A star

```
graph = {
   'A': [(2, 'B'), (3, 'E')],
   'B': [(1, 'C')],
   'C': [],
   'D': [(1, 'G')],
   'E': [(6, 'D')],
   'F': [],
   'G': []
}

def heuristic(n):
   H_dist = {
      'A': 11,
      'B': 6,
      'C': 99,
      'D': 1,
      'E': 7,
      'G': 0
   }
   return H_dist[n]

def get_neighbors(v):
   if v in graph:
      return graph[v]
   else:
      return None

def aStar(start_node, stop_node):
   open_set = {start_node}
   closed_set = set()
   g = {}
   parents = {}

   g[start_node] = 0
   parents[start_node] = start_node
```

```
while len(open_set) > 0:
    n = None
    for v in open_set:
        if n is None or g[v] + heuristic(v) < g[n] + heuristic(n):
            n = v

    if n == stop_node or not graph[n]:
        path = []
        while parents[n] != n:
            path.append(n)
            n = parents[n]
        path.append(start_node)
        path.reverse()
        print('Path found: {}'.format(path))
        print('Path cost is ',tentative_g_score)
        return path

    open_set.remove(n)
    closed_set.add(n)

    for edge in get_neighbors(n) or []:
        weight, neighbor = edge
        if neighbor in closed_set:
            continue

        tentative_g_score = g[n] + weight

        if neighbor not in open_set or tentative_g_score < g.get(neighbor, float('inf')):
            g[neighbor] = tentative_g_score
            parents[neighbor] = n
            open_set.add(neighbor)

    print('Visited nodes (closed set):', closed_set)
    print('Open list:', open_set)

print('No path found.')
return None
```

```
result = aStar('A', 'G')
```