字符串

简单对四种情况一一判定。

饮料

答案其实就是 $n/(k-1)$，注意到 n 很大，k 很小，模拟除法的竖式运算，最后去除前导零。

染色 (color)

- 50%: 直接模拟即可，复杂度$O(n*q)$
- 100%:
  考虑到正向模拟不好操作,每次刷完之后可能被再次更新，然而我们只关心最后的结果，那么反向处理即可
  即:把所有操作按时间逆序处理。
  那么逆序后，先处理的操作不会被后处理的操作影响，即当前修改的部分不会再次被修改，因此在每次操作完后，删掉当前行/列即可，同时修改剩余行数，列数，并记录答案。

城市道路 (countpath.cpp)

20分题解

由于城市数很少，而数字很少，那么道路数量也就不会很多，所以可以直接用 DFS 来进行搜索。

50分题解

动态规划。设 $dp(i)$ 表示当前从城市 1 到达城市 i 的路径数量，那么最后的答案也就是 $dp(N)$。转移方程就从前面某一个城市 j 直接到达城市 i，走法就是 $j \to i$ 的边数，转移方程如下：

$$dp(i)=\sum_{j=1}^{i-1} dp(j)\times F(A_i \& A_j)$$

初值是 $dp(1)=1$。时间复杂度为 $O(n^2 \log A_i)$。

100分题解

上述转移方程可以优化，主要集中于后面的 $F(A_i \& A_j)$，将其拆开为 $\sum_{k=0}^{30}[(A_i \& A_j)_k == 1]$。其中 $[\cdot]$ 是 Iverson 括号，里面是一个表达式，当表达式为 True 时值为 1，否则为 0。这个和式等同于下面的代码：

```
int ret = 0;
for (int k = 0; k <= 30; k++)
    if (((a[i] & a[j]) >> k) & 1)
        ret++;
return ret;
```

而条件 $(A_i \& A_j)_k == 1$ 又可以拆成 $(A_i)_k == 1 \text{ AND } (A_j)_k == 1$，也就是 A_i 与 A_j 的第 k 位都必须为 1。

那么转移方程就可以改写为：

$$
\begin{aligned}
dp(i) &= \sum_{j=1}^{i-1} dp(j) \sum_{k=0}^{30} [(A_i \& A_j)_k == 1] \\
&= \sum_{j=1}^{i-1} dp(j) \sum_{k=0}^{30} [(A_i)_k == 1 \text{ AND } (A_j)_k == 1] \\
&= \sum_{j=1}^{i-1} dp(j) \sum_{k=0}^{30} [(A_i)_k == 1] \cdot [(A_j)_k == 1]
\end{aligned}
$$

两个和式枚举的变量互不影响，所以可以交换两个和式的位置：

$$
dp(i) = \sum_{k=0}^{30} [(A_i)_k == 1] \sum_{j=1}^{i-1} dp(j) \cdot [(A_j)_k == 1]
$$

观察右边的 $\sum\limits_{j=1}^{i-1} dp(j) \cdot [(A_j)_k == 1]$，实际上就是一个前缀和。这个前缀和针对每一个二进制位来维护，维护的是前面 $i-1$ 个城市中 A_j 的第 k 位为 1 的所有 $dp(j)$ 之和。这个维护可以一边转移一边维护。

其实，上述做法的描述比较数学。形象一点的话，实际上就是考虑每一个二进制位对 $dp(i)$ 的 “贡献”。针对这种二进制的题目，一个比较常见的思路就是拆位，将每一个二进制位单独考虑。比如在本题中，每一个二进制位对应的就是一张没有重复边的图，那么转移也就比较简单，之后再向多个二进制位来考虑。本题中，不同二进制位之间其实是没有影响的，也就可以各个二进制位单独计算，然后再加起来即可。

时间复杂度：$O(n \log A_i) = O(30n)$。