State Space Representation for Heuristic Search

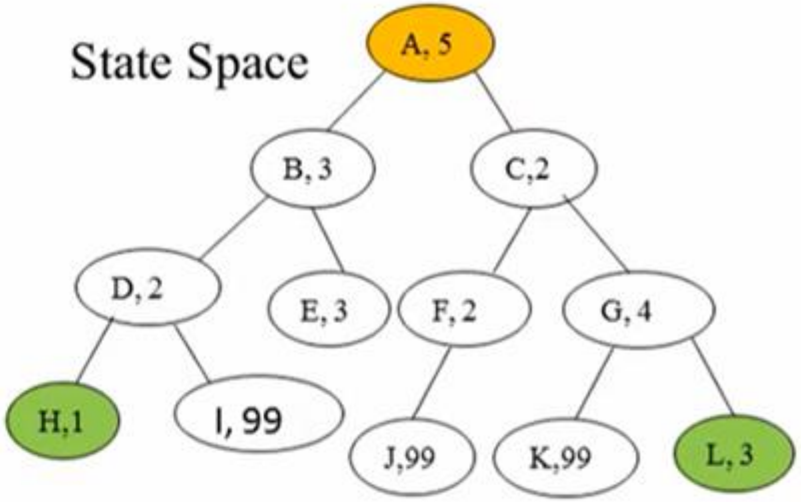
Example: State Space Representation

Let A,B,C,D, …… represents a state in a solution space. The following moves are legal.

A5 to B3 and C2
B3 to D2 and E3
C2 to F2 and G4
D2 to H1 and I99
G4 to K99 and L3

Start = {A}
Goal = {H and L}

Note: The numeric value after a character represents its heuristic value.
A5 → A is node and 5 is its heuristic value.



S ={
(A,5):[(B,3),(C,2)],
(B,3):[(D,2),(E,3)],
(C,2):[(F,2),(G,4)],
(D,2):[(H,1),(I,99)],
(G,4):[(K,99),(L,3)]
}

Hill Climbing Algorithm

Hill Climbing is a strategy of finding the node with ***better heuristic value*** than the current node value.

The goal is to solve problem using ***optimization principle***. In case of *minimization* problem the goal is to find the *minimum cost solution* where as in *maximization* problem it to find *maximum gain/profit in the solution*.

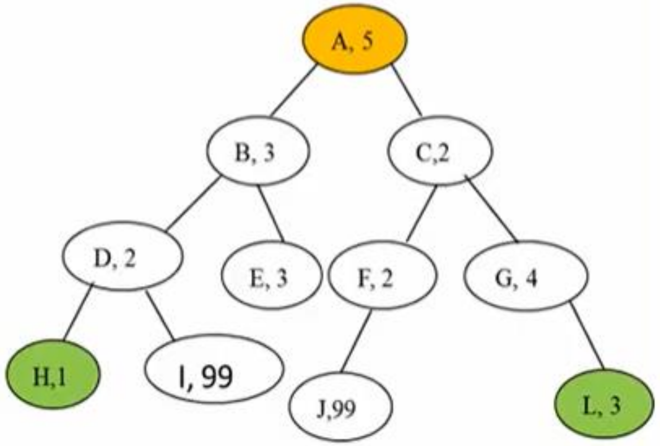
Examples:

In **TSP** the goal is to get a tour with *minimum* cost.

In **Production line of Industry** the goal is to get *maximum* output from the machine.

```
Def Hill_Climbing(Start)
N={Start}
Child = {MOVEGEN(N)}
SORT(Child)
newNode=Pick front node from Child
While (h(newNode) <= h(N)) do
            N=newNode
            Child = {MOVEGEN(N)}
            SORT(Child)
            newNode = Pick front node from Child
End While
Return newNode
```

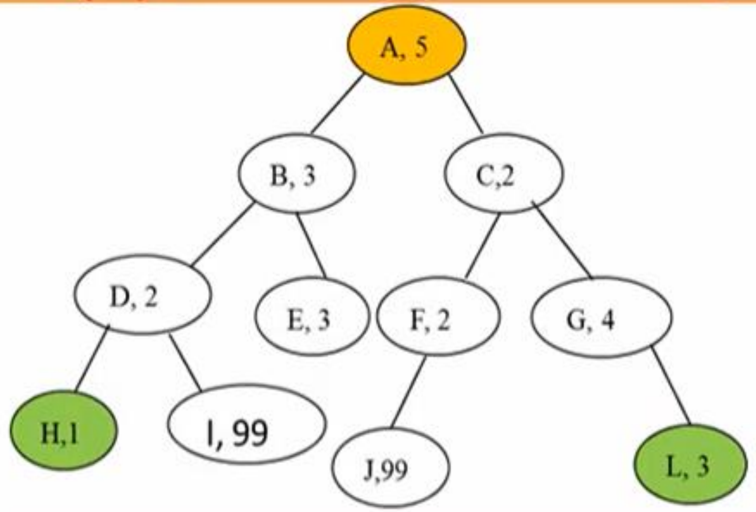
```
Def MOVEGEN(N)
Succ ={}
For N in S do
  Succ= Succ U {Children of N}
Return Succ

Def SORT(L)
Sort list L in ascending order of the Heuristic
value of Nodes using function h(n)
Return L
```



Hill Climbing Algorithm

```
Def Hill_Climbing(Start)
N={Start}
Child = {MOVEGEN(N)}
SORT(Child)
newNode=Pick front node from Child
While (h(newNode) <= h(N)) do
        N=newNode
        Child = {MOVEGEN(N)}
        SORT(Child)
        newNode = Pick front node from Child
End While
Return newNode
```

```
Def Hill_Climbing(A)
N={(A,5)}
Child = {(B,3), (C,2)}
Child={(C,2), (B,3)}
newNode=(C,2)
While (2 <= 5 ) do
        N= (C,2)
        Child = {(F,2), (G,4)}
        Child = {(F,2), (G,4)}
        newNode = (F,2)
End While
Return newNode
```



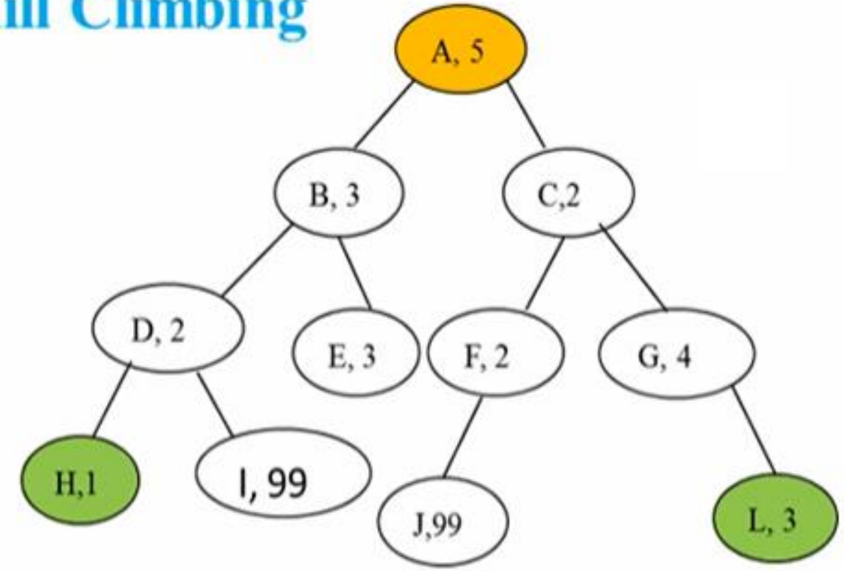
```
Def Hill_Climbing(Start)
N={Start}
Child = {MOVEGEN(N)}
SORT(Child)
newNode=Pick front node from Child
While (h(newNode) < h(N)) do
        N=newNode
        Child = {MOVEGEN(N)}
        SORT(Child)
        newNode = Pick front node from Child
End While
Return newNode
```

```
Def Hill_Climbing(A)



While (2 <= 2 ) do
        N= (F,2)
        Child = {(J,99)}
        Child = {(J,99)}
        newNode = (J,99)
End While
Return newNode
```

```
Def Hill_Climbing(A)



While (99 <= 2 ) do
        N= (F,2)
        Child = {(J,99)}
        Child = {(J,99)}
        newNode = (J,99)
End While
Return newNode
```

Hill Climbing Algorithm

Analysis of Hill Climbing



1. The Hill Climbing (HC) follows *Steepest Gradient ascent* using the heuristic function h(n).

2. The HC may get stuck in local Maxima or local Minima, i.e. Local Optima.

3. The HC does not guarantee the Solution so it is not complete.

4. The HC takes linear Space across the path while finding solution.

Hill Climbing

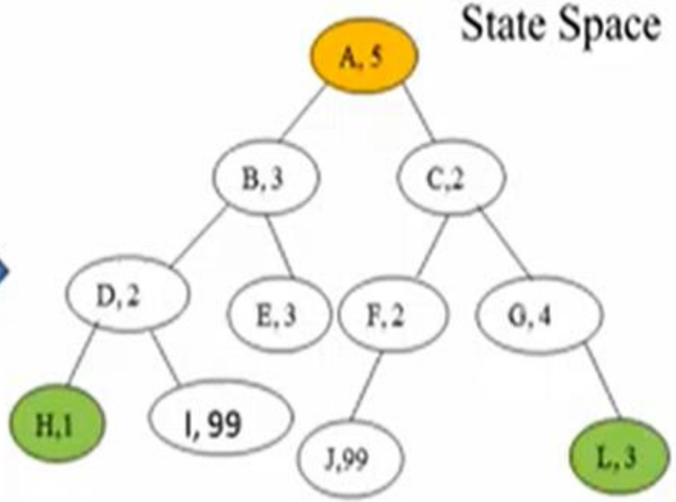
The State Space for Search Algorithm

Let A,B,C,D, …… represents a state in a solution space. The following moves are legal.
A5 to B3 and C2
B3 to D2 and E3
C2 to F2 and G4
D2 to H1 and I99
F2 to J99
G4 to K99 and L3

Start = {A}
Goal = {H and L}

Note: The numeric value after a character represents its heuristic value.
A5 → A is node and 5 is its heuristic value.

State Space



Key

Value

Dictionary

S ={ A:[[B,3],[C,2]],   B:[[D,2],[E,3]],   C:[[F,2],[G,4]],   D:[[H,1],[I,99]], F: [[J,99]] ,G:[[K,99],[L,3]] }

Hill Climbing

The **GOALTEST() , MOVEGEN(), Heu()** and **APPEND()**

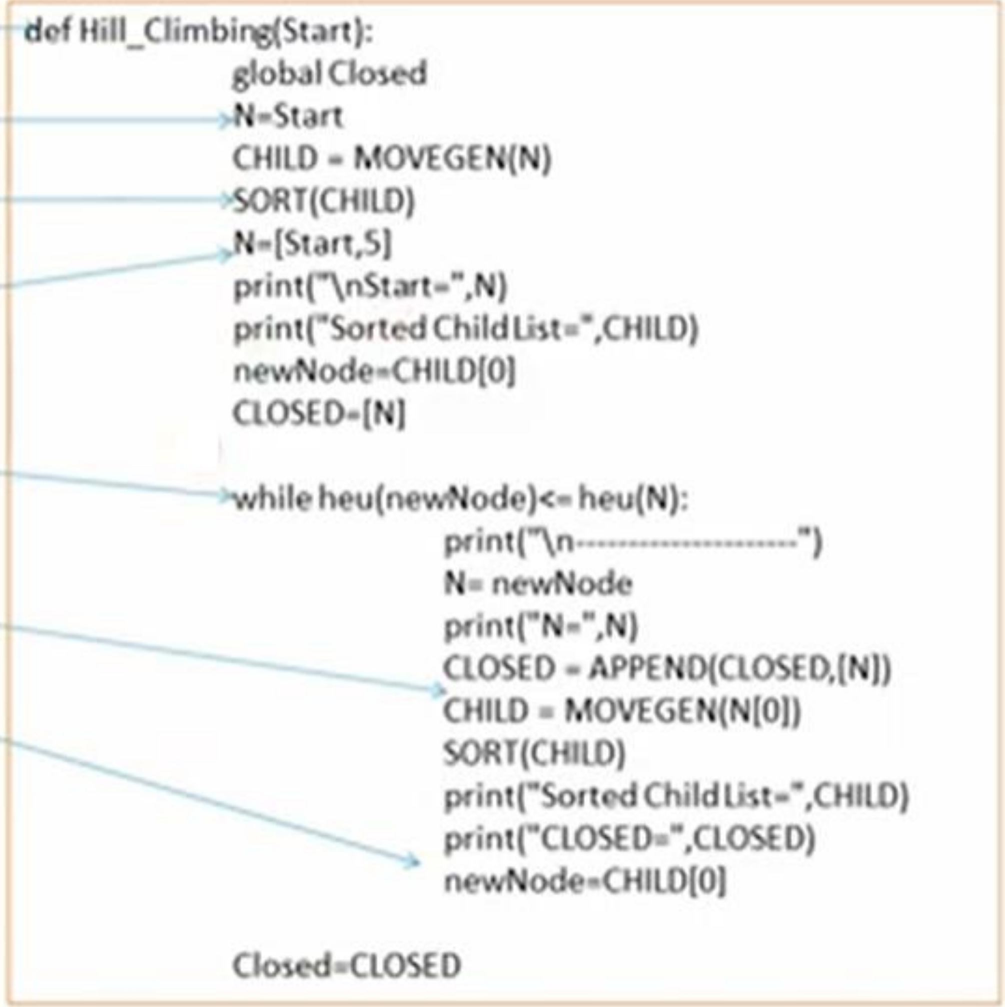
SuccList ={ 'A':[['B',3],['C',2]], 'B':[['D',2],['E',3]], 'C':[['F',2],['G',4]], 'D':[['H',1],['I',99]],'F': [['J',99]] ,'G':[['K',99],['L',3]]}
Start='A', Closed = list()

```
Def MOVEGEN(N)
Succ ={}
For N in S do
  Succ= Succ U {Children of N}
Return Succ

Def SORT(L)
Sort list L in ascending order of the Heuristic value of Nodes
using function h(n)
Return L

Def h (Node)
   return Heuristic of Node

Def APPEND(list1, list2)
New_list = list1 + list2
Return New_list
```

```
def MOVEGEN(N):
              New_list=list()
              if N in SuccList.keys():
                            New_list=SuccList[N]

              return New_list

def SORT(L):
              L.sort(key = lambda x: x[1])
              return L

def heu(Node):
              return Node[1]

def APPEND(L1,L2):
              New_list=list(L1)+list(L2)
              return New_list
```

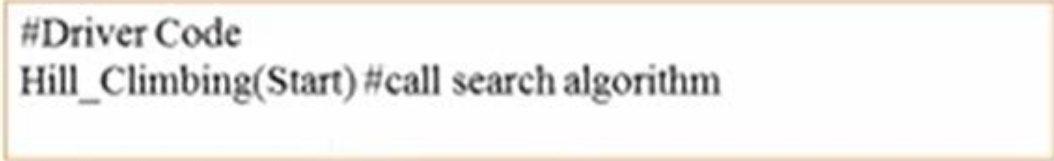
Code of Hill_Climbing

```
Def Hill_Climbing(Start)
N={Start}
Child = {MOVEGEN(N)}
SORT(Child)
newNode=Pick front node from Child
While (h(newNode) <= h(N)) do
          N=newNode
          Child = {MOVEGEN(N)}
          SORT(Child)
          newNode = Pick front node from Child

End While
Return newNode
```

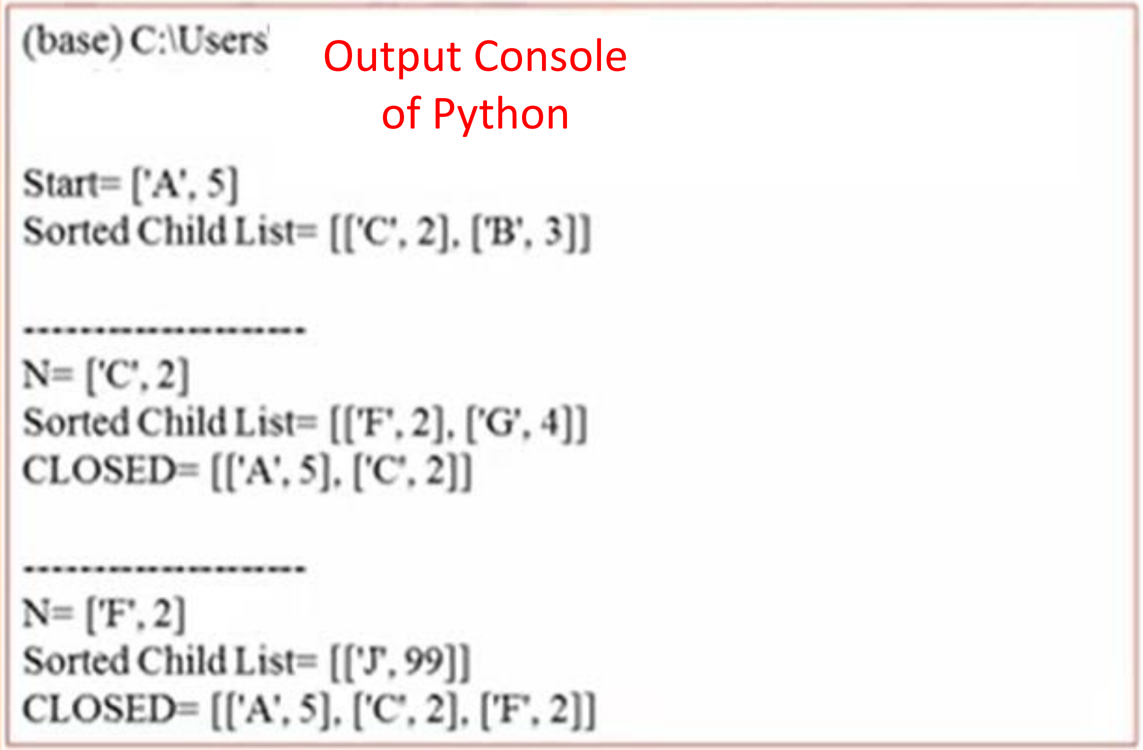


```
def Hill_Climbing(Start):
          global Closed
          N=Start
          CHILD = MOVEGEN(N)
          SORT(CHILD)
          N=[Start,5]
          print("\nStart=",N)
          print("Sorted Child List=",CHILD)
          newNode=CHILD[0]
          CLOSED=[N]

          while heu(newNode)<= heu(N):
                    print("\n-------------------")
                    N= newNode
                    print("N=",N)
                    CLOSED = APPEND(CLOSED,[N])
                    CHILD = MOVEGEN(N[0])
                    SORT(CHILD)
                    print("Sorted Child List=",CHILD)
                    print("CLOSED=",CLOSED)
                    newNode=CHILD[0]

          Closed=CLOSED
```

Run Hill_Climbing ()

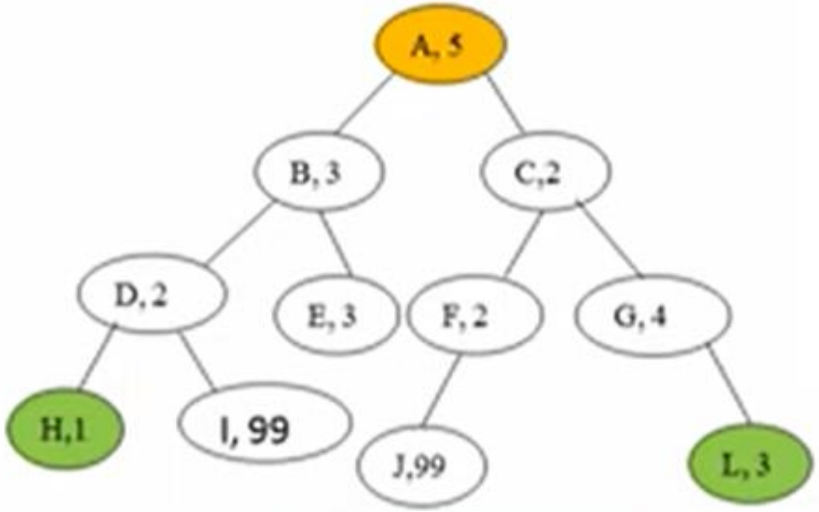


```
#Driver Code
Hill_Climbing(Start) #call search algorithm
```



Output Console of Python

```
(base) C:\Users


Start= ['A', 5]
Sorted Child List= [['C', 2], ['B', 3]]

--------------------
N= ['C', 2]
Sorted Child List= [['F', 2], ['G', 4]]
CLOSED= [['A', 5], ['C', 2]]

--------------------
N= ['F', 2]
Sorted Child List= [['J', 99]]
CLOSED= [['A', 5], ['C', 2], ['F', 2]]
```



```
S =
{
(A,5):[(B,3),(C,2)],
(B,3):[(D,2),(E,3)],
(C,2):[(F,2),(G,4)],
(D,2):[(H,1),(I,99)],
(F,2): [(J,99)]
(G,4):[(K,99),(L,3)]
}
```