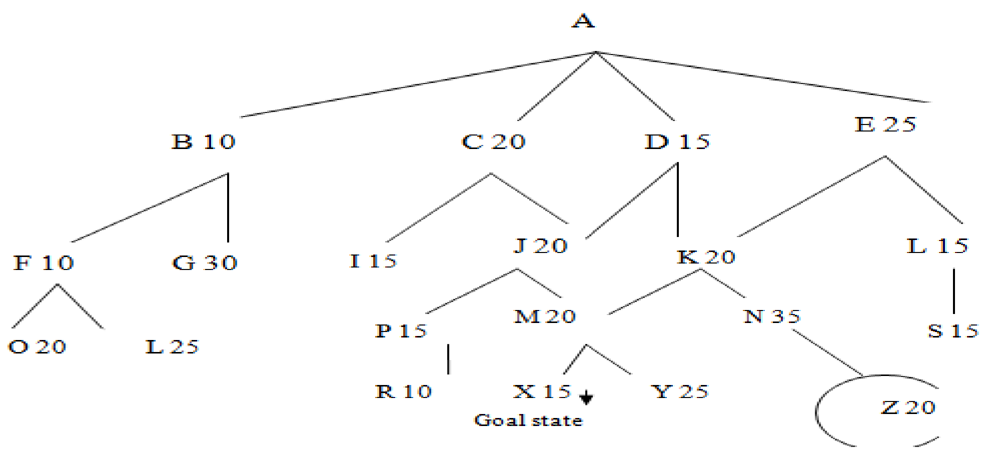
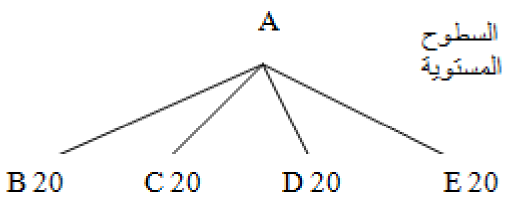
Hill Climbing Problems:

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



إتباع أعلى كلفة تؤدي إلى Z وليس إلى X وهو الهدف المطلوب الوصول إليه

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



السطوح المستوية

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 one direction.

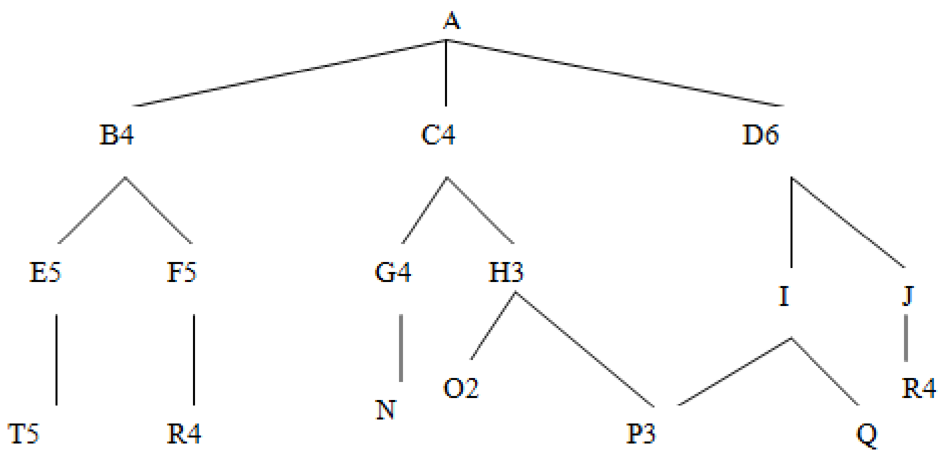
<u>Best-Frist- Search: -</u>

Best-First-search is a way of combining the advantages of both depth-first and breadth-first search into a single method.
The actual operation of the 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 node, 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) .

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
            }
```



Open=[A5]	Closed=[]
Open=[B4,C4,D6]	Closed=[A5]
Open=[C4,E5,F5,D6]	Closed=[B4,A5]
Open=[H3,G4,E5,F5,D6]	Closed=[C4,B4,A5]
Open=[O2,P3,G4,E5,F5,D6]	Closed=[H3,C4,B4,A5]
Open=[P3,G4,E5,F5,D6]	Closed=[O2,H3,C4,B4,A5]
Open=[G4,E5,F5,D6]	Closed=[P3,O2,H3,C4,B4,A5]

The solution path is: A5 -B4 -C4 -H3 –O2-P3

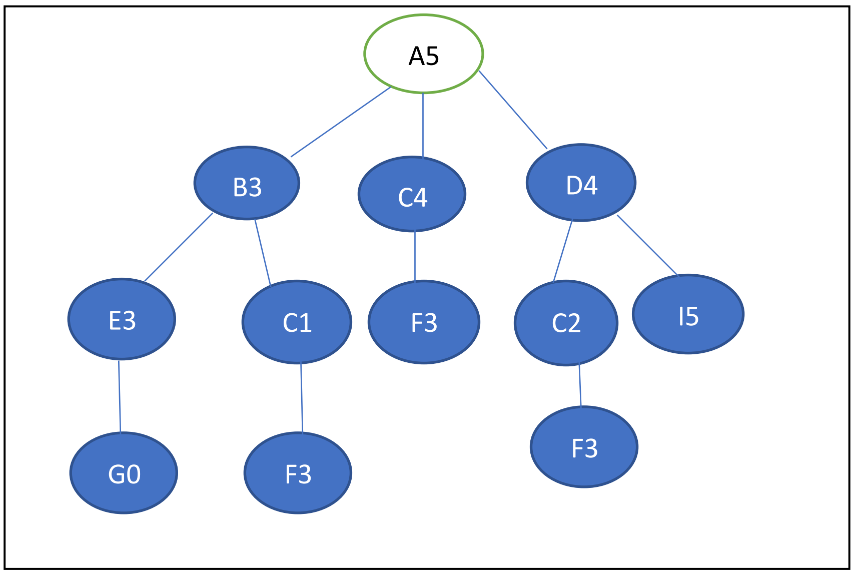
A *search algorithm

A* algorithm is simply defined 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, an
d h(n), the cost to get from the node to the goal:

$$\boxed{f(n) = g(n) + h(n).}$$

Since g(n) gives the path cost from the start node to node n, and h(n) is the estimated cost of the cheapest path from n to the goal, we have
f (n) = estimated cost of the cheapest solution through 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* search is both complete and optimal.

Example:



- Open=[A5] Closed []
- Open= [D4, B5, C6] Closed=[A5]
- Open= [C4, B5, I7] Closed= [A5, D4]
- Open= [B5, F6, I7] Closed= [A5, D4, C4]
- Open= [C3, E5, F6, I7] Closed= [A5, D4, C4, B5]
- Open= [E5, F6, I7] Closed= [A5, D4, C4, B5, C3]
- Open= [G3, F6, I7] Closed= [A5, D4, C4, B5, C3, E5]
- Close= [A5, D4, C4, B5, C3, E5, G3]

A* Algorithm Properties

1) Admissibility

●Admissibility means that h(n) is less than or equal to the cost of the minimal path from n to the goal.

●the admissibility should satisfy these two conditions:

●1-$h(n) \leq h^*(n)$

● 2-$g(n) \geq g^*(n)$

2) Monotonicity (consistency)

A heuristic function h is monotone if: -

a. For all state n_i and n_j, where n_j is a descendant of n_i

$h(n_i)-h(n_j) \leq cost(n_i,n_j)$.
Where cost (n_i,n_j) is the actual cost of going from state n_i to n_j.

b. The heuristic evaluation of the goal state is zero, or
h (goal)=0.

3) Informedness

For two A*heuristics h1and h2, if $h1(n) \leq h2(n)$, for all states n in the search space, heuristics h2 is said to be more informed than h1.