AI — SR

① What is AI ?

Intelligence – Acquire, understand and apply knowledge.

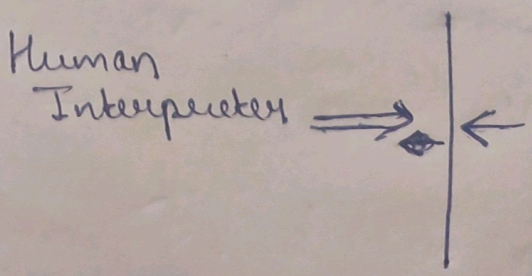
* Reasoning
* Inference.

Branch of Computer Science that builds Computing system with intelligence.

Systems think Like human	Systems think rationally
Systems act Like human	Systems act rationally

① System acts like human.

② Turing Test.



II Thinks like human.

III Acts ~~like~~ Think rationally
↳ does right thing.
↳ Connect different facts rationally.
↳ Challenge - knowledge with uncertainity.

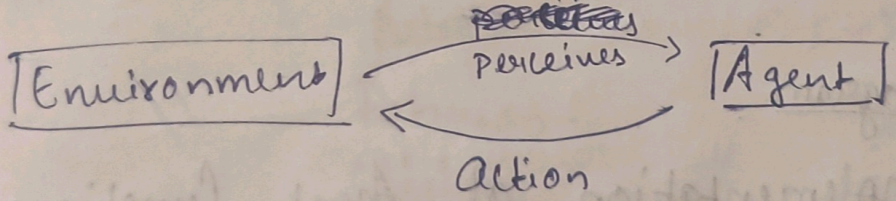
(IV) Act Rationally.

Acting based on knowledge base to maximize performance.

* System that acts rationally is a general approach to build intelligent system.
  - Chess engine
  - Expert systems.

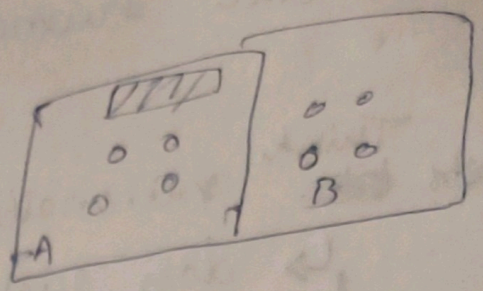
Agent

↳ perceives something through environment and performs some action.



* Percept
  Percept sequence.
* Agent performs some action based on percept sequence and not the actual input

Ro



Percept	Action
A, clean	Right
(A, clean) (A, clean)	Right

Agent Function

Mapping from percept sequence to action

Agent Program.

Agent Program.

T – lifetime of agent

P – # percept

$\sum_{t=1}^{T} P^t$ — Size of table.

∴ Agent function is not preferred as size of table to store fn can be large

Agent Program.

Agent Program

- Implementation of Agent function
- Mathematical Model.
- Agent fn takes decision on percept history.
- Agent program takes decision on percept state.

Agent = Agent Program + Architecture.

Task of AI – build specific Agent Program.

Rational Agent

Performance Measure
- ↳ depends on task.
- ↳ depends on type of agent.

For rationality we should have performance measure.

Performance ~~measure is depends~~ on
- ~~①~~ action
- ~~②~~ Knowledge base
- percept sequence.

④ for a given percept sequence, a rational agent ~~tries to maximize~~ takes an action that tries to maximize the performance measure depending on knowledge base.

Reference Book – Russel and Norvig & for today's portion.
~~Peterson~~
Peterson.
Nielson.

④ Why tabular form not considered.
④ Diff. between Agent fn & agent prog.

Agent Environment

* Agent is something which perceives something from environment and performs some action
* Agent function and Agent Program
  Action depends on environment.

Types of Environment :-

1. Depending on Observability :-

Fully Observable vs Partially Observable.

↳ Agent can take all the relevant information

↳ Noise in observation
↳ Agent needs to think beyond the received ~~inform~~ information
↳ history or state information is needed to take particular action.

2. Based on Determinism.

Deterministic Vs Stochastic ⟶ Action and past state does not determine next state

Action and past state determine next state

↳ ~~Action determined~~.
Past ~~and present~~ next ~~state~~

↳ For fully observable.

Action ~~does not~~ Depend ~~on past state~~ and ~~next~~ state

↳ for partially observable.

3. Episodic vs Sequential.

↳ Action does not depend on previous episode.

E.g. identifying defective parts in assembly line.

↳ Decision of one episode depend on previous episodes.

E.g. chess playing

④ Static vs Dynamic.

Static
- Environment is fixed (not changing) w.r.t. time when agent is taking action

Dynamic
- Environment updates w.r.t time when agent takes action
- Decision making is harder.

⑤ Single Agent vs Multiagent.

Single Agent
- Only one agent in the environment

Multiagent
- Multiple agent in the environment
- Performance of one agent affects other agent.

⑥ Discrete vs Continuous.
- Discrete → E.g Chess
- Continuous → Automated taxi

Different types of Agent

Objective of AI — building the agent program.

1. Simple reflex agent.
   - * Agent can take some action based on current state description.
   - * Agent can do this if environment is fully observable.
   - * Agent program is simple and rule based
   - * limited intelligence.
   - * Cannot take right decision in partially observable environment.

2. Model based reflex agent
   - → suitable when environment is not fully observable.
   - → How agent action affects world and how the world evolves is used to determine action

(3.) Goal based Agent.

(*) Along with current state, information of Goal is required to take right decision

(*) Take decision in favour of reaching a goal.

(4.) Utility based Agent.

* There may be number of possible action.
* Take decision based on performance measured.
* Performance measure is described in terms of utility function.

Problem.

1. Initial State - Description of the world.
2. Action - Possible actions that can be taken by an agent.

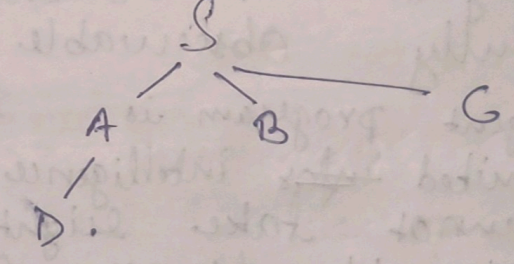
Action is available or not.

Present state + Action = New state.

Action can be described in terms of Successor function.

Successor fn :-

In(S), Go(A) = In(A)

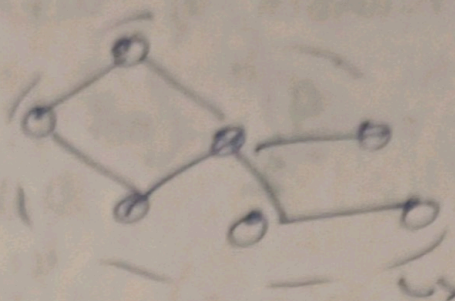


3. Goal test - Whether we reach the goal or not.

(4.) Path Cost :- Cost of the actions.

Define problem formally. Also show that 8 puzzle is a problem. — Question for exam

State - Description of the world at any moment or any time is called state.



↳ State space.

* State Space - All the states reachable from initial state is the state space.

* Goal State,
* A number of possible goals may be there.
* Given initial state agent needs to reach a goal state.

∴ This is a searching problem.

⊛ Solution of problem is to find a sequence of actions.

∴ State space problem is a searching problem.

States which are not reachable from initial state are called not in state space.

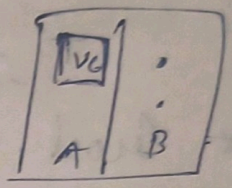
8 puzzle problem

* State: Configuration of 8 tiles & one blanks space in a 3×3 array
* Initial: Any of the state is considered as initial state.

* Action: Moving blank space in left, right, up, down.
* Goal test: Checking of the goal.
* Path Cost: Number of actions / moves.

State space = $\frac{9!}{2}$ (not 9! because not all the states are reachable. because space can move only one).

* Vaccum cleaner problem ($2^2 \times 2$).



8-queen problem

* State: Placement of 8 queens in a 8x8 matrix.
* Initial State :- No queens in the board.
* Action: Place one queen at one box.
* Goal test :- Check if queens are attackable from each other or not.
* Path cost — Placement of one queen

State space = $64 \times 63 - - 57$

Redefine to reduce state space

State: Placement of any n (where $0 < n < 8$) number of queens starting from left such that no two queens are adjacent.

Water Jug Problem

You have two jugs. 3l and 4l. Fill 4l jug if only 2l water is there in it.

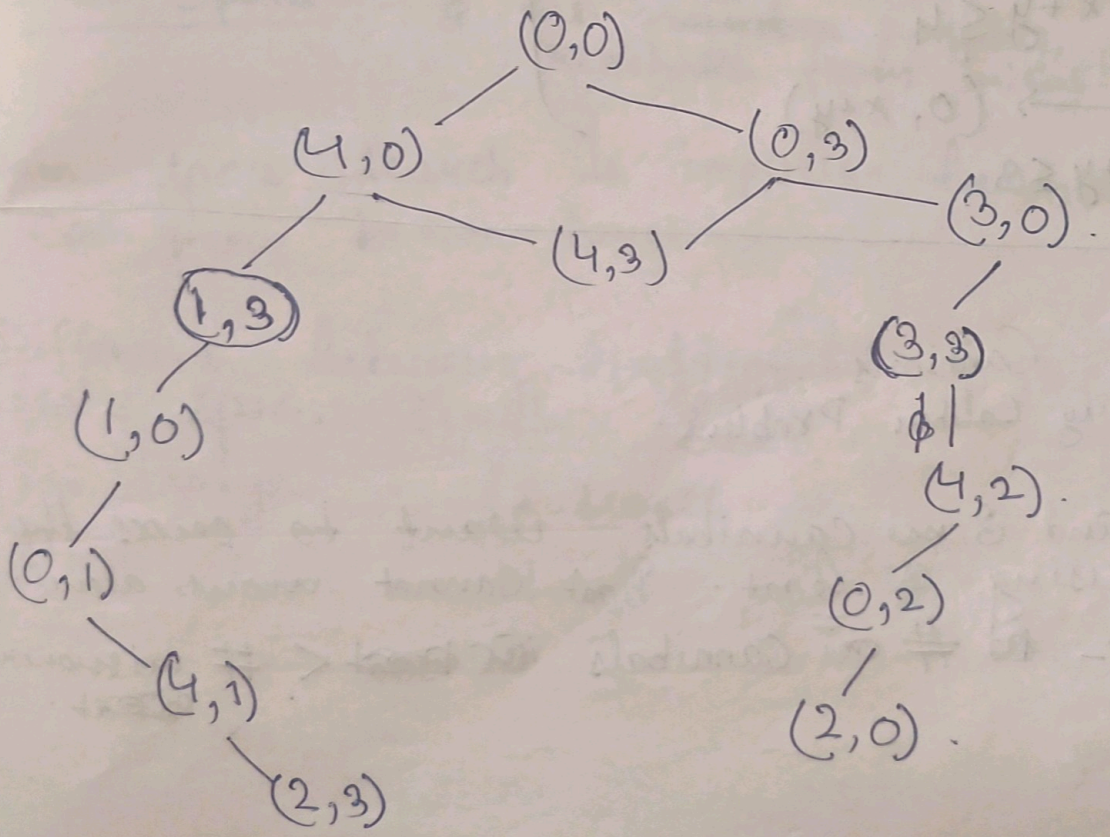
State :- x: amount of water in the 4l jug.
y: " " " 3l jug.

Initial State : $(x, 4) = (0, 0)$.

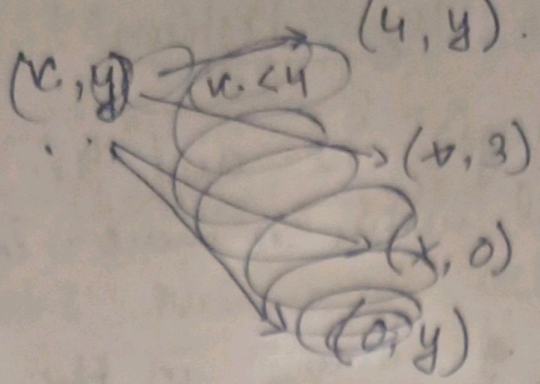
Goal State : $x = 2l$. 2l water in the 4l jug.

Action :- ~~Empty a jug or fill the jug.~~

1. Fill the jug (one at a time)
2. Transfer from one jug to another.
3. Empty the jug (one at a time).



Action



Actions

(1) $(x,y) \xrightarrow[x<4]{} (4,y)$

(2) $(x,y) \longrightarrow (x,3)$

(3) $(x,y) \longrightarrow (x,0)$

(4) $(x,y) \longrightarrow (0,y)$

(5) $(x,y) \longrightarrow (y, y-(4-x))$
$x+y \geq 4,\ y > 0$.

(6) $(x,y) \longrightarrow (x-(3-y), 3)$
$x+y \geq 3,\ x > 0$.

(7) $(x,y) \longrightarrow (x+y, 0)$
$x+y \leq 4$.

(8) $(x,y) \longrightarrow (0, x+y)$
$x+y \leq 3$.

State Space.

23/1/24

Cannibals & Missionary Problem.

3 missionary and 3 cannibals want to cross the river using a boat. Boat cannot move alone.
Constraint:- # cannibals < # missionary in boat.

State Description :- Using a three tuple. (# missionary (left side), # Cannibals (left side), L/R) ← Position of boat.

Initial State : (3, 3, L) or ~~(3, 3, L, 0, 0)~~

(3,3,L) / 2C moved → (3,1,R)

(3,3,L) / 0,0

~~3,3,~~ (3,3,L) — 2C → (3,1,R); — 1C → (3,2,R); — 1C, 1M → (2,2,R)

(2,3,R), (1,3,R) } Not valid states

Only consider the valid states.

(3,2,L)

While drawing state space :- * Don't consider an action which brings to any previous state

* Don't consider invalid states.

State Space :- The ~~number~~ possible valid states reachable from initial state.

State space search is implicit because it can grow ~~to~~ too large.

Difference between traditional search (dfs) and state space search :- State space ~~search~~ is too large ∴ ss search is implicit. Previous states can be therefore disregarded.

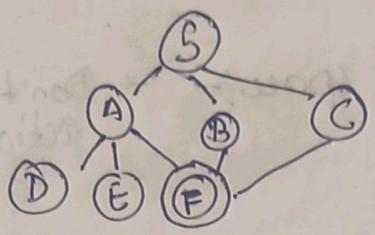
Performance Measure of State Space Search Technique:-

It is measured based on 4 parameters:-

① Completeness - A technique is Complete if it returns the solution. If a solution is there then search method can find it.

② Time Complexity - Number of nodes explored.

③ Space Complexity - Number of nodes generated



Open nodes are those which are generated. We can store them in list/ queue/ stack.

Open list/open nodes:

$O_p \rightarrow S_2, A, B, S$

$CL \rightarrow S$

Closed list is list of nodes which are already visited.

Branching factor (b) = No. of branches from a node.

$d = {}^{max}_{n}$ depth of Shallowest goal.

(iv) Optimality

(b) Search Methods

(A) Uninformed search

(1) Except the problem defn there is no other information is available

(2) Information that

(3) Blind search

(4) No preference for any node.

Eg.

Informed Search

* Some extra info. is available.

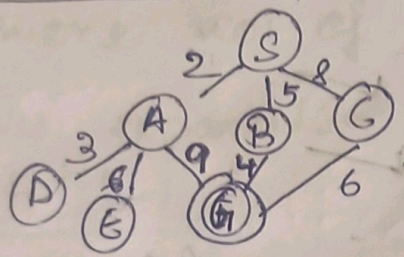
* Extra information like some state is better than another.

* goodness of a node.

E.g. Extra info about distance, between any two distance is given.

(b) Informed Search Techniques :-

BFS (Breadth First Search) :-



Exp. node	OP	CL.
	S	
S	ABC	S.
A	BCDEG	S, A
B	CDEG G'	SAB
C	DEG G' G''	SABC
D	EG G' G''	SABCD
E	G G' G''	SABCDE
G		

Optimal ✗ ∵ 9+2 = 11 is not optimal.

Complete ✓ G' → successor of B.

Time Complexity :- $1 + b + b^2 + \dots b^d = \frac{b^{d+1} - 1}{b-1}$

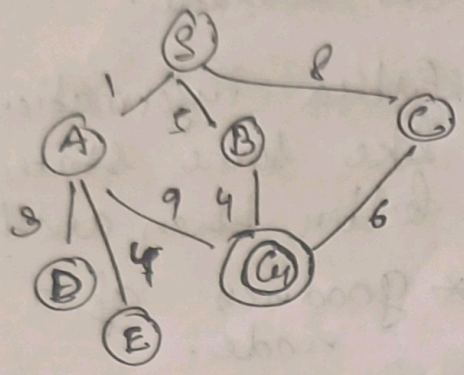
$= O(b^d)$ if $b << d$.

Space Complexity – $O(b^d)$

(*) DFS :-
- Not optimal
- Complete ✗
- TC : $O(b^d)$
- SC : $O(b \times d)$

Uniform Cost Search.

↳ Select node depending on G_i value: ~~dy~~ cost from ~~goal node~~ initial node.



Exp node.	Op	Cl.
S		
S	A(1) B(5) C(8)	S.
A	D(4) E(5) B(5) C(8) G1(10)	S, A
D	E(5) B(5) C(8) G1(10)	S, A, D.
E	B(5) C(8) G1(10)	SADE
B	C(8) G1'(9) G1(10)	SADEB.
C	G1'(9), G1(10) G1''(14)	SADEBC
G1'		SADEBC

Complete ✓

Optimal ✓

T.C. : Same as bfs if edge cost is same.

$= O(b^{C^*/\epsilon})$. C^* = Optimal cost

ϵ = minimum cost of an action.

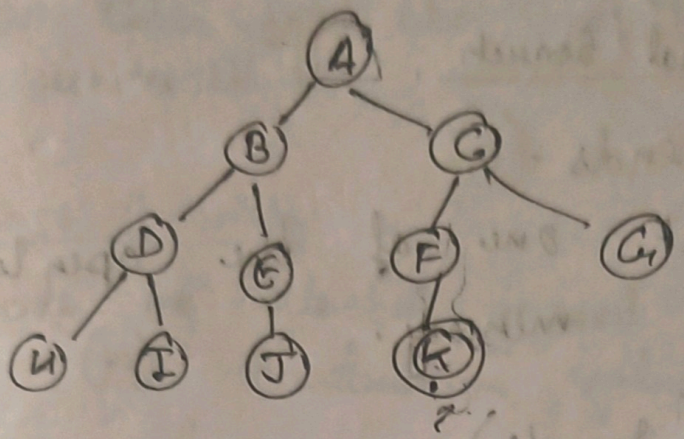
$b^{C^*/\epsilon} >> b^d$.

∴ Time req. is more than normal bfs.

Iterative Deepening Depth First Search.

Takes advantages of both techniques

DFS with some depth bound; increase the depth bound if the solution is not present



Complete ✓

Optimal ✓ (Consider case where edge costs are same.

T.C :- $O(b^d)$ $[1(d+1) + bd + b^2(d-1) - - - b^d]$.

S.C :- $(b \times d)$

Although T.C. is same but we need to expand more no. of nodes than BFS/DFS.

No. of nodes expanded :-

$$N_{BFS} = \frac{b^{d+1}-1}{b-1}$$

$$N_{ID} = \sum_{j=0}^{d} \frac{b^{j+1}-1}{b-1} = \frac{1}{b-1}\left(b\sum_{j=0}^{d} b^j - \sum_{j=0}^{d} 1\right).$$

$$= \frac{1}{(b-1)}\left(b \cdot \frac{b^{d+1}-1}{b-1} - (d+1)\right).$$

$$= \frac{b^{d+2}-b-bd-b+d+1}{(b-1)^2} = \frac{b^{d+2}-bd-2b+d+1}{(b-1)^2}$$

For large d, $N_{ID} = \frac{b^{d+2}}{(b-1)^2}$.

$$\frac{N_{ID}}{N_{BFS}} = \frac{b}{(b-1)^2}$$

Bidirectional Search

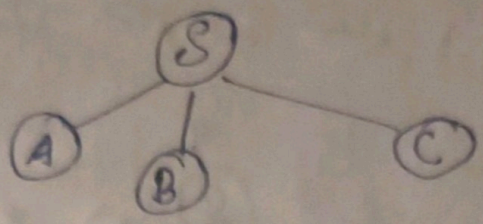
- Search from both ends.
- We need at least one of the open list to be stored in memory.

$$T.C^{n} O(2.b^{d/2})$$

- For bidirectional search we must should have an inverse action to reach parent from a child node.

Also, if there are multiple goals then where to start?

Informed Search



* Status of any state from the goal is known.
* based on estimations.

n

1) Best First Search.

Best node is selected based on some evaluation value $f(n)$. Evaluation value is just some cost.

Heuristic function

* Heuristic is basically a rule of thumb for making some choice. We can get heuristic based on the domain of a problem. There is no such general way.
* Heuristic does not guarantee a solution.
* near optimal solution.

Travelling Salesman Problem

For n cities :- Complexity → Exponential

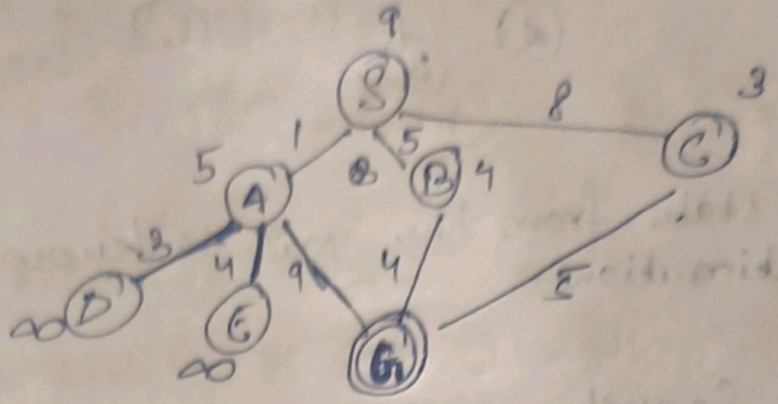
Using Heuristics. Complexity = $(n-1)+(n-2) \cdots 1$

$= O(n^2)$

Heuristic estimation — choose the nearest neighbour.

Greedy Best First Search:-

$f(n) = h(n)$



	exp. node	Op.	cl.
		S	
	S	A C(3) B(4) A(5)	S.
	C	G(0) B(4) A(5)	S C.
	G		

↳ Goal.

∴ Using greedy best first search, we are getting the soln but it is not optimal.

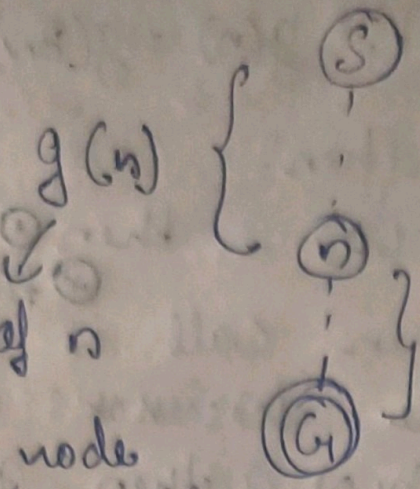
Running Time: $O(b^d)$

Space:- $O(b^d)$

Complete: ✗.

↳ It is not complete because at every time node is chosen based on estimation.

A* search algorithm

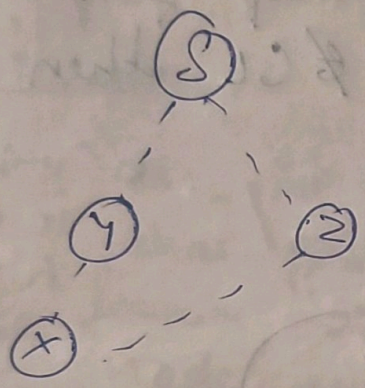


Cost of n from start node S so far

$h(n)$ → estimated cost from n to Goal.

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

estimated cost from start S to goal through S.



(*) g value may be updated

Algorithm:-

1. Op = { Initial state S }.
   f(S) = h(S) ; CL = Null.
2. Do the following until goal is found
3. If Op == Null, then return fail.
4. else. Select the best node Bn from Op.
5. If Bn is Goal then terminate with success
6. ~~else~~ Else generate all the successors of Bn:

6.1 for each successor n of B_N do.

Parent (n) ← B_N; $g(n) = g(B_N) + C(B_N, n)$

6.2 If $n \in OP$ then call it OLD

(Imp. for exam) if $g(n) < g(OLD)$ then ① $g(old) \leftarrow g(n)$ ② $f(OLD) = g(OLD)$ + h

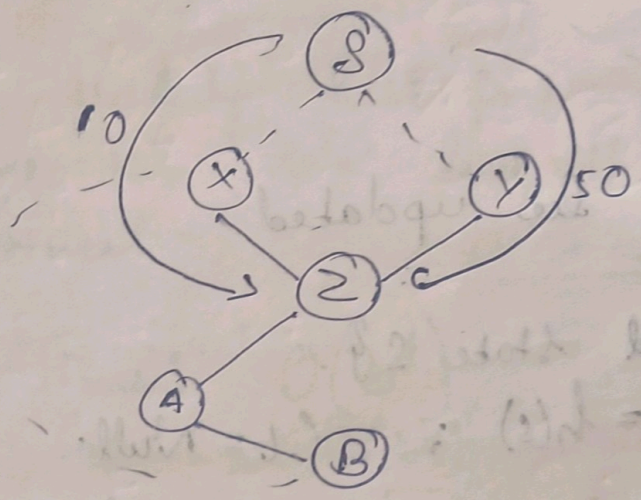
6.3 If $n \in CL$ then call it OLD and add it to the successors of B_N

If $g(n) < g(OLD)$ then $g(OLD) \leftarrow g(n)$

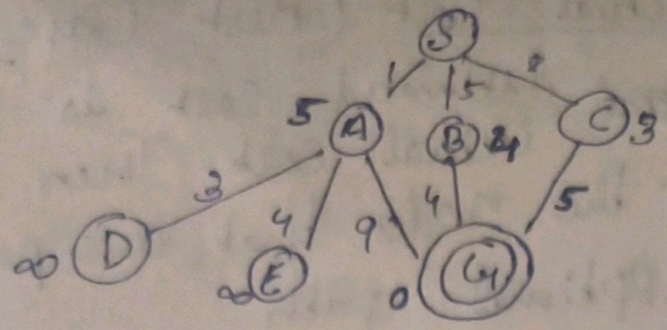
To propagate the new g value to depth first traversal starting at n. and terminate if there is a branch with no successor or a node with better path already found.

6.4 If $n \notin OP$ and $n \notin CL$ then add it to OP.

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



(B) Why is there a need to propagate g value if a node is in closed list.



Exp node	Openlist	CL
	S	
S	A(6), B(9), C(11)	S
A	B(9), G1(10), C(11), D(∞), E(∞)	S, A
B	G1(9), C(11), D(∞), E(∞) (updated)	S, A, B
G1		

Performance Measure :-

Complete :- Yes. If there is infinite depth then that path will not be explored forever. ~~Since we can always get a node with ... value less than~~ Since after some time the f value will be high and that node will not be explored any further.

Time Complexity :- Worst Case :- $O(b^d)$

Best Case : $O(d)$

If we only travel along the shortest path that leads to goal. This happens when our estimation is ~~equal to the Actual~~ Correct

S
:
n
:
G1
} h(n)

(*) $h(n)$ = Actual Cost (Best Case)

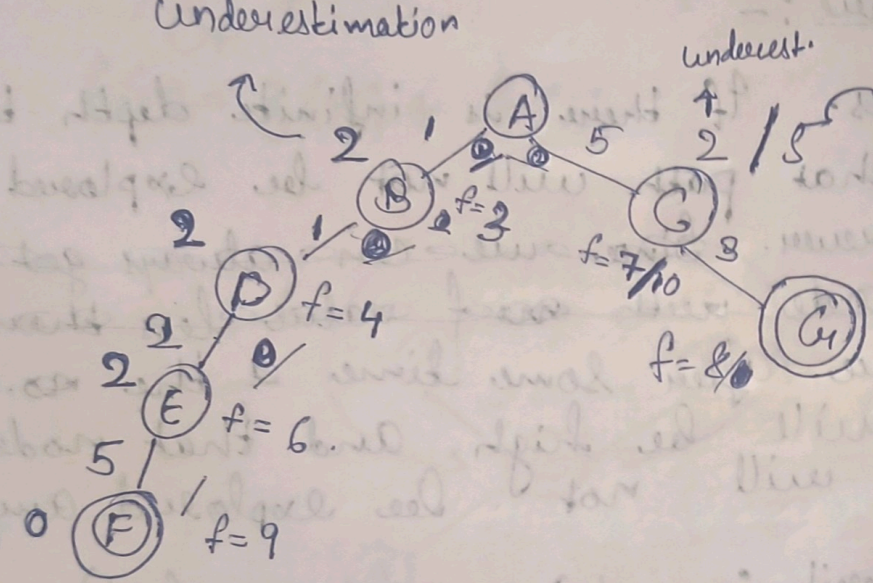
$h(n)$ < Actual Estimated cost is same as actual cost. Then we explore only the nodes that are present in the optimal path.

(*) $h(n) <$ Actual Cost

We will have to visit more nodes than the nodes present in optimal path. But optimal path is guaranteed.

(*) $h(n) >$ Actual Cost

We might not get the Optimal path.



After F is visited, G is explored as it's cost is lesser than F. So G will be pushed before F and explored.

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

↳ optimal cost to reach goal

↳ optimal cost to reach n from start node.

④ $g^*(n) \leq g(n)$

Admissibility of A*

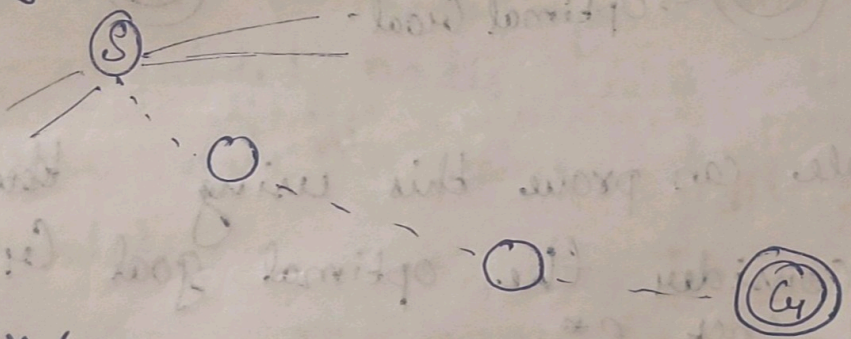
① Every arch (operator) has some minimum positive cost. $\epsilon > 0$.

② Finite branching factor (b).

③ For any node $h(n) \leq h^*(n)$.

Properties of A*

① For any node n in optimal path, $f^*(n) = C^*$ where C^* is the optimal cost to reach the goal node.



Proof.

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

$$C(S,n) + \underset{\min}{C}(n, G_1)$$

$$= \underset{\min}{C}(S, G_1) = C^*$$

$S \to n_1 \to n_2 \cdots n_g$

$$f^*(S) = f^*(n_1) = f^*(n_2) = \cdots f^*(n_g) = C^*$$

(2.) Any time before A* terminates; there exists a node n in open list (OPEN) in the optimal path such that $f(n) \leq f^*(n)$ (C^*)

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

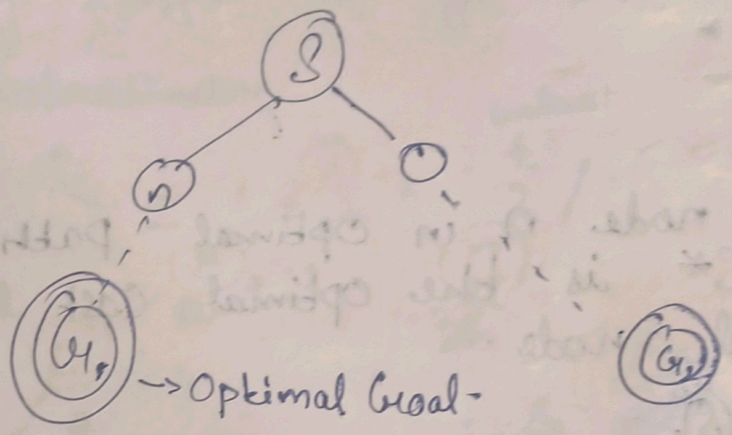
As the node is in optimal path $g(n) = g^*(n)$

$\because$ Graph is admissible, $h(n) \leq h^*(n)$

$$f(n) = g(n) + h(n) = g^*(n) + h(n) \leq g^*(n) + h^*(n)$$

$$\Rightarrow f(n) \leq f^*(n)$$

(3.) Optimality of A* :-



Proof :- We can prove this using Contradiction

Let us consider the optimal goal G_1 with optimal cost C^*.

Consider G_2 is suboptimal goal.

So $f(G_2) > C^*$ ——— (1).

* Assume A* terminates after expanding G_2.

Now n is an unexpanded node in OPEN in optimal path.

Any time before A* terminates there exists a node n in optimal path in OPEN, such that $f(n) \leq C^*$ — (2) —— Proved before

~~A* prefers G_2 over n - as A* has~~ terminated after

As n & G_2 both are in OPEN and A* chooses G_2 over n. $\therefore f(G_2) \leq f(n)$

$f(G_2) \leq C^*$.

So there is the contradiction from eqn -①.
i.e. $f(G_2) > C^*$.

(4) (5) Prove that A* must terminate.

Assume A* does not terminate. and minimum edge cost is $\epsilon > 0$.

$g(n) \geq l \cdot \epsilon$. ; l is no. of edges from S to n.

As A* is not terminating then g-value of ~~nodes~~ expanded nodes will increase.
∴ For some node, n' g-value of a node will be unbounded, $g(n') > C^*$.

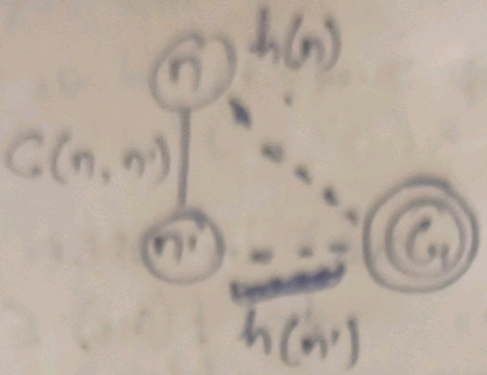
$\because f = g + h$.

∴ $f(n')$ will also be unbounded.

∴ $f(n') > C^*$.

But there exists a node n in Open list (OPEN) such that $f(n) \leq C^*$

Consistent heuristic



(*) $h(n) \leq h(n') + c(n, n')$

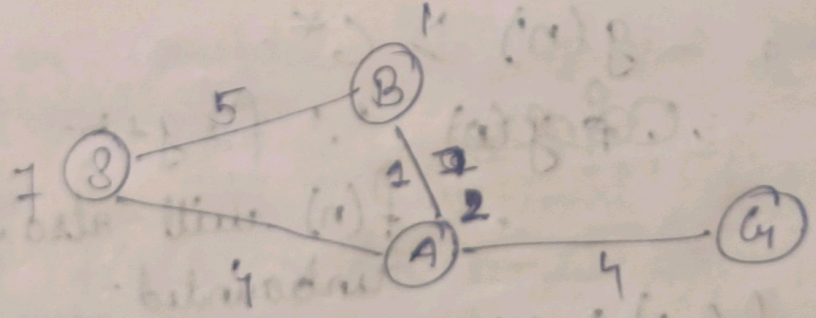
Condition for Consistency.

* Let n be a node and n' be its successor, then the consistency condition should hold for all the nodes in graph.

Concave

H.W. (*) Can we get optimal result if the heuristic is not Consistent.

(*) Design some heuristic which is inconsistent but admissible.



Inconsistent but heuristic is admissible.

(5) Prove that if the heuristic is consistent then for the nodes of the state space the f-values are non-decreasing.

(*) f value may be decreasing if heuristic is inconsistent

$$h(n) \leq h(n') + c(n, n') \quad \text{— (1)}$$

$$\Rightarrow h(n') + c(n, n') - h(n) \geq 0$$

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

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

~~$f(n) - f(n') = h(n) - h(n') + g(n) - g(n')$~~

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

$$f(n') - f(n) = h(n') - h(n) + c(n, n')$$

$$f(n') - f(n) = h(n') + c(n, n') - h(n)$$

$$f(n') - f(n) \geq 0 \quad \text{— from (1)}$$

$$f(n') \geq f(n)$$

Hence proved

(6) Show that if heuristic is consistent then it is admissible too.

Let us consider an optimal path

$$S \rightarrow n_1 \rightarrow n_2 \cdots n_g.$$

For any two node if heuristic is consistent then

$$h(n) \leq h(n') + c(n, n')$$

For any two node s and n_1 where n_1 is the successor of s

$$h(s) \leq h(n_1) + c(s, n_1).$$

$$h(s) - h(n_1) \leq c(s, n_1).$$

Similarly

For n_1 and n_2:

$$h(n_1) - h(n_2) \leq c(n_1, n_2).$$

⋮ till the nth goal node.

For n_{g-1} & n_g

$$h(n_{g-1}) - h(n_g) \leq c(n_{g-1}, n_g).$$

Adding all the lhs & rhs.

$$h(s) - h(n_g) \leq c(s, n_1) + c(n_1, n_2) - - - c(n_{g-1}, n_g).$$

$\Rightarrow h(s) - h(n_g) \leq c^*$ ∵ it is the optimal path.

As n_g is goal ∴ $h(n_g) = 0$.

$\Rightarrow h(s) \leq c(s, n_1) + c(n_1, n_2) - - - c(n_{g-1}, n_g)$

c^* ∵ it is optimal path.

$\Rightarrow h(s) \leq h^*(s)$ or $h^*(s)$

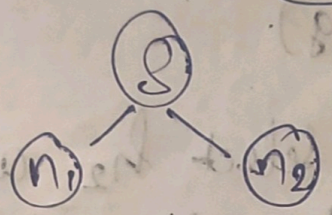
Question

(a) If edge is admissible then it is Consistent. — Prove or Disprove -

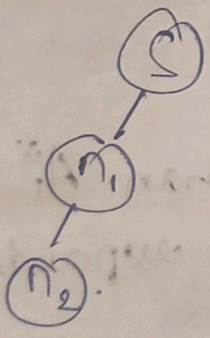
If A^* expands a node n_2 immediately after n_1, then $f(n_1) \leq f(n_2)$.

Condition (I) - both n_1 & n_2 will be there in OPEN.

Condition (II) - n_2 is not there in OPEN when n_2 is expanded



Condition (II) n_2 is not there in OPEN when n_2 is expanded



Q. A* expands all the nodes for $f(n) < C^*$.

↳ Prove.

For a given problem there might be more than one heuristic, which one to choose.

$h_1(g) \leq h^*(n)$ } Admissible.
$h_2(g) \leq h^*(n)$ }

$h_2(g) > h_1(g)$.

(*) Then better to select h_2 as it may lead to expanding lesser no. of nodes.

Take largest values nearer to the actual cost

A_2^* expands lesser nodes than A_1^*

A_2^* " atleast all the nodes expanded by A_1^*.

4/2/24

If we have a heuristic larger enough but admissible then that heuristic is preferable.

$A_1^* \rightarrow h_1$
$A_2^* \rightarrow h_2$

$h_2 > h_1$

$h_1 \leq h^*$
$h_2 \leq h^*$

$A_1^*, A_2^* \rightarrow$ Admissible.

Theorem:
($h_2 > h_1$)

If A_2^* is more informed than A_1^* then at the termination then A_1^* visits all the nodes expanded by A_2^*.

E.g. :- A_2^* : 10 nodes
A_1^* : $\geq$ 10 nodes.

As A_2^* is more informed than A_1^*, then

$h_2 > h_1$

Proof by Contradiction:-
(Induction)

Let us assume ~~there is a depth~~ different nodes upto depth k that are expanded by A_2^*.

~~Upto depth k, all the nodes visited~~

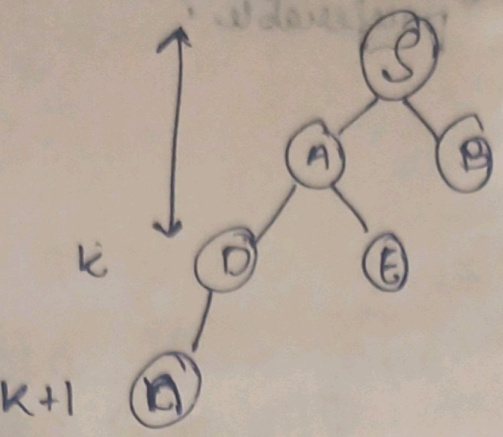
Assume a ~~search~~ search tree generated by A_2^* upto depth k.

A_1^* ~~expands~~ also expands all the nodes till depth k.

Base:- for $k=0$, A_1^* and A_2^* both visit start node.

Assume there is a node n at depth k+1 of search tree of A_2^* & A^* ~~before termination~~

terminates without visiting n.



All the parents of n are expanded by A_1^*

∵ A_1^* never visits n, and A_2^* visits it, this implies

$$g_1(n) \leq g_2(n). \quad \text{— (2)}$$

Cost of s to n by A_1^* cannot be greater than cost of s to n by A_2^*

For a node n not expanded by A_1^* then $f_1(n) \geq f^*(s)$ — (3) (else n would have been expanded).

As A_1^* does not expand n so

$$f_1(n) \geq f^*(s) \quad \text{— (4)}$$

If A^* expands a node n then $f(n) \leq f^*(n)$

As A_2^* expands n, ∴ $f_2(n) \leq f^*(s)$.

∴ $f_1(n) \geq f_2(n)$ (from 3 & 4).

$$\Rightarrow g_1(n) + h_1(n) \geq g_2(n) + h_2(n).$$

$\therefore g_1(n) \leq g_2(n)$

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

This is a contradiction.

Hence the theorem is proved.

Theorem:

In A* algorithm with consistent heuristic when a node is expanded then all the node with lower f value are already expanded.

Proof:-

~~Let A* is about to expand a node n~~

Consider ~~a path~~ in the path

$P = \langle n_1, n_2 - - - n_i, n_{i+1} - - n_k - - \rangle$

~~such that~~ and $f(n_k) < f(n)$ [n is later than n_k].

Assume n_i is the last expanded node,

$\therefore n_{i+1}$ is in the OPEN also n is in OPEN.

As heuristic is consistent so f values are non dec. along paths.

$\therefore f(n_{i+1}) \leq f(n_k)$.

$f(n_{i+1}) < f(n)$. [Assume $f(n)$ is chosen over n_{i+1}]

If n_2 is expanded immediately after n_1 then $f(n_1) < f(n_2)$.

$\therefore f(n_{i+1}) \leq f(n_{i+1})$

This is the contradiction $\therefore$ Our assumption is wrong.

Theorem:-

If the heuristic is consistent then when A* expands a node n then it already found the optimal path upto n.

Consider that A* is about to expand a node n in the optimal path.

$$P = n_0 n_1 \cdots n_l n_{l+1} \cdots n$$

Assume n_l is the last expanded node

$$g^*(n_{l+1}) + h(n_{l+1}) = g^*(n_l) + c(n_l, n_{l+1}) + h(n_{l+1}).$$

$$\geq g^*(n_l) + h(n_l)$$

As the heuristic is consistent then.

$$h(n_l) \leq h(n_{l+1}) + c(n_l, n_{l+1})$$

For a path n_j is successor of n_i

$$n_j \geq n_i$$

$$g^*(n_j) + h(n_j) \geq g^*(n_i) + h(n_i)$$

$$g^*(n) + h(n) \geq g^*(n_{l+1}) + h(n_{l+1})$$

$$g^*(n_{l+1}) = g(n_{l+1}))$$

$$\therefore g^*(n) + h(n) \geq f(n_{l+1})$$

But from computation we know, $g(n) \geq g^*(n)$. —— (1)

A* expands the node n

A* is about to expand n then $f(n) \leq f(n_{n+1})$

$\therefore g^*(n) + h(n) \geq f(n)$.

$\Rightarrow g^*(n) \geq g(n)$ ← (II)

∴ from (I) & (II) $g(n) = g^*(n)$

13/2/24

Admissible heuristics for 8-puzzle

If blank position is at

1. Center — 4 moves (u, l, r, d)
2. corner — 2 moves (l/r, u/d)
3. edge — 3 moves (l, r, d/u)

On average branching factor b = 3.

Let's say on average we need 22 actions.

in that case $bd = 3^{22}$ (visit states).

G_1 = 1 2 3 / 4 5 6 / 7 8

Initial = 8 1 3 / 4 2 5 / 7 6

(1.) Assume we can move any tile to any position (hypothetical).

For this heuristic, # misplaced tiles = 5 (w.r.t. goal) (1, 2, 5, 6, 8)

* No. of minimum moves required = 5.
* # misplaced tiles can give me a possible heuristic.
* To prove that this heuristic is admissible, we need to prove that the actual cost is greater.
* ∵ we cannot move any tile to any pos., this heuristic is minimum.

2. Minimum Distances of the tile from their goal positions.

$(1(1) + 1(2) + 0(3) + 0(4) + 2(5) + 2(6) + 0(7) + 3(8)) = 8$

But this is possible only when all the adjacent path is blank.

~~But this is not~~ actual

But in actual case this is not possible and ∴ Actual cost > 8.

∴ This is the minimum possible no. of actions assuming the heuristic.

(*) h_2 is better than h_1.

If $h = 0$, then the ~~cost is~~ search is uniform cost search.

b^* = effective branching factor.
b = actual " ".
N = no. of nodes expanded.

$N = 1 + b^* + b^{*2} \cdots b^{*d}$.

If effective branching factor (b^*) is close to 1 then the heuristic is a good heuristic.

Let's assume
$h_1 \sim 30$ nodes.
$h_2 \sim 22$ nodes. (b^* close to 1.1)

Exam
What is the effect of branching factor on heuristic?

→ Performance of A* depends on heuristic, better heuristic → lesser node expanded.
as a result lower branching factor.
b^* close to 1 for a good heuristic.

Generating Admissible Heuristics.

1. Relaxed problem Approach.
A problem with less restriction (fewer constraints) is a relaxed problem.
E.g. 5 constraints → 3 constraints

→ Relaxed problem state space > orig. problem state space.

→ for the relaxed version, we get additional edges and actions.

→ for relaxed version we may get the soln using lower cost as extra edges are there.

→ Solution of relaxed problem can be considered as heuristic as the cost will be lesser than actual cost.

E.g:- For 8 puzzle problem.

Orig:- A tile can move from square X to square Y, if X is vertically or horizontally adjacent to Y & Y is empty/blank.

Relaxed:- A tile can move from square X to (hr_1) square Y.

Another relaxed:- A tile can move from sq X to (hr_2) sq. Y if Y is hor. or vert. adjacent to X.

* hr_2 is obviously better than hr_1 &.

2.) Sub problem. Approach.
Only consider a part of the problem.

Init:-
8 1 3
4 2 5
7 6 .

Goal:-
1 2 3
4 5 6
7 8 .

Sub problem:- Only consider 4 tiles.

* 1 3
4 2 *
* * .

Goal:
1 2 3
4 * *
* * .

Soln. to subproblem is definitely less than actual cost.

1 2 3
4 7 6 → Soln to the
8 . 5 subproblem.

3. Pattern Database.

→ W.r.t. to a subproblem find a similar subproblem with known cost.

If you have a number of subprob

```
* * 4        matched        1 3 4
2 * 6.        to      →     2 7 6
* 8.                        8 5.
```

↓

heuristic for this sub problem is similar to heuristic of this problem.

If you have a number of subproblems and their actual heuristic and we try to match the given subproblem with the pattern in the database and the heuristic for the given subproblem can be taken similar to matched pattern heuristic.

If more than two patterns match take the one with higher cost (better heuristic).

```
* * 4  h=5        1 3 4          1 3 4  h=6.
2 * 6.           2 7 6          2 * *
* 8 .            8 5.           * * .
```

matched matched.

Two subproblem (Chemicals) Cost cannot be added to make a greater subproblem. because the subproblems Cost cannot be may not be independent.

(3) Disjoint Pattern Database.

Take independent subproblems
If subproblems are disjoint, Cost can be added.

(4) Learning from Experience.

(a) Solve lots of problem to gain experience

(b) Same $h = c_1x_1 + c_2x_2$.

Cost can be approximated using other
↑ States that arise during the search.

Reference - Russel Norvig.

6 Questions - 5 to answer. (Each of 10 marks)
Weight is 10.