In [2]:

```
import heapq
from typing import List

class Node:
    def __init__(self, vertex: int, fScore: int):
        self.vertex = vertex
        self.fScore = fScore

    def __lt__(self, other):
        return self.fScore < other.fScore

class Edge:
    def __init__(self, dest: int, weight: int):
        self.dest = dest
        self.weight = weight

class Graph:
    def __init__(self, numVertices: int):
        self.numVertices = numVertices
        self.adjList = [[] for _ in range(numVertices)]

    def addEdge(self, source: int, destination: int, weight: int):
        self.adjList[source].append(Edge(destination, weight))

    def aStarAlgorithm(self, start: int, goal: int, heuristic: List[int]) -> List[int]:
        g = [float('inf')] * self.numVertices
        f = [float('inf')] * self.numVertices
        parent = [-1] * self.numVertices

        g[start] = 0
        f[start] = heuristic[start]

        openList = []
        heapq.heappush(openList, Node(start, f[start]))

        while openList:
            current = heapq.heappop(openList)
            currentVertex = current.vertex

            if currentVertex == goal:
                path = []
                vertex = goal
                while vertex != -1:
                    path.append(vertex)
                    vertex = parent[vertex]
                path.reverse()

                pathCost = g[goal]
                print("Path cost:", pathCost)
                return path

            for neighbor in self.adjList[currentVertex]:
                neighborVertex = neighbor.dest
                tentativeG = g[currentVertex] + neighbor.weight

                if tentativeG < g[neighborVertex]:
                    parent[neighborVertex] = currentVertex
                    g[neighborVertex] = tentativeG
                    f[neighborVertex] = g[neighborVertex] + heuristic[neighborVertex]
                    heapq.heappush(openList, Node(neighborVertex, f[neighborVertex]))

        return []

# Example usage
n = int(input("Enter the size of the graph: "))
graph = Graph(n)

size = int(input("Enter the size of input: "))
print("Enter edges of the graph:")
for i in range(size):
    print(f"Enter the value of {i+1} edge and its weight: ", end="")
    j, k, w = map(int, input().split())
    if j < n and k < n:
        graph.addEdge(j, k, w)
    else:
        print("Invalid Input")

heuristic = []
print("Enter the heuristic values for the vertices of the graph:")
for i in range(n):
    print(f"Enter {i} vertex's heuristic value: ", end="")
    h = int(input())
    heuristic.append(h)

startVertex = int(input("Enter the starting vertex of the graph: "))
goalVertex = int(input("Enter the ending vertex of the graph: "))

path = graph.aStarAlgorithm(startVertex, goalVertex, heuristic)
```

```
if path:
    print("Optimal path found:", end="")
    for vertex in path:
        print(" ", vertex, end="")
    print()
else:
    print("Path not found!")
```

```
Enter the size of the graph: 5
Enter the size of input: 7
Enter edges of the graph:
Enter the value of 1 edge and its weight: 0 1 4
Enter the value of 2 edge and its weight: 0 2 2
Enter the value of 3 edge and its weight: 1 3 2
Enter the value of 4 edge and its weight: 1 4 2
Enter the value of 5 edge and its weight: 2 2 3
Enter the value of 6 edge and its weight: 2 3 2
Enter the value of 7 edge and its weight: 3 4 2
Enter the heuristic values for the vertices of the graph:
Enter 0 vertex's heuristic value: 7
Enter 1 vertex's heuristic value: 6
Enter 2 vertex's heuristic value: 5
Enter 3 vertex's heuristic value: 3
Enter 4 vertex's heuristic value: 2
Enter the starting vertex of the graph: 0
Enter the ending vertex of the graph: 4
Path cost: 6
Optimal path found:  0  2  3  4
```

In [ ]: