

T	B	R		D	L				N
0	1	2	3	4	5	6	7	8	9

A G E

9 5 3

D, G, L, R



P

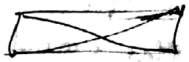
GERALD

+ DONALD

ROBERT

A

E



(10)

DONALD

+GERALD

ROBERT

E = 9

A = 4

6

22/10/17

Game Playing

Problem reduction Approach can be used to find playing strategy for certain kind of games.

There are two reasons that games appeared to be good domain to explore machine intelligence.

(1) Games provide structured task in which it is easy to measure SUCCESS or FAILURE

(2) Games do not require large amount of Knowledge

[This reason is not true for all the games but only true for simplest games.]

In games, players know completely what they have done and can do.

In AI, our interest is in only two players game where aim is to win the game on the result is draw

Problem reduction approach finds the winning strategy through the process of proving that the game can be won

Suppose we name the two players as PLUS and MINUS

Now let us consider the problem of finding winning Strategy for the PLUS player from the configuration represented by x^t

The superscript 't' stands for either '+' or '-'

x^+ represents a configuration which it is PLUS player turn to move next.

x^- represents a configuration which it is MINUS player turn to move next.

We would like to know that PLUS can win from x^t or at least PLUS can draw from x^t

Let us describe the problem of knowing that PLUS can win from configuration x^t by the expression $W(x^t)$

Suppose . it is PLUS's turn to move next from the configuration x^t and there are N legal moves.

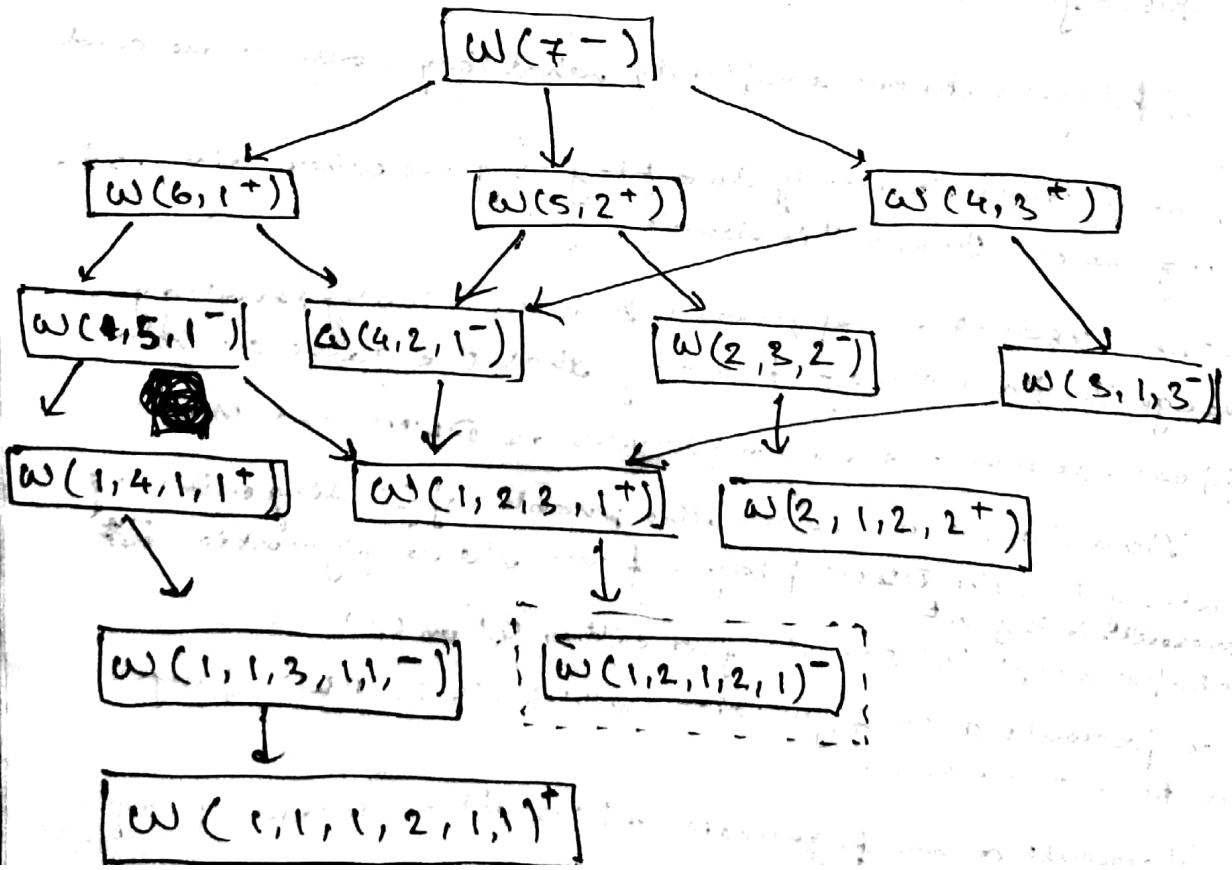
. $x^{t_1}, x^{t_2}, \ldots, x^{t_n}$

The problem reduction operator generates what is often called game tree

Illustrative Example:

Grundy's Game:

Two players in front of them a single pile of objects say a stack of pennies. The first player divide the original stack into two stacks that must be unequal. Each player alternatively does the same to some single stack until finally every stack has either just one or two penny The player who first can not play is the loser



Two things can be done to improve the effectiveness of the search based problem solving

① Improve the generate procedure, so that only good moves are generated

② Improve the test procedure so that the best path will be recognised and explored.

Minimax Procedure

It is a look ahead Procedure and depth limited depth first search.

Suppose situation analyser that converts overall judgement about a situation into a single overall quality number.

+ve number by convention favours one player
-ve number by convention favours other player

The degree of favour goes with the absolute value of the number

The procedure of determining the quality number is called static evaluator

At the end of limited exploration of moves one finds static evaluation score produced by the situation analyzer

The player who is hoping for +ve number is called maximizing player and his opponent is called Minimizing player.

Minimax Procedure:

1. Determine if the limit of search has been reached or if the level is a minizing level or if the level is a maximizing level

(1a) If the limit of search has been reached then compute the static value of the current position relative to the appropriate player & Report the result.

(1b) If the level is the minimizing level then apply minimax on the children of the current position & report minimum of the result

(1c) If the level is the maximizing level then apply minimax on the children of the current position & report maximum of the result