Search Strategies
2nd class / AI Branch

م. سُـرى محمـود عبـدالله

Email: - 110050@uotechnology.edu.iq

Google Site: **Google scholar**: https://scholar.google.com/citations?user=ep015F0AAAAJ&hl=en

Research gate: https://www.researchgate.net/profile/Sura_Abdullah

What is Search?

Search is an important aspect of AI. Search can be defined as a problem-solving technique that enumerates a problem space from an initial position in search of a goal position (or solution). The manner in which the problem space is searched is defined by the search algorithm or strategy. As search strategies offer different ways to enumerate the search space. Ideally, the search algorithm selected is one whose characteristics match that of the problem at hand.

State Space Search

The **state space search** is a collection of several states with appropriate connections (links) between them. Any problem can be represented as such a space search to be solved by applying some rules with technical strategy according to the suitable intelligent search algorithm.

To provide a formal description of a problem must do the following operations:

1. Define space search **(state space)** that contains all possible states.
2. Specify one or more states within that space that describe possible situations from which the problem-solving process may start. These states are called the **initial states**.
3. Specify a set of rules that describe the actions available.
4. Specify one or more states that would be acceptable as solutions to the problem. These states are called **goal states**.
5. Determine a suitable **search strategy** to reach the goal.

To successfully design and implement search algorithms, a programmer must be able to analyze and predict their behavior. Questions that need to be answered include:

- Is the problem solver guaranteed to find a solution?
- Will the problem solver always terminate, or can it become caught in an infinite loop?
- When a solution is found, is it guaranteed to be optimal?
- What is the complexity of the search process in terms of time usage? Memory usage?
- How can the interpreter most effectively reduce search complexity?
- How can an interpreter be designed to most effectively utilize a representation language?

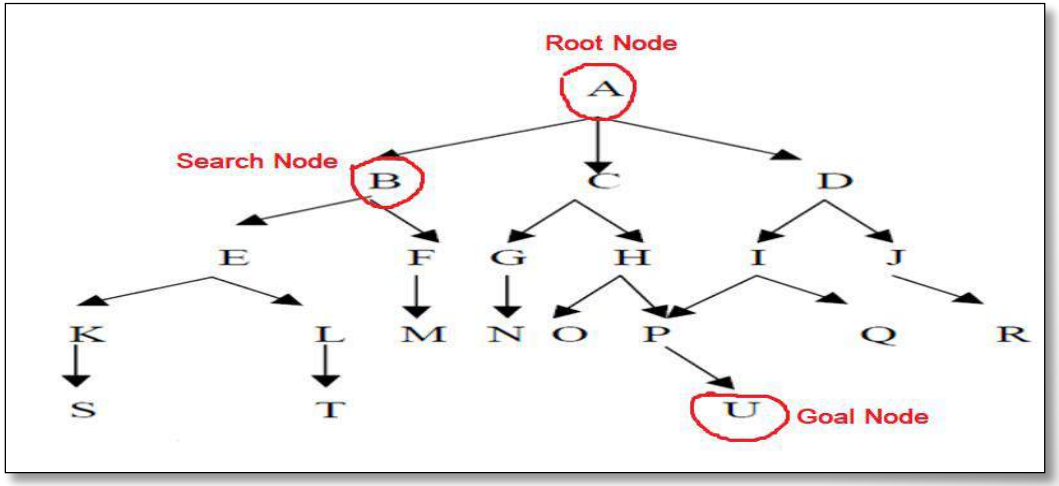
To get a suitable answer to these questions, the search can be structured into three parts.

A ***first part*** presents a set of definitions and concepts that lay the foundations for the search procedure into which induction is mapped. The ***second part*** presents an alternative approach that has been taken to induction as a search procedure and finally the ***third part*** presents the version space as a general methodology to implement induction as a search procedure.

If the search procedure contains the principles of the above three requirement parts, then the search algorithm can give a guarantee to get an optimal solution for the current problem.

Some common terms in the searching issues

- **Search Tree:** is a tree in which the root node is the start state and has a reachable set of children.
- **Search Node:** is a node in the search tree.
- **Goal:** is a state that an agent is trying to reach.
- **Action:** is something (operators, possible moves, rules) that an agent can choose to do.
- **Branching Factor:** is the number of actions available to the agent.



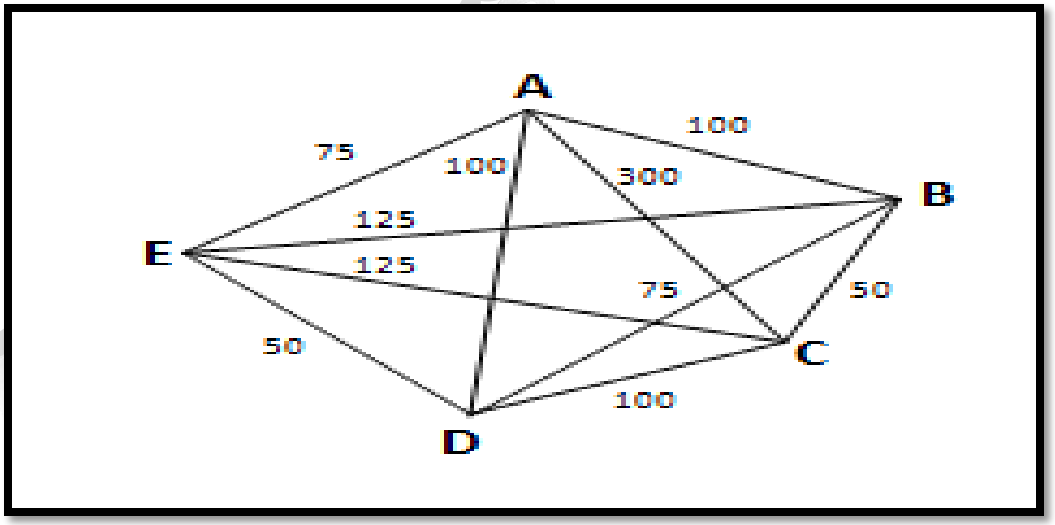
Traveling Salesman Problem (TSP)

The TSP concept depends on finding a path for a specified number of cities (visiting all cities only once and returning to the city that started with) where the distance of the path is optimized by finding the shortest path with minimized cost.

Example: The below figure shows a full connected graph, (A,B,C,etc) are cites and the numbers associated with the links are the distances between the cities.

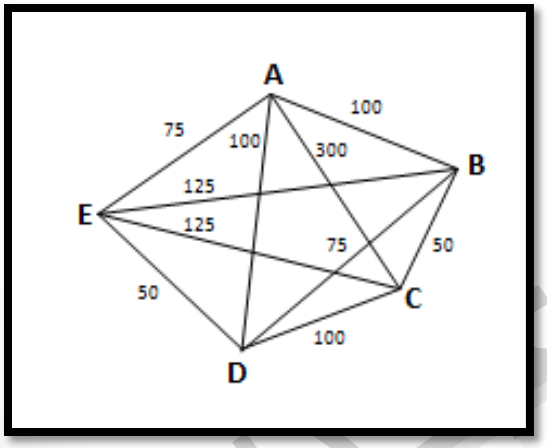
Starting at A, find the shortest path through all the cities, visiting each city exactly once returning to A.

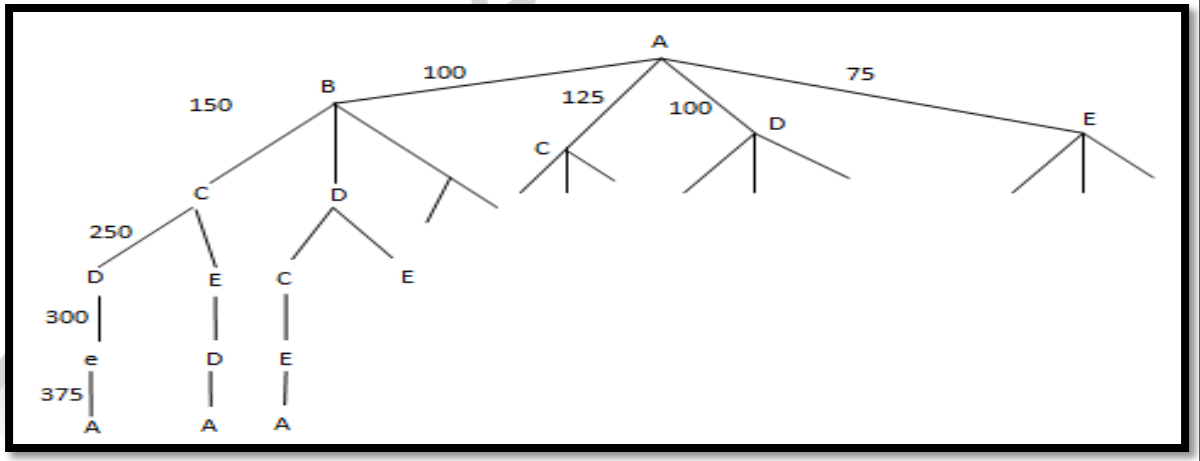
ملاحظه ذكر الرسم يكون غير موجهه



- The complexity of exhaustive search in the traveling Salesman is (N-1)! , where N is the No. of cities in the graph. There are several techniques that reduce the search complexity:

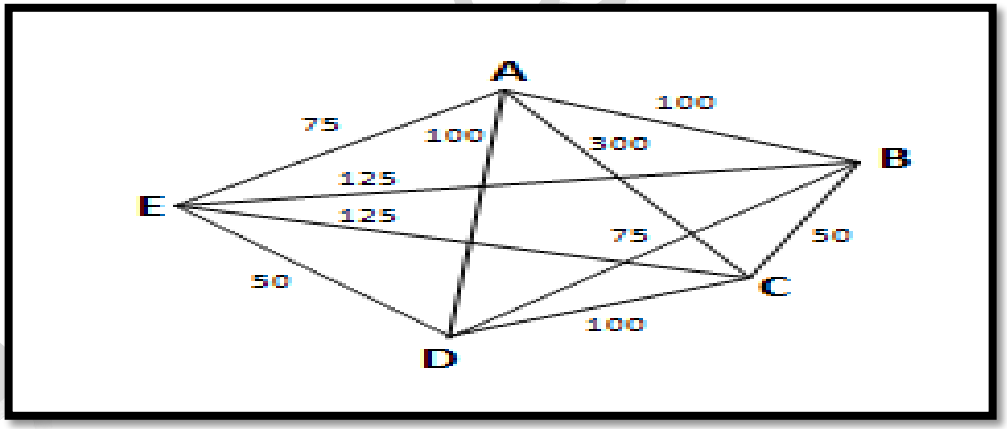
1- Branch and Bound Algorithm: Generate one path at a time, keeping track of the best circuit so far. Use the best circuit so far as a bound of future branches of the search. Figure below illustrate branch and bound algorithm.





A B C D E A= 375
A B C E D A= 425
A B D C E A= 474 ……

2- Nearest Neighbor Heuristic: At each stage of the circuit, go to the nearest unvisited city. This strategy reduces the complexity to N, so it is highly efficient, but it is not guaranteed to find the shortest path, as the following example:



Distance of nearest neighbor path is A E D B C A=550
Is not the shortest path, the comparatively high distance of arc (C, A) defeated the heuristic.

Search Techniques Types

Usually, types of intelligent search are classified into three classes; ***Blind***, ***Heuristic*** and ***Random search***.

1- Blind Search is a technique to find the goal without any additional information that helps to infer the goal, with this type there is no consideration with process time or memory capacity.

2- Heuristic Search always has an evaluating function called the heuristic function which guides and controls the behavior of the search algorithm to reach the goal with minimum cost, time and memory space.

3- Random Search is a special type of search in which it begins with the initial population that is generated randomly and the search algorithm will be the responsible for generating the new population based on some operations according to a special type function called fitness function.

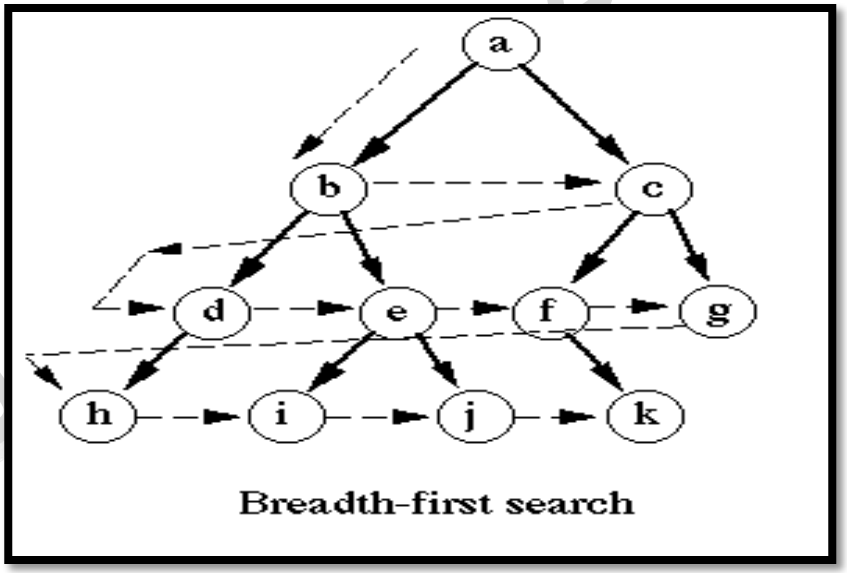
Blind Search

- The blind search strategies (also called **uninformed search**) don't have any additional information about states beyond that provided in the problem definition. All they can do is generate successors and distinguish a goal state from a non-goal state.
- Thus blind search strategies have not any previous information about the goal nor the simple paths lead to it. However blind search is not bad since more problems or applications need it to be solved; in other words, there are some problems give good solutions if they are solved by using blind search strategies.
- This type of search takes all nodes (states) of the search tree in a specific order until it reaches to goal. The order can be in breadth and the strategy will be called breadth-first search, or in depth and the strategy will be called depth-first search.

Blind Search

1- Breadth – First – Search Algorithm

In breadth-first search, when a state (node) is examined, all of its siblings are examined before any of its children. The space is searched level-by-level, proceeding all the way across one level before doing down to the next level.



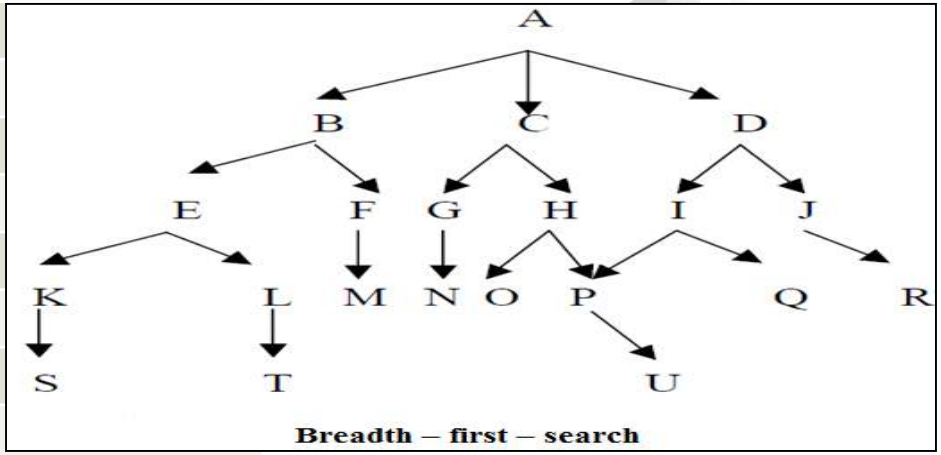
Breadth-first search

[open]	[closed]
[A]	[]
[B,C,D]	[A]
[C,D,E,F]	[B,A]
[D,E,F,G,H]	[C,B,A]
[E,F,G,H,I,J]	[D,C,B,A]
[F,G,H,I,J,K,L]	[E,D,C,B,A]
[G,H,I,J,K,L,M]	[F,E,D,C,B,A]
[H,I,J,K,L,M,N]	[G,F,E,D,C,B,A]
[I,J,K,L,M,N,O,P]	[H,G,F,E,D,C,B,A]
[J,K,L,M,N,O,P,Q]	[I,H,G,F,E,D,C,B,A]
[K,L,M,N,O,P,Q,R]	[J,I,H,G,F,E,D,C,B,A]
[L,M,N,O,P,Q,R,S]	[K,J,I,H,G,F,E,D,C,B,A]
[M,N,O,P,Q,R,S,T]	[L,K,J,I,H,G,F,E,D,C,B,A]
[N,O,P,Q,R,S,T]	[M,L,K,J,I,H,G,F,E,D,C,B,A]
[O,P,Q,R,S,T]	[N,M,L,K,J,I,H,G,F,E,D,C,B,A]
[P,Q, R,S,T]	[O, N,M,L,K,J,I,H,G,F,E,D,C,B,A]
[Q, R,S,T,U]	[P, O, N,M,L,K,J,I,H,G,F,E,D,C,B,A]
[R,S,T,U]	[Q, P, O, N,M,L,K,J,I,H,G,F,E,D,C,B,A]
[S,T,U]	[R,Q, P, O, N,M,L,K,J,I,H,G,F,E,D,C,B,A]
[T,U]	[S, R,Q, P, O, N,M,L,K,J,I,H,G,F,E,D,C,B,A]
[U] **Goal**	[T, S, R,Q, P, O, N,M,L,K,J,I,H,G,F,E,D,C,B,A]
[]	[U,T, S, R,Q, P, O, N,M,L,K,J,I,H,G,F,E,D,C,B,A]

For Example

Start State: A

Goal State: U



Breadth – first – search

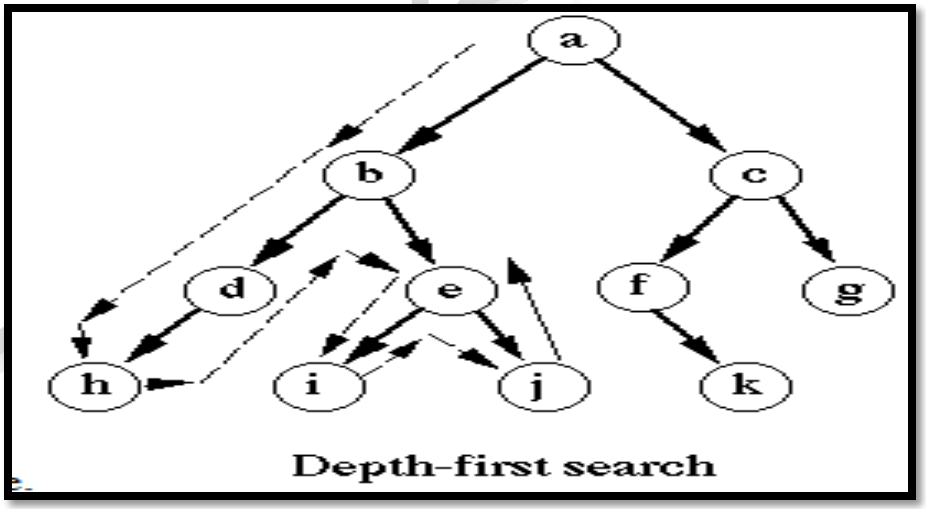
Final Path: A→B→C→D→E→F→G→H→I→J→K→L→M→N→O→P→Q→R→S→T→U

Breadth – First – Search Algorithm

Begin
Open: = [start];
Closed: = [ ];
While open ≠[ ] do
Begin
Remove left most state from open, call it x;
If x is a goal then return (success)
Else
Begin
Generate children of x;
Put x on closed;
Eliminate children of x on open or closed;
Put remaining children on right end of open
End
End
Return (failure)
End.

Blind Search
2- Depth – First – Search Algorithm

In a depth-first search, when a state is examined, all of its children and their descendants are examined before any of its siblings. The depth-first search goes deeper into the search space whenever this is possible only when no further descendants of a state can found owe it.



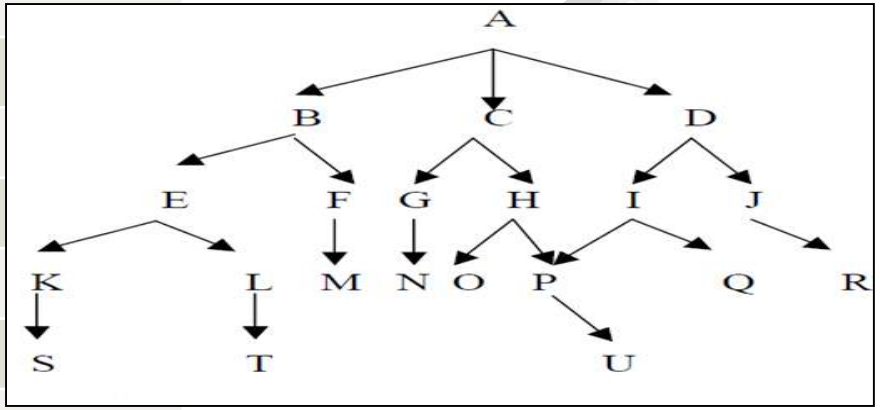
Depth-first search

[open]	[closed]
[A]	[]
[B,C,D]	[A]
[E,F,C,D]	[B,A]
[K,L,F,C,D]	[E,B,A]
[S,L, F,C,D]	[K,E,B,A]
[L, F,C,D]	[S,K,E,B,A]
[T, F,C,D]	[L,S,K,E,B,A]
[F,C,D]	[T,L,S,K,E,B,A]
[M,C,D]	[F,T,L,S,K,E,B,A]
[C,D]	[M,F,T,L,S,K,E,B,A]
[G,H,D]	[C,M,F,T,L,S,K,E,B,A]
[N,H,D]	[G,C,M,F,T,L,S,K,E,B,A]
[H,D]	[N,G,C,M,F,T,L,S,K,E,B,A]
[O,P,D]	[H,N,G,C,M,F,T,L,S,K,E,B,A]
[P,D]	[O,H,N,G,C,M,F,T,L,S,K,E,B,A]
[**U**,D]	[P,O,H,N,G,C,M,F,T,L,S,K,E,B,A]
[D]	[U,P,O,H,N,G,C,M,F,T,L,S,K,E,B,A]

For Example

Start State: A

Goal State: U



Final Path: A→B→E→K→S→L→T→F→M→C→G→N→H→O→P→U

Depth – First – Search Algorithm

Begin
Open: = [start];
Closed: = [ ];
While open ≠ [ ] do
Remove leftmost state from open, call it x;
If x is a goal then return (success)
Else
begin
Generate children of x;
Put x on closed;
Eliminate children of x on open or closed;
put remaining children on left end of open
End;
Return (failure)
End.

Heuristic Search

The heuristic search strategies (also called **Informed Search**). A **heuristic** is a method that might not always find the best solution but is guaranteed to find a good solution in reasonable time. By sacrificing completeness it increases efficiency. Heuristic search is useful in solving problems which:-

- Could not be solved any other way.
- Solution takes an infinite time or very long time to compute.

There are several methods for heuristic search, from these are:-

1- Hill Climbing.

2- Best-First Search.

3- A algorithm.

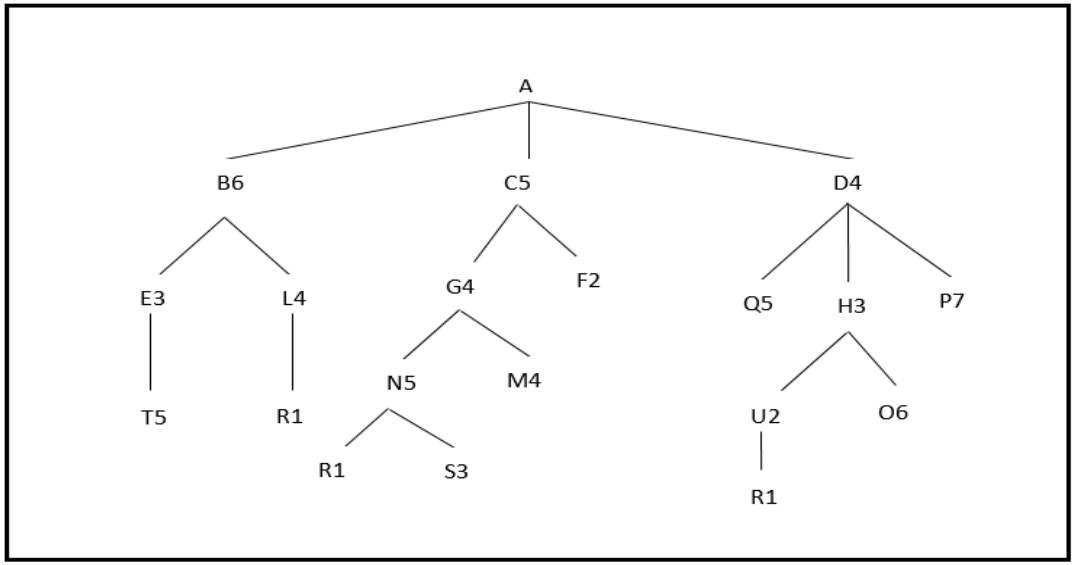
4- A* algorithm.

Heuristic Search
1) Hill Climbing

- The idea here is that you don't keep the big list of states around. You just keep track of the one state you are considering, and the path that got you there from the initial state. At every state, you choose the state leads you closer to the goal according to the **heuristic estimate,** and continue from there.
- The name "**Hill Climbing**" comes from the idea that you are trying to find the top of a hill, and you go in the direction that is up from wherever you are. This technique often works, but since it only uses local information.

Example1:

Search for **R** with local minima from **A**.

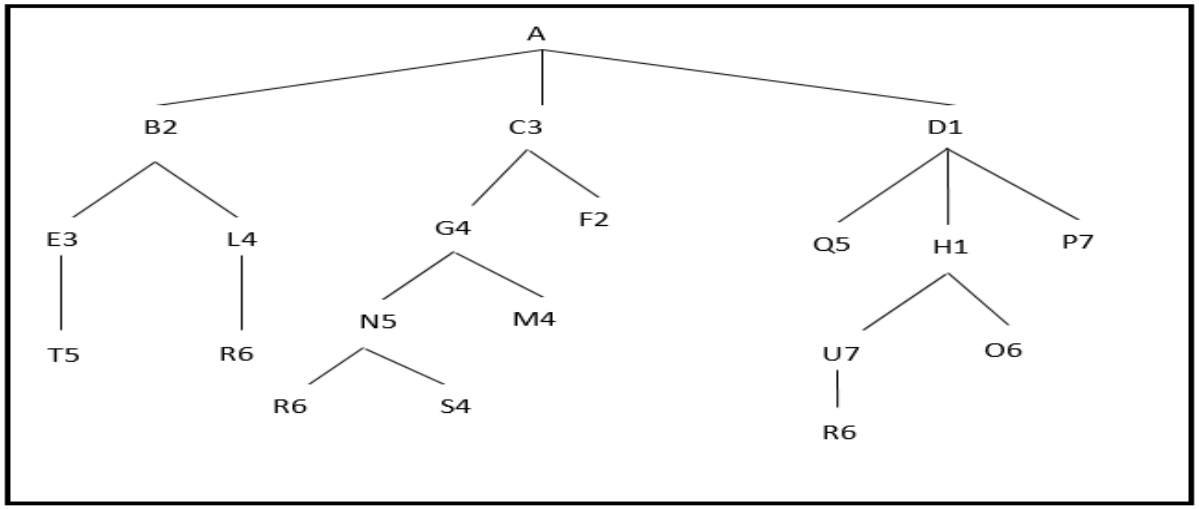


[open]	[closed]	X
[A]	[]	A
[D4,C5,B6]	[A]	D4
[H3,Q5,P7]	[D4,A]	H3
[U2,O6]	[H3,D4,A]	U2
[R1]	[U2,H3,D4,A]	R1
[]	[R1,U2,H3,D4,A]	

Final Path: A→D4→H3→U2→R1

Example2:

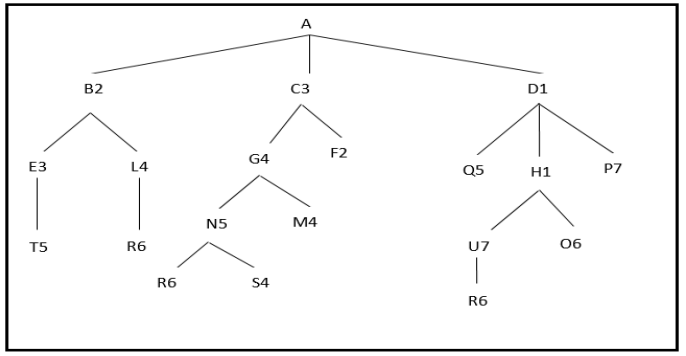
Search for **R** with local maxima from **A**.



[open]	[closed]	X
[A]	[]	A
[C3,B2,D1]	[A]	C3
[G4,F2]	[C3,A]	G4
[N5, M4]	[G4,C3,A]	N5
[R6,S4]	[N5,G4,C3,A]	R6
[S4]	[R6,N5,G4,C3,A]	

Final Path: A→C3→G4→N5→R6

Hill Climbing Algorithm

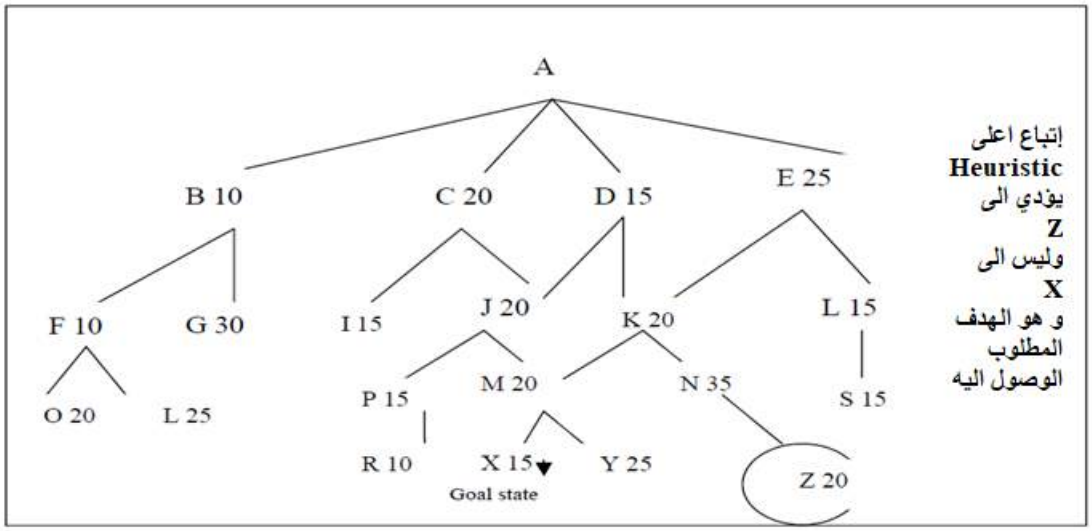


Begin
Cs=start state;
open=[start];
stop=false;
Path=[start];
While (not stop) do
{
if (Cs=goal) then
return (path);
generate all children of Cs and put them into open
if (open=[]) then
stop=true
else
{
X:= Cs;
for each state in open do
{
compute the heuristic value of Y (h(Y));
if Y is better than X then
X=Y
}
if X is better than Cs then
Cs=X
else
stop =true;
}
}
return (failure);
end

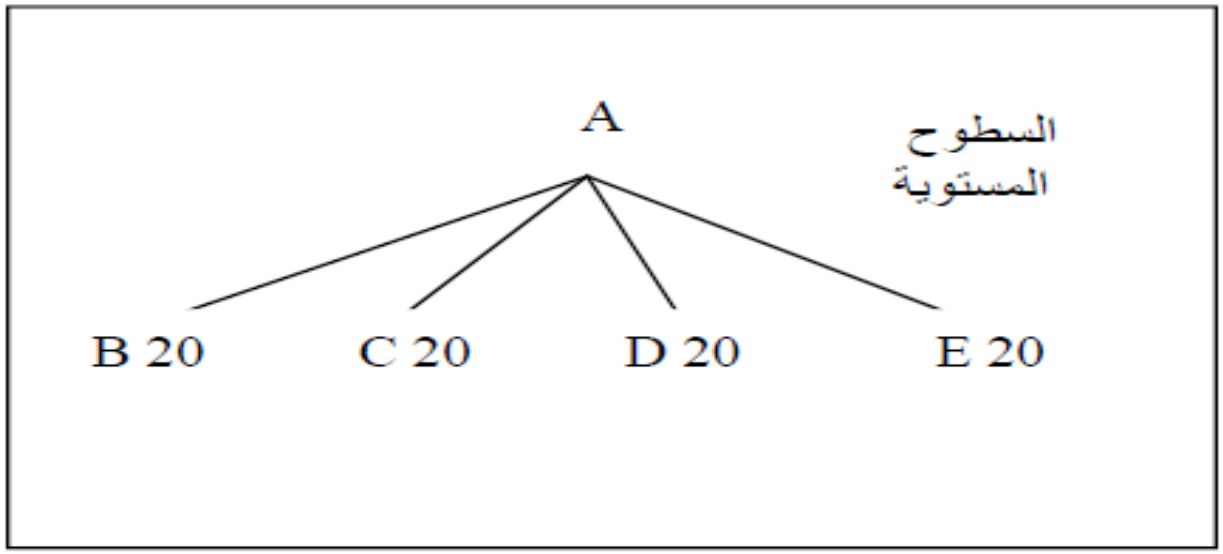
Hill Climbing Problems

Hill climbing may fail due to one or more of the following reasons:-

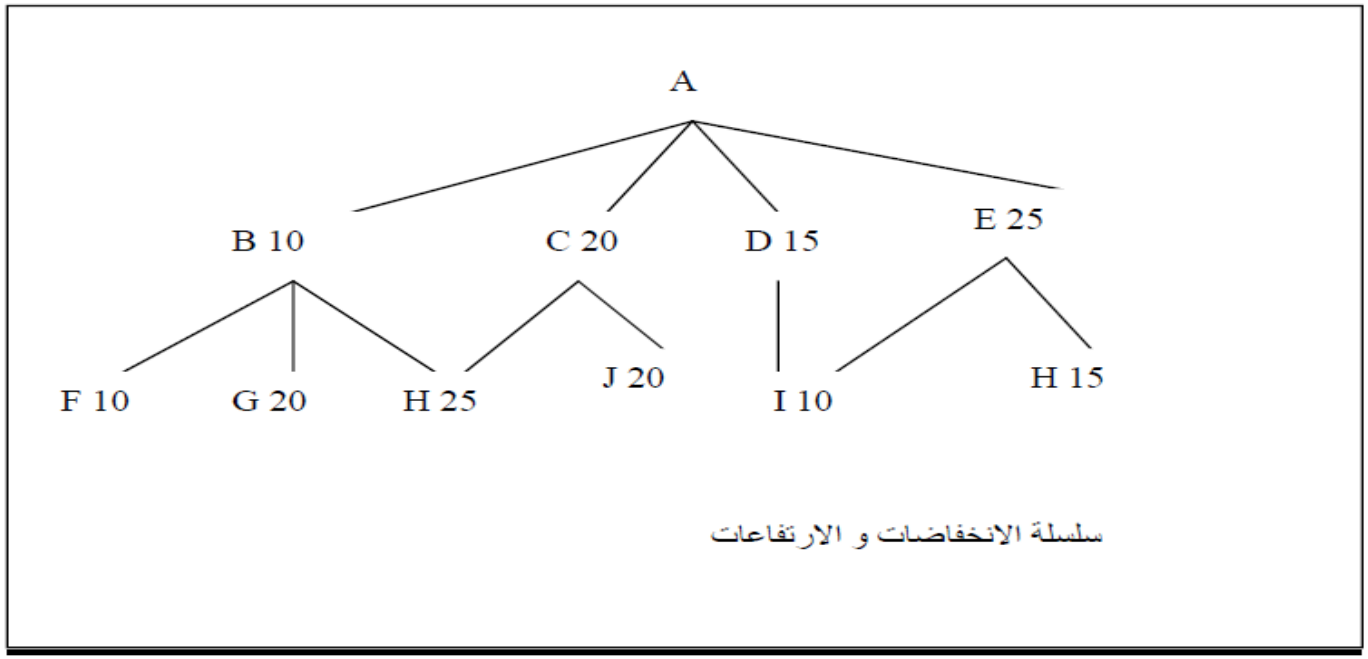
1- A local maxima : Is a state that is better than all of its neighbors but is not better than some other states.



2- A Plateau: Is a flat area of the search space in which a number of states have the same best value, in plateau it's not possible to determine the best direction in which to move.



3- A ridge: Is an area of the search space that is higher than surrounding areas, but that cannot be traversed by a single move in any direction.



Heuristic Search
2) Best-First Search

- Best-First search is a way of combining the advantages of both depth-first search and breadth-first search into a single method.

- The actual operation of this algorithm is very simple. It proceeds in steps, expanding one node at each step, until it generates a node that corresponds to a goal state.

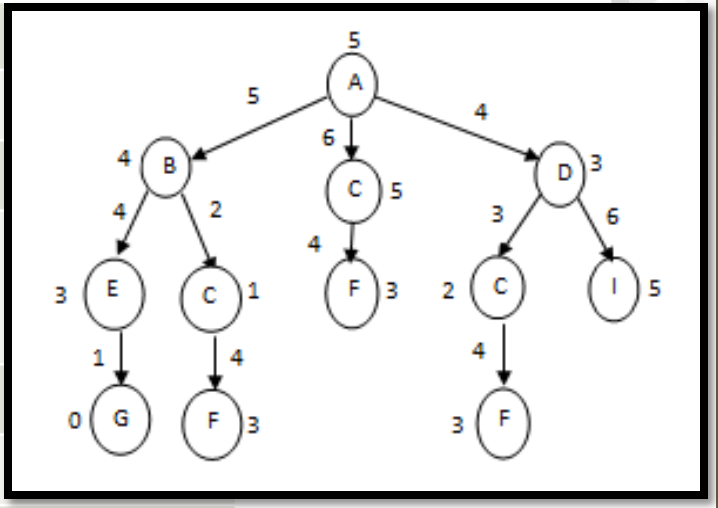
- At each step, it picks the most promising of the nodes that have so far been generated but not expanded. It generates the successors of the chosen nodes, applies the heuristic function to them, and adds them to the list of open nodes, after checking to see if any of them have been generated before. By doing this check, we can guarantee that each node only appears once in the graph, although many nodes may point to it as a successor. Then the next step begins.

In Best-First search, the search space is evaluated according to a heuristic function. Nodes yet to be evaluated are kept on an OPEN list and those that have already been evaluated are stored on a CLOSED list. The OPEN list is represented as a priority queue, such that unvisited nodes can be queued in order of their evaluation function. The evaluation function f(n) is made from only the heuristic function (h(n)) as:

$$f(n) = h(n) .$$

For Example: Search for **G** from **A** with **minimum heuristic**

[open]		[closed]
[A5]		[]
[B4,C5,D3]		[A5]
[D3,B4,C5]	"sorted nods"	[A5]
[C2,I5,B4,~~C5~~]	"delete duplicate node which has a high heuristic"	[D3,A5]
[C2,B4,I5]	"sorted nods"	[D3,A5]
[F3,B4,I5]		[C2, D3,A5]
[B4,I5]		[F3, C2, D3,A5]
[E3,C1,I5]		[B4, F3, C2, D3,A5]
[C1,E3,I5]	"sorted nods"	[B4, F3, ~~C2~~, D3,A5]
[C1,E3,I5]		[B4, F3, C1, D3,A5]
[~~F3~~,E3,I5]		[B4, F3, C1, D3,A5]
[E3,I5]		[B4, F3, C1, D3,A5]
[G0,I5]		[E3,B4, F3, C1, D3,A5]
[I5]		[G0,E3,B4, F3, C1, D3,A5]



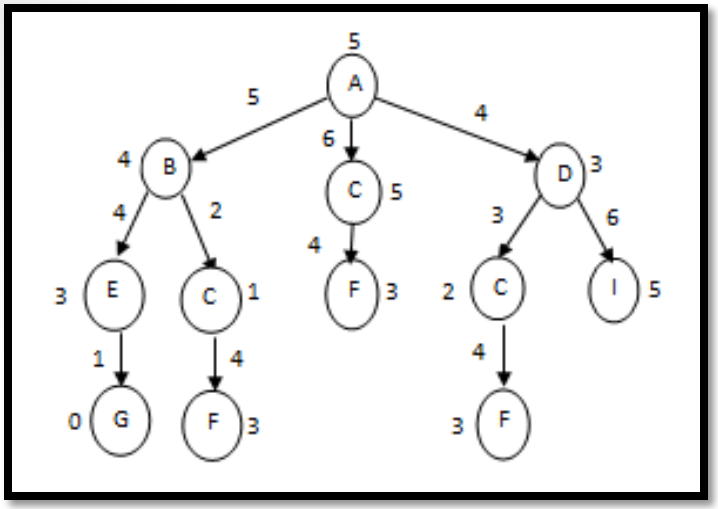
ملاحظة 1 : اذا كانت node في قائمة open افضل من node في قائمة closed تحذف node في قائمة closed و نأخذ نسخة من node في قائمة open و نضعها مكان node المحذوفة في قائمة closed

ملاحظة 2 : اذا كانت node في قائمة closed افضل من node في قائمة open عندها تبقى node في قائمة closed و تحذف node من قائمة open

F(n)= A5→D3→C1→F3→B4 →E3 →G0 = 19

Final Path: A0→D4→C2→F4→B5 →E4 →G1 = 20

Best-First-Search Algorithm



{
open:=[start];
closed:=[];
While (open ≠[]) do
{
Remove the leftmost from open, call it X;
If (X= goal) then
Return the path from start to X
Else
{
Generate children of X;
For each child of X do
Do case
The child is not already on open or closed;
{
assign a heuristic value to the child state ;
Add the child state to open;
}

The child is already on open:
If the child was reached along a shorter path than the state currently on open then
give the state on open this shorter path value.

The child is already on closed:
If the child was reached along a shorter path than the state currently on open then
{
Give the state on closed this shorter path value
Move this state from closed to open
}
}
Put x on closed;
Re-order state on open according to heuristic (best value first)
}
Return (failure);
}

Heuristic Search
3) A - Search Algorithm

A algorithm is simply define as a best-first search plus specific function. This specific function represent the actual distance **(levels)** between the initial state and the current state and is denoted by **g(n)**. A notice will be mentioned here that the same steps that are used in the best-first search are used in an **A** algorithm but in addition to the **g(n)** as follow:

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

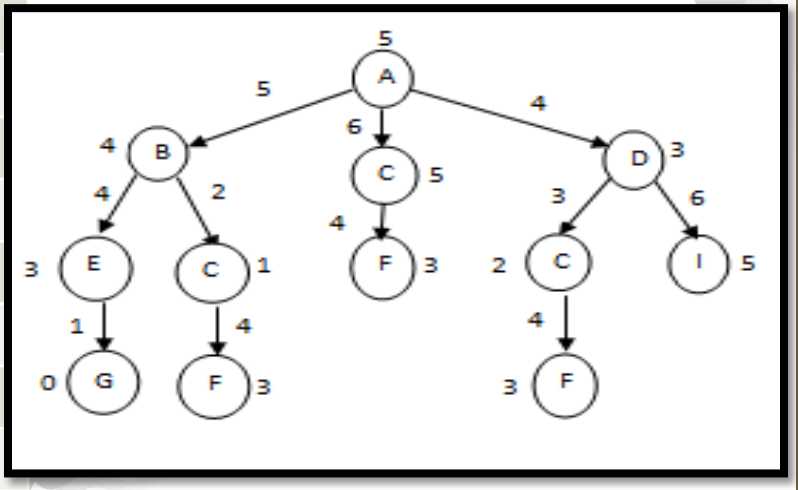
Where:

h(n):- is a **heuristic estimate** of the distance from **state (n)** to the **goal state** .

g(n):- is the **actual length** of path from **state (n)** to the **start state**.

For Example: Search for **G** from **A** with **minimum heuristic**

[open]		[closed]
[A5]		[]
[B4,C5,D3]		[A5]
[B5,C6,D4]	"plus level g(n)"	[A5]
[D4,B5,C6]	"sorted nodes"	[A5]
[C2,I5,B5,C6]		[D4,A5]
[C4,I7,B5,C6]	"plus level g(n)"	[D4,A5]
[C4,B5,I7]	"sorted nodes"	[D4,A5]
[F3,B5,I7]		[C4,D4,A5]
[F6,B5,I7]	"plus level g(n)"	[C4,D4,A5]
[B5,F6,I7]	"sorted nodes"	[C4,D4,A5]
[E3,C1,F6,I7]		[B5,C4,D4,A5]
[E5,C3,F6,I7]	"plus level g(n)"	[B5,C4,D4,A5]
[C3,E5,F6,I7]	"sorted nodes"	[B5,C4,D4,A5]
[C3,E5,F6,I7]		[B5,C3,D4,A5]
[F3,E5,F6,I7]		[B5,C3,D4,A5]
[F6,E5,F6,I7]	"plus level g(n) then delete duplicate node"	[B5,C3,D4,A5]
[E5,F6,I7]	"sorted nodes"	[B5,C3,D4,A5]
[G0,F6,I7]		[E5,B5,C3,D4,A5]
[G3,F6,I7]	"plus level g(n)"	[E5,B5,C3,D4,A5]
[F6,I7]		[G3, E5,B5,C3,D4,A5]



ملاحظة: اذا كانت node في قائمة open افضل من node في قائمة closed تحذف node في قائمة closed و نأخذ نسخة من node في قائمة open و نضعها مكان node المحذوفة في قائمة closed

F(n)= A5→D4→C3→B5 →E5 →G3 = **25**

Final Path: A0→D4→C2→B5 →E4 →G1 = **16**

Heuristic Search
4) A-Star Search Algorithm

- **A*** algorithm is simply define as a best-first search plus specific function. This specific function represents the actual distance **(levels)** between the current state and the goal state and is denoted by **h(n)**. It evaluates nodes by combining **g(n)**, the cost to reach the node, and **h(n)**, the cost to get from the node to the goal as follow:

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

Where

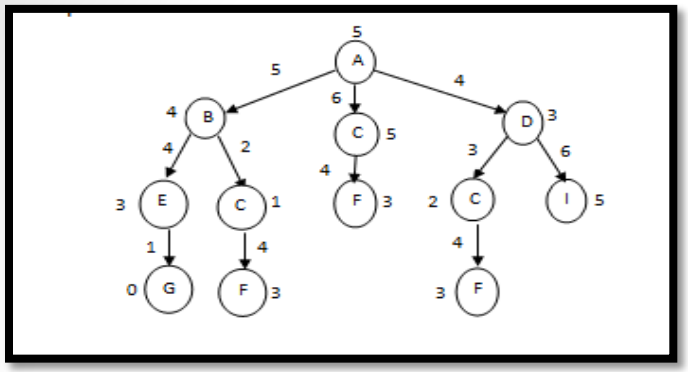
g(n): is the **cost** of the path from the **start node** to node **n**.

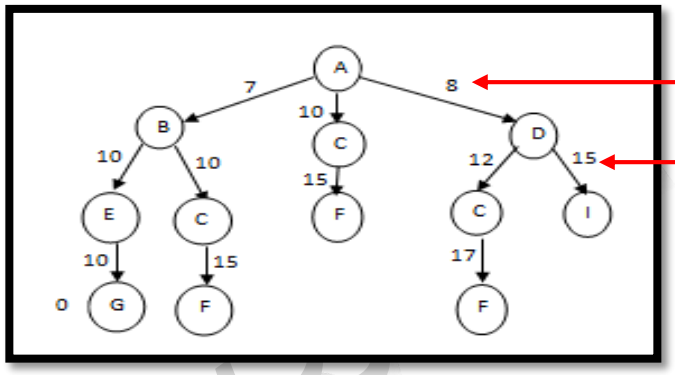
h(n): is a **heuristic estimated cost** from node **n** to the **goal node**.

f(n): is a **total estimated cost** of the cheapest path through node **n**.

- Thus, if we are trying to find the cheapest solution, a reasonable thing to try first is the node with the lowest value of **g(n) + h(n)**. It turns out that this strategy is more than just reasonable: provided that the heuristic function **h(n)** satisfies certain conditions, A* algorithm is both **complete** and **optimal**.

For Example: Search for **G** from **A** with **minimum heuristic**





[open]	[closed]
[A]	[]
[B7,C10,D8]	[A]
[B7,D8,C10] "sorted nodes"	[A]
[E10,C10,D8,C10]	[B7,A]
[D8,E10,C10] "sorted nodes"	[B7,A]
[C12,I15,E10,C10]	[D8,B7,A]
[E10,C10,I15] "sorted nodes"	[D8,B7,A]
[G10,C10,I15]	[E10,D8,B7,A]
[C10,I15]	[G10,E10,D8,B7,A]

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

- $g(n)$: الكلفة الفعليه ما بين start state و node الحالية
- $h(n)$: عدد levels مابين node الحالية و goal state

$f(D) = 4 + 4 = 8$

$f(I) = (6+4) + 5 = 15$

- $g(n)$: الكلفة الفعليه ما بين start state(A) و node(I)
- $h(n)$: عدد levels مابين node(I) و goal state (G)

F(n)= A→B7→D8→E10 →G10 = **35**

Final Path: A0→B5→D4→E4 →G1 = **14**

A* Algorithm Properties

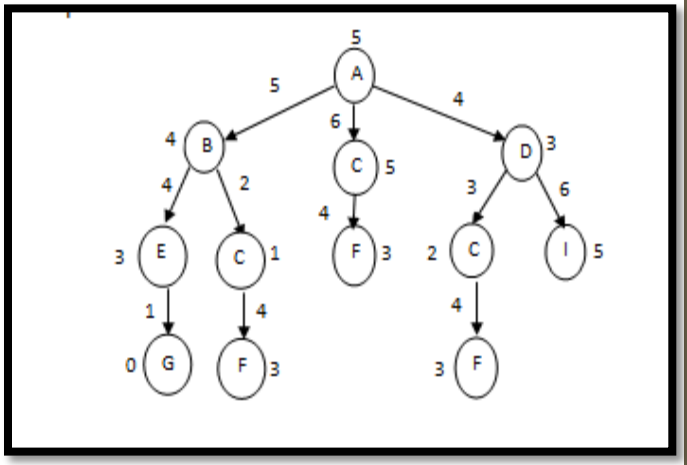
1) Admissibility

Admissibility means that **h(n)** is less than or equal to the cost of the minimal path from **node n** to the **goal**, in other words:

h(n) ≤ Distance to goal

state	h	distance from (n) to goal	
A	5	(5+4+1)	10
B	4	(4+1)	5
C ‘A’	5	(6+5+4+1)	16
D	3	(4+5+4+1)	14
E	3	(1)	1
C ‘B’	1	(2+4+1)	7
F ‘A’	3	(4+6+5+4+1)	20
C ‘D’	2	(3+4+5+4+1)	17
I	5	(6+4+5+4+1)	20
F ‘B’	3	(4+2+4+1)	11
F ‘D’	3	(4+3+4+5+4+1)	21
G	0	0	

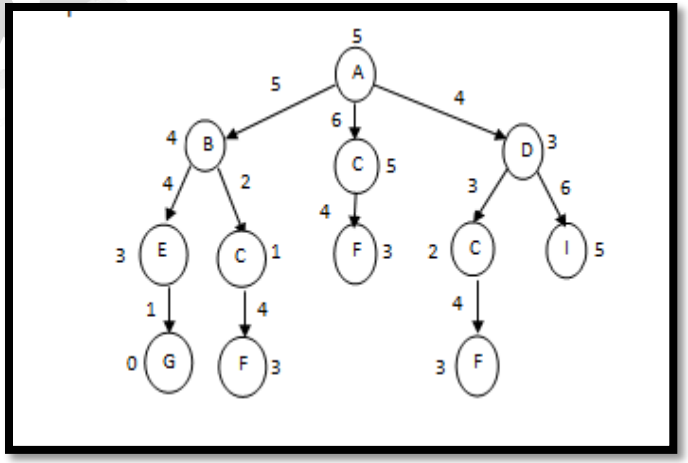
Not Admissible



2) Consistency (Monotonicity)

Consistency means that the difference between the heuristic of a state (node) and the heuristic of its descendent is less than or equal the cost between them, and the heuristic of the goal equal zero, in other words:

1. **$h(n_i)-h(n_j) \leq cost(n_i,n_j)$.**
2. **h(goal) = 0.**



Example:

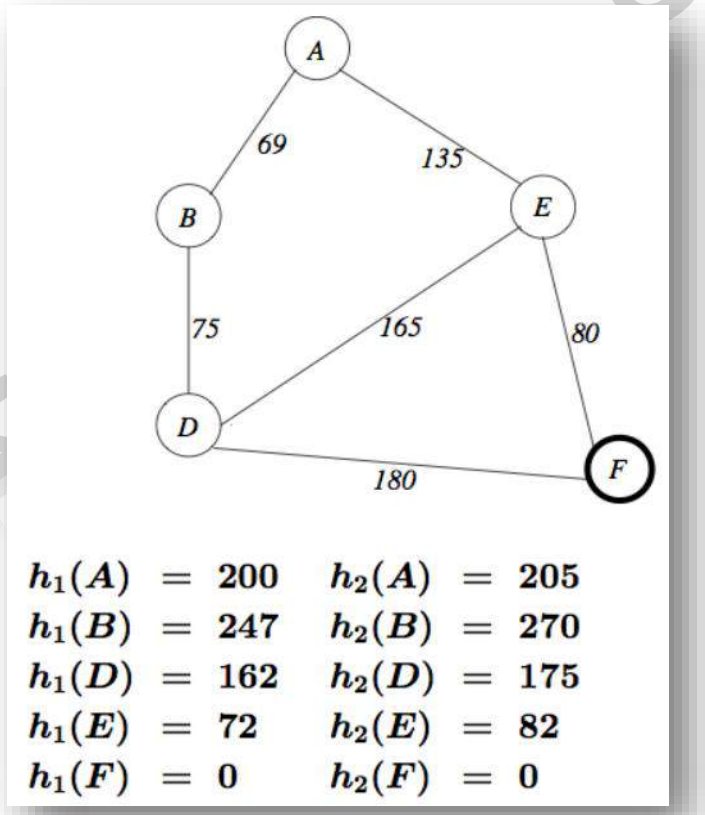
h(A)- h(B) ≤ cost (A,B). // 5-4 ≤ 5 yes

h(A)- h(C) ≤ cost (A,C). // 5-5 ≤ 6 yes

h(A)- h(D) ≤ cost (A,D). // 5-3 ≤ 4 yes

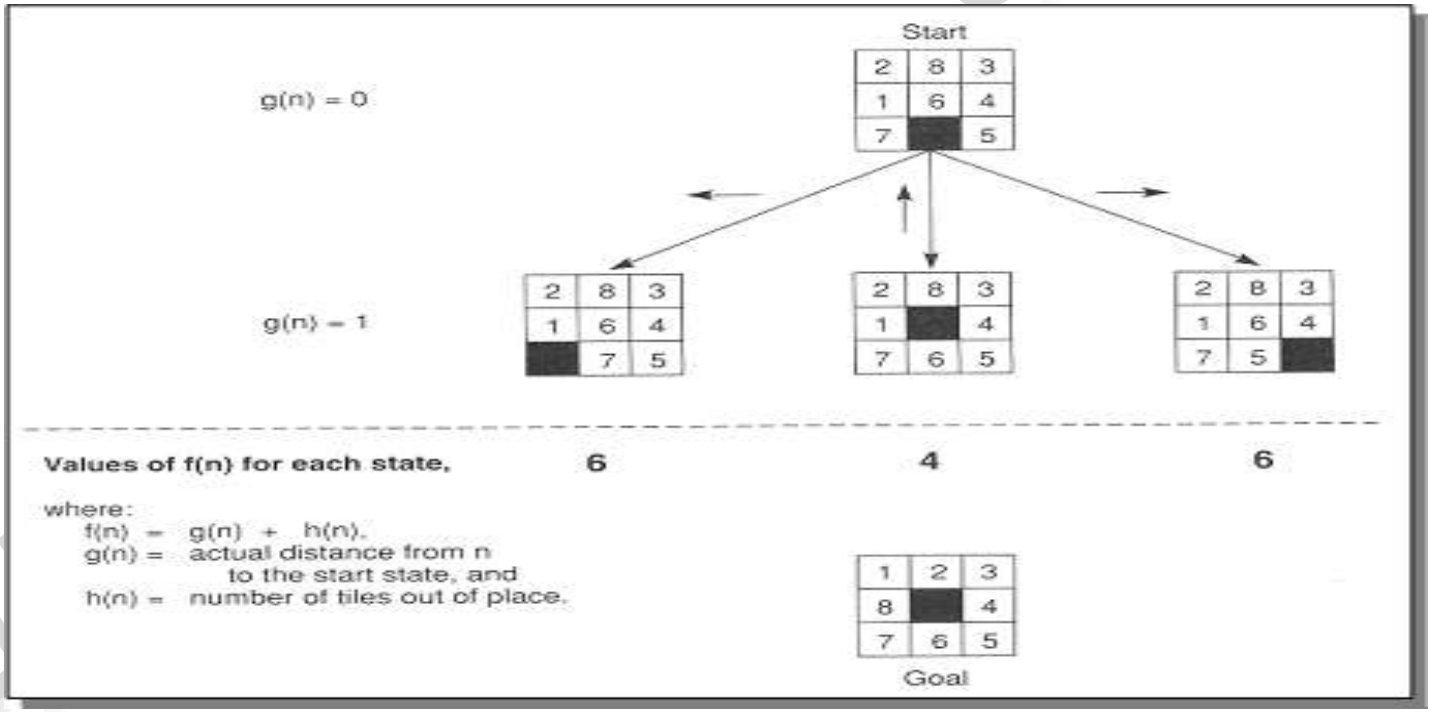
3) Informedness

For two A* heuristics **h1** and **h2** , if **h1(n) ≤ h2(n)**, for all **states (nodes) n** in the search space, heuristic **h2** is said to be **more informed** than heuristic **h1**.



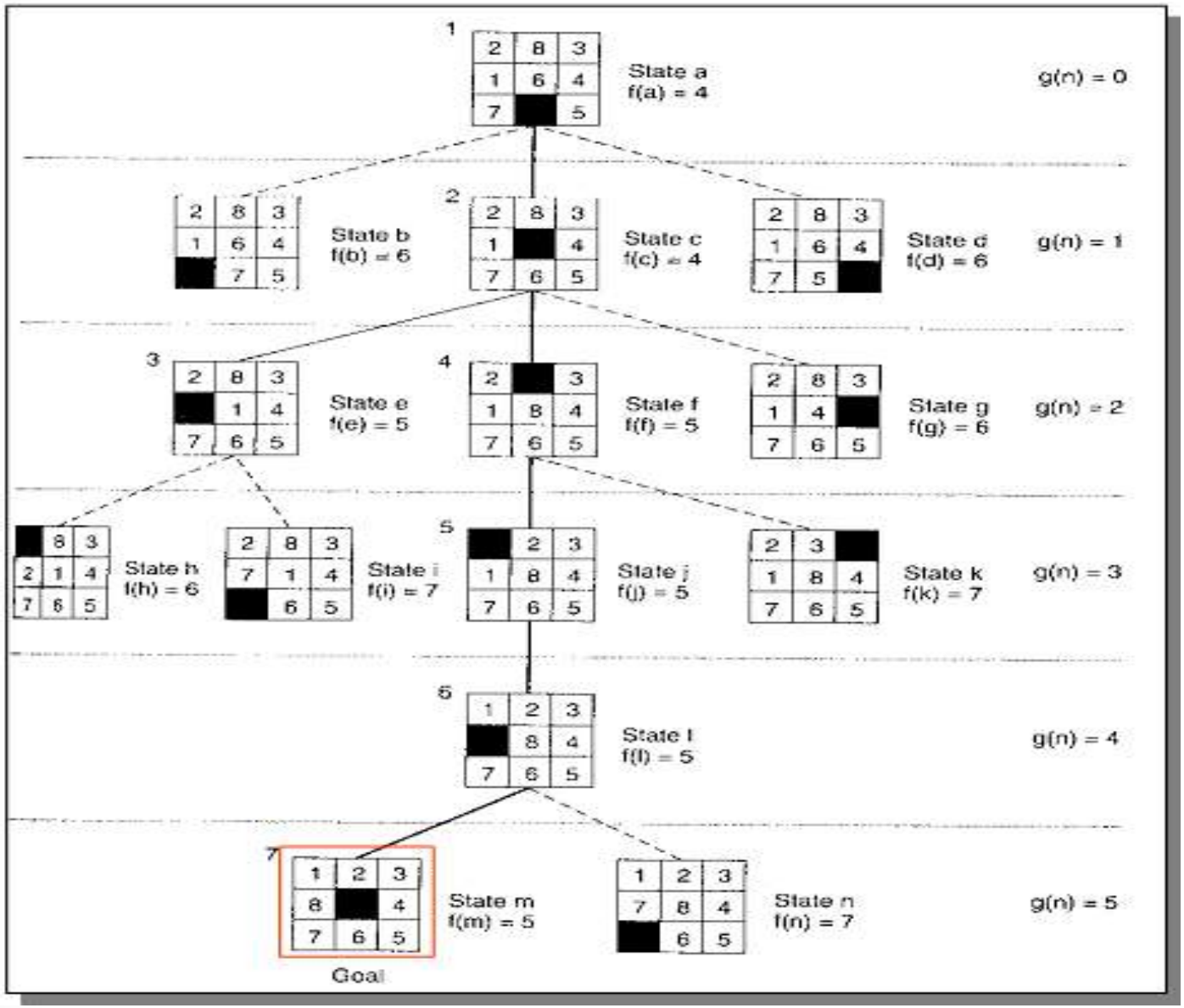
Complex Search Space and Problem Solving Approach

1) 8-puzzle Problem



To summarize:

1. Operations on states generate children of the state currently under examination.
2. Each new state is checked to see whether it has occurred before (is on either open or closed), thereby preventing loops.
3. Each state n is given an I value equal to the sum of its depth in the search space g(n) and a heuristic estimate of its distance to a goal h(n). The h value guides search toward heuristically promising states while the **g** value prevents search from persisting indefinitely on a fruitless path.
4. States on open are sorted by their f values. By keeping all states on open until they are examined or a goal is found, the algorithm recovers from dead ends.
5. As an implementation point, the algorithm's efficiency can be improved through maintenance of the open and closed lists, perhaps as heaps or leftist trees.


1
State a
f(a) = 4
g(n) = 0
State b
f(b) = 6
2
State c
f(c) = 4
State d
f(d) = 6
g(n) = 1
3
State e
f(e) = 5
4
State f
f(f) = 5
State g
f(g) = 6
g(n) = 2
State h
f(h) = 6
State i
f(i) = 7
5
State j
f(j) = 5
State k
f(k) = 7
g(n) = 3
5
State l
f(l) = 5
g(n) = 4
7
State m
f(m) = 5
State n
f(n) = 7
g(n) = 5
Goal

After implementation of **A** algorithm, the Open and Closed is shown as follows:

open	closed
[a4]	[]
[c4,b6,d6]	[a4]
[e5,f5,b6,d6,g6]	[c4, a4]
[f5,b6,d6,g6,h6,i7]	[e5, c4, a4]
[j5,b6,d6,g6,h6,i7,k7]	[f5 e5, c4, a4]
[l5, b6,d6,g6,h6,i7,k7]	[j5, f5 e5, c4, a4]
[m5, b6,d6,g6,h6,i7,k7,n7]	[l5, j5, f5 e5, c4, a4]
	[m5,l5, j5, f5 e5, c4, a4]

Success, m=goal!!

Example: Consider 8-puzzle problem with ***start state*** is shown as follows:

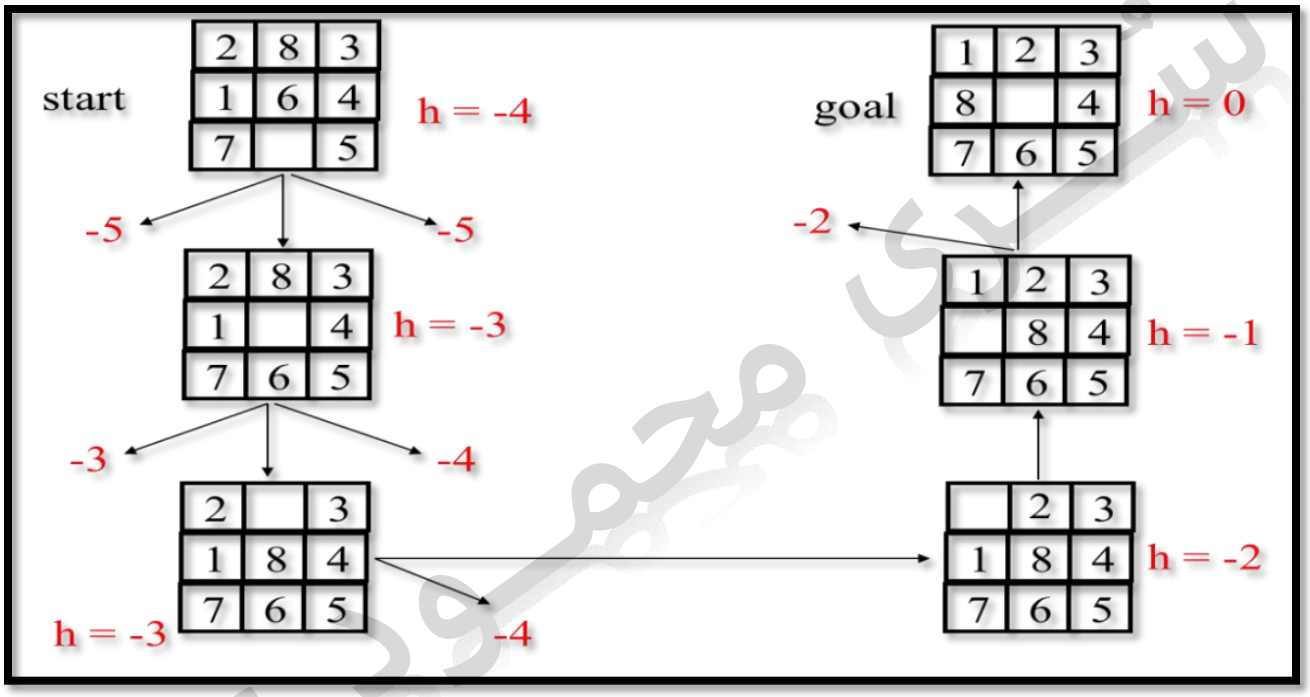
2	8	3
1	6	4
7		5

And the **goal state** is:

1	2	3
8		4
7	6	5

Assume the heuristic is calculated as following:
h (n) = -(number of tiles out of place)
Draw the path to get the goal using Hill Climbing search algorithm?

- **Answer**


start
2 8 3
1 6 4
7 5
h = -4
-5
-5
2 8 3
1 4
7 6 5
h = -3
-3
-4
2 3
1 8 4
7 6 5
h = -3
-4
goal
1 2 3
8 4
7 6 5
h = 0
-2
1 2 3
8 4
7 6 5
h = -1
2 3
1 8 4
7 6 5
h = -2

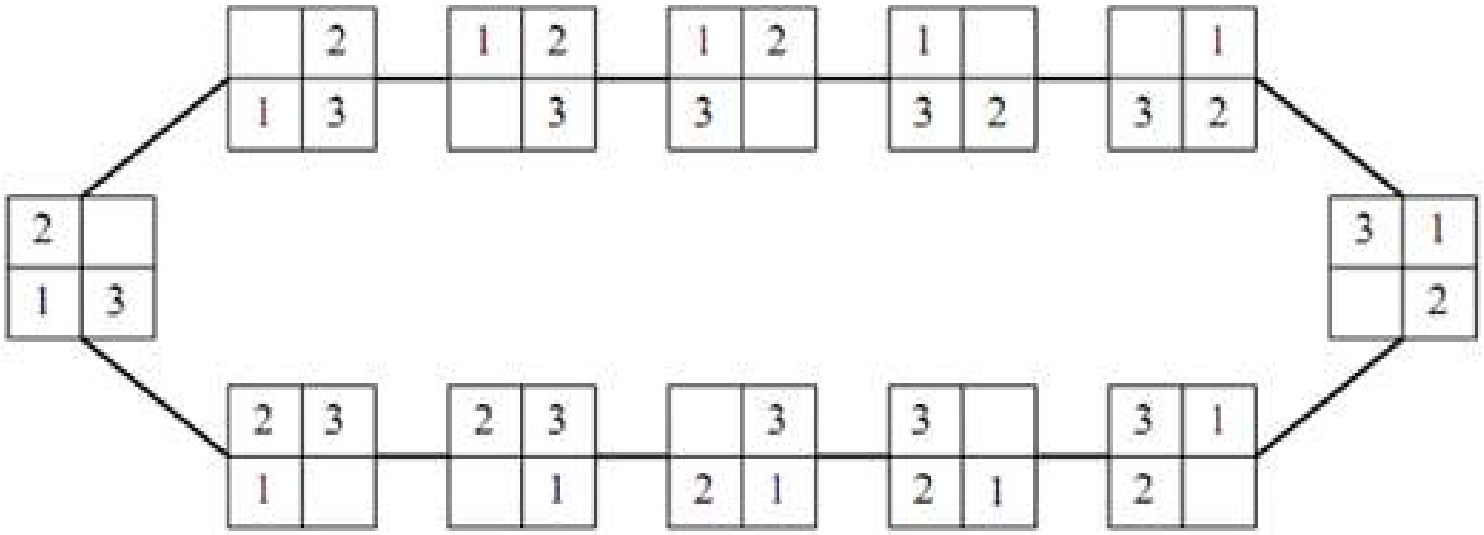
Example: Consider the 3-puzzle problem, which is a simpler version of the 8-puzzle where the board is 2 × 2 and there are three tiles, numbered 1, 2, and 3. there are four moves: move the blank up, right, down, and left. The cost of each move is 1. Consider this start state:

Start

2	
1	3

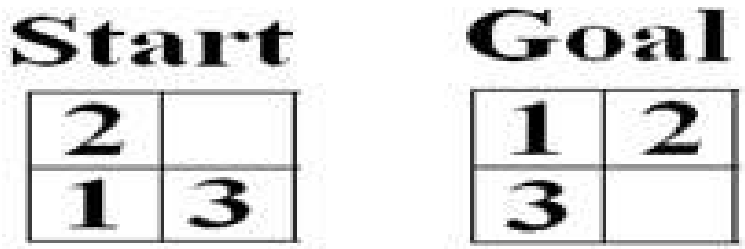
Draw the entire non-repeating state space for this problem, labeling nodes and arcs clearly?

Answer


2 1 3
2 3 1 3
1 2 3 1 3
1 2 3 1 3 2
1 2 3 1 3 2
3 1 2
2 3 1 3 2 1
2 3 1 2 3 1
3 2 1 3 2 1
3 1 2 3 1 2

Example

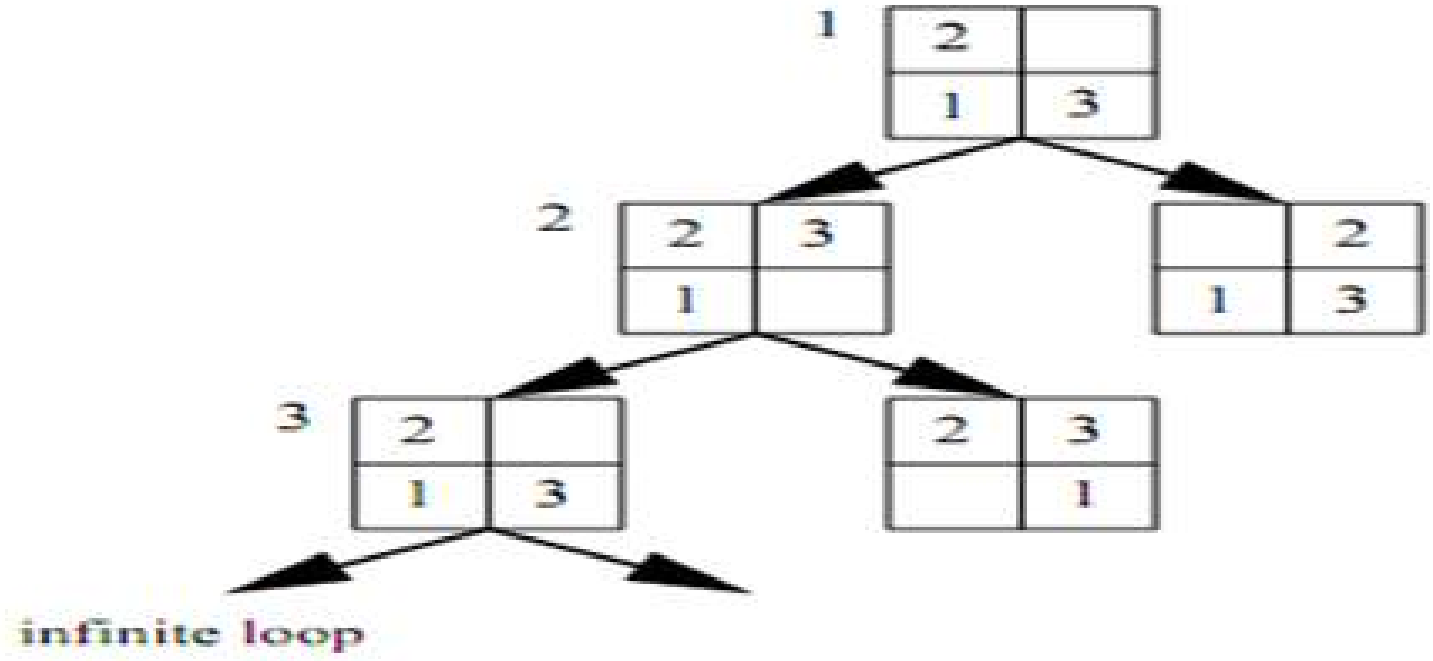
Assume the start and goal states are:



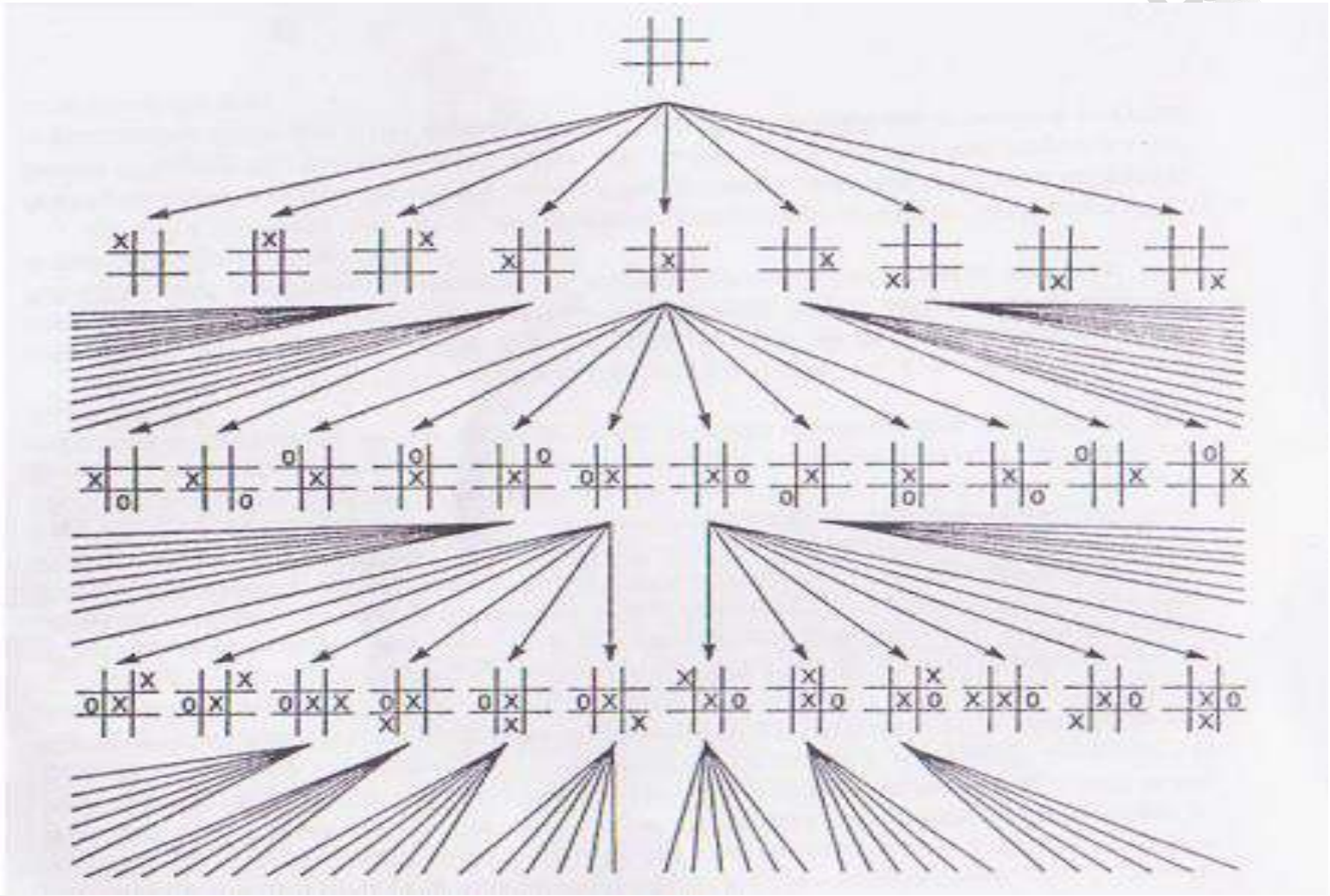
Are the goal is found using Depth First Search algorithm? If not explain why?

Answer

The goal cannot found using Depth First Search algorithm because there is an infinite loop as shown below:



2) Tic – Tac – Toe Game

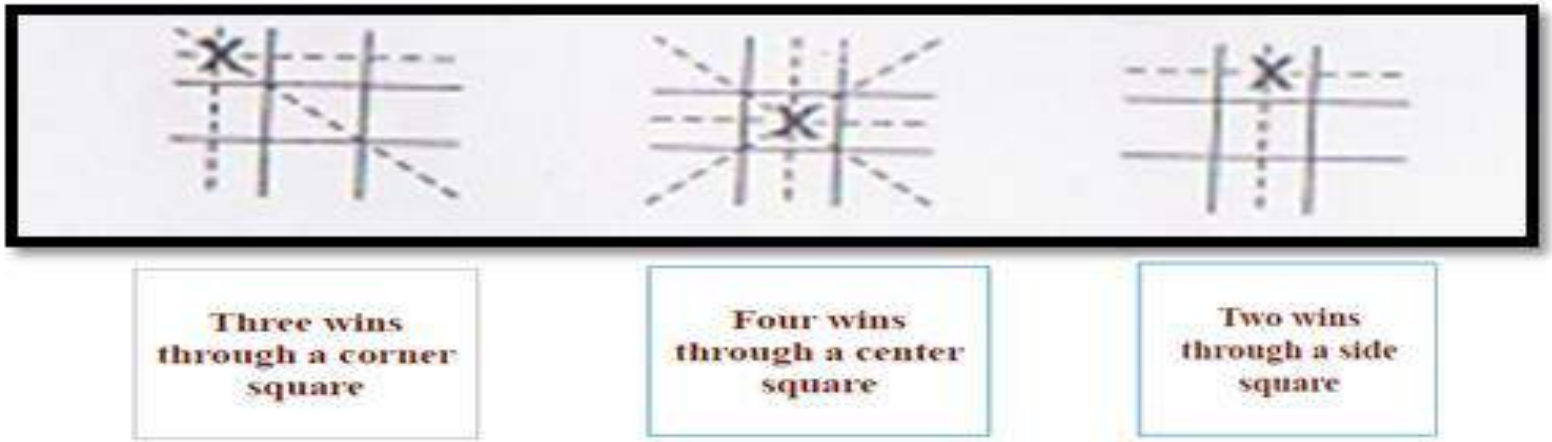


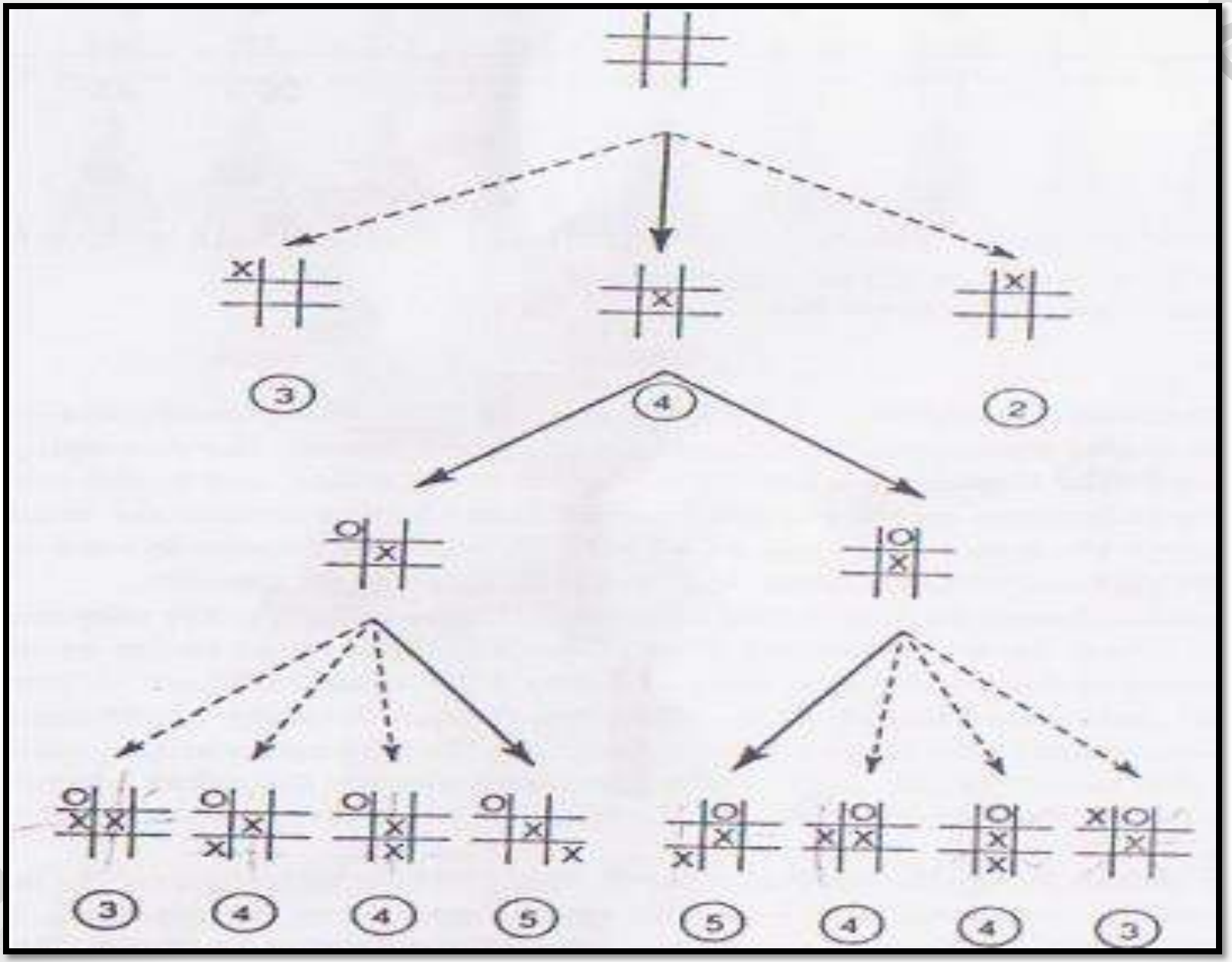
The complexity of the search space is 9!

9! =9*8*7*……..*1

Therefore it is necessary to implement two conditions:

1. Problem reduction
2. Guarantee the solution




3
4
2
3
4
4
5
5
4
4
3

Using Heuristic in Games

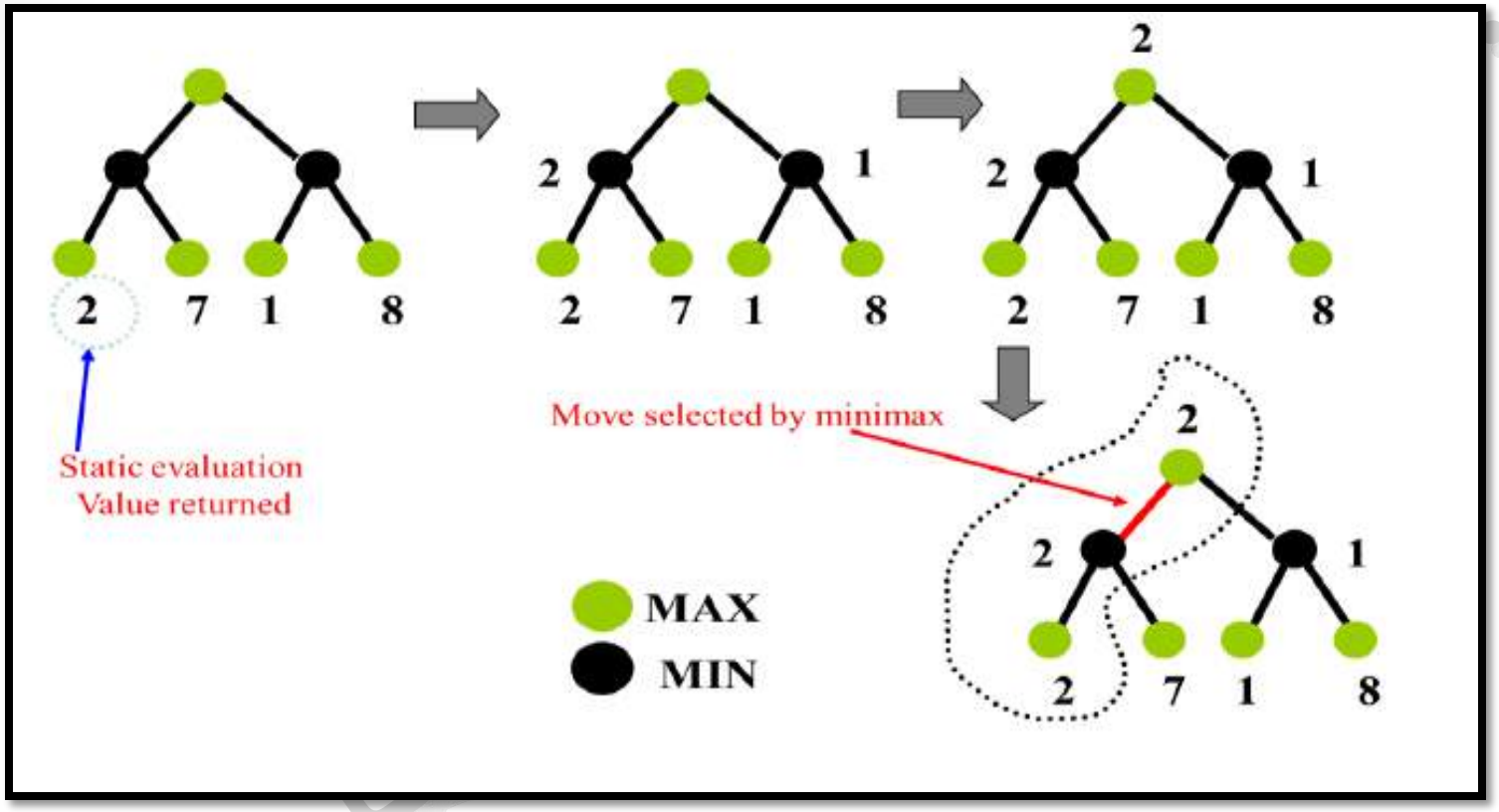
1) **Minimax Search Algorithm**

The minimax algorithm is a useful method for simple two-players games. It is a method for selecting the best move given an alternating game where each player opposes the other working toward a mutually exclusive goal. Each player knows the moves that are possible given a current game state, so for each move, all subsequent moves can be discovered.

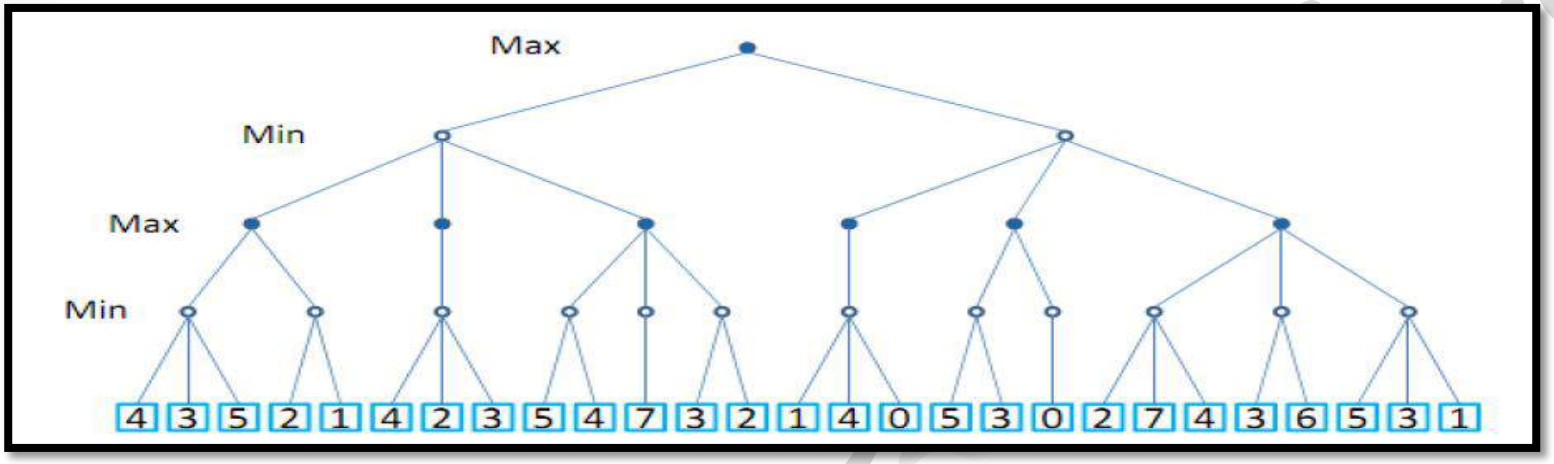
The minimax algorithm assumes two players are represented as MAX and MIN. The leaf nodes values of the tree are filled bottom-up with the evaluated values. The nodes that belong to the MAX player will select the maximun value of their children, and, the nodes that belong to the MIN player will select the minimun value of their children.

The minimax algorithm performs three tasks:

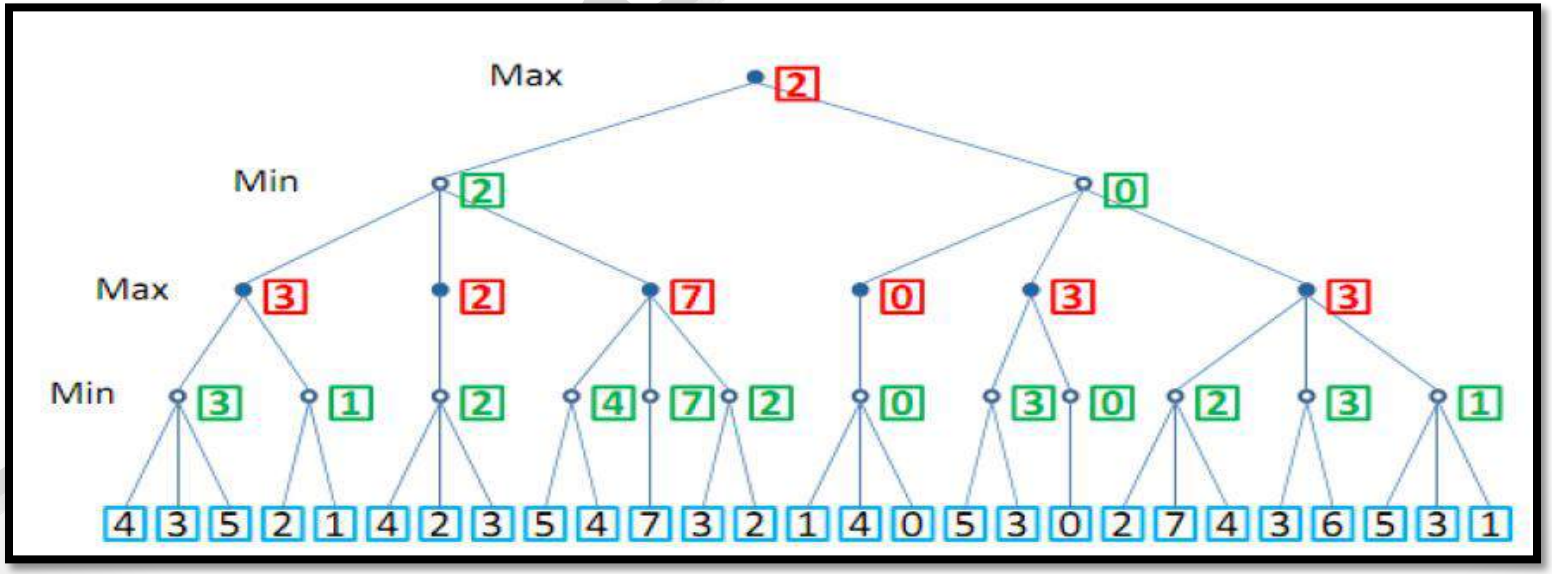
1. Construct tree (depth-bound).
2. Compute evaluation values.
3. Propagate upwards (min/max).


2 7 1 8
2
2 7 1 8
2 1
2
2 7 1 8
2 1
2
Static evaluation
Value returned
Move selected by minimax
2
2 1
2 7 1 8
MAX
MIN

Example: **Perform the MiniMax algorithm on the tree below:**



Solution:

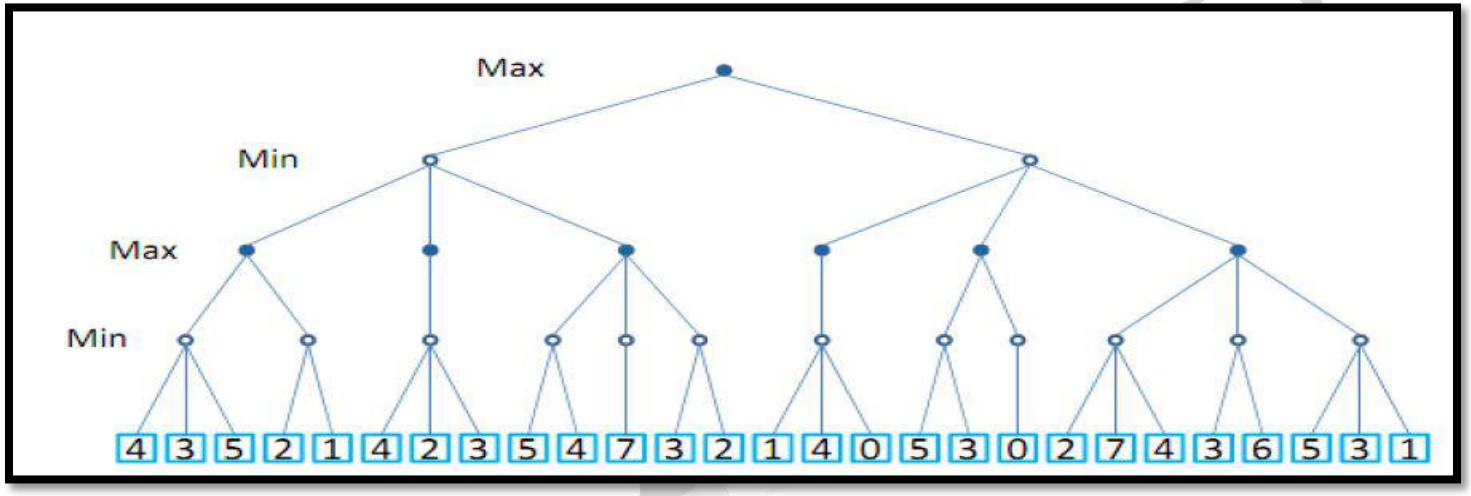


2) Alpha-Beta Algorithm

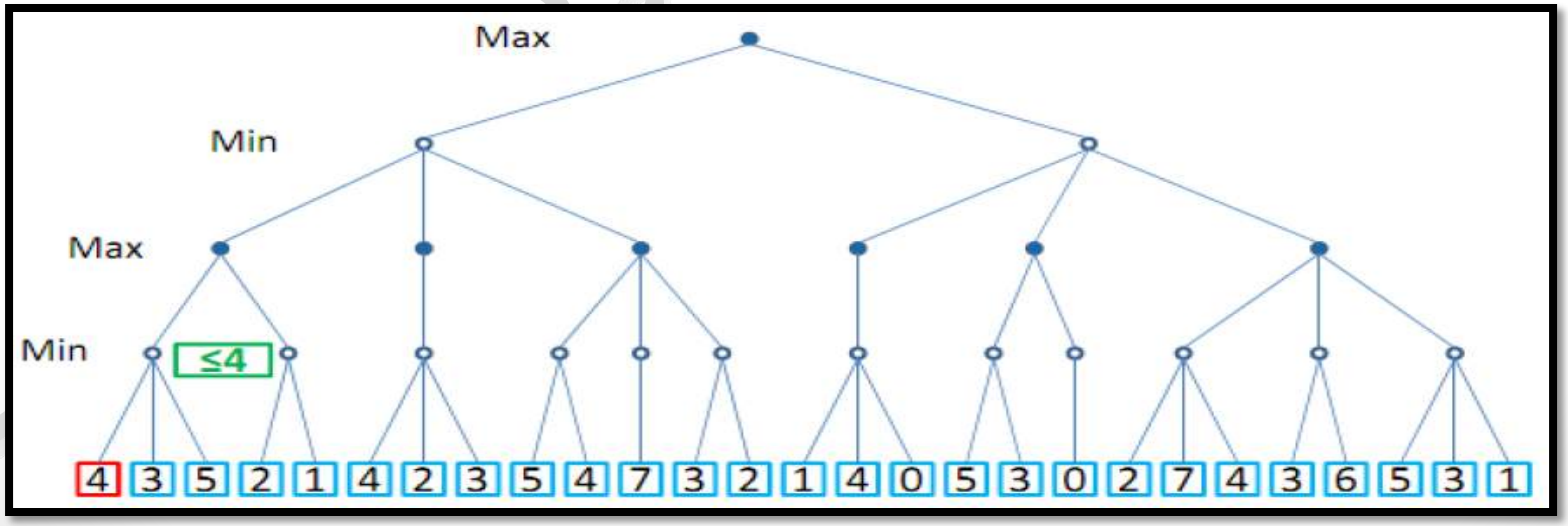
Alpha-beta pruning is a simple algorithm that minimizes the game-tree search for moves that are obviously bad. The basic idea of alpha-beta pruning is the identifying moves that are not beneficial, and remove them from the game tree. The higher in the game tree that branches are pruned the greater effect in minimizing the search space of the tree.

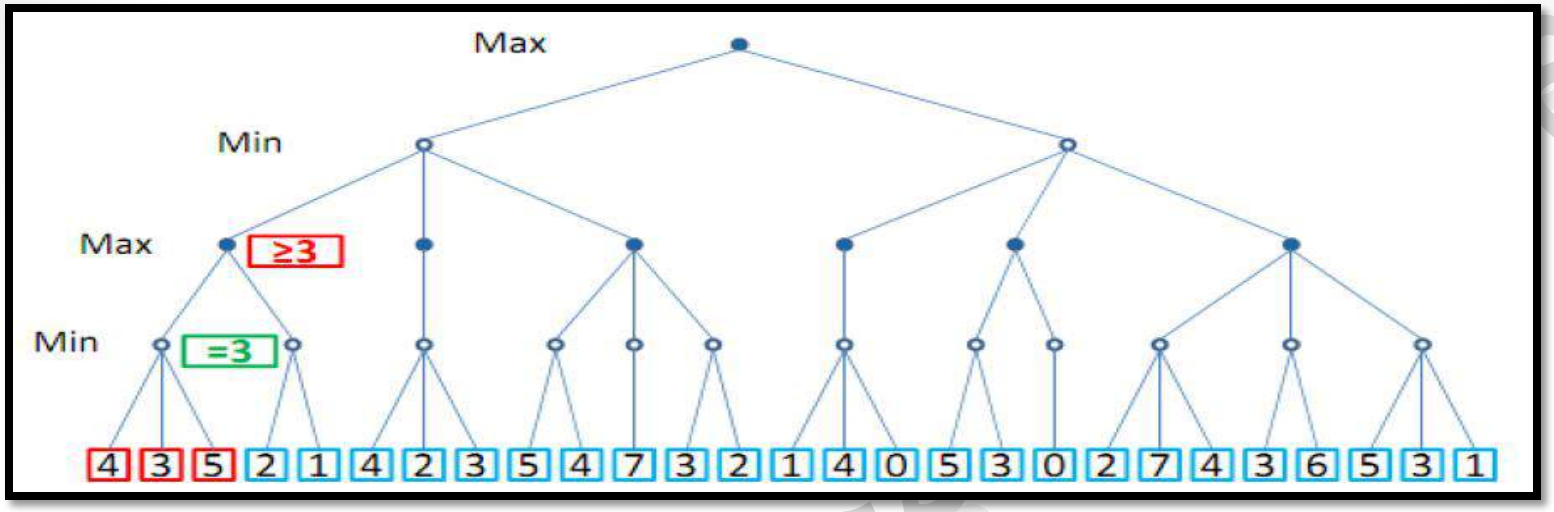
- Alpha = the value of the best choice we've found so far for MAX (highest)
- Beta = the value of the best choice we've found so far for MIN (lowest)
- When maximizing, cut off values lower than Alpha
- When minimizing, cut off values greater than Beta

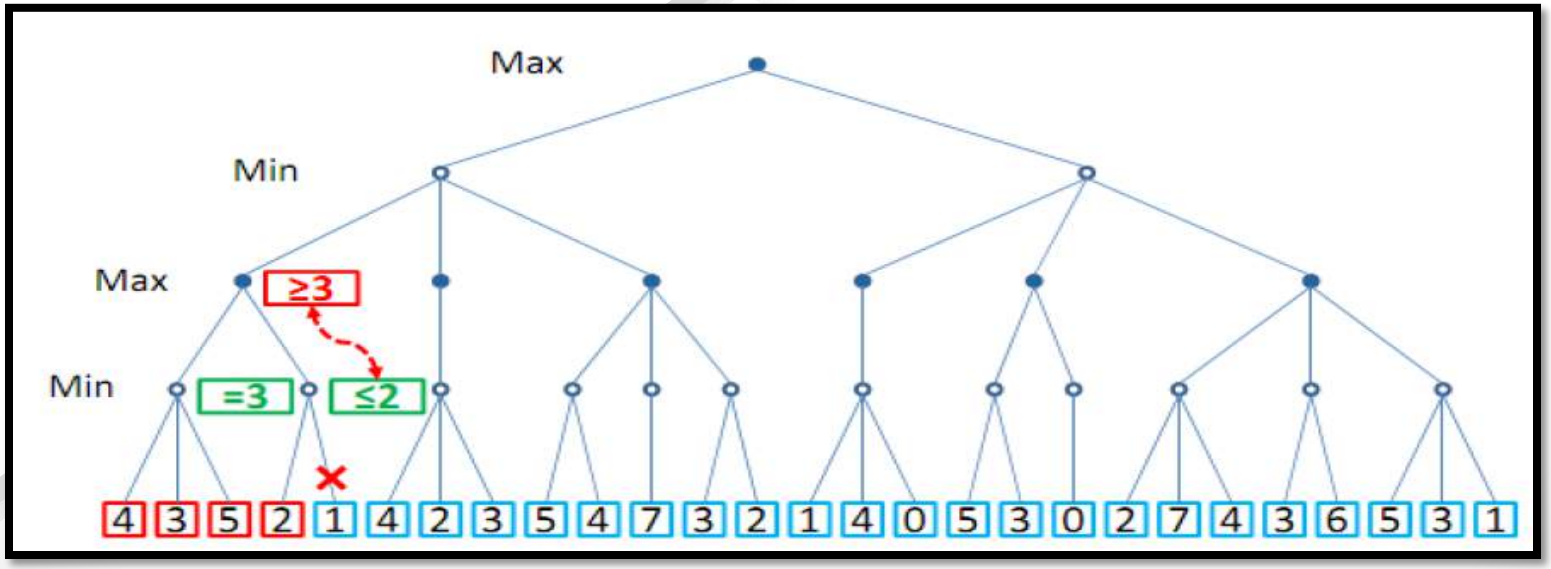
Example: Perform the MiniMax algorithm with **αβ-pruning** to the figure below and then enumerate and list (from left to right) the evaluation leaves that should be avoided.

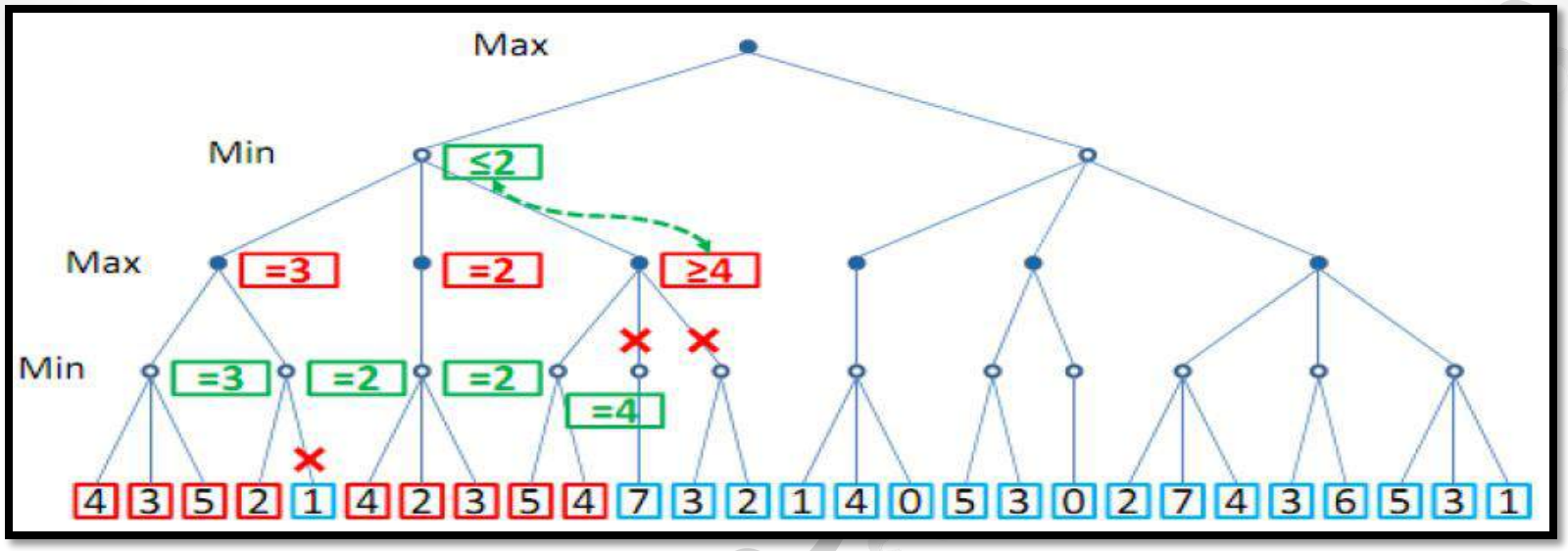


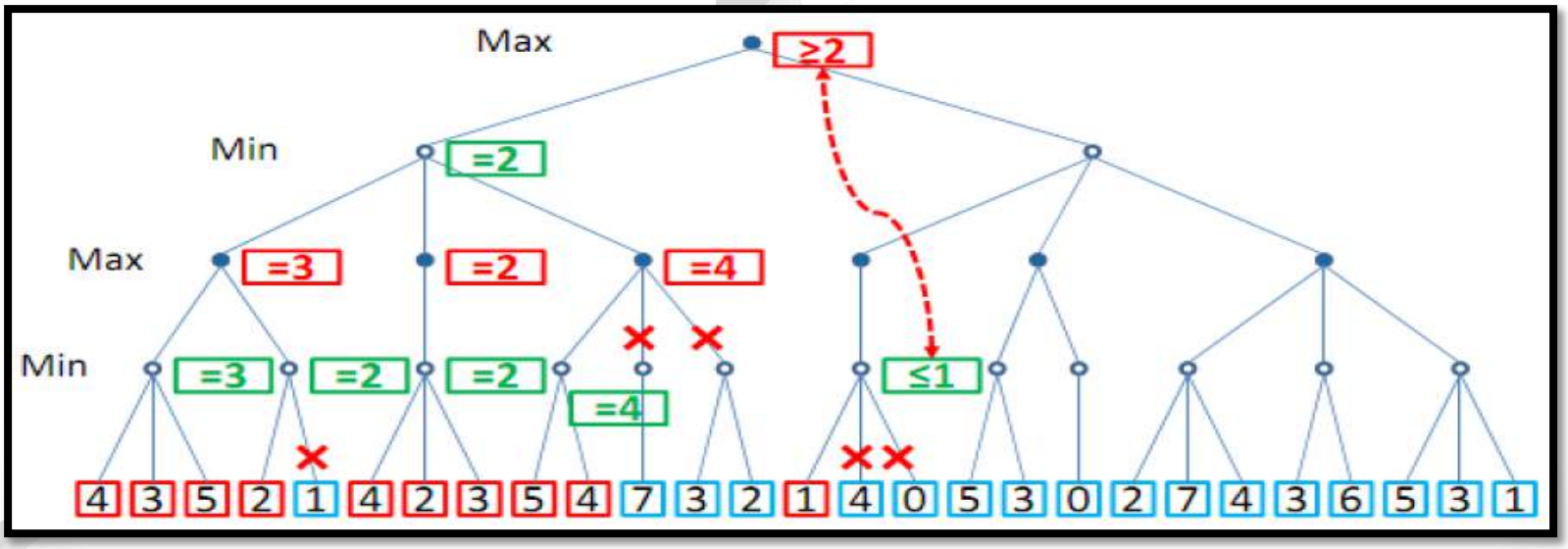
Solution:

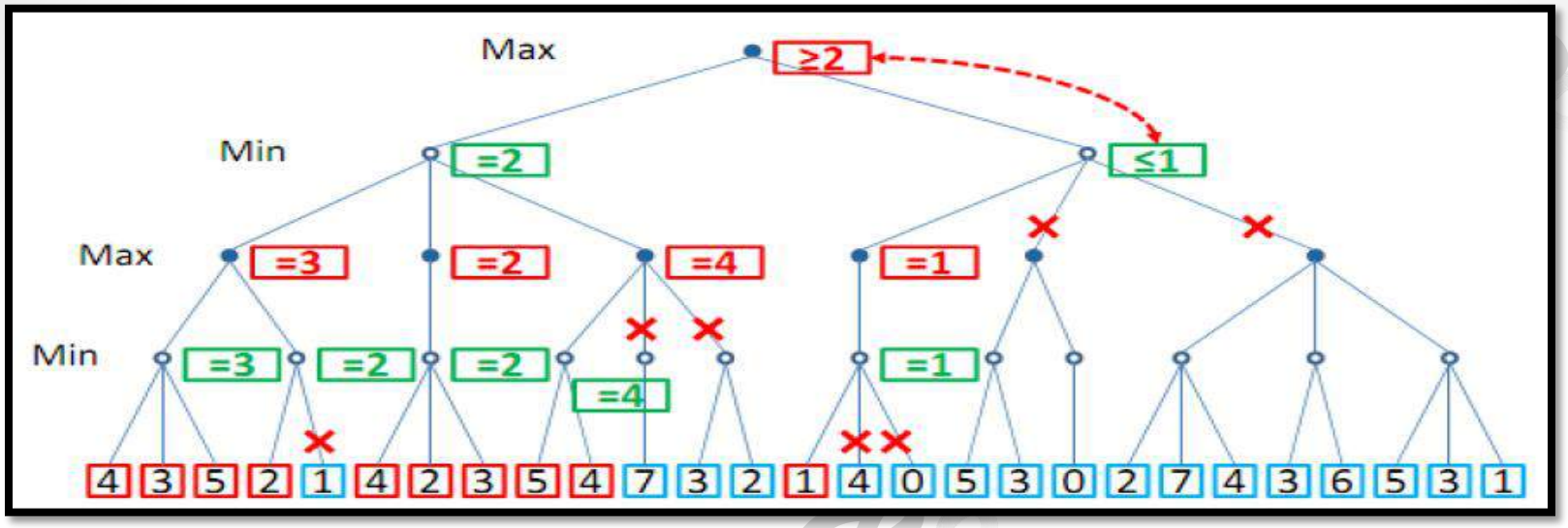


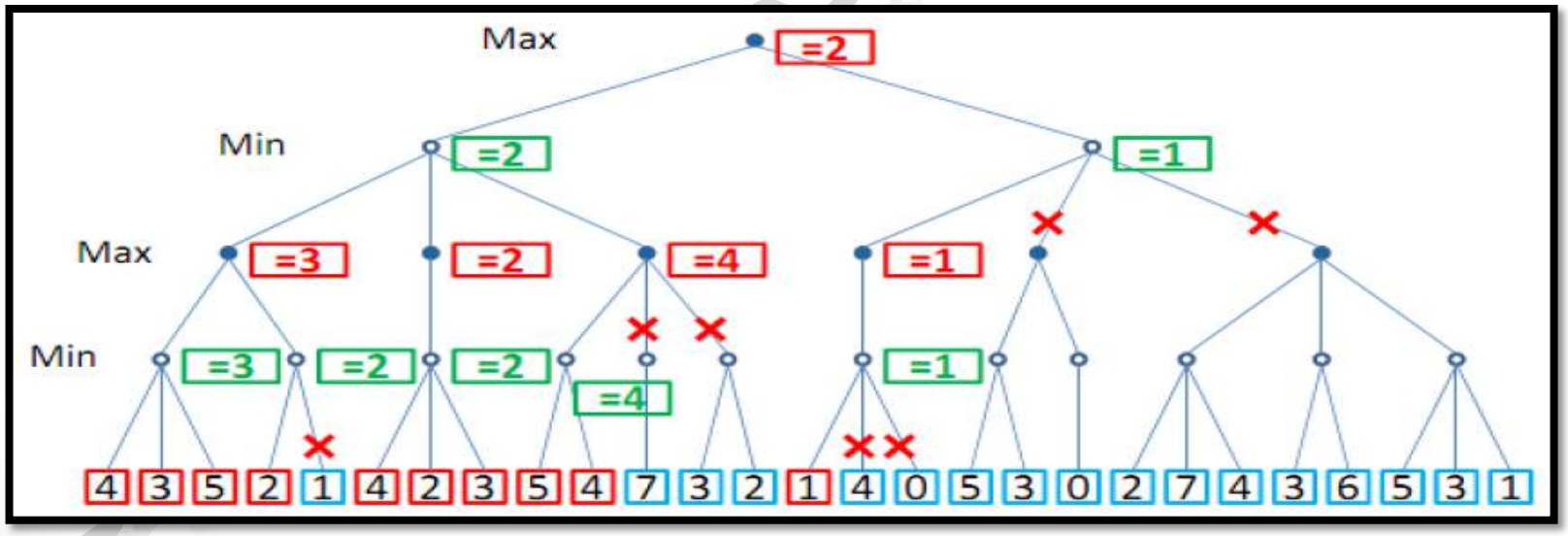

Max
Min
Max
≥3
Min
=3
4 3 5 2 1 4 2 3 5 4 7 3 2 1 4 0 5 3 0 2 7 4 3 6 5 3 1


Max
Min
Max
≥3
Min
=3
≤2
4 3 5 2 1 4 2 3 5 4 7 3 2 1 4 0 5 3 0 2 7 4 3 6 5 3 1


Max
Min
≤2
Max
=3
=2
≥4
Min
=3
=2
=2
=4
4 3 5 2 1 4 2 3 5 4 7 3 2 1 4 0 5 3 0 2 7 4 3 6 5 3 1


Max
≥2
Min
=2
Max
=3
=2
=4
Min
=3
=2
=2
=4
≤1
4 3 5 2 1 4 2 3 5 4 7 3 2 1 4 0 5 3 0 2 7 4 3 6 5 3 1





There are **17** evaluation leaves have been avoided and they listed as **[1,7,3,2,4,0,5,3,0,2,7,4,3,6,5,3,1].**

Using Heuristic in Games

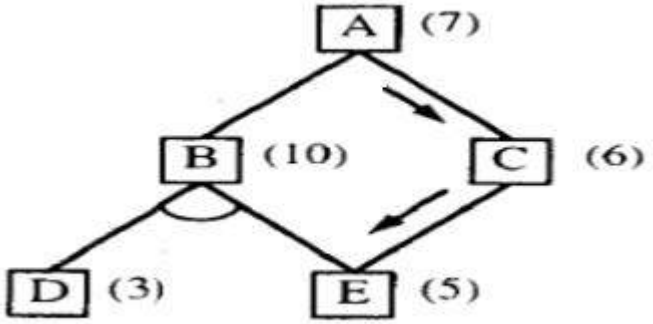
2) **AND/OR Graph Algorithm**

An and/or tree is a graphical representation of the reduction of problems (or goals) to conjunctions and disjunctions of sub-problems (or sub-goals).

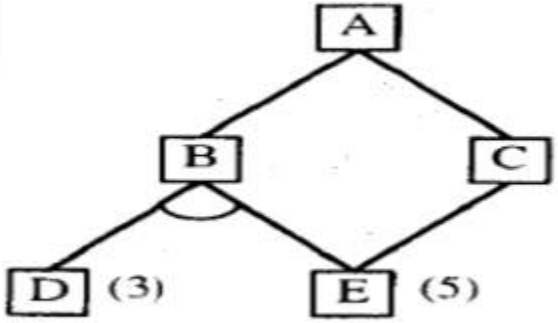
- **Nodes** represent sub-problems.
- **And links** represent sub-problems decomposition.
- **OR links** represent alternative solutions.
- The **start node** is an initial problem.
- **Terminal nodes** (nodes with no successors) are solved sub-problems.

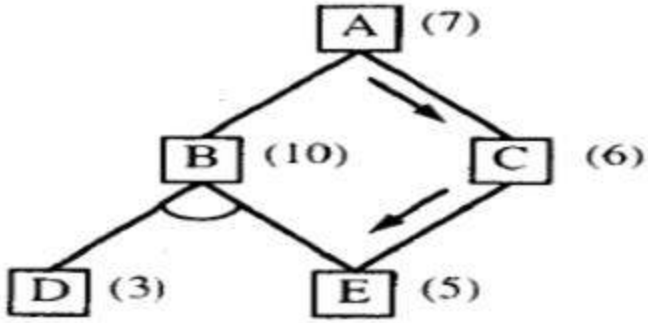
Solution graph

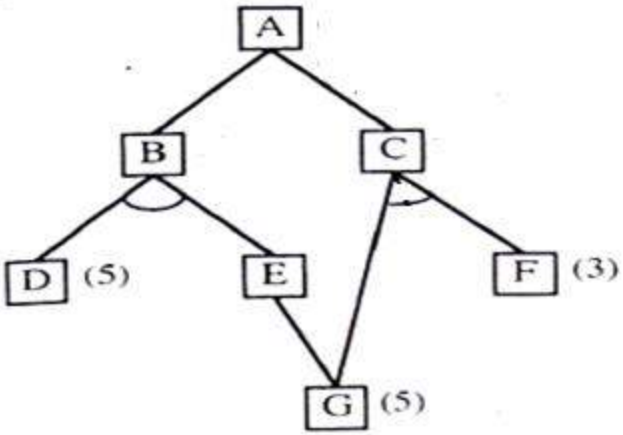
- It is an AND/OR sub graph such that:
    1. It contains the start node.
    2. All it terminal nodes are solved primitive problems.
    3. If it contains an AND node L, it must contain the entire group of AND links that leads to children of node L.
- The cost of a solution graph is the sum cost of its arcs.

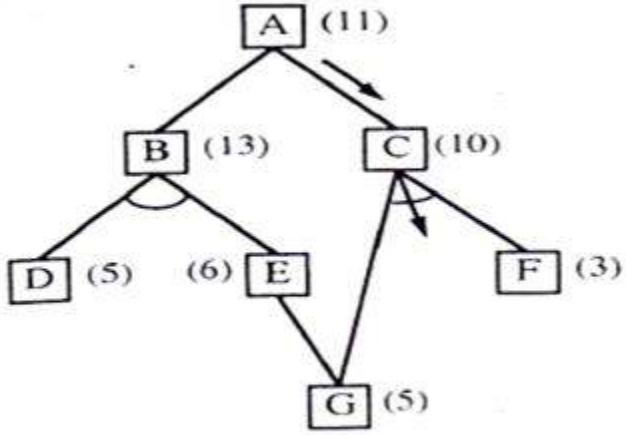


Example: find the lowest cost of the start node, for the following graphs:









Example: find the lowest cost of the start node, for the following graphs: