In [3]:

```
import heapq
graph={
    # to store g-score and h-score
    # list first value is the g-score, second value is the h-score,i.e., heuristic
    'A':{'B':[2,2],'C':[3,2]},
    'B':{'D':[3,5],'E':[1,1]},
    'C':{'F':[2,0]},
    'D':{},
    'E':{'F':[1,0]},
    'F':{}
}

# The algorithm will retrieve the graph as follow:
#graph['A'] this return {'B':[2,2],'C':[3,2]}
#graph['A']['B'] this return [2,2]
#graph['A']['B'][0] return the edge length
#graph['A']['B'][1] return the distance of the node to destination
def astar(graph,start_node,end_node):
    # astar: F=G+H, we name F as f_distance, G as g_distance, H as heuristic
    #Assign all the nodes, a f_distance value as infinity as initial value
    f_distance={node:float('inf') for node in graph}
    #The f_ditance value of start node is 0
    f_distance[start_node]=0
    #Assign all the nodes, a g_distance value as infinity as initial value
    g_distance={node:float('inf') for node in graph}
    #The g_ditance value of start node is 0
    g_distance[start_node]=0

    #Keep the track of parent node in came_form
    came_from={node:None for node in graph}
    came_from[start_node]=start_node
    queue=[(0,start_node)] #use queue as list
    while queue:
        f_distance,current_node=heapq.heappop(queue)
        if current_node == end_node:
            print('found the end_node')
            return f_distance, came_from
        #for all the neighbors of the current node calculate g_distance
        for next_node,weights in graph[current_node].items():
            temp_g_distance=g_distance[current_node]+weights[0]
            #g_distance of current node is less than the g_distance of neighbor
            #Update the g_distance of next node to the smaller distance value.
            if temp_g_distance<g_distance[next_node]:
                g_distance[next_node]=temp_g_distance
                heuristic=weights[1]
                f_distance=temp_g_distance+heuristic
                came_from[next_node]=current_node
                heapq.heappush(queue,(f_distance,next_node))

    return f_distance, came_from
#Driver Code
Node_distance, Path=astar(graph,'A','F')
print("Node distance: ", Node_distance)
print(Path)
```

```
found the end_node
Node distance:  4
{'A': 'A', 'B': 'A', 'C': 'A', 'D': 'B', 'E': 'B', 'F': 'E'}
```

In [ ]: