1

使用tarjan缩点后，如果只存在一个出度为0的联通块，则该联通块所有点是能从任何点出发都能到达的，如果只存在一个入度为0的联通块，则该联通块所有点能抵达其它任何点。

2

为了快速的求出来次小生成树，我们需要知道一个结论：次小生成树和最小生成树之间只有一条边的差异。

这一条结论为什么是正确的呢？

我们先来看看 $Kruskal$ 是怎么求最小生成树的：

$Kruskal$ 是按照边权排序，按照贪心的思路一条一条的加进树边，所以如果要求出次小生成树，那么我们可以放弃一条小的边不选，加入另一条边使图联通。

如果我们"放弃"了两条边，那么只"放弃"一条边的生成树的边权和显然小于等于"放弃"两条边的生成树的边权和。

所以求出（非）严格次小生成树的朴素算法为：先建立一棵最小生成树。再枚举删去最小生成树上的每一条路径，再在新构成的生成树中选出一棵边权之和最小生成树的即可。

时间复杂度为 $O(nm\log m)$。当然这种算法还不够优秀，我们可以继续优化。

非严格次小生成树我们可以这样优化：

枚举边的时候，枚举没有被并入到最小生成树的边（我们将把这条边加入到生成树中，显然现在的图形不再是一棵树，所以我们要原图中删去一条最大边，使新图仍然联通）。

加入边权值为 W_1。查询树上每一条的路径，在路径选取一个边权最大值 W_2。则当前枚举的答案为
$W-W_2+W_1$
（W 为最小生成树的边权之和）。

枚举所有的边之后，取最小值即可。

我们可以用倍增/树割/LCT来实现查询树上最大值的操作，故复杂度为：$O(m\log n)$

以下给出倍增的预处理和求解的代码

```
void dfs(int u, int parent, int d, int edge_weight) {
    depth[u] = d;
    fa[u][0] = parent;
    max_edge[u][0] = edge_weight;

    // 预处理2^j级祖先
    for (int j = 1; j < MAXLOG; j++) {
        fa[u][j] = fa[fa[u][j-1]][j-1];
        max_edge[u][j] = max(max_edge[u][j-1], max_edge[fa[u][j-1]][j-1]);
    }

    // 遍历所有子节点
    for (auto& [v, w] : graph[u]) {
        if (v != parent) {
            dfs(v, u, d + 1, w);
        }
    }
```

```
}
int find(int u, int v) {
    if (depth[u] < depth[v]) swap(u, v);

    int result = 0;

    // 将u调整到与v相同的深度
    for (int j = MAXLOG - 1; j >= 0; j--) {
        if (depth[u] - (1 << j) >= depth[v]) {
            result = max(result, max_edge[u][j]);
            u = fa[u][j];
        }
    }

    if (u == v) return result;

    // 同时向上跳，找到LCA
    for (int j = MAXLOG - 1; j >= 0; j--) {
        if (fa[u][j] != fa[v][j]) {
            result = max(result, max(max_edge[u][j], max_edge[v][j]));
            u = fa[u][j];
            v = fa[v][j];
        }
    }

    // 最后一步到LCA
    result = max(result, max(max_edge[u][0], max_edge[v][0]));

    return result;
}
```

再来看看严格次小生成树：

不难发现：求非严格最小生成树时，枚举一条边 W_1，之后再寻找一条生成树上的最大边 W_2。

显然 $W_1 \geq W_2$，因此可能由此得到的次小生成树并非严格。所以我们可以在每次查询时，需要找到严格次小的 W_1，即 $W_1 > W_2$，这样我们就可以得到严格次小生成树了。

维护仍可以用倍增或者树剖思想。

如果是倍增，用倍增数组维护祖先最大值与严格次大值，合并的时候把两个区间的这四个值（两个最大值与两个次大值）排序，再寻找最大值与严格次大值即可。

3

$\prod_i C_i > 1$等价于$-\sum_i \log C_i < 0$，故可以对边权取对数后用bellman-ford求负环。

4

Claim: 若 $1 \to u$ 的最短路径过 v，那么一定是 $1 \to v$ 的最短路径 $+v \to u$ 的最短路径。

考虑包含节点 1 的最小环 $1 \to \cdots \to x \to \cdots \to 1$（用有向箭头表示，以方便区分环上 1 到 x 的两条路径）。

Claim: 环上 $1 \to x$ 和 $x \to 1$ 的路径，至少有一条是 1 到 $\cancel{u}$ x 的最短路径。

若 $1 \to x$ 是最短路径，结论自然成立。

否则，1 到 x 的最短路径一定不包含 $1 \to x$ 经过的所有点。

Proof: 假设 1 到 x 的最短路径过了环上路径 $1 \to x$ 的某些点，那么可以在所有经过的点中找到点 y，使得 1 到 x 的最短路径没有经过环上 $1 \to y$ 中的所有点。则 1 到 y 的最短路径也不会经过环上路径 $1 \to y$ 中的任意一个点，不然可以找到一个更靠前的 y。但显然，此时环上路径 $1 \to y$ 和 1 到 y 的最短路径构成了一个更小的环，与前提矛盾。

那么，1 到 x 的最短路径一定是环上路径 $x \to 1$，否则同样与最小环的要求矛盾。

所以最小环上存在一个不同于 1 的点 x，环上路径 $1 \to x$ 上的点 u 都以环上路径 $1 \to u$ 为最短路径，环上路径 $x \to 1$ 上的点 v（除 x 外）都以环上路径 $v \to 1$ 为最短路径。

所以环上一定存在一条边 (x, y)，使得最小环由 $dis(1, x) + (x, y) + dis(y, 1)$ 构成。

先跑以 1 为源点的单源最短路，以所有最短路径构建以 1 为根的树，遍历每一条边 (x, y)，只考虑 x, y 在根节点的不同子树中的情形，选取 $\min_{(x,y)} dis(1, x) + (x, y) + dis(y, 1)$ 即为答案。

考虑满足上述形式的最小环 $1 \to \cdots \to x \to y \to \cdots \to 1$。

由于可能存在多条最短路径，所以在第一步最短路径可能会使得 x, y 出现在同一个子树中。

此时考虑 x, y 的 LCA z，到 z 的最短路径一定只有环上路径 $1 \to z$，否则存在一个更小环。由于已知环上路径 $1 \to x$ 和 $1 \to y$ 不重合，所以 x, y 中至少有一个点存在一条不经过环上路径 $1 \to z$ 上所有点的最短路径 l，且它不经过树上路径 $1 \to x$ 和 $1 \to y$ 中的任意点。不妨设该路径为 $1 \to y$，则 $1 - z - y - l$ 的权值一定小于等于 $1 - z - x - y - l$（因为 $1 - z - y$ 是最短路），故只需考虑 $1 - z - y - l$，而路径 l 与 x, y 不在同一子树。故只考虑不同子树的情况答案不会变劣。