In [2]:

```
class Node:

    def __init__(self, f=0, g=999999, h=0, name=0):
        self.f = f
        self.g = g
        self.h = h
        self.name = name

    def setNeighbours(self, neighbours={}):
        self.neighbours = neighbours



# assume a 5 node bidirectional graph as follows
graph = [
          [-1, 1, 4, -1, -1],
          [1, -1, 2, 5, 12],
          [4, 2, -1, 2, -1],
          [-1, 5, 2, -1, 3],
          [-1, 12, -1, 3, -1]#g(n)
        ]


# assume heuristics for each node
heuristics = [7, 6, 2, 1, 0]#H(n)

s = Node(h=heuristics[0], name=0)
a = Node(h=heuristics[1], name=1)
b = Node(h=heuristics[2], name=2)
c = Node(h=heuristics[3], name=3)
d = Node(h=heuristics[4], name=4)


s.setNeighbours([a,b])
a.setNeighbours([s, b, c ,d])
b.setNeighbours([s, a, c])
c.setNeighbours([a, b, d])
d.setNeighbours([a, c])


startNode = s
goalNode = d


def astar(start,goal):

    closedSet = set([])
    openSet = set([start])

    cameFrom = {}

    start.g = 0
    start.f = start.h


    while len(openSet)!=0:

        current = findNodeWithLowestFScore(openSet)
```

```
        if current==goal:
            return contruct_path(cameFrom, current)

        openSet.remove(current)
        closedSet.add(current)

        #print(current.name, current.f, current.g, current.h)

        for neighbour in current.neighbours:

            #print(neighbour.name, neighbour.f, neighbour.g, neighbour.h)

            if neighbour in closedSet:
                continue

            if neighbour not in openSet:
                openSet.add(neighbour)


            tentative_gScore=current.g+graph[current.name][neighbour.name]

#print(tentative_gScore)
            if tentative_gScore>= neighbour.g:
                continue

            cameFrom[neighbour] = current
            neighbour.g = tentative_gScore
            neighbour.f = neighbour.g + neighbour.h



    return -1

def findNodeWithLowestFScore(openSet):
    fScore = 999999
    node = None
    for eachNode in openSet:
        if eachNode.f<fScore:
            fScore = eachNode.f
            node = eachNode

    return node


def contruct_path(cameFrom, current):

    totalPath = []
    while current in cameFrom.keys():
        current = cameFrom[current]
        totalPath.append(current)

    return totalPath



if __name__=="__main__":


    path = astar(startNode, goalNode)

    print("Path is : ", end="" )
```

```
    for node in path[::-1]:
        print(str(node.name) + "-->", end="")
    print(goalNode.name)

    print("\nCost = " + str(goalNode.g))
```

```
Path is : 0-->2-->3-->4

Cost = 9
```

In [ ]: