Problem Solving and Search

Outline

- ❖ Problem-solving agents
- ❖ Problem Formulation
- ❖ Blind Search
- ❖ Heuristic Search

Problem-Solving Agents

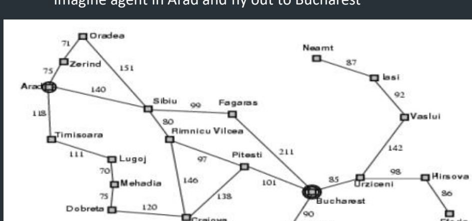
- ❖ They are a kind of goal-based agent.
- ❖ They decide what to do by finding sequences of actions that lead to desirable states.
- ❖ Intelligent agents are supposed to **maximize** their performance measure.
- ❖ This can be simplified if the agent can adopt a **goal** and aim at satisfying it.
- ❖ **Goal formulation**, based on the current situation and the agent's performance measure, is the first step in problem solving.
- ❖ **Goal** is set of states. The agent's task is to find out which sequence of actions will get it to a goal state.
- ❖ **Problem formulation** is the process of deciding what sorts of actions and states to consider, given a goal.

Problem-Solving Agents Cntd…

- ❖ Once we formulate the problem, we need to solve them. This is done by search through state space. Looking for best sequence to achieve the goal is **search**. A search algorithm takes a problem as input and returns a **solution** in the form of action sequence.
- ❖ **Initial State:** It is the starting state or initial step of the agent towards its goal.
- ❖ **Actions:** It is the description of the possible actions available to the agent.
- ❖ **Goal Test:** It determines if the given state is a goal state.
- ❖ **Path cost:** It assigns a numeric cost to each path that follows the goal. An optimal solution has the lowest path cost among all the solutions.
- ❖ One a solution is found the actions it recommends can be carried out – **execution phase**
- ❖ **"formulate, search, solution, execute"** design for the agent.

Problem-Solving Agents Example

Imagine agent in Arad and fly out to Bucharest



Problem-Solving Agents Example Cntd…

On holiday in Romania; currently in Arad.

- ❖ **Goal Formulation**
  To reach Bucharest
- ❖ **Problem Formulation**
  states: various cities
  actions: drive between cities
- ❖ **Search: Problem Space** – It is the environment in which the search takes place.
- ❖ **Solution**
  sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest

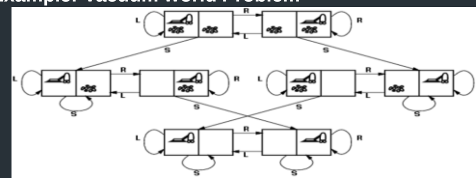
Problem Formulation:

Problem is collection of information that the agent will use to decide what to do. A problem can be defined formally by the following components:

- Initial state
- Possible actions
- Goal test
- Path cost
- Solution
- Optimal Solution

Problem solving approach

Example: Vacuum world Problem



State space graph

Vacuum world: Problem solving approach

States: The agent is in one of two locations, each of which might or might not contain dirt – 8 possible states

Initial state: any state

Possible actions: *Left*, *Right*, *Suck*

Goal test: no dirt at all locations

Path cost: 1 per action

8-Queens Problem: Problem solving approach

States: Any arrangement of 0 to 8 queens on the board

Initial state: No queens on the board

Possible actions: Add a queen to an empty square

Goal test: 8 queens on the board and none are attacked

Path cost: 1 per action

Problem solving approach: To travel from Arad to Bucharest

Initial State
e.g. "At Arad"

Successor Function/Possible Action
A set of action state pairs
S(Arad) = e.g. {<Go(Sibiu),In(Sibiu)>, <Go(Timisoara),In(Timisoara)>, <Go(Zerind), In(Zerind)>}

Goal Test
-whether a given state is a goal state
e.g. x = "at Bucharest"

Path Cost
sum of the distances traveled
e.g. Time to go Bucharest

8-Puzzle Problem: Problem solving approach

2	1	3
4	8	-
7	6	5

Start State

1	2	3
4	5	6
7	8	-

Goal State

States: location of each of the eight tiles and blank in one tile

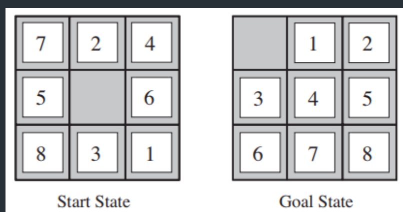
Initial state: any state

Possible actions: *blank moves left, right, up, down*

Goal test: given

Path cost: 1 per action

8



8 Puzzle

- State spaces
  - Location of each tile in blank space = n! / 2
- Initial state = Any state in state space
- Actions
  - Left, Right, Up, Down depending on blank space
- Transition state –
  - Given a state and action → returns the resulting state
- Goal test – Match with the goal state

8 Puzzle

- Path Cost - Sum of step cost (1)
- Abstraction - ??
  - Ignoring the intermediate locations where the block is sliding.
  - Shaking the board when pieces get stuck.
  - Ruled out extracting the pieces with a knife and putting them back again.

- Starting with the number 4,
  - A sequence of factorial, square root, and floor operations will reach any desired positive integer.

$$\left\lfloor \sqrt{\sqrt{\sqrt{\sqrt{\sqrt{(4!)!}}}}} \right\rfloor = 5 \, .$$

Problem formulation(3)

- States – Positive number
  - Infinite state spaces
- Initial state - 4
- Action –
  - Apply factorial, square root, floor operation
    - Factorial for integers only
- Transition Model
  - The mathematical definitions of the operations.
- Goal test – State is the desired positive integer

Other Example Problems

Travelling Salesperson problem (TSP)

Is a famous touring problem in which each city must be visited exactly once.
Goal: shortest tour

VLSI Design

Positioning million of components and connections on a chip to minimize circuit delay, maximize manufacturing yield.

Searching for Solution: Search Strategies

The performance measure of an algorithm should be measured. There are four ways to measure the performance of an algorithm.

Completeness: It measures if the algorithm guarantees to find a solution (if any solution exist).

Optimality: It measures if the strategy searches for an optimal solution.

Time Complexity: The time taken by the algorithm to find a solution.

Space Complexity: Amount of memory required to perform a search.

Searching for Solution: Search Strategies

There are two types of strategies that describe a solution for a given problem:

- **Uninformed Search (Blind Search)**
- **Informed Search (Heuristic Search)**

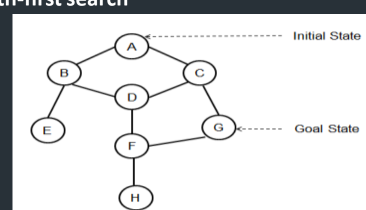
Uninformed Search (Blind Search)

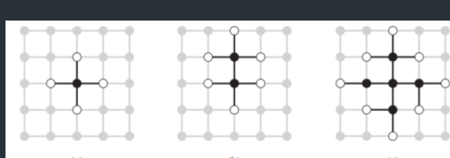
strategies use only the information available in the problem definition.

There are following types of uninformed searches:

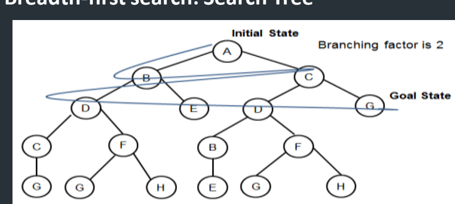
- Breadth-first search
- Depth-first search
- Uniform cost search
- Depth-limited search
- Iterative deepening search

Breadth-first search





Breadth-first search: Search Tree



Breadth-first search Algorithm

```
Let fringe be a list containing the initial state

Loop
        if fringe is empty return failure
        Node ← remove-first (fringe)
        If Node is a goal
                then return the path from initial state to Node
        else generate all successors of Node and
                add the newly generated nodes to the back of fringe
End
```

Note: Fringe: Collection of nodes that got generated but not yet expanded.

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
     if EMPTY?(frontier) then return failure
     node ← POP(frontier)  /* chooses the shallowest node in frontier */
     add node.STATE to explored
     for each action in problem.ACTIONS(node.STATE) do
        child ← CHILD-NODE(problem, node, action)
        if child.STATE is not in explored or frontier then
           if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
           frontier ← INSERT(child, frontier)
```

Evaluating BFS against four criteria

Complete? : Yes, provided the branching factor is finite

Optimal? : Yes, BFS will search the goal node even in the lowest level, if step cost is 1

Number of nodes generated: $1 + b + b^2 + b^3 + \ldots + b^d$
Hence,

Time Complexity : $O(b^d)$, time to search the entire tree
Space Complexity : $O(b^d)$, no.of nodes generated till depth d

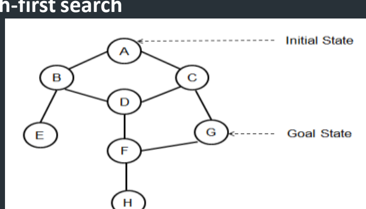
b: branching factor and
d: depth of the shallowest node
m: max depth of the search tree

Time and Memory Requirements

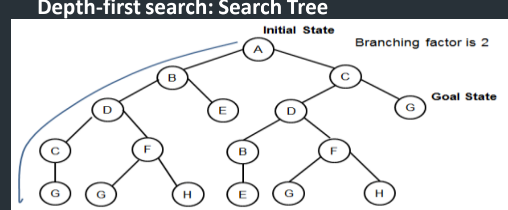
Depth	Nodes	Time		Memory	
0	1	1	millisecond	100	kbytes
2	111	0.1	second	11	kilobytes
4	11,111	11	seconds	1	megabyte
6	10^6	18	minutes	111	megabytes
8	10^8	31	hours	11	gigabytes
10	10^{10}	128	days	1	terabyte
12	10^{12}	35	years	111	terabytes
14	10^{14}	3500	years	11,111	terabytes

Time and memory requirements for breadth-first search, assuming a branching factor of 10, 100 bytes per node and searching 1000 nodes/second

Depth-first search



Depth-first search: Search Tree



Depth-First Search Algorithm

Let fringe be a list containing the initial state

Loop
if fringe is empty return failure
Node ← remove-first (fringe)
If Node is a goal
then return the path from initial state to Node
else generate all successors of Node and
add the newly generated nodes to the front of fringe
End

Evaluating DFS against four criteria

b: branching factor
d: depth of shallowest goal node
m: maximal depth of a leaf node

Number of nodes generated (worst case):
$1 + b + b^2 + \ldots + b^m = O(b^m)$

Time complexity: $O(b^m)$, to explore the entire search space

Space complexity: O(bm), Only a single path from root to goal node

Not complete and optimal if the search tree is infinite

Depth Limited Search

Depth First Search with depth limit L.
Nodes at depth L have no successors.
Choice of depth parameter is important.

Too deep is wasteful of time and space
Too shallow and we may never reach a goal state

Depth Limited Search – Evaluation Criteria

Space requirements are O(bl)

Time requirements are O(bl)

DLS is not optimal and complete

it also introduces an additional source of incompleteness if we choose L < d

Happens when d is unknown.

It be non-optimal if we choose L > d.

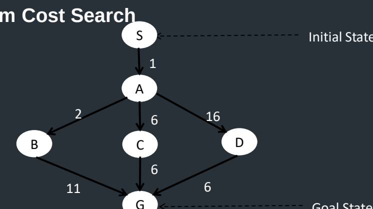
Uniform Cost Search

Uniform Cost Search works by expanding the node n with the lowest path cost.

Note: BFS will expand the shallowest unexpanded node.

Hence in uniform cost search if the step cost are equal then it is BFS.

Uniform Cost Search

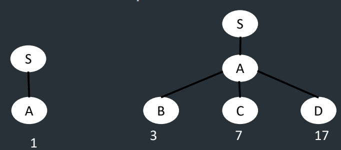


Uniform Cost Search

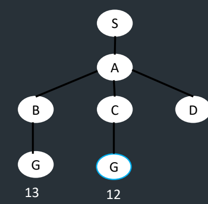
The cost of the path to each node N is
g(N) = Σ costs of arcs

The goal is to generate a solution path of minimal cost

The nodes N in the queue FRINGE are sorted in increasing g(N)



Uniform Cost Search



Properties of uniform cost search

Complete?
Yes, if step cost is greater than some positive constant ϵ

Optimal?
Yes –nodes expanded in increasing order of path cost

Time?
Number of nodes with path cost ≤ cost of optimal solution (C^*)

$O(b^{C^*/\epsilon})$

This can be greater than $O(b^d)$:

Space?

$O(b^{C^*/\epsilon})$

Iterative Deepening Search

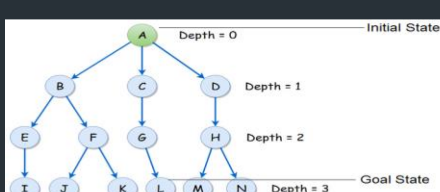
Provides the best of both breadth-first and depth-first search

It does this by increasing the depth limit with each iteration until it reaches d, the depth of shallowest goal node.

Algorithm:
Consider DFS as the procedure

1. Check the root
2. Do a DFS search for a path length of 1
3. If there is no path of length 1, do a DFS search for path length of 2
4. If there is no path of length 2, do a DFS search for path length of 3
5. Continue this until you reach the path

Iterative Deepening Search: Example



Iterative Deepening Search: Example

Solution

Depth	IDS Traversal
0	A
1	A B C D
2	A B E F C G D H
3	A B E I F J K C G **L** D H M N

Goal State

Iterative Deepening Search: Evaluation

Iterative deepening search is:
- Complete
- Optimal if step cost =1

Time complexity is:
$(d+1)(1) + db + (d-1)b^2 + \dots + (1)\, b^d = O(b^d)$

Space complexity is: O(bd) or O(d)

Note: Uninformed search methods - less efficient -exponential time complexity.

Informed Search (Heuristic Search)

Informed Search methods use problem specific knowledge.
They are more efficient.
Use heuristic to identify the best search path

Heuristic

-rule of thumb
It is the criteria, methods or principles for deciding which among several alternative courses of action promises to be the most effective in order to achieve goal.

Example: Euclidean distance between to points
8 puzzle problem: Number of tiles out of place

Heuristic Example

8 Puzzle Problem

8	2	3
1	6	4
	7	5

Initial State

1	2	3
8		4
7	6	5

Goal State

Heuristic function value= Number of tiles out of place= 5

Informed Search (Heuristic Search)

There are following types of informed searches:

- Best first search
- A* Search

Best first search

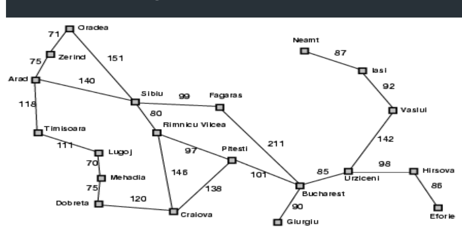
Idea: Node selection - heuristic function, h(n)
Node expansion based on smallest or largest h(n) w.r.t the problem requirement

Algorithm

Consider FRINGE as PQ (Priority Queue) with Initial-State
Loop Do
If FRINGE empty **then** return failure
  Node <- Remove-First (FRINGE)
If Node is Goal-State
  then return solution (path: Initial State to Node)
Else add the generated nodes to the FRINGE based on f(n)
End Loop

Special cases
Greedy Search/Greedy Best First Search
A* Search

Example for Greedy Best First Search: To find the best cost to travel from Arad to Bucharest with straight-line dist.



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy Best first search

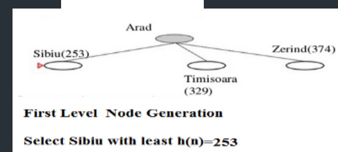
Greedy best-first search expands the node with smallest estimated cost to reach goal.

h(n) = heuristic function= estimate of cost from *n* to *goal*
e.g., $h_{SLD}(n)$ = straight-line distance from *n* to Bucharest

Greedy Search Example: Solution

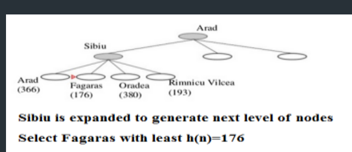
i)

ii)



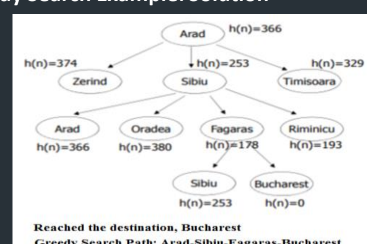
Greedy Search Example: Solution

iii)



Greedy Search Example: Solution

iv)



Properties of greedy best-first search

Optimality: Arad →Sibiu →Fagaras → Bucharest is not
Complete: No – can get stuck in loops
Time and Space Complexity: $O(b^m)$
'm' is the depth of search space

Note: Arad→Sibiu→Rimnicu Vilcea→Pitesti→Bucharest is shorter!

A* Search

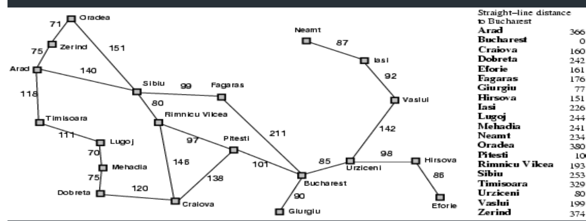
Proposed by Hart, Nilsson and Rafael in 1968

Best first search with f(n)=g(n)+h(n)
g(n)= sum of edge costs from start to n
h(n)= estimate of lowest cost path from n to goal

A* Search Algorithm

```
open = nodes on frontier; closed = expanded nodes;
open = {<s,nil>}
While open is not empty{
  remove from open the node <n,p>with min. f(n)
  place <n,p> in closed
  if n is goal node , return success (path p)
  for each edge e connecting n and m with cost c
        if <m,q> is on closed and {p|e} is cheaper than q
              then remove m from closed, put <m,{p|e}> on open
        elseif <m,q> is on open and {p|e} is cheaper than q
              then replace q with {p|e}
        else if m is not in open, put <m,{p|e}> on open
}
Return failure
```

Example for A* Search: To find the best cost to travel from Arad to Bucharest with straight-line dist.



A* Search Example: Solution

i)

Arad
366=0+366

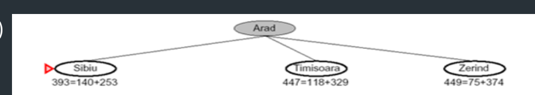
Initial State is Arad

ii)

Evaluation function, f(n)=g(n)+h(n)
=0+366
=366

A* Search Example: Solution

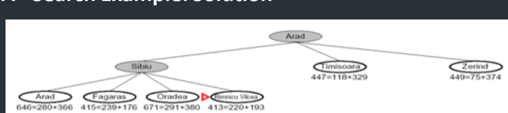
ii)



Three nodes generated from Arad
Sibiu with least f(n)=393 is selected next

A* Search Example: Solution

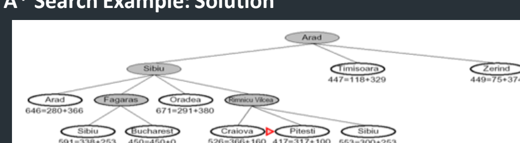
iii)



Four nodes generated from Sibiu.
Rimnicu Vilcea with least f(n)=413 is selected next

A* Search Example: Solution

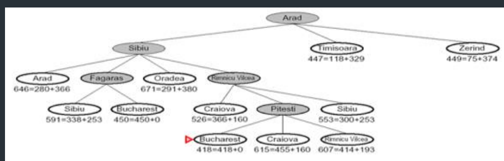
iv)



Three nodes generated from Rimnicu Vilcea
Pitesti with least f(n)=553 is selected next

A* Search Example: Solution

v)



Reached destination, Bucharest
A* Search Path: Arad-Sibiu-Rimnicu Vicea-Pitesti-Bucharest

A^* search

- Conditions for optimality:
  - Admissibility and consistency
- Admissible heuristics are by nature optimistic
  - An admissible heuristic never overestimates the cost to reach the goal,
  - Straight-line distance is admissible
    - the shortest path between any two points is a straight line
    - $h_{SLD}(n)$ - never overestimates the actual road distance

Admissible heuristics

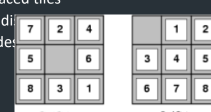
- A heuristic *h(n)* is admissible
  - If for every node *n*,
    - $h(n) \le h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from *n*.
    - Obvious fact in mathematical term.

Admissible heuristics

E.g., for the 8-puzzle:
- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan di

(i.e., no. of squares from des

- $h_1(S)$ = ?
- $h_2(S)$ = ?



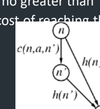
Admissible heuristics

E.g., for the 8-puzzle:
- $h_1(n)$ = number of misplaced til
- $h_2(n)$ =no. of squares from desir
- $h_1(S) = 8$

- $h_2(S) =$ 3+1+2+2+2+3+3+2 = 18

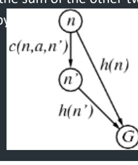
Consistency

- A heuristic h(n) is consistent if
  - For every node n
  - Every successor n' of n generated by any action
  - The estimated cost of reaching the goal from n is no greater than
  - The step cost of getting to n' plus the estimated cost of reaching the goal from n:
    h(n) ≤ c(n, a, n') + h(n')
  - The general **triangle inequality**



Consistency

- Each side of a triangle
  - Cannot be longer than the sum of the other two sides.
  - The triangle is formed by closest to n.



Consistency

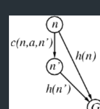
- If there were a route from n to G_n via n'
  - Cheaper than h(n),
  - It violate the Optimality property that h(n) is a lower bound on the cost to reach G_n.
    - Every consistent heuristic is also admissible.
  - Consistency is a stricter requirement than admissibility
  - General triangle inequality is satisfied when each side is measured by the straight-line distance

Optimality of A*

- *if* h(n) *is consistent,*
  - *then the values of* f(n) *along any path are nondecreasing (Increasing).*
  - The proof follows directly from the definition of consistency.
  - Suppose n' is a successor of n;
  - then g(n') = g(n) + c(n, a, n') for some action a,
  - f(n') = g(n') + h(n')

 = g(n) + c(n, a, n') + h(n')

 ≥ g(n) + h(n) = f(n)

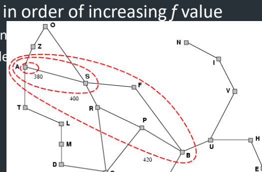


Optimality of A*

- Prove that whenever A∗ selects a node n for expansion,
  - The optimal path to that node has been found
  - If Not,
  - There would have to be another frontier node n' on the optimal path from the start node to n,
  - By the graph separation property ,
    - f nondecreasing along any path,
    - n' would have lower f-cost than n , it would have been selected first.

Contours in state space

- A^* expands nodes in order of increasing f value
  - Gradually adds "f-con
  - Contour i has all node



Performance of A*

- Complete? Yes
  - Unless there are infinitely many nodes with f ≤ *f(G)*
- Time? Exponential
- Space? Keeps all nodes in memory
- Optimal? Yes

Properties of A* Search

Optimality: Yes, if h(n) is admissible because it find optimal cost to reach goal

Complete: Yes, if branching factor is finite

Time and Space Complexity: *$O(b^d)$*
where d is depth of search space

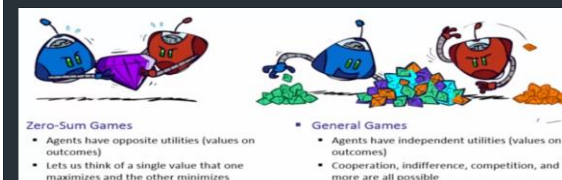
Game Problems: Types

i) Adversarial Games

-zero sum games

ii) Co-operative Games

-General games



Adversarial Games

It is deterministic, fully observable environment in which the two agents whose action must alternate and in which the utility values at the end of the game are always equal and opposite.

Ex: winning chess(+1), lose(-1)

Game example: Tic-tac-toe, chess, checkers

Adversarial Search in Game Playing

- Two player games
- Zero sum game
- Two players are MAX and MIN
- Two players-take turns-for maximizing and minimizing utility function
- MAX player take his turn first

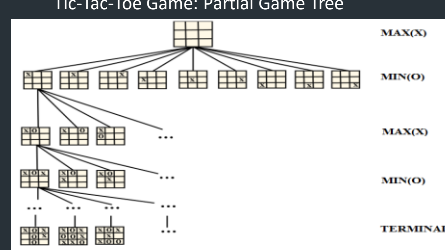
Game Problem as Search

A game can be defined as search with the following components

- **Initial State:** the board position and recognizes the player to move
- **Successor Function:** legal move a player can make and the resulting state
- **Terminal Test:** determines when the game is over
- **Utility Function:** Objective/Payoff Function
  It gives numeric value for terminal state. In chess win, loss or draw as +1,-1,0

Two Player Game Tree

Tic-Tac-Toe Game: Partial Game Tree



MINIMAX Algorithm

Generate the whole game tree

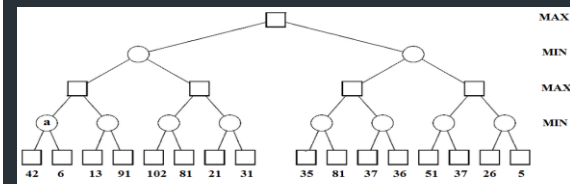
Apply the utility function to each terminal state

Use utility of the terminal states to determine the utility of the nodes one level higher up in the search tree

From bottom to top
- In max turn, choose MAX value of its successors
- In min turn, choose MIN value of its successors

Finally in ROOT node choose the move that gives MAX utility value

MINIMAX Example



MINIMAX Example

Two types of node: Max and Min Node.

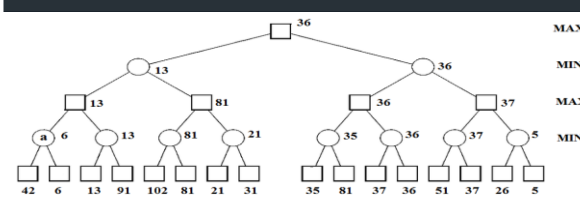
Max node's strategy is to maximize the cost/profit of player A
Min node's strategy is to minimize the cost/profit of player A

Square node: Player A move
Naught node: Opponent move

In the pervious slide, Player A will expand the states of the game and arrive with the profit he may get for that depth of the tree.

In node 'a' of previous slide the opponent gets the chance to move. So he will push to the state with cost value '6'. Hence Player cannot expect the cost/Profit better than '6' in node 'a'.

MINIMAX Example : Solution



Properties of MINIMAX

- Complete: Yes (If Game tree is finite)
- Optimality: Yes
- Time complexity: $O(b^d)$
- Space complexity: O(bd) –It is like depth first search where d is depth of tree

Alpha-Beta Pruning

It is a search algorithm that seeks to decrease the number of nodes that are evaluated by the minimax algorithm in its search tree.

Alpha-beta pruning is a way of finding the optimal minimax solution while avoiding searching subtrees of moves which won't be selected. In the search tree for a two-player game, there are two kinds of nodes, nodes representing *your* moves and nodes representing *your opponent's* moves.

Alpha-Beta Pruning

During Minimax, keep track of two additional values

Alpha

- In Max node update Alpha. Your best score via any path

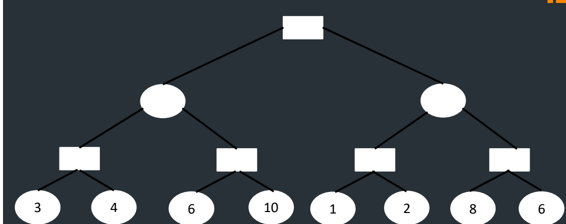
Beta

- In Min node update Beta. Opponent's best score via any path.

Alpha-Beta Pruning

- Max player (you) will never make a move that could lead to a worse score for you
- Min player (opponent) will never make a move that could lead to a better score for you
- Stop evaluating a branch whenever:
  - Beta Bound: Exploration of Max node is stopped, when the value equals or exceed Beta.
  - Alpha Bound: Exploration of Min node is stopped, when the value equals or falls below Alpha.
- Based on observation that for all viable paths utility value n will be **α <= n <= β**

Alpha-Beta Pruning Example



Alpha-Beta Pruning Example : Solution