192. There are n cities numbered from 0 to n-1. Given the array edges where edges[i] = [fromi, toi, weighti] represents a bidirectional and weighted edge between cities fromi and toi, and given the integer distanceThreshold. Return the city with the smallest number of cities that are reachable through some path and whose distance is at most distanceThreshold, If there are multiple such cities, return the city with the greatest number. Notice that the distance of a path connecting cities i and j is equal to the sum of the edges' weights along that path.

Example 1:

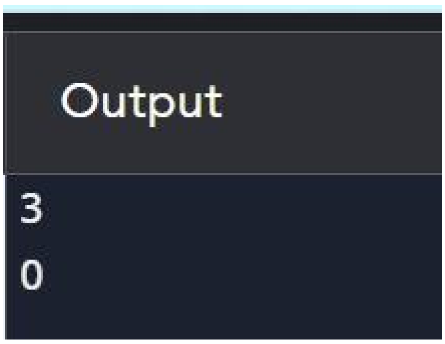
Input: n = 4, edges = [[0,1,3],[1,2,1],[1,3,4],[2,3,1]], distanceThreshold = 4

Program:

```
import heapq


def findTheCity(n, edges, distanceThreshold):
    graph = {i: {} for i in range(n)}
    for u, v, w in edges:
        graph[u][v] = w
        graph[v][u] = w

    def dijkstra(src):
        pq = [(0, src)]
        dist = {i: float('inf') for i in range(n)}
        dist[src] = 0

        while pq:
            d, node = heapq.heappop(pq)
            if d > dist[node]:
                continue
            for neighbor, weight in graph[node].items():
                if (new_dist := d + weight) < dist[neighbor]:
                    dist[neighbor] = new_dist
                    heapq.heappush(pq, (new_dist, neighbor))

        return sum(1 for d in dist.values() if d <= distanceThreshold)
```

```
    min_reachable = n
    res = -1

    for i in range(n):
        reachable = dijkstra(i)
        if reachable <= min_reachable:
            min_reachable = reachable
            res = i

    return res

# Example 1
n = 4
edges = [[0,1,3],[1,2,1],[1,3,4],[2,3,1]]
distanceThreshold = 4
print(findTheCity(n, edges, distanceThreshold))  # Output: 3

# Example 2
n = 5
edges = [[0,1,2],[0,4,8],[1,2,3],[1,4,2],[2,3,1],[3,4,1]]
distanceThreshold = 2
print(findTheCity(n, edges, distanceThreshold))  # Output: 0
```

Output:



Timecomplexity:O(n^3)