Artificial Intelligence (AI) :-

Turing Test (1950) :-

The computer is interrogated by a human is a teletype.

It passes, if the human can't tell, if there is a computer or human at the other end, the ability to solve problem (computational problem).

Solving problems (computational problem)-

- Search : Efficient Trial and error.
- SPACE / TIME complexity trade off is
- Use of domain knowledge (Heuristics)

RUSSEL/ NORWIG

Popular techniques-

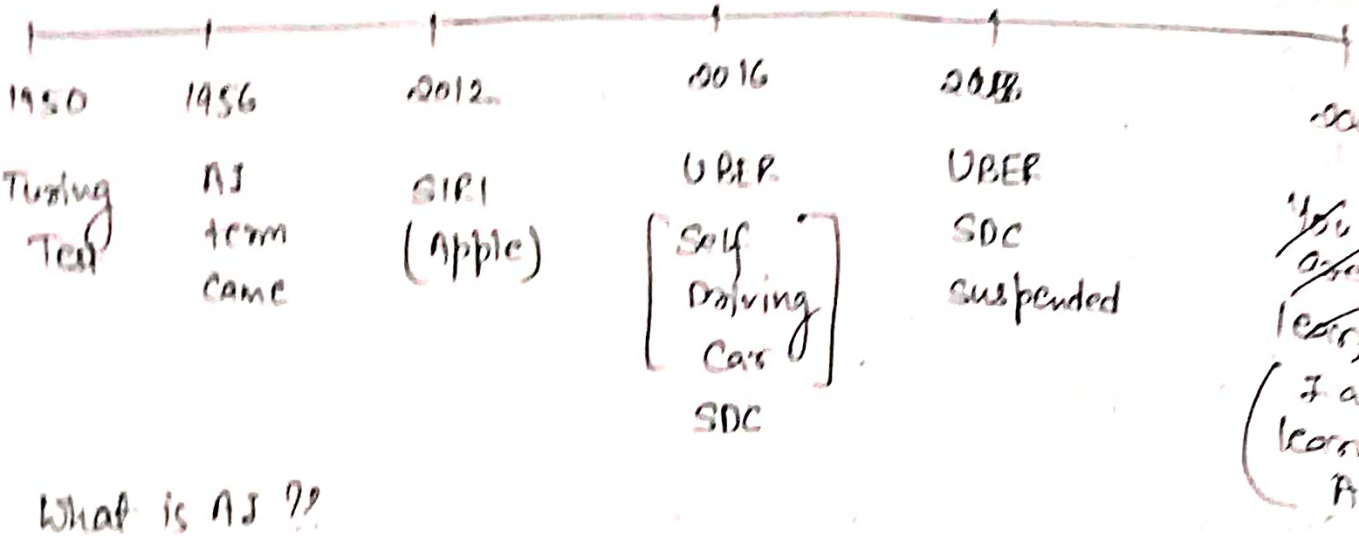
- Linear programming
- Integer programming
- Dynamic programming
- Heuristics programming
- Evolutionary Algorithm.

Insertion in linked List O(1)

AGENT- A computer program but it should be capable of doing more.

Responsible AI :-

History of AI:



What is AI ??

- Automated problem solving
- ML
- Logic and Deduction

Human Computer Interaction
- NLP
- Computer Vision
- Robotics

Fundamentals :-

① The notion of expressing computation as an Algorithm.

② Godel's incompleteness theorem (1931)

③ Church-Turing thesis (1936)
- Turing machine is capable of computing any computational problem.

④ Two notions of intractability
- NP completeness
- Reduction.

Logic and Deductions-

First order logic, temporal logic.

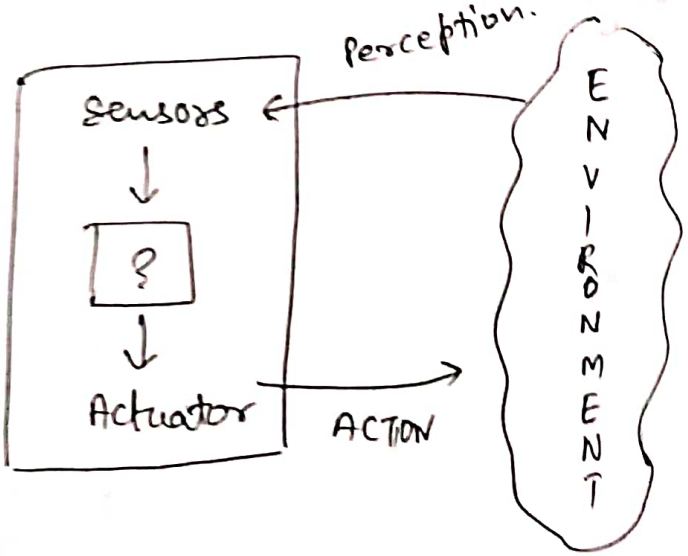
- Planning
- Reasoning under uncertainity
- Learning

An agent is just something that acts

All computational problem do something but AGENT does something more.—

- operate autonomously
- perceive their environment.
- persist over a prolonged time period.
- adapt to change, create and pursue goals

A rational agent is one that acts so as to achieve the best outcome, or when there is uncertainity the best expected outcome.



Rational agent is one that acts so as to achieve the best outcome, if there uncertainity the best expected outcome

Agent Type	Performance Measure	Environment	Actuator	Sensor
Taxi Driver	Safe, fast, comfort, legal, luxury	Road, Pedestrian	Steering, accelerator, brake, horn	Speedometer, GPS, Camera, Sonar
Medical diagnosis System	Health of the patient	Hostel Staff, Patients	Tests, Treatment, Referrals	Questionnaire
Interactive English Tutor	Students score	Set of students	Test, Exercise	Smart phone keyboard

Environment Types -

↓ Fully observable
Single Agent
[Cross Word Puzzle]

↓ Partially observable
Multi-agent
(Chess-game)

↓ Unobservable
[No sensor]

↓
Agent is sensors give it access to the complete state of the environment needed to chose an action then environment is fully observable.

Episodic vs sequential

Discrete / continuous

Known / Unknown

Uncertain
↓
Stochastic

Environment

Deterministic

Critic

WINE
IRIS — see my journal

Search Frameworks

Uninformed / Blind

Informed / Heuristic

Problem Reduction

Game tree search.

Basic Search Problem

Given $[S, s, O, G]$ where S is the [implicitly specified set] set of states $s \in S$ is the start state. ~~O is~~

O is the state transition operators

$G \subseteq S$ is the set of goals

To find a sequence of state transition leading from s to G

8- Puzzle Problem

1	2	3
4	5	6
7	8	

G

6	7	2
5	1	4
8	b	3

s

O: Movement of blank tiles in up, down, left, right direction:

n - Queen Problem:

G ⊙ : Placing 8 queen in non attacking position.

(I) : A sequence / state in any arrangement of 0-8 queens on board.

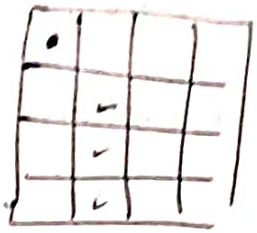
Operators add a queen to any sequence

8 Queen

First queen second

(8×56) (8×56)

Total = $(8 \times 56)^8$



→ 8^8

(why ??)

(II) Place a queen in the leftmost empty column

(III) Row-wise placement of queens

Missionery Cannible Problem :-

MMM
CCC

$S = \{ \langle x, y, z \rangle , x \in \{0,1,2,3\} , y \in \{0,1,2,3\}$

x: M left, y: C left

$z \in \{0,1\}$

$i = \langle 3,3,0 \rangle$

$G = \langle 0,0,1 \rangle$

$\langle 0,0,0 \rangle$ ← Not feasible

$O\{\langle o_1, o_2 \rangle$

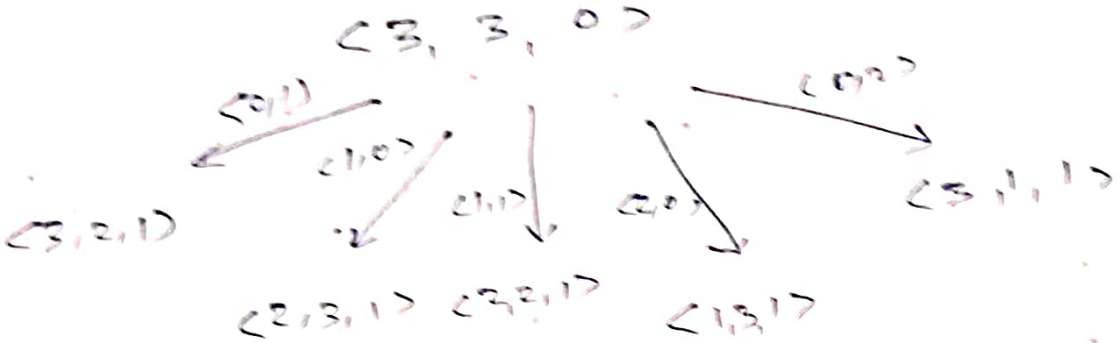
o_1 ↑ Number of missionary on boat

o_2 ↳ No of cannibal on boat

Minimum state transition required to reach goal state

<0,0> x
<1,0>
<0,1>
<1,1>
<0,2>
<2,0>
<2,2> x

operators



≒ Deterministic vs Stochastic Environment

↓ Next state of the environment can be determined by the current state of environment

↓ There are multiple, unpredictable outcomes

Environment is Fully observable + Deterministic

⇓

Agent need not to deal with any uncertainty.

(3,3,0)

(3,2,0)

(3,2,1)

(3,1,0)

(3,1,1)

(2,2,0)

(2,2,1)

(2,3,0)

(2,1,1)

(0,3,0)

(0,2,0)

(0,2,1)

(0,1,0)

(0,1,1)

(0,0,1)

(0,2,0)

2016 : AlphaGo :-

AlphaGo defeated Lee Sedol (4-1)

DeepMind

Geoffrey Hinton

[Artificial Neural Network]

(GodFather of AI)

Turing Test

A computer passes turing test if a human interrogator after posing some questions cannot tell whether the responses are given by a person or a computer.

To pass test computer would need :-

(I) NLP (Natural language Processing) enable to communicate succesfully in English.

(II) Knowledge representation

(III) Automated reasoning

(IV) Machine learning

Episodic and sequential

↓

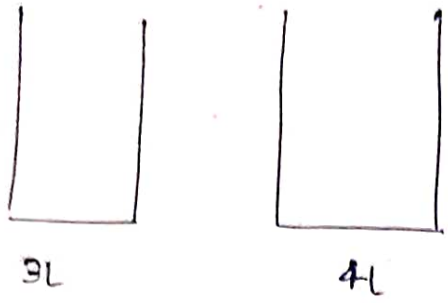
subsequent episode are independent of the previous episodes

↓

Agent is engaged with sequence of connected episodes.

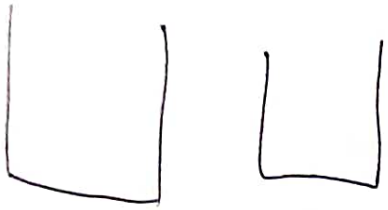
Jug / Jar Problems-

0, 1, 2, 3, 4, 5, 6, 7

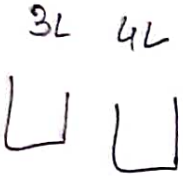


0 ✓
1 ✓
2 ✓
4 ✓
8 7 ✓

0
1 2
3
4 5, 6
7



0 1 2 3 4 5 6 7 8 9
✓ ✓ ✓ ✓ ✓



(8, 8, 0, 4)

<x, y>

Water in Jar 1 — Water in Jar 2

8 , 8 , 0 , 4

$\{(x, y) \mid 0 \le x \le 4, \; 0 \le y \le 5\}$

$x, y \in Z$

operators

Jar 1

$(0, y)$: making empty

$(x, 0)$: making Jar 2 empty

$(3, y)$: Fill up Jar 1

$(x, 4)$: Fill up Jar 2.

$(x+k, y-k)$ Transfer k litres of water from Jar 1 to Jar 2

$$k = \min(3-x, y)$$

$(x-k, y+k)$ Transfer k litre of water from Jar 1 to Jar 2

$$k = \min(x, 4-y)$$

G: $\{(x,y) \mid 0 \le x \le 3, 0 \le x \le 4, x, y \in Z\}$

$x+y = T$

TRAVELLING SALESMAN PROBLEM

- Must visit every city except source one and only once.
- start from source and come back to source
- Minimise cost

S, S, D, G, C

↓ P

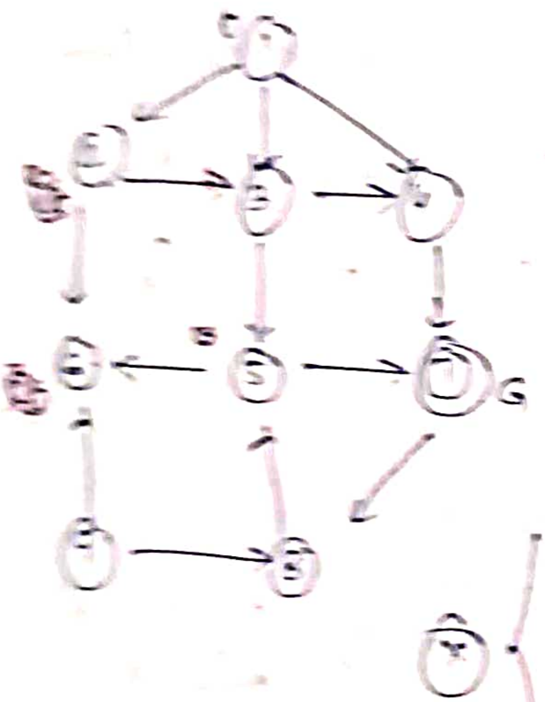
↓ Intermediate Paths

↓ Path

↓ time memory

Performing DFS

OPEN (stack)



1	Visit 1
4, 3, 2	Visit 2
4, 3, 6	Visit 6
4, 3	Visit 3
4, 7	

4, 3	Visit 3
4, 5	Visit 5
4, 7	Goal

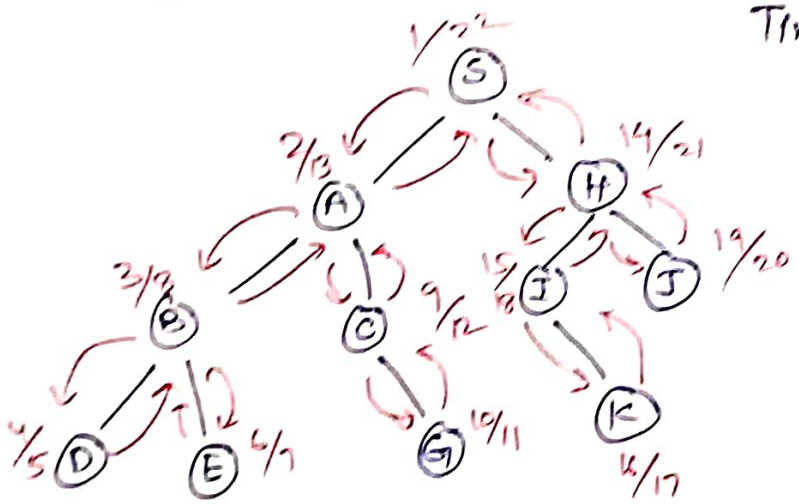
Depth - First Search :-

- Uses stack data structure
- Backtracking is used in dfs.
- Required less memory
- Take less time to reach goal than BFS algo.

Disadvantages:-

- There is a possibility that many states keep recurring and no guarantee of solution.
- DFS algo goes for deep down searching and may go to ∞ loop.

$O(|V| + |E|)$



Time complexity

$= O(b^m)$

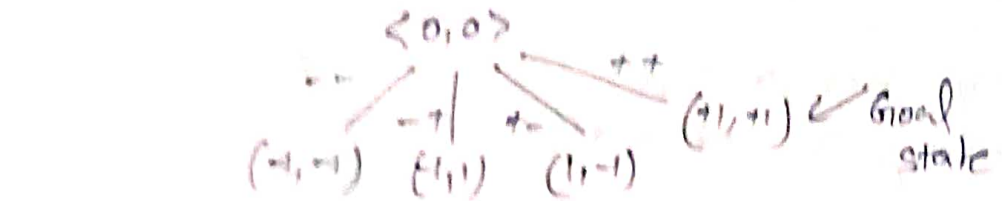
b : branching factor

m : maximum depth of any node

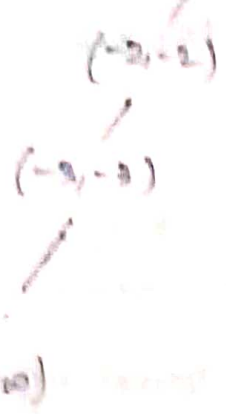
max depth of any state

Space complexity : $O(bm)$

DFS search is not a optimal search technique



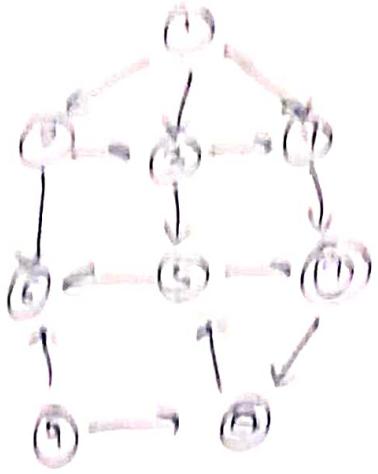
Here we can not reach Goal state



- Incomplete
- Non-optimal

Completeness:

BFS



Time Complexity:

$O(b^d)$

d = depth of goal state
b = branching factor.

Space complexity

$O(b^d)$

Queue

1

2, 3, 4

3, 4, 6

3, 4, 5, 6

6, 5, 7

Goal state

	Time	space		
Bfs	b^d	b^d	Yes	Yes
Dfs	b^m	$b \cdot m$	No	No

How can we improve dfs ??

dfs can be improved if we define certain threshold

depth limited dfs. ⇒ ddfs.

Depth limited (Dfs) DDFS	b^l	$b \cdot l$ l: depth limit	✗	✗
Iterative Deepening DFS (IDFs)	$O(b^d)$	~~$O(b^d)$~~ $O(b \cdot d)$	✓	✓
Bidirectional BFS If possible	$O(b^{d/2})$	$O(b^{d/2})$	✓	✓

IDDFS

↓
combination of DFS and BFS algorithms.

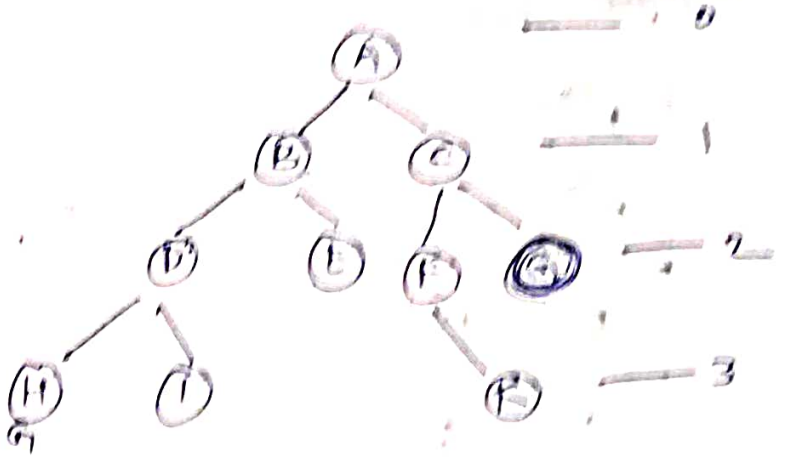
Finds out best depth limit and does it by gradually increasing the limit until goal is reached.

Used when search space is large and depth of goal node is ~~ten~~ not known.

disadvantages-

IDDFS- repeats all the works of previous phase.

eg



1st iteration ⇒ A

2nd " ⇒ A, B, C

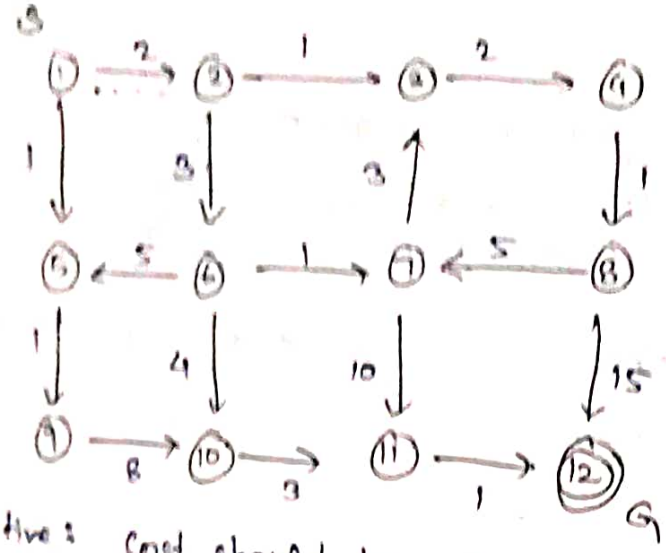
3rd " ⇒ A, B, D, E, C, F, G

4th Iter. ⇒ A, B, D, H, I, E, C, F, K, G

complete and optimal

Bidirectional Bfs

- one should know the goal in advance
- Bidirection Bfs.



Objective : Cost should be minimum

OPEN

1 (0)

2 (2) 5 (1)

2 (2) 9 (2)

9 (2) 6 (5) 3 (3)

6 (5) 3 (3) 10 (10)

6 (5) 10 (10) 4 (5)

4 (5) ~~10 (10)~~ 7 (6) 10 (9)

7 (6) 10 (9) 8 (6)

10 (9) 8 (6) 11 (8)

10 (9) ~~11 (8)~~ 12 (11)

~~11 (8)~~ 12 (11) 11 (12)

12 (11) 11 (12) 12 (13)

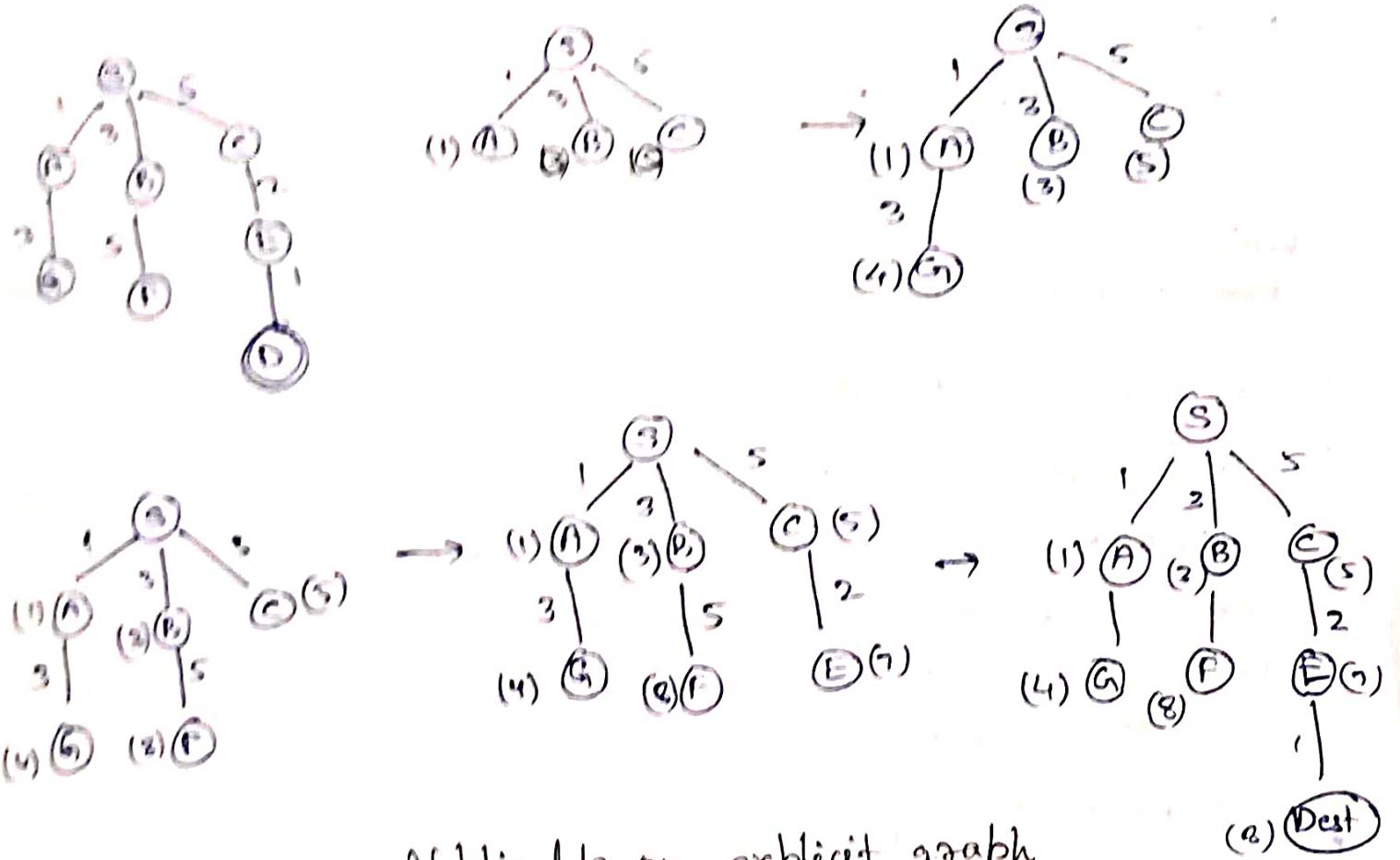
CLOSE

1 (0) 5 (1) 2 (2) 9 (2) 3 (3) 6 (5) 7 (6)

8 (6) 10 (9) 11 (12) 12 (13)

Uniform cost search

- variant of Dijkstra's algorithm
- useful for infinite graphs and that graph which are also large to represent in memory.



Dijkstra Algo - Applicable on explicit graph

Uniform cost search - Applicable on both explicit and implicit graph.

Search and optimisation

Given - (s, S, O, G, C)

To find :-

- A minimum cost sequence of transition to a goal state.
- A sequence of transition to the minimum cost goal. → [Think of example]

Basic ~~First~~ search Algo :-

(i) Initialise set OPEN = {s}

(ii) Fail if open = {} Terminate with failure

(iii) SELECT : select a state n from OPEN

(iv) Terminate : If $n \in G$, terminate with success

(v) Expand : Generate the successor of n using O and insert in OPEN.
(O ↑ operator)

(vi) LOOP: Go to step 2.

Best first algo in AI finds the shortest path from a ~~starting~~ node to goal node in graph.
Uses two lists to monitor the traversal while searching

(i) open list : Monitors the immediate nodes available to traverse at the moment.

(ii) CLOSED list : Monitor nodes that are being transferred already.

Algo works by expanding graph nodes in order of increasing the distance from ~~goal node~~ starting node until goal node is reached.

saving Explicit space:-

(i) Initialise CLOSED = {,} and OPEN = {s}

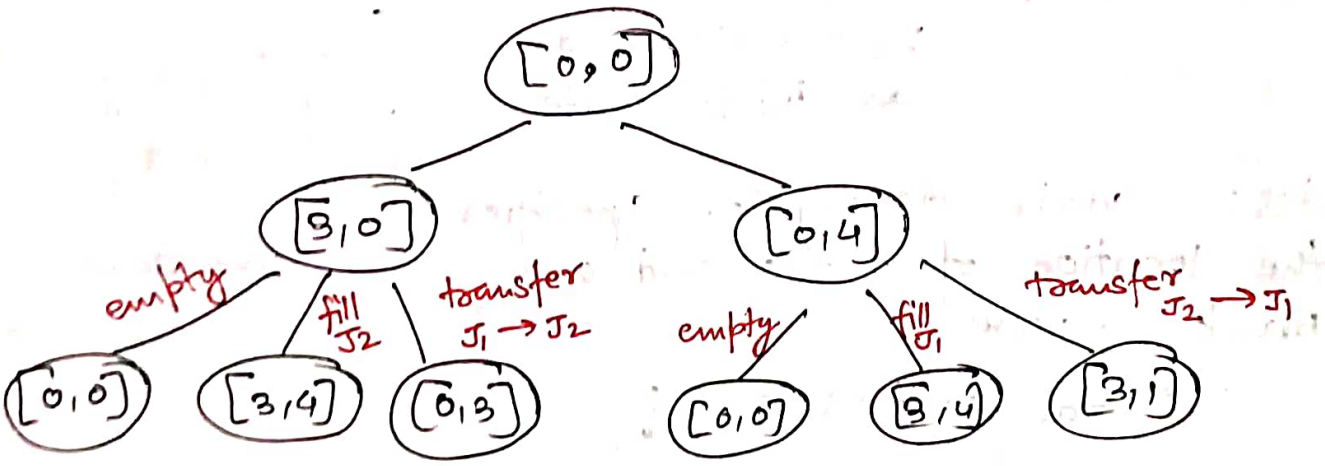
(ii) ⊗

(iii) A select a state n from OPEN and SAVE n is CLOSED

(iv) ⊗

(v) For each successor, m, insert m is OPEN only if

$m \notin$ [OPEN ∪ CLOSED]



Bfs → Queue
Dfs → stack

Problem solving Agents

Well defined problems and solutions

I - Initial state

S - state space

O - Successor function

G - Goal state

C - Path cost

7	2	4
5		6
8	3	1

(initial state)

eg-

8 Puzzle Problem :-

A nine x nine grid with one blank space tile. Blank space tile can move or can slide into the space. Goal is to reach the goal state

Initial state :- Any state can be designated as initial state

States :- State description specifies the location of 8 tile and one blank square.

Total states = 9!

1	2	3
4	5	6
7	8	

Goal state

Successor function : Legal sliding action like (move up, down, left, right.

Goal state :-

Final state that need to be is required

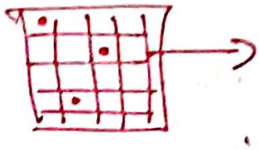
8 Queen Problem :

SR- Any arrangement of 0 to 8 queens on the board.

S- No queens on the board

O- Add a queen to any empty square

G- 8 queens on the boards in non-attacking positions.

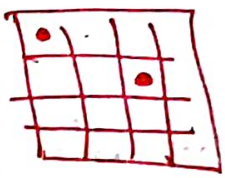


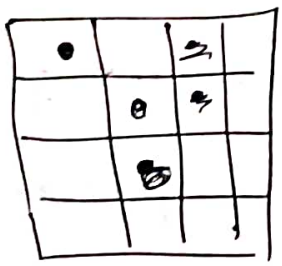
In this formulation there are $64 C_8$ possibilities ie 3×10^{14}

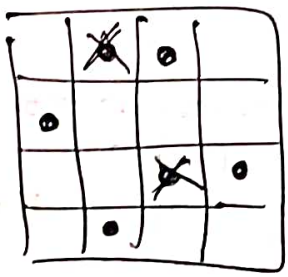
Better formulation will be-

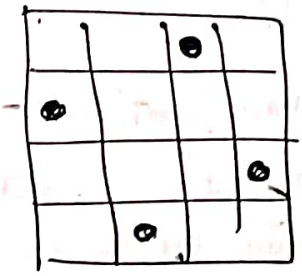
State : Arrangement of queens, one per column in the leftmost n columns, with no queen attacking another. → (now 2,057 possibilities)

Successor function : Add a queen in the left most empty column such that it is not attacked by other queen.







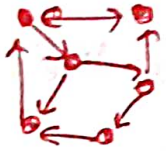


Goal state

Traveling Salesman Problem (NP-hard)

- Each city must be visited exactly once
- Aim is to find shortest tour.

- Visit every city except source once and only once.
- Start from source and return back to the source.
- Minimise cost [cost can be time and money]



S: Source

G: find a path with min cost so that every city is travelled exactly once and ~~salesman~~ returns back to ~~its~~ ~~sto~~ source.

S: Multiple Paths available.

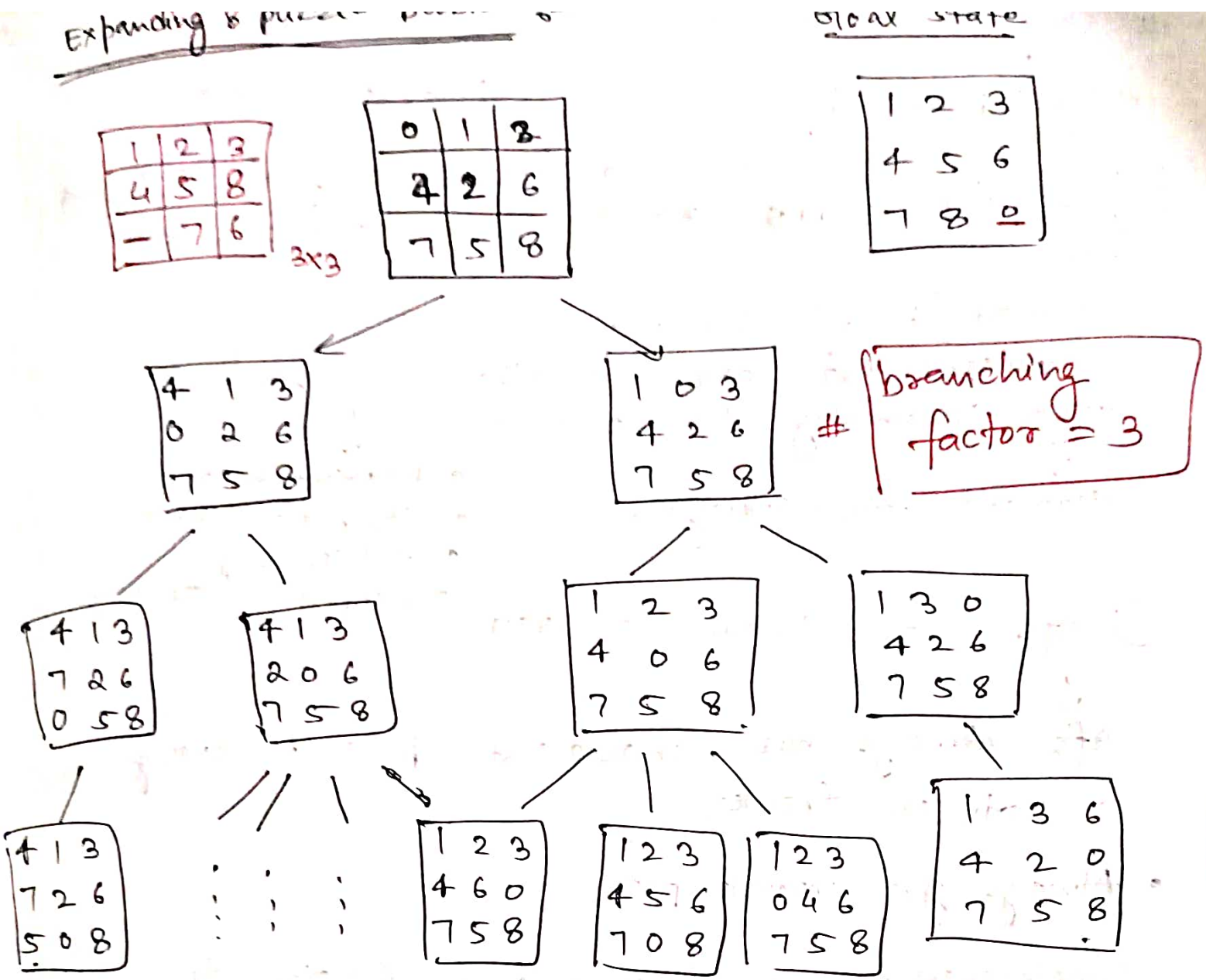
O: The different transition to go from one city to another

Real life problems -

1. Robot Navigation - Generalisation of root finding problem.
2. VLSI layout - Positioning of millions of components and connection on chip to minimise area.
3. Automatic assembly sequencing

Aim is to find an order in which to assemble the parts so as to ~~maximise~~ minimise labour and time.

Expanding a puzzle ... goal state



Searching for solution

Set of all nodes (leaf) available for expansion at any given point is called frontier.

⇓

(Open list)

Closed list

↑ explored set of nodes.

Branching factor - no. of successor node (maximum)

Uninformed Search strategies :-

Blind search

Generate successors

distinguish a goal state and non-goal state

Bfs | Dfs | DDFS | IDDFS | Bidirectional

UCS

Informed / Heuristic Search :

strategies that known whether one goal is more promising than other one.

⇓

Heuristic search strategies.

- Best first search
- Heuristic Search
- Hill climbing
- Steepest hill climbing

① Bfs – Queue is used

↓

Bfs always has shallowest path to every node on frontier.

- Always give shortest path.

Completeness – If there exist a solution to problem, and algorithm is able to find to find the solution is called complete algo.

Bfs – complete

Time complexity : $O(b^d)$

b : branching factor

d : depth of search tree

TC : $b + b^2 + b^3 + \cdots + b^d = O(b^d)$

space complexity : $O(b^d)$

[d : depth of shallowest solution]

10^4 nodes ⇒ 11 ms

10^6 nodes ⇒ 1.1 sec. 1 GB

Exponential time complexity – takes a lot of time to find solution for large problems.

Consumes more space.

Exponential space complexity – very bad

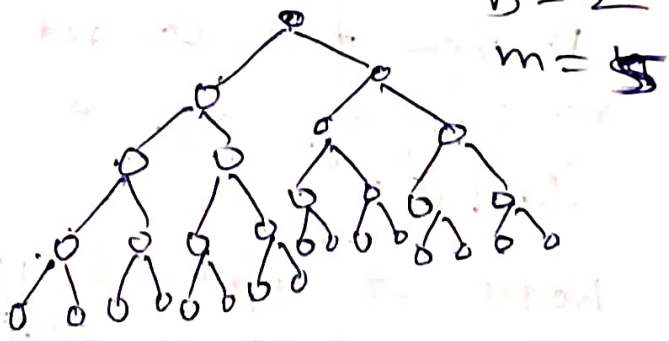
(11) Depth First Search

- Uses Stack
- Incomplete (not guaranteed to find a solution)
- Not optimal
- Linear space complexity : $O(b \cdot m)$
- Exponential time complexity : $O(b^m)$

m : depth

Expands deepest node in current fringe

for a state space with branching factor b and maximum depth m, storage required by dfs is $(bm+1)$



Backtracking search

only one successor is generated at a time rather than all successors, each partially expanded node remembers which successor to generate next.

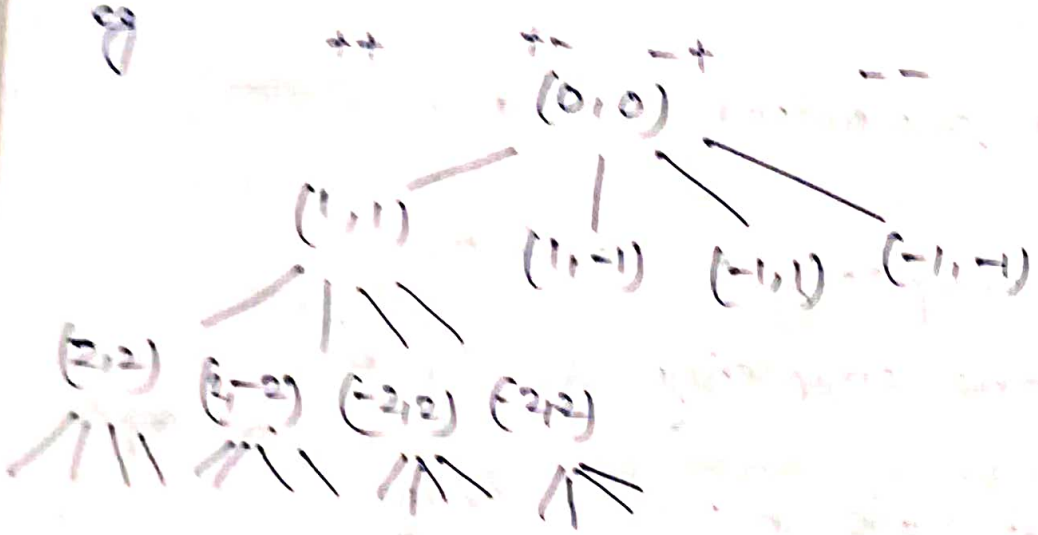
Memory needed ~~$O(bm)$~~

$O(m)$ rather than $O(bm)$

Drawback of dfs :-

- Can make a wrong choice
- Gets stuck on infinite path

eg



Depth Limited Search

Problem of unbounded trees can be alleviated by supplying dfs a predetermined depth limit ℓ.

Nodes at depth ℓ will be treated as if they have no successor.

It is also incomplete if

eg- If we choose $l < d$ (depth of solution)
no solution can ~~not~~ not be found.

Time complexity : $O(b^l)$

Space " : $O(b \cdot l)$

* Iterative Deeping depth-first search (IDDFS) (IDPS)

It search by gradually increasing the depth limit from 0, 1, 2, 3 to .... so on

[Combined benefits of dfs and bfs]

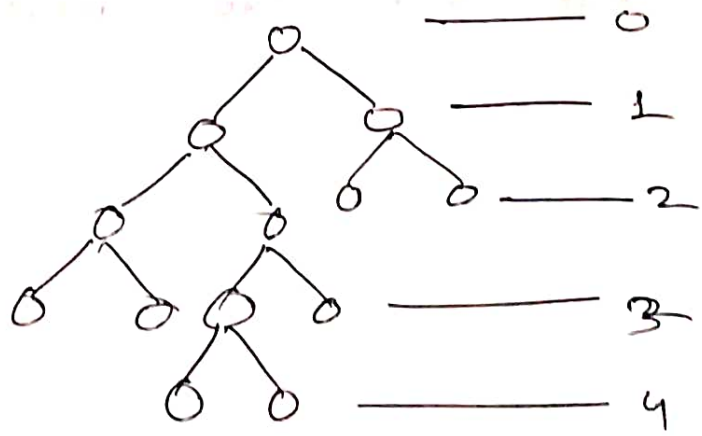
It memory requirements are modest $O(b \cdot d)$

It is complete.

Time complexity : $O(b^d)$

It is optimal.

Space complexity : $O(b \cdot d)$



Bidirectional search :

Idea behind bidirectional search is to run two simultaneous searches - one forward from initial state and other backward from goal, stops when two searches meet in the middle

Motivation is that $b^{d/2} + b^{d/2}$ and much less than b^d

Time complexity : $O(b^{d/2})$

Space complexity : $O(b^{d/2})$ ← space requirement is more

weakness ↓

Complete + Optimal (if both searches are bfs)

Note :

- If there is one goal state, bidirectional search is fast works well.

Searching from start node is similar to searching from goal node.

But there are more than on goal state

For eg

All the states ~~that~~ satisfying the "checkmate" goal test in chess. A backward search would need to construct a compact description of all states that lead to checkmate by one move 'm'.

Those descriptions we have test against the states generated by forward search. There is no efficient way to do this.

Algo	Time	Space	completeness	optimality
Bfs	$O(b^d)$	$O(b^d)$	Yes ✓	Yes ✓
Depth first search	$O(b^m)$	$O(b \cdot m)$	No ✗	No ✗
Depth limited	$O(b^l)$	$O(b \cdot l)$	No ✗	No ✗
Iterative deepening	$O(b^d)$	$O(b \cdot d)$	Yes ✓ (if b is finite)	Yes ✓
Bidirectional	$O(b^{d/2})$	$O(b^{d/2})$	Yes ✓	Yes ✓

⇒ Space complexity of dfs is less than of bfs.

$O(b \cdot m)$ ⇓ $O(b^d)$

⇒ Bfs ⇒ (optimal + Complete)

Dfs ⇒ Not optimal + Not complete

Uniform Cost Search :-

Bfs always gives shortest path if all steps are equal.

Bfs always expands for the shallowest unexpanded node.

Note :-

Uniform cost search expands the nodes with lowest path cost.

Note: if all steps costs are equal it is same as bfs.

does not care about number of steps in path but only cares about total cost.

Therefore it may get stuck in infinite loop if there is a node with zero cost.

Complete + Optimal

↓

If all path costs $\geq \epsilon$, i.e. greater than ϵ

Uniform cost search is guided by path costs rather than b and so it complexity is complex to determine.

Let C be cost for optimal solution and every action costs at least ϵ. then

Time complexity $O\left(b^{C/\epsilon}\right)$ which is much greater than $O\left(b^{d}\right)$

so it is used only to explore large trees with small steps.

Dijkstra

No negative edge is allowed

UCS

Negative edge is allowed

Instead of inserting all nodes into the priority queue only source node is added and taken one by one insertion when needed

useful for infinite graphs

UCS :-

assumes all operators have a cost.

1. Initialise : set OPEN = {s}, CLOSED = { }, ~~set~~ c{s} = 0
2. Fail : If open = { }, terminate with failure
3. SELECT : select the min cost state (n) from open and save n in closed.
4. Terminate : If n ∈ G, terminate ~~from~~ with success.
5. Expand : Generate the successors of n using 0. for each success m.

If m ∉ [OPEN ∪ CLOSED] (OPEN ∪ CLOSED)

set c(m) = c(n) + c(n, m)

c(m) = c(n) + c(n, m)

If m ∈ (OPEN ∪ CLOSE)

set c(m) = min { c(m), c(n) + c(n, m) }

(#) if c[m] has decreased and m ∈ CLOSED move it to OPEN.

u.cost = 0 cost (u, u)

v.cost = 5 cost (u, v)

~~Dijkstra~~ UCS Algorithm :-

Initialisation : OPEN (S) CLOSED = { } S.cost = 0

Iteration : Remove the lowest cost state n from OPEN CLOSED.append(n)

{ Priority Queue / Min heap }

Terminate : If $n \in G$, then terminate with success.
with success

Selection : Apply all $o \in O$ on 'n' to obtain weight of n

$m \notin (OPEN \cup CLOSED)$

$m.cost = \underbrace{n.cost + cost(n, m)}_{B}$

if $m \in OPEN$ and $m.cost > B$

Revise cost of m as-

$m.cost = B$

* [if $m \in CLOSE$ and $m.cost > B$

Revise cost of of m as -

$m.cost = B$

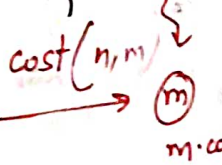
This terminates condition ~~is~~ is used only when all operator cost are +ve

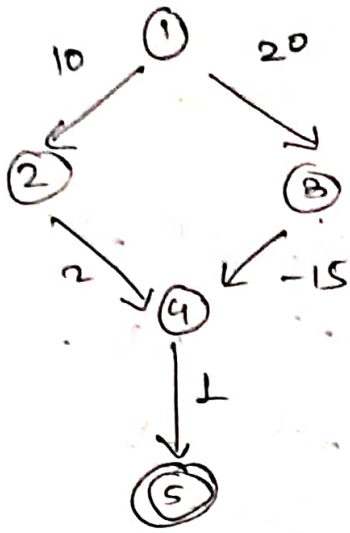
Move m from CLOSED to OPEN

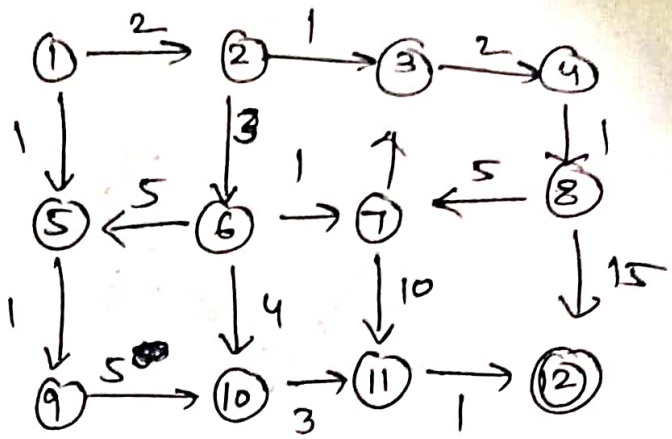
Termination with failure :- If OPEN = [ ] terminate with failure

↓ used when there is -ve edge

Termination condition for graph having -ve edge [Exhaustive Search]







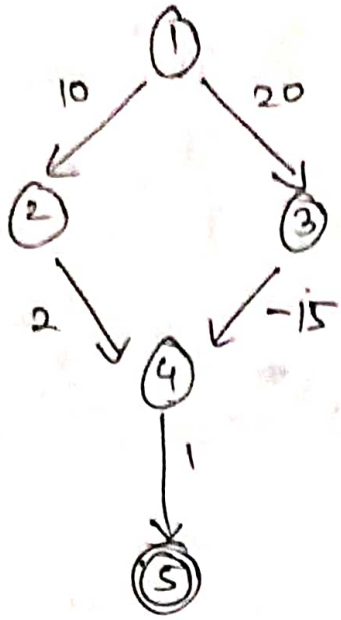
CLOSED

$1^{(0)}$ $5^{(1)}$ $2^{(2)}$ $9^{(2)}$ $3^{(3)}$
$6^{(5)}$ $4^{(5)}$ $7^{(6)}$ $8^{(6)}$
$10^{(7)}$ $11^{(10)}$ $12^{(11)}$

OPEN

$1^{(0)}$
$2^{(2)}$ $5^{(1)}$
$2^{(2)}$ $9^{(2)}$
$9^{(2)}$ $3^{(3)}$ $6^{(5)}$
$3^{(3)}$ $6^{(5)}$ $10^{(7)}$
$6^{(5)}$ $10^{(7)}$ $4^{(5)}$
~~$5^{(5)}$~~ $10^{(7)}$ ~~$4^{(5)}$~~ ~~$8^{(6)}$~~ $7^{(6)}$
$10^{(7)}$ $7^{(6)}$ $8^{(6)}$
$10^{(7)}$ $8^{(6)}$ $11^{(6)}$
10^{7} $11^{(16)}$ $12^{(21)}$ G
12^{21} 11^{10}
12^{11}

Goal state reach but we are not terminating here → because Goal is still in open



OPEN

~~1(0)~~, 2(10), 3(20)
3(20) 4(12) .
3(20) 5(13)
3(20)
4(5)
5(6)

CLOSED

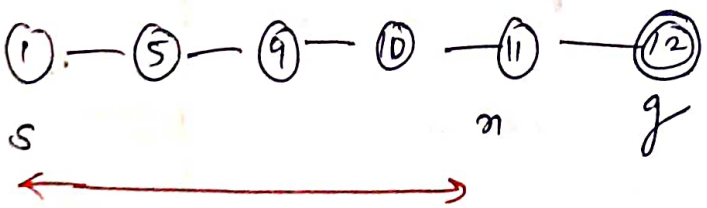
1(0) 2(10) ~~4(12)~~ * ~~5(13)~~
3(20) 4(5) 5(6)

Theorem:

(A) UCS applied on a ~~problem~~ problem having +ve operator cost will return optimal solution.

(B) Proposition:

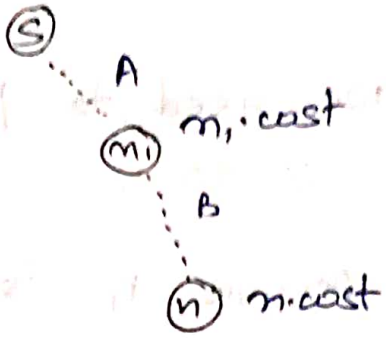
At least one node in the path from s to the nes [having optimal cost should be OPEN before n is being removed from [OPEN]



At least one node in the path from s to nes [having optimal cost] should be in OPEN before n is being removed from OPEN.

(c) There does not exist any other path from s to n having lesser cost than n·cost.

(1) At least one node belonging to the path having optimal cost from s to n should be in OPEN always, before retriving n.



In path-1 :

$A + B = C^*(n)$

Assume : $\alpha < A$

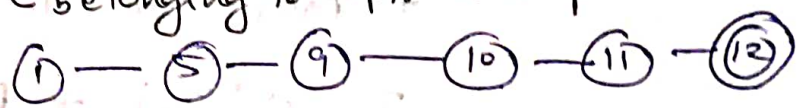
$\alpha + B < C^*(n)$

⟶ ⟵

optimal cost from S to n $= C^*(n) = A + B$

$g \in G$: closest optimal goal from s

(D) $C^*(n) \leq C^*(g)$, $\forall\, n \in S$ will be expanded before g.
↑ belonging to optimum path.



↑ Optimal cost of n from s

↑ optimal cost of goal state from s.

$C^*(10) = 7$

$C^*(9) = 2$

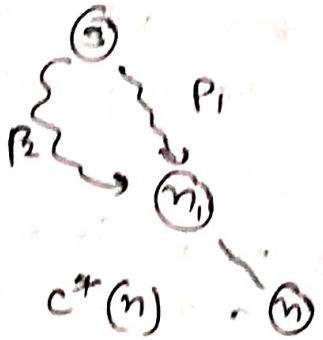
$C^*(12) = 11$

$C^*(10) \leq C^*(g)$
$C^*(9) \leq C^*(g)$
$C^*(12) \leq C^*(g)$

True always.

UCS ⇒ $C^*(n) \leq C^*(g)$

(2) if s to n ⟶ optimal
then s to n_1 ⟶ optimal



P is the optimal path from s to n through n_1 with cost $c^*(n)$.

P_1 is the subpath of P from s to n_1

We need to prove : P_1 is optimal subpath from s to n_1

Assume P_1 is not optimal path from s to n_1 with cost $c(n_1)$

Let's $\exists$ P_2 which is optimal path from s to n_1 ⇒ $c^*(n_1) < c(n_1)$

Path from s to n = optimal path from s to n_1
+ Path from n_1 to n
= $c^*(n_1) + p(n_1, n) < c^*(n)$

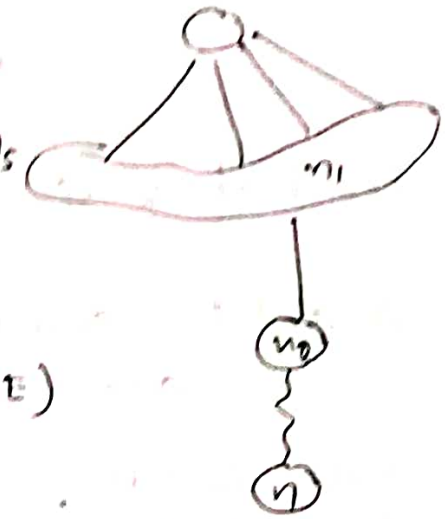
Base Case :

N_p : all the nodes reachable directly from s are in OPEN in the first step of algo.

Let p be the optimal path from s to a node $n \in S$

N_S P consist of at least one node in

Let assume, s to n consist of n_1 and n_2, n_1 is directly reachable from s is in OPEN



Until n_1 is not being removed,

(P.) is true.

when n_1 is expanded, n_2 will be visited.

(i) n_2 is not in (OPEN ∪ CLOSE)
⇒ n_2 is brought to OPEN

(ii) $n_2 \in$ (OPEN ∪ CLOSED) ⇒ cost of n_2 will be visited through n_1 because of C.

..... using induction

(P.) can be proved

Note:

If Path costs are unit / equal

UCS
⇓
Bfs.

UCS with equal path costs
⇓
Bfs

UCS :-

If all operator costs are +ve then the algo finds the min cost sequence of transition to a goal.

Branch and Bound :-

(i) INITIALISE set OPEN = [s] CLOSED = []
set c[s] = 0 $c^* = \infty$
↑ Initial bound

(ii) Terminate : if OPEN = [] then return c^*

(iii) SELECT : select a state n from OPEN and save in CLOSED.

(iv) NEW BOUND ASSIGNMENT : If $n \in G$ and $c(n) < c^*$ then (lesser cost goal found)
set $c^* = c(n)$ and Goto step2.

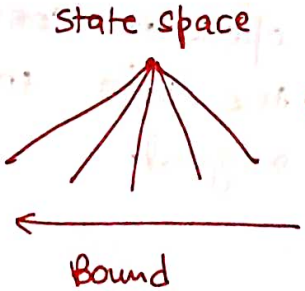
(v) EXPAND If $c(n) < c^*$ generate the successor of n for each successor m
cost of n is less than bound c^*

if $m \notin$ [OPEN ∪ CLOSED]
set $c(m) = c(n) + c(n,m)$
and insert m in OPEN
[find cost of successor if it is not present in OPEN ∪ CLOSED]

if $m \in$ [OPEN ∪ CLOSED]
Bound B
$B = c(n) + c(n,m)$
set $c(m) = \min(c(m), B)$
if c(m) has decreased
and m ∈ CLOSED move to OPEN

Found a better path so m ∈ CLOSE move it to OPEN

(vi) Goto step 2.



If one solution is found in one branch, other branches are explored in search of better solution

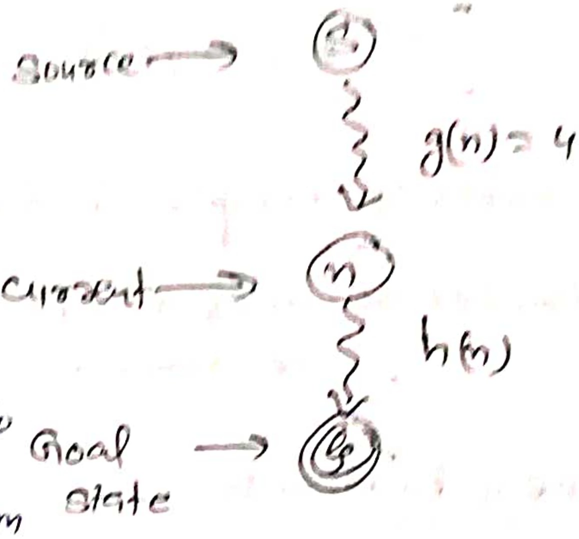
Heuristic :-

* method of learning and solving problems that allow people to discover things themselves and learn from their own experiences

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

$g(n)$ ↑ Actual cost of n from s

$h(n)$ ↑ Estimated cost of n to ~~from~~ reach G from n



↳ Informed search.

Patna-Bihta-Ara Example

$h^*(n)$ = optimal cost to reach G from n

$h(n) \leq h^*(n)$ underestimate

$h(n) > h^*(n)$ overestimate

eg- Heuristic values are calculated by some functions like Euclidean distance, Manhattan Distance

1	3	2
6	5	4
	8	7

n

→

1	2	3
4	5	6
7	8	

G

Blind search take 3^{20} case to be explored to reach solution

3.5 Billion

348 crore possibilities

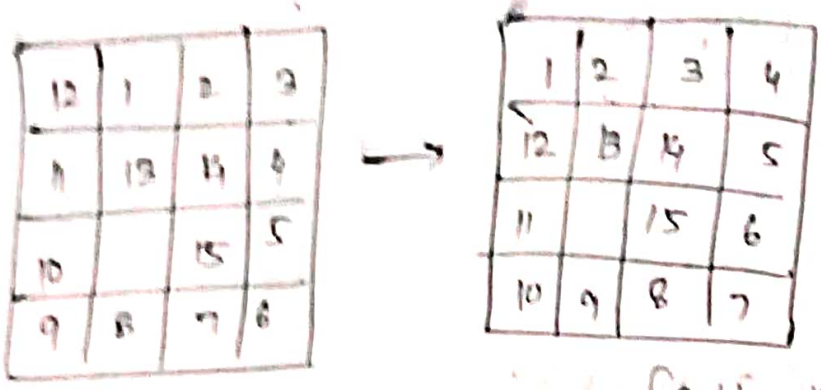
Number of tiles misplaced = 1+1+1+1+1+1 = 6

Manhattan distance = 0+1+1+2+0+2+2 = 8

⇓

~~the~~ moves to get the tiles at correct position.

(11)



$h(n) = 12$

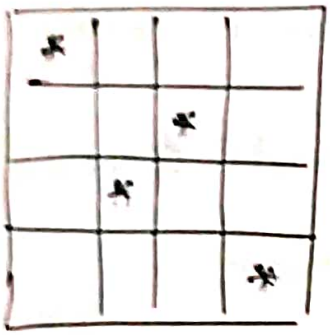
Number of misplaced tiles = 12

Number of tiles need to be moved = 12

$h(n) = 12$

{ Explore the path in which the number of misplaced tiles is less }

N Queen Problem :-



$h(n) = 2$

Number of queens in attacking position.

{ Explore that path in which no. of queen at attacking pos is less }

Heuristic for TSP :-

$h(n)$ = cost of MST (underestimate)

= $2 \times$ cost of MST : $k \geq 2$ (overestimate)

When we want solution in less time.

$g(n)$: function of state and operator

$h(n)$: function of state only (will not change)

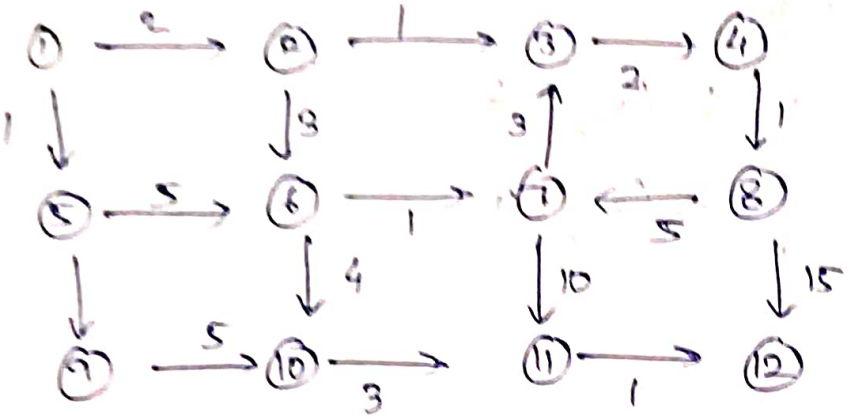
Heuristic ⟶ gives good solution but does not guarantee optimality

Informedness :-

h_1 and h_2 are 2 underestimated heuristics.

h_1 is more informed than h_2 if $\forall n \in S$

$$h_1(n) \geq h_2(n)$$



$h(1) = 11$

$h(2) = 11$

$h(3) = 18$

$h(4) = 16$

$h(5) = 10$

$h(9) = 9$

$h(10) = 4$

$h(11) = 1$

$h(12) = 0$

optimal path is heuristic

If $\forall n$ $h(n) = h^*(n)$ then only the nodes in the optimal path will be expanded

↑ Retrieved from OPEN

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

$$f(1) = g(1) + h(1) = 0 + 11 = 11$$

$$f(2) = g(2) + h(2) = 11 + 2 = 13$$

$$f(3) = g(3) + h(3) = 3 + 18 = 21$$

$f(4)$

$f(5) =$ $= 11$

$f(n) = g(n) + h(n) \quad = 2 + 9 = 11$

When the heuristic is underestimate ie,

$$\forall n \quad h(n) \leq h^*(n)$$

$h^* \Rightarrow$ Algo

$h_1(n_1) = 5$ | $h_1(n_1) = 6$

$h_2(n_1) = 7$ | $h_2(n_2) = 4$

Given k underestimated heuristic, derive the best heuristic $h_1, h_2, \ldots h_k \Rightarrow$ k underestimate heuristic.

Best: $\forall n \quad \max(h_1, h_2, \ldots h_k)$

Note: If $h(n) = 0$ then the heuristic will boil down to UCS.

A* Algorithm :-

① Initialize: OPEN = [s] ; CLOSED = [ ] s.cost = h(s)

Iteration. Remove the lowest ~~cost~~ $f(n)$ state n based on

from open.

Selection : $\{m.cost = n.cost + cost(n, m)\}$ ✗

$$\begin{cases} f(m) = g(m) + h(m) \\ g(m) = g(n) + cost(n, m) \end{cases}$$

$m.cost > n.cost + cost(n, m)$ ✗

$$g(m) > g(n) + cost(n, m)$$

$h(n) = 0 \Rightarrow$ UCS $\Rightarrow$ MANY NODES WILL BE EXPANDED.

$h(n) = h^*(n) \Rightarrow$ min. nodes will be expanded.

Inference of these two statements

Heuristic between these two

Blindly Searching

We know which path will lead to optimality.

A* uses heuristic search with underestimated heuristic. It always gives optimal solution.

Ques-
Which nodes are expanded in UCS, but not in A*??

Sol:- In A* only nodes in optimal path are expanded.

In UCS nodes other than optimal path nodes may be expanded.

In A*
$\forall n,$ $f(n) \leq [g^*(g) = f^*(g)]$
$g(n) + h(n)$

In UCS $\forall n$

$g(n) \leq g^*(g)$
$c(n) \leq c^*(n)$

$g(n) < g^*(n) < f(n)$
$=$
$g(n) + h(n)$

In UCS

$f(n) < f(G)$ ——— (1)

these many nodes are expanded

In A*

$g(n) + h(n) \leq f(G)$

↖ These nodes are expanded.

Nodes that are not expanded in A*

$g(n) + h(n) > f(G)$ ——— (11)

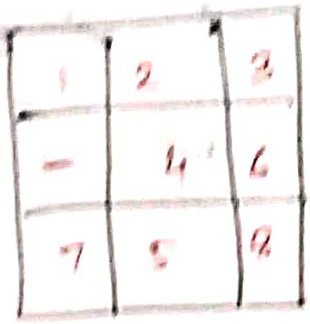
Taking intersection of (1) and (11)

$g(n) + h(n) > f(G) \cap f(n) < f(G)$

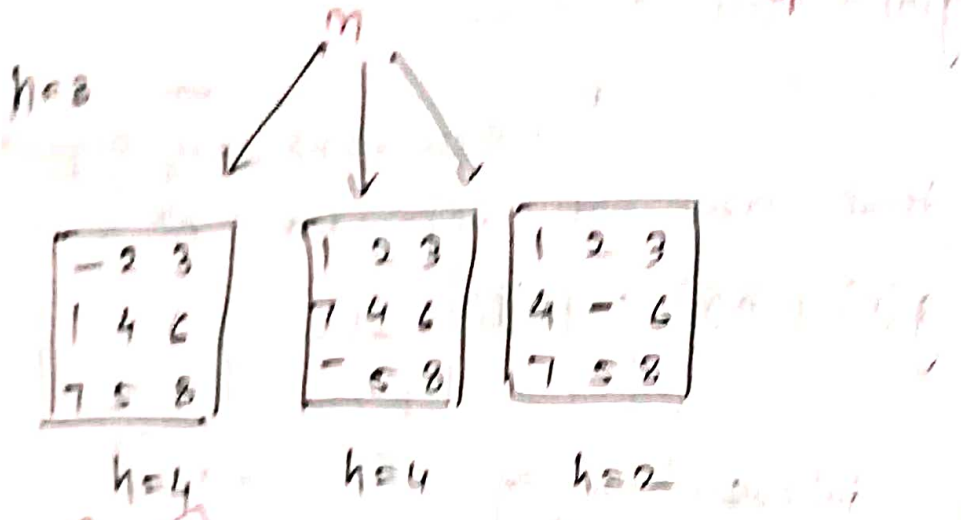
These nodes are expanded in UCS but not in A*

8 Puzzle Problem with heuristic

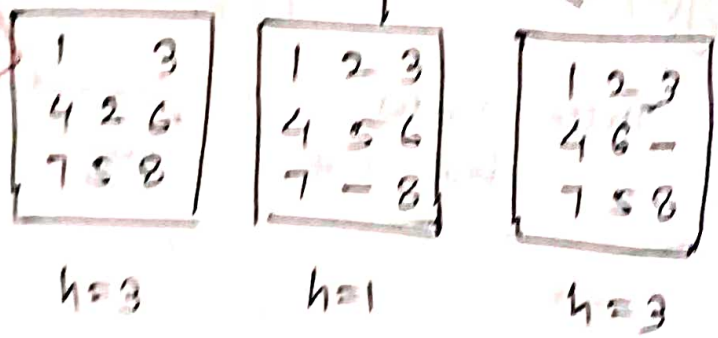
Informed search

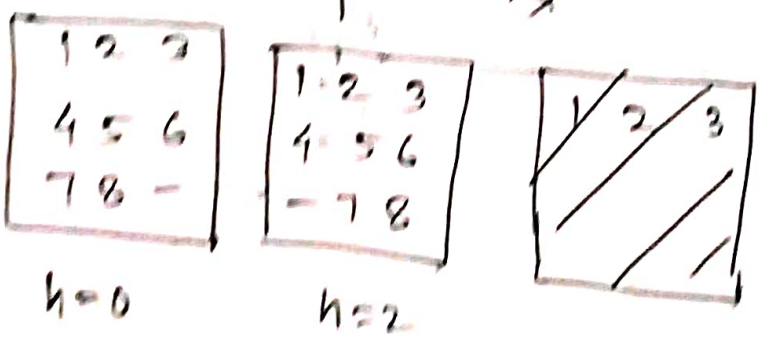


G



We can find the solution in polynomial using heuristic but it might possible that solution is not optimal





[solution]

A* Algorithm

(i) Initialise : Set OPEN = {s} CLOSED = {}
$g(s) = 0 \quad f(s) = h(s)$

(ii) Fail : if OPEN = [] (empty) terminate with failure.

(iii) Select : select the min cost state n from OPEN and save n is CLOSED. $f(n) = g(n) + h(n)$

(iv) Terminate : If $n \in G$, terminate with success and return $f(n)$.

(v) Expand : for each successor m of n

if $m \notin$ [OPEN ∪ CLOSED]

Update the cost of reaching the state m → set $g(m) = g(n) + c(n, m)$

set $f(m) = g(m) + h(m)$ → update $f(m)$

Insert m in OPEN. ← estimated cost

if $m \in$ [OPEN ∪ CLOSED]

set $g(m) = \min\{ g(m), g(n) + c(n, m) \}$ ← prev cost, ← new cost

set $f(m) = g(m) + h(m)$

if m is already present update the cost g(m) if new cost is less than that.

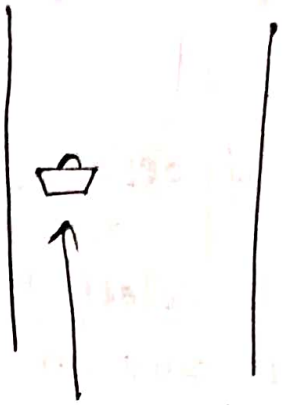
if $f(m)$ has decreased and $m \in$ CLOSED move m to OPEN

(vi) Loop : Goto step 2.

Cannibals and Missionaries

If number of cannibals is greater than missionaries then cannibals kill missionaries

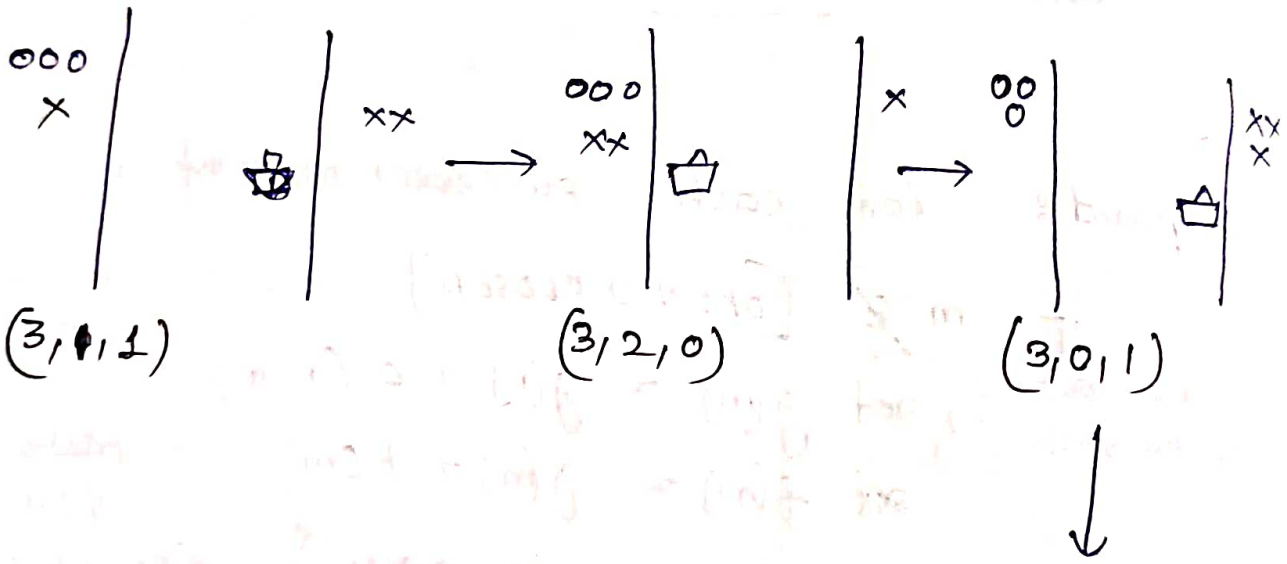
MMM
CCC

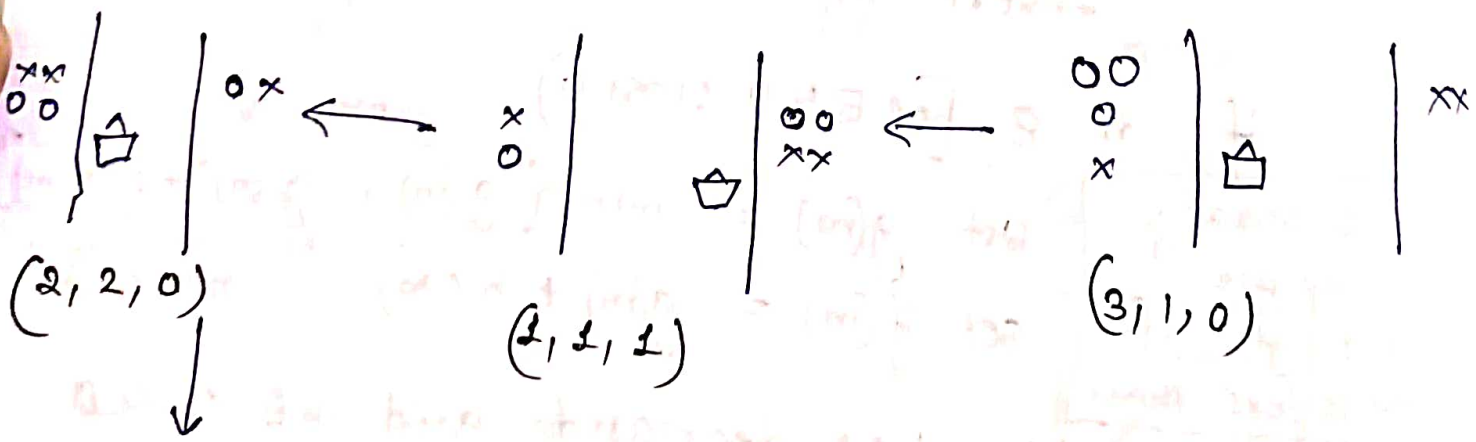


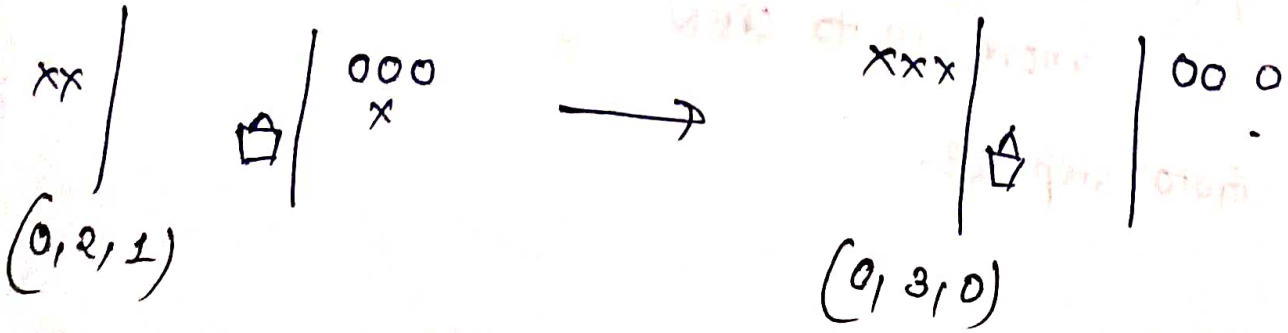
Two are allowed at a time.

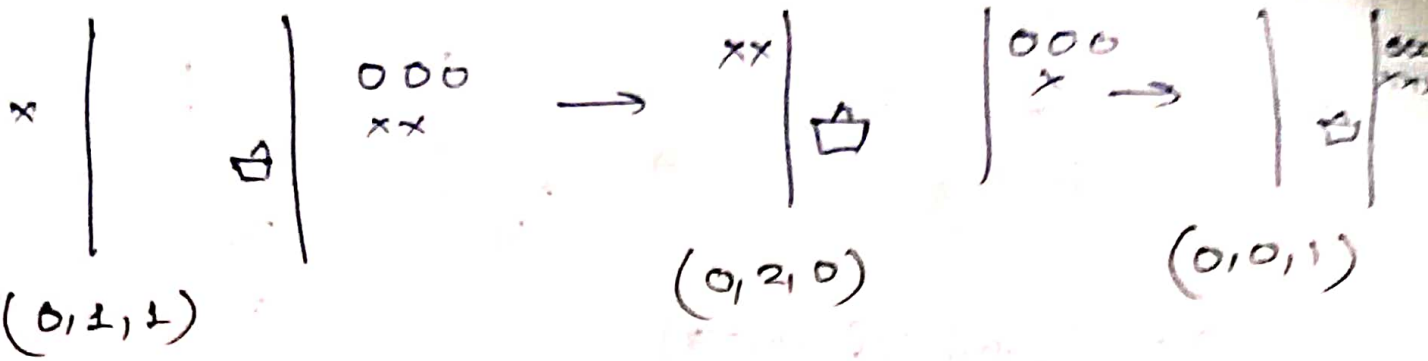
S- (3, 3, 0)
M C B

G- (0, 0, 1)









Solution:

(3,3,0)
(3,1,1)_
(3,2,0)
(3,0,1)_
(3,1,0)
(1,1,1)_
(2,2,0)
(0,2,1)_
(0,3,0)
(0,1,1)_
(0,2,0)
(0,0,1)_

Total 6 rounds

(S, s, o, G)

s- start state (3,3,0)

G- Goal state (0,0,1)

S- Intermediate states

O- operators. number of missionaries or cannibals that are transferred.

operators

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

(m, c)

m: missionaries or cannibals transferred to the other side