Hill Climbing Search

Hill Climbing is a heuristic search used for mathematical optimization problems in the field of Artificial Intelligence.

In examining a search tree, hill climbing will move to the first successor node that is “better” than the current node—in other words, the first node that it comes across with a heuristic value lower than that of the current node. Now hill climbing proceeds as with depth-first search, but at each step, the new nodes to be added to the queue are sorted into order of distance from the goal.

HCS.png

Simple Hill Climbing

Simple hill climbing is the simplest way to implement a hill-climbing algorithm. It only evaluates the neighbor’s node state at a time and selects the first one which optimizes current cost and set it as a current state. It only checks it’s one successor state, and if it finds better than the current state, then move else be in the same state.

- **Step 1:** Evaluate the initial state, if it is goal state then return success and Stop.
- **Step 2:** Loop Until a solution is found or there is no new operator left to apply.
- **Step 3:** Select and apply an operator to the current state.
- **Step 4:** Check new state:
    - **i.** If it is goal state, then return success and quit.
    - **ii.** else if it is better than the current state then assigns new state as a current state.
    - **iii.** else if not better than the current state, then return to step 2.
- **Step 5:** Exit.

Steepest-Ascent hill climbing

The steepest-Ascent algorithm is a variation of the simple hill-climbing algorithm. This algorithm examines all the neighboring nodes of the current state and selects one neighbor node which is closest to the goal state. This algorithm consumes more time as it searches for multiple neighbors.

- **Step 1:** Evaluate the initial state, if it is goal state then return success and stop, else make the current state as your initial state.
- **Step 2:** Loop until a solution is found or the current state does not change.
    - **i.** Let S be a state such that any successor of the current state will be better than it.
    - **ii.** For each operator that applies to the current state;
        - Apply the new operator and generate a new state.
        - Evaluate the new state.
        - If it is goal state, then return it and quit, else compare it to the S.
        - If it is better than S, then set new state as S.
        - If the S is better than the current state, then set the current state to S.
- **Step 3:** Exit.

Stochastic hill climbing

Stochastic hill climbing does not examine for all its neighbors before moving. Rather, this search algorithm selects one neighbor node at random and evaluate it as a current state or examine another state.

Task:

tree.png

In [2]:

```python
#Implement Simple HCS
MyGraph = {
    'A' : ['B','C'],
    'B' : ['A','C','D'],
    'C' : ['A','B','E'],
    'D' : ['B','E','F'],
    'E' : ['C','D','F'],
    'F' : ['D','E'],
}
HeuristicVales={
    'A' : 25,
    'B' : 8,
    'C' : 20,
    'D' : 6,
    'E' : 12,
    'F' : 0,
}
StartNode='A'
GoalNode='F'

def HCS_simple():
    Node=StartNode
    print('Traversing::')
    while True:
        print(Node)
        if Node==GoalNode:
            print('Goal Node Found!')
            break
        minHeurNode=Node
        for n in MyGraph[Node]:
            if HeuristicVales[minHeurNode] > HeuristicVales[n]:
                minHeurNode=n
        if minHeurNode==Node:
            print('Maxima Goal point '+Node+' is Found!')
            break
        Node=minHeurNode
HCS_simple()
```

```
Traversing::
A
B
D
F
Goal Node Found!
```

In [8]:

```python
#Implement Steepest-Ascent HCS
MyGraph = {
    'A' : ['B','C'],
    'B' : ['A','C','D'],
    'C' : ['A','B','E'],
    'D' : ['B','E','F'],
    'E' : ['C','D','F'],
    'F' : ['D','E'],
}
HeuristicValues={
    'A' : 25,
    'B' : 8,
    'C' : 20,
    'D' : 6,
    'E' : 12,
    'F' : 0,
}
StartNode='A'
GoalNode='F'

def HCS_Steepest_Ascent():
    Node=StartNode
    Queue=[Node]
    Visited=[]
    print('Traversing::')
    while Queue:
        Node=Queue.pop()
        print(Node)
        if Node==GoalNode:
            print('Goal Node Found!')
            break

        childList=[]
        for x in reversed(MyGraph[Node]):
            if x not in Visited and HeuristicValues[Node]>HeuristicValues[x
                childList.append(x)
        Queue.extend(childList)
        Visited.append(Node)

HCS_Steepest_Ascent()
```

```
Traversing::
A
B
D
F
Goal Node Found!
```

In [9]:
```python
#Implement Stochastic HCS
import random
MyGraph = {
    'A' : ['B','C'],
    'B' : ['A','C','D'],
    'C' : ['A','B','E'],
    'D' : ['B','E','F'],
    'E' : ['C','D','F'],
    'F' : ['D','E'],
}
HeuristicValues={
    'A' : 25,
    'B' : 8,
    'C' : 20,
    'D' : 6,
    'E' : 12,
    'F' : 0,
}
StartNode='A'
GoalNode='F'

def HCS_stochastic():
    Node=StartNode
    print('Traversing::')
    while True:
        print(Node)
        if Node==GoalNode:
            print('Goal Node Found!')
            break
        randChildNode=MyGraph[Node][random.randint(0,len(MyGraph[Node])-1)]
        if randChildNode==Node:
            print('Maxima Goal point '+Node+' is Found!')
            break
        Node=randChildNode
HCS_stochastic()
```

```
Traversing::
A
B
D
E
D
B
A
B
C
B
D
E
D
F
Goal Node Found!
```