题解

Jump

为了方便定义第 $0, n+1$ 个点分别为 k, m，对应的 $b_i = 0$。

那么转移方程是显然的：$dp(i) = \max\limits_{0 \le j < i} \{dp(j) + b_i - \lceil \frac{T_i - T_j}{D} \rceil \times a\}$。

这个东西乍一看没法优化，因为那个上取整很不好弄。

此时有一个经典套路，利用余数的关系分讨拆开取整。

就是

$$\lceil \frac{T_i - T_j}{D} \rceil = \begin{cases} \lfloor \frac{T_i}{D} \rfloor - \lfloor \frac{T_j}{D} \rfloor + 1, & (T_i \bmod D > T_j \bmod D) \\ lfloor \frac{T_i}{D} \rfloor - \lfloor \frac{T_j}{D} \rfloor, & \text{otherwise.} \end{cases}$$

下取整可以换成 C++ 自带除法。

于是方程就可以分别写成 i, j 相关项，于是我们用一个权值线段树维护一下 $dp(i) + a \times \lfloor \frac{T_i}{D} \rfloor$，每次转移分别询问前后缀 $\max$ 即可。

代码参考

```
#include <cmath>
#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>

#define endl '\n'
#define int long long

using namespace std;
using i64 = long long;

const int si = 1e5 + 10;
const int inf = 1e18 + 10;

int n, m, d, k, a;
int t[si], b[si], c[si], dp[si];

int rt = 0;
class SegTree {
    private:
        int tot = 0;
        int ls[si * 50], rs[si * 50], val[si * 50];
        int node() { ++tot, ls[tot] = rs[tot] = 0, val[tot] = -inf; return tot;
}
        void pushup(int p) { val[p] = max(val[ls[p]], val[rs[p]]); }
    public:
        void init() { val[0] = -inf; } // ?
        void modify(int &p, int l, int r, int x, int v) {
            if(!p) p = node();
            if(l == r) return val[p] = max(val[p], v), void();
            int mid = (l + r) >> 1;
            if(x <= mid) modify(ls[p], l, mid, x, v);
```

```
            else modify(rs[p], mid + 1, r, x, v);
            pushup(p);
        }
        int qmax(int p, int l, int r, int ql, int qr) {
            if(!p) return -inf;
            if(ql <= l && r <= qr) return val[p];
            int mid = (l + r) >> 1, ret = -inf;
            if(ql <= mid) ret = max(ret, qmax(ls[p], l, mid, ql, qr));
            if(qr > mid) ret = max(ret, qmax(rs[p], mid + 1, r, ql, qr));
            return ret;
        }
} tr;

signed main() {
    cin.tie(0) -> sync_with_stdio(false);
    cin.exceptions(cin.failbit | cin.badbit);

    cin >> k >> m >> d >> a >> n;
    t[0] = k, t[n + 1] = m;
    b[0] = b[n + 1] = 0, c[0] = t[0] / d, c[n + 1] = t[n + 1] / d;

    tr.init();
    for(int i = 1; i <= n; ++i)
        cin >> t[i] >> b[i], c[i] = t[i] / d;
    dp[0] = 0, tr.modify(rt, 0, d - 1, t[0] % d, a * c[0]);
    for(int i = 1; i <= n + 1; ++i) {
        dp[i] = -inf;
        int v1 = tr.qmax(rt, 0, d - 1, 0, t[i] % d - 1) - a * c[i] - a + b[i];
        int v2 = tr.qmax(rt, 0, d - 1, t[i] % d, d - 1) - a * c[i] - a + b[i] +
a;
        dp[i] = max(v1, v2), tr.modify(rt, 0, d - 1, t[i] % d, dp[i] + a *
c[i]);
    }
    cout << dp[n + 1] << endl;

    return 0;
}
```

Match

注意到两个串匹配的充要条件是：对于 $\forall i \in [1, len]$，两个串对应位置的字符到上一个对应字符的距离相等。

比如 $S = aba, T = bab$，写出来就可以变成 $S' = 002, T' = 002$。

于是问题可以转化为一个简单的字符串匹配了。

但是注意到 $S = abab, T = aba$，若我们正在匹配 $S[2..4]$ 和 T，直接写出来就是：$S'[1..3] = 022, T' = 002$，发现不匹配了，这是因为 $S[1]$ 这个位置的贡献本来应该不算，直接取原串就会死掉。

所以我们删掉一个位置的时候，要考虑当前位置是否会对后面某一个位置产生影响。

这个利用字符串 Hash 可以 $O(1)$ 做，就找到对应位置，减去即可。

剩下的每次 $O(1)$ 扩展，都是字符串 Hash 的基本操作，很简单。

代码参考

```
#include <cmath>
#include <cstdio>
#include <vector>
#include <cstring>
#include <iostream>
#include <algorithm>

#define endl '\n'

using namespace std;
using i64 = long long;
using u64 = unsigned long long;

const int si = 1e6 + 10;
const int base = 1e8 + 7;

int Q, C, n, m;
int s[si], t[si];
int pre[si], nxt[si];
int pret[si], nxtt[si];
int prec[si], nxtc[si];
u64 Hash[si], pw[si], has;

int main() {

#ifdef ONLINE_JUDGE
    freopen("match.in", "r", stdin);
    freopen("match.out", "w", stdout);
#endif

    cin.tie(0) -> sync_with_stdio(false);
    cin.exceptions(cin.failbit | cin.badbit);

    cin >> Q >> C;
    Hash[0] = 0, has = 0, pw[0] = 1;
    for(int i = 1; i <= si - 10; ++i)
        pw[i] = pw[i - 1] * base;

    for(int nw = 1; nw <= Q; ++nw) {
        cin >> n >> m;
        for(int i = 1; i <= n; ++i) cin >> s[i];
        for(int i = 1; i <= m; ++i) cin >> t[i];

        for(int i = 0; i <= n; ++i) pre[i] = prec[i] = 0, nxt[i] = nxtc[i] = n +
1;
        for(int i = 1; i <= n; ++i) pre[i] = prec[s[i]], prec[s[i]] = i;
        for(int i = n; i >= 1; --i) nxt[i] = nxtc[s[i]], nxtc[s[i]] = i;
        for(int i = 0; i <= m; ++i) prec[i] = pret[i] = 0, nxtc[i] = nxtt[i] = m
+ 1;
        for(int i = 1; i <= m; ++i) pret[i] = prec[t[i]], prec[t[i]] = i;

        has = Hash[0] = 0;
        for(int i = 1; i <= n; ++i) Hash[i] = 0;
        for(int i = 1; i <= n; ++i) if(!pre[i]) pre[i] = i;
        for(int i = 1; i <= m; ++i) if(!pret[i]) pret[i] = i;
```

```
        for(int i = 1; i <= n; ++i) if(nxt[i] == n + 1) nxt[i] = i;
        for(int i = 1; i <= n; ++i) s[i] = i - pre[i];
        for(int i = 1; i <= m; ++i) t[i] = i - pret[i];
        for(int i = 1; i <= n; ++i) Hash[i] = Hash[i - 1] * base + s[i];
        for(int i = 1; i <= m; ++i) has = has * base + t[i];
        std::vector<int> ans;
        u64 val = Hash[m];
        for(int i = 1; i + m - 1 <= n; ++i) {
            if(has == val) ans.push_back(i);
            val = val - s[i] * pw[m - 1], val = val * base + s[i + m];
            if(nxt[i] && nxt[i] <= i + m)
                val = val - s[nxt[i]] * pw[i + m - nxt[i]];
            if(nxt[i]) s[nxt[i]] = 0;
        }
        cout << (int)ans.size() << endl;
        for(auto x : ans) cout << x << " ";
        cout << endl;
    }

    return 0;
}
```

Graph

注意到答案一定是最短路长度。

那么，直接求一次最短路，然后考虑 $S \to T$ 的路径上的一条边 (u, v)，它只需要被染色成 $\max(dis[S \to u], dis[S \to v])$ 即可。

其本质是找到合适的，类似于按层染色的方式使得方案成立。

正确性显然。

代码参考

```
#include <map>
#include <cmath>
#include <queue>
#include <cstdio>
#include <cstring>
#include <utility>
#include <iostream>
#include <algorithm>

#define endl '\n'
#define int long long

using namespace std;
using i64 = long long;

const int si = 4e2 + 10;

int n, m;
int tot = 0, head[si];
struct Edge { int ver, Next; } e[si * si << 1];
inline void add(int u, int v) { e[tot] = (Edge){v, head[u]}, head[u] = tot++; }

int S, T;
int dis[si];
```

```cpp
bool inq[si];
std::queue<int> q;
std::map<int, std::pair<int, int>> edg;

void Spfa(int s) {
    memset(dis, 0x3f, sizeof dis);
    memset(inq, false, sizeof inq);
    dis[s] = 0, q.push(s), inq[s] = true;
    while(!q.empty()) {
        int u = q.front();
        q.pop(), inq[u] = false;
        for(int i = head[u]; ~i; i = e[i].Next) {
            int v = e[i].ver;
            if(dis[v] > dis[u] + 1) {
                dis[v] = dis[u] + 1;
                if(!inq[v]) q.push(v), inq[v] = true;
            }
        }
    }
}

signed main() {

    freopen("graph.in", "r", stdin);
    freopen("graph.out", "w", stdout);

    cin.tie(0) -> sync_with_stdio(false);
    cin.exceptions(cin.failbit | cin.badbit);

    edg.clear();
    memset(head, -1, sizeof head);

    cin >> n >> m >> S >> T;
    for(int i = 1; i <= m; ++i) {
        int u, v;
        cin >> u >> v;
        add(u, v), add(v, u);
        edg[i] = make_pair(u, v);
    }
    Spfa(S), cout << dis[T] << endl;
    for(int i = 1; i <= m; ++i) {
        auto [u, v] = edg[i];
        int ans = max(dis[u], dis[v]);
        cout << (ans <= dis[T] ? ans : 0) << endl;
    }

    return 0;
}
```

Xor

看到这个问题，很容易想到直接把所有数丢进 Trie 树来考虑。

我们可以做一个 dp，设 $dp(u)$ 表示在以 u 为根的子树中选两个数，使得它们的异或和 $\geq X$ 的方案数。

我们假设 u 的层数为 i（从高到低插入），那么如果 $2^{i-1} > X$，显然不管我们怎么从 $ls(u), rs(u)$ 当中选，都一定可以选出大于等于 X 的方案。

所以 $dp(u) = dp(ls(u)) \times dp(rs(u))$.

其中 $ls(u)$ 表示 $tr[u, 0]$，rs 反之。

然后考虑，如果 2^{i-1} 恰好是 X 的最高位，那么此时一定分别从两边选，且每一边至多选一个，不然异或之后全是 0 了，不合法。

于是首先算上只选一个和空集的方案数，这部分是 $siz(u) + 1$。

剩下的部分的话，问题就转化为，在 $ls(u), rs(u)$ 当中分别选一个，使得它们的异或和大于等于 X，这个也可以 dp。

我们设 $f(x, y)$ 表示这个 dp 数组，并记 x, y 的层数为 j。

分类讨论 `bit(j) = X >> j & 1` 的取值，可以得到转移：

$$f(x,y) = \begin{cases} f(ls(x), ls(y)) + f(rs(x), rs(y)) + siz(ls(x)) \times siz(rs(y)) + siz(ls(y)) \times siz(rs(x)) & (bit(j) = 0) \\ f(ls(x), rs(y)) + f(ls(y), rs(x)) & (bit(j) = 1) \end{cases}$$

1 的话就是两边必须错开选，0 的话两边怎么选都行。

注意边界的时候要返回 siz!

而且 Trie 树处理的时候 bit 不要卡着上界 63，小一点，不然溢出了就寄了。

```
#include <cmath>
#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>

#define endl '\n'
#define int long long

using namespace std;
using i64 = long long;

const int si = 3e5 + 10;
const int mod = 998244353ll;

int n, X, tot = 1;
int tr[si * 64][2], val[si * 64], siz[si * 64];

void insert(int value) {
    int p = 1;
    for(int i = 62; i >= 0; --i) {
        int ch = (value >> i) & 1;
        if(!tr[p][ch]) tr[p][ch] = ++tot;
        siz[p] += 1, p = tr[p][ch];
    }
    val[p] = value, siz[p] += 1;
}
int calc(int x, int y, int dep) {
    if(!x || !y) return 0;
    if(dep == -1) return (val[x] xor val[y]) >= X ? siz[x] * siz[y] : 0ll;
    if(X >> dep & 1)
```

```
        return (calc(tr[x][0], tr[y][1], dep - 1) + calc(tr[x][1], tr[y][0], dep
- 1)) % mod;
    else return (calc(tr[x][0], tr[y][0], dep - 1) % mod + calc(tr[x][1], tr[y]
[1], dep - 1) % mod
                + siz[tr[x][0]] * siz[tr[y][1]] % mod + siz[tr[x][1]] * siz[tr[y]
[0]] % mod) % mod;
}
int dp(int u, int dep) {
    if(dep == -1) return (siz[u] + 1) % mod;
    if(!u) return 1ll;
    if((1ll << dep) > X)
        return dp(tr[u][0], dep - 1) * dp(tr[u][1], dep - 1) % mod;
    return (siz[u] + 1 + calc(tr[u][0], tr[u][1], dep - 1)) % mod;
}

signed main() {

#ifdef ONLINE_JUDGE
    freopen("xor.in", "r", stdin);
    freopen("xor.out", "w", stdout);
#endif

    cin.tie(0) -> sync_with_stdio(false);
    cin.exceptions(cin.failbit | cin.badbit);

    cin >> n >> X;
    for(int i = 1, x; i <= n; ++i)
        cin >> x, insert(x);
    cout << (dp(1, 62) - 1 + mod) % mod << endl;

    return 0;
}
```

这个题的启发式做题技巧是挺值得再次复习的。

不知道下次什么时候能用上。