CS010 802: ARTIFICIAL INTELLIGENCE

MODULE 2

Search Methods- Best First Search- Implementation in Python- OR Graphs, The A * Algorithm, Problem Reduction- AND-OR Graphs, the AO* algorithm, Constraint Satisfaction. Games as search problem, MINIMAX search procedure, Alpha–Beta pruning.

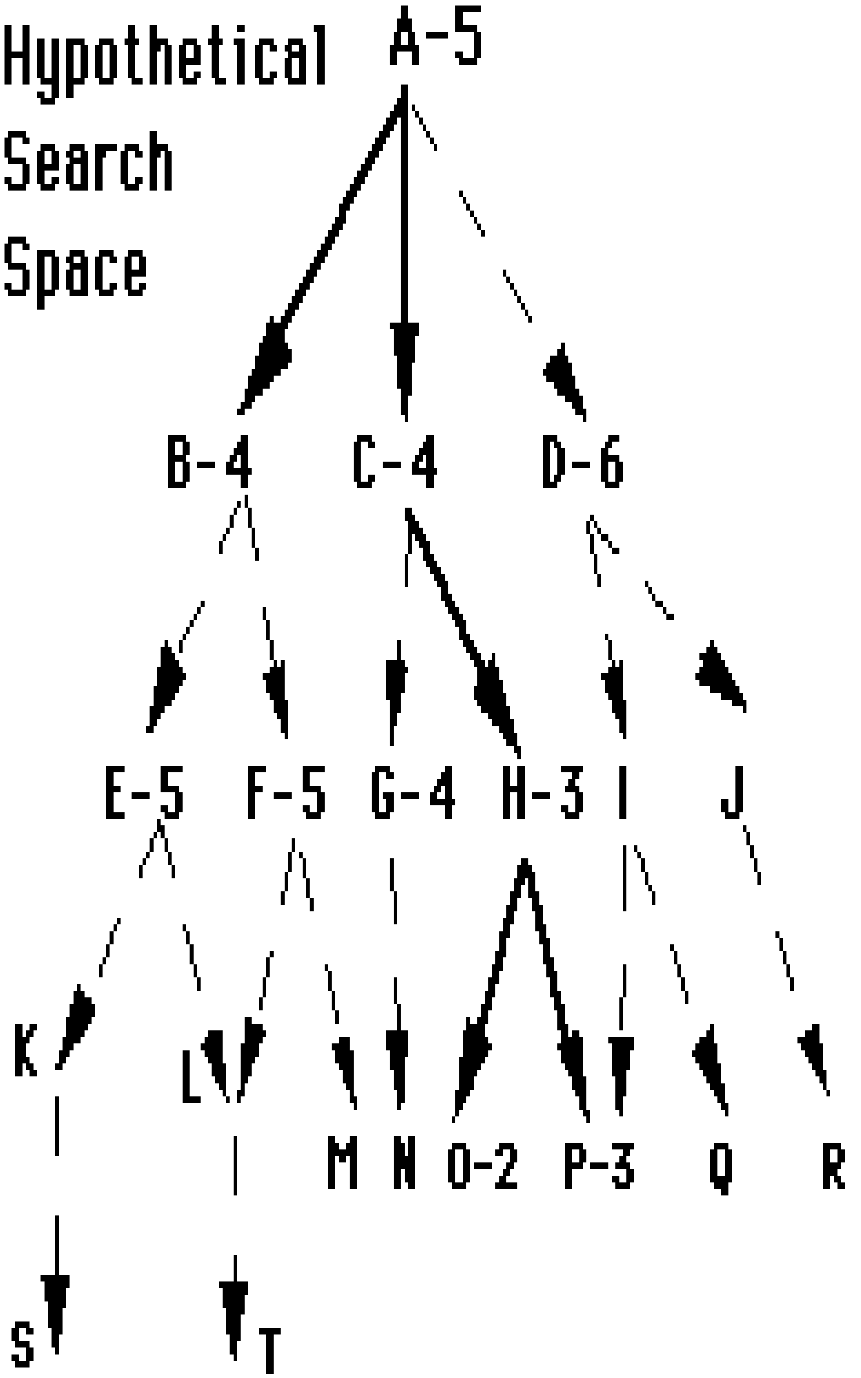
BEST FIRST SEARCH or Informed search

- Best First search is a way of combining the advantage of both DFS and BFS.
- Depth first search is a good because it allows a solution to be found without all competing branches having to be expanded.
- BFS is good because it does not get trapped on dead-end paths.
- One way of combining the two is to follow a single path at a time, but switch paths whenever some competing path looks more promising than the current one does.

- Informed search strategies use problem specific knowledge for finding solutions
  - Eg: Best first search , A* algorithm
- At each step of best first search process, we select the most promising of the nodes we have generated so far.
- If one of them is a solution, we can quit.
- If not, all those new nodes are added to the set of nodes generated so far. Again most promising branch is explored.
- But, if a solution is not found, the branch will start to look less promising branch, previously ignored. The old branch is not forgotten and the search can be return to it whenever all the others get bad enough that it is again the most promising path.
- The **best first search** algorithm uses an **open** [] list to keep track of the current fringe of the search. ie. Nodes that have been generated and the heuristic function had applied to them. OPEn is actually a priority queue in which elements with highest priority are the most promising nodes.
- **closed []** list to keep a record of states already visited or examined before.

```
procedure best_first_search {
      open := [Start];
      closed := [];
      while open != [] do {
        remove the leftmost state from open, call it X;
        if X = goal then return path from Start to X;
        else {
          generate children of X;
          for each child of X do
            case {
              the child is not on open or closed:  {
                 assign the child a heuristic value;
                 add the child to open;
              }
              the child is already on open:  {
                if this child was reached by a shorter path
                   then give the state on open the shorter path }
the child is already on closed:  {
                if this child was reached by a shorter path {
                   then remove the state from closed;
                   add this child to open;
                }
              }
            }
          put X on closed;
          re-order states on open by heuristic merit (best leftmost)
        }
      }
      return failure
  }
```

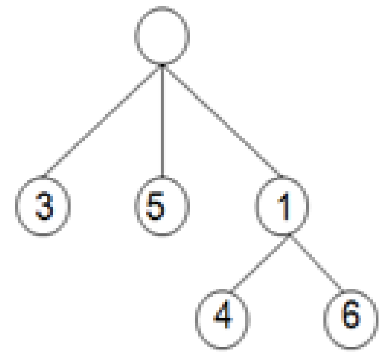
Hypothetical Search Space

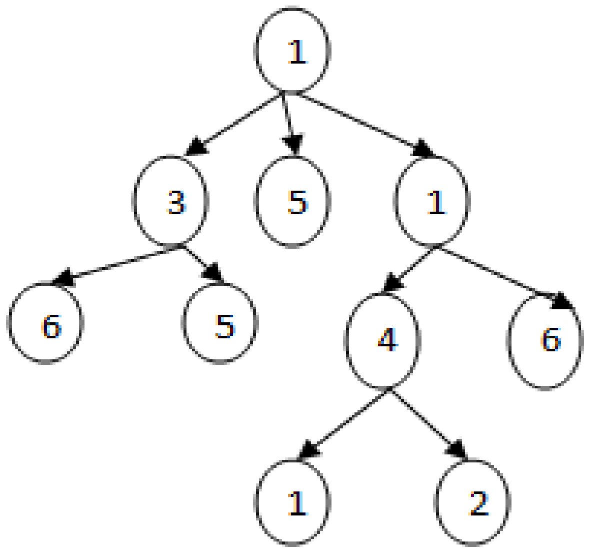


Trace of Best First Search

1. open=[A5]; closed = [];
2. eval A5; open=[B4,C4,D6];
   closed=[A5];
3. eval B4; open=[C4,E5,F5,D6];
   closed=[B4,A5];
4. eval C4; open=[H3,G4,E5,F5,D6];
   closed=[C4, B4,A5];
5. eval H3; open=[O2,P3,G4,E5,F5,D6];
   closed=[H3,C4, B4,A5];
6. eval O2; open=[P3,G4,E5,F5,D6];
   closed=[O2,H3,C4, B4,A5];
7. eval P3 = GOAL

- Consider the following hypothetical search space and the trace of the best-first-search algorithm on it. The numbers associated with the state names give the heuristic value of the state.
- Use an evaluation function f(n).
- Always choose the node from fringe that has the lowest f value.

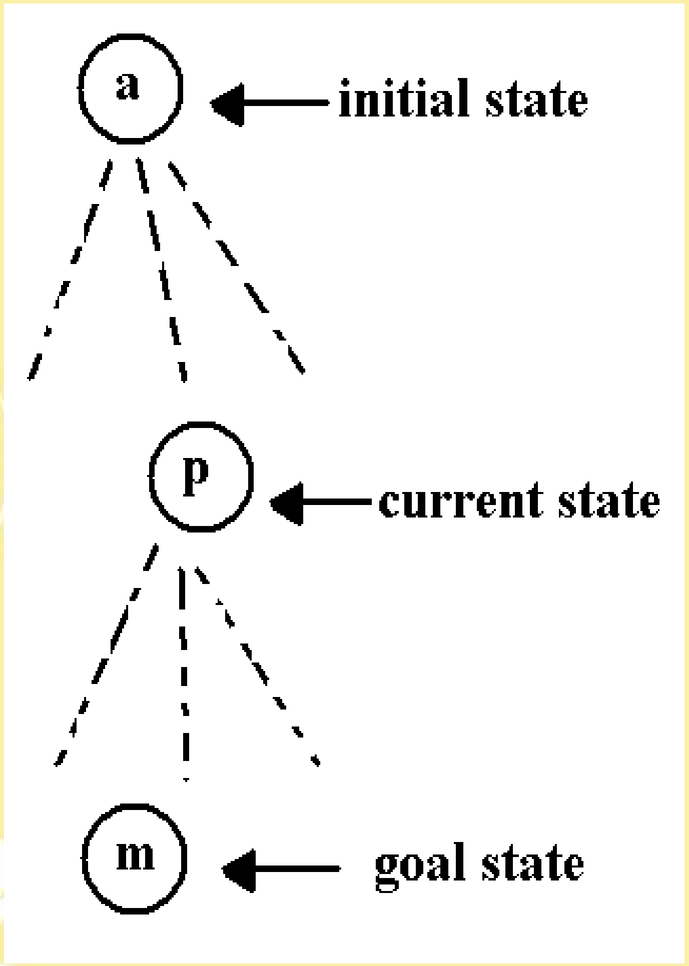




- Each node in the above shown graph represents a point in the problem space.
- Each node will contain an indication of how promising it is, a point back to the best node from which it came from and list nodes which were generated from it.
- The list of successors will make it possible, if a best path found through an already existing node, to propagate to its successors.
- We call it as **OR- GRAPH** since each of its branches represents an alternative problem- solving path.

Implementing heuristic evaluation functions

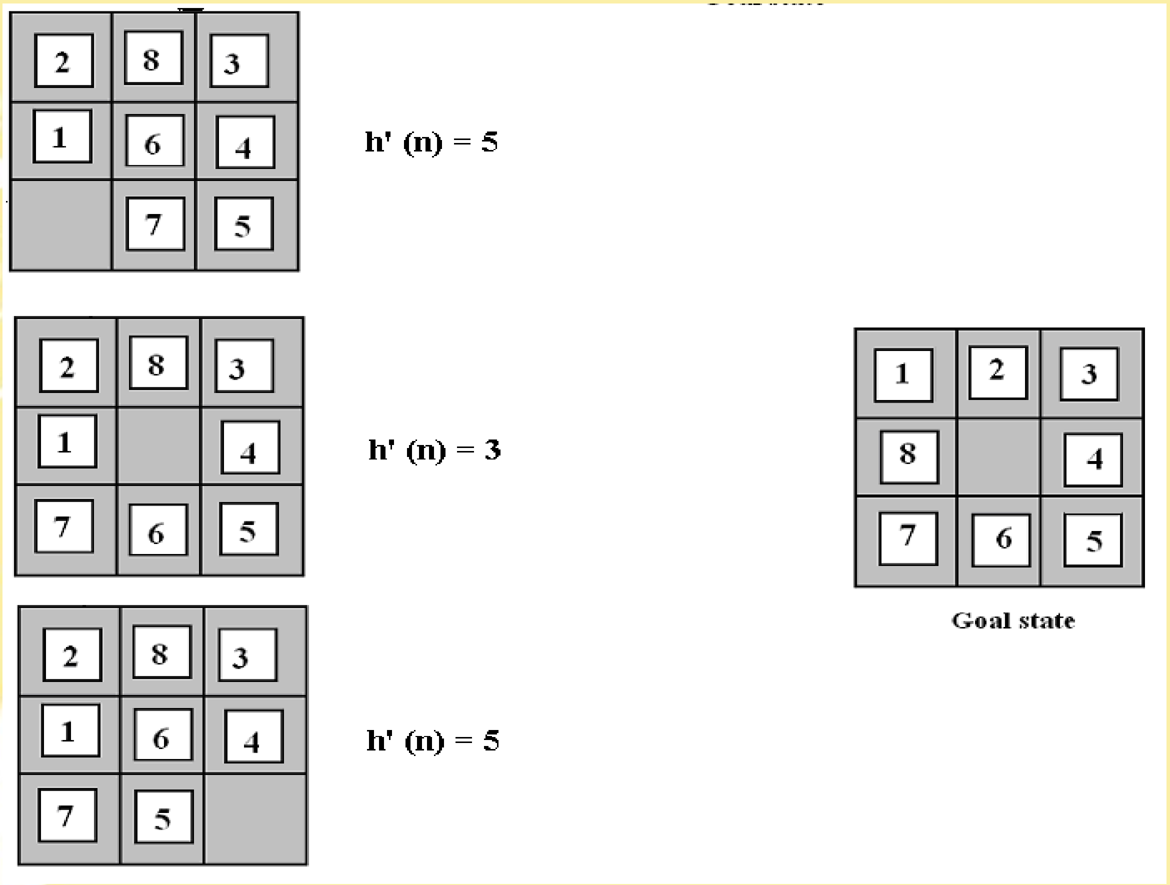
- We need a heuristic function that estimates a state. We call this function f’. it is convenient to define this function as the sum of 2 components, g and h’.



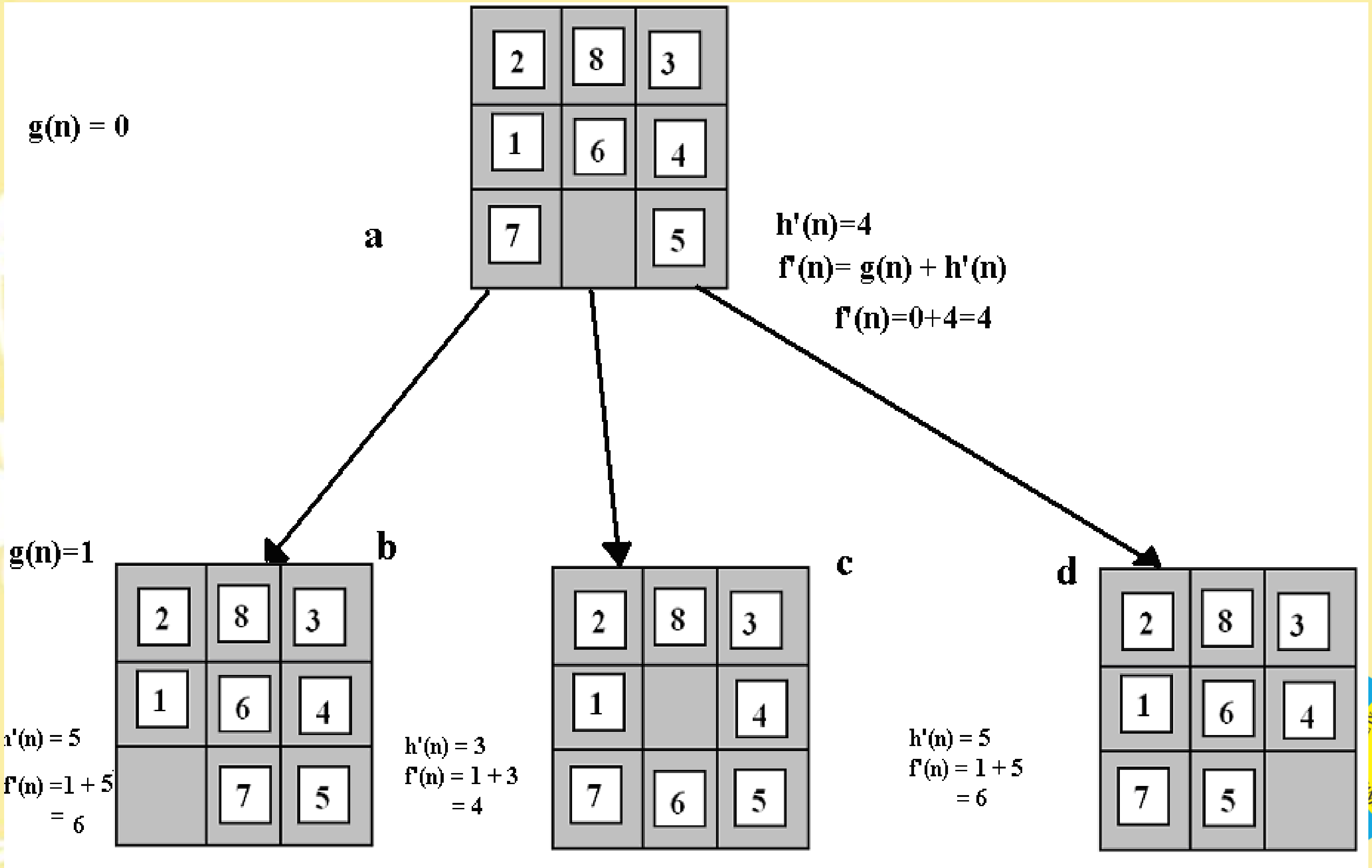
- The function **g** is the actual cost of getting from the initial state (a) to the current state (p).
- The function h’ is an estimate of the cost of getting from the current state (p) to a goal state (m). Thus
- **f’ = g + h’**

- the function f’, then, represents an estimate of the cost of getting from the initial state (a) to a goal state (m) along the path that generated the current state (p).
- If this function f’ is used for evaluating a state in the best first search algorithm, the algorithm is called A* algorithm.

- performance of a heuristic for solving 8 puzzle problems.
- Figure shows the start and goal states, along with the first 3 states generated in the search.
- We consider a heuristic that counts the tiles out of place in each state when it is compared with the goal.



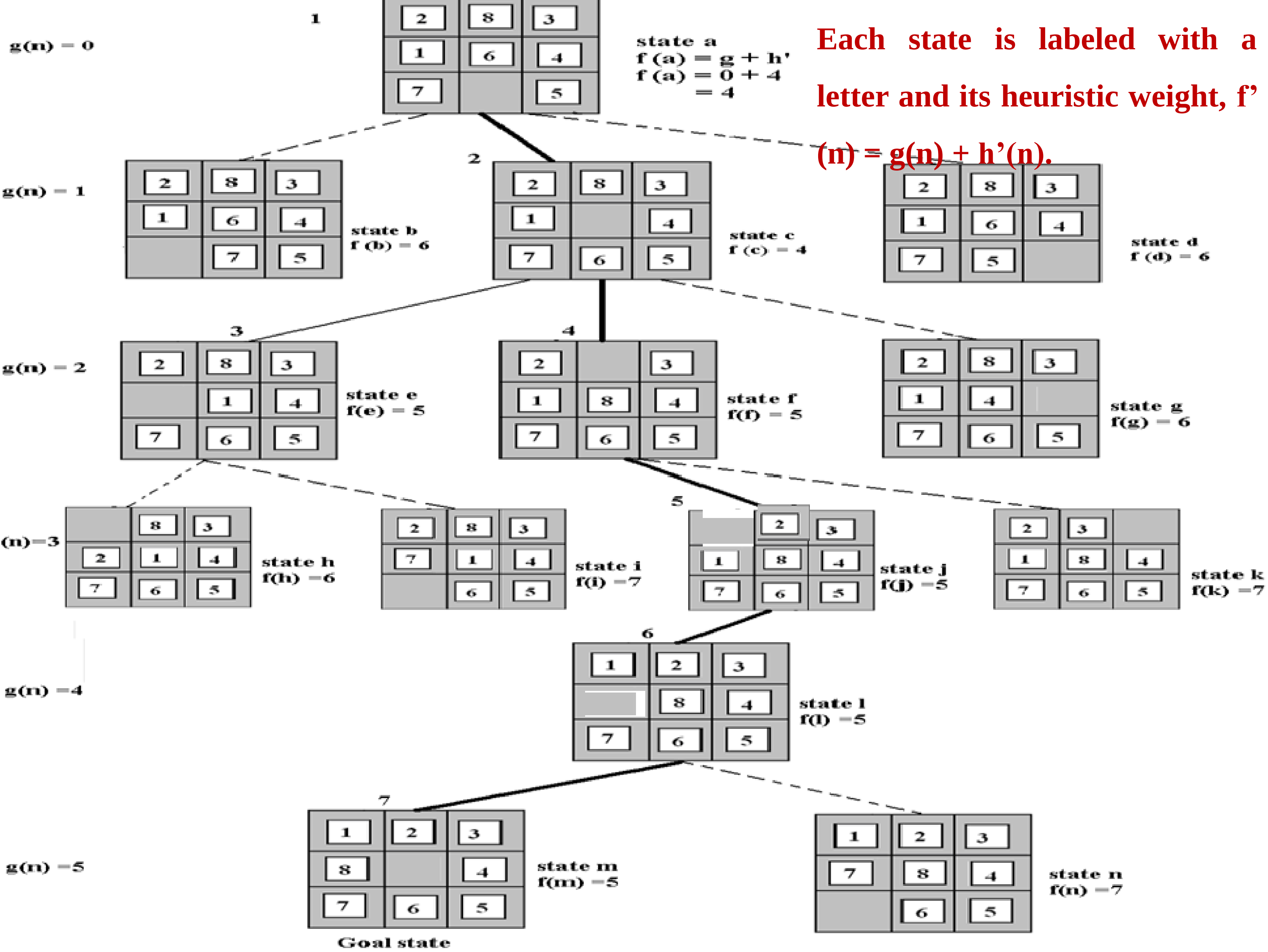
- The distance from staring state to its descendents (g) can be measured by maintaining a **depth count** for each state.
- This count is 0 for the beginning state and is incremented by 1 for each level of the search.
- It records the actual number of moves that have been used to go from the start state in a search to each descendent.



- The best first search of 8 puzzle graph using f as defined above appears below.
-

Open	Closed
a4	
c4, b6, d6	a4
e5, f5, b6, d6, g6	a4, c4
f5, h6, b6, d6, g6, i7	a4, c4, e5
j5, h6, b6, d6, g6, k7, i7	a4, c4, e5, f5
l5, h6, b6, d6, g6, k7, i7	a4, c4, e5,f5, j5
m5, h6, b6, d6, g6, n7, k7, i7	a4, c4, e5, f5, j5, l5
success, m= goal	

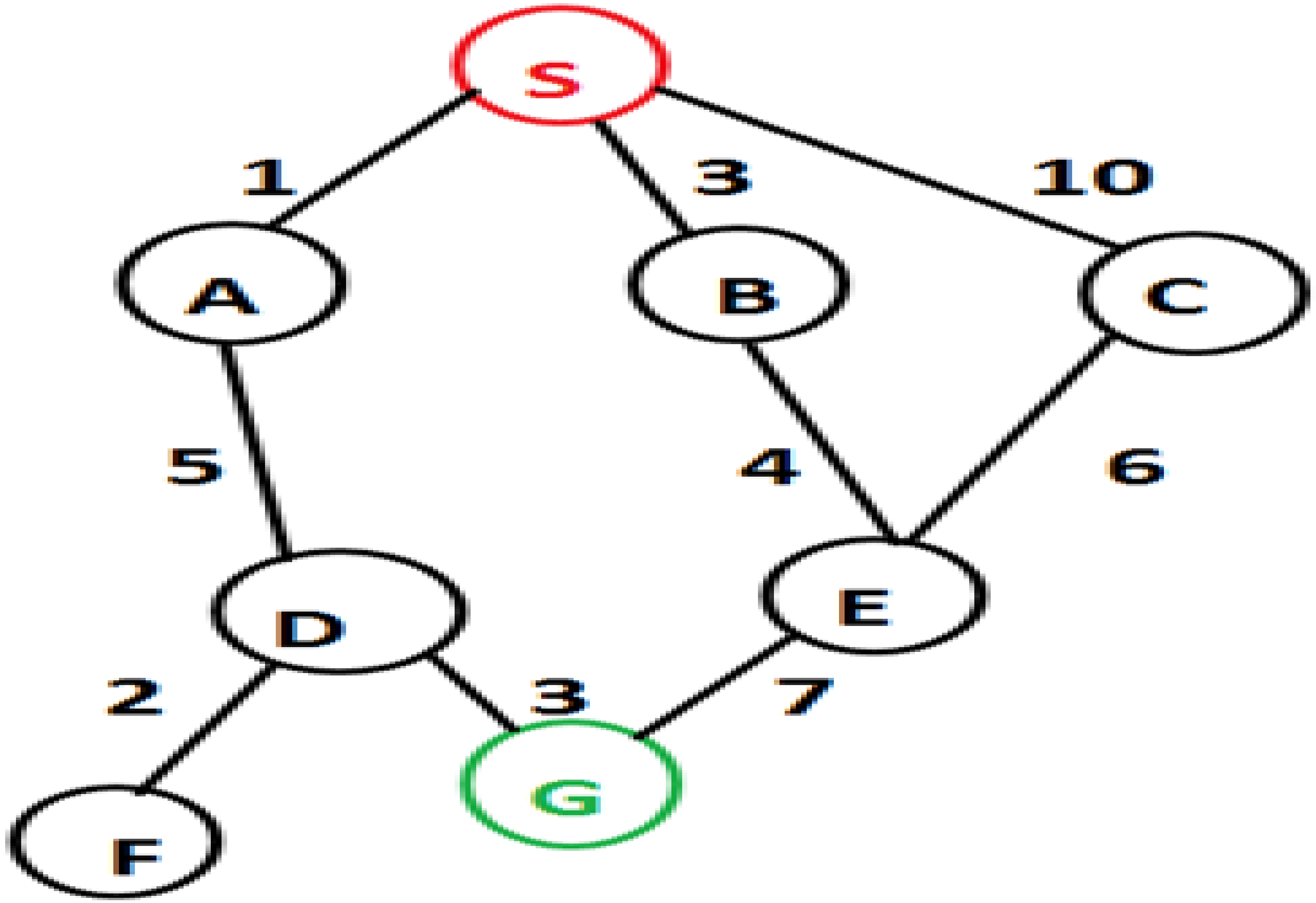
Each state is labeled with a letter and its heuristic weight, $f'(n) = g(n) + h'(n)$.

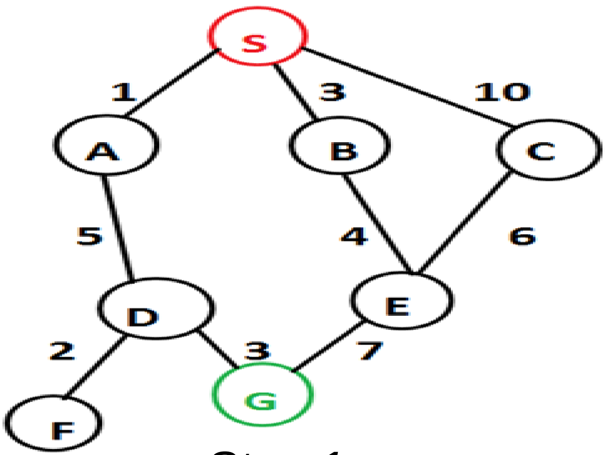


A* ALGORITHM

```
OPEN=[Start]                    // start_node S
Set  the start node g=0 [cost of getting from Initial node to Current node ]
Set h’ value to whatever it is.          f’ =g+h’      ;  0+h’ =h’
CLOSED=[]        // empty
Until the Goal node is found, repeat the following procedure
{     2.1   If OPEN=[ ]   exit with failure
      2.2    else                  {
a) Select the first node on OPEN, remove it from OPEN and put it on CLOSED.
   Call this node as current_node.
b) If(current_node is Goal node)
{    Exit successfully with the solution obtained by tracing the path    along the
   pointers from Start node S to Goal node G    }
c) If((current_node is not  Goal node)
{    Expand the current_node and Generate its successors. Move current_node
   to the CLOSED. Create a pointer back to Current Node from all successors.
  Compute  g(Successors)  =  g(current_node)+the  cost  of  getting  from
   current_node to successors.
                         f’=g+h’   }
```

```
d) for( each successor)   {
  I ) If(the successor is in the CLOSED list and current f' value is lower)
{ Update the successor with the new lower f' value.
   Remove successor from CLOSED and add to OPEN.
   Change successors parent to current_node.
}
II) If ( successor in the OPEN list and current f' value is lower)
{Update the successor with the new lower f' value.
      Change successors parent to current_node.
}
III ) If ( successor not  in the OPEN or CLOSED)
{Add successors to OPEN }
}
e. Reorder the list OPEN according to heuristic merit
                  }
}
STOP
```


S
1
3
10
A
B
C
5
4
6
D
E
2
3
7
G
F



Step 1:

OL: S (NULL ; 0)

OL: Open List

CL: EMPTY

CL: Closed List

Step 2:

OL: A (S,1) , B(S,3) , C(S,10)

CL:S (NULL; 0)

Step 3:

OL: B(S,3), C(S,10), D(A,6)

CL: S(NULL; 0),A(S,1)

Step 4:

OL:C(S,10), D(A,6), E(B,7)

CL: S(NULL; 0),A(S,1),B(S,3) ,

Step 5:

OL: D(A,6), E(B,7)

CL: S(NULL; 0),A(S,1),B(S,3) , C(S,10)

Step 6:

OL: E(B,7), F(D,8)

CL: S(NULL; 0),A(S,1),B(S,3) C(S,10), D(A,6)

Step 7:

OL: F(D,8),G(E,14)

CL: S(NULL; 0),A(S,1),B(S,3) , C(S,10), D(A,6), E(B,7)

Step 8:

OL: G(E,14)

CL: S(NULL; 0),A(S,1),B(S,3) , C(S,10),D(A,6), E(B,7), F(D,8)

Step 9:

OL: --

CL: S(NULL; 0),A(S,1),B(S,3) , C(S,10),D(A,6), E(B,7), F(D,8),G(E,14)