```
# Python3 implementation of above approach

# returns the minimum cost in a vector( if
# there are multiple goal states)
def uniform_cost_search(goal, start):

 # minimum cost upto
 # goal state from starting
 global graph,cost
 answer = []

 # create a priority queue
 queue = []

 # set the answer vector to max value
 for i in range(len(goal)):
  answer.append(10**8)

 # insert the starting index
 queue.append([0, start])

 # map to store visited node
 visited = {}

 # count
 count = 0

 # while the queue is not empty
 while (len(queue) > 0):

  # get the top element of the
  queue = sorted(queue)
  p = queue[-1]

  # pop the element
  del queue[-1]

  # get the original value
  p[0] *= -1

  # check if the element is part of
  # the goal list
  if (p[1] in goal):

   # get the position
   index = goal.index(p[1])

   # if a new goal is reached
   if (answer[index] == 10**8):
    count += 1

   # if the cost is less
   if (answer[index] > p[0]):
    answer[index] = p[0]
```

```
  # pop the element
  del queue[-1]

  queue = sorted(queue)
  if (count == len(goal)):
   return answer

 # check for the non visited nodes
 # which are adjacent to present node
 if (p[1] not in visited):
  for i in range(len(graph[p[1]])):

   # value is multiplied by -1 so that
   # least priority is at the top
   queue.append( [(p[0] + cost[(p[1], graph[p[1]][i])])* -1, graph[p[1]][i]])

 # mark as visited
 visited[p[1]] = 1

 return answer

# main function
if __name__ == '__main__':

 # create the graph
 graph,cost = [[] for i in range(8)],{}

 # add edge
 graph[0].append(1)
 graph[0].append(3)
 graph[3].append(1)
 graph[3].append(6)
 graph[3].append(4)
 graph[1].append(6)
 graph[4].append(2)
 graph[4].append(5)
 graph[2].append(1)
 graph[5].append(2)
 graph[5].append(6)
 graph[6].append(4)

 # add the cost
 cost[(0, 1)] = 2
 cost[(0, 3)] = 5
 cost[(1, 6)] = 1
 cost[(3, 1)] = 5
 cost[(3, 6)] = 6
 cost[(3, 4)] = 2
 cost[(2, 1)] = 4
 cost[(4, 2)] = 4
 cost[(4, 5)] = 3
 cost[(5, 2)] = 6
 cost[(5, 6)] = 3
 cost[(6, 4)] = 7
```

```
# goal state
goal = []

# set the goal
# there can be multiple goal states
goal.append(6)

# get the answer
answer = uniform_cost_search(goal, 0)

# print the answer
print("Minimum cost from 0 to 6 is = ",answer[0])

# This code is contributed by mohit kumar 29
```