CQUPC2022 个人题解

A

根据 $x \oplus x = 0$

$$\begin{aligned} & a \oplus b = c \\ \Rightarrow & a \oplus b \oplus a = c \oplus a \\ \Rightarrow & a \oplus c = b \end{aligned}$$

坑点：必须开 `unsigned long long`。

B

显然每个盒子最多可以分 $\left\lfloor \frac{\sum_{i=1}^{n} a_i}{n} \right\rfloor$ 个，用总数减去这个就是剩余的。

C

直接三次方循环计算矩阵即可。

D

先把所有字符读进来，然后以符号分隔开，挨个处理。

极丑代码：

```cpp
#include<bits/stdc++.h>
using namespace std;
#define int long long

signed main() {
    string s;
    cin >> s;
    int op = 0;
    vector<string> v;
    string now;
    for(int i = 0; i < s.length(); i++) {
        if(s[i] == '+' || s[i] == '-') {
            if(now != "") {
                if(op == 1) {
                    now = "-" + now;
                }
                v.push_back(now);
            }
            op = s[i] == '-';
```

```
                now = "";
            } else {
                now += s[i];
            }
        }
        if(now != "") {
            if(op == 1) {
                now = "-" + now;
            }
            v.push_back(now);
        }
        int ans = 0;
        map<char, int> mp;
        mp['I'] = 1;
        mp['V'] = 5;
        mp['X'] = 10;
        mp['L'] = 50;
        mp['C'] = 100;
        mp['D'] = 500;
        mp['M'] = 1000;
        for(int i = 0; i < v.size(); i++) {
//          cout << v[i] << ' ';
            int start = v[i][0] == '-';
            int now = 0;
            for(int j = start; j < v[i].size(); j++) {
                if(j + 1 < v[i].size() && mp[v[i][j]] < mp[v[i][j + 1]]) {
                    now -= mp[v[i][j]];
                } else {
                    now += mp[v[i][j]];
                }
//              cout << "j:" << j << " now:" << now << '\n';
            }
            ans = ans + (start == 1 ? -now : now);
        }
        cout << ans;

        return 0;
}
```

E

纯纯的模拟，极丑代码：

```
#include<bits/stdc++.h>
using namespace std;
char ch[1005][1005];
int main(){
    int t;
```

```
    cin>>t;
    while (t--)
    {
        int a,b,c;
        cin>>a>>b>>c;
        for (int i=1;i<=(b+c)*2+1;i++) for (int j=1;j<=(a+b)*2+1;j++) ch[i][j]='.';
        for (int i=1;i<=(b+c)*2+1;i+=2) for (int j=1;j<=(a+b)*2+1;j+=2) ch[i][j]='+';
//        for (int i=1;i<=b;i++) for (int j=(b-i)*2+4,pp=1;pp<=a;pp++,j+=2) ch[i*2-1][j]='-';
//        for (int i=b*2+1,pp=10;pp<=c;pp++,i+=2) for (int j=2;j<=2*a;j+=2) ch[i][j]='-';
        for (int i=1;i<=b;i++) for (int j=2;j<=(a+b)*2+1;j+=2) ch[i*2][j]='/';
        for (int i=2;i<=(b+c)*2+1;i+=2) for (int j=a*2+2;j<=(a+b)*2+1;j+=2) ch[i][j]='/';
        for (int i=1;i<=b;i++) for (int j=(b-i)*2+4,pp=1;pp<=a;pp++,j+=2) ch[i*2-1][j]='-';
        for (int i=b*2+1;i<=(b+c)*2+1;i+=2) for (int j=1;j<=a;j++) ch[i][j*2]='-';
        for (int i=b*2+2;i<=(b+c)*2+1;i+=2) for (int j=0;j<=a;j++) ch[i][j*2+1]='|';
        for (int i=1;i<=b;i++) for (int j=1;j<=c;j++) ch[(b-i+j)*2][(a+b)*2+1-(b-i+1)*2+2]='|';
        for (int i=1;i<=b*2;i++) for (int j=1;j<=b*2-i+1;j++) ch[i][j]='.';
        for (int i=1;i<=b*2;i++) for (int j=(a+b)*2+1-i+1;j<=(a+b)*2+1;j++) ch[(b+c)*2+1-b*2+i][j]='.';
        for (int i=1;i<=(b+c)*2+1;i++)
        {
            for (int j=1;j<=(a+b)*2+1;j++) printf("%c",ch[i][j]);
            printf("\n");
        }
    }
    return 0;
}
```

F

化简之后，就是要求 $\sum\left\lfloor\frac{a}{b}x\right\rfloor$，显然，$x$ 以 b 为一个周期，可以算出一个周期里面有多少个不一样的值，相同周期的值可以只处理一次，然后最后再暴力把剩余的算出来。

```
#include<bits/stdc++.h>
using namespace std;

typedef long long ll;
const ll mo=1e9+7;
```

```cpp

int main(){
    int t;
    cin>>t;
    while (t--)
    {
        ll a,b,n,tmp=0,pre=-1,sum=0;
        scanf("%lld%lld%lld",&a,&b,&n);
        if (a<=b)
        {
            ll c=n%mo*((n-1)%mo)%mo;
            if (c%2) c=(c+mo)/2;
            else c/=2;
            printf("%lld\n",(n%mo*a%mo+c)%mo);
            continue;
        }
        ll ans=n%mo*a%mo;
        for (ll i=0;i<b;i++)
        {
            ll c=a*i/b;
            if (c!=pre) pre=c,sum++,tmp+=c,tmp%=mo;
        }
        ans+=tmp*((n/sum)%mo)%mo;
        ans%=mo;
        ll tt=((n/sum)%mo*((n/sum-1)%mo))%mo;
        if (tt%2) tt=(tt+mo)/2;
        else tt/=2;
        ans+=tt*a%mo*b%mo;
        ans%=mo;
        pre=-1;
        for (ll cnt=0,i=0;cnt<(n%sum);i++)
        {
            ll c=i*a/b;
            if (c!=pre) pre=c,cnt++,ans+=c%mo+(n/sum)%mo*a%mo,ans%=mo;
        }
        printf("%lld\n",ans%mo);
    }
    return 0;
}
```

G

相当于图着色，使得相邻点颜色不同，问最少颜色数量。

~~显然图的最少着色数等于最大完全子图（最大团）的大小。~~ 哦豁，"显然" 错了。（考虑有$n>3,n$是奇数个点形成的一个环的图，最大团显然是2，但是最少着色数是3）（不过可以学习一下最大团的写法

随机化乱搞代码 (不是求最大团) (随机化求图着色，随机！随机！随机！不一定对！)：

```
#include<bits/stdc++.h>
using namespace std;

vector<int> a[105];
int r[105][105];
int flag[105];

int main(){
    int n,m,p,q;
    cin>>n>>m;
    for (int i=1;i<=n;i++) for (int j=1;j<=n;j++) r[i][j]=0;
    for (int i=1;i<=m;i++) cin>>p>>q,r[p][q]=r[q][p]=1;
    srand(time(0));
    int ans=0,f=0;
    while (1)
    {
        int ff=0,rem=0;
        for (int i=1;i<=n;i++) flag[i]=1;
        vector<int> a;
        a.clear();
        while (rem<n)
        {
            if (ff>500) break;
            int x=rand()%n+1;
            while (flag[x]==0) x=rand()%n+1;
            int fff=1;
            for (int k1:a) if (!r[k1][x]&&fff) fff=0;
            if (!fff)
            {
                ff++;
                continue;
            }
            a.push_back(x);
            flag[x]=0;
            rem++;
        }
        if (rem>ans) ans=rem,f=0;
        else f++;
        if (f>100) break;
    }
    printf("%d\n",ans);
    return 0;
}
```

H

只需要维护一颗 01Trie，Trie 的每个结点记录这个结点作为根的子树里的 c_i 的最小值，每次 `insert` 的时候去更新叶子节点到根的那一条链即可。

（数据比较随机，先排个序，然后 $O(n^2)$ 的暴力好像也能过。。。）

```
#include<bits/stdc++.h>
using namespace std;
#define int long long

const int inf = 0x3f3f3f3f3f3f3f3fLL;

class trie {
public:
    int tot;
    vector<int> mi;
    vector<vector<int> > ch;
    trie() : mi(1, inf), ch(1, vector<int> (2, -1)), tot(0) {}
    void insert(int x, int w) {
        int now = 0;
        for(int i = 30; i >= 0; i--) {
            int cur = (x >> i) & 1;
            if(ch[now][cur] == -1) {
                ch[now][cur] = ++tot;
                mi.push_back(inf);
                vector<int> tmp = {-1, -1};
                ch.push_back(tmp);
                now = tot;
            } else {
                now = ch[now][cur];
            }
            mi[now] = min(mi[now], w);
        }

    }

    int query(int x, int k) {
        int now = 0, ret = inf;
        for(int i = 30; i >= 0; i--) {
            int cur = (x >> i) & 1;
            int bitk = (k >> i) & 1;
            if(cur == 0 && bitk == 0) {
                if(ch[now][1] != -1) {
                    ret = min(ret, mi[ch[now][1]]);
                }
                if(ch[now][0] == -1) {
                    return ret;
                }
```

```
                now = ch[now][0];
            } else if(cur == 1 && bitk == 1) {
                if(ch[now][0] == -1) {
                    return ret;
                }
                now = ch[now][0];
            } else if(cur == 1 && bitk == 0) {
                if(ch[now][0] != -1) {
                    ret = min(ret, mi[ch[now][0]]);
                }
                if(ch[now][1] == -1) {
                    return ret;
                }
                now = ch[now][1];
            } else if(cur == 0 && bitk == 1) {
                if(ch[now][1] == -1) {
                    return ret;
                }
                now = ch[now][1];
            }
        }
        return ret;
    }
};

signed main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    int n, k; cin >> n >> k;
    vector<int> h(n + 1), c(n + 1);

    for(int i = 1; i <= n; i++) {
        cin >> h[i];
    }
    for(int i = 1; i <= n; i++) {
        cin >> c[i];
    }
    trie t;
    int ans = inf;
    for(int i = 1; i <= n; i++) {
        ans = min(ans, c[i] + t.query(h[i], k));
        t.insert(h[i], c[i]);
    }
    cout << ans;
    return 0;
}
```

I

三个点两两距离相等，那其中两个点一定到他们的最近公共祖先 lca 距离相等，另外一个点一定不在 lca 为根的子树里，并且到 lca 距离也相等。

可以考虑在 lca 的位置统计答案，但是需要知道 lca 上面有多少个点距离 lca 为 d，这个并不好统计。

于是考虑在 $LCA(x,y,z)$ （三个点的 LCA）处统计答案。

考虑在树上 dfs 时做动态规划，设 $g(i,j)$ 表示 i 这个点为根的子树中，有多少个无序二元对满足：“假设二元对的LCA是 lca，他们到 lca 的距离相等，并且 'i 到 lca 的距离' 与 '二元对到 lca 的距离' 之差为 j”。再设 $f(i,j)$ 表示 i 这个点作为根的子树中，离 i 的距离为 j 的点有多少个。

那么对于一个 $LCA(x,y,z)$ 显然，这个点对答案的贡献是：

$$g(i,0)+\sum_{x,y\in\text{son}(i),x\neq y} f(x,j-1)\times g(y,j+1)$$

dp 的转移也容易写出来了：

$$g(i,j)=\sum_{x,y\in\text{son}(i),x\neq y} f(x,j-1)\times f(y,j-1)$$
$$g(i,j)=\sum_{x\in\text{son}(i)} g(x,j+1)$$
$$f(i,j)=\sum_{x\in\text{son}(i)} f(x,j-1)$$

但是我们发现，这样不仅时间爆炸，甚至数组都开不下，于是考虑**长链剖分/DSU On Tree**优化 dp。（学习链接：[树链剖分 - OI Wiki (oi-wiki.org)]()）

我们为每一条重链开一个 dp 内存，那么在 f,g 转移的时候，重边可以 $O(1)$ 转移，轻边的 dp 值直接往重链上合并。一条轻边合并之后，对应的重链直接从图中删去，那么最后的复杂度就是 $O(\sum len_{heavy})=O(n)$。

具体细节可以看 std:

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

const int N = 1e5 + 8;

static int n, dep[N], son[N];
static long long answer, allocated[100 * N], *allo_head = allocated, *dp_g[N], *dp_f[N];
static std::vector<int> edges[N];

int solve_depths(int u, int f) // 处理每个点的深度和为每一个重链分配内存
```

```
{
    dep[u] = dep[f] + 1;
    son[u] = u;
    for (int v : edges[u])
        if (v != f && solve_depths(v, u) > dep[son[u]])
            son[u] = son[v];
    for (int v : edges[u])
        if (u == 1 || son[v] != son[u])
        {
            int size = dep[son[v]] - dep[u] + 8;
            dp_g[son[v]] = allo_head;
            allo_head += size * 3;
            dp_f[son[v]] = allo_head;
            allo_head += size * 1;
        }
    return dep[son[u]];
}

void solve_answer(int u, int f)
{
    for (int v : edges[u])
        if (v != f)
        {
            solve_answer(v, u);
            if (son[v] == son[u]) // 直接用指针做重边的转移
            {
                dp_g[u] = dp_g[v] + 1;
                dp_f[u] = dp_f[v] - 1;
            }
        }

    auto &gu = dp_g[u];
    auto &fu = dp_f[u];

    answer += gu[0];
    fu[0] = 1;
    for (int v : edges[u])
    {
        // 把轻边合并到重链上，一边合并，一边算答案
        if (v != f && son[v] != son[u])
        {
            int size = dep[son[v]] - dep[u];
            auto &gv = dp_g[v];
            auto &fv = dp_f[v];
            for (int i = 0; i < size; i++)
                answer += gv[i + 1] * fu[i] + gu[i + 1] * fv[i];
            for (int i = 0; i < size; i++)
            {
                gu[i] += gv[i + 1];
```

```
                gu[i + 1] += fv[i] * fu[i + 1];
                fu[i + 1] += fv[i];
            }
        }
    }
}

int main()
{
    scanf("%d", &n);
    for (int i = 0, u, v; i < n - 1; i++)
    {
        scanf("%d%d", &u, &v);
        edges[u].push_back(v);
        edges[v].push_back(u);
    }

    solve_depths(1, 0);
    solve_answer(1, 0);
    printf("%lld\n", answer);

    return 0;
}
```

J

假如我们用 $g(i,j)=f(i-j\pi)$，显然有 $g(i,j)=g(i-1,j)+g(i,j+1)$。

解法1（乱搞）：直接 $O(n^2)$ 求出所有的 $g(i,j)$，记得使用滚动数组压缩一维，否则空间不够。时间复杂度 $O(\frac{n^2}{2\pi})$。因为 $i-j\pi$ 是正数才有意义，所有第二层循环从 i/π 开始循环，这样可以让复杂度除以一个 π，这个非常重要，直接让复杂度从 $9e8$ 变为 $2e8$，刚好能卡过去。

解法2：考虑转移的式子，长得很像杨辉三角，实际上稍微推一下发现的确是组合数，直接 $O(n)$ 预处理组合数 $O(1)$ 得出答案。

（数据只出到 $3e4$ 是因为怕 $i-j\pi$ 精度不够）

乱搞：

```
#include<bits/stdc++.h>
#define int long long
using namespace std;

const double pi = acos(-1);
const int mod = 1e9 + 7;
const int N = 3e4+5,M=3e4+5;
int f[N],g[N],ans[N];

```

```
signed main() {
    int i,j,las;
    f[1]=f[2]=1;
    for(i=1;i<=3;i++)ans[i]=1;
    ans[4]=2;ans[5]=3;ans[6]=4;ans[7]=5;
    for(i=8;i<=M;i++){
        las=i/pi;
        for(j=las;j>=1;j--){
            if(i%2){
                if(i-j*pi<4)f[j]=1;
                else f[j]=f[j+1]+g[j];
                f[j] = f[j] >= mod ? f[j] - mod : f[j];
            }
            else{
                if(i-j*pi<4)g[j]=1;
                else g[j]=g[j+1]+f[j];
                g[j] = g[j] >= mod ? g[j] - mod : g[j];
            }
        }
//      cout<<ans[i-1]+g[1]<<'\n';
        if(i%2)ans[i]=ans[i-1]+f[1];
        else ans[i]=ans[i-1]+g[1];
        ans[i] = ans[i] >= mod ? ans[i] - mod : ans[i];
    }
    int t; cin >> t;
    while(t--) {
        int x; cin >> x;
        cout << ans[x] << '\n';
    }
    return 0;
}
```

正解std：

```
#include <bits/stdc++.h>
using namespace std;
typedef long long LL;
typedef unsigned long long ULL;
#define MEM(a,b) memset((a),(b),sizeof(a))
const LL INF = 1e9 + 7;
const int N = 1e5 + 10;
const long double pi = acos(0.0) * 2;
LL fact[N];
LL rfact[N];
void init()
{
    fact[0] = 1;
    for (int i = 1; i < N; i++) fact[i] = fact[i - 1] * i % INF;
    rfact[0] = rfact[1] = 1;
```

```
    for (int i = 2; i < N; i++) rfact[i] = (INF - INF / i) * rfact[INF % i] % INF;
    for (int i = 2; i < N; i++) rfact[i] = rfact[i - 1] * rfact[i] % INF;
}
LL C(int n, int m)
{
    return fact[n] * rfact[m] % INF * rfact[n - m] % INF;
}
int main(int argc, char** argv)
{
    init();
    int ncase;
    cin >> ncase;
    while (ncase--)
    {
        int n;
        cin >> n;
        if (n < 4)
        {
            puts("1");
            continue;
        }
        n -= 4;
        int i = 1;
        int j = n + pi;
        LL ans = 1;
        int cnt = 0;
        while (i <= int((n + pi) / pi))
        {
            cnt++;
            j = int(n + pi - i * pi);
            ans += C(i + j, i);
            i++;
        }
        cout << ans % INF << endl;

    }
    return 0;
}
```

K

考虑朴素的做法：从 x 到 y 肯定是先从 x 出发，一直不买，直到不能走为止，假如停在了 t。此时必须购买油才能向前走，此时我们不一定非要在 t 处买油，而是提前在 $x \sim t$ 之间油价最小的地方买油，买到刚好能走到下一个点就够了。

我们可以提前预处理出一个点 i 不买油的情况下，最远能走到的位置 j ，然后建一条 i 连向 $j+1$ 的边，其边权为 w （i 到 $j+1$ 不买油的情况下还需要**多少油**）。这个可以通过倒过来枚举点，做一个 $dp(i)=j$ 表示点 i 最远能到的点 j ，那么一开始令 $x=i$:

- 如果当前 $x=n, dp(i)=n$。
- 如果 x 能走到 $x+1$，那么将 x 扩展到 $dp(x)$，更新当前剩余油量。
- 如果 x 不能走到 $x+1$，那么 $dp(i)=x$。

根据摊还分析，这个过程的复杂度是 $O(n)$。由于 $i\sim j$ 里面都可以随意走，因此从 i 到 $j+1$ 可以使用 $i\sim j$ 里的最小价格。我们更新 $i\sim j$ 里的价格为 $p_{\min}$，**更新完后的数组**记为 p。

现在考虑如果走我们建的边 $u_1\to u_2\to\ldots\to u_k$ ，记前缀油价最小为 $prep_i=\min_{j\le i}\{p_{u_j}\}$，那么需要的油价就是 $\sum_{i=1}^{k-1} prep_i\times w_i$。

这个 min 直接导致了我们没办法在树上进行倍增。考虑消除 min，将相同的 $prep$ 合并在一起（这一步用单调栈容易实现）。

然后就可以快乐的倍增了。

std:

```
#include <bits/stdc++.h>
using namespace std;
typedef long long LL;
typedef unsigned long long ULL;
#define MEM(a,b) memset((a),(b),sizeof(a))
const LL INF = 1e9 + 7;
const int N = 2e5 + 10;
const int K = 100;
LL d[N];
int g[N];
LL sumg[N];
LL sum[N];
LL p[N];
int nexv[N];
int fa[20][N];
int fa2[20][N];
LL cost[20][N];
LL dp[K][K];
int n, q;
int main()
{
    scanf("%d%d", &n, &q);
    d[1] = 0;
    for (int i = 2; i <= n; i++)
    {
        scanf("%lld", &d[i]);
        d[i] += d[i - 1];
    }
```

```
    sumg[0] = 0;
    sum[0] = 0;
    for (int i = 1; i <= n; i++)
    {
        scanf("%d%lld", &g[i], &p[i]);
        sum[i] = sum[i - 1] + g[i];
    }
    p[n] = 0;
    d[n + 1] = INF*INF;
    nexv[n] = n;
    for (int i = n - 1; i > 0; i--)
    {
        nexv[i] = i;
        int o = i;
        int rest = 0;
        while (d[nexv[o] + 1] - d[o] <= rest + g[o])
        {
            p[i] = min(p[i], p[o]);
            rest = rest + g[o] - (d[nexv[o] + 1] - d[o]);
            o = nexv[o] + 1;
        }
        nexv[i] = nexv[o];
        g[i] = rest + g[o] + d[o] - d[i];
        p[i] = min(p[i], p[o]);
    }
    sumg[n] = 0;
    fa2[0][n] = n;
    for (int i = n - 1; i > 0; i--)
    {
        int f = nexv[i] + 1;
        f = min(f, n);
        fa2[0][i] = f;
        sumg[i] = sumg[f] + max(0LL, d[f] - d[i] - g[i]);
    }
    stack<int> s;
    for (int i = 1; i <= n; i++)
    {
        while (!s.empty())
        {
            int x = s.top();
            if (p[x] > p[i])
            {
                fa[0][x] = i;
                cost[0][x] = (sumg[x] - sumg[i])*p[x];
                s.pop();
            }
            else break;
        }
        s.push(i);
```

```
    }
    while (!s.empty())
    {
        int x = s.top();
        fa[0][x] = n;
        cost[0][x] = 0;
        s.pop();
    }
    for (int k = 0; k + 1 < 20; k++)
    {
        int o = 1 << k;
        for (int i = 1; i <= n; i++)
        {
            int f = fa[k][i];
            fa[k + 1][i] = fa[k][f];
            int f2 = fa2[k][i];
            fa2[k + 1][i] = fa2[k][f2];
            cost[k + 1][i] = cost[k][i] + cost[k][f];
        }
    }
    while (q--)
    {
        int x, y;
        scanf("%d%d", &x, &y);
        int o = 19;
        LL ans = 0;
        while (1)
        {
            if (o == -1) break;
            if (fa[o][x] <= y)
            {
                int f = fa[o][x];
                ans += cost[o][x];
                x = f;
                if (fa[0][f] > y) break;
            }
            o--;
        }
        LL blank = 0;
        o = 19;
        int t = y;
        LL pr = p[x];
        while (1)
        {
            if (o == -1) break;
            int f = fa2[o][x];
            if (nexv[f] >= y) t = min(y, f);
            else
            {
```

```
                blank += sumg[x] - sumg[f];
                x = f;
            }
            o--;
        }
        ans += max(0LL, (blank + sumg[x] - sumg[t])) * pr;
        printf("%lld\n", ans);
    }
    return 0;
}
```

L

通过观察发现 $L_1, L_2, \ldots, L_n$ 和 $L_{n+1}, L_{n+2}, \ldots, L_{n+m}$ 中的最大值必须相等，其他值完全随意。

如果最大值不相等，考虑最大值所在的那一个格子，这个格子会同时影响一列和一行，因此不可能最大值不相等。

其他值一定可以随便填：

对于行：假设列的最大值出现在第 $mxcol$ 列，考虑任意一行 row，L_{row} 填到第 $mxcol$ 列就可以了。

对于列：相似的，假设列的最大值出现在第 $mxrow$ 列，考虑任意一行 col，L_{n+col} 填到 $mxrow$ 行就可以了。

我们假设 $val = \max\{L_1, L_2, \ldots, L_{n+m}\}$ 则根据容斥，行和列分别都有："答案=最大值小于等于 val 的情况减去最大值严格小于 val 的情况"，根据乘法原理，两者相乘就是最大值为 val 时的答案，即 $ans_{val} = (val^n - (val-1)^n)(val^m - (val-1)^m)$

最后的答案就是：

$$\sum_{val=1}^{k} (val^n - (val-1)^n)(val^m - (val-1)^m)$$
$$= \sum_{val=1}^{k} val^{n+m} - val^n(val-1)^m - val^m(val-1)^n + (val-1)^{n+m}$$

对于上述式子的每一项，我们可以证明它是一个关于 k 的 $n+m+1$ 次多项式，我们需要 $n+m+2$ 个点即可确定这个多项式，因此可以先算出前 $n+m+2$ 项，然后**拉格朗日插值**计算最后的答案。

普通的**拉格朗日插值**是 $O(n^2)$ 的（多项式快速插值也需要 $O(n\log^2 n)$）显然不能通过，考虑**插值多项式** $p(x) = \sum_{i=1}^{n} y_i \sum_{i\neq j} \dfrac{x - x_j}{x_i - x_j}$，我们的点值是间隔为 1 的，因此可以预处理 $\sum_{i\neq j} x_i - x_j$ 的部分，仅需要 $O(n)$ 即可求出 $p(k)$。

复杂度：$O(Tn\log n)$ $\log n$ 来自快速幂。

题目时限比较紧，需要一些常数优化。

代码：

```cpp
#include<bits/stdc++.h>
using namespace std;
#define int long long

const int mod = 1e9 + 7;
const int N = 1e6 + 5;

int ksm(int a, int b, int MOD_KSM = mod) {
    int ret = 1;
    while(b) {
        if(b & 1) {
            ret = ret * a % MOD_KSM;
        }
        a = a * a % MOD_KSM;
        b >>= 1;
    }
    return ret;
}

int facinv[N], fac[N];
void init_fac() {
    fac[0] = fac[1] = 1;
    for(int i = 2; i < N; i++) {
        fac[i] = (long long) fac[i - 1] * i % mod;
    }
    facinv[N - 1] = ksm(fac[N - 1], mod - 2);
    for(int i = N - 2; i >= 0; i--) {
        facinv[i] = (long long) facinv[i + 1] * (i + 1) % mod;
    }
}

int LagrangeInterpolation(vector<int> &y, int l, int r, int n) {
    if(n <= r && n >= l) return y[n];
    vector<int> lg(r - l + 3), rg(r - l + 3);
    int ret = 0;
    lg[0] = 1;
    rg[r - l + 2] = 1;
    for(int i = l; i <= r; i++) {
        lg[i - l + 1] = (long long) lg[i - l] * (n - i) % mod;
    }
    for(int i = r; i >= l; i--) {
        rg[i - l + 1] = (long long) rg[i - l + 2] * (n - i) % mod;
    }
    for(int i = l; i <= r; i++) {
        if(r - i & 1) {
```

```
            ret = (ret - (long long) y[i] * lg[i - l] % mod * rg[i - l + 2] % mod * facinv[i - l] % mod * facinv[r - i] % mod + mod) % mod;
        } else {
            ret = (ret + (long long) y[i] * lg[i - l] % mod * rg[i - l + 2] % mod * facinv[i - l] % mod * facinv[r - i] % mod) % mod;
        }
    }
    return ret;
}
#define dec(a, b) a - b < 0 ? a - b + mod : a - b;
#define add(a, b) a + b >= mod ? a + b - mod : a + b;


signed main() {
    init_fac();
    int t; cin >> t;
    while(t--) {
        int n, m, k;
        cin >> n >> m >> k;
        vector<int> y(n + m + 3);
        for(int i = 1; i <= n + m + 2; i++) {
            int a = dec(ksm(i, n), ksm(i - 1, n));
            int b = dec(ksm(i, m), ksm(i - 1, m));
            y[i] = add(y[i - 1], a * b % mod);
        }
        cout << LagrangeInterpolation(y, 1, n + m + 2, k) << '\n';
    }
    return 0;
}
```