# 公交路线

给你一个数组 routes ，表示一系列公交线路，其中每个 routes[i] 表示一条公交线路，第 i 辆公交车将会在上面循环行驶。

- 例如，路线 routes[0] = [1, 5, 7] 表示第 0 辆公交车会一直按序列 1 -> 5 -> 7 -> 1 -> 5 -> 7 -> 1 -> ... 这样的车站路线行驶。

现在从 source 车站出发（初始时不在公交车上），要前往 target 车站。 期间仅可乘坐公交车。

求出 **最少乘坐的公交车数量** 。如果不可能到达终点车站，返回 -1 。

**示例 1：**

**输入：** routes = [[1,2,7],[3,6,7]], source = 1, target = 6

**输出：** 2

**解释：** 最优策略是先乘坐第一辆公交车到达车站 7 , 然后换乘第二辆公交车到车站 6 。

**示例 2：**

**输入：** routes = [[7,12],[4,5,15],[6],[15,19],[9,12,13]], source = 15, target = 12

**输出：** -1

```
class Solution {
public:
    int numBusesToDestination(vector<vector<int>>& routes, int S, int T) {
        if (S == T) return 0;//起点与终点
        int N = routes.size();
        map<int, set<int> > m; // 存储车站能通到哪些路线路线
        for (int i = 0; i < N; ++i) {
            for (auto j : routes[i]) {
                m[j].insert(i);//routes[0]表示第 0 辆公交车,第 0 辆公交车作为其中一条路线存储在 map
            }
        }
        vector<bool> visited(N, false); // 哪些路线被遍历过了
        queue<int> q; // 存储已经遍历过的路线
        for (auto x : m[S]) {
            q.push(x);
            visited[x] = true;//遍历过的路线入队，并标志已经遍历过该路线
        }
```

```
        int step = 0;
        while (!q.empty()) {
            ++step;
            int s = q.size();
            for (int i = 0; i < s; ++i) {
                int t = q.front();
                q.pop();//先进先出原则  获取队头元素后出队，即获取所遍历过的路线
                for (auto j : routes[t]) {
                    if (j == T) return step;//随着 step++计数，当前队头序列如果出现目标 target，则返回 step
                    for (auto x : m[j]) {
                        if (!visited[x]) {
                            q.push(x);//将没有访问过的将其入队
                            visited[x] = true;
                        }
                    }
                }
            }
        }
        return -1;
    }
};
```