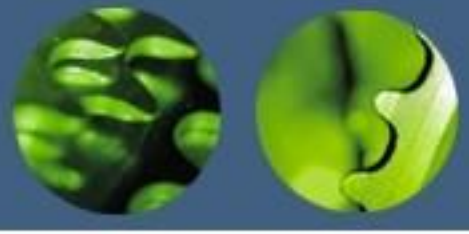
3. Problem Solving by Searching

Compiled By

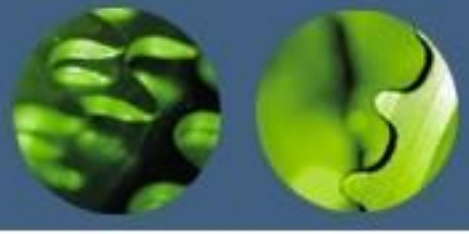
Ujjwal Rijal

rijalujjwal09@gmail.com



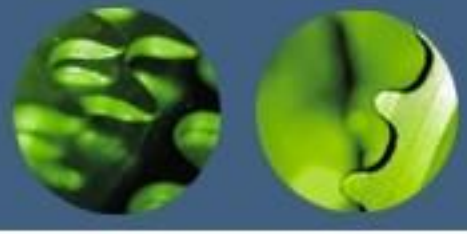
Problem Solving

- ✓ **Problem solving, particularly in artificial intelligence, may be characterized as a systematic search through a range of possible actions in order to reach some predefined goals or solutions.**



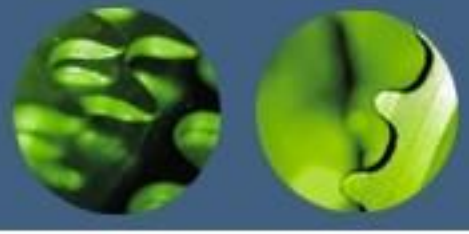
Problem Solving

- ✓ Problem Solving is fundamental to many AI-based applications.
- ✓ There are two types of problems:
  - The problem like computation of sine of a particular angle or a square root of a number. We can solve these kind of problems through deterministic procedure.
  - In the real world, very few problems lend themselves to straightforward solutions.
- ✓ Many real world problems can be solved only by searching for a solution.
- ✓ AI is concerned with these kind of problem solving.



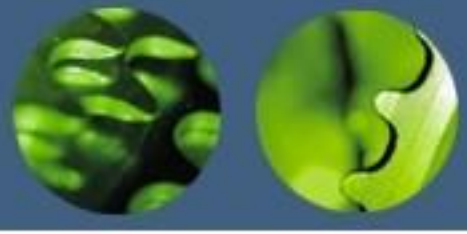
State

- ✓ A state is a representation of elements at a given moment.
- ✓ A problem is defined by its elements and their relations.
- ✓ At each instant of a problem, the elements have specific descriptors and relations; the descriptors tell - how to select elements ?
- ✓ Among all possible states, there are two special states called
  - Initial State is the start point.
  - Final State is the goal state.



State

- ✓ A Successor Function is needed for state change.
- ✓ The successor function moves one state to another state.
- ✓ It
  - – is a description of possible actions; a set of operators.
  - – is a transformation function on a state representation, which converts that state into another state.
  - – defines a relation of accessibility among states.
  - – represents the conditions of applicability of a state and corresponding transformation function

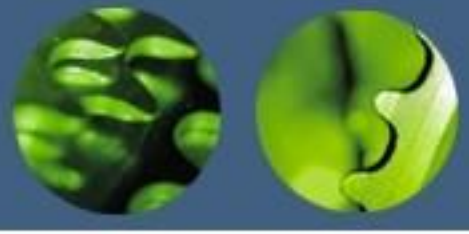


State Space

A State space is the set of all states reachable from the initial state.

Definitions of terms:

- ✓ A state space forms a graph (or map) in which the nodes are states and the arcs between nodes are actions.
- ✓ In state space, a path is a sequence of states connected by a sequence of actions.
- ✓ The solution of a problem is part of the map formed by the state space.



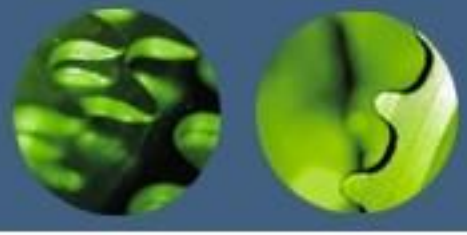
Structure of State Space

The Structures of state space are trees and graphs.

- ✓ Tree has only one path to a given node; i.e., a tree has one and only one path from any point to any other point.
- ✓ Graph consists of a set of nodes (vertices) and a set of edges (arcs).
- ✓ Arcs establish relationships (connections) between the nodes; i.e., a graph has several paths to a given node.

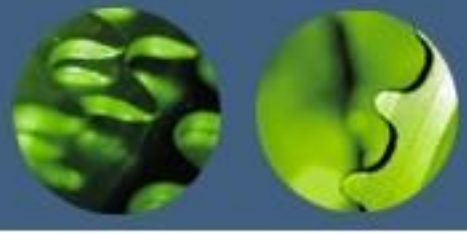
Search process explores the state space.

- ✓ In the worst case, the search explores all possible paths between the initial state and the goal state.



Problem solution

- ✓ In the state space, a solution is a path from the initial state to a goal state or sometime just a goal state.
- ✓ A solution cost function assigns a numeric cost to each path; It also gives the cost of applying the operators to the states.
- ✓ A solution quality is measured by the path cost function
- ✓ An optimal solution has the lowest path cost among all solutions.



Problem Solving

Problem Definitions :

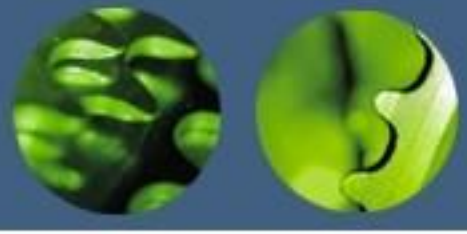
- ✓ A problem is defined by its elements and their relations.
- ✓ To provide a formal description of a problem, we need to do following:

a. Define a state space that contains all the possible configurations of the relevant objects, including some impossible ones.

b. Specify one or more states, that describe possible situations, from which the problem-solving process may start. These states are called initial states.

c. Specify one or more states that would be acceptable solution to the problem. These states are called goal states.

d. Specify a set of rules that describe the actions (operators) available.

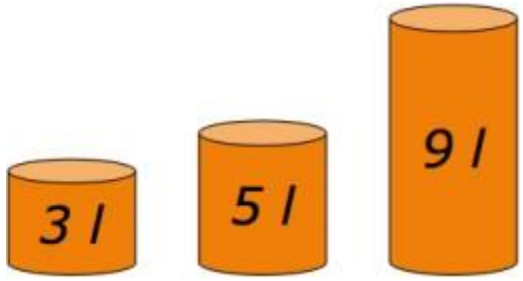


Problem Solving

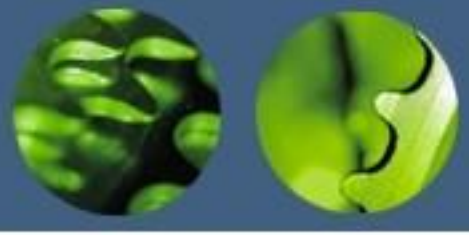
- ✓ The problem can then be solved by using the rules, in combination with an appropriate control strategy, to move through the problem space until a path from an initial state to a goal state is found.
- ✓ This process is known as search.
  - Search is fundamental to the problem-solving process.
  - Search Is a general mechanism that can be used when more direct method is not known.
  - Search provides the framework into which more direct methods for solving sub-parts of a problem can be embedded.
- ✓ **A very large number of AI problems are formulated as search problems.**

Example: Measuring Problem

Problem : Water Jug Problem



Problem : Using these buckets, measure 7 liters of water



Example: Measuring Problem

Measure 7 liters of water using a 3-liter, a 5-liter, and a 9-liter buckets.

- ✓ Formulate goal: Have 7 liters of water in 9-liter bucket
- ✓ Formulate problem:
  - States: amount of water in the buckets
  - Operators: Fill bucket from source, empty bucket
- ✓ Find solution: sequence of operators that bring you from current state to the goal state

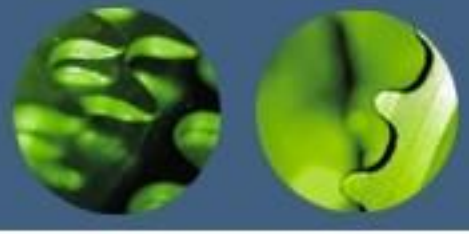
Example: Measuring Problem

Solution 1:

a	b	c	
0	0	0	start
3	0	0	
0	0	3	
3	0	3	
0	0	6	
3	0	6	
0	3	6	
3	3	6	
1	5	6	
0	5	7	**goal**

- **Solution 2:**

a	b	c	
0	0	0	start
0	5	0	
3	2	0	
3	0	2	
3	5	2	
3	0	7	**goal**



Searching Process

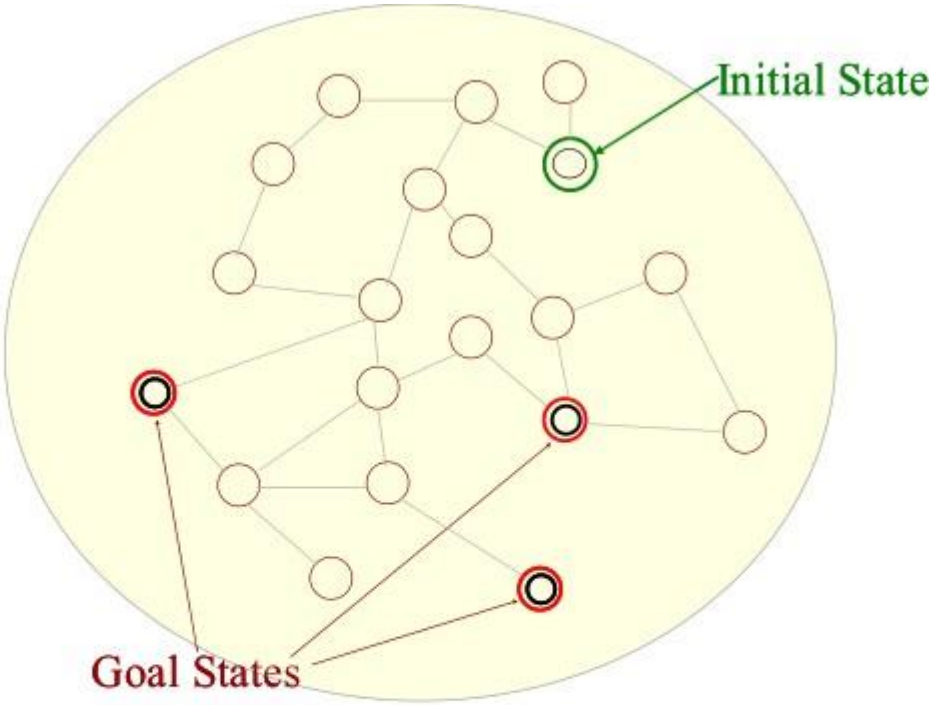
The generic searching process can be very simply described in terms of the following steps:

Do until a solution is found or the state space is exhausted.

1. Check the current state
2. Execute allowable actions to find the successor states.
3. Pick one of the new states.
4. Check if the new state is a solution state

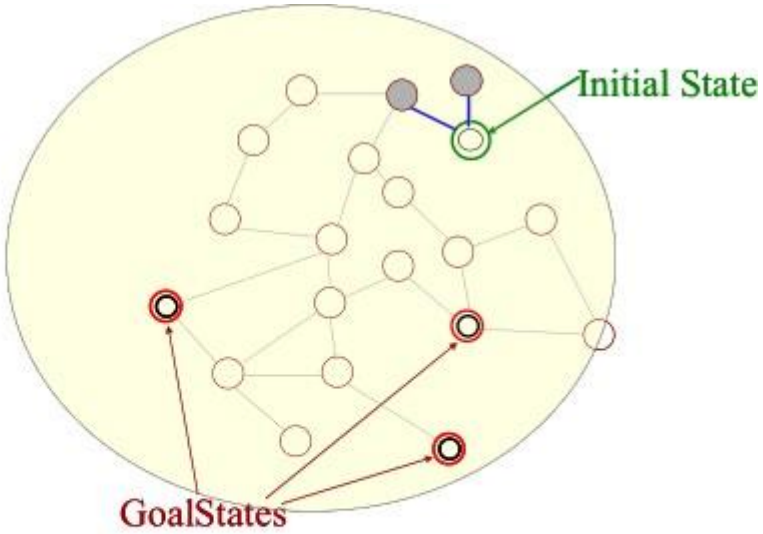
If it is not, the new state becomes the current state and the process is repeated

Illustration of Searching Process



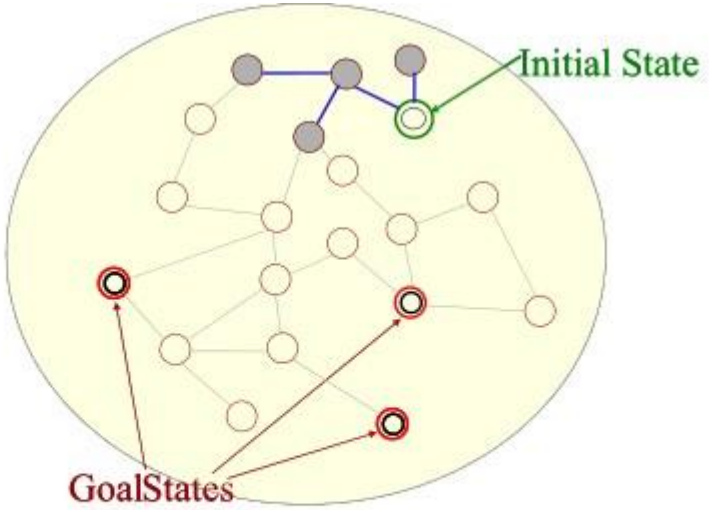
s_0 is the initial state.
The successor states are the adjacent states in the graph.
There are three goal states.

Illustration of Searching Process



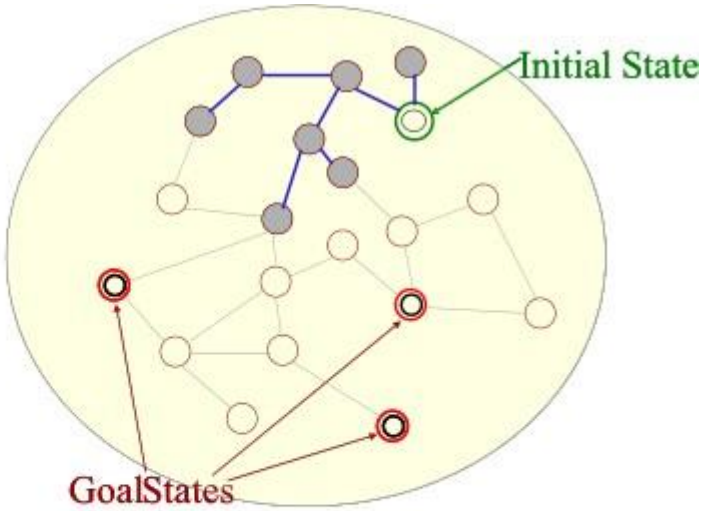
The two successor states of the initial state are generated.

Illustration of Searching Process



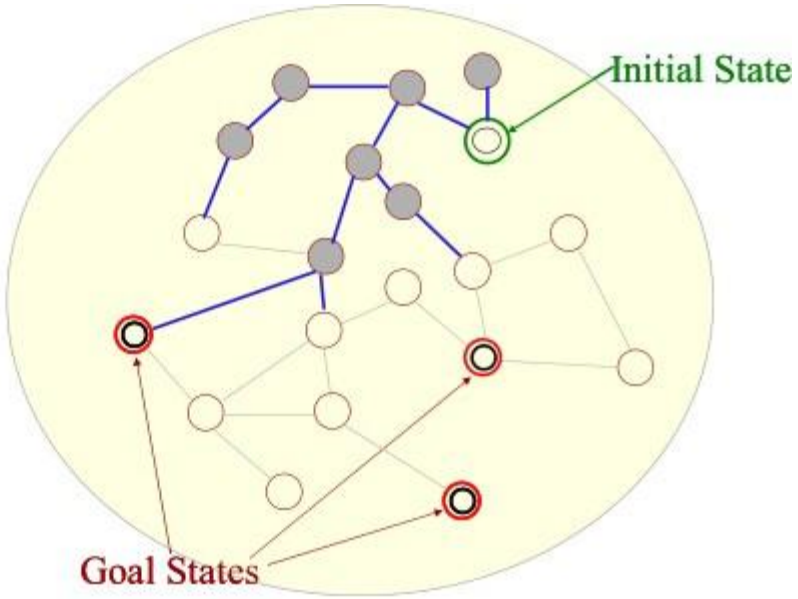
The successors of these states are picked and their successors are generated.

Illustration of Searching Process



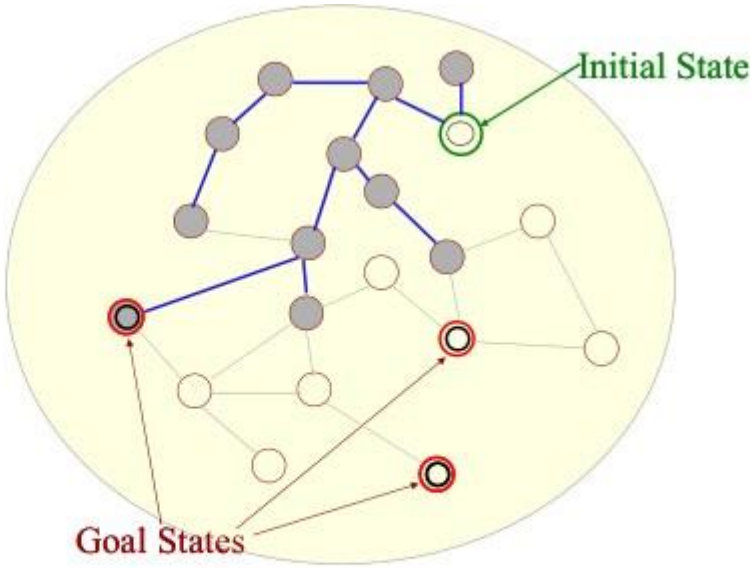
Successors of all these states are generated.

Illustration of Searching Process

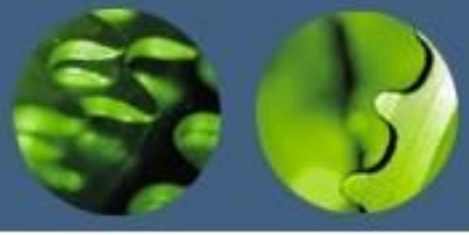


The successors are generated.

Illustration of Searching Process



A goal state has been found.



Well-defined problems

A problem can be defined formally by five components:

1. The initial state that the agent starts in.
2. A description of the possible actions available to the agent.
3. A description of what each action does
4. The goal test, which determines whether a given state is a goal state.
5. A path cost function that assigns a numeric cost to each path.

Example – 8 Puzzle

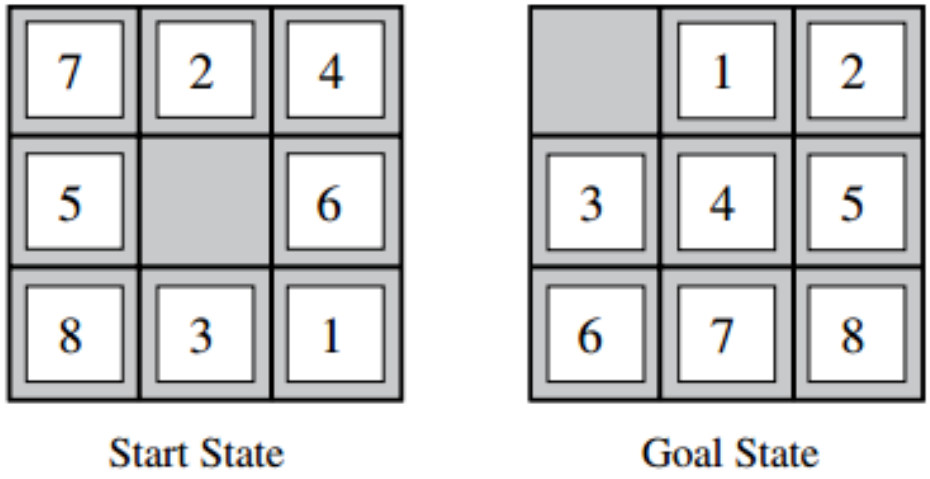
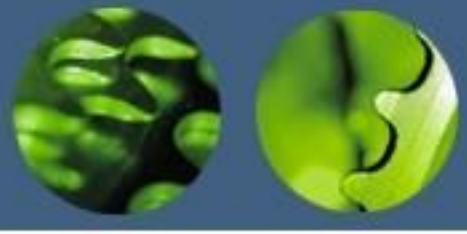


Figure 3.4 A typical instance of the 8-puzzle.



Example – 8 Puzzle

✓States: A state description specifies the location of each of the eight tiles and the blank in one of the nine squares.

✓ Initial state: Any state can be designated as the initial state.

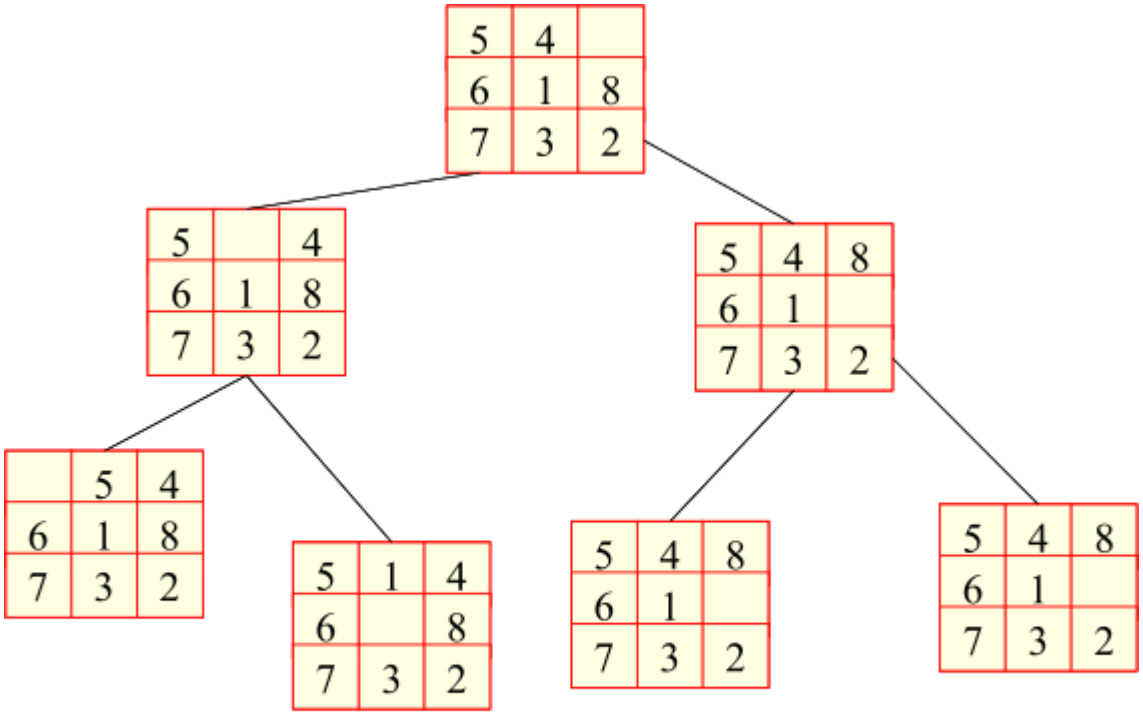
✓ Actions: The simplest formulation defines the actions as movements of the blank space Left, Right, Up, or Down. Different subsets of these are possible depending on where the blank is.

✓Transition model: Given a state and action, this returns the resulting state; for example, if we apply Left to the start state in Figure 3.4, the resulting state has the 5 and the blank Switched.

✓ Goal test: This checks whether the state matches the goal configuration shown in Figure 3.4. (Other goal configurations are possible.)

✓ Path cost: Each step costs 1, so the path cost is the number of steps in the path.

Example – 8 Puzzle



8-puzzle partial state space

Example – 8-Queens Problem

- ✓ The goal of the 8-queens problem is to place eight queens on a chessboard such that no queen attacks any other.
- ✓ (A queen attacks any piece in the same row, column or diagonal.)

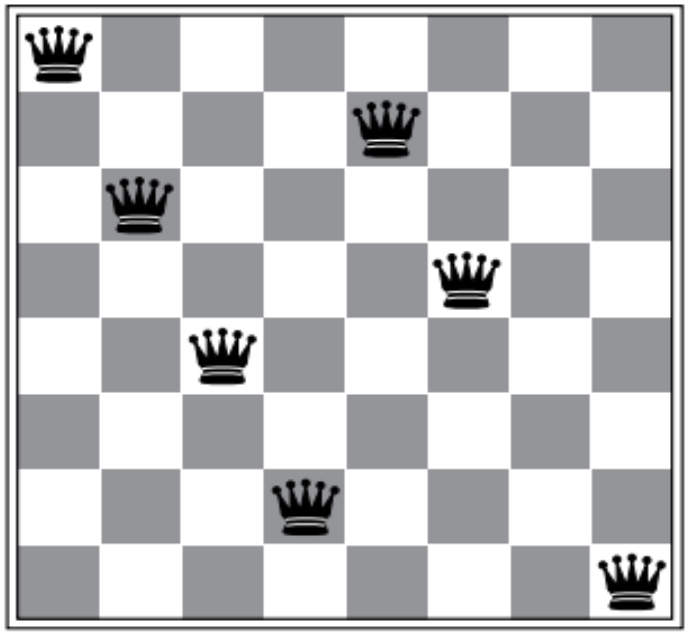
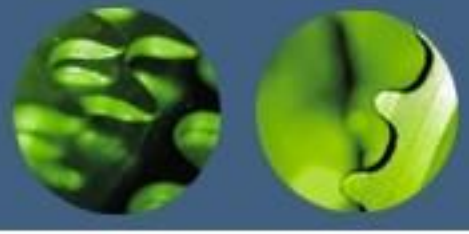
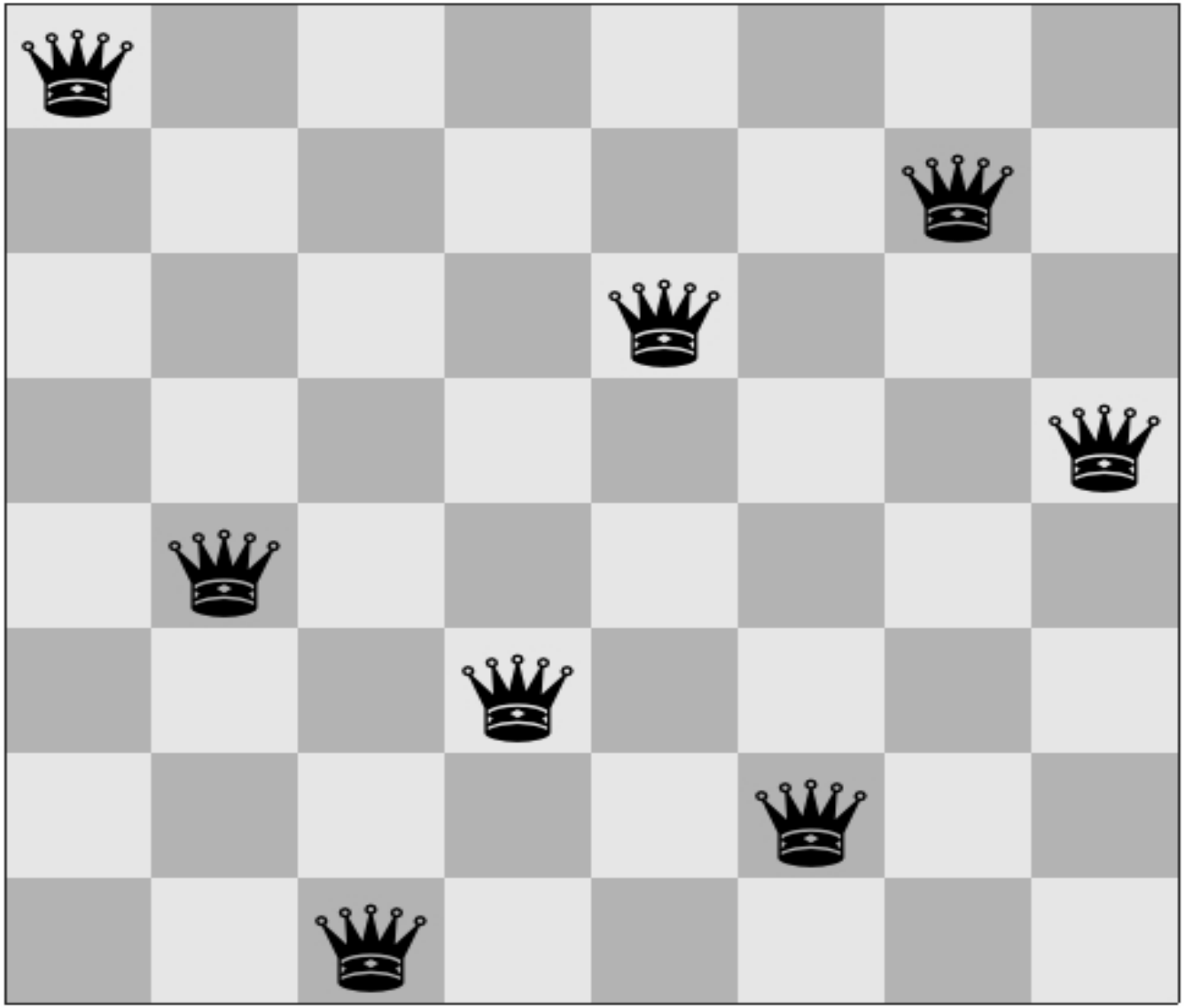


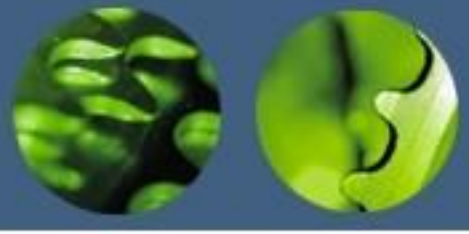
Figure 3.5 Almost a solution to the 8-queens problem. (Solution is left as an exercise.)



Example – 8-Queens Problem

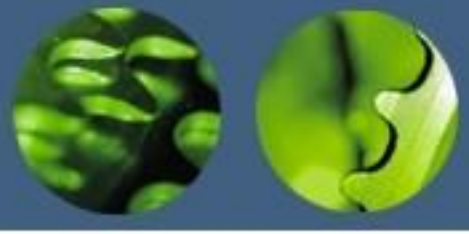
- ✓ States: Any arrangement of 0 to 8 queens on the board is a state.
- ✓ Initial state: No queens on the board.
- ✓ Actions: Add a queen to any empty square.
- ✓ Transition model: Returns the board with a queen added to the specified square.
- ✓ Goal test: 8 queens are on the board, none attacked.





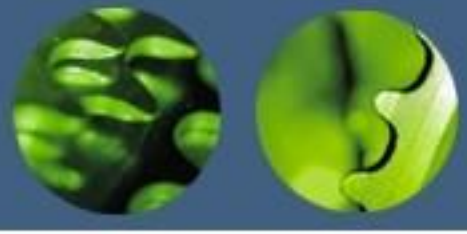
Solving Problems by Searching

- ✓ Searching the process finding the required states or nodes.
- ✓ Searching is to be performed through the state space.
- ✓ Search process is carried out by constructing a search tree.
- ✓ Search is a universal problem-solving technique.



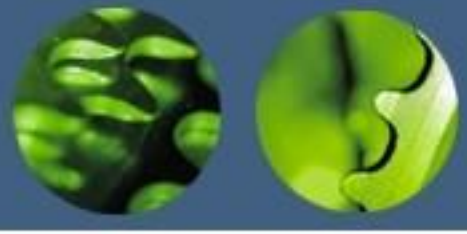
Solving Problems by Searching

- ✓ Search involves systematic trial and error exploration of alternative solutions.
- ✓ Useful when the sequence of actions required to solve a problem is not known
  - Path finding problems, e.g, eight puzzle, traveling salesman problem
  - Two player games, e.g., chess and tic tac toe
  - Constraint satisfaction problems, e.g., eight queens



Evaluating Search Strategies

- ✓ Completeness
  - – Guarantees finding a solution whenever one exists
- ✓ Time complexity
  - – How long (worst or average case) does it take to find a solution?
  - – Usually measured in terms of the number of nodes expanded
- ✓ Space complexity
  - – How much space is used by the algorithm?
  - – Usually measured in terms of the maximum size of the “nodes” list during the search
- ✓ Optimality/Admissibility
  - – If a solution is found, is it guaranteed to be an optimal one?
  - –That is, is it the one with minimum cost?



Uninformed vs. Informed Search

❖ **There are two types of search algorithms:**

✓ Uninformed search strategies

– Also known as "blind search," uninformed search strategies use no information about the likely "direction" of the goal node(s)

–Uninformed search methods: Breadth-first, depth-first, depth-limited, uniform-cost, iterative deepening, bidirectional search.

✓ Informed search strategies

– Also known as "heuristic search," informed search strategies use information about the domain to (try to) (usually) head in the general direction of the goal node(s)

–Informed search methods: Hill climbing, best-first, greedy search, A* search, Simulated Annealing search.

Chapter-3

Problem Solving by Searching

Date:
Page:

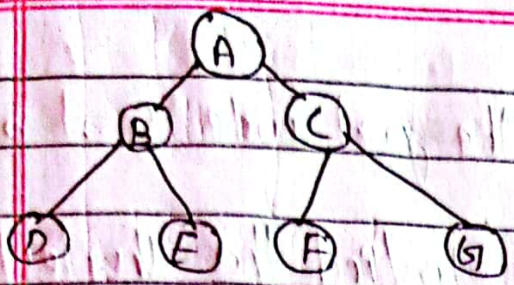
There, Uninformed Search → There are basically following types of uninformed search strategies or algorithms
(1) Breadth First Search (BFS)
(2) Depth First Search (DFS)
(3) Uniform Cost Search (UCS)
(4) Depth - Limited Search (DLS)
(5) Iterative Deepening Depth First Search (IDDFS)
Iterative Deepening Search (IDS)
(6) Bidirectional Search, etc.

Now, they are discussed below:

(1) Breadth first Search (BFS) →
→ Uninformed search technique or blind search technique
→ It maintains "queue" in order to store and retrieve total of nodes traversed i.e. maintains FIFO ordering
→ It expands the shallowest node.
→ Also called level search technique.
→ It works only on the present value but not on the heuristic or estimated values.
→ It maintains "Fringe" in order to maintain the total of nodes or states.
→ Generally, Fringe is the collection of nodes that are waiting for the to be expanded.

(A) Demonstration:

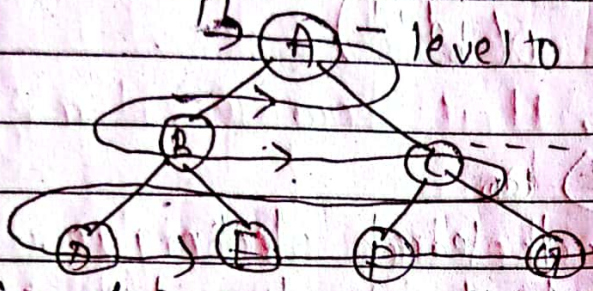
Date:
Page:



Find path from A to D.

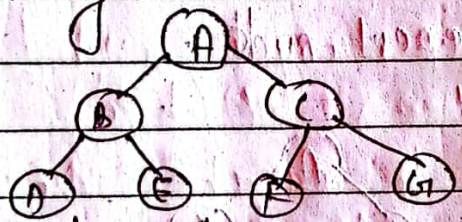
Solution:

⇒ In general approach, BFS works as follows:-



i.e. {A, B, C, D, E, F, G}

Now, for the given condition to find the path from A to D is given by:



(i) Is A the goal node?

⇒ No, to expand A to B and C. i.e. fringe = [B, C]

(ii) Is B, the goal node?

⇒ No, Now expand to D and E i.e. fringe = [C, D, E]

(iii) Is C the goal node?

⇒ No, expand the node C to F and G i.e. fringe = [D, E, F, G]

(iv) Is D the goal node?

⇒ Yes, Hence, the path to the goal node D is found.

(x) Properties of BFI

(i) Completeness → Complete if the goal node is at a finite depth.

(ii) Optimality → It is guaranteed to find the shortest path

(iii) Time complexity → $O(b^{d+1})$, where b = branch factor or successor, d = depth

(iv) Space complexity ⇒ $O(b^{d+1})$

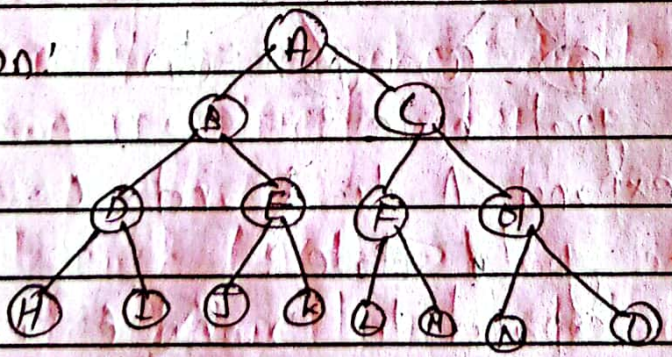
(2) Depth first search ⇒

→ Uninformed or Blind search technique.

→ It maintains stack to store and retrieve the states/nodes or node i.e. works on the principle of LIFO (last in first out).

→ It expands the deepest node.

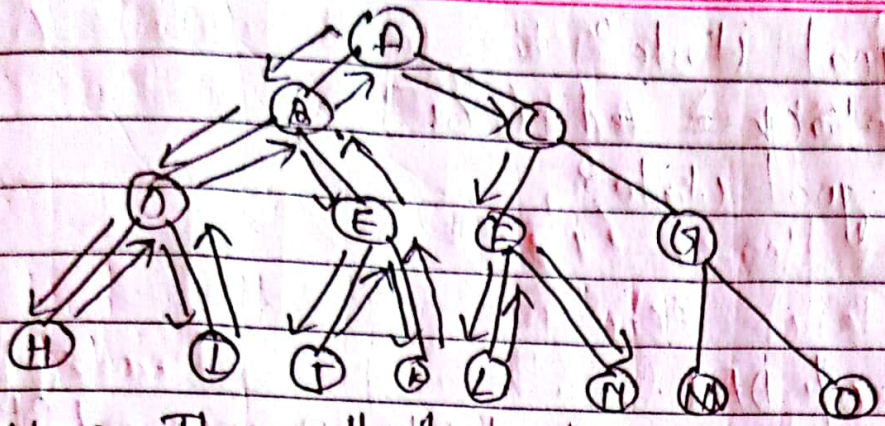
(xi) Demonstration:



Find the path from A to N.

⇒ The general solution to the DFI could be:

Date:
Page:



Now, The path from A to M is given by:

(i) Is A the goal state?
→ No, so, Stack = [B, C]

(ii) Is B a goal state?
→ No. So, Stack = [D, E, C]

(iii) Is D a goal state?
→ No. So Stack = [H, I, E, C]

(iv) Is H a goal state?
→ No. So Stack = [I, E, C]

(v) Is I a goal state?
→ No. So, Stack = [E, C]

(vi) Is E a goal state?
→ No. So, Stack = [J, K, F, C]

(vii) Is J a goal state?
→ No So Stack = [K, F, C]

(viii) Is K a goal state?
→ No, So Stack = [F, C]

(ix) Is C a goal state?
→ No. So Stack = [F, G]

(x) Is F a goal state?
→ No. So Stack = [L, M, G]

(x) Is L a goal state?
⇒ NO. Its stack = [L1, L2]

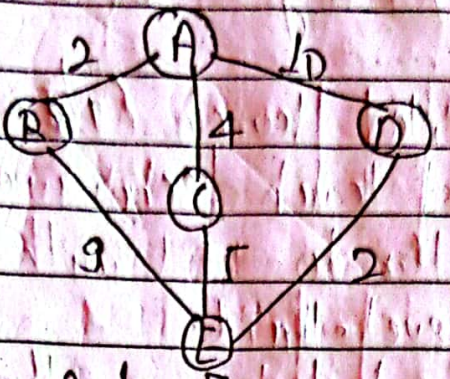
(xi) Is M a goal state?
⇒ Yes.

(#) Properties of DFS:
(i) Completeness → NO, it fails in case of depth or search space.
(ii) optimality → NO, It is because it may find non optimal goal first.
(iii) Time complexity → The time complexity of DFS is given by $O(b^m)$; where, b = branching factor and m = maximum depth.
It is more terrible if m is much larger than d, but solutions are dense, may be much faster than BFS.
(iv) Space complexity ⇒ $O(b, m)$. It is linear space i.e. we only need to remember single path along expanded unexplored nodes.

(3) Uniform cost search (UCS) →
⇒ BFS finds the shallowest goal but it is not always to find the optimal solution.
→ Uniform cost search can be used if the cost of travel from one node to another is available.
→ UCS always expands the lowest cost node on the fringe.
→ The first solution is guaranteed to be the cheapest because a cheaper node is expanded earlier and so will

have been found first.

Demonstration →



Find path from A to E.

→ Expand A to [B, C, D]

→ The path to B is the cheapest one with path cost 2.

→ So Expand B to E.

→ i.e. Total path cost = 2 + 9 = 11

→ This might not be the optimal solution, since the path

→ Since the path AC cost 4 (< 11)

→ Expand C to E. i.e.

Total path cost = 4 + 5 = 9

→ The path cost from A to D is 10 (> 9).

→ Hence, the optimal path is A-C-E with total path cost of 9 units.

* Properties of Uniform cost search

(i) Completeness ⇒ Complete, if the cost of every step is greater than or equal to some small positive constant "e".

(ii) Optimality ⇒ optimal, if the cost of every step is greater than or equal to some positive constant "e".

(iii) Time complexity ⇒ $O(b^{c^*/e})$, where c^* is the cost of optimal path,

Date:
Page:

e is the small positive constant

(iv) Space complexity ⇒ $O(b^{1+\epsilon})$

(4) Depth-Limited Search (DLS) ⇒

→ Uninformed search technique.

→ It is similar to DFS with a pre-determined depth limit or level limit.

→ In this search technique, the node at the depth limit will be treated as it has no successor nodes further.

→ DLS can solve the drawback of the infinite path in DFS.

→ DLS can be terminated with following two conditional of failure:-

(i) Standard failure value → It indicates that problem does not have any solution.

(ii) Cut-off failure value → It defines no solution for the problem within given depth limit.

* Advantages

→ It is memory efficient

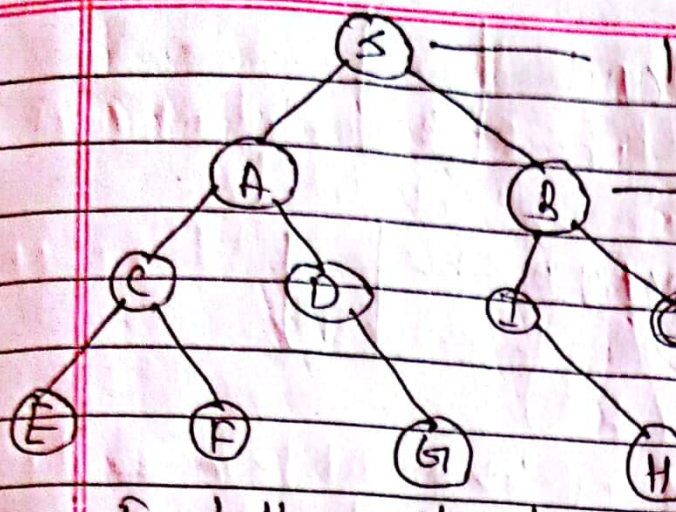
* Disadvantages

→ Incomplete (if limit l is less than depth)
i.e. limit (l) < depth (d)

→ It may not be optimal if the problem has more than one solution. (if $l > d$)

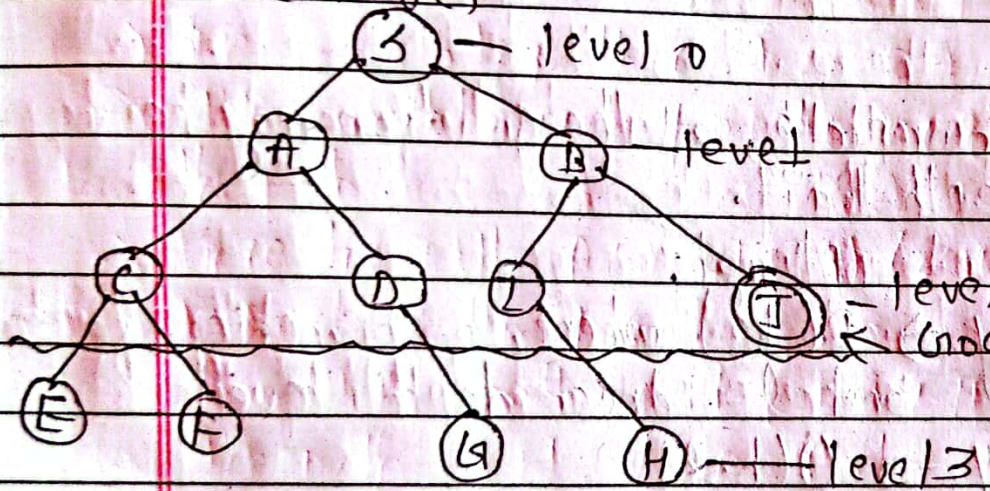
* Demonstration

Date:
Page:



Find the path from S to J. Note that "J" is the goal state with depth limit 2.

⇒ Now,
we have,



→ Here, S is assumed to be the starting node also given that the depth limit is 2. Here, the goal node J is possible to found at Depth limit search is similar to DFS with depth limit. "J" So, by using stack, we can find the goal state as follows:-

i.e. S = | A | B | ⇒ SA = | C | D | B |

Here we reached the depth limit

Then, [A,C] = | D | B | ⇒ [A,C,D] = | | B |

Again,
[A,C,D,B] = | I | I | ⇒ [A,C,D,B,I] = | | J |

Finally,
[A,C,D,B,J,I] = | | |

Here, goal state "I" is found using Depth-limited search technique.

* Properties:
(i) Completeness → Incomplete, all the solution may be beyond the specified depth limit.
(ii) Optimality → Not optimal.
(iii) Time complexity → $O(b^l)$ where, b = branching factor and l = tree depth level
(iv) Space complexity → $O(b \cdot l)$

(v) Iterative Deepening Search (IDS) or, Iterative Deepening Depth First Search (IDDFS) → Uniform search technique.
→ Combination of both DFS and BFS.
→ It is implemented as DFS followed by BFS.
→ Best depth limit is found at by gradually increasing the limit.
→ It combines the advantages of BFS and DFS strategies.

Date:
Page:

taking the completness and optimality of BFS, and the modest memory requirements of DFS.

→ Initially, depth (d) = 0; and if goal node or solution is not found out, then it increase every iteration by "1".

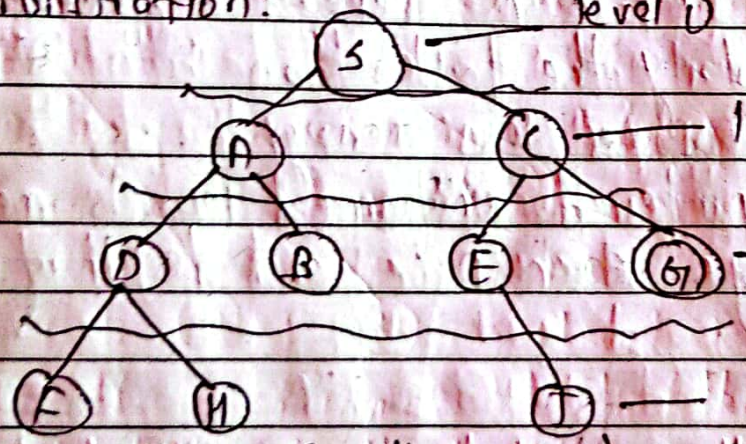
(*) Advantages

(i) Incorporates the benefits of both BFS and DFS i.e. optimality and completness of BFS and the modest memory requirements (ie. less memory) of DFS.

(*) Disadvantages

→ Repeat the work of previous.

(*) Demonstration:



Here, "G" is the goal node.

→ Now,

IDS is implemented as follows:-

1st iteration, d = 0 [S]

2nd iteration, d = 0+1 = 1 [S → A → C]

3rd iteration, d = 1+1 = 2 [S → A → D → B → C → E → G]

∴ Hence, the goal node "G" is found.

Date:
Page:

(*) Properties of IDS

(i) Completeness → IDS is like BFS, it is complete when the branching factor "b" is finite.

(ii) Optimality → IDS is like BFS, it is optimal when the steps are of the same costs.

(iii) Time complexity → $O(b^d)$

(iv) Space complexity → IDS is like DFS in its space complexity, taking $O(b \cdot d)$ of memory.

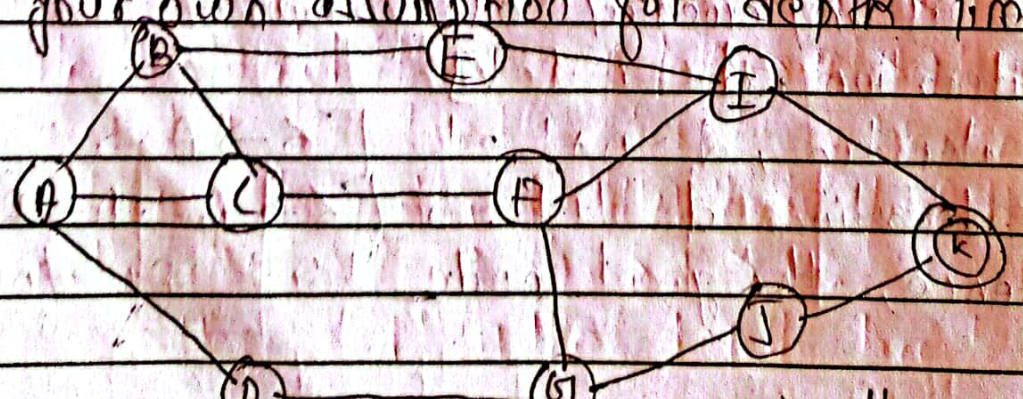
* Note:

→ BFS generates some nodes at depth "d+1", where IDS doesn't. The result is that IDS is actually faster than BFS, despite the repeated generation of nodes.

→ IDS is particularly the faster than both BFS and DFS because IDS consumes less memory.

→ IDS is the preferred uninformed search method when there is a large search space and depth of the solution is not known.

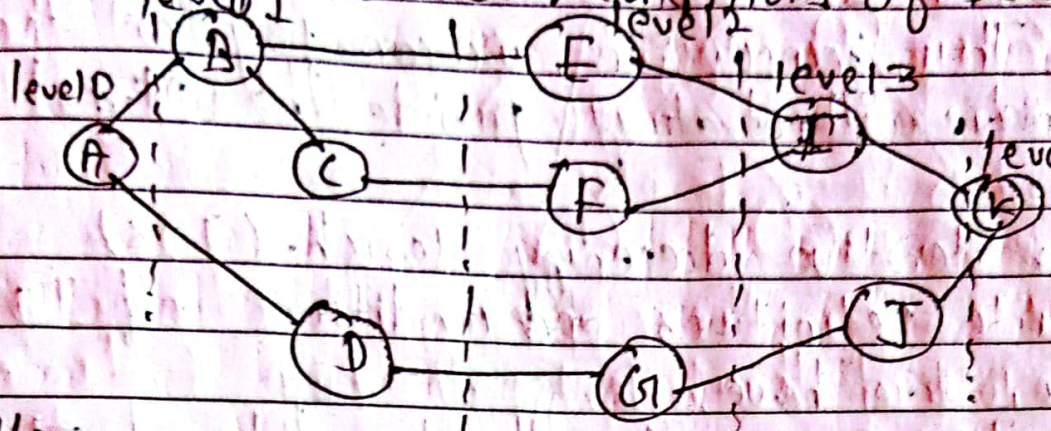
2076
Q (##) Given the following state space illustrate how depth limited search and iterative deepening search works. Use your own assumption for depth limit.



Here, A is the start and K is the goal.

→ solution

(a) For Depth limited search:
(write descriptions and conditions of DLS)



Here,
"A" is assumed to be starting node and "K" be the goal node.
Let, depth limit be 4. Here, we find our goal node "K" because DLS is similar to DFS with depth limit. So, by using stack, we can find the goal node "K", i.e.

A ⇒ | B |
| C |
| D |

, AB ⇒ | E |
| C |
| D |

, ABE ⇒ | I |
| C |
| D |

, ABEI ⇒ | K |
| C |
| D |

Thus, ABEIK ⇒ | C |
| D |

Hence, by using DLS search technique, we found a path to reach the goal node and the path is:
A → B → E → I → K.

But, if we let the depth limit to be "three" then we cannot reach the goal node "K" because goal node "K" is on the level "4".

Thus, In conclusion, we can say that the DLS technique will not always provide the optimal solution.

(ii) For Iterative deepening search. (IDS)
(With definitions and condition of IDS)

Now, we have,

IDS incorporates the benefits of both DFS and BFS. So, applying this condition on the search space or state space, we get:

We know that, In IDS, best depth limit can be found out by gradually increasing the depth limit.

So, for the given search space, we get:

1st iteration, d = 0 [A]

2nd iteration, d = 0+1 = 1 [A → B → C → D]

3rd iteration, d = 1+1 = 2 [A → B → [redacted] E → C → A → D →

4th iteration, d = 2+1 = 3 [A → B → E → I → C → F → D → G → I]

5th iteration, d = 3+1 = 4 [A → B → E → I → K]

Hence, the goal node "K" is found and the optimal path is "A → B → E → I → K".

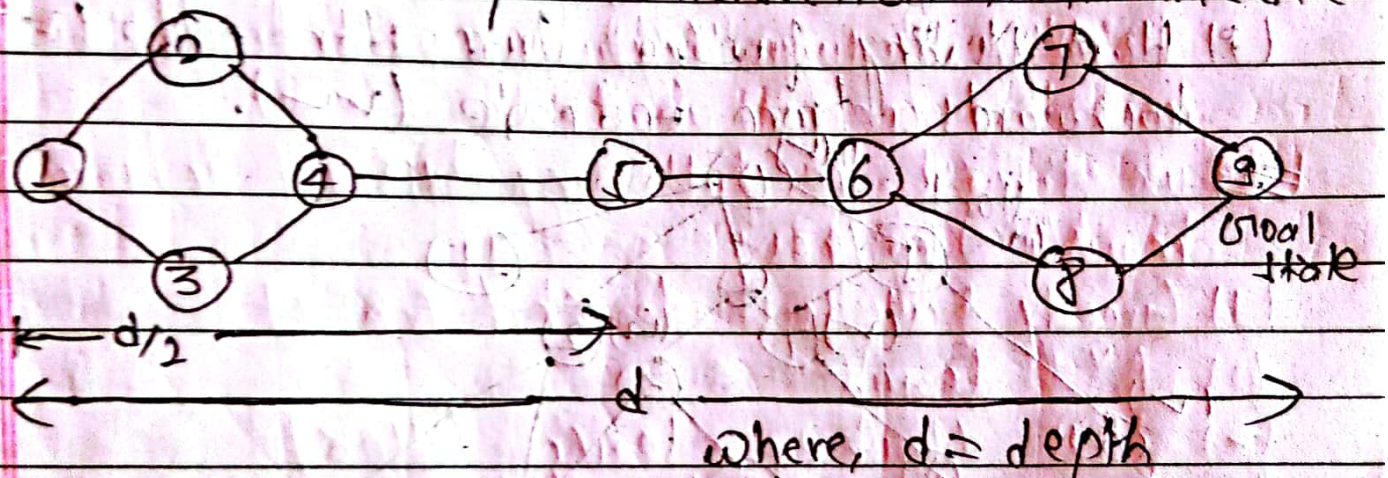
Here, "d" is the depth limit and IDS

Date:
Page:

implemented as DFS followed by BFS.
Hence, IDS provided the optimal solution.

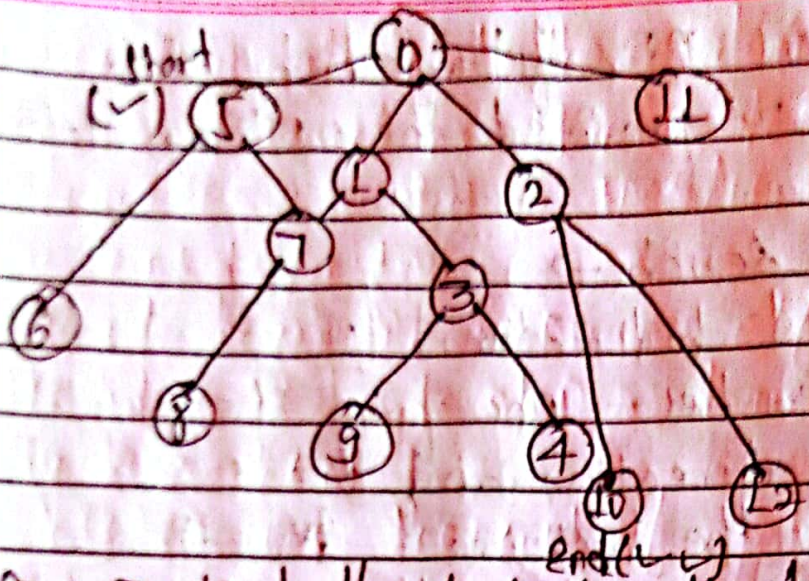
(6) Bidirectional search ⇒

→ The idea behind bidirectional search is to run two simultaneous searches — one from the initial state in the forward direction and the other from the goal state in the backward direction — hoping that the two searches meet in the middle.
→ It means that the path concatenation of the required solution i.e. the path intersection in the middle.



(*) When to use bidirectional search?
⇒ We can considered bidirectional search approach when -
→ Both initial and goal state are unique and completely defined.
→ The branching factor is exactly the same in both the direction i.e. forward and backward.

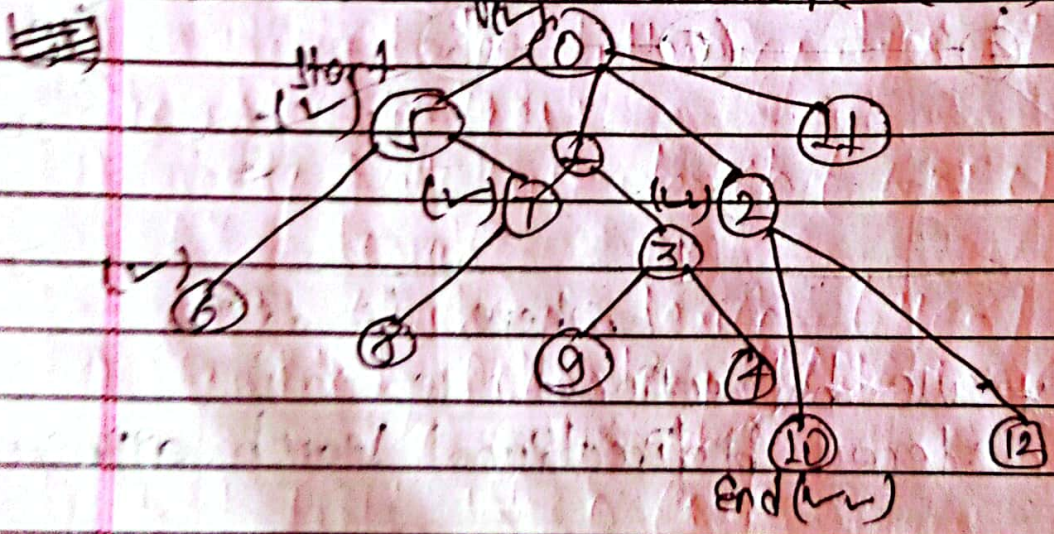
(*) Demonstration:



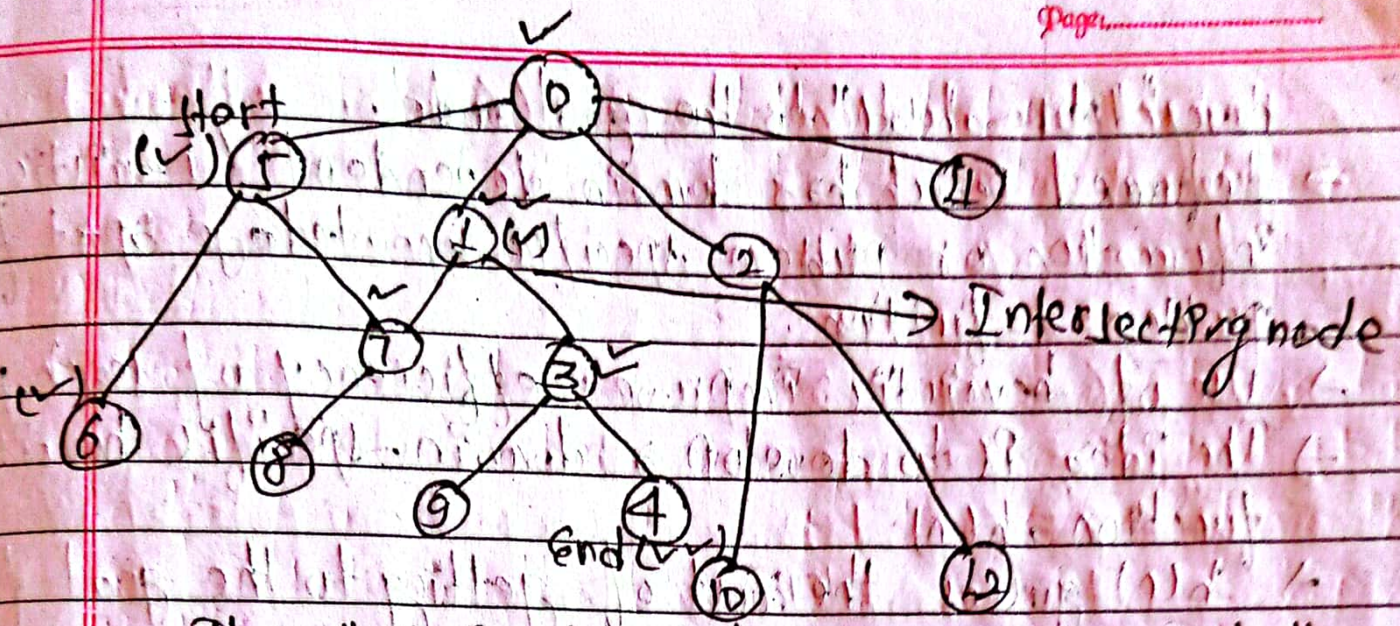
Aim: To find the shortest path from 5 to 4 using bidirectional search.

→ Do BFS from both direction

(i) Start moving forward from start node (5) and backward from end node (4).



(ii) Similar to BFS, at every point explore the next level of nodes till we find an intersecting node, and on finding the intersecting node, trace back to find the path.



Thus "1" is the intersecting node and the path is also found which is I → 0 → 1 → 3 → 4.

(*) Properties / Performance Measures of Bidirectional search.

(i) Completeness → Yes, bidirectional search is complete when we use BFS in both searches, the search that starts from the initial state and the other from the goal state. But using DFS, this may not be achieved.

(ii) Optimality → Bidirectional search is optimal when BFS is used for search and paths are of uniform costs - all steps of same cost. But using DFS, this may not be achieved.

(iii) Time complexity → $O(b^{d/2})$

(iv) Space complexity → $O(b^{d/2})$

(#) Informed search (Heuristic search)

→ In uninformed search, we don't try to evaluate which of the nodes on the frontier are the most promising. We

never "look-ahead" to the goal node.

→ Informed search has domain dependent (heuristic) information or problem specific knowledge apart from problem definition.

→ Use of heuristic improves efficiency of search.

→ The idea is to develop a domain-specific heuristic function, $h(n)$

→ $h(n)$ guides the cost of getting to the goal from node "n".

→ Thus, we can say that the heuristic or informed search guides the search process in the most profitable direction by suggesting which path to follow first when more than one is available.

→ We discuss the following type of informed or heuristic searches:

(1) Greedy Best First search (GBFS)
(2) A* search
(3) Hill Climbing search (HCS)
(4) Simulated Annealing (SA) search.

(i) Best First search →

→ In this strategy, the node whose state is judged to be the closest to the goal state is expanded first.

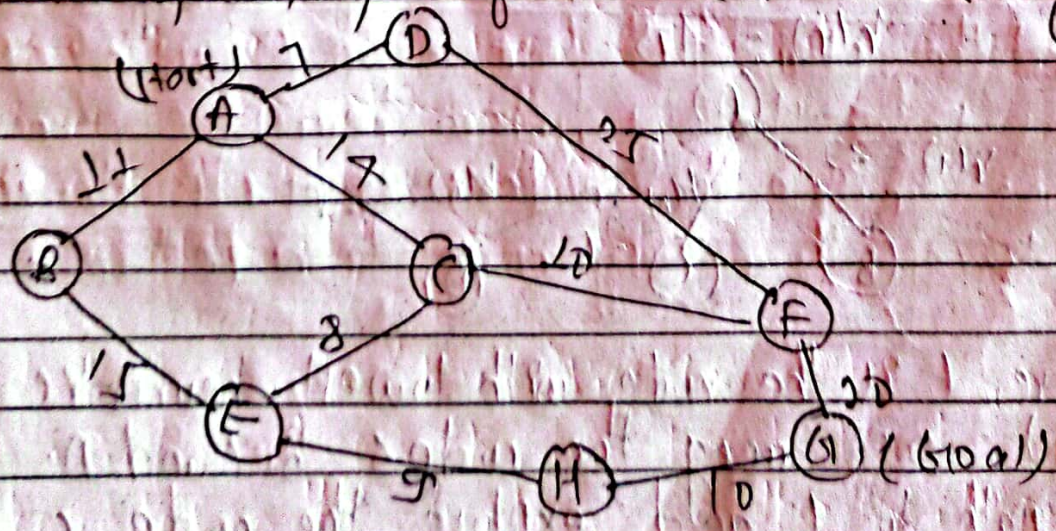
→ It evaluates the nodes by using just the heuristic function, $h(n)$. Hence, in this case, evaluation function $f(n) = h(n)$

[i.e. $f(n) = h(n)$]

→ h(n) = 0, if "n" is the goal.
→ The evaluation function must represent some estimates of the cost of the path from state to the closest goal state.
→ The heuristic function is the estimated cost of the cheapest path from node to the goal.
→ The best first search can further be divided into two major types :- (1) Greedy Best First Search
(2) A* Search

(I) Greedy Best First Search →
→ In this search strategies, the node whose state is just to be the closest to the goal state is expanded first.
→ It evaluates the nodes by just using the heuristic function i.e. f(n) = h(n)
h(n) = 0 if "n" is the goal.
→ One example of heuristic function may be the straight line distance to the goal in the route finding problem.

(#) Demonstration:
Find the optimal path from node A to G using GBFS.



Date:
Page:

Straight line distances to node (G) from other nodes given below.

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

⇒ Solution:

Let h(n) be the straight line distances from a vertex to the goal (G).

Now, we have

(i) for "A".

f(n) = h(n)

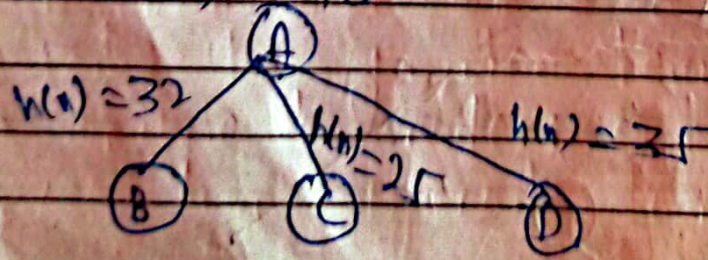
i.e. f(n) = 40

i.e. h(n) = 40

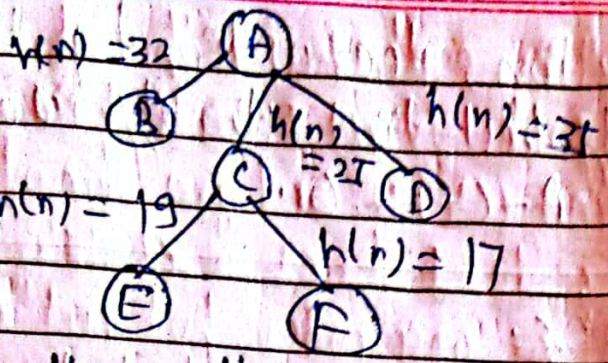
(A)

(ii) Now, we explore node "A" as follows:

h(n) = 40

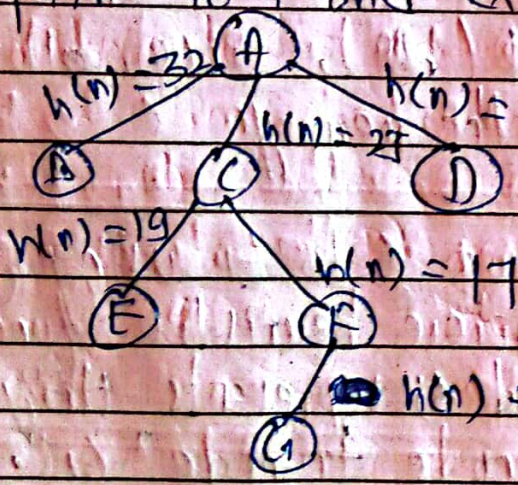


(iii) Here, C is the node with lowest heuristic function h(n) = 25 among the nodes B, C and D. So, we choose the path to C and explore the node C as follows



Here, the node "C" is expanded towards the node "A" as well. But we discards the path from C to A because we have already a path from A to C, and we do not explore the paths repeatedly. i.e. node "A" ~~becomes~~ form ~~a forest~~ CLOSED path. It means that we only explore the open nodes but not the "CLOSED" ones.

(Qu) Here F is the node with lowest heuristic function i.e. $h(n) = 17$ among the nodes E and F. So we check the path to F and explore the node as follows.



Here, the node "F" is expanded towards node "D" as well. But we discards this path from F to D.

Date:
Page:

Thus, in conclusion, the optimal path from node A to G using greedy Best First strategy on the above given search space is A-C-F-G.

* Properties of GBFS:

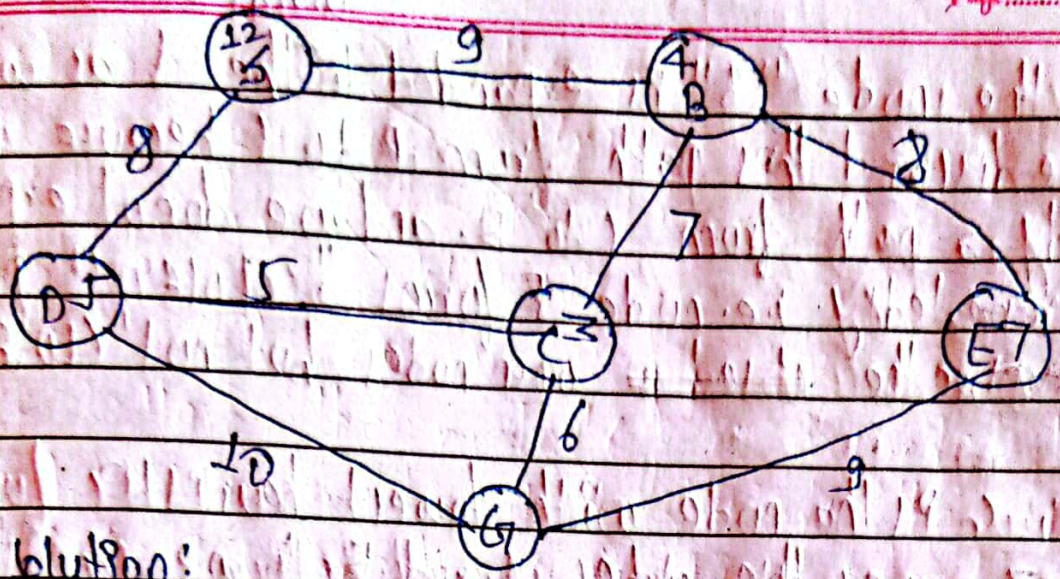
(I) Completeness ⇒ This algorithm can start down on a path and never return to any other possibility i.e. can get stuck in loops. Hence, this algorithm is not complete.

(II) Optimality ⇒ This algorithm looks for the nearest best search choice and doesn't make a careful analysis of the long-term options. Hence, it may give longer solution. Even if a shorter solution exists. Thus, this algorithm is not optimal.

(III) Time complexity ⇒ $O(b^m)$, where, m = maximum depth of the search space.

(IV) Space complexity ⇒ $O(b^m)$, since, all the nodes have to be kept in the memory.

(*) Find the optimal path from S to G using greedy Best First search strategy; where heuristic function h(n) is given alongside the nodes; represents straight line distances from any node to node "G".



⇒ Solution:

Let $h(n)$ be the straight line distance from any vertex to the goal (G).

Now, we have

(i) For "S"

$f(n) = h(n)$

i.e. $f(n) = 12$

(S)

(ii) Now, we have explore node "S" as follows:

$h(n) = 12$

S

$h(n) = 7$ $h(n) = 4$

D B

(iii) Here, B is the node with lowest heuristic function i.e. $h(n) = 4$ among the nodes D and B. Now choose the path to B and explore B as follows

$h(n) = 7$ S $h(n) = 4$

D B $h(n) = 7$

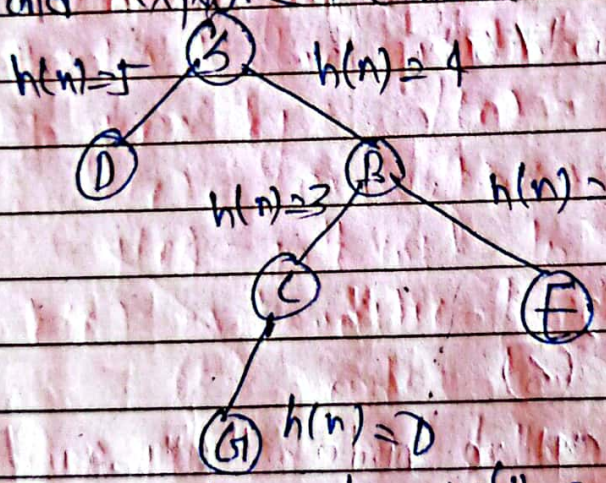
$h(n) = 3$

C E

towards

Here, the node "B" is expanded towards I as well. But we discard this path from B to I because we have already a path from C to B. and we don't explore the path repeatedly. i.e. node I forms a "CLOSED" path. We only explore the ~~visited~~ open nodes but not "CLOSED" one.

Q) Here, C is the node with lowest heuristic function h(n) = 3 among the nodes C and E. So we choose path to C and explore the node as follows.



Here, the node "G" is expanded towards B as well. But we discard the path from C to B.

Thus inconclusion, the optimal path from I to G using GBFS strategy on the above given search tree is I - B - C - G. #

Date:
Page:

(v) A* search (Read as A-star) → A* is a Best First, Informed Graph search Algorithm.

→ It is an admissible heuristic search; it means that A* search gives the guarantee of optimal solution.

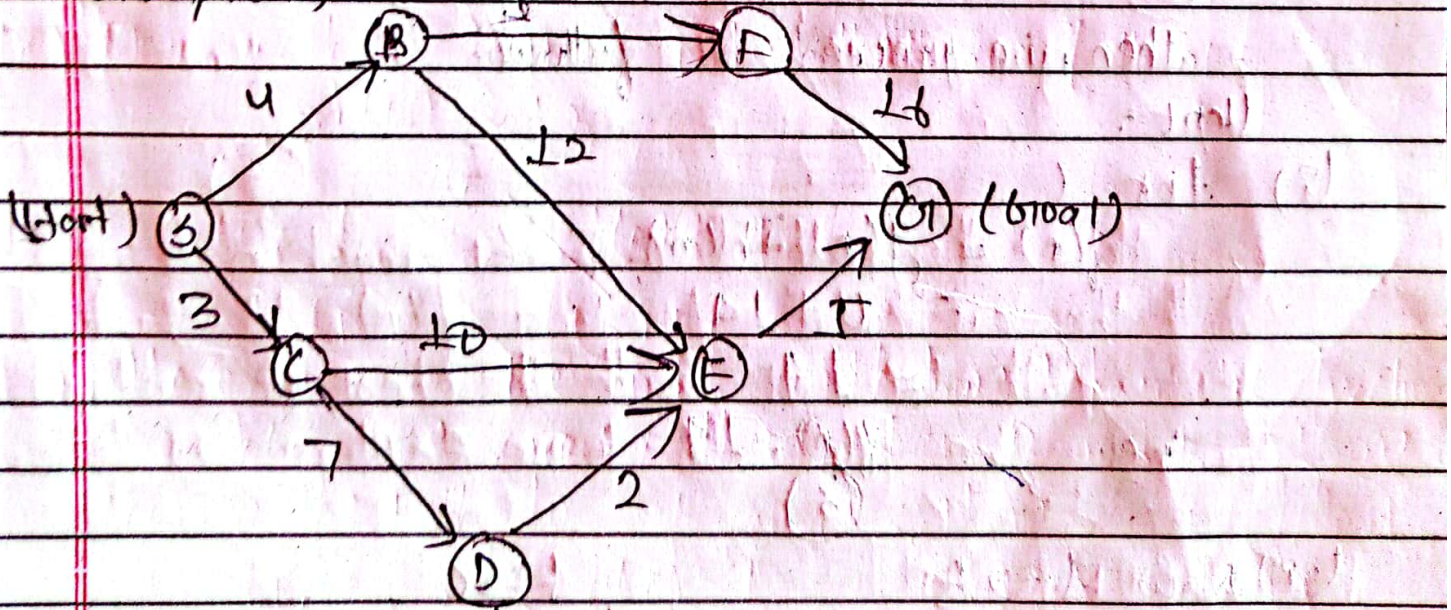
→ This algorithm is a Best First search Algorithm which evaluates nodes by combining $g(n)$, the cost to reach the node, and $h(n)$, the cost to get the goal node from any other node. Thus, the evaluation function $f(n)$ is given by:

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

Here, $f(n)$ is the estimated cost of the cheapest solution through "n".

(*) Demonstration:

Example (1)



Find the optimal path from the node S to G1 using A* Best First search algorithm. The straight line distances to the goal node "G1" from other nodes are given by

Date:
Page:

A → G = 14
B → G = 12
C → G = 11
D → G = 6
E → G = 4
F → G = 11

⇒ Solution

h(n) be the heuristic function which represents straight line distances from other node to goal node "G"

The evaluation function f(n), in case of A* search is given by: f(n) = g(n) + h(n)

where, f(n) = evaluation function
g(n) = path cost to a node given in the graph
h(n) = heuristic function ie. cost from other nodes to goal node "G".

Then, we proceed as follows:

Step 1:

(i) for A

f(n) = g(n) + h(n)
= 0 + 14
= 14

i.e. f(n) = 14 (S)

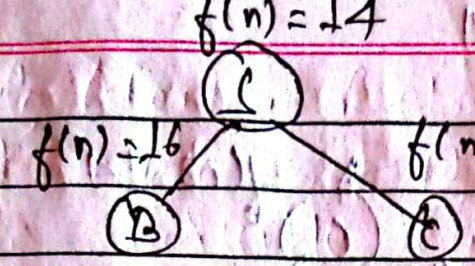
(ii) Now A → B

f(n) = 4 + 12
= 16

A → C

f(n) = 3 + 11
= 14

Date:
Page:



i.e.

Here, we choose the path "I-C : Pune", C "has got the cheapest evaluation function's value among B and C i.e. f(n) = 14

(iii) Then,

I C → D

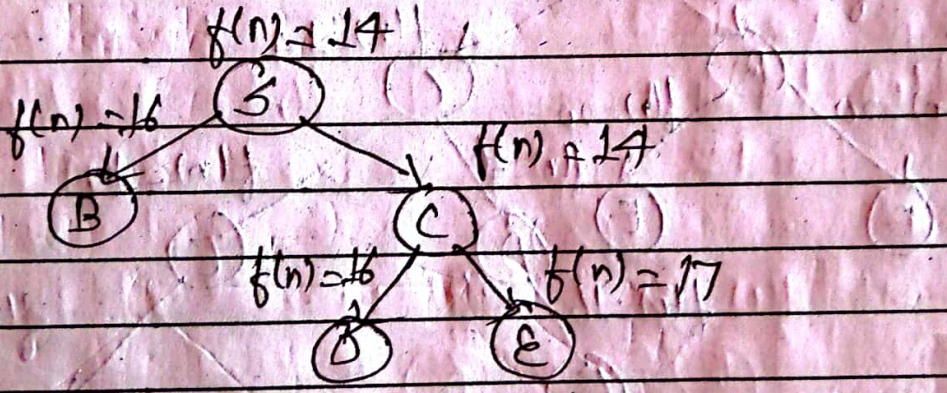
$\Rightarrow f(n) = 3+7+6$

$= 16$

And I C → E

$f(n) = 3+10+4$

$= 17$



Here, we reject the path I C → E and select one of the paths among I B to B and I C to D.

(iv) Now, I B → F

$\Rightarrow f(n) = 4+7+11$

$= 9+11$

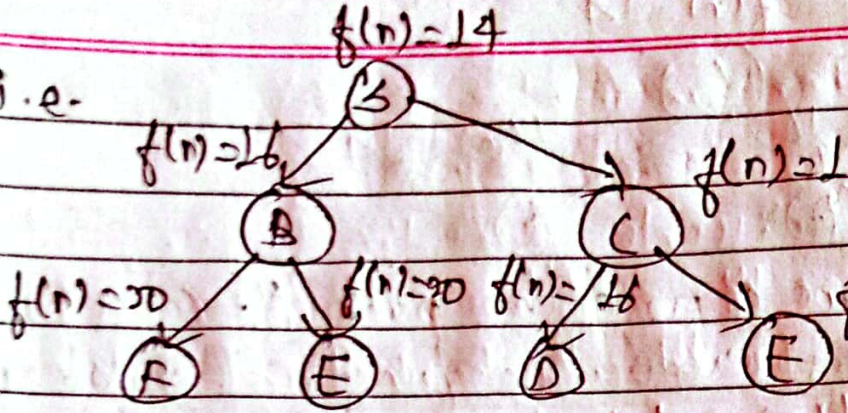
$= 20$

And, I B → E

$f(n) = 4+12+4$

$= 20$

Date:
Page:



(v) Then,

I(D→E

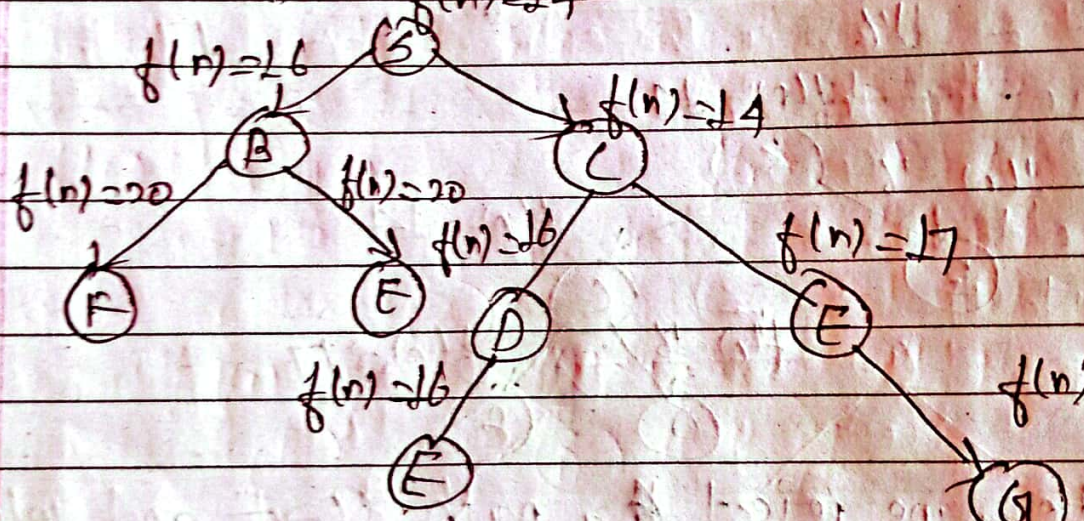
f(n)=3+7+2+4

=16

And I(E→G

f(n)=3+10+5+0

=18

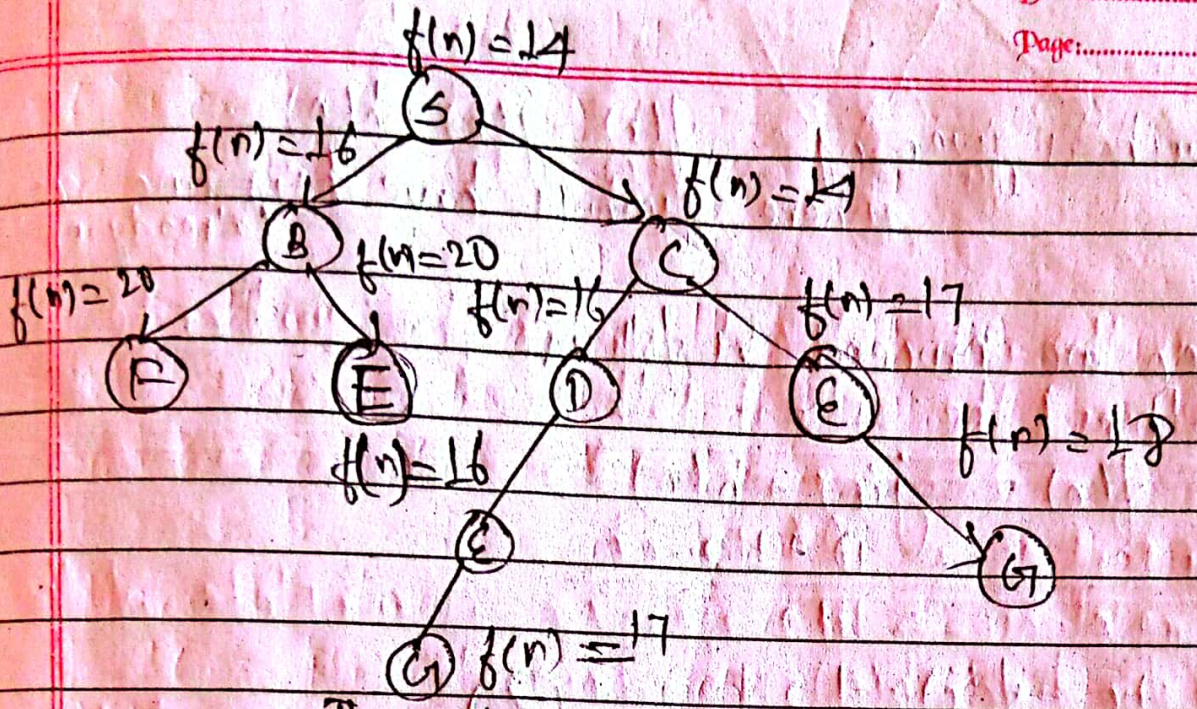


Here, the path I→C→E→G reached the goal node with the evaluation function f(n)=18. But there is still a path I→C→D→E with the cheapest evaluation function value f(n)=16. So we explore at fallback

(vi) I(DE→G

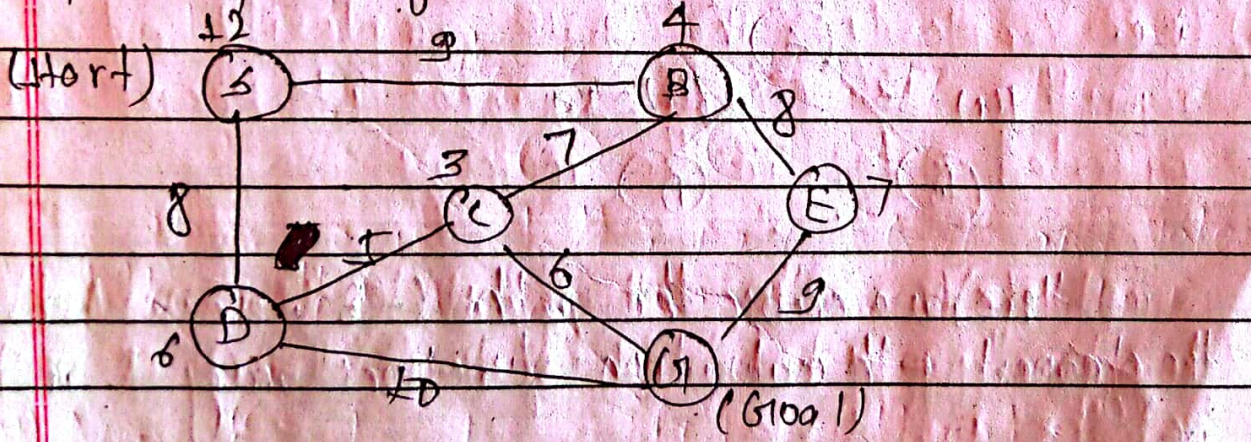
f(n)=3+7+2+5+0 =17

i.e.



Thus, the optimal cost to reach the goal node is 17 units and the optimal path is $S \Rightarrow C \Rightarrow D \Rightarrow E \Rightarrow G_1$

Example - 2: Perform A* search strategy to find the optimal path from S to G_1.



$S \Rightarrow G_1 = 12$, $B - G_1 = 4$, $D \Rightarrow G_1 = 6$, $E \Rightarrow G_1 = 7$,
$C \Rightarrow G_1 = 3$, $G_1 \Rightarrow G_1 = 0$

⇒ Solution:

$h(n)$ be the heuristic function which represent straight line distances from other node to goal node.

The evaluation function $f(n)$ is sum of A*

Date:
Page:

Search is given by: $f(n) = g(n) + h(n)$
where, symbol have their usual meanings already written.

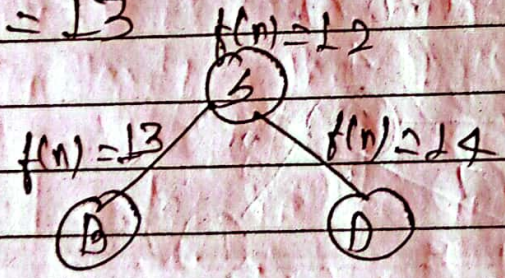
Then we proceed as follows:

(i) For 1,
$f(n) = g(n) + h(n)$
$= 0 + 12$
$= 12$
i.e. $f(n) = 12$ (1)

(ii) Now 1 → B
$f(n) = 9 + 4$
$= 13$

1 → D
$f(n) = 8 + 6$
$= 14$

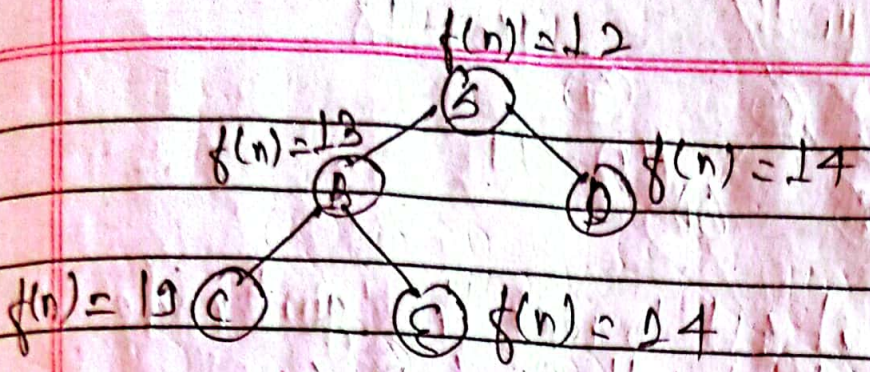
i.e.



Here, we choose the path 1–B. Since B has got the cheapest path evaluation functions among B and D
i.e $f(n) = 13$

(iii) Then, 1B → C
$f(n) = 9 + 7 + 3$
$= 9 + 10$
$= 19$

1B → E
$f(n) = 9 + 8 + 7$
$= 9 + 15$
$= 24$



Here, we reject the paths $B \to C$ and $B \to E$ and select $1-D$.

(iv) Then,

$1D \to C$

$f(n) = 8+5+3$

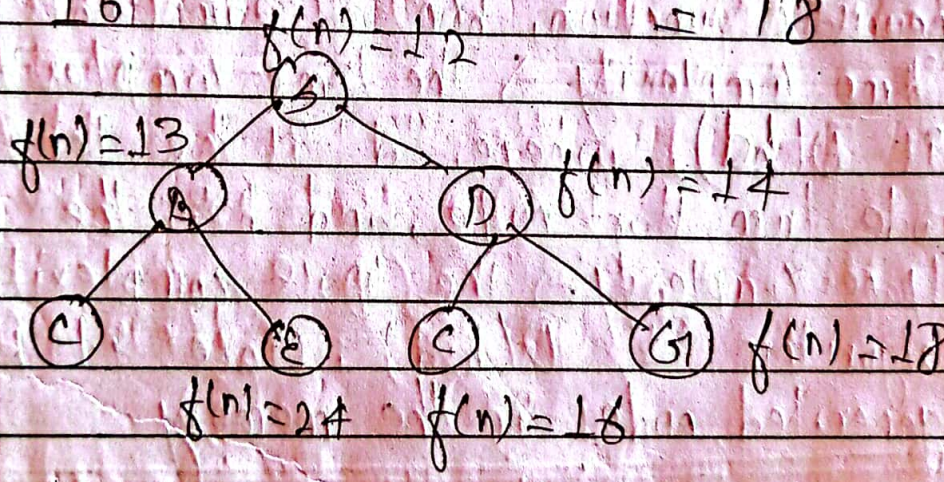
$= 16$

$1D \to G_1$

$f(n) = 8+10+0$

$= 18$

i.e.



Here we reject the path $1D \to G_1$ but we select a path $1-D-C$ because, $f(n)=18 > f(n)=16$

Then

$1DC \to G_1$

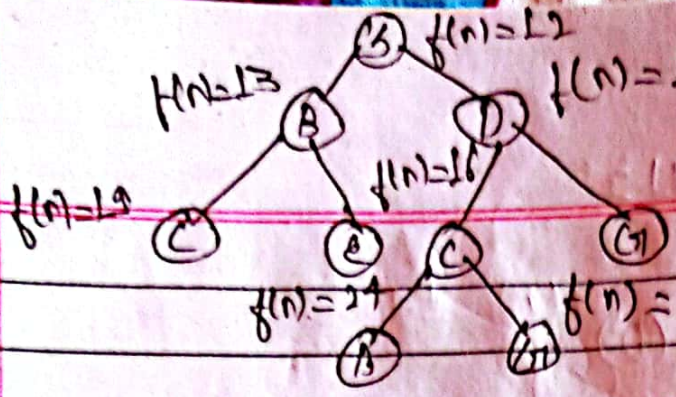
$f(n) = 8+5+6+0$

$= 14+5 = 19$

$1DC \to B$

$f(n) = 8+5+7+4$

$= 13+7+4 = 24$



Thus, the optimal cost to reach the goal node is 19 units and the optimal path is A-D-G1

(*) Properties of A* search.

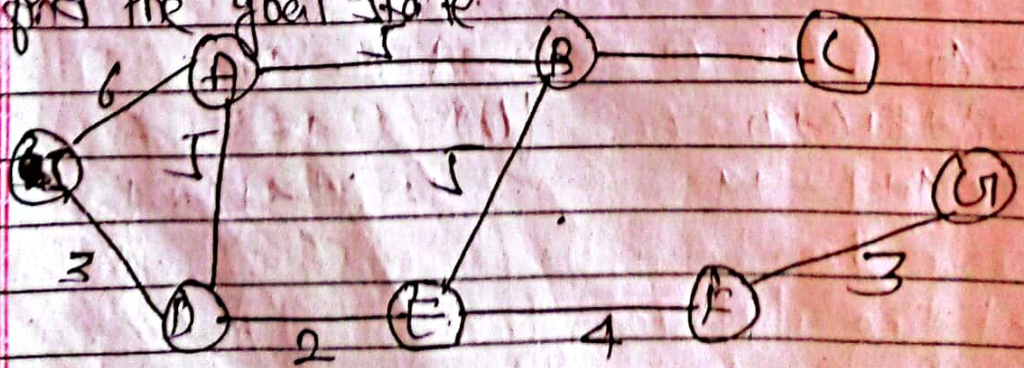
(i) Completeness → This algorithm is complete if the heuristic function, h(n) is admissible.

(ii) Optimality → This algorithm is optimal if heuristic function, h(n) is admissible.

(iii) Time complexity → The time complexity of this algorithm is O(b^d), where, "d" is the length of the node from start node, and b is the branching factor.

(iv) Space complexity → The space complexity of this algorithm is O(b·d), since it keeps all the generated nodes in the memory.

Example: (3) Given the following state space representation, show how greedy best first and A* search is used to find the goal state.



Date:
Page:

(S) the start node and (G) the goal node. The heuristics of the states are $h(S)=12$, $h(A)=8$, $h(B)=7$, $h(C)=5$, $h(D)=9$, $h(E)=4$, $h(F)=2$, $h(G)=0$.

⇒ Solution

For A* search

$h(n)$ be the heuristic function which represent straight line distances from other node to goal node.

The evaluation function $f(n)$ in role of A* search is given by $f(n) = g(n) + h(n)$

Then we proceed as follows:-

(i) For S

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

$= 0 + 12 = 12$

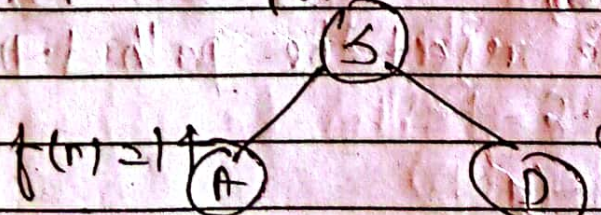
$f(n) = 12$ (S)

(ii) Now, $S \to A$ and $S \to D$

$f(n) = 6 + 8$ $f(n) = 3 + 9$

$= 14$ $= 12$

i.e.

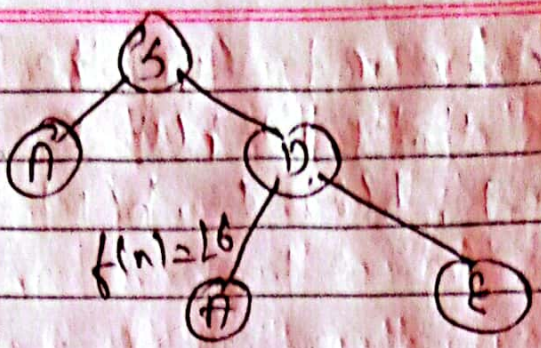


Here, we choose the path $S \to D$. Since D has got the cheapest evaluation function among A and D, i.e. $f(n) = 12$

(iii) Then $D \to A$ $D \to E$

$f(n) = 3 + 5 + 8$ $f(n) = 3 + 2 + 4$

$= 16$ $= 9$



Here we reject a path S-D→A and we select S-D-E

(iv) Then,

S-D-E-B

$f(n) = 3+2+7+7$

$= 7+7+7$

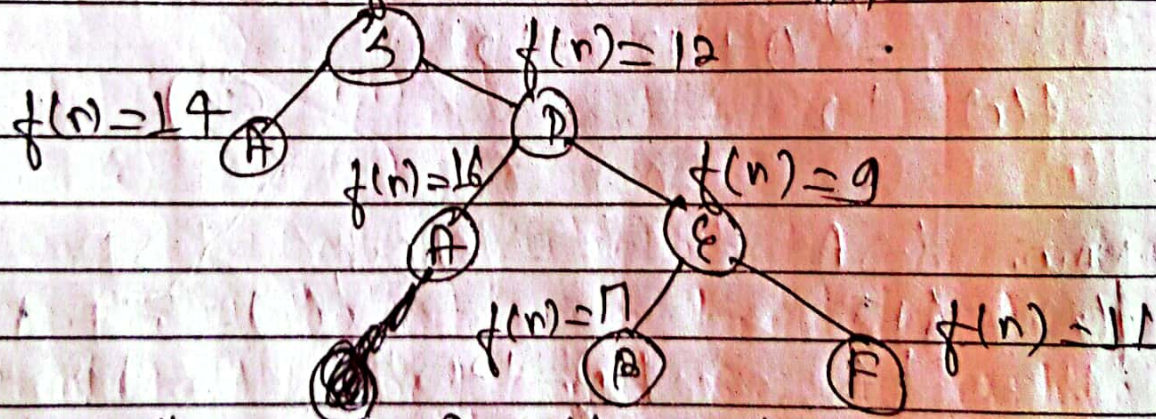
$= 17$

S-D-E-F

$f(n) = 3+2+4+2$

$= 9+2$

$= 11$



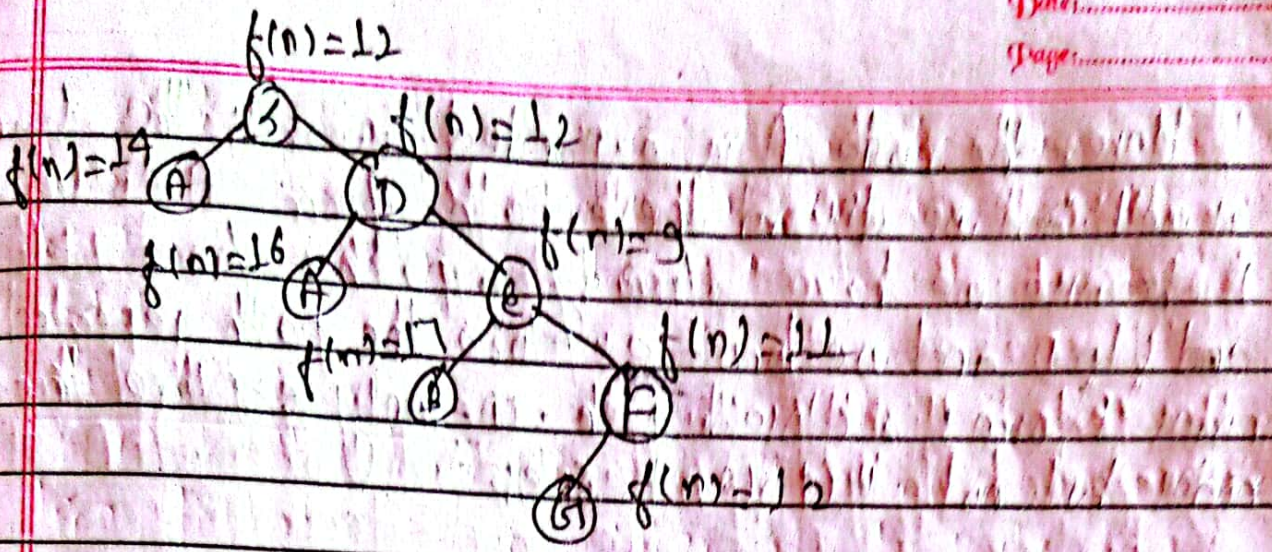
Here we reject the path S-D-E-B and we select S-D-E-F

Then S-D-E-F-G

$f(n) = 3+2+4+3+0$

$= 5+4+3+0$

$= 12$



Thus, the optimal cost to reach the goal node is 12 units and the optimal path is S → B → E → G.

For Greedy Best first search.

Let h(n) be the straight line distances from any vertex to the goal (G).

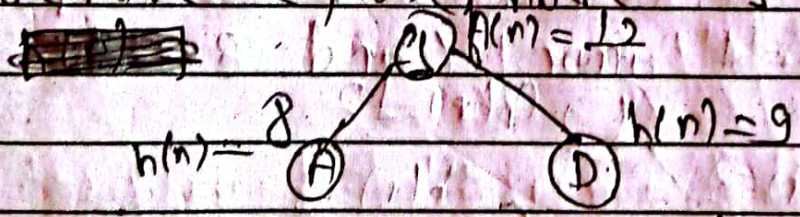
Now, we have,

(i) For "S"

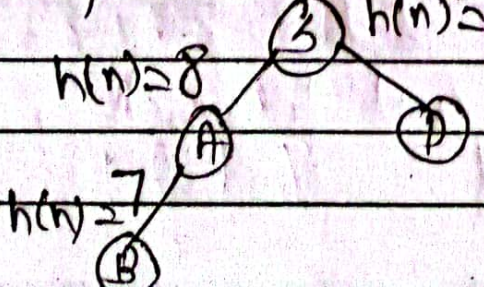
$f(n) = h(n)$

$= 12$ i.e. (S)

(ii) Now, we have explore node "S" as follows

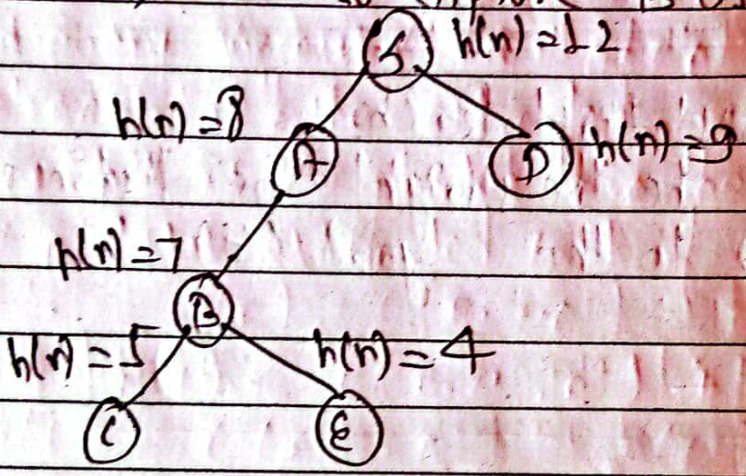


(iii) Here, A is the node with lowest heuristic function i.e. $h(n) = 8$ among the nodes A and D. So we choose the path to A and explore as follows.

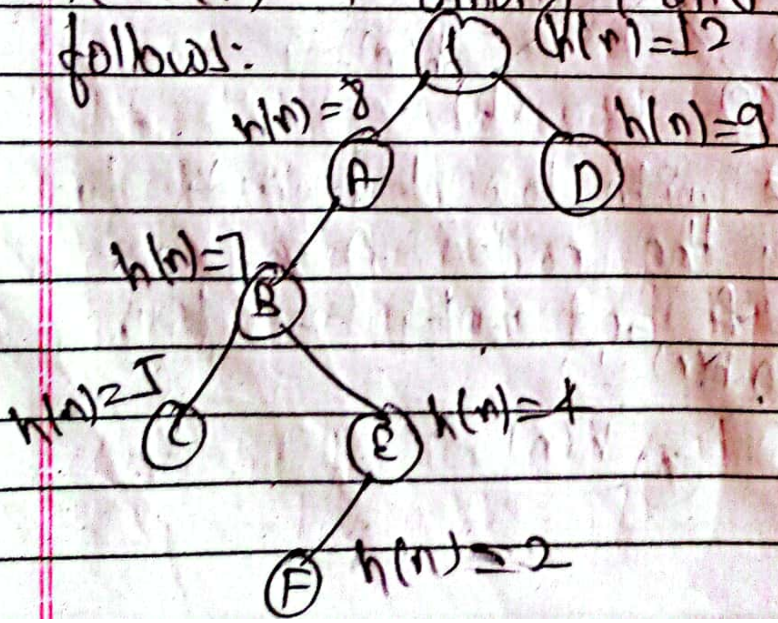


Here, the node A is expanded towards the I and G as well. But we discard these path from A to I and A to D because we have already a path from I to A and I to D and we don't explore the path repeatedly. node "I" form a closed path. We only explore the open node but not "CLOSED" one.

(iv) Here, B is only the node with lowest heuristic func. i.e. $h(n) = 7$ to explore B as follows:

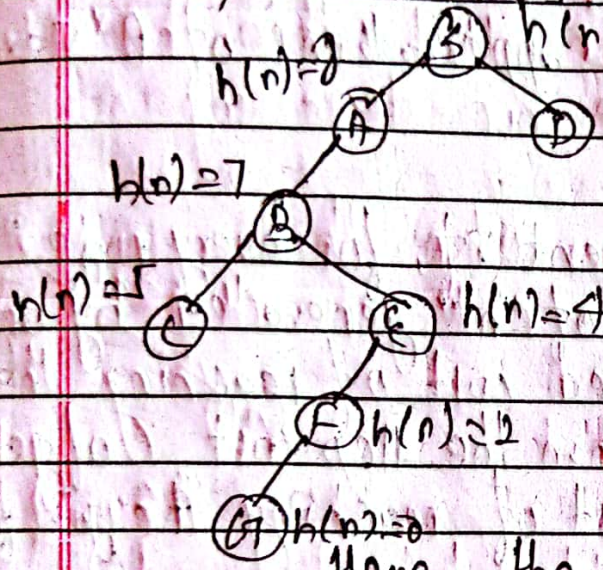


(v) Here, E is the node with lowest heuristic function i.e. $h(n) = 4$ among I and E. So, explore E as follows:



Here, the repeated path again comes, so we discard the repeated path.

(vi) Here, the node with lowest heuristic function to explore G is as follows:-



Here, the node "G" is expanded only towards F but we discard the path since it is repeated.

Thus, in conclusion, the optimal path from S to G using GBFS strategy in above given search is S - A - B - E - F - G. #

(#) Conditions for optimality: Admissibility and Consistency

→ The first condition we require for optimality is that h(n) be an admissible heuristic.

→ An admissible heuristic is one that never overestimates the cost to reach the goal.

→ Because, g(n) is the actual cost to reach "n" along

Date.....
Page.....

the current path, $f(n) = g(n) + h(n)$. We have an immediate consequence that $f(n)$ never estimates the true cost of a solution along the current path through "n".

→ The second, slightly stronger condition called "consistency" is required only for the applications of A* search to graphs.

→ A heuristic $h(n)$ is consistent if, for every node "n" and every successor "n'" of "n" generated by any action "a", the estimated cost of reaching the goal from "n" is no greater than the step cost of getting to n' plus estimated cost of reaching the goal from "n'".

ie. $\boxed{h(n) \leq c(n, a, n') + h(n')}$

→ Every consistent heuristic is also admissible.
→ Consistency is therefore a stricter requirement than admissibility.

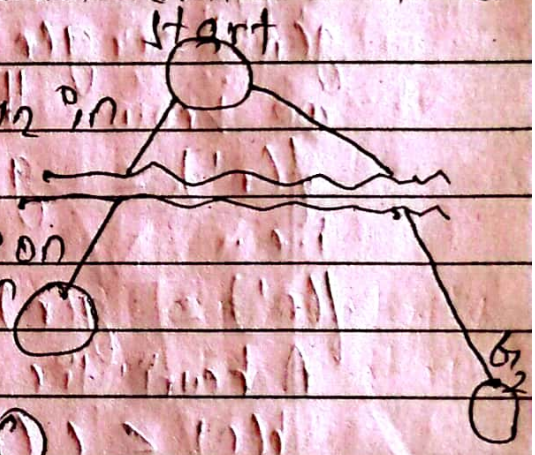
(*) formulating admissible heuristics:
→ n is a node
→ h is a heuristic
→ h(n) is cost indicated by "h" to reach a goal from "n".
→ c(n) is the actual cost to reach a goal from "n".
→ h is admissible if

Date:
Page:

$\forall n,\ h(n) \leq c(n)$, in response to
$f(n) = g(n) + h(n)$ #

Theorem: A* is optimal if h(n) is admissible.

⇒ We know that, if estimated distance h(n) never exceed the true distance h*(n) between the current node to goal node, then the A* algorithm will always find a shortest path. This is known as the admissibility of A* algorithm and h(n) is an admissible heuristic.



Now, suppose suboptimal goal (G_2) in the queue.

Let "n" be an unexpected node on a shortest path to optimal goal G_1 and C^* be the cost of optimal goal node.

Then,

$f(G_2) = g(G_2) + h(G_2)$

$f(G_2) = g(G_2)$ $(\because h(G_2) = 0)$

$f(G_2) > C^*$ ... (i)

Again, since h(n) is admissible, it does not overestimate the cost of completing the solution path.

i.e. $f(n) = g(n) + h(n) \leq C^*$ ... (ii)

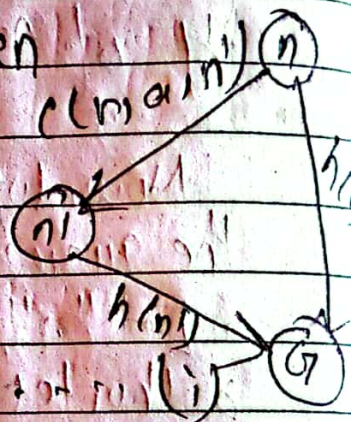
Now from (i) & (ii), we get,

$f(n) \leq C^* < f(G_2)$

Since $f(G_2) > f(n)$, A* will never select G_2

for expansion.
Thus A* algorithm gives us optimal solution when heuristic function is admissible.
Hence, proved #

(#) Theorem: If h(n) is consistent, then the values of f(n) along the path are non-decreasing.
⇒ Proof:
Suppose n' is successor of n, then
$g(n') = g(n) + c(n, a, n')$
we know that,
$f(n') = g(n') + h(n')$
i.e.
$f(n') = g(n) + c(n, a, n') + h(n') \quad \ldots (i)$



A heuristic is consistent if:
$h(n) \leq c(n, a, n') + h(n') \quad \ldots (ii)$
Now, from (i) & (ii), we get,
$f(n') = g(n) + c(n, a, n') + h(n') \geq g(n) + h(n)$
$= f(n)$
i.e. $f(n') \geq f(n)$
Thus, f(n) is non-decreasing along my path if h(n) is consistent.
Hence, proved #

Date:
Page:

(#) Local Search Algorithms and optimization problems

→ If the path to the goal doesnot matter we might considered a different class of algorithm that doesnt worry about paths at all.

→ Local search algorithms operate using a single current node rather than multiple paths and generally move only to the neighbours of that node.

→ Typically, the paths followed by the search are not retained.

→ Although local search algorithms are not systematic, they have two key advantages :-

(i) They use very little memory, usually a constant amount

(ii) And, they can often find reasonable solutions in large or infinite state spaces for which systematic algorithms are unsuitable.

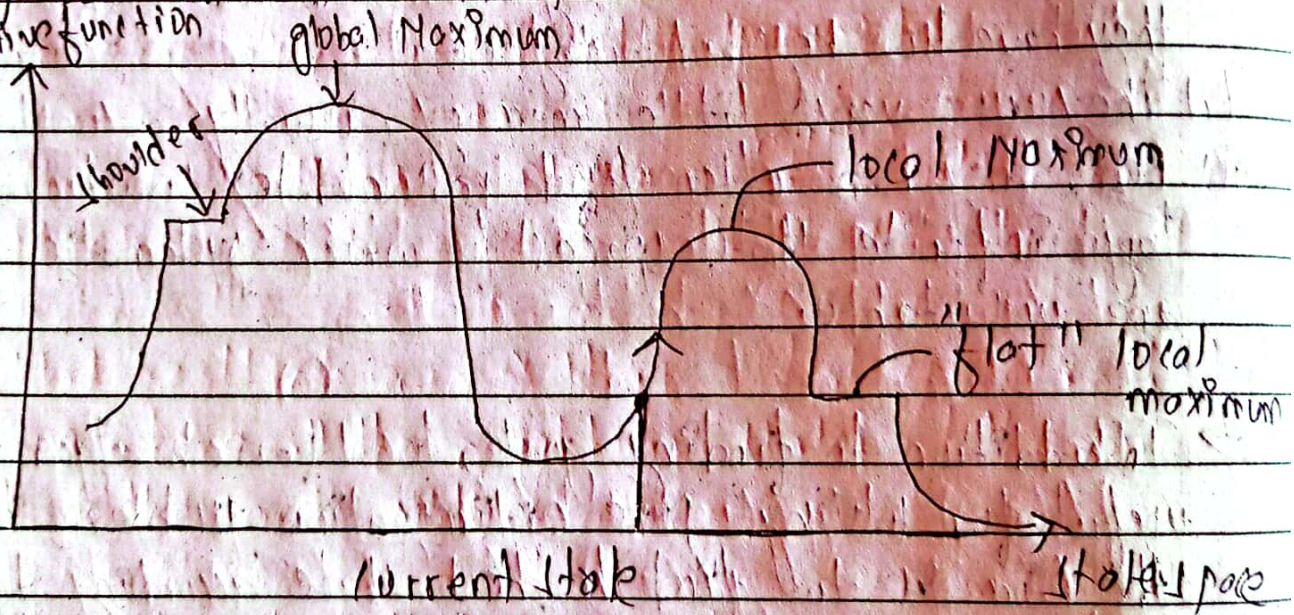
→ In addition to finding goals, local search algorithms are useful for solving pure optimization problem in which the aim is to find the best state according to an objective function.

→ For Example: nature provides an objective function — reproductive fitness — that Darwinian evolution could be seen as attempting to optimized but there is no "goal test" and no "path cost" for this problem.

→ Landscape has both "location" (defined by the state) and "elevation" (defined by the value of the heuristic cost function or objective function).

→ If elevation corresponds to the cost then the aim is to find the lowest valley — a "global minimum".
→ If elevation corresponds to an objective function, then the aim is to find the highest peak — a "global maximum".

→ Local search algorithm explores the landscape.
→ A local search algorithm always finds a goal if one exists and an optimization algorithm always finds global minimum/maximum.



③ Hill Climbing search:
→ It is a simple hill climbing search.
→ Hill climbing Algorithm is a local search algorithm which continuously moves in the direction of increasing elevation or value to find the peak of the mountain or best solution to the problem.
→ This algorithm terminates when it reaches a peak

value where no neighbour has a higher value.

→ Hill climbing is mostly used when a good heuristic is available. In this algorithm, we don't need to maintain and handle the search tree or graph as it only keeps a single current state.

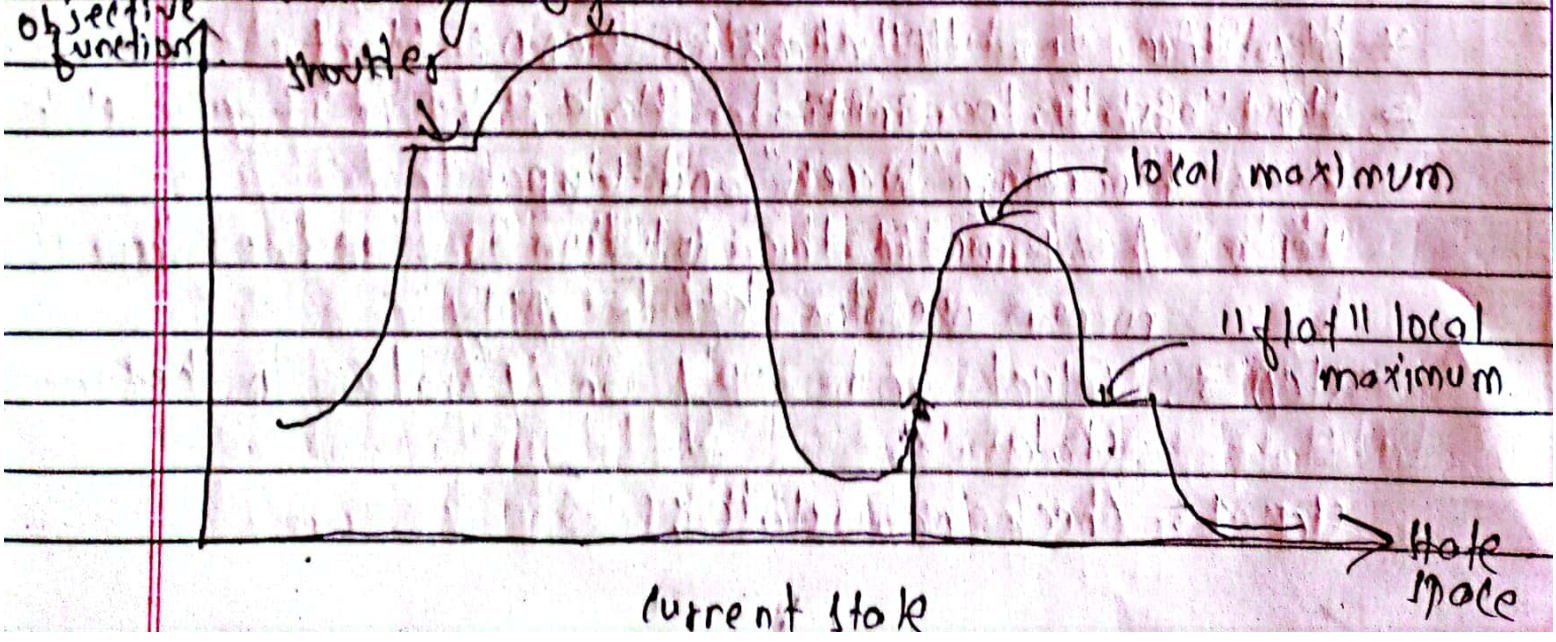
→ It is a local search algorithm but no knowledge of global domain.

→ It uses greedy approach as it moves in the direction which optimizes the cost.

→ It is the variant of generate and test method. This method produces feedback which helps to decide which direction to move in the search space.

→ No back tracking — It doesn't backtrack search space, as it doesn't remember the previous states i.e. if the best state is not found then continue with the selected node's expansion.

(*) ~~Features of Hill climbing~~ State space Land-scape for Hill climbing



Date:
Page:

Fig: Different regions in the state-space landscape

(i) Local maximum ⇒ It is a state which is better than its neighbour states, but there is also another state which is higher than it.

(ii) Global maximum ⇒ It is the best possible state of the landscape. It has the highest value of objective fun

(iii) Current state ⇒ It is a state in a landscape diagram where an agent is currently present

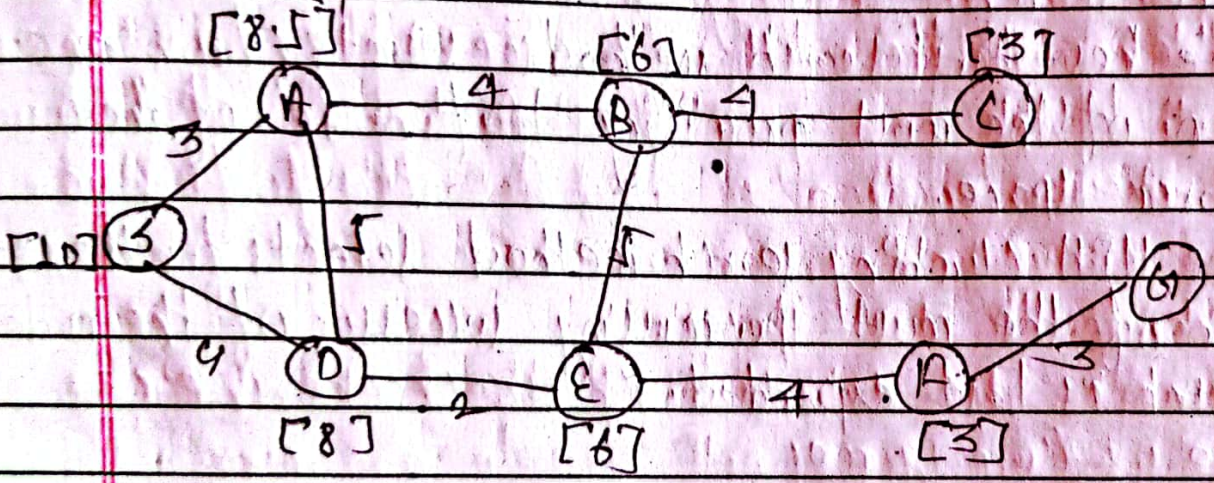
(iv) Shoulder ⇒ It is a plateau ~~region~~ region which has an edge.

(v) Flat local maximum ⇒ It is a flat space in the landscape where all the neighbour states of the current state have the same value.

* Algorithm for Hill climbing
Step 0: Start
⇒ Step 1: Evaluate initial state; if it is goal state then return "Success" and stop.
Step 2: Loop until the solution is found or there is no new operator left to apply.
Step 3: Select and apply an operator to the current state.
Step 4: Check new state:

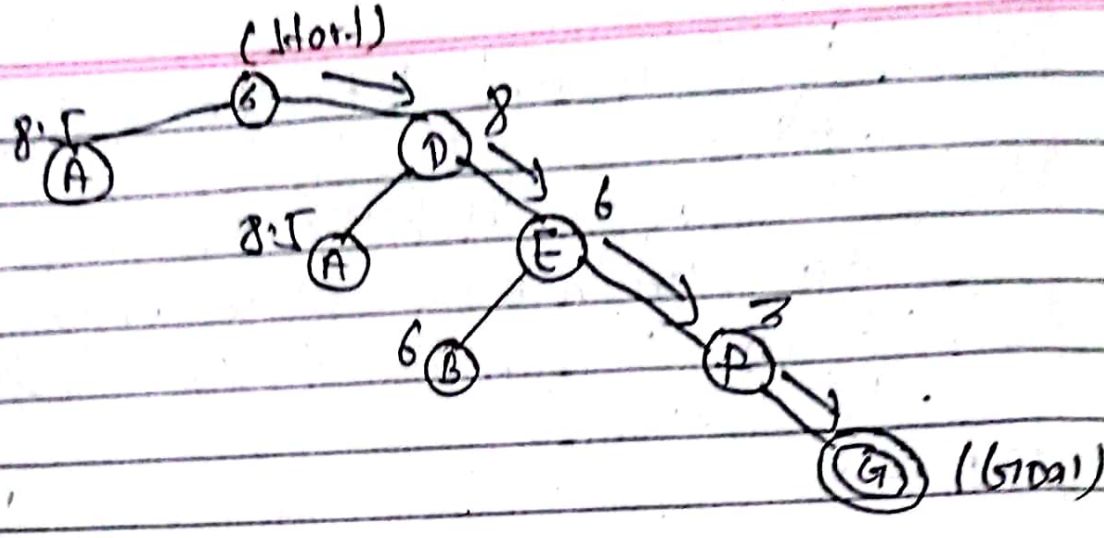
4.1 : If it is goal state, then return "success" and quit.
4.2 : Else, if it is better than the current state, then assign new state as a current state.
4.3 : If not better than the current state, then return to step 2
Step 5 : Exit

(#) Perform Hill climbing search Algorithm on the following search space:



⇒ Solution:
Here, In the given case, it is considered that the most promising successor of a node is the one that has the shortest straight line distance to the goal node "G". In the given figure, the straight line distances between each city and goal "G" is indicated in the square bracket i.e, the heuristic.
Now, the hill climbing search from S to G. Proceed as follows:

The most promising node of a node is its nearest neighbour node.



(X) Differences between Hill climbing search and Best First search.

→ The Best First search method selects for an expansion of the most promising leaf node of the current search space.

→ The Hill climbing search method selects for the expansion of the most promising successor of the node which is last expanded.

(X) Problems with Hill climbing

(I) Local Maximum → A local maxima is a peak state in the landscape which is better than each of its neighbouring state, but there is another state also which is higher than local maxima. It means that it gets stuck at the local maxima where there are no better neighbours.

(II) Plateau → A plateau is a flat surface of the search space for the neighbour states of the current state which all

Date:
Page:

contain the same evaluation value, because of this algorithm doesn't find any direction to move. A hill climbing search might be lost in plateau area.

(iii) Ridge ⇒ A ridge is a sequence of local maxima i.e. a special form of local maxima. A ridge has an area which is higher than its surrounding areas, but its left has a slope, and cannot be reached in a single move.

(4) Simulated Annealing (SA) →

→ A hill climbing algorithm which never makes a move towards a lower value guaranteed to be incomplete because it can get stuck on a local maxima, And, if algorithm applies a random walk, by moving a successor it may complete but not efficient.

→ Simulated Annealing is an algorithm which yields both completness and efficiently.

→ In mechanical term, "annealing" is a process of hardening a metal or glass to a high temperature then cooling gradually, so this allows the metal to reach a low energy crystalline state.

→ This above illustrated same process is used in simulated annealing in which the algorithm picks a random move instead of picking the best move.

→ If the random move improves the state, then it follows the same path. Otherwise, the algorithm follows

the path which has the probability of less than 1 or it moves downhill and chooses another path

(H) Game playing and Adversarial search Techniques.
→ It is one of the subfields of A.I. Game playing involves abstract pure form of computation that seems to require intelligence so that game playing has clear relationships to intelligence and yet is well defined states and rules.
→ A game can be formally defined as a kind of search problem with the following components:
(i) Initial state (iii) A terminal test
(ii) A successor function (iv) A utility function.

⇒ Deterministic Two-player:
⇒ Example: Tic-Tac-Toe, chess, checkers, etc
⇒ Zero-sum games
• One player maximizes the result.
• Other minimizes the result
⇒ Mini-Max search
• A state-space search tree.
• Player's alternate
• Each layer consist of a round of moves*.
• Choose move to the position with highest mini-Max value = Best Achievable utility against best play

Date:
Page:

⇒ Searching for the next move
(i) Evaluation function → Heuristic to evaluate utility of a state without any exhaustive search.
(ii) Pruning → Makes the search more efficient by discarding portions of the search tree that cannot improve the quality of result.

⇒ A Game formulated as a search Player:
(i) Initial state: A Board position and turn.
(ii) Operators → Definition of legal moves
(iii) Terminal state → Conditions for when game is over
(iv) Utility function → A numeric value that describes the outcome of the game. Example: -1, 0, 1, for loss, draw and win respectively (AKA payoff function)

⇒ Deterministic single player:
⇒ Deterministic, single player, perfect information:
- know the rules.
- know what actions do.
- know when you win.
- Example: 8-puzzle, Rubik's cube, freecell, etc.
- It is just a search.

⇒ Slight reinterpretation
- Each node stores a value: the best outcome it can reach.
- This is the maximal outcome of its children. (The max value).

- Note that, we donot have pathr ivms of before. (utilities at the end).

⇒ After search, can pick a move that leads to the best node.

(#) Minimax Algorithm:

⇒ Minimax algorithm is a kind of backtracking that is used in decision making and game theory to find the optimal move for a player assuming that your opponent also plays optimally.

⇒ It is widely used in turn based games such as Tic-Tac-Toe, chess, Backgammon, etc.

⇒ In mini-max, the two players are called maximizer and minimizer, known as MAX and MIN respectively.

⇒ The max player tries to get the highest score possible that is (i.e.) max will try to maximize its utility (Best move).

⇒ The minimizer tries to do the opposite of the maximizer and get the lowest score possible (worst move).

⇒ Normally, nodes representing your opponent's moves are drawn as circles or downward pointing triangles which are also known as MIN nodes. The goal at a min node is to minimize the value of the subtree rooted at that node.

⇒ To do this, a MIN node chooses the child with the

least value and that becomes the value of the MIN node.

→ Normally, we start with MAX node and reach to the terminal by selecting maximum at MAX node and minimum at MIN node alternatively.

Demonstration

Let us considered a game adversary Tic-Tac-Toe. A Game can be formally defined as a kind of search problem through "initial state", "successor function", "terminal test", and "utility function".

Let us assign the following value for the game: 1 for win by X, 0 for draw, and "-1" for lose by X. Given the values of the terminal nodes (win for X (1), lose for X (-1), or draw (0)), the values of the non terminal nodes are computed as follows:-

(i) The value of the node where it is the turn of player "X" to move is the maximum of the values of its successor. (because "X" tries to maximize its outcome).

(ii) The value of a node where it is the turn of "O" to move is the minimum of the values of its successor (because "O" tries to minimize the outcome of "X").

→ The values of the leaves of the tree are given by the following rules of the game:-

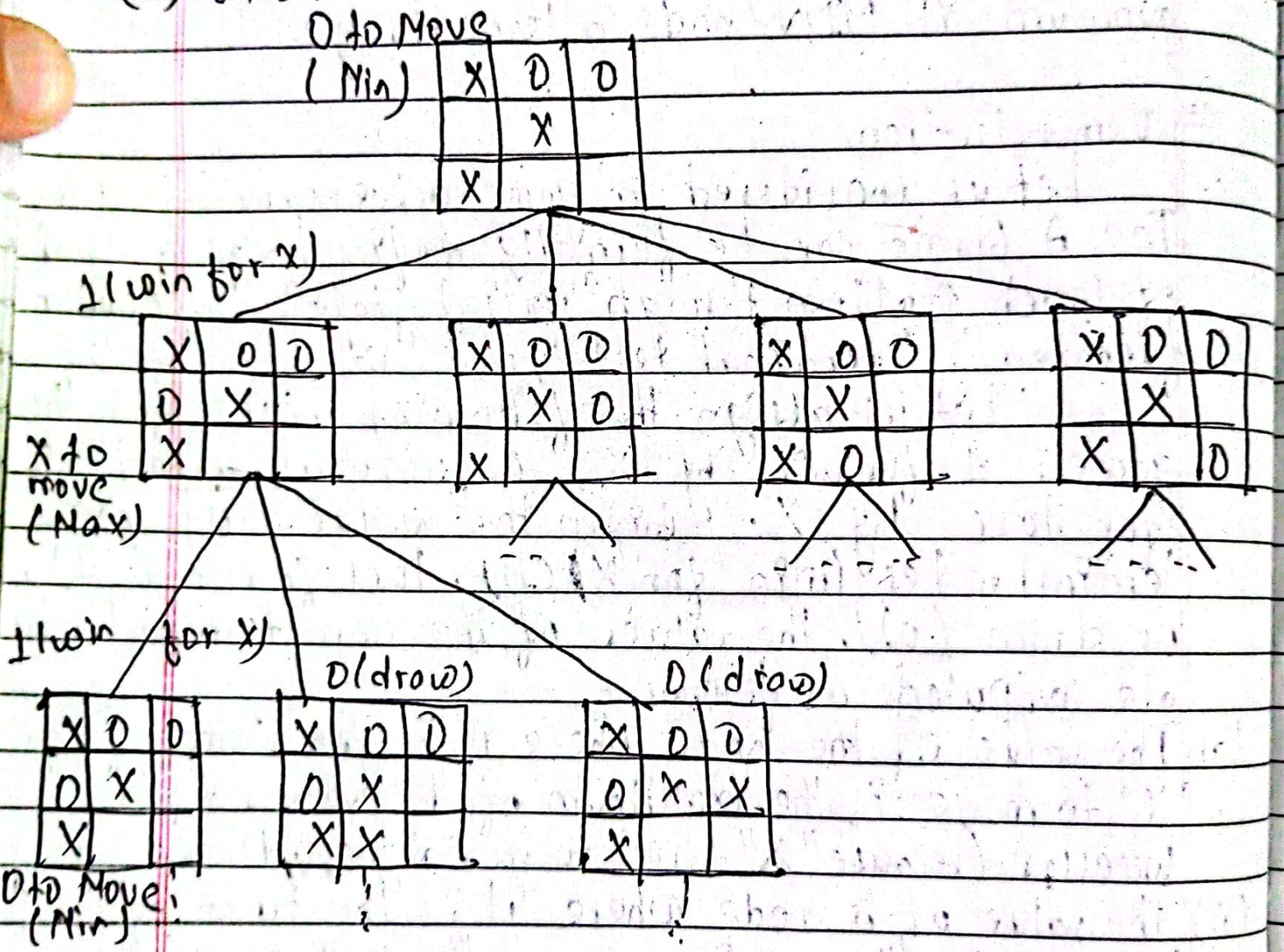
(i) If there are three (3) "X" in a row, column,

Date:
Page:

diagonal.

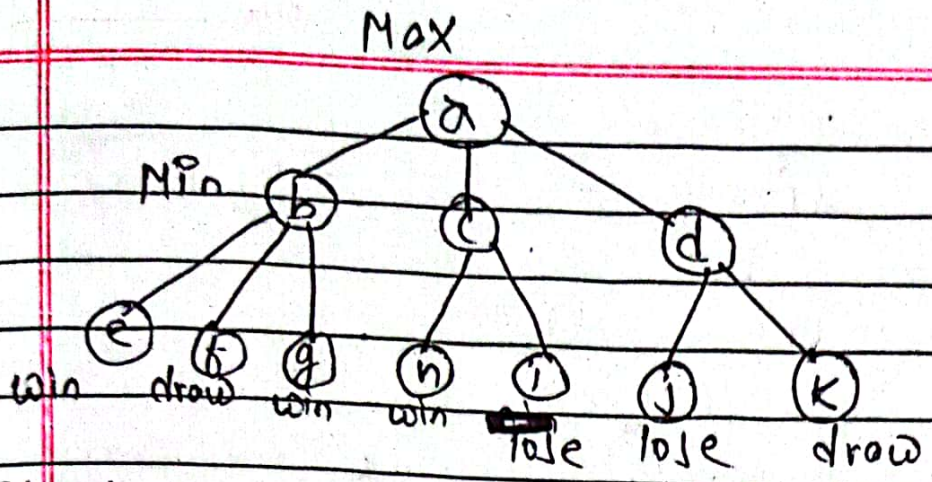
(ii) -1 if there are three "O" in a row, column, or diagonal.

(iii) 0 / otherwise



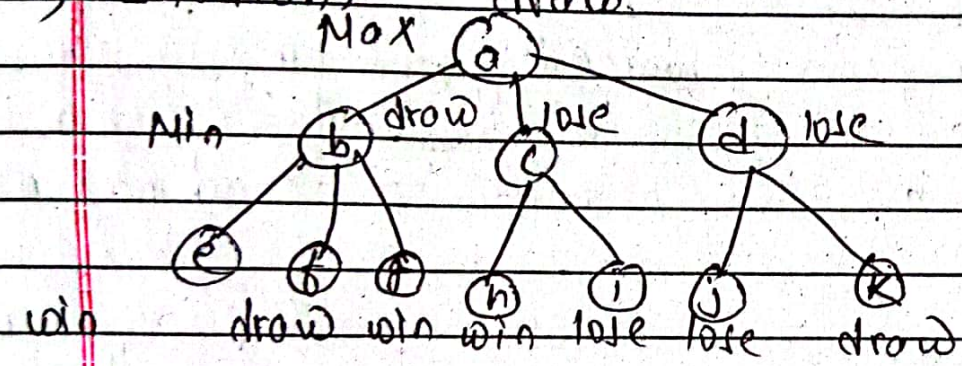
Example 1:

Considered the following game tree drawn from the point of view of the maximizing player



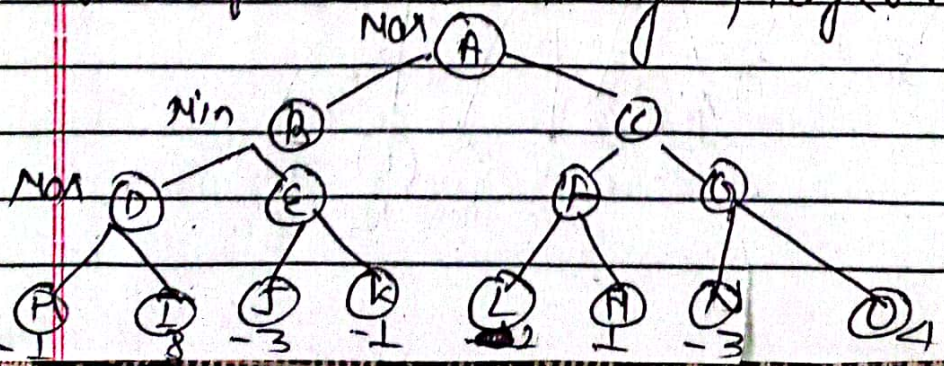
(Q) What move should be chosen by the max player, and what should be the response of min player, assuming that both are using the MinI-Max procedure?

⇒ Solution;

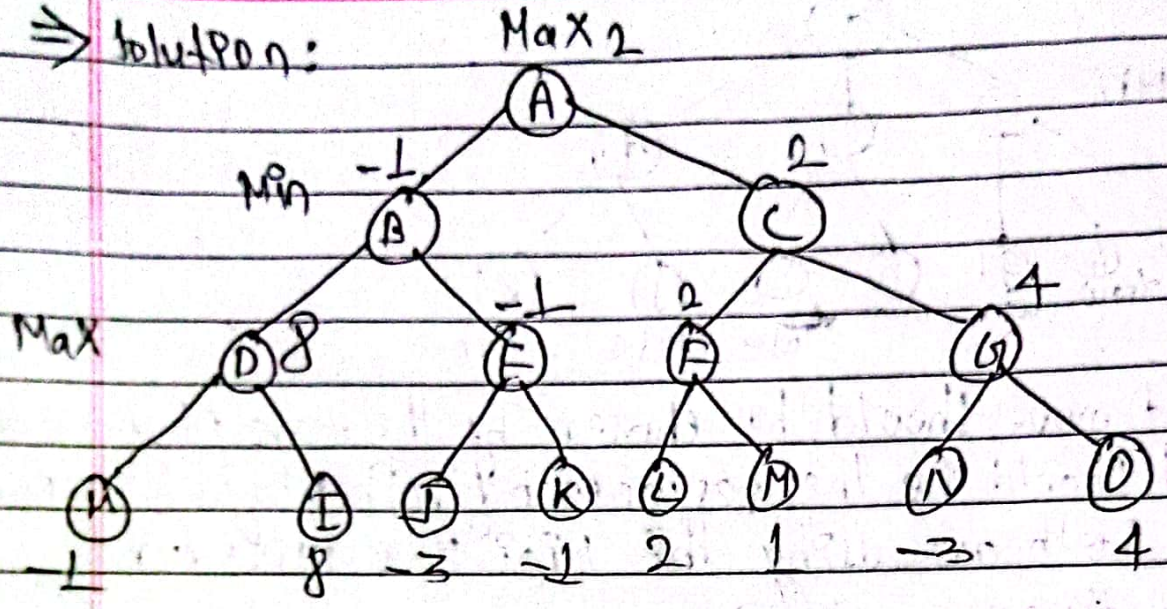


⇒ Here, from the above game tree represented through MinI-Max procedure, we come to know that Max will move to "b" and Min will respond by moving to "f".

(Q) Considered the following game tree drawn from the view of the maximizing player:

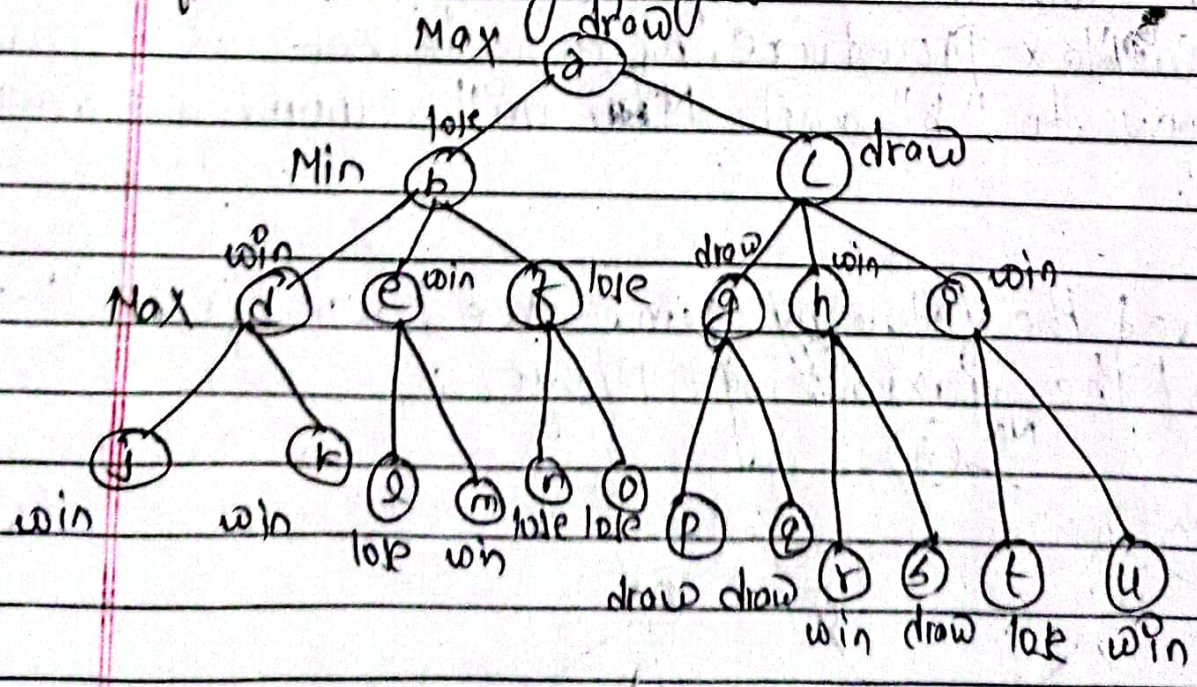


⇒ Solution:



→ Here from the above obtained game tree we come to know that Max will move to the node "C" and Min will respond by moving to node "F"; and max moves to "L".

(Q) Consider the following game tree drawn from the POV of the maximizing player.



Date:.....

Page:.....

⇒ Here, from the above obtained game tree constructed through the use of MiniMax algorithm, we come to know that Max will move to "c", Min will respond by moving to "g", max will move to either "p" or "q".

(*) Properties of Mini-Max algorithm:

(i) Completeness ⇒ MiniMax algorithm is complete. It will definately find a solution (if exists), in the finite search tree.

(ii) Optimality ⇒ MiniMax algorithm is optimal if both the opponents are playing optimally.

(iii) Time complexity ⇒ As it performs DFS for the game tree. So, the time complexity of MiniMax algorithm is $O(b^m)$, where, b = branching factor of the gametree, and m = maximum depth of the tree.

(iv) Space complexity ⇒ Space complexity of MiniMax algorithm is also similar to DFS which is $O(b.m)$.

(*) Limitation of MiniMax algorithm:

⇒ The main drawback of MiniMax algorithm is that it really gets slow for the complex games such as chess etc.

⇒ This type of game has a huge branching factor, and the player has lots of choices to decide.

⇒ This limitation of the MiniMax algorithm can be improved by making the use of "Alpha-Beta pruning".

(#) Alpha-Beta Pruning:

→ It is a modified version of MINI-MAX algorithm, i.e. it is an optimization technique for the minimax algorithm.

→ As we have seen, in the MINI-MAX algorithm that the number of game state it has to examine are exponential in depth of the tree. Since, we cannot eliminate the exponent so we can cut it to half.

→ Hence, there is a technique by which without checking each node of the game tree, we can compute the correct MINI-MAX decision, and this technique is called "pruning".

→ This involves two threshold parameters "alpha" and "beta" for future expansion, so it is called as alpha Beta pruning. It is also called as Alpha-Beta Pruning.

→ Alpha-Beta pruning can be applied at any depth of the tree, and sometimes it not only prune the tree leaves but also the entire subtree.

→ The two parameters can be defined as:

(i) alpha (α) → The best (highest-value) choice we have found so far at any point along the path of MAXimizer. The initial value of alpha is "$-\infty$".

(ii) Beta (β) → The best (lowest-value) choice we have found so far at any point along the path of MINimizer. The initial value of beta is "$+\infty$".

(#) Conditions for $\alpha\beta$ pruning.

→ The "max" player will only update the value of "α"

Date:
Page:

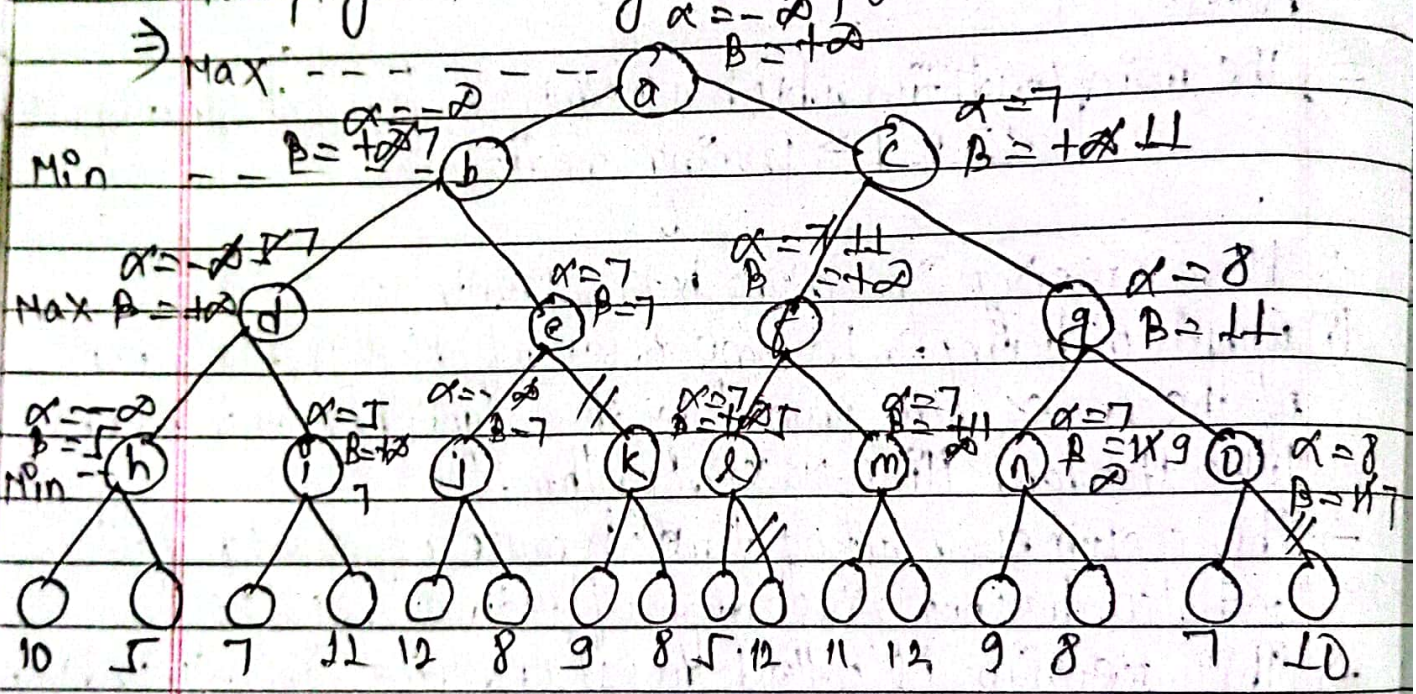
→ The "min" player will only update the value of "beta".
→ While backtracking the tree, the node values will be passed to the upper nodes instead of values of alpha and beta.
→ We will only pass the alpha and beta values to the child nodes.
⇒ The main condition which is required for α-β pruning is:
$\alpha \geq \beta$ → pruning condition.

* Move ordering in alpha beta pruning:
(i) Worst ordering → In some cases of α-β pruning, none of the nodes are pruned by the algorithm and works like standard Mini-Max algorithm.
→ This consumes a lot of time because of α and β factors, and also does not give any effective results.
→ This scenario is called worst ordering in α-β pruning. In this case, the best move occurs on the right side of the tree.

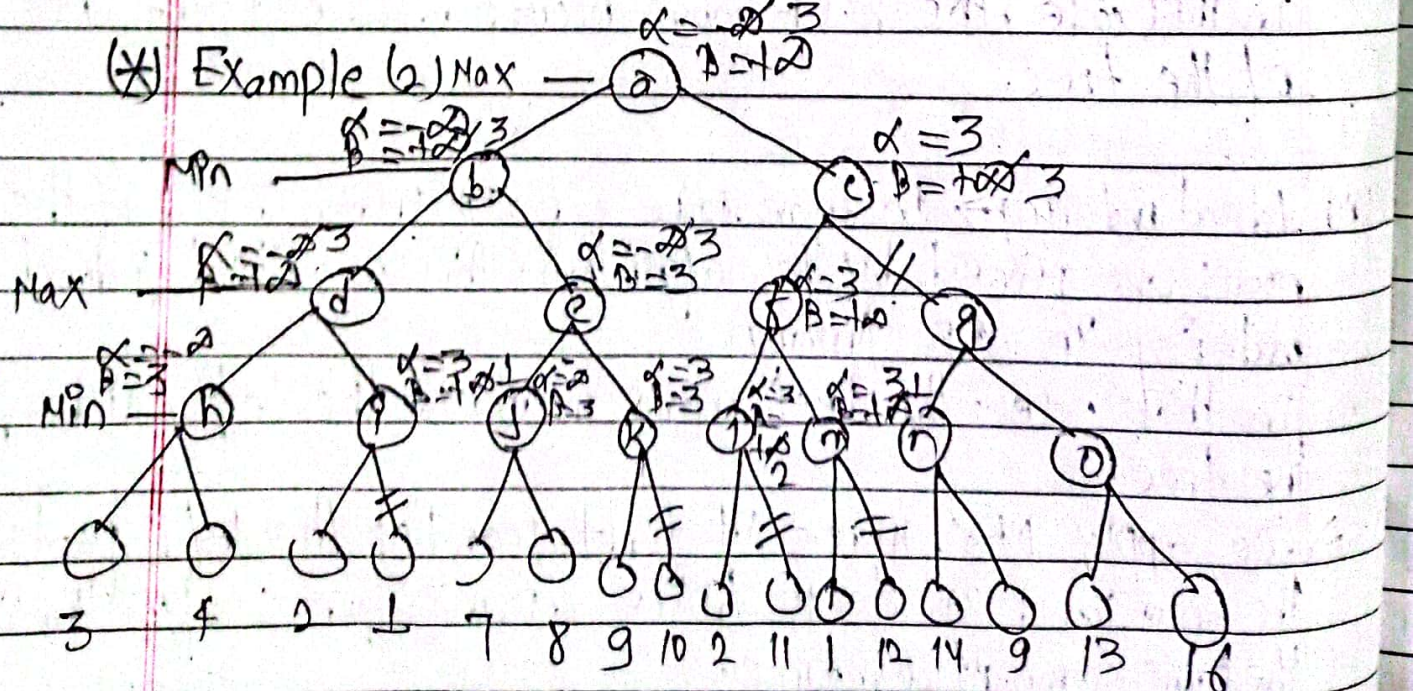
(ii) Ideal ordering → In some case of α-β pruning, lots of the nodes are pruned by the algorithm. This is called ideal ordering in α-β pruning.
→ In this case, the best move occurs on the left side of the tree.
→ We apply DFS, Hence it first searches the left part of the tree and go deep twice as mini-max algorithm in the same amount of time.

→ Hence the optimal path is a-b-d-9 and the optimal value of the maximizer "a" is 7. Then, the max will move to b and min will respond by moving to P and max will move to P.

Example → 1:
Considered the following game tree drawn from the point of the maximizing player and use the αβ procedure to determine the moves which should be chosen by the two players, assuming a DFS generation of the tree.



(*) Example (2) Max

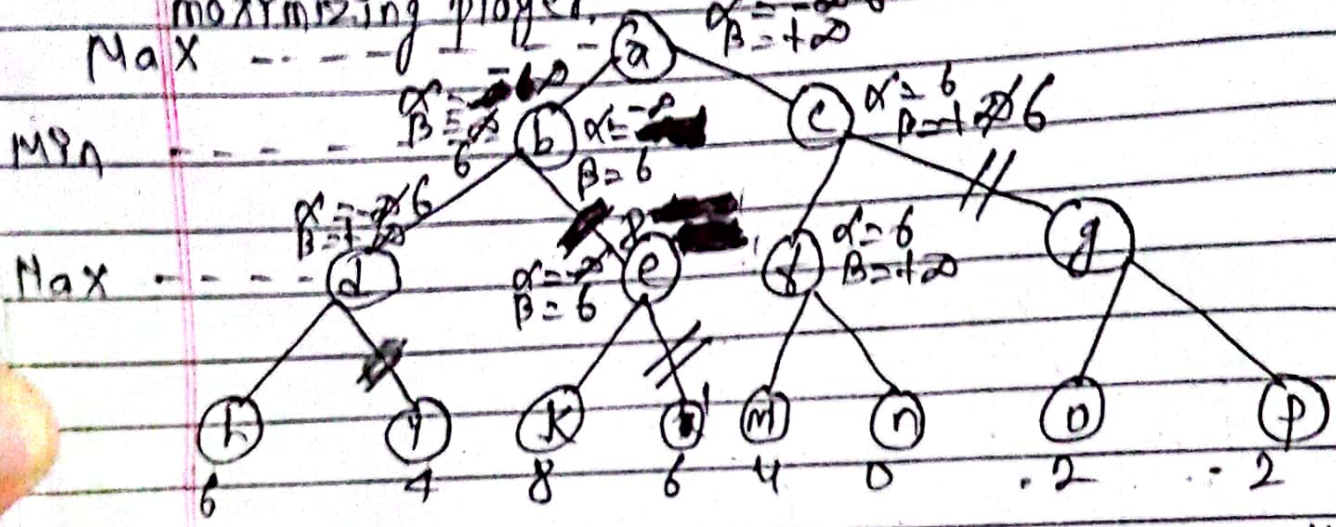


(*) Considered the following two-person game tree, draw the part of work of the minimax Pog player.

Assume a static evaluation function which produces the following evaluation function:

(I) = Win
(R) = Win
(L) = Lose
(2) = Lose
(3) = Lose
(Q) = Win
(P) = Draw
(T) = Draw
(S) = Draw
= Win

(*) Considered the following game tree from the pov of maximizing player:



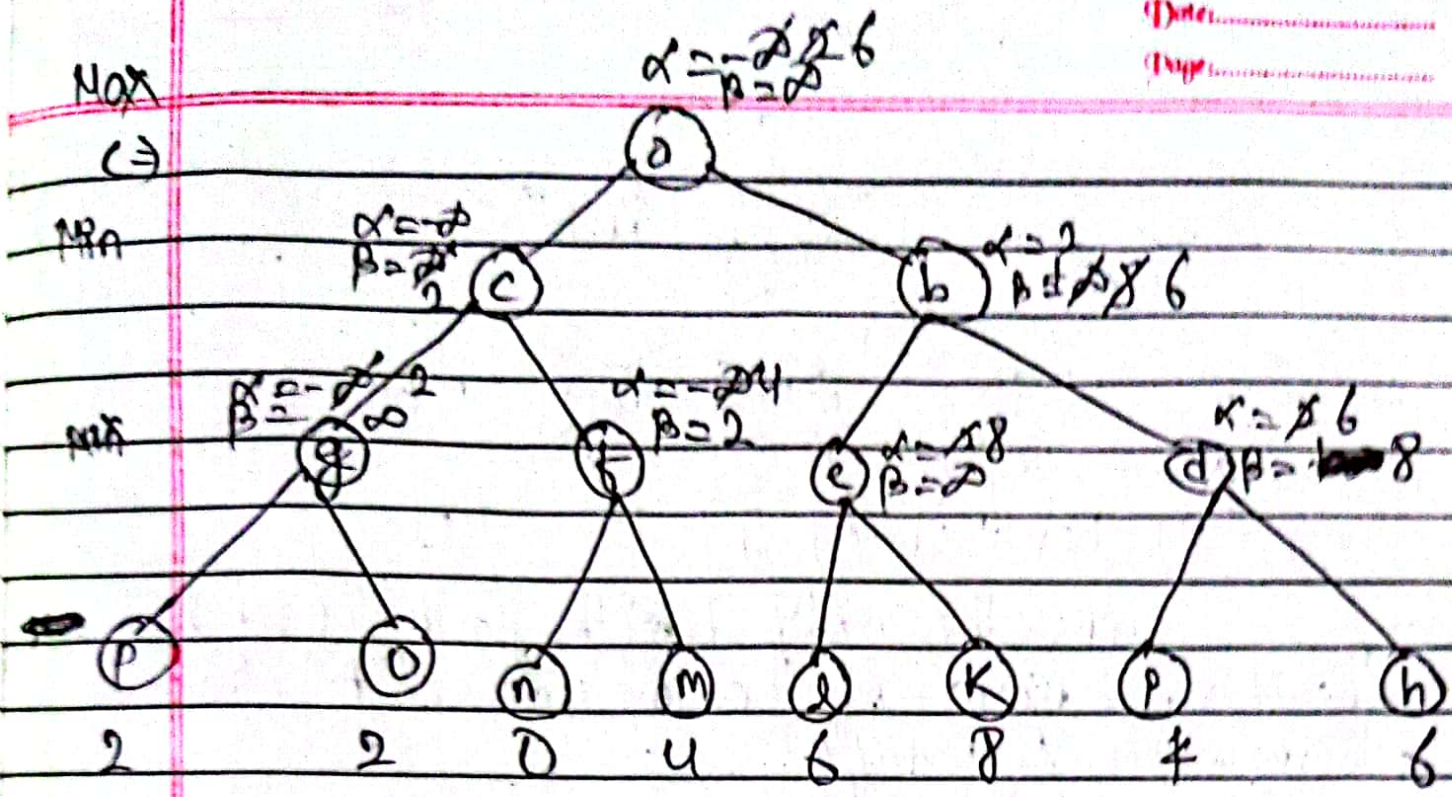
(a) Use the Min-Max procedure and show what move should be chosen by the two players.

(b) Use the α-β pruning procedure and show what nodes would not be examined.

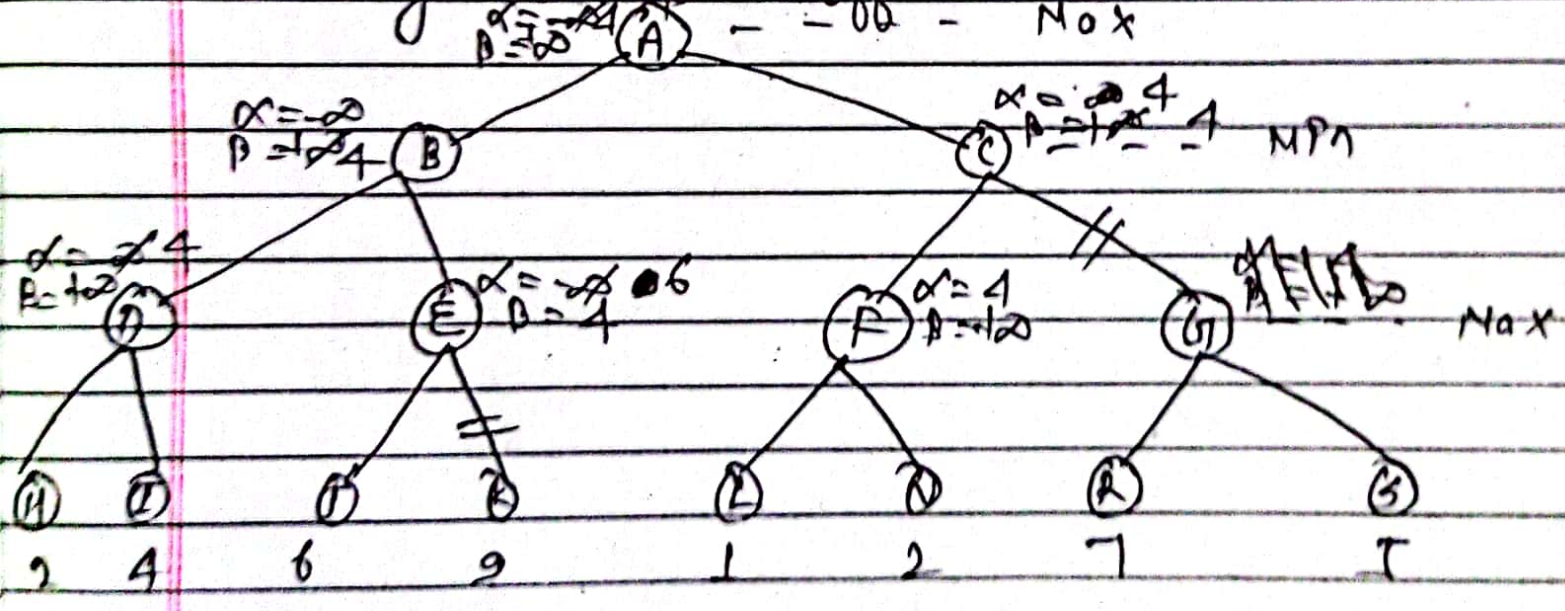
(c) Consider the mirror image of the above tree and apply again the α-β pruning procedure. What do you notice?

a ⇒ Here, the optimal path is a-b-d-h and the optimal value of the maximizer is 6. Then, the max will move to b and min will respond by moving to d and max will move to h.

b ⇒ The node "i", "g", "o" and "p" would not be examined.

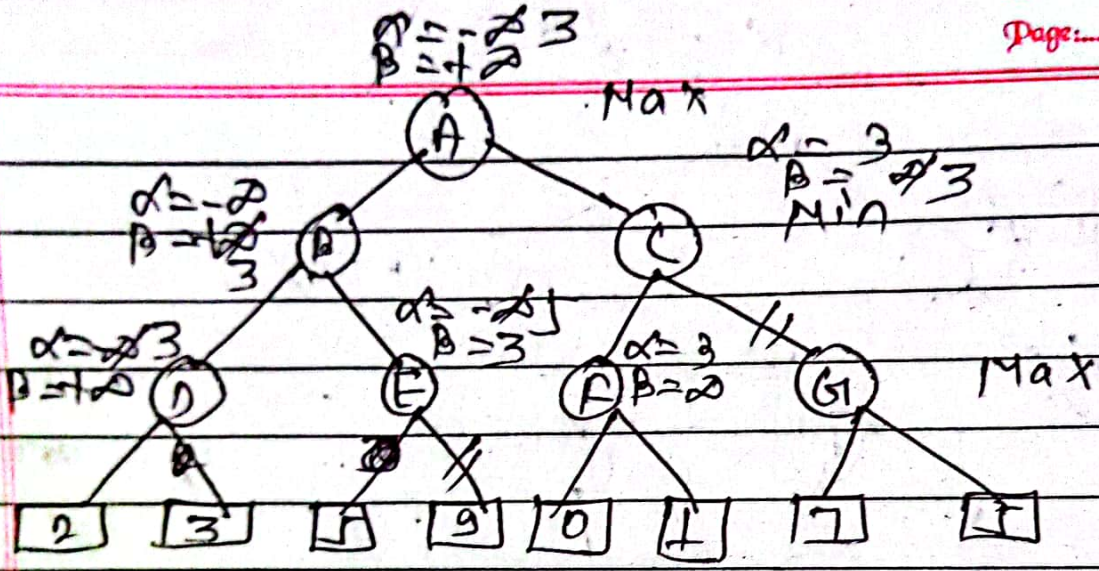


(Q) Given the following search space determine if there exists any α and β cutoff.



Date:
Page:

(2)

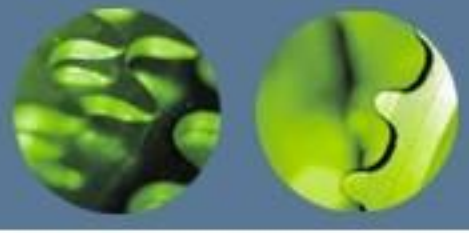


Here the optimal path is a-b-d and the optimal value of the maximizer "A" is 3. There exits 2 cutoffs.

3. Problem Solving By Searching (Continued..)

Ujjwal Rijal

rijalujjwal09@gmail.com



Constraint Satisfaction Problems (CSP)

✓ Formally,

A constraint satisfaction problem consists of three components, X, D, and C:

X is a set of variables, $\{X_1, \ldots, X_n\}$.

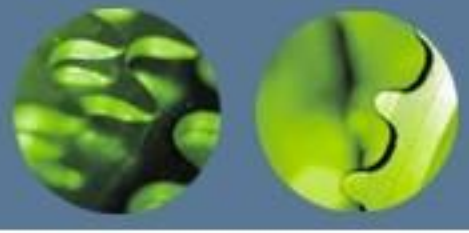
D is a set of domains, $\{D_1, \ldots, D_n\}$, one for each variable.

C is a set of constraints that specify allowable combinations of values.

Each domain D_i consists of a set of allowable values, $\{v_1, \ldots, v_k\}$ for variable X_i. Each constraint C_i consists of a pair $\langle scope, rel \rangle$, where *scope* is a tuple of variables that participate in the constraint and *rel* is a relation that defines the values that those variables can take on. A relation can be represented as an explicit list of all tuples of values that satisfy the constraint, or as an abstract relation that supports two operations: testing if a tuple is a member of the relation and enumerating the members of the relation. For example, if X_1 and X_2 both have the domain {A,B}, then the constraint saying the two variables must have different values can be written as $\langle (X_1, X_2), [(A, B), (B, A)] \rangle$ or as $\langle (X_1, X_2), X_1 \neq X_2 \rangle$.

Example Problem: Map Coloring

- ✓ We are given the task of coloring each region either red, green, or blue in such a way that no neighboring regions have the same color.



Example Problem: Map Coloring

- ✓ To formulate this as a CSP, we define the variables to be the regions. $X = \{WA, NT, Q, NSW, V, SA, T\}$
- ✓ The domain of each variable is the set $D_i = \{red, green, blue\}$.
- ✓ The constraints require neighboring regions to have distinct colors.
- ✓ Since there are nine places where regions border, there are nine constraints:

$$C = \{SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V, \\ WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V\}.$$

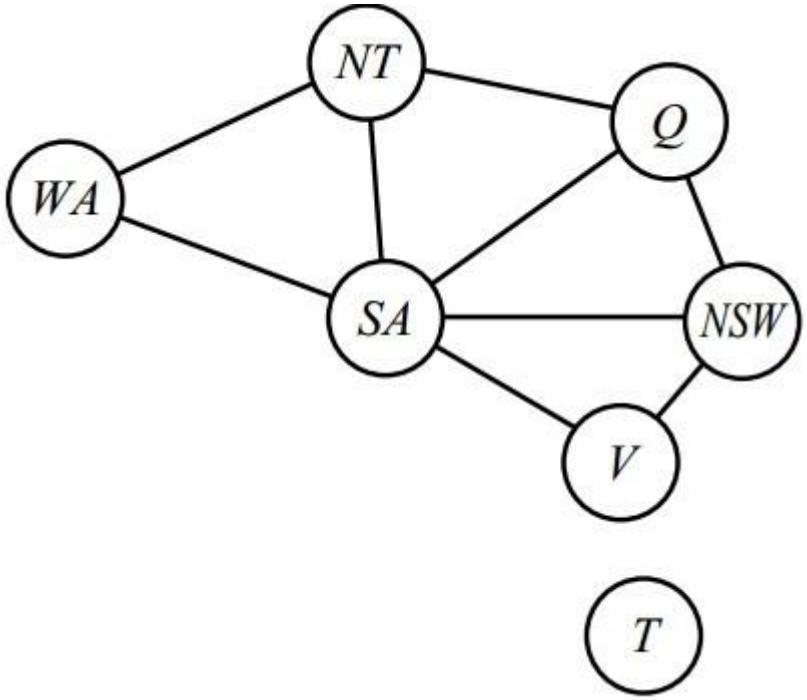
Example Problem: Map Coloring

- ✓ There are many possible solutions to this problem, such as

$$\{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = red\}.$$

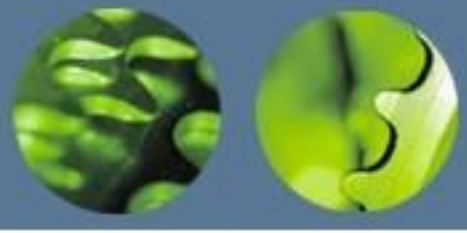
- ✓ It can be helpful to visualize a CSP as a constraint graph

- ✓The nodes of the graph correspond to variables of the problem, and a link connects any two variables that participate in a constraint.



Example Problem: Map Coloring

- ✓ There are many possible solutions to this problem, such as



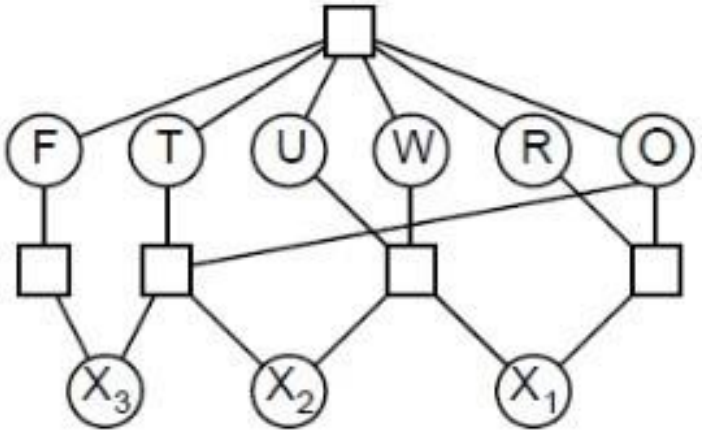
Constraint Satisfaction Problems

Why formulate a problem as a CSP?

- ✓ CSP solvers can be faster than state-space searchers because the CSP solver can quickly eliminate large swatches of the search space.
- ✓ For example, once we have chosen {SA = blue} in the Australia problem, we can conclude that none of the five neighboring variables can take on the value blue.

Example Problem: Crypt-arithmetic

```
   T W O
+  T W O
-------
F O U R
```



Variables: $F\ T\ U\ W\ R\ O\ X_1\ X_2\ X_3$
Domains: $\{0,1,2,3,4,5,6,7,8,9\}$
Constraints
$alldiff(F,T,U,W,R,O)$
$O+O=R+10\cdot X_1$, etc.

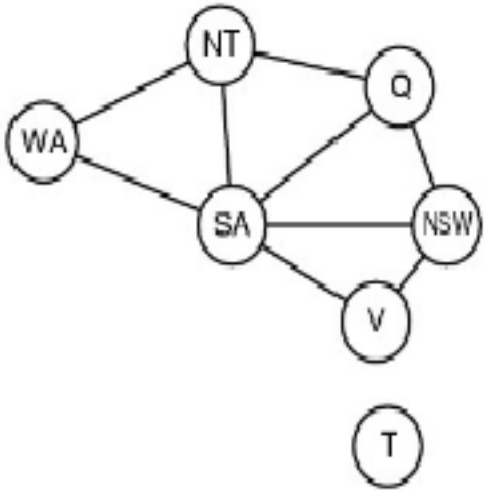
A Crypt-arithmetic problem. Each letter stands for a distinct digit; the aim is to find a substitution of digits for letters such that the resulting sum is arithmetically correct, with the added restriction that no leading zeroes are allowed.

CSP as a Search Problem

- Initial state:
  - {} – all variables are unassigned
- Successor function:
  - a value is assigned to one of the unassigned variables with no conflict
- Goal test:
  - a complete assignment
- Path cost:
  - a constant cost for each step
- Solution appears at depth n if there are n variables
- Depth-first or local search methods work well

CSP Solvers Can be Faster

- CSP solver can quickly eliminate large part of search space
- If {SA = blue}
- Then 3^5 assignments can be reduced to 2^5 assignments, a reduction of 87%



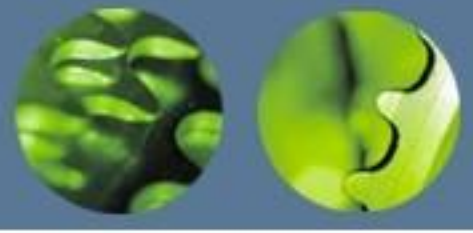
- In a CSP, if a partial assignment is not a solution, we can immediately discard further refinements of it

Types of Constraints

- Unary constraints involve a single variable,
  - e.g., SA ≠ green
- Binary constraints involve pairs of variables,
  - e.g., SA ≠ WA
- Higher-order constraints involve 3 or more variables
  - e.g., cryptarithmetic column constraints

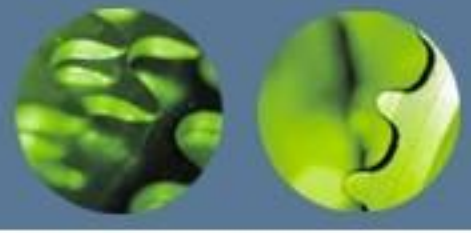
Real-World CSPs

- Assignment problems
  - e.g., who teaches what class
- Timetabling problems
  - e.g., which class is offered when and where?
- Transportation scheduling
- Factory scheduling



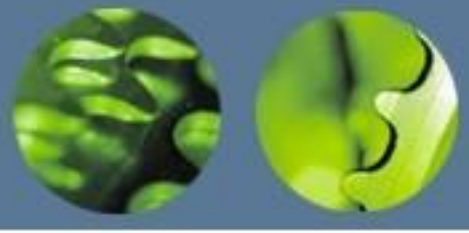
What Search Algorithm to Use?

- ✓ Since we can formulate CSP problems as standard search problems, we can apply search algorithms we discussed before.

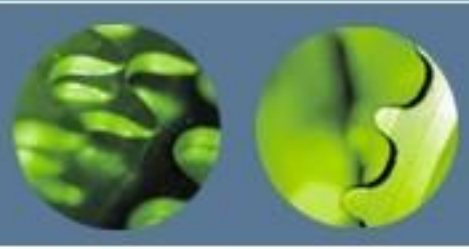


Backtracking Search

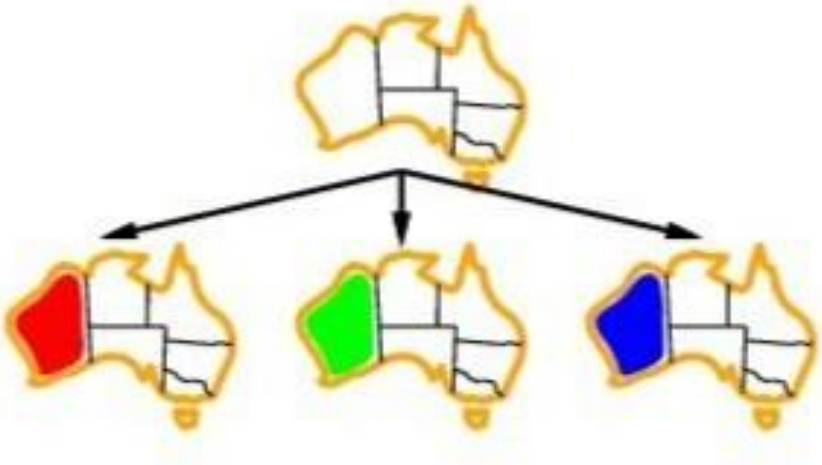
- ✓ Backtracking search is used for a depth-first search that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign.
- ✓ Backtracking search is the basic uninformed algorithm for CSPs.

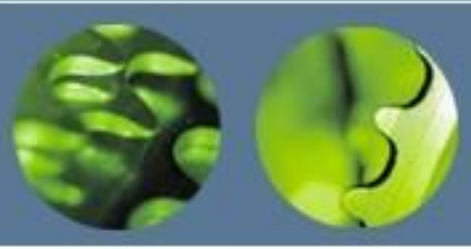


Backtracking Example

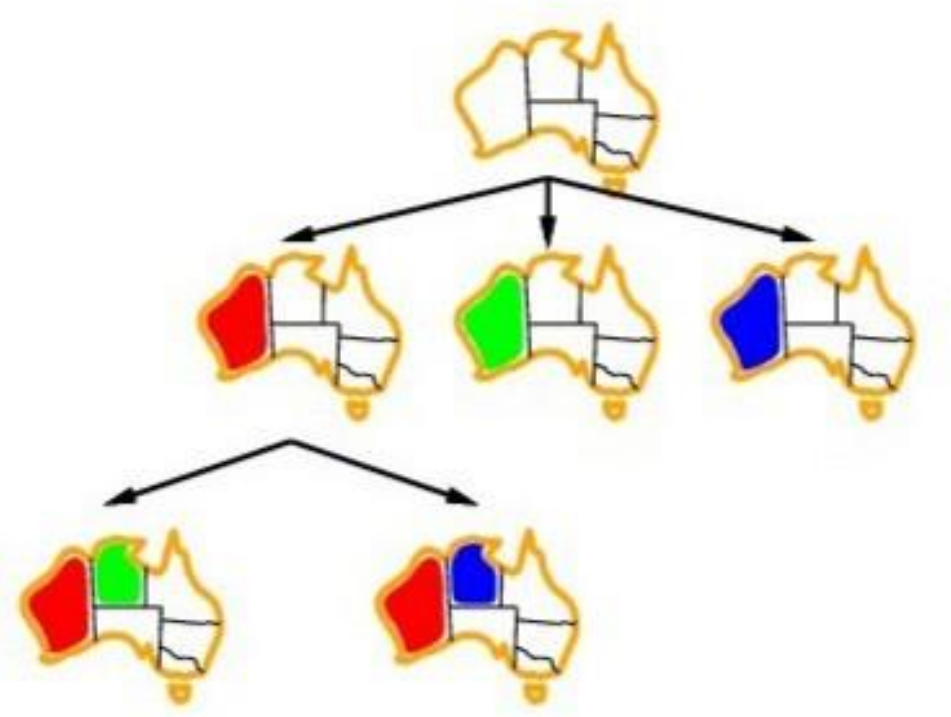


Backtracking Example

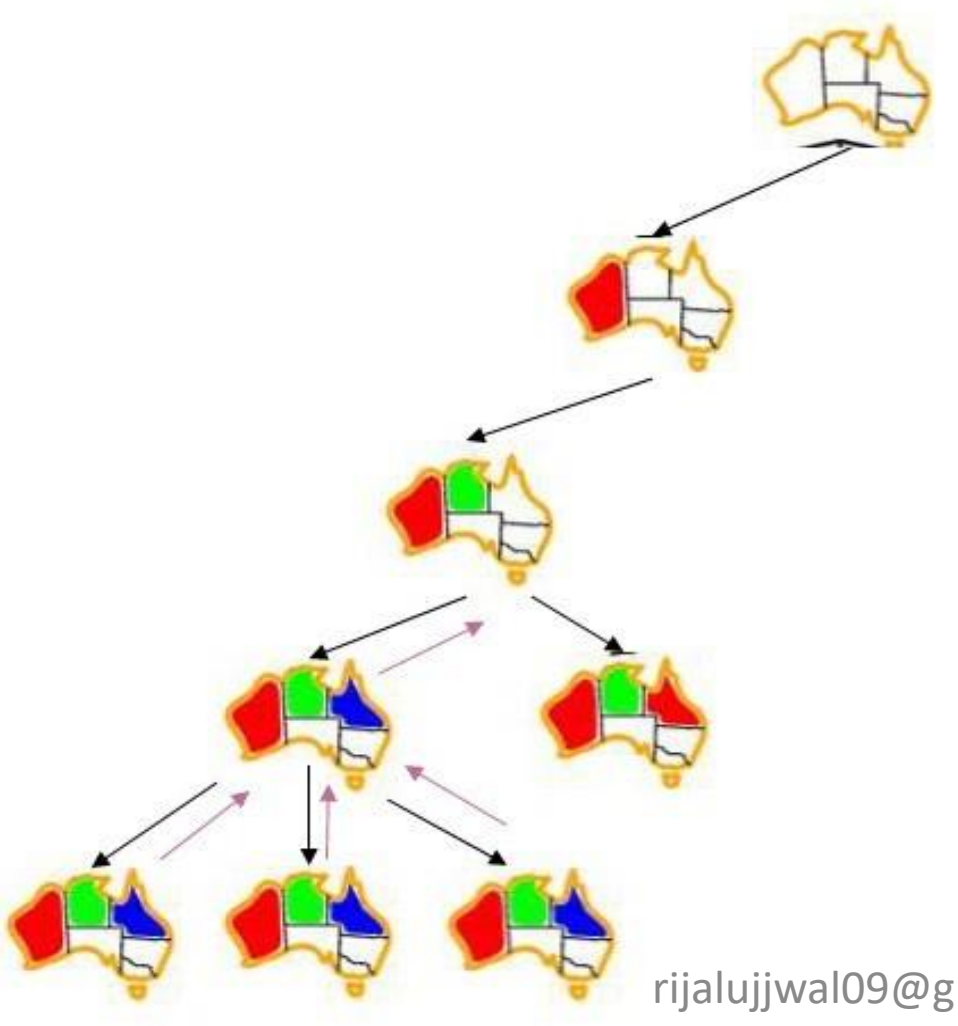


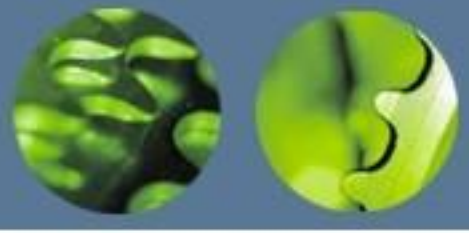


Backtracking Example



Backtracking Example



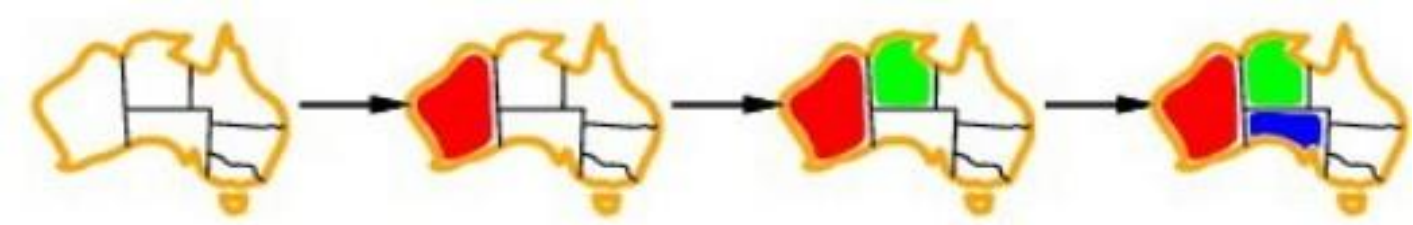


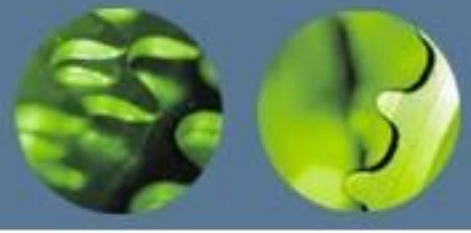
Improving Backtracking Efficiency

- ✓ We can solve CSPs efficiently without domain-specific knowledge, addressing the following questions.
  - In what order should variables be assigned, values be tried?
  - What are the implications of the current variable assignments for the other unassigned variables?
  - When a path fails, can the search avoid repeating this failure?

Variable and Value Ordering

- ✓ Minimum remaining values(MRV)
  - – choosethe variable with the fewest “legal” values
  - – also called most constrained variable or fail- first heuristic
  - – does it help in choosing the first variable?



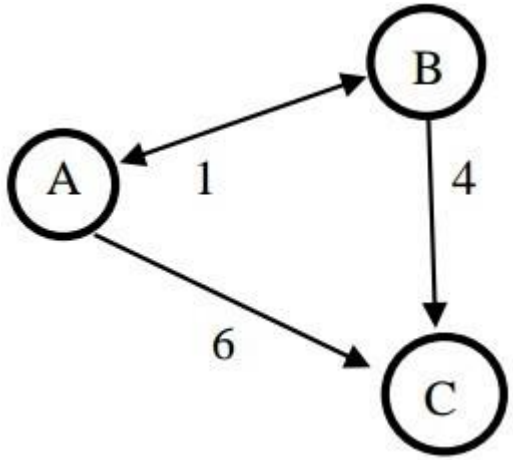


Review Questions

- ✓ Explain the uninformed search techniques with example.
- ✓ If we set the heuristic function h(n)=g(n) for both greedy as well A*. What will be effect in the algorithms? Explain?
- ✓ The mini-max algorithm returns the best move for MAX under the assumption that MIN play optimally. What happens when MIN plays sub-optimally?

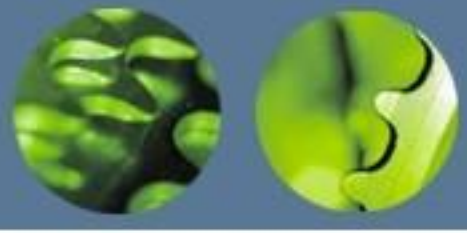
Review Questions

✓ Consider the following graph, steps cost is given on the arrow: Assume that the successors of a state are generated in alphabetical order, and that there is no repeated state checking. A is the starting node and C is goal node.



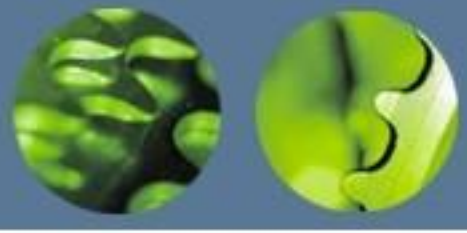
a. Of the four algorithms breadth-first, depth-first and iterative-deepening, which will find a solution in this case?

b. Write sequence of node expanding by algorithm if it finds solution.



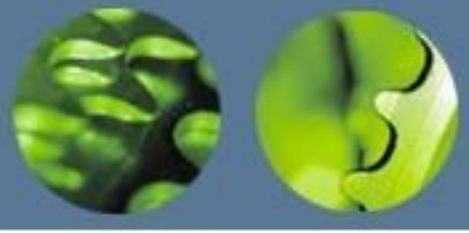
Review Questions

- ✓ Justify the searching is one of the important art of AI. Explain in detail about depth first search and breadth first search techniques with an example.
- ✓ What is meant by admissible heuristic? What improvement is done in A* search than greedy Search? Prove that A* search gives us optimal solution if the heuristic function is admissible.
- ✓ Construct a state space with appropriate heuristics and local costs. Show that greedy best first search is not complete for the state space. Also,illustrate A* is complete and guarantees solution for the same state space.



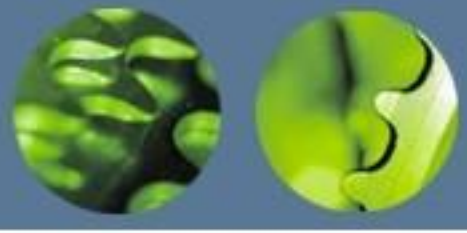
Review Questions

- ✓ Illustrate with an example, how uniform cost search algorithm can be used for finding goal in a state space.
- ✓ Justify that AI can't exist without searching. Explain in detail about any two types of informed search with practical examples.
- ✓ What is state space representation of problem? Represent the route finding problem having four cities in to state representation (you can choose any ordering of cities and links) and devise the complete problem formulation.



Review Questions

- ✓ What is heuristic information? Suppose that we run a greedy search algorithm with h(n) – g(n) and h(n) = g(n). What sort of search will the greedy search follow in each case?



Assignment #3

1) The mini-max algorithm returns the best move for MAX under the assumption that MIN play optimally. What happens when MIN plays sub-optimally?
2) What is state space representation of problem? Represent the route finding problem having four cities in to state representation (you can choose any ordering of cities and links) and devise the complete problem formulation.
3) What is meant by admissible heuristic? What improvement is done in A* search than greedy Search? Prove that A* search gives us optimal solution if the heuristic function is admissible.