2023 年牛客多校第六场题解

出题人：杭州电子科技大学

1 A Tree

题意：给定 n 个点的一棵边带权的树，点有黑白二色（0,1 表示），现在可以以 a_i 的价值翻转第 i 个点的颜色，一对异色点 (u,v) 的价值为树上路径的最大边权值。问经过任意颜色翻转后，价值减去代价的最大值。$1 \le n \le 3 \times 10^3$，$1 \le a_i \le 10^9$。

解法：注意到点对的价值等于路径上最大边权值，不难想到 Kruskal 重构树——即根据边权从小到大的顺序枚举边，考虑加入这条边前后该边两端点所在连通块合并的贡献。显然由于边权是从小到大枚举的，因而每次尝试根据边 (u,v,w) 执行并查集合并的时候，该边两端点 u,v 所在连通块点集 S_u,S_v 点两两之间（$x \in S_u, y \in S_v$）的价值就为该边边权 w。

不难发现，一条边合并的贡献仅与两侧点集中黑点和白点数目有关，而与两侧点集内部黑点和白点分布无关——原点集内部黑白点分布不会经过这条新加入的边，因而不会对该边和该边边权贡献计算产生任何影响。因而可以考虑对于每个点集 S_u 维护一个 dp 数组 f_{S_u}，第 i 项表示该集合内有 i 个白点的最大价值（该点集内已经有的价值减去该点集内翻转代价），初始时 $f_{S_u,a_u}=0$ 而 $f_{S_u,a_u \oplus 1}=-a_i$ 即颜色不同时要减去翻转代价。当一条边要合并时，转移是显然的：

$$f_{S'_u,i}=\max_{0\le k\le |S_u| \wedge 0 \le i-k \le |S_v|} f_{S_u,k}+f_{S_v,i-k}+w(k(|S_v|-i+k)+(|S_u|-k)(i-k))$$

使用启发式合并就可以做到整体 $\mathcal{O}(n^2)$ 的合并速度。

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 3000;
const long long inf = 0x3f3f3f3f3f3f3f3fll;
int father[N + 5], a[N + 5], siz[N + 5];
long long cost[N + 5];
int getfather(int x)
{
    return father[x] == x ? x : father[x] = getfather(father[x]);
}
vector<long long> f[N + 5];
void merge(int u, int v, long long w)
{
    u = getfather(u);
    v = getfather(v);
    if (u == v)
        return;
    if (siz[u] > siz[v])
        swap(u, v);
    int n = f[u].size() - 1, m = f[v].size() - 1;
    vector<long long> temp(n + m + 1);
    for (auto &x : temp)
        x = -inf;
```

```
    for (int i = 0; i <= n + m; i++)
        for (int j = 0; j <= n; j++)
            if (i - j <= m && i >= j)
            {
                int k = i - j;
                int cnt = j * (m - k) + (n - j) * k;
                temp[i] = max(temp[i], f[u][j] + f[v][k] + cnt * w);
            }
    f[v] = temp;
    f[u].clear();
    father[u] = v;
    siz[v] += siz[u];
}
struct edge
{
    int u, v;
    long long w;
    bool operator<(const edge &b) const
    {
        return w < b.w;
    }
    edge(int _u, int _v, long long _w) : u(_u), v(_v), w(_w) {}
};
int main()
{
    int n;
    scanf("%d", &n);
    for (int i = 1; i <= n; i++)
    {
        scanf("%d", &a[i]);
        father[i] = i;
        siz[i] = 1;
        f[i].resize(2);
    }
    for (int i = 1; i <= n; i++)
    {
        scanf("%lld", &cost[i]);
        f[i][a[i] ^ 1] = -cost[i];
    }
    vector<edge> q;
    for (int i = 1, u, v, w; i < n; i++)
    {
        scanf("%d%d%d", &u, &v, &w);
        q.emplace_back(u, v, w);
    }
    sort(q.begin(), q.end());
    for (auto [u, v, w] : q)
```

```
        merge(u, v, w);
    auto ans = f[getfather(1)];
    printf("%lld", *max_element(ans.begin(), ans.end()));
    return 0;
}
```

2 B Distance

题意：给定长度为 n 的两个序列 $\{a\}_{i=1}^n$ 和 $\{b\}_{i=1}^n$，从中选出两个大小相等的集合 $S \subseteq A, T \subseteq B$，每次可以选择 $x \in S$ 或 $y \in T$ 并执行 $x \leftarrow x+1$ 或 $x \leftarrow x-1$ 或 $y \leftarrow y+1$ 或 $y \leftarrow y+1$。记将两个集合变成相同的最小操作次数为 $c(S,T)$，求 $\sum\limits_{S \subseteq A}\sum\limits_{T \subseteq B} c(S,T) \bmod 998\,244\,353$。$1 \le n \le 3 \times 10^3$，$1 \le a_i, b_i \le 10^9$。

解法：显然本题不太能枚举集合去依次计算贡献，但是注意到如果两个子集 $S \subseteq A, T \subseteq B$ 有序且能匹配，则必然是 S 中第 i 大元素 S_i 转移到 T_i，因而可以考虑拆分算贡献——即每一对 (a_i, b_j) 如果是匹配的，那么有多少种情况让它们匹配，每种情况对答案贡献是多少（这里是显然的 $|a_i - b_j|$），独立去看这一对对答案的贡献。

考虑 $\{a\}_{i=1}^n$ 前 i 个数和 $\{b\}_{i=1}^n$ 前 j 个数选出相同个数的数字的方案数为 $f_{i,j}$，不妨令 $i \le j$，则有：

$$\begin{aligned} f_{i,j} &= \sum_{k=0}^{i} \binom{i}{k}\binom{j}{k} \\ &= \sum_{k=0}^{i} \binom{i}{i-k}\binom{j}{k} = \binom{i+j}{i} \end{aligned}$$

此处使用范德蒙德卷积公式。

此处也可以有另一种理解：考虑证明转移式 $f_{i,j} \leftarrow f_{i-1,j} + f_{i,j-1}$ 的正确性。假定 $f_{i-1,j-1}, f_{i,j-1}, f_{i-1,j}$ 正确，考虑它如何转移到 $f_{i,j}$：(i,j) 同选或同不选，或者从 $f_{i,j-1}$ 或 $f_{i-1,j}$ 转移而来。而 $f_{i-1,j}$ 和 $f_{i,j-1}$ 中各包含一份重复的 $f_{i-1,j-1}$，所以 $f_{i-1,j}$ 和 $f_{i,j-1}$ 恰好完全囊括所有的情况。因而 $f_{i,j} \leftarrow f_{i-1,j} + f_{i,j-1}$ 转移式正确。

因而对于 a_i 和 b_j 匹配的情况，前后缀的方案数使用组合数公式计算即可。总复杂度 $\mathcal{O}(n^2)$。

```
#include <bits/stdc++.h>
using namespace std;
const int P = 998244353, N = 2000;
long long pre[N + 5][N + 5], suf[N + 5][N + 5];
int a[N + 5], b[N + 5];
int main()
{
    int n;
    scanf("%d", &n);
    for (int i = 1; i <= n; i++)
        scanf("%d", &a[i]);
    for (int i = 1; i <= n; i++)
        scanf("%d", &b[i]);
    sort(a + 1, a + n + 1);
    sort(b + 1, b + n + 1);
    for (int i = 0; i <= n; i++)
        pre[i][0] = pre[0][i] = 1;
```

```
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            pre[i][j] = (pre[i - 1][j] + pre[i][j - 1]) % P;
    for (int i = 0; i <= n + 1; i++)
        suf[i][n + 1] = suf[n + 1][i] = 1;
    for (int i = n; i >= 1; i--)
        for (int j = n; j >= 1; j--)
            suf[i][j] = (suf[i][j + 1] + suf[i + 1][j]) % P;
    long long ans = 0;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
        {
            long long cur = abs(a[i] - b[j]);
            ans = (ans + cur * pre[i - 1][j - 1] % P * suf[i + 1][j + 1]) % P;
        }
    printf("%lld", ans);
    return 0;
}
```

3 C idol!!

题意：给定 n，求 $\prod_{i=1}^{n} i!!$ 的末尾 0 个数。$1 \le n \le 10^{18}$。

解法：利用性质 $(n-1)!! \times n!! = n!$ 将双阶乘组合一下。对于偶数，原式等价于 $\prod_{i=1}^{\frac{n}{2}} (2i)!$，而奇数等价于 $\prod_{i=1}^{\frac{n+1}{2}} (2i-1)!$。

显然在统计阶乘末尾 0 的时候，5 的数目会远少于 2 的数目，因而本质等价于统计含 5 因子的个数——枚举 k，统计有多少个数会是 5^k 的倍数，对所有 k 求和。

首先考虑求 $n!$ 一项末尾 0 个数。对于一个最高含有 5^k 的阶乘项 n（即 $\text{argmax}\{k|5^k|n\}$），对于 5^i 次项，$n!$ 中至少有 $\left\lfloor \frac{n}{5^i} \right\rfloor$ 个乘积项存在 5^i 即 i 个 5 因子，则有 $\left\lfloor \frac{n}{5^i} \right\rfloor - \left\lfloor \frac{n}{5^{i+1}} \right\rfloor$ 个数有且仅有 i 个 5 因子。考虑对这些求和：

$i = 1 \to 5$，$n = 30$，6 个含 5 的倍数，但是只有 5 个是恰好最高包含 5——存在 25 包含了 5^2。

$$\begin{aligned}
\text{ans} &= \sum_{i=1}^{N} i \left(\left\lfloor \frac{n}{5^i} \right\rfloor - \left\lfloor \frac{n}{5^{i+1}} \right\rfloor \right) \\
&= \sum_{i=1}^{N} (i - (i-1)) \left\lfloor \frac{n}{5^i} \right\rfloor - N \left\lfloor \frac{n}{5^{N+1}} \right\rfloor \\
&= \sum_{i=1}^{k} \left\lfloor \frac{n}{5^i} \right\rfloor
\end{aligned}$$

$i = 1$，$n/5 - n/25$，$i = 2$，$n/25 - n/125$。

其中 N 充分大，满足 $5^N \ge n$，因而 $\left\lfloor \frac{n}{5^{N+1}} \right\rfloor = 0$。因而 $n!$ 一项的贡献（5 因子数目）为 $\sum_{i=1}^{k} \left\lfloor \frac{n}{5^i} \right\rfloor$。

因而原式（下文以偶数为例，奇数同理转化为 $2k-1$）等价于：

$$\sum_{i=1}^{N}\left(\sum_{j=1}^{\frac{n}{2}}\left\lfloor\frac{2j}{5^i}\right\rfloor\right)$$

其中 N 充分大，满足 $5^N \geq n$。内层括号等价于类欧形式 $\sum_{i=1}^{n}\left\lfloor\frac{ai+b}{c}\right\rfloor$，可以直接套用类欧公式计算。

当然由于本例中式子较为简单，因而可以直接使用等差数列计算，一次性计算 2×5^k 的范围内数字的和，这个和是等差数列。这样做复杂度为 $\mathcal{O}(\log n)$。

```cpp
#include <bits/stdc++.h>
#define fp(i, a, b) for (int i = a, i##_ = b; i <= i##_; ++i)
#define fd(i, a, b) for (int i = a, i##_ = b; i >= i##_; --i)

using namespace std;
using ll = long long;
using i128 = __int128_t;
ll n; i128 c;
void print(i128 x) {
    if (!x) return;
    print(x / 10);
    putchar(x % 10 + '0');
}
void Solve() {
    scanf("%lld", &n);
    if (n & 1) {
        ll k = n + 1;
        while (k) c += k / 5, k /= 5;
        k = (n + 1) / 2;
        while (k) c -= k / 5, k /= 5;
        --n;
    }
    auto S2 = [](i128 x) { return x * (x + 1) / 2; };
    for (i128 k = 5; k <= n; k *= 5)
        c += k / 2 * S2(n / k - 1) + 2 * S2(n / k / 2) + n / k * ((n % k + 1) / 2);
    if (!c) puts("0");
    else print(c), puts("");
}
int main() {
    int t = 1;
    // scanf("%d", &t);
    while (t--) Solve();
    return 0;
}
```

4 E Sequence

题意：给定长度为 n 的序列 $\{a\}_{i=1}^{n}$，q 次询问区间 $[l,r]$ 上是否可以找到长度为 $k+1$ 数列 $\{b\}_{i=0}^{k}$ 满足 $l=b_0\le b_1<b_2<\cdots b_{k-1}<r=b_k$ 且 $\forall i\in[0,k-1]$，$\sum_{b_i+1}^{b_{i+1}}a_i$ 均为偶数。多测，$1\le T\le 10^4$，$\sum n,\sum q\le 10^5$，$1\le a_i\le 10^{10}$。

解法：显然 $\sum_{i=l}^{r}a_i$ 必为偶数，否则无解。

考虑前缀和的奇偶性。显然对于前缀和数组 s 而言，$s_{l-1},s_{b_1},s_{b_2},\cdots,s_r$ 的奇偶性全部相同。因而只需要前缀和处理前缀和数组的奇偶性，统计 $[l-1,r]$ 区间上和 s_{l-1} 奇偶性相同的 s_x 个数即可。复杂度 $\mathcal{O}(\sum n+\sum q)$。

```cpp
#include <bits/stdc++.h>
#define fp(i, a, b) for (int i = a, i##_ = int(b); i <= i##_; ++i)
#define fd(i, a, b) for (int i = a, i##_ = int(b); i >= i##_; --i)

using namespace std;
using ll = long long;
const int N = 1e5 + 5;
int n, q, sum[N], cnt[N][2];
void Clear() {
    fp(i, 1, n) sum[i] = cnt[i][0] = cnt[i][1] = 0;
}
void Solve() {
    scanf("%d%d", &n, &q);
    ll x;
    fp(i, 1, n) {
        scanf("%lld", &x);
        sum[i] = sum[i - 1] ^ (x & 1);
        cnt[i][0] = cnt[i - 1][0] + (sum[i] == 0);
        cnt[i][1] = cnt[i - 1][1] + (sum[i] == 1);
    }
    int l, r, k;
    while (q--) {
        scanf("%d%d%d", &l, &r, &k);
        if (sum[r] != sum[l - 1]) puts("NO");
        else if (cnt[r][sum[r]] - cnt[l - 1][sum[r]] >= k)
            puts("YES");
        else
            puts("NO");
    }
}
int main() {
    int t = 1;
    scanf("%d", &t);
    while (t--) Solve();
    return 0;
```

```
}
```

5 G Gcd

题意：初始时集合 S 内有两个数 x, y，一次操作中可以选取两个数 a, b 将 $a - b$ 或 $\gcd(|a|, |b|)$ 插入集合。问最后能否使得元素 z 在集合中。$0 \le x, y, z \le 10^9$。

解法：不难注意到这个操作就是辗转相减，因而可以得到 $\gcd(a, b)$。这时所有 $\gcd(a, b)$ 的倍数（因为可以产生负数）就都可以得到了。注意特判 $z = 0$ 的情况。

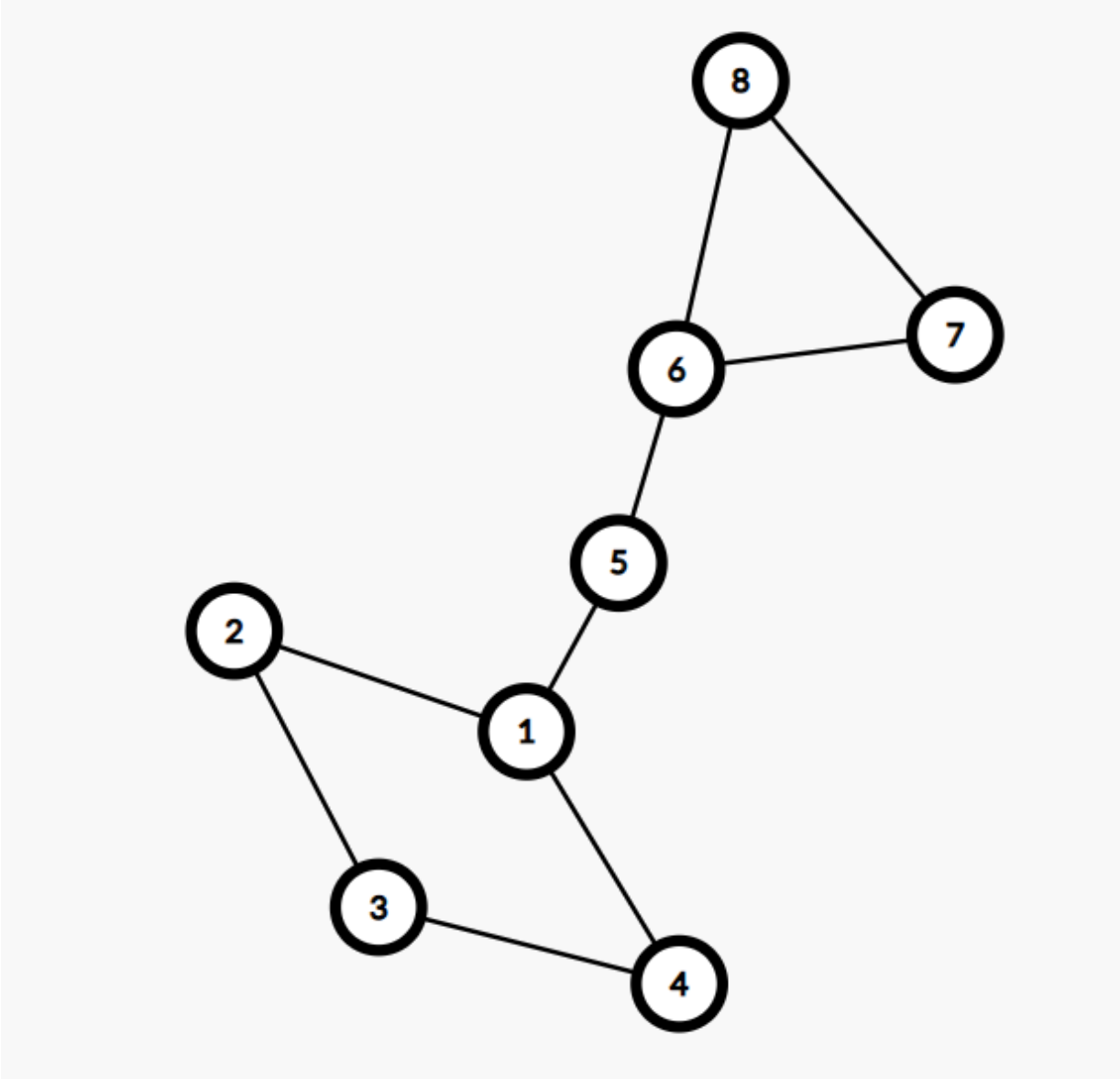
```cpp
#include <bits/stdc++.h>

using namespace std;
void Solve() {
    int a, b, c, d = 0;
    scanf("%d%d%d", &a, &b, &c);
    puts((!c && (!a || !b)) || (c && c % __gcd(a, b) == 0) ? "YES" : "NO");
}
int main() {
    int t = 1;
    scanf("%d", &t);
    while (t--) Solve();
    return 0;
}
```

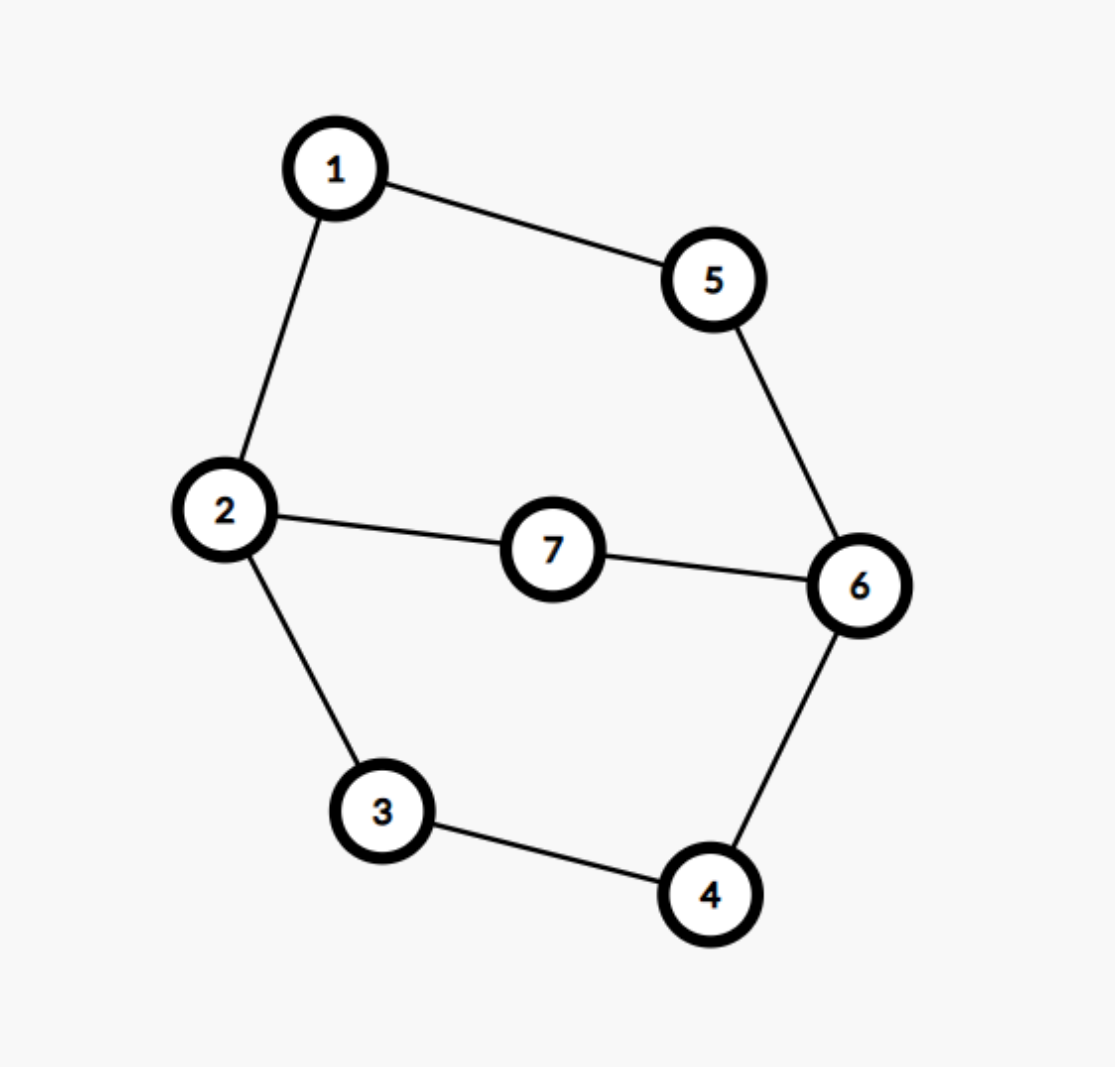
6 H traffic

题意：给定一个 n 个点 $n + 1$ 条边的无向连通图，每条边边权是一个关于 t 的一次函数 $a_i t + b_i$，要求分别算出 $t = 0, 1, 2, 3, \cdots, T$ 时图的最小生成树的边权和。$1 \le n \le 10^5$，$0 \le a_i, b_i \le 10^8$。

解法：n 个点 $n + 1$ 条边的图存在两个环，现在需要保留其中一个生成树。那么可能存在以下两种情况：



对于两个独立环，那么最大去除两条边等价于每一个环去除一条边。



对于嵌环（两个环有公共边），那么这时图上共有三个本质不同的环和三条半环链（图中 $2 \to 1 \to 5 \to 6, 2 \to 7 \to 6, 2 \to 3 \to 4 \to 6$）。这时三条半环边上任取两条半环链，每条半环链上去除一条边。

因而问题转化为 $[0, T]$ 上查询若干个一次函数（半环链上的边权值）的最大值。这个问题可以使用李超树直接解决。当然本题由于都是对全域 $[0, T]$ 上操作，因而单调队列维护这一凸包或者半平面交都是可以的。

问题转化为如何快速找环。这里可以考虑使用并查集，依次枚举边并尝试合并，如果边上两点已经联通，则说明发现了环，将返祖边该两点颜色进行染色。等所有边处理完成后，再进行一次 dfs 将颜色进行传播。不难发现如果点在链上，则根据数学归纳法得到异或出来的值必然为 0，而返祖边所在的环上的每个点都会染上返祖边的颜色。这时点上颜色就和边上颜色相同。如果颜色使用恰当（如 2^k），则可以快速找到每个点和每条边在哪些环上。

```cpp
#include <bits/stdc++.h>
#define fp(i, a, b) for (int i = a, i##_ = b; i <= i##_; ++i)
#define fd(i, a, b) for (int i = a, i##_ = b; i >= i##_; --i)

using namespace std;
using ll = long long;
const int N = 1e5 + 5;
// 李超树
class LC_SegT {
    struct Func {
        ll k, b;
```

```
        Func(ll k = 0, ll b = 0) : k(k), b(b) {}
        ll operator()(const ll x) const { return k * x + b; }
    } tr[N << 2];
#define lc (p << 1)
#define rc (p << 1 | 1)
    void insert(int p, int L, int R, int a, int b, Func k) {
        int m = (L + R) >> 1;
        if (a <= L && R <= b) {
            Func &F = tr[p];
            int res = (k(L) > F(L)) + (k(R) > F(R));
            if (res == 2) F = k;
            else if (res) {
                if (k(m) > F(m)) swap(F, k);
                if (k(L) > F(L)) insert(lc, L, m, a, b, k);
                else insert(rc, m + 1, R, a, b, k);
            }
            return;
        }
        if (a <= m) insert(lc, L, m, a, b, k);
        if (b > m) insert(rc, m + 1, R, a, b, k);
    }
#undef lc
#undef rc
  public:
    int n;
    void init(const int Lim) { for (n = 1; n < Lim; n <<= 1); }
    void insert(Func k) { insert(1, 1, n, 1, n, k); }
    ll qry(int x) {
        ll res = 0;
        for (int p = x + n - 1; p; p >>= 1)
            res = max(res, tr[p](x));
        return res;
    }
} T[3];
int n, m, q, dft, fa[N], id[N], tg[N], dfn[N];
ll sA, sB;
struct Edge { int a, b; };
vector<Edge> E;
vector<int> C[5];
map<pair<int, int>, int> mp;
vector<pair<int, int>> G[N], stk;
int GF(int x) { return fa[x] == x ? x : fa[x] = GF(fa[x]); }
int merge(int x, int y) { return x = GF(x), y = GF(y), fa[x] = y, x != y; }
void dfs(int u, int p) {
    // u->v的边有边编号e
    for (auto [v, e] : G[u]) {
        if (e == p) continue;
```

```
        dfs(v, e), id[u] ^= id[v];
    }
    if (p >= 0) tg[p] = id[u];
}
void Solve() {
    scanf("%d%d", &n, &q), m = 1;
    fp(i, 1, n) fa[i] = i;
    for (int i = 0, u, v, a, b; i <= n; ++i) {
        scanf("%d%d%d%d", &u, &v, &a, &b);
        sA += a, sB += b, E.push_back({a, b});
        if (merge(u, v))
            G[u].push_back({v, i}), G[v].push_back({u, i});
        else
            id[u] ^= m, id[v] ^= m, tg[i] = m, m <<= 1;
    }
    dfs(1, -1);
    fp(i, 0, n) if (tg[i]) C[tg[i] - 1].push_back(i);
    fp(i, 0, 2) {
        T[i].init(q + 1);
        for (auto x : C[i])
            T[i].insert({E[x].a, E[x].b - E[x].a});
    }
    fp(i, 1, q + 1) {
        vector<ll> val(3);
        fp(j, 0, 2) val[j] = T[j].qry(i);
        sort(val.begin(), val.end());
        printf("%lld\n", sB - val[1] - val[2]);
        sB += sA;
    }
}
int main() {
    int t = 1;
    // scanf("%d", &t);
    while (t--) Solve();
    return 0;
}
```

7 J Even

题意：给定一个长度为 n 的序列 $\{a\}$。对于一个区间 $[l,r]$，在一次操作中，若区间内有正偶数，则令区间中偶数的最大值 $x \leftarrow \frac{x}{2}$，否则令最大值 $x \leftarrow \frac{x-1}{2}$。$q$ 次询问，求对一个区间操作 k 次以后，区间最大值。$1 \le n \le 10^4$，$1 \le q \le 10^5$，$1 \le a_i, k \le 10^9$。

解法：下面考虑一个区间 $[l,r]$ 中如何处理。在一个区间中，如果存在偶数，则必定是先将偶数部分全部除干净再开始处理奇数。因而在有奇有偶的情况下，可以先把偶数单独拎出来。

不妨设这个区间中偶数分别为 $\{p_1\times 2^{k_1},p_2\times 2^{k_2},\cdots,p_m\times 2^{k_m}\}$，其中 $p_1,p_2,\cdots,p_k$ 均为奇数，则可以考虑维护可重集 $\{\{p_1\times 2^{k_1},p_1\times 2^{k_1-1},\cdots,p_1\},\{p_2\times 2^{k_2},p_2\times 2^{k_2-1},\cdots,p_2\},\cdots,\{p_m\times 2^{k_m},\cdots,p_m\}\}$ 的第 k 大——一次操作处理的是一个最大值，并将它删掉。进行 k 次操作等价于新序列的第 $k+1$ 大（即尚未处理的数字）。

如果整个区间都是奇数，则处理一次之后如果变成偶数就会一直处理它直到整个序列再次恢复全部奇数的情况。假设对于奇数 a_i，它的处理链为 $a_i\to a_{i,1}\to a_{i,2}\cdots\to a_{i,k}$，其中 $a_i,a_{i,k}$ 为奇数，$a_{i,1},a_{i,2},\cdots,a_{i,k-1}$ 为偶数。则对于区间 $\{a_1,a_2,\cdots,a_l\}$ 可以考虑维护新序列 $\{\{a_1,a_1,a_1,\cdots,a_1,a_{1,k_1}\},\{a_2,a_2,a_2,\cdots,a_2,a_{2,k_2}\},\cdots,\{a_l,a_l,a_l,\cdots,a_l,a_{l,k_l}\}\}$。注意下标就是 a_i 不是 $a_{i,j}$，因为出现了偶数优先处理偶数，等价于在选取最大值的过程中仍然是原数字进行操作。因而处理 k 次操作仍然等价于新序列的第 $k+1$ 大。由于每个数字只会操作 $O(\log V)$ 轮，因而整个新序列长度仅有 $O(n\log V)$，使用主席树维护区间第 k 大的复杂度为 $O(\log^2 n)$，因而总复杂度 $\mathcal{O}(q\log V\log^2(n\log V))$。

```cpp
#include <bits/stdc++.h>
#define fp(i, a, b) for (int i = a, i##_ = b; i <= i##_; ++i)
#define fd(i, a, b) for (int i = a, i##_ = b; i >= i##_; --i)

using namespace std;
using ll = long long;
const int N = 1e4 + 5, M = 1 << 30;

struct Node {
    Node *l, *r; int s;
    Node(Node *t) { if (t) *this = *t; }
};
int n, q, cnt, res;
vector<Node *>t0, t1;
Node *add(Node *t, int l, int r, int x, int v) {
    t = new Node(t);
    t->s += v;
    if (r == l + 1) return t;
    int m = (l + r) >> 1;
    if (x < m) t->l = add(t->l, l, m, x, v);
    else t->r = add(t->r, m, r, x, v);
    return t;
}
int qry(Node *t1, Node *t2, int l, int r, int k) {
    if (r == l + 1) {
        if (l & 1) cnt = t1->s - t2->s, res = k;
        return l;
    }
    int s = t1->r->s - t2->r->s, m = (l + r) >> 1;
    if (s > k) return qry(t1->r, t2->r, m, r, k);
    return qry(t1->l, t2->l, l, m, k - s);
}
void Solve() {
    scanf("%d%d", &n, &q);
    Node *z = new Node({});
    z->l = z->r = z;
```

```
    t0.resize(n + 1), t0[0] = z, t1 = t0;
    for (int i = 1, x; i <= n; ++i){
        t0[i] = t0[i - 1], t1[i] = t1[i - 1];
        scanf("%d", &x);
        while (x % 2 == 0) t0[i] = add(t0[i], 0, M, x, 1), x /= 2;
        while (x) {
            int t = x == 1 ? 1 : __builtin_ctz(x / 2) + 1;
            t1[i] = add(t1[i], 0, M, x, t), x >>= t;
        }
    }
    for (int l, r, k, sum, ans; q--;) {
        scanf("%d%d%d", &l, &r, &k), --l;
        sum = t0[r]->s - t0[l]->s;
        // printf("sum=%d\n", sum);
        if (k <= sum)
            ans = max(qry(t0[r], t0[l], 0, M, k), qry(t1[r], t1[l], 0, M, 0));
        else {
            k -= sum;
            int x = qry(t1[r], t1[l], 0, M, k),
                c = x == 1 ? 1 : __builtin_ctz(x / 2) + 1;
            if (res <= cnt - c) ans = x;
            else ans = max(x >> (res - cnt + c), qry(t1[r], t1[l], 0, M, k - res +
cnt));
        }
        printf("%d\n", ans);
    }
}
int main() {
    int t = 1;
    // scanf("%d", &t);
    while (t--) Solve();
    return 0;
}
```