2021 蓝桥杯热身赛 #2 题解

----by lr580

> 以下所有题解仅提供一种或多种正确解法。并不必然代表下面提供的解法是最优解，且并不必然代表其他的解法不可行。因此，如果有别的思路，也欢迎各位大佬在 SCNUOJ 讨论区分享你的解法。若题解有误，欢迎指正~ ヾ(o·ω·)/

异世界的历法

根据题意，有：1 异世界小时 $=\dfrac{55}{60}$ 原世界小时，即：1 原世界小时 $=\dfrac{60}{55}$ 异世界小时

注意结果是浮点数，但是也可以向下取整(因为多出的小数部分不会对年月日时产生贡献)

解法一：按题意逐天模拟或逐小时模拟等均可。逐天模拟时间复杂度为 $\mathrm{O}(1080\times 24\times 60\div 55\div 40)$ 。

解法二：直接计算结论。注意到异世界的年月日进制为 10 ，直接用十进制表示即可，0 为序数第一， 1 为第二，以此类推。一小时为 $\dfrac{1}{40}$ 天，起始时间可以表示为 $58026+\dfrac{5}{40}$，计算后最后舍掉小数部分，输出时序数从 1 开始所以再 $+1$ 。时间复杂度为 $\mathrm{O}(1)$ 。

C++ 参考程序(解法一)

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
ll y = 580, m = 3, d = 7;
signed main()
{
    for (double h = 1080 * 24 * 60.0 / 55 + 15; h > 40;) //ll也行
    {
        h -= 40;
        ++d;
        if (d > 10)
        {
            d = 1; //注意从1开始
            ++m;
        }
        if (m > 10)
        {
            m = 1;
            ++y;
        }
    }
    printf("%lld %lld %lld\n", y, m, d);
    return 0;
}
```

Python 参考程序(解法一)

```python
y = 580
m = 3
d = 7
h = 1080*24*60/55+15
while h > 0:
    if h < 40:
        break
    h -= 40
    d += 1
    if d > 10:
        d = 1
        m += 1
    if m > 10:
        m = 1
        y += 1
print(y, m, d)
```

C++ 参考程序(解法二)

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
ll x = 58026 + (15.0 / 40 + 1080 * 24 * 60.0 / 55 / 40);
signed main()
{
    printf("%lld %lld %lld", x / 100, 1 + x / 10 % 10, 1 + x % 10);
    return 0;
}
```

Python 参考程序(解法二)

```python
x=int(58026+15/40+1080*24*60/55/40)
print(x//100, x//10 % 10+1, x % 10+1)
```

异世界的文章

解法一：一种暴力解法是可以直接枚举所有三字词，判断首尾是否依次是 `a`，`k` 即可，最后输出计数答案。时间复杂度 $\mathrm{O}(n^3)$。

解法二：可以枚举字符串的两端，如果子串 $[i,j]$ 两端点分别是 `a`，`k`，那么 $\forall x \in (i,j)$，由下标 i,x,j 组成的都是符合条件的，加上区间长度 $i-j-1$ 即可。时间复杂度 $\mathrm{O}(n^2)$。

解法三：一次遍历，记录当前所有出现的 `a` 的下标之和 c 和 `a` 的数目，每次遍历到 `k`，设当前 `k` 的下标为 j，有 m 个 `a`，下标分别为 p_i，即求：$\sum_{i=1}^{m} j-p_i-1=mj-c-m$。时间复杂度 $\mathrm{O}(n)$。

> 附：题目所给文章原文见 这里 (密码：`mermaid`)

C++ 参考程序(解法一)

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
char s[] = ""; //把题目字符串复制到这里
ll n, ans;
signed main()
{
    n = strlen(s);
    for (ll i = 0; i < n - 2; ++i)
    {
        if (s[i] != 'a')
        {
            continue;
        }
        for (ll j = i + 1; j < n - 1; ++j)
        {
            for (ll k = j + 1; k < n; ++k)
            {
                if (s[k] != 'k')
                {
                    continue;
                }
                ans++;
            }
        }
    }
    printf("%lld", ans);
    return 0;
}
```

Python 参考程序(解法一)

```python
s = '' # 把题目字符串复制到这里；本程序可能需要跑十几秒
n = len(s)
ans = 0
for i in range(n-2):
    if s[i] != 'a':
        continue
    for j in range(i+1, n-1):
        for k in range(j+1, n):
            if s[k] != 'k':
                continue
            ans += 1
print(ans)
```

C++ 参考程序(解法二)

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
char s[] = ""; //把题目字符串复制到这里
ll n, ans;
signed main()
{
```

```
    n = strlen(s);
    for (ll i = 0; i < n - 2; ++i)
    {
        if (s[i] != 'a')
        {
            continue;
        }
        for (ll j = i + 2; j < n; ++j)
        {
            if (s[j] != 'k')
            {
                continue;
            }
            ans += j - i - 1;
        }
    }
    printf("%lld", ans);
    return 0;
}
```

Python 参考程序(解法二)

```
s = '' # 把题目字符串复制到这里
n = len(s)
ans = 0
for i in range(n-2):
    if s[i] != 'a':
        continue
    for j in range(i+2, n):
        if s[j] != 'k':
            continue
        ans += j-i-1
print(ans)
```

C++ 参考程序(解法三)

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
char s[] = ""; //把题目字符串复制到这里
ll n, ans, c, m;
signed main()
{
    n = strlen(s);
    for (ll i = 0; i < n; ++i)
    {
        if (s[i] == 'a')
        {
            ++m;
            c += i;
        }
        else if (s[i] == 'k')
        {
            ans += m * i - c - m;
        }
    }
    printf("%lld", ans);
```

```
    return 0;
}
```

Python 参考程序(解法三)

```python
s = '' # 把题目字符串复制到这里
n = len(s)
ans = m = c = 0
for i in range(n):
    if s[i] == 'a':
        c += i
        m += 1
    elif s[i] == 'k':
        ans += m*i-c-m
print(ans)
```

异世界的融资

> 解法一前置知识：DFS；解法二前置知识：状态压缩、最短(长)路

解法一：本题直接暴力 **DFS** 即可，设整型 `unabled` 数组， `unabled[i]` 代表当前的方案里，第 `i` 个王室的禁选数，被选中(不能重复选)或被当前选中的王室敌对都会增加禁选数。每次向下搜索时，对 `unabled` 里当前所选王室的敌对王室自增，同时自己自增(不能重复选自己)，方案金钱累积自增，搜索完毕后进行回溯，减少当前王室的敌对王室计数和金钱累积。搜索结束的标志是所有王室都禁选了(没有进一步的选择了)。带回溯可以不传入 DFS 参数。也可以尝试使用数位压缩+不回溯的方法，则 DFS 需要参数。

注意本题卡 `int` ，需要使用 `long long` 才能过题。搜索的最差复杂度是 $\mathrm{O}(n!)$ ，但实际跑时由于剪枝和题目本身敌对王室较多，跑 2 秒不到就可以出结果了。

解法二：本题可以尝试以**状态压缩建图**跑**最长路**来解，王室数目为 n 点数为 $n'=2^n$ ，设第 i 个点的第 i 个数位为当前是否禁用该王室，起点是 0 ，终点是 2^n-1 ，边权是资金，跑单源最长路可得答案。使用堆优化的 Dijkstra 算法求最长路，最差时间复杂度为 $\mathrm{O}(2^n\log 2^n)=\mathrm{O}(n2^n)$ 。空间复杂度是 $\mathrm{O}(2^n)$ 。然而事实上本题的图是稀疏图，所以一瞬间就得出答案了。

输入里每行有不定个整数，对 C/C++ 选手，输入处理可以先读入整行，再用 `sscanf` 或 `stringstream` 处理。

> 从本题开始，不再提供 Python 题解，仅提供 C++ 题解

C++ 参考程序(解法一)

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define mn 30
struct royal
{
    ll money;
    vector<ll> enemies;
} a[mn];
ll unabled[mn];
ll now, ans, n, v;
void dfs()
{
```

```
    ll unables = 0;
    for (ll i = 1; i <= n; ++i) //有多少个王室当前禁选
    {
        unables += unabled[i] != 0;
    }
    if (unables == n) //都禁选了，搜索结束
    {
        ans = max(ans, now);
        return;
    }
    for (ll i = 1; i <= n; ++i)
    {
        if (0 == unabled[i])
        {
            ++unabled[i];
            for (auto j : a[i].enemies)
            {
                ++unabled[j];
            }
            now += a[i].money;
            dfs();
            --unabled[i]; //回溯
            for (auto j : a[i].enemies)
            {
                --unabled[j];
            }
            now -= a[i].money;
        }
    }
}
signed main()
{
    freopen("C.txt", "r", stdin); //将题目的表复制到同目录下该文件里
    for (n = 1; n <= 25; ++n)
    {
        string s;
        getline(cin, s);
        stringstream ss(s);
        ss >> v >> v; //读走编号，然后再读资金数
        a[n].money = v;
        while (ss >> v) //读取一行内不定个整数
        {
            a[n].enemies.push_back(v);
        }
    }
    dfs();
    printf("%lld", ans);
    return 0;
}
```

C++ 参考程序(解法二)

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define mn 25
struct royal
```

```
{
    ll money;
    ll enemies; //数位压缩，数位为0是未禁用，否则是禁用
} a[mn + 3];
ll ans, v, n;
struct node
{
    ll d, i;
    bool operator<(const node &x) const { return d > x.d; }
};
priority_queue<node> q;
ll d[3 + (1 << mn)];
bool vis[3 + (1 << mn)];
void dijkstra()
{
    d[0] = 0;
    q.push({0, 0});
    while (!q.empty())
    {
        node t = q.top();
        q.pop();
        ll u = t.i;
        if (vis[u])
        {
            continue;
        }
        vis[u] = true;
        for (ll i = 0; i < mn; ++i) //枚举所有王室
        {
            if (u & (1 << i)) //禁选了第i个王室
            {
                continue;
            }
            ll v = u | a[i].enemies | (1 << i); //禁选自己和敌对王室
            if (d[v] < d[u] + a[i].money)       //最长路
            {
                d[v] = d[u] + a[i].money;
                if (!vis[v])
                {
                    q.push({d[v], v});
                }
            }
        }
    }
}
signed main()
{
    freopen("C.txt", "r", stdin); //将题目的表复制到同目录下该文件里
    for (n = 0; n < mn; ++n)      //输入与解法一有所不同
    {
        string s;
        getline(cin, s);
        stringstream ss(s);
        ss >> v >> v; //读走编号，然后再读资金数
        a[n].money = v;
        while (ss >> v) //读取一行内不定个整数
        {
            a[n].enemies |= (1 << (v - 1));
```

```
        }
    }
    dijkstra();
    printf("%lld", d[(1 << mn) - 1]);
    return 0;
}
```

异世界的地图

> 前置知识：最短路

求因数可倍数可以暴力处理，单次复杂度均为 $O(n)$，总复杂度为 $O(n^2)$，也可以稍作优化，单次复杂度依次为 $O(\sqrt{n}), O(1)$ (等差数列优化)，总复杂度为 $O(n\sqrt{n}+n)=O(n\sqrt{n})$。

注意到果冻当天可以使用最多一次传送，意味着当天可以不使用传送。那么如果在当前点当天不能够得到最小消耗，可以原地等待一天，就可以得到当前点当天的最小消耗。这就意味着，可以选择晴雨消耗的最小值为边权直接建图即可，而不需要建双层的分层图。根据输入的 `Y/N` 选择将某条边建为有向还是无向，无向边等于有向边及其反边。

本题使用**单源最短路**算法均可过题，常见的如 Dijkstra，Bellman-Ford 和~~已死算法~~ SPFA。

在堆优化下，下面分别给出的邻接矩阵(写法一)或邻接表(写法二)或链式前向星(写法三)建图的 Dijkstra 最短路算法。设点数为 n，边数为 m。时间复杂度为 $O(n\log n)$。三种写法的空间复杂度依次是 $O(n^2), O(m), O(m)$。但实际链式前向星效率最高。

C++ 参考程序(邻接矩阵)

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define n 1437
#define m 4096
#define big 0x3ffffffa
ll d[n + 3], e[n + 3][n + 3];
struct node
{
    ll v, i;
    bool operator<(const node &x) const { return v > x.v; }
};
bool vis[n + 3];
void dijkstra(ll s)
{
    for (ll i = 1; i <= n; ++i)
    {
        d[i] = big;
    }
    d[s] = 0;
    priority_queue<node> q;
    q.push({0, s});
    while (!q.empty())
    {

        node t = q.top();
        ll u = t.i;
        q.pop();
        if (vis[u])
```

```
        {
            continue;
        }
        vis[u] = true;
        for (ll v = 1; v <= n; ++v)
        {
            if (d[v] > d[u] + e[u][v])
            {
                d[v] = d[u] + e[u][v];
                if (!vis[v])
                {
                    q.push({d[v], v});
                }
            }
        }
    }
}
signed main()
{
    for (ll i = 1; i <= n; ++i)
    {
        for (ll j = 1; j <= n; ++j)
        {
            e[i][j] = big; //无穷代表不存在(后序与新边取最小值去重边)
        }
    }
    for (ll i = 1, u = 0, v = 0, w1 = 0, w2 = 0, w; i <= m; ++i)
    {
        u = (1399 * i * i + u * u) % n + 1;
        v = (1427 * i * i + v * v) % n + 1;
        if (u == v)
        {
            u = u % n + 1;
        }
        char db = i % 2 ? 'Y' : 'N';
        for (ll j = 1; j <= i; ++j) //直接暴力判断即可
        {
            if (i % j == 0)
            {
                w1 += j;
            }
        }
        for (ll j = i; j <= m; ++j)
        {
            if (j % i == 0)
            {
                w2 += j;
            }
        }
        w1 = w1 % m + 1;
        w2 = w2 % m + 1;
        w = min(w1, w2);
        e[u][v] = min(e[u][v], w); //重边取最小
        if (db == 'Y')
        {
            e[v][u] = min(e[v][u], w);
        }
    }
```

```
    dijkstra(1);
    printf("%lld", d[n]);
    return 0;
}
```

C++ 参考程序(邻接表)

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define n 1437
#define m 4096
#define big 0x7ffffffa
ll d[n + 3];
vector<pair<ll, ll>> e[n + 3];
struct node
{
    ll v, i;
    bool operator<(const node &x) const { return v > x.v; }
};
bool vis[n + 3];
void dijkstra(ll s)
{
    for (ll i = 1; i <= n; ++i)
    {
        d[i] = big;
    }
    d[s] = 0;
    priority_queue<node> q;
    q.push({0, s});
    while (!q.empty())
    {

        node t = q.top();
        ll u = t.i;
        q.pop();
        if (vis[u])
        {
            continue;
        }
        vis[u] = true;
        for (auto i : e[u])
        {
            ll v = i.first;
            if (d[v] > d[u] + i.second)
            {
                d[v] = d[u] + i.second;
                if (!vis[v])
                {
                    q.push({d[v], v});
                }
            }
        }
    }
}
signed main()
{
```

```
    for (ll i = 1, u = 0, v = 0, w1 = 0, w2 = 0, w; i <= m; ++i)
    {
        u = (1399 * i * i + u * u) % n + 1;
        v = (1427 * i * i + v * v) % n + 1;
        if (u == v)
        {
            u = u % n + 1;
        }
        char db = i % 2 ? 'Y' : 'N';
        for (ll j = 1; j <= i; ++j) //直接暴力判断即可
        {
            if (i % j == 0)
            {
                w1 += j;
            }
        }
        for (ll j = i; j <= m; ++j)
        {
            if (j % i == 0)
            {
                w2 += j;
            }
        }
        w1 = w1 % m + 1;
        w2 = w2 % m + 1;
        w = min(w1, w2);
        e[u].push_back({v, w});
        if (db == 'Y')
        {
            e[v].push_back({u, w});
        }
    }
    dijkstra(1);
    printf("%lld", d[n]);
    return 0;
}
```

C++ 参考程序(链式前向星)

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define n 1437
#define m 4096
#define big 0x7ffffffa
struct edge
{
    ll to, nx, w;
} e[m * 2 + 3];
ll hd[n + 3], cnt, d[n + 3];
void adde(ll &u, ll &v, ll &w)
{
    e[++cnt] = {v, hd[u], w};
    hd[u] = cnt;
}
struct node
{
```

```
    ll v, i;
    bool operator<(const node &x) const { return v > x.v; }
};
bool vis[n + 3];
void dijkstra(ll s)
{
    for (ll i = 1; i <= n; ++i)
    {
        d[i] = big;
    }
    d[s] = 0;
    priority_queue<node> q;
    q.push({0, s});
    while (!q.empty())
    {

        node t = q.top();
        ll u = t.i;
        q.pop();
        if (vis[u])
        {
            continue;
        }
        vis[u] = true;
        for (ll i = hd[u], v; i; i = e[i].nx)
        {
            v = e[i].to;
            if (d[v] > d[u] + e[i].w)
            {
                d[v] = d[u] + e[i].w;
                if (!vis[v])
                {
                    q.push({d[v], v});
                }
            }
        }
    }
}
signed main()
{
    for (ll i = 1, u = 0, v = 0, w1 = 0, w2 = 0, w; i <= m; ++i)
    {
        u = (1399 * i * i + u * u) % n + 1;
        v = (1427 * i * i + v * v) % n + 1;
        if (u == v)
        {
            u = u % n + 1;
        }
        char db = i % 2 ? 'Y' : 'N';
        for (ll j = 1; j * j <= i; ++j) //优化下的找因子
        {
            if (i % j == 0)
            {
                w1 += j == i / j ? j : j + i / j;
            }
        }
        //优化下的找倍数
        ll num = m / i;                     //项数
```

```
77          ll isum = num * (i + i * num) / 2; //等差数列前num项和
78          w2 += isum;
79          w1 = w1 % m + 1;
80          w2 = w2 % m + 1;
81          w = min(w1, w2);
82          adde(u, v, w);
83          if (db == 'Y')
84          {
85              adde(v, u, w);
86          }
87      }
88      dijkstra(1);
89      printf("%lld", d[n]);
90      return 0;
91  }
```

异世界的招聘

因为精度要求很高(小数点后六位)，故本题不可以直接用蒙特卡罗法或其他随机算法进行大量模拟求均值来达到频率 $\approx$ 概率，只能尝试用数学方法推导较精确值。

可以用 **DP** 来解本题。设 `dp[i][j]` 表示还差 i 个魔法师，还差 j 个其他职业者时，期望还需要招 $dp_{i,j}$ 人才能招满。令 $n=10, m=80$ ，则初始状态为 $dp_{0,0}=0$ ，所求为 $dp_{n,m}$ 。

先考虑简单的情况，假设已经招满了魔法师时，则每次招聘有 4% 概率招到其他职业者， 96% 概率找不到其他职业者(因为魔法师满了，所以招到魔法师也等于不符合条件)。由于招了一次人，所以积累了一次计数。对 $j>0$ ，有：

$$dp_{0,j}=1+4\%\times dp_{0,j-1}+96\%\times dp_{0,j}$$

移项，化简，得：

$$dp_{0,j}=dp_{0,j-1}+25$$

同理，假设已经招满了其他职业者时，对 $i>0$ ，有：

$$dp_{i,0}=1+1\%\times dp_{i-1,0}+99\%\times dp_{i,0}$$

移项，化简，得：

$$dp_{i,0}=100+dp_{i-1,0}$$

> 即使不用这种方法递推，这两个方程也可以很快由概率论知识得知。

考虑同样用这个方法进行一般情况的递推，类比可得：

$$dp_{i,j}=1+1\%\times dp_{i-1,j}+4\%\times dp_{i,j-1}+95\%\times dp_{i,j}$$

移项，化简，得：

$$dp_{i,j}=20+0.2\times dp_{i-1,j}+0.8\times dp_{i,j-1}$$

这个 DP 方程较难线性递推，可以考虑用**记忆化**搜索 **DFS** 求出。时空复杂度均为 $\mathrm{O}(nm)$ 。

C++ 参考代码

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef double db;
#define mn 88
db dp[mn][mn];
bool vis[mn][mn];
db dfs(ll x, ll y)
{
    if (vis[x][y])
    {
        return dp[x][y];
    }
    vis[x][y] = 1;
    dp[x][y] = 20 + 0.2 * dfs(x - 1, y) + 0.8 * dfs(x, y - 1);
    return dp[x][y];
}
signed main()
{
    vis[0][0] = 1;
    for (ll i = 1; i < mn; ++i)
    {
        dp[i][0] = 100 + dp[i - 1][0];
        dp[0][i] = 25 + dp[0][i - 1];
        vis[i][0] = vis[0][i] = 1;
    }
    printf("%.6lf", dfs(10, 80));
    return 0;
}
```

异世界的剧情分支

> 前置知识：40% 分解法： 杨辉三角、取模公式； 80% 分解法： 逆元； 100% 分解法： Lucas 定理

分析题意，可简化为将 m 个无区别的球放入 n 个有区别的桶里，求有多少种方法。使用隔板法，将 m 个球放在一排，在 m 个球中间组成的 $m-1$ 个空隙中插入 $n-1$ 个隔板，可以将其花费为 n 个区域，每个区域等效于一个桶里。插入隔板的方案数为 $m-1$ 个空隙选 $n-1$ 个位置，故所求为 C_{m-1}^{n-1}

对 20% 分数的数据，由于 $C_a^b=\dfrac{a!}{b!(a-b)!}$，直接暴力上阶乘即可。复杂度为 $\mathrm{O}(tn)$。

对 40% 分数的数据，可以用**杨辉三角**递推预处理算出所有组合数： $C_a^b=C_{a-1}^b+C_{a-1}^{b-1}$，时间复杂度为 $\mathrm{O}(n^2+t)=\mathrm{O}(n^2)$，空间复杂度为 $\mathrm{O}(n^2)$

对于 80% 分数的数据，可以预处理范围内的全部阶乘和全部阶乘**逆元**，时间复杂度为 $\mathrm{O}(n\log n+n+t)=\mathrm{O}(n\log n)$，空间复杂度为 $\mathrm{O}(n)$

对 100% 分数的数据，可以在预处理逆元和逆元阶乘基础上，使用 **Lucas 定理**。设 $p=99991$，时间复杂度为 $\mathrm{O}(p\log p+p+t\log n)=\mathrm{O}(t\log n)$，空间复杂度为 $\mathrm{O}(p)$

C++ 参考程序(100%得分解法)

```cpp
#include <bits/stdc++.h>
typedef long long ll;
#define p 99991
ll a[p + 5], inv[p + 5], t, n, m;
ll qpow(ll a, ll b)
{
    ll r = 1;
    for (; b; b >>= 1, a = a * a % p)
    {
        if (b & 1)
        {
            r = r * a % p;
        }
    }
    return r;
}
ll c(ll d, ll u) //d个里选u个的组合数
{
    if (u > d)
    {
        return 0;
    }
    return a[d] * inv[u] % p * inv[d - u] % p;
}
ll lucas(ll d, ll u)
{
    if (!u)
    {
        return 1;
    }
    return c(d % p, u % p) * lucas(d / p, u / p) % p;
}
signed main()
{
    inv[0] = a[0] = 1;
    for (ll i = 1; i <= p; ++i)
    {
        a[i] = a[i - 1] * i % p;
        inv[i] = qpow(a[i], p - 2);
    }
    scanf("%lld", &t);
    while (t--)
    {
        scanf("%lld%lld", &n, &m);
        printf("%lld\n", lucas(n - 1, m - 1));
    }
    return 0;
}
```

C++ 参考程序(80%得分解法)

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define mn 100010
```

```
ll p = 99991, t, n, m, a[mn], inv[mn];
ll qpow(ll a, ll b)
{
    ll r = 1;
    for (; b; b >>= 1, a = a * a % p)
    {
        if (b & 1)
        {
            r = r * a % p;
        }
    }
    return r;
}
ll c(ll d, ll u)
{
    return a[d] * inv[u] % p * inv[d - u] % p;
}
signed main()
{
    inv[0] = a[0] = 1;
    for (ll i = 1; i < mn; ++i)
    {
        a[i] = a[i - 1] * i % p;
        inv[i] = qpow(a[i], p - 2);
    }
    scanf("%lld", &t);
    while (t--)
    {
        scanf("%lld%lld", &n, &m);
        printf("%lld\n", c(n - 1, m - 1));
    }
    return 0;
}
```

C++ 参考程序(40%得分解法)

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define mn 1010
ll c[mn][mn], t, n, m, p = 99991; //c[d][u]代表 C(down,up-1)
signed main()
{
    c[0][1] = 1;
    for (ll i = 1; i < mn; ++i)
    {
        for (ll j = 1; j <= i + 1; ++j)
        {
            c[i][j] = (c[i - 1][j - 1] + c[i - 1][j]) % p;
        }
    }
    scanf("%lld", &t);
    while (t--)
    {
        scanf("%lld%lld", &n, &m);
        printf("%lld\n", c[n - 1][m]); //从n-1个选m-1个
    }
```

```
    return 0;
}
```

C++ 参考程序(20%得分解法)

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
ll fact(ll n)
{
    ll r = 1;
    for (ll i = 2; i <= n; ++i)
    {
        r *= i;
    }
    return r;
}
ll c(ll u, ll d)
{
    return fact(d) / fact(u) / fact(d - u);
}
ll t, n, m;
signed main()
{
    scanf("%lld", &t);
    while (t--)
    {
        scanf("%lld%lld", &n, &m);
        printf("%lld\n", c(m - 1, n - 1));
    }
    return 0;
}
```

异世界的人偶训练

双方均使用最优策略进行游戏，所以这是一道**博弈论**题目。

先考虑如何判断字符串是否是回文串，根据回文串定义，只需要依次判断正数第 i 个字符与倒数第 i 个字符是否都相等即可，一个循环便可解决。时间复杂度为 $\mathrm{O}(n)$ 。

首先考虑回文串的情况。

1. 若只有一个 0 时，是回文串，先手不能翻转，只能将其变为 1 ，游戏结束，先手耗费 1 魔力，后手耗费 0 魔力，后手必胜。
2. 若只有两个 0 ，先手只能将任意一个变为 1 ，随后后手可以翻转字符，之后根据限制，先手也只能再将剩下一个 0 变为 1 ，先手耗 2 ，后手耗 0 ，后手必胜。
3. 若全是 0 且个数为偶数个 (大于 2 个)，那么先手只能任选一个变为 1 ，无论选哪一个，此时后手可以选择与之对称的另一个 0 变为 1 ；在这之后，双方耗费相同，等效于把字符串中这两个对称的 1 去掉，把双方耗费相抵消，变成只有 $n-2$ 个 0 的子问题；不断如此操作，直到最后还剩下两个 0 时，化为上述第二点情况，所以先手耗 2 ，后手耗 0 (不计抵消时是先手耗 $\lfloor \frac{n}{2} \rfloor + 2$ ，后手耗 $\lfloor \frac{n}{2} \rfloor$)，后手必胜。
4. 如果有奇数个 0 (大于 1 个)，先手可以选择最中央那个 0 变为 1 (即第 $\lfloor \frac{n}{2} \rfloor$)，在这之后，等效于把这个 1 去掉，变成先手多耗费了 1 魔力下的 $n-1$ 个 0 的子问题，且 $n-1$ 是偶数，根据上述第

三点，最终结果为：剩下的操作中先手耗 $\lfloor\frac{n-1}{2}\rfloor$，后手耗 $\lfloor\frac{n-1}{2}\rfloor+2$，加上最开始的消耗，共计先手耗 $\lfloor\frac{n-1}{2}\rfloor+1$，后手耗 $\lfloor\frac{n-1}{2}\rfloor+2$，所以先手少消耗 1，所以先手必胜。

5. 如果有对称的 1，那么可以把这些 1 全部删掉，变成只有 0 的上述四种情况之一。

综上所述，在回文串下，不可能平局；设回文串有 s_1 个 0。当 $s_1=1$ 或 $s_1 \bmod 2=0$ 时，后手必胜，否则，先手必胜。到此为止解决了前 60% 得分的情况。

现在考虑不是回文串的情况。

1. 若所有字符都是非对称的(即前第 i 个字符与后第 i 个字符均不相等)，那么先手可以一直翻转字符，强迫后手一直将 0 变 1；最终若一共有 s_2 个非对称的 0，结果为先手耗 0，后手耗 s_2，先手必胜。
2. 如果有对称的 0，对称的 0 的个数可以设为 s_1 (在回文串下即 $s_2=0$，$s_1=0$ 的个数)，假设 $s_1=1$，即 n 为奇数只在第 $\lfloor\frac{n}{2}\rfloor$ 个字符是 0。特别地，若 $s_2=1$ 时，先手翻转后，后手可以把中间的 0 变为 1，则下次先手不能翻转只能把另一个 0 变 1；若先手不翻转，应选择把非对称 0 变为 1 (否则后手可以翻转)，然后后手再把剩下的 0 变为 1。所以当 $s_1=1,s_2=1$ 时，平局。
3. 否则，若 $s_1>0$；直到非对称 0 被全部变为 1 前，先手同样可以一直翻转。直到非对称 0 变成 1 后，原本的先后手不变，转化为之前讨论过的回文串问题。

   在回文串问题里，$s_1>1$ 且 $s_1 \bmod 2=1$ 时先手必胜；

   否则，若 $s_1=1$ 后手比先手少耗 1，但 $s_2\geq 1$，所以依然先手必胜 (或根据上述情况二平局)

   否则，若 $s_1 \bmod 2=0$ 后手比先手少耗 2。那么，当 $s_2>2$ 时，先手必胜；

   否则，若 $s_2=1$，先手可以先直接处理掉唯一不对称的 0，此时是 $s_1 \bmod 2=0$ 回文串问题，转化为上述子问题时先后手互换，所以转回来时得先手比后手少耗 2，再抵消掉一开始不对称的 0，得先手比后手少耗 1，先手必胜。

   否则，则 $s_2=2$，先手可以先翻转一次，强迫对方处理非对称 0 (对方处理对称 0 对对方而言更不优)之后，转化为 $s_2=1$ 的情况，根据上文，先手比后手少耗 1，由于后手处理了一个非对称 0，所以最终先手比后手少耗 2，先手必胜。

   综上所述，$s_1>0$ 时，先手必胜。

总结上述分析，发现对于非回文情况：若 $s_1=s_2=1$，平局；否则，先手必胜。

由于回文串等效于 $s_2=0$，所以最终总结为：

- $s_2=0$ 的前提下，$s_1=1$ 或 $s_1 \bmod 2=0$，后手必胜
- 否则，若 $s_2=s_1=1$，平局
- 否则，先手必胜

> 注：此题是 codeforces 原题，链接 在此

C++ 参考程序(100%分数代码)

```cpp
#include <bits/stdc++.h>
typedef long long ll;
using namespace std;
char s[1010];
ll t, n, s1, s2;
int main()
{
    scanf("%lld", &t);
    while (t--)
    {
        scanf("%lld%s", &n, s);
```

```
        s1 = s2 = 0;
        for (ll i = 0, j = n - 1; i <= j; i++, j--)
        {
            if (s[i] == s[j])
            {
                if (s[i] == '0')
                {
                    if (i == j) //最中间字符
                    {
                        s1 += 1;
                    }
                    else
                    {
                        s1 += 2;
                    }
                }
            }
            else
            {
                s2++;
            }
        }
        if (s2 == 0 && (s1 == 1 || s1 % 2 == 0))
        {
            printf("-1\n");
        }
        else if (s1 == 1 && s2 == 1)
        {
            printf("0\n");
        }
        else
        {
            printf("1\n");
        }
    }
}
```

异世界的稳定测试

> 前置知识：前缀和

先不考虑移动操作，仅考虑测试操作：

直接计算测试的复杂度为平均数计算的复杂度加标准差计算复杂度，均为求和，单次计算复杂度为 $O(n)$ 。

对于测试操作，两边取平方，有：

$$s^2 = \frac{\sum_{i=l}^{r}(x_i - \overline{x})^2}{r-l+1}$$

$$s^2 = \frac{\sum_{i=l}^{r}(x_i^2 - 2x_i\overline{x} + \overline{x}^2)}{r-l+1}$$

$$s^2=\frac{\sum_{i=l}^{r}x_i^2-2(r-l+1)\overline{x}\sum_{i=l}^{r}x_i+(r-l+1)\overline{x}^2}{r-l+1}$$

$$s^2=\frac{\sum_{i=l}^{r}x_i^2-2\overline{x}\sum_{i=l}^{r}x_i+(r-l+1)\frac{\sum_{i=l}^{r}x_i}{r-l+1}\overline{x}}{r-l+1}$$

$$s^2=\frac{\sum_{i=l}^{r}x_i^2-2\overline{x}\sum_{i=l}^{r}x_i+(r-l+1)\frac{\sum_{i=l}^{r}x_i}{r-l+1}\overline{x}}{r-l+1}$$

$$s^2=\frac{\sum_{i=l}^{r}x_i^2-2\overline{x}\sum_{i=l}^{r}x_i+\overline{x}\sum_{i=l}^{r}x_i}{r-l+1}$$

$$s^2=\frac{\sum_{i=l}^{r}x_i^2-\overline{x}\sum_{i=l}^{r}x_i}{r-l+1}$$

设前缀和 $a_i=\sum_{j=1}^{i}x_j$， 平方前缀和数组 $b_i=\sum_{j=1}^{i}x_j^2$，由前缀和公式，有：

$$\overline{x}=\frac{\sum_{i=l}^{r}x_i}{r-l+1}=\frac{\sum_{i=1}^{r}x_i-\sum_{i=1}^{l-1}x_i}{r-l+1}=\frac{a_r-a_{l-1}}{r-l+1}$$

同理根据前缀和公式，原式有：

$$s^2=\frac{b_r-b_{l-1}-\overline{x}(a_r-a_{l-1})}{r-l+1}$$

$$s=\sqrt{\frac{b_r-b_{l-1}-\overline{x}(a_r-a_{l-1})}{r-l+1}}$$

那么在不移动的前提下，预处理复杂度为 $\mathrm{O}(n)$，单次计算复杂度为 $\mathrm{O}(1)$

现考虑移动，可以发现在题给移动规则下，队列可以等效于循环队列，每次移动相当于把队首右移一位。对于这个循环队列构成的环，为计算方便，可考虑将其转化为链，把原队列复制拓展一倍为二倍长，把队首下标 p 设为 0。每移动一次把队首下标加一，若大于等于 n，重新从 0 开始，即 $p_{now}=(p_{prev}+k)\bmod n$，那么，取 $[l,r]$ 区间将等效于取 $[l+p,r+p]$，由于 $2\le r\le n,0<p<n$，可以保证 $1\le r+p\le 2n$，不超过二倍长的链。则每次移动只需要更改坐标 p，时间复杂度为 $\mathrm{O}(1)$

总时间复杂度为 $\mathrm{O}(n+m)$，空间复杂度为 $\mathrm{O}(n)$

C++ 参考代码(100%分解法)

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define il inline
typedef double db;
#define sc(x) scanf("%lld", &x)
#define mn 200010 //二倍长
ll n, m, x[mn], c, l, r, k, s[mn], s2[mn], p;
signed main()
{
    sc(n), sc(m);
    for (ll i = 1; i <= n; ++i)
    {
        sc(x[i]), x[i + n] = x[i];
    }
    for (ll i = 1; i <= n * 2; ++i)
```

```
    {
        s[i] = s[i - 1] + x[i];
        s2[i] = s2[i - 1] + x[i] * x[i];
    }
    while (m--)
    {
        sc(c);
        if (c == 1)
        {
            sc(k);
            p = (p + k) % n;
        }
        else
        {
            sc(l), sc(r);
            ll slr = s[p + r] - s[p + l - 1];
            ll s2lr = s2[p + r] - s2[p + l - 1];
            db avg = 1.0 * slr / (r - l + 1);
            db ans = sqrt(1.0 * (s2lr - slr * avg) / (r - l + 1));
            printf("%lf\n", ans);
        }
    }
    return 0;
}
```

> 20%分写法即完全暴力模拟，这里不展示代码

异世界的模块合并

> 前置知识：并查集、链表、二分

对价值积表达式 $w = \prod_{i=1}^{n} 2^{n^2+i}$ 和阶乘表达式 $(p!)^p$ ，发现它们都指数增长，很快就超过了基本数据类型所能表示的范围，并且即使用高精度，可能也会 TLE 或 MLE 。考虑如何优化计算，降低计算的数量级。

使用幂的性质 $2^a \times 2^b = 2^{a+b}$ 与四则运算结合律，得：

$$\prod_{i=1}^{n} 2^{n^2+i} = 2^{\sum_{i=1}^{n} n^2+i}$$

对下列等式：

$$2^{\sum_{i=1}^{n} n^2+i} = (p!)^p$$

以 2 为底数，两边取对数，得：

$$\sum_{i=1}^{n} n^2 + i = \log_2 (p!)^p$$

对等式右边，用 $p! = \prod_{i=1}^{p} i$ 拆开，且使用对数的两条性质 $log_a b^c = c log_a b$ 和 $\log_c ab = \log_c a + log_c b$ ，加上四则运算结合律，得：

$$log_2 (p!)^p = p \log_2 p! = p log_2 \prod_{i=1}^{p} i = p \sum_{i=1}^{p} \log_2 i$$

即最终问题转化为比较 $\sum_{i=1}^{n} n^2 + i$ 与 $p \sum_{i=1}^{p} \log_2 i$ 的大小关系。

由于 $1 \le n \le 10^5, 1 \le p \le 10^7$ ，故：

$$\sum_{i=1}^{n} n^2 + i < \sum_{i=1}^{n} n^2 + n = n^3 + n^2 \approx 10^{15}$$

$$p \sum_{i=1}^{p} \log_2 i < p \sum_{i=1}^{p} \log_2 p = p^2 \log_2 p \approx 2.2 \times 10^{15}$$

所以转化后的式子可以分别用 $long\ \ long$, $double$ 等变量来存储，避免了高精度。

如果使用暴力模拟来进行依模块找组、合并、统计，每次操作复杂度均为 $\mathrm{O}(n)$ 。因此实施第 k 种方案复杂度为 $\mathrm{O}(kn + kn + n) = \mathrm{O}(kn)$ 。考虑优化。

下面分析合并操作。依据模块找组，是经典的**并查集**操作。每次合并两组，只需要每次把较小组的父亲设为较大组即可。最后统计剩余的组，即父亲仍为自己的组，求对数和即可。依模块找组的均摊复杂度可视为 $\mathrm{O}(1)$ ，合并复杂度为 $\mathrm{O}(1)$ ，统计复杂度为 $\mathrm{O}(n)$ 。

特别地，考虑两组都是同一组的情况，需要找到下一个不同的组，如果用遍历枚举的方法，每次查找需要 $\mathrm{O}(n)$ 的复杂度，复杂度太高。考虑使用**链表**优化。可以用静态循环双链表维护所有未被合并的组，初始化为 n 节点循环双链表，每次合并把较小的节点进行删除操作。这样删除和查找的复杂度都是 $\mathrm{O}(1)$ ，初始化的复杂度为 $\mathrm{O}(n)$ 。由此，合并的复杂度仍为 $\mathrm{O}(1 + 1 + 1) = \mathrm{O}(1)$ 。

因此实施第 k 种方案复杂度为 $\mathrm{O}(n + k + k + n) = \mathrm{O}(n)$ 。

如果要遍历所有方案，共有 $n - 1$ 种方案，总时间复杂度为 $\mathrm{O}(n(n-1)) = \mathrm{O}(n^2)$ 。仍然不能过题。考虑继续优化。

下面证明一个结论：有 m 个组时的价值积一定不小于有 $m - 1$ 个组时的价值积。赛时只要发现这点，就可以继续往下做了，而不必严格证明。

严格证明如下：考虑最坏情况，即 m 个组取价值最小的前 m 组， $m - 1$ 个组取价值最大的后 $m - 1$ 个组，设 m 个组的价值积为 s_1 ， $m - 1$ 个组的价值积为 s_2 ，有：

$$s_1 = \prod_{i=1}^{m} 2^{n^2+i}$$

$$s_2 = \prod_{i=1}^{m-1} 2^{n^2-i+1}$$

以 2 为底数取对数，得：

$$\log_2 s_1 = \sum_{i=1}^{m} n^2 + i = mn^2 + \frac{m(m+1)}{2}$$

$$\log_2 s_2 = \sum_{i=1}^{m-1} n^2 - i + 1 = (m-1)n^2 - \frac{m(m-1)}{2} + (m-1)$$

作差，得：

$$log_2 s_1 - log_2 s_2 = n^2 + 1 + m^2 - m$$

由于 $m \ge 1$ ，故 $m^2 - m \ge 0$ ，故 $\log_2 s_1 - \log_2 s_2 > 0$

即： $\log_2 s_1 > log_2 s_2$ ，两边以 2 为底作指数还原，得： $s_1 > s_2$

即使在最坏情况下，均有 $s_1 > s_2$ ，而不难发现，一般情况下， m 个组的价值积 $s_m \ge s_1$ 恒成立；$m - 1$ 个组的价值积 $s_2 \ge s_{m-1}$ 恒成立，即下面不等关系恒成立：

$$s_m > s_{m-1}$$

因此，严格证明了有 m 个组时的价值积一定不小于有 $m-1$ 个组时的价值积。换句话说，以 m 为自变量时，价值积单调递增。所以价值积满足单调性。因此，可以**二分** m，即用二分答案法来二分变量 k。二分次数为 $\log_2(n-1)\approx\log_2 n$。每次二分实施方案 k，根据上文，单次实施方案时间复杂度为 $\mathrm{O}(n)$，因此，找方案的总时间复杂度为 $\mathrm{O}(n\log n)$。

注意到计算 $\log_2(p!)^p=p\sum_{i=1}^{p}\log_2 i$ 需要时间，对数运算复杂度为 $\mathrm{O}(1)$，所以计算该表达式的值所需时间为 p 次对数运算的时间，即复杂度为 $\mathrm{O}(p)$。一次计算好后存储起来多次调用即可。

综上所述，总时间复杂度为 $\mathrm{O}(p+n\log n)$。

C++ 参考代码(100%分数解法)

```cpp
#include <bits/stdc++.h>
using namespace std;
#define mn 500010
typedef long long ll;
typedef double db;
db ps;
ll fa[mn], n, p, lf, rf, cf, k, nx[mn], pr[mn];
ll findfa(ll x)
{
    while (x != fa[x])
    {
        x = fa[x] = fa[fa[x]];
    }
    return x;
}
bool ok()
{
    for (ll i = 1; i <= n; ++i)
    {
        fa[i] = i;
        nx[i] = i + 1;
        pr[i] = i - 1;
    }
    nx[n] = 1, pr[1] = n;
    for (ll j = 1; j <= cf; ++j)
    {
        ll u = (j + cf) % n + 1;
        ll v = ((j + cf) * (j + cf)) % n + 1;
        ll fu = findfa(u), fv = findfa(v);
        if (fu == fv)
        {
            fv = nx[fv];
        }
        ll minf = min(fu, fv), maxf = max(fu, fv);
        fa[minf] = maxf;
        nx[pr[minf]] = nx[minf];
        pr[nx[minf]] = pr[minf];
    }
    ll cnt = 0;
    for (ll i = 1; i <= n; ++i)
    {
        if (fa[i] == i)
        {
            cnt += n * n + i;
        }
    }
```

```
    return cnt > ps;
}
signed main()
{
    scanf("%lld%lld", &n, &p);
    lf = 1, rf = n - 1;
    for (ll i = 1; i <= p; ++i)
    {
        ps += log2(i);
    }
    ps *= p;
    while (lf <= rf)
    {
        cf = (lf + rf) / 2;
        if (ok())
        {
            k = cf;
            lf = cf + 1;
        }
        else
        {
            rf = cf - 1;
        }
    }
    if (k > 0)
    {
        printf("%lld", k);
    }
    else
    {
        printf("impossible");
    }
    return 0;
}
```

C++ 参考代码(80%分数解法)

```
#include <bits/stdc++.h>
using namespace std;
#define mn 500010
typedef long long ll;
typedef double db;
db ps;
ll fa[mn], n, p, lf, rf, cf, k;
ll findfa(ll x)
{
    while (x != fa[x])
    {
        x = fa[x] = fa[fa[x]];
    }
    return x;
}
bool ok()
{
    for (ll i = 1; i <= n; ++i)
    {
        fa[i] = i;
```

```
    }
    for (ll j = 1; j <= cf; ++j)
    {
        ll u = (j + cf) % n + 1;
        ll v = ((j + cf) * (j + cf)) % n + 1;
        ll fu = findfa(u), fv = findfa(v);
        if (fu == fv)
        {
            for (fv = fu + 1; fv != fu; fv = fv % n + 1)
            {
                if (fa[fv] == fv)
                {
                    break;
                }
            }
        }
        ll minf = min(fu, fv), maxf = max(fu, fv);
        fa[minf] = maxf;
    }
    ll cnt = 0;
    for (ll i = 1; i <= n; ++i)
    {
        if (fa[i] == i)
        {
            cnt += n * n + i;
        }
    }
    return cnt > ps;
}
signed main()
{
    scanf("%lld%lld", &n, &p);
    lf = 1, rf = n - 1;
    for (ll i = 1; i <= p; ++i)
    {
        ps += log2(i);
    }
    ps *= p;
    while (lf <= rf)
    {
        cf = (lf + rf) / 2;
        if (ok())
        {
            k = cf;
            lf = cf + 1;
        }
        else
        {
            rf = cf - 1;
        }
    }
    if (k > 0)
    {
        printf("%lld", k);
    }
    else
    {
        printf("impossible");
```

```
    }
    return 0;
}
```

> 40%分解法即去掉二分后的解法。20%分解法使用任意暴力解法均可，若不对数化需要注意 2^m 的计算需要使用 $double$ ，会爆 $long\ \ long$ 乃至 $\_\_int128$ 。

异世界的魔法优化

> 前置知识：树链剖分、线段树

先假设是在数组上维护上述操作，要维护 10^5 次操作，直接暴力显然超时。尽管两个操作看起来都像是树状数组所维护的内容，但树状数组无法同时实现区间修改和区间求值。所以考虑使用**线段树**维护操作。

题目要求支持三种操作：区间减 $lowbit$ (最低位)； 区间最高位左移； 区间求和。但是除了区间求和外，由于不满足结合律，都不是线段树支持的基础操作。现对其进行转化，使其能由线段树维护。

若只有区间减 $lowbit$ 操作和区间求和操作，可以令线段树每个节点维护区间求和值。

可以发现， a_i 只有 $\lceil \log a_i \rceil$ 位，所以至多进行 $\log a_i$ 次操作后就会变为 0 。这是一种数值快速下降到稳定的操作(相似的操作还有区间开根、区间求欧拉函数) 。可以考虑暴力修改，每次区间操作直接精确到单点，然后再经由 $pushup$ 操作得到区间求和。对于已经是 0 的数，可以标记标签代表操作完成，并且 $pushup$ 传递标签。当发现区间带标签时，意味着整个区间都是 0 ，则无需继续往下。 (不能直接判断求和值是否为 0 ，这是因为存在 $\sum a \bmod p = 0$ 的可能性) 由此，最坏情况下单次操作需要覆盖整个区间，每个数都被操作 $\log a_i$ 次，共有 n 个数，加上线段树本身的复杂度 $\mathrm{O}(\log n)$ ，所以总共最多区间 $lowbit$ 的复杂度为 $\mathrm{O}(n \log n \log a_i)$ 。在这之后，无论怎么操作，根节点已带标签，每次操作直接在根节点阻断，复杂度为 $\mathrm{O}(1)$ 。

现在考虑把区间最高位左移也加进去。这个操作只与最高位有关，可以单独维护最高位，把每个数 a_i 拆分为最高位 s_1 和其余位 s_2 ，使得 $s_1 + s_2 = a_i$ 。让线段树分别维护 s_1, s_2 。那么每次区间左移，等效于对线段树维护的 s_1 进行区间乘法，乘以 2 。该操作可以用懒标记优化，使得复杂度为 $\mathrm{O}(\log n)$ 。m 次操作总复杂度为 $\mathrm{O}(m \log n)$ 。

那么区间求和只需要分别将 s_1, s_2 求和加起来即可。

由此，在数组上便可以维护上述的所有操作，总时间复杂度为 $\mathrm{O}(n \log n \log a_i)$ 。

现在考虑将上述操作在树上实现。对树链全部节点进行多次更改和多次查询，显然是**树链剖分**的常见应用。任取根节点，用轻重链剖分把树划分为若干树链。根据树链剖分的性质，任意一条链均可由不超过 $\log n$ 条重链或重链的一部分拼接而成。在使用 DFS 序重建树上维护线段树，可以把对这 $\log n$ 条重链的操作转化为 $\log n$ 次线段树区间操作。因此，总时间复杂度是 $\mathrm{O}(n \log^2 n \log a_i)$ 。由于 $\log^2 n \log a_i \approx 8 \times 10^3$ ，可以过题。

> 对 20% 分的做法，可以暴力地从 l 开始 **DFS** 找 r ，维护 DFS 路径，并暴力对链操作即可。时间复杂度是 $\mathrm{O}(nq)$ 。

> 注：本题强化改编自2021暑假杭电集训(即2021"MINIEYE杯"中国大学生算法设计超级联赛)8-1004题 `Counting Stars` 。原题链接 在此 ，与原题的区别在于增加了树剖。

C++ 参考代码(100%分解法)

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define mn 200010
```

```
ll n, q, a[mn], mod = 998244353;
ll hd[mn], cnt;
struct edge
{
    ll to, nx;
} e[mn];
void addedge(ll u, ll v)
{
    e[++cnt] = {v, hd[u]};
    hd[u] = cnt;
}
ll dep[mn], fa[mn], siz[mn], son[mn]; //son是重儿子
void dfs1(ll u, ll f, ll deep)
{
    dep[u] = deep, fa[u] = f, siz[u] = 1;
    ll heavysiz = -1;
    for (ll i = hd[u], v; i; i = e[i].nx)
    {
        v = e[i].to;
        if (v == f)
        {
            continue;
        }
        dfs1(v, u, deep + 1);
        siz[u] += siz[v];
        if (siz[v] > heavysiz)
        {
            son[u] = v;
            heavysiz = siz[v];
        }
    }
}
ll id[mn], w[mn], top[mn], dfn; //dfs序；重建树权重；树链最浅端点
void dfs2(ll u, ll f)
{
    id[u] = ++dfn, w[dfn] = a[u], top[u] = f;
    if (!son[u])
    {
        return;
    }
    dfs2(son[u], f);
    for (ll i = hd[u], v; i; i = e[i].nx)
    {
        v = e[i].to;
        if (v == fa[u] || v == son[u])
        {
            continue;
        }
        dfs2(v, v);
    }
}
ll a1[mn], a2[mn];                                          //最高位值，其他位值和
ll s1[mn << 2], s2[mn << 2], lz[mn << 2], tg[mn << 2]; //最高位，其他位和，懒标记，lowbit标记
#define lfs p << 1
#define rfs p << 1 | 1
#define llcf ll cf = (lf + rf) >> 1
void pushup(ll p)
```

```
{
    s1[p] = (s1[lfs] + s1[rfs]) % mod;
    s2[p] = (s2[lfs] + s2[rfs]) % mod;
    tg[p] = tg[lfs] & tg[rfs];
}
void build(ll p, ll lf, ll rf)
{
    lz[p] = 1, tg[p] = 0;
    if (lf == rf)
    {
        ll x = w[lf];
        for (ll k = 30; k >= 0; --k) //将w拆分为最高位s1和其他位之和s2
        {
            if ((1 << k) <= x) //倒找最大位
            {
                s1[p] = 1 << k;
                s2[p] = x - s1[p];
                break;
            }
        }
        return;
    }
    llcf;
    build(lfs, lf, cf);
    build(rfs, cf + 1, rf);
    pushup(p);
}
ll lowbit(ll x) { return x & (-x); }
void pushdown(ll p)
{
    lz[lfs] = lz[lfs] * lz[p] % mod; //区间乘法懒标记
    lz[rfs] = lz[rfs] * lz[p] % mod;
    s1[lfs] = s1[lfs] * lz[p] % mod;
    s1[rfs] = s1[rfs] * lz[p] % mod;
    tg[lfs] |= tg[p]; //更大的区间低位减完了，子肯定也都完了(0+0=0)
    tg[rfs] |= tg[p];
    if (tg[lfs]) //完了就置零
    {
        s2[lfs] = 0;
    }
    if (tg[rfs])
    {
        s2[rfs] = 0;
    }
    lz[p] = 1;
}
void update_better(ll p, ll lf, ll rf, ll lc, ll rc) //区间[lc,rc] lowbit
{
    if (lf == rf)
    {
        if (s2[p])
        {
            s2[p] -= lowbit(s2[p]);
        }
        else //低位减完了
        {
            s1[p] = 0; //其他位减完了，这次减最高位
            tg[p] = 1; //标记减完了
```

```
        }
        return;
    }
    pushdown(p);
    llcf;
    if (lc <= cf && !tg[lfs]) //左子还没减完
    {
        update_better(lfs, lf, cf, lc, rc);
    }
    if (rc > cf && !tg[rfs])
    {
        update_better(rfs, cf + 1, rf, lc, rc);
    }
    pushup(p);
}
void update_worse(ll p, ll lf, ll rf, ll lc, ll rc) //区间[lc,rc]最高位左移
{
    if (lc <= lf && rf <= rc)
    {
        s1[p] = s1[p] * 2 % mod;
        lz[p] = lz[p] * 2 % mod;
        return;
    }
    pushdown(p);
    llcf;
    if (lc <= cf)
    {
        update_worse(lfs, lf, cf, lc, rc);
    }
    if (rc > cf)
    {
        update_worse(rfs, cf + 1, rf, lc, rc);
    }
    pushup(p);
}
ll res;
void query(ll p, ll lf, ll rf, ll lc, ll rc) //区间[lc,rc]求和
{
    if (lc <= lf && rf <= rc)
    {
        res = (res + s1[p] + s2[p]) % mod;
        return;
    }
    pushdown(p);
    llcf;
    if (lc <= cf)
    {
        query(lfs, lf, cf, lc, rc);
    }
    if (rc > cf)
    {
        query(rfs, cf + 1, rf, lc, rc);
    }
}
auto f = query; //函数指针，指向三种操作之一
void operate(ll u, ll v)
{
    while (top[u] != top[v])
```

```
    {
        if (dep[top[u]] < dep[top[v]])
        {
            swap(u, v);
        }
        f(1, 1, n, id[top[u]], id[u]);
        u = fa[top[u]];
    }
    if (dep[u] > dep[v])
    {
        swap(u, v);
    }
    f(1, 1, n, id[u], id[v]);
}
signed main()
{
    scanf("%lld", &n);
    for (ll i = 1; i <= n; ++i)
    {
        scanf("%lld", &a[i]);
    }
    for (ll i = 1, u, v; i < n; ++i)
    {
        scanf("%lld%lld", &u, &v);
        addedge(u, v), addedge(v, u);
    }
    dfs1(1, 0, 1);
    dfs2(1, 1);
    build(1, 1, n);
    scanf("%lld", &q);
    for (ll c, l, r; q; --q)
    {
        scanf("%lld%lld%lld", &c, &l, &r);
        if (c == 1)
        {
            f = update_better;
        }
        else if (c == 2)
        {
            f = update_worse;
        }
        else if (c == 3)
        {
            res = 0;
            f = query;
        }
        operate(l, r);
        if (c == 3)
        {
            printf("%lld\n", res);
        }
    }
    return 0;
}
```

C++ 参考程序(20%分解法)

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define mn 100010
struct edge
{
    ll to, nx;
} e[mn * 2];
ll hd[mn], cnt, n, q, s1[mn], s2[mn], a, c, l, r, mod = 998244353;
ll nx[mn], found, ans;
void adde(ll &u, ll &v)
{
    e[++cnt] = {v, hd[u]};
    hd[u] = cnt;
}
void dfs(ll u, ll fa, ll des)
{
    if (u == des)
    {
        nx[u] = -1;
        found = 1;
        return;
    }
    for (ll i = hd[u], v; i; i = e[i].nx)
    {
        v = e[i].to;
        if (v != fa && !found)
        {
            nx[u] = v;
            dfs(v, u, des);
        }
    }
}
signed main()
{
    scanf("%lld", &n);
    for (ll i = 1; i <= n; ++i)
    {
        scanf("%lld", &a);
        for (ll k = 30; k >= 0; --k)
        {
            if ((1 << k) <= a)
            {
                s1[i] = 1 << k;
                s2[i] = a - s1[i];
                break;
            }
        }
    }
    for (ll i = 1, u, v; i < n; ++i)
    {
        scanf("%lld%lld", &u, &v);
        adde(u, v), adde(v, u);
    }
    for (scanf("%lld", &q); q; --q)
```

```
    {
        scanf("%lld%lld%lld", &c, &l, &r);
        found = ans = 0;
        dfs(l, -1, r);
        for (ll u = l; u != -1; u = nx[u])
        {
            if (c == 1)
            {
                if (!s2[u])
                {
                    s1[u] = 0;
                }
                else
                {
                    s2[u] -= s2[u] & (-s2[u]);
                }
            }
            else if (c == 2)
            {
                s1[u] = s1[u] * 2 % mod;
            }
            else
            {
                ans = (ans + s1[u] + s2[u]) % mod;
            }
        }
        if (c == 3)
        {
            printf("%lld\n", ans);
        }
    }
    return 0;
}
```