PROGRAM :

```
graph = [
    ['S', 'A', 3, 10.5], ['S', 'C', 4, 9.2], ['A', 'B', 4, 6], ['A', 'C', 5, 9.2],
    ['C', 'A', 5, 10.5], ['C', 'D', 8, 6.2], ['B', 'F', 4, 9999], ['B', 'G', 2, 0],
    ['B', 'D', 4, 6.2], ['D', 'B', 4, 6], ['D', 'E', 2, 4.5], ['E', 'G', 10, 0]
]
start = input("Enter the start node : ")
goal = input("Enter the goal node  : ")
# create a list of all the nodes
temp1 = set()
temp2 = set()
for i in graph:
    temp1.add(i[0])
    temp2.add(i[1])
nodes = temp1.union(temp2)
# create a cost dictionary and initialize the cost to 0 for all the nodes
cost = dict()
for node in nodes:
    cost[node] = 0
# for storing the path traversed
path = []
def hill_climb_search(graph, path: set, cur_node):
    if cur_node not in path:
        path.append(cur_node)
    # if the goal node is found stop.
    if cur_node == goal:
        return
    # find the neighbors
    neighbors = {}
    for node in graph:
        if node[0] == cur_node:
            # cost(neighbor) = cost(parent) + cost(parent -> neighbor)
            cost[node[1]] = cost[node[0]] + node[2]
            neighbors[node[1]] = cost[node[0]] + node[2] + node[3] # calculate f value for neighbor
    # if a node has no neighbors, we have reached a deadend, so stop
    if len(neighbors) == 0:
        return
    # choose the best to expand
    best_neighbor = min(neighbors, key=neighbors.get)
    hill_climb_search(graph, path, best_neighbor)
hill_climb_search(graph, path, start)
print(f"The path to the goal node is: ", "->".join(path))
print("cost : ", cost[goal])
```

OUTPUT :

```
Enter the start node : S
Enter the goal node  : G
The path to the goal node is:  S->C->D->E->G
cost :  24
```