T1 大数定理

解析

可以发现我们并不在意首尾两个数的大小，本题解中我们默认其为 0。

不妨先二分答案，小于等于 mid 的数看作 0 其他看作 1，那么我们每次只能操作 1 将序列删空。而实际上删空等价于将某个 1 移到中间（易证），而这样的 1 一定是 $\frac{n+1}{2}$ 左右的那两个 1。

假设这两个 1 分别在 u,v 处（题解第一行保证了一定存在这两个 1），且我们不妨考虑把 u 移到中间的情况，可以发现操作 u 及其左边的位置都是严格不优的。而我们能至多操作 $n-v$ 次（每次操作必然会删去一个 v 右侧的位置），因此我们至多能完成 $u \geqslant \frac{(n-2(n-v))+1}{2}$ 的情况，化简得 $\frac{n-1}{2} \geqslant v-u$。

实际上符合的情况我们也能完成，因为这个 $n-v$ 次的上界若达不到，则必然是 v 把 $[u+1,v-1]$ 这一段删空了，在这之前 u 一定成为了中点（因为中点始终向左移，且没有越过 u）。

对另一侧的分析可以得到相同的结果，即 $v-u \leqslant \frac{n-1}{2}$，直接模拟即可做到 $O(n\log n)$。

能不能做到更好的呢？我们继续分析：不合法的条件实际上等价于，$[2,n-1]$ 存在一个长度大于等于 $\frac{n-1}{2}$ 的 0 连续段（有这样的连续段一定会覆盖中点，因此会使得 $v-u > \frac{n-1}{2}$），我们也就得让每个 $[2,n-1]$ 内长度为 $\frac{n-1}{2}$ 的连续段都没有 0，取所有段最大值的最小值即为答案，使用单调队列模拟滑动窗口即可。

复杂度 $O(n)$。出题人也实现了 $O(n\log n)$ 的做法，运行速度与 $O(n)$ 差距不大。

参考代码

```cpp
#include <bits/stdc++.h>
using namespace std;
const int maxn = 5000005;
int T, n, m, hd, tl, ans;
int a[maxn], q[maxn];
int main()
{
    ios::sync_with_stdio(false);
    cin >> T;
    while (T--)
    {
        cin >> n;
        for (int i = 1; i <= n; i++)
            cin >> a[i];
        assert(n & 1);
        m = (n - 1) / 2, ans = 1e9;
        hd = 1, tl = 0;
        for (int i = 2; i <= n - 1; i++)
        {
            while (hd <= tl && i - q[hd] + 1 > m)
                hd++;
            while (hd <= tl && a[q[tl]] <= a[i])
                tl--;
            q[++tl] = i;
            if (i >= m + 1)
                ans = min(ans, a[q[hd]]);
```

```
        }
        printf("%d\n", ans);
    }
    return 0;
}

```

T2 中心极限定理

解析

踩掉马这一动作较难维护，一个朴素的想法是直接状压，可以得到 $O(n^2 2^m)$ 左右的复杂度。

但是注意到我们踩掉一个马的作用其实很有限，踩掉一个马后再走三步，这个马原本能攻击到的区域就和我们没有关系了。因此我们只需要记录前两步怎么走的，只考虑这三步踩掉的马即可。

复杂度 $O(n^2)$。

参考代码

```cpp
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1005, mod = 998244353;
int n, m;
int a[maxn][maxn], b[maxn][maxn], f[maxn][maxn][2][2];
int dx[5] = {0, 1, 1, -1, -1}, dy[5] = {0, -1, 1, -1, 1};
inline void inc(int &x, int y)
{
    x += y;
    if (x >= mod)
        x -= mod;
}
int hav(int x, int y)
{
    return x >= 0 && x <= n && y >= 0 && y <= n && a[x][y] && b[x][y] == 0;
}
int chk(int x, int y)
{
    for (int d = 1; d <= 4; d++)
        if (hav(x + dx[d], y + dy[d]) == 0 && (hav(x + dx[d], y + 2 * dy[d]) || hav(x + 2 * dx[d], y + dy[d])))
            return 1;
    return 0;
}
int main()
{
    scanf("%d%d", &n, &m);
    for (int i = 1, x, y; i <= m; i++)
        scanf("%d%d", &x, &y), a[x][y] = 1;
    if (chk(0, 0))
    {
        puts("0");
        return 0;
    }
    f[0][0][0][0] = 1;
    for (int i = 0; i <= n; i++)
        for (int j = 0; j <= n; j++)
            for (int c = 0; c <= 1; c++)
                for (int d = 0; d <= 1; d++)
                    if (f[i][j][c][d])
                    {
                        b[i][j] = 1;
```

```
                        if (i >= c && j >= (c ^ 1))
                            b[i - c][j - (c ^ 1)] = 1;
                        if (i >= c + d && j >= (c ^ 1) + (d ^ 1))
                            b[i - c - d][j - (c ^ 1) - (d ^ 1)] = 1;
                        if (chk(i, j + 1) == 0)
                            inc(f[i][j + 1][0][c], f[i][j][c][d]);
                        if (chk(i + 1, j) == 0)
                            inc(f[i + 1][j][1][c], f[i][j][c][d]);
                        b[i][j] = 0;
                        if (i >= c && j >= (c ^ 1))
                            b[i - c][j - (c ^ 1)] = 0;
                        if (i >= c + d && j >= (c ^ 1) + (d ^ 1))
                            b[i - c - d][j - (c ^ 1) - (d ^ 1)] = 0;
                    }
    printf("%d\n", (0ll + f[n][n][0][0] + f[n][n][0][1] + f[n][n][1][0] +
f[n][n][1][1]) % mod);
    return 0;
}
```

T3 散步

解析

我们考察最后的那条路径，其形态形如 $t \to p_{i_1} \to \cdots \to p_{i_2} \to t \to p_{i_3} \to \cdots \to p_{i_{2d}} \to t (\to p_{i_{2d+1}})$（$t$ 度数为奇数就有最后一段）。

对于 t 度数为偶数的情况，如果不考虑所有与 t 相连的边，我们就是要找一组 t 的邻居之间的匹配，使得存在边不交的路径集连接匹配的邻居。一个必要条件就是每个连通块大小是偶数，且如果满足该条件，可以用经典的树上贪心解决。

具体地，对于某个连通块，我们保留一棵生成树并任取一个点为根，设计一个搜索算法，遍历到点 x 时尽可能匹配其子树内的点，要求在子树大小为偶数时将其两两匹配，奇数时从根连出一条路径。那么我们可以发现我们只需对其每个儿子施用此算法，于是每个儿子会贡献至多一条路径上来，我们贪心地将自己儿子之间的路径配对，如果自身时关键点也检查一下是否可以配对，这一贪心算法一定能达到上述的要求。

复杂度 $O(n)$。

参考代码

```cpp
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1000005;
int T, n, m, t, tmps, tot;
int vis[maxn], tag[maxn], fa[maxn], tmp[maxn];
vector<int> ans;
vector<int> v[maxn];
int dfs(int x)
{
    int now = tag[x] ? x : 0;
    vis[x] = 1;
    for (int i = 0; i < v[x].size(); i++)
    {
        int y = v[x][i];
        if (vis[y])
            continue;
        int res = dfs(y);
        fa[y] = x;
        if (now && res)
        {
            for (int i = res; i != x; i = fa[i])
                ans.emplace_back(i);
            ans.emplace_back(x), tmps = 0;
            for (int i = now; i != x; i = fa[i])
                tmp[++tmps] = i;
            for (int i = tmps; i >= 1; i--)
                ans.emplace_back(tmp[i]);
            ans.emplace_back(t);
            now = 0;
        }
        else
            now |= res;
    }
```

```cpp
    return now;
}
int main()
{
    ios::sync_with_stdio(false), cin.tie(0), cout.tie(0);
    cin >> T;
    while (T--)
    {
        cin >> n >> m >> t, tot = 0, ans.clear();
        for (int i = 1, x, y; i <= m; i++)
        {
            cin >> x >> y;
            if (x == t || y == t)
                tag[x + y - t] = 1;
            else
                v[x].emplace_back(y), v[y].emplace_back(x);
        }
        ans.emplace_back(t);
        for (int i = 1; i <= n; i++)
            if (vis[i] == 0)
            {
                int res = dfs(i);
                if (res)
                {
                    tot++;
                    if (tot > 1)
                        break;
                    ans.insert(ans.begin(), res);
                }
            }
        if (tot > 1)
            puts("-1");
        else
        {
            printf("%d ", ans.size());
            for (int i = 0; i < ans.size(); i++)
                printf("%d%c", ans[i], i == ans.size() - 1 ? '\n' : ' ');
        }
        for (int i = 1; i <= n; i++)
            vis[i] = tag[i] = 0, v[i].clear();
    }
    return 0;
}
```

T4 买宝石

解析

考虑将所有点和询问按照时间进行排序，并按照这个顺序加入点和进行询问。

考虑使用树套树维护每一个点到根节点链上宝石相关信息。一个点 (x, y) 的权值表示点 rk_x 到祖先路径上，所有价格为 y 的宝石的价值和。

加入一个点 x 的时候，只会改变 x 子树内点的信息，每一个点加入了 k_x 个价格为 w_x 的宝石，在树套树上 $([dfn_x, dfn_x + siz_x), w_x)$ 这个矩阵里每一个元素增加 $k_x w_x$。

处理一个终点在 x 的询问的时候，本质上是在 $(dfn_x, [1, n])$ 这个区域内进行二分。由于我们使用的是树套树，所以这一列实际上是 $O(\log n)$ 个线段树的并，使用多棵线段树上同时二分的技巧即可。

时空复杂度均为 $O((n+q)\log^2 n)$，想要常数小一点可以把外层的线段树换为树状数组。

参考代码

```
#include <bits/stdc++.h>
#define mid (l + r >> 1)
#define lowbit(x) (x & -x)
using namespace std;
const int maxn = 100005, maxt = maxn * 200;
int n, m, tot, ps, res;
int k[maxn], w[maxn], t[maxn], qt[maxn], qx[maxn], lc[maxt], rc[maxt], pos[maxn], rt[maxn], ans[maxn], mn[maxt];
long long qc[maxn], sum[maxt];
vector<int> v[maxn], qv[maxn];
void modify(int l, int r, int &now, int p, int k)
{
    if (now == 0)
        now = ++tot;
    sum[now] += 1ll * k * p;
    if (l == r)
    {
        if (sum[now])
            mn[now] = l;
        return;
    }
    if (p <= mid)
        modify(l, mid, lc[now], p, k);
    else
        modify(mid + 1, r, rc[now], p, k);
    mn[now] = min(mn[lc[now]], mn[rc[now]]);
}
int query(int l, int r, long long k)
{
    if (l == r)
        return l;
    int MN = 1e9;
    long long s = 0;
    for (int i = 1; i <= ps; i++)
        s += sum[lc[pos[i]]], MN = min(MN, mn[rc[pos[i]]]);
    if (k > s && k - s >= MN)
```

```cpp
    {
        for (int i = 1; i <= ps; i++)
            pos[i] = rc[pos[i]];
        return query(mid + 1, r, k - s);
    }
    for (int i = 1; i <= ps; i++)
        pos[i] = lc[pos[i]];
    return query(l, mid, k);
}
void upd(int x, int typ)
{
    for (int i = t[x]; i <= n; i += lowbit(i))
        modify(0, n, rt[i], w[x], typ * k[x]);
}
void dfs(int x, int last)
{
    upd(x, 1);
    for (int i = 0; i < qv[x].size(); i++)
    {
        int k = qv[x][i];
        ps = 0;
        long long s = 0;
        for (int j = qt[k]; j; j -= lowbit(j))
            pos[++ps] = rt[j], s += sum[rt[j]];
        ans[k] = query(0, n, min(qc[k], s));
    }
    for (int i = 0; i < v[x].size(); i++)
        if (v[x][i] != last)
            dfs(v[x][i], x);
    upd(x, -1);
}
int main()
{
    ios::sync_with_stdio(false), cin.tie(0), cout.tie(0);
    mn[0] = 1e9, cin >> n;
    for (int i = 1; i <= n; i++)
        cin >> k[i] >> w[i] >> t[i];
    for (int i = 1, x, y; i < n; i++)
        cin >> x >> y, v[x].emplace_back(y), v[y].emplace_back(x);
    cin >> m;
    for (int i = 1; i <= m; i++)
        cin >> qt[i] >> qc[i] >> qx[i], qv[qx[i]].emplace_back(i);
    dfs(1, 0);
    for (int i = 1; i <= m; i++)
        cout << ans[i] << ' ';
    return 0;
}
```