**Greedy Best First Search**

Greedy best-first search algorithm always selects the path which appears best at that moment. In this algorithm, we expand the node which is closest to the goal node and the closest cost is estimated by heuristic function, i.e.

f(n)= h(n).

Were, h(n)= estimated cost from node n to the goal.

The greedy best first algorithm is implemented by the priority queue.

It is the combination of depth-first search and breadth-first search algorithms. It uses the heuristic function and search. Best-first search allows us to take the advantages of both algorithms. With the help of best-first search, at each step, we can choose the most promising node.

**Greedy Best first search algorithm:**

**Step 1:** Place the starting node into the OPEN list.

**Step 2:** If the OPEN list is empty, Stop and return failure.

**Step 3:** Remove the node n, from the OPEN list which has the lowest value of h(n), and places it in the CLOSED list.

**Step 4:** Expand the node n, and generate the successors of node n.

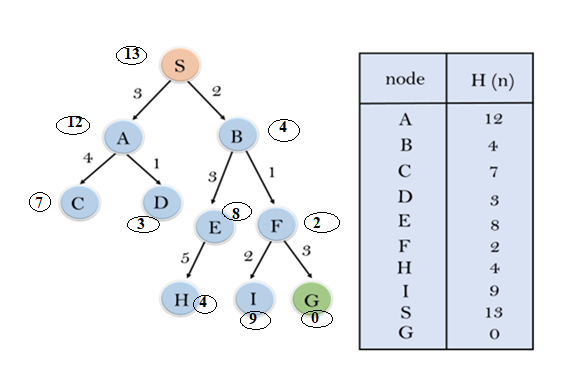
**Step 5:** Check each successor of node n, and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.

**Step 6:** For each successor node, algorithm checks for evaluation function f(n), and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both lists, then add it to the OPEN list.

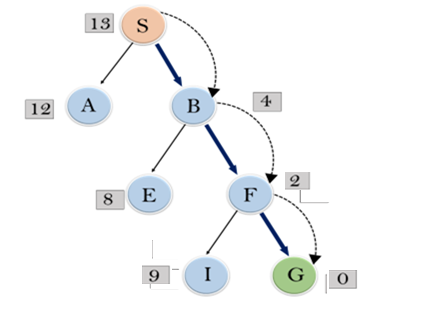
**Step 7:** Return to Step 2.

**Example1:**

Consider the below search problem, and we will traverse it using greedy best-first search. At each iteration, each node is expanded using evaluation function f(n)=h(n) , which is given in the below table.



In this search example, we are using two lists which are **OPEN** and **CLOSED** Lists. Following are the iteration for traversing the above example.



**Expand the nodes of S and put in the CLOSED list**

**Initialization:** Open [A, B], Closed [S]

**Iteration 1:** Open [A], Closed [S, B]

**Iteration 2:** Open [E, F, A], Closed [S, B]  
                  : Open [E, A], Closed [S, B, F]

**Iteration 3:** Open [I, G, E, A], Closed [S, B, F]  
                  : Open [I, E, A], Closed [S, B, F, G]

Hence the final solution path will be: **S----> B----->F----> G**

**CODE:**

graph = {

's': ['a', 'b'],

'a': ['c', 'd'],

'b': ['e', 'f'],

'c': [],

'd': [],

'e': ['h'],

'f': ['i', 'g'],

'g': [],

'h': [],

'i': []

}

heuristic = {

's': 13,

'a': 12,

'b': 4,

'c': 7,

'd': 3,

'e': 8,

'f': 2,

'g': 0,

'h': 4,

'i': 9

}

def gbfs(graph, heuristic, start, goal):

visited = set()

queue = [(heuristic[start], [start])] # Priority queue (heuristic, path)

while queue:

queue.sort(key=lambda x: x[0])

(h, path) = queue.pop(0)

current\_node = path[-1]

if current\_node == goal:

return path

visited.add(current\_node)

for neighbor in graph[current\_node]:

if neighbor not in visited:

new\_path = path + [neighbor]

queue.append((heuristic[neighbor], new\_path))

return None

start\_node = input("Enter the start node: ")

goal\_node = input("Enter the goal node: ")

traversed\_path = gbfs(graph, heuristic, start\_node, goal\_node)

# Output the result

if traversed\_path:

print("Path found and traversed path is:", traversed\_path)

else:

print("Path not found.")