✓ 2023河南萌新联赛第（四）场：河南大学 解题报告

A - 汇编语言与接口技术

模拟，直接写即可

```cpp
#include <iostream>
#include <unordered_map>
#include <vector>
#include <stack>
using namespace std;
const int mod = 1 << 16;
struct Node
{
    string cmd, op1, op2;
};
void chose(char a);
stack<int> stk;
int cnt;
vector<Node> cmds;
unordered_map<string, int> mp = {{"ax", 0}, {"bx", 0}, {"cx", 0}, {"dx", 0}};
void a()
{
    string cmd, op1, op2;
    while (cin >> cmd)
    {
        Node node;
        op1 = "", op2 = "";
        if (cmd.size() == 1)
        {
            chose(cmd.front());
            break;
        }
        else if (cmd == "mov" or cmd == "add" or cmd == "sub")
        {
            cin >> op1 >> op2;
            node.cmd = cmd, node.op1 = op1, node.op2 = op2;
            cmds.push_back(node);
        }
        else
        {
            cin >> op1;
            node.cmd = cmd, node.op1 = op1, node.op2 = op2;
            cmds.push_back(node);
        }
    }
}
void d()
{
    string s;
    cin >> s;
    cout << mp[s] << endl;
}
void g()
```

```
{
    int v;
    cin >> v;
    for (int i = cnt; i < v; i++)
    {
        Node cmd = cmds[i];
        if (cmd.cmd == "mov")
        {
            if (cmd.op2.front() <= '9')
                mp[cmd.op1] = stoi(cmd.op2);
            else
                mp[cmd.op1] = mp[cmd.op2];
        }
        else if (cmd.cmd == "add")
        {
            if (cmd.op2.front() <= '9')
                mp[cmd.op1] += stoi(cmd.op2);
            else
                mp[cmd.op1] += mp[cmd.op2];
        }
        else if (cmd.cmd == "sub")
        {
            if (cmd.op2.front() <= '9')
                mp[cmd.op1] -= stoi(cmd.op2);
            else
                mp[cmd.op1] -= mp[cmd.op2];
        }
        else if (cmd.cmd == "pop")
        {
            mp[cmd.op1] = stk.top();
            stk.pop();
        }
        else if (cmd.cmd == "push")
        {
            if (cmd.op1.front() <= '9')
                stk.push(stoi(cmd.op1));
            else
                stk.push(mp[cmd.op1]);
        }
        else if (cmd.cmd == "mul")
        {
            if (cmd.op1.front() <= '9')
                mp["ax"] *= stoi(cmd.op1);
            else
                mp["ax"] *= mp[cmd.op1];
        }
        else if (cmd.cmd == "div")
        {
            if (cmd.op1.front() <= '9')
                mp["ax"] /= stoi(cmd.op1);
            else
                mp["ax"] /= mp[cmd.op1];
        }
        mp[cmd.op1] %= mod;
    }
    cnt = v;
}
void u()
```

```
{
    for (int i = cnt; i < cmds.size(); i++)
    {
        cout << cmds[i].cmd << " " << cmds[i].op1 << " " << cmds[i].op2 <<
endl;
    }
}
void r()
{
    string s;
    cin >> s;
    int val;
    cin >> val;
    mp[s] = val % mod;
}
void chose(char op)
{
    switch (op)
    {
    case 'a':
        a();
        break;
    case 'd':
        d();
        break;
    case 'g':
        g();
        break;
    case 'u':
        u();
        break;
    case 'r':
        r();
        break;
    default:
        break;
    }
}
void solve()
{
    char op;
    while (cin >> op)
    {
        chose(op);
    }
}
int main()
{
    solve();
    return 0;
}
```

B - 序列的与和

由于n不超过20，因此可以枚举所有情况，进行判断即可

```cpp
#include <iostream>
#define int long long
#define ull unsigned long long
#define rep(i, a, n) for (int i = (a); i <= (n); i++)
#define lowbit(x) ((x) & (-(x)))
using namespace std;
const int N = 30, mod = 1e9 + 7;
ull a[N];
void solve()
{
    int n, k;
    cin >> n >> k;
    rep(i, 1, n) cin >> a[i];
    int ans = 0;
    for(int op = 1; op < (1<<n); ++op)
    {
        ull val = 0xffffffffffffffff;
        rep(i, 1, n)
            if((op >> (i-1)) & 1) val &= a[i];
        int cnt = 0;
        while(val) val -= lowbit(val), cnt ++;
        if(cnt == k) ans ++;
    }
    cout << ans << '\n';
}
signed main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    solve();
    return 0;
}
```

C - 卡片翻转

出题 IDEA

一场 `ARC` 的 `B` 题改编而来

B - Grid Rotations

解题思路

不考虑操作二的情况下，考虑每个点在旋转之后的位置：

如果一个点满足 $X \leq a$

旋转之后有 $X = (a + 1) - X$

如果一个点满足 $X > a$

旋转之后有 $X = n + (a + 1) - X$

所以有 $X = (a - X + 1) \pmod{n}$

此处的模运算与常规的模运算稍微有点区别

此处的模运算操作的值域在 $[1, n]$ 之间， 所以代码上需要稍加处理

考虑操作二，其实就是相当于把卡片的 X 坐标轴与 Y 坐标轴交换，所以如果操作二的次数为奇数，处理操作一的时候就交换输入的 X, Y 就好了

最后输出的时候注意一下卡片有没有进行翻转即可

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

using i64 = long long;

int mod(i64 &x, int p) {
    x %= p; x += p; x %= p;
    return x;
}

int main() {
    int n, m, T;
    std::cin >> n >> m >> T;
    std::vector<std::vector<int> > a(n + 1, std::vector<int>(m + 1));
    std::vector<std::vector<int> > b(n + 1, std::vector<int>(m + 1));

    // Input
    for(int i = 0; i < n; ++ i)
        for(int j = 0; j < m; ++ j)
            std::cin >> a[i][j];

    // Operate
    i64 A = 0, B = 0, tot1 = 0, tot2 = 0;
    while (T --) {
        int op; std::cin >> op;
        if (op == 1) {
            ++ tot1; int cura, curb;
            std::cin >> cura >> curb;
            if (tot2 & 1)
                std::swap(cura, curb);
            A = cura - A - 1; mod(A, n);
            B = curb - B - 1; mod(B, m);
        } else {
            ++ tot2;
        }
    }

    // Resize
    for(int i = 0; i < n; ++ i)
        for(int j = 0; j < m; ++ j) {
```

```
            i64 nxtx = (tot1 & 1) ? (A - i) : (A + i);
            i64 nxty = (tot1 & 1) ? (B - j) : (B + j);
            mod(nxtx, n), mod(nxty, m);
            b[nxtx][nxty] = a[i][j];
        }

    // Output
    if (tot2 & 1) {
        for(int i = 0; i < m; ++ i)
            for(int j = 0; j < n; ++ j)
                std::cout << b[j][i] << " \n"[j == n - 1];
    } else {
        for(int i = 0; i < n; ++ i)
            for(int j = 0; j < m; ++ j)
                std::cout << b[i][j] << " \n"[j == m - 1];
    }
    return 0;
}
```

D - 幂运算

出题 IDEA

就是有天看离散数学书的时候想到的，感觉能和快速幂整合一下出出来

解题思路

其实易得

$$2^{2^n} = 2^{2^{2^{2^{\cdots}}}}$$

所以做 n 次平方运算就好啦

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

using i64 = long long;

int main() {
    i64 n, p; std::cin >> n >> p;
    i64 base = 2ll;
    while (n --)
        base = (base * base) % p;
    std::cout << base << "\n";
    return 0;
}
```

E - 平均数

出题 IDEA

就是出去聚餐的时候想到的，感觉写法很优雅，而且挺适合用来出签到题，就出了

解题思路

方差最小的方案就是使得所有的数字尽量靠近平均数

所以数字要么就是 $\lfloor \frac{S}{n} \rfloor$，要么就是 $\lceil \frac{S}{n} \rceil$

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

using i64 = long long;

int main() {
    i64 n, sum; std::cin >> n >> sum;
    for(i64 i = n; i; -- i) {
        std::cout << sum / i << " \n"[i == 1];
        sum -= sum / i;
    }
    return 0;
}
```

F - 小富的idea

对于任意一个墨滴，计算出它与其他所有墨滴的融合时间，并按时间从小到大排序，用并查集存储所有墨滴，然后从小到大枚举所有的融合时间，如果某个时间点发生两个墨滴融合，那么当前时间之后纸上墨滴数就减一，当然，如果融合的墨滴本身就在一个墨块里，总墨滴数不变标记出所有的墨滴减少的时间点，最后对于每次查询，输出当前墨滴数量即可

```cpp
#include <bits/stdc++.h>
using namespace std;
using ar3 = array<int, 3>;
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    int n;
    cin >> n;
    vector<ar3> e;
    vector<ar3> a(n);
    vector<int> fa(n);
    auto p2 = [](int x)
    {
        return x * x;
    };
    function<int(int x)> f = [&](int x)
    {
        return fa[x] == x ? x : fa[x] = f(fa[x]);
    };
    auto merge = [&](int x, int y)
```

```
    {
        int fx = f(x), fy = f(y);
        if (fx != fy)
            fa[fx] = fy, --n;
    };
    function<int(int, int)> dis = [&](int x, int y)
    {
        if(a[x][0] == a[y][0] and a[x][1] == a[y][1]) return 0;
        if (!a[x][2] and !a[y][2])
            return INT_MAX;
        int v = a[x][2] + a[y][2];
        return (int)ceil(sqrt(p2(a[x][0] - a[y][0]) + p2(a[x][1] - a[y][1])) /
v);
    };
    for (int i = 0; i < n; ++i)
        fa[i] = i;
    for (auto &[x, y, v] : a)
        cin >> x >> y >> v;
    for (int i = 0; i < n; ++i)
        for (int j = i + 1; j < n; ++j)
            e.push_back({dis(i, j), i, j});

    sort(e.begin(), e.end());
    int q;
    cin >> q;
    vector<int> ans(q);
    vector<pair<int, int>> que(q);
    for (int i = 0; i < q; ++i)
        cin >> que[i].first, que[i].second = i;
    sort(que.begin(), que.end());
    int now = 0;
    for (int i = 0; i < q; ++i)
    {
        while (now < (int)e.size() and e[now][0] <= que[i].first)
        {
            merge(e[now][1], e[now][2]);
            ++now;
        }
        ans[que[i].second] = n;
    }
    for (int i : ans)
        cout << i << '\n';
    return 0;
}
```

G - 继续来数数

考虑 n 个数都互不相同的情况，此时任意拿出一个子序列都不会发生重复，答案就是一个组合数：C_n^k。

考虑有 $n-1$ 个数互不相同的情况，那么会出现一种重复的情况：两个相同的数选其一，并且它们之间不再选其他数字。

比如：$[1,2,3,4,5,3,6]$。如果我们选出一个子序列 $[1,3]$ ，此时选的是第一个三还是第二个三就不确定，出现重复，如果是 $[1,3,4]$ 我们就可以定位到此时选了第一个三。

因此，在求出 C_n^k 总的方案数，减去重复情况：重复数字选其一，然后在除了重复数字之间的数中把剩下 $k-1$ 个选完，也就是 $C_{n-1-(pos_{second}-pos_{first})}^{k-1}$

需要注意要保证 $k-1 \leq n-1-pos_{second}+pos_{first}$ ，否则不可能出现重复（子序列太长，一定会选至少一个相同数之间的元素）。

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

const int MAXN = 10 + 1e5 ,mod=1e9 + 7;
int fac[MAXN],invfac[MAXN],inv[MAXN];
void init() {
    inv[1] = invfac[1] = invfac[0] = fac[1] = fac[0] = 1;
    for(int i = 2;i < MAXN;i ++) {
        inv[i] = (1ll * mod - mod / i) * inv[mod % i] % mod;
        fac[i] = 1ll * fac[i - 1] * i % mod;
        invfac[i] = 1ll * invfac[i - 1] * inv[i] % mod;
    }
}
int C(int n,int m) {
    if(m > n) return 0;
    return 1ll * fac[n] * invfac[m] % mod * invfac[n - m] % mod;
}
void solve()
{
    int n, k; cin >> n >> k;
    vector<int> a(n + 1);
    map<int,int> p;
    int dis = 0;
    for(int i = 1;i <= n;i ++) {
        cin >> a[i];
        if(p[a[i]]) {
            dis = i - p[a[i]];
        }else {
            p[a[i]] = i;
        }
    }
    if(!dis) {
        cout << C(n,k) << '\n';
    }else {
        cout << ((C(n, k) - C(n - dis - 1,k - 1)) % mod + mod) % mod << '\n';
    }
}
int main()
{
    ios::sync_with_stdio(false);cin.tie(0);cout.tie(0);

    init();

    int T;cin>>T;
    while(T--)
        solve();
```

```
    return 0;
}
```

H - 游戏高手

可以看出，每次进行战斗，都会使参加战斗的人的耐受减少，故想要赢需最后与人交战，再次发现，$k=2$比$k>2$时战斗后的玩家的耐受更低。（可以自己手动模拟一下）故应让玩家们两两先进行战斗。还有一个贪心的结论是，对于一堆玩家进行战斗，每次耐受值高的的两个玩家先进行战斗，会使得局面对于要取得胜利的玩家更优。因此贪心的策略已可以想出。再次观察，耐受值低的玩家能够存活，则耐受值高的玩家也必定能存活，因此本题可用二分求解，时间复杂度为$O(n\ log\ n)$

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

using namespace std;
typedef long long ll;
const ll N=2e5+10;

ll a[N],b[N],n;

bool ok(int mid)
{
    ll s=0;
    for(int i=n;i>=1;i--)
    {
        if(i==mid)continue;
        if(s==0)s+=a[i];
        else s=(s+a[i])/2;
    }
    if(s<=a[mid])return true;
    return false;
}

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0),cout.tie(0);
    cin>>n;
    for(int i=1;i<=n;i++)cin>>a[i],b[i]=a[i];
    sort(a+1,a+1+n);
    int l=1,r=n;
    while(l<r)
    {
        int mid=l+r>>1;
        if(ok(mid))r=mid;
        else l=mid+1;
    }
    for(int i=1;i<=n;i++)
    {
        if(b[i]>=a[r])
        cout<<1;
        else cout<<0;
```

```
    }
}
```

I - yh的线段

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

int n;
struct we{
    int l,r;
    bool operator <(const we &k)const{
        return r<k.r;
    }
}hh[1000005];
int main(){
    ios::sync_with_stdio(false);
    cin.tie(0);cout.tie(0);

    //对于重复相交的线段我们只用统计一次即可，因为可以删除任何数量的线段，
    //为了找出最大数量的好线段，我们可以按线段的右端点排序存储，然后遍历统计即可
    cin>>n;
    for(int i=1;i<=n;i++)cin>>hh[i].l>>hh[i].r;
    sort(hh+1,hh+1+n);
    int ans=0;
    int f=-1,l=-1;
    for(int i=1;i<=n;i++){
        if(hh[i].l<=f)continue;
        if(hh[i].l<=l){
            ans++;
            f=hh[i].r;
        }
        else l=hh[i].r;
    }
    cout<<ans<<"\n";
}
```

J - 异次元抓捕

题目背景

天使问题是由英国数学家约翰·何顿·康威提出的一个博弈论问题，在2006年已获解答。天使问题是关于一个叫天使与恶魔（Angels and Devils）的双人游戏，其规则如下：

1. 两名玩家分别扮演大使和恶魔
2. 游戏开始前，指定一个正整数 K，称之为天使的力量
3. 游戏在一个无限大的方格棋盘上进行，开始时棋盘是空的，天使留在棋盘上的某一个方格称为天使的起始点），恶魔并不存在于棋盘上
4. 每一轮中，恶魔可以在棋盘上放置一个路障，路障不可以放置在天使停留处
5. 每一轮中，天使可以向相邻格移动至多k步，移动过程中可以穿过路障，移动终点必须停留在没有路的格中，纵横斜格均算作相邻格

6．从恶魔开始，双方交替进行（若从天使开始，从上面的规则描述，亦可等价转换为从恶魔开始的局面）
7．若在一轮中，天使无法移动，则恶魔获胜
8．如果天使能够无限地继续游戏，则天使获胜

天使问题可以陈述为：是否存在某个K，使得力量为K的天使拥有必胜策略？

题解

只有$k=1$的时候输出`YES`

感兴趣的可以去查阅相关资料

K - 奖励关

不难发现获得最优积分只需要前两种操作，以操作1操作2交替的方式即可获得最优解，以这种方式两步即可得到一点积分

$$result=\left\lceil\frac{n}{2}\right\rceil$$

L - 7是大奖?

思路

经典数位DP。

问题就是让我们计算$[L,R]$中所有5的个数加上7的个数乘三加上存在连续7个7的数字个数乘300。

根据前缀和的思想，计算出$[1,R]$的答案-$[1,L-1]$的答案即为所求。

考虑记忆化搜索。定义数字数位最高为tot位，定义$f_{i,j,k}$表示考虑高$(i,tot]$已经安排完毕，而$[1,i]$不确定，并且此时无$limit$限制时，数字k已经出现j次时我们要求的数位k的数量之和。

这里$limit$限制表示，之后未确定的数字是否可以随便选，比如最大上限是98765，此时我们搜了前两个数：98???，如果我们选986??，那么显然之后?可以随意填，则限制解除，如果选987??，随意填就会超出最大上界。

当考虑`dfs(pos,num,limit,cnt)`的答案，它将继续搜索`dfs(pos-1,num,limit && i==up, cnt + (i==num))`这类子问题的答案，该问题已经被记忆化了，因此只需要在$O(tot*maxcnt)$的复杂度下即可解决该问题。

类似的，对于连续7个7，仍然考虑记忆化搜索，需要结合一点状态压缩思想，定义记忆化数组$g_{i,j,k}$表示在$[1,i]$仍未确定且无$limit$限制的情况下状态为j,k时的答案，具体状态定义如下：

- i：考虑低i位仍未确定
- j：二进制状态，共7位，表示当前已经确定的后7位是否填的数字7，如果为1，说明该位为7，否则不是7
- k：二进制状态，共1位，表示已经确定的数字中是否已经存在了连续的7个7。

std

```cpp
#include <bits/stdc++.h>
#define elif else if
#define SZ(x) (int)x.size()
#define rep(i,a,n) for(int i = (a);i <= (n);i++)
#define dec(i,n,a) for(int i = (n);i >= (a);i--)
#define inf 0x3f3f3f3f
using namespace std;
using ll = long long;
using PII = pair<int,int>;
template<class T> void print(T x){cout << x << '\n';}
template<typename T> void print(vector<T> &a){for(int i = 0;i < a.size();i ++) cout << a[i] << " \n"[i + 1 == a.size()];}
template <class Head, class... Tail> void print(Head&& head, Tail&&... tail) {cout << head << ' '; print(forward<Tail>(tail)...);}
mt19937 mrand(random_device{}());
int rnd(int x) { return mrand() % x;}
const int mod = 1e9 + 7;
ll tot, a[20];
ll f[20][20][10], g[20][1 << 7][2];
void get(ll x) {// 取得数字每一位
    memset(a, 0, sizeof a);
    tot = 0;
    while (x) {
        a[++tot] = x % 10;
        x /= 10;
    }
}
ll dfs(int pos, int num, bool limit, int cnt) {
    if (!pos) return cnt; // 如果所有数字都确定直接返回cnt
    if (~f[pos][cnt][num] and !limit) return f[pos][cnt][num]; // 如果在不受限下，已经搜过答案，直接返回
    int up = limit ? a[pos] : 9; // 判断是否限制解除，如果解除，可选数字设为9
    ll ans = 0;
    for (int i = 0; i <= up; i++) { // 搜索所有子问题
        ans += dfs(pos - 1, num, limit && i == up, cnt + (i == num));
    }
    if (!limit) f[pos][cnt][num] = ans; // 如果未受限，更新记忆化表
    return ans;
}
ll dfs2(int pos, int st, bool limit, bool have) { // st是已经确定的后7位是否为7，have表示已经确定数字中是否已经有7个7
    if (!pos) return have; // 返回是否用用
    if (~g[pos][st][have] and !limit) return g[pos][st][have];
    ll ans = 0;
```

```
    int up = limit ? a[pos] : 9;
    for (int i = 0; i <= up; i++) {
        int nst = ((st & ((1 << 7) - 1 ^ (1 << 6))) << 1) | (i == 7); // 位运算求next state
        ans += dfs2(pos - 1, nst, limit && i == up, have || nst == (1 << 7) - 1);
    }
    if (!limit) g[pos][st][have] = ans;
    return ans;
}
ll solve(ll n) {
    get(n);
    ll ans = 3 * dfs(tot, 7, true, 0) + dfs(tot, 5, true, 0);
    ans += 300 * dfs2(tot, 0, true, false);
    return ans;
}

signed main() {
    ios::sync_with_stdio(false);cin.tie(0);cout.tie(0);
    memset(f, -1, sizeof f); // 由于f，g都是在未受限情况下定义的，因此可以在多样例中重复使用
    memset(g, -1, sizeof g);
    int T; cin>>T;
    while(T--) {
        ll l, r; cin >> l >> r;
        print(solve(r) - solve(l - 1));
    }
    return 0;
}
```

M - 找孙子

出题 IDEA

原idea是找对于每个节点，孙子节点的个数的（就是一对爷爷-孙子节点贡献最大只能为1，不会因为中间父亲节点的不同而产生大于1的贡献）

然后不会 $O(n^2)$ 以内的做法，感觉不够优雅，就降低难度出了一个 $O(n)$ 的题目

解题思路

在新题面下，我们发现，只要记录每个节点的出度，然后对于每个爷爷节点，遍历他的儿子节点，因为图是一个有向无环图，所以他的儿子节点不会连到他自身，于是直接统计爷爷节点的所有儿子节点的出度和即可

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

const int M = 3e6 + 5;
const int N = 1e6 + 5;

int n, m, in[N], out[N], ans[N];

std::vector<int> a[N];
```

```cpp

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr), std::cout.tie(nullptr);
    std::cin >> n >> m;
    while (m --) {
        int u, v;
        std::cin >> u >> v;
        a[u].push_back(v);
        ++ out[u]; ++ in[v];
    }
    std::queue<int> q;
    for(int i = 1; i <= n; ++ i)
        if (in[i] == 0)
            q.push(i);
    while (!q.empty()) {
        int u = q.front(); q.pop();
        for(auto &v : a[u]) {
            ans[u] += out[v];
            if ((-- in[v]) == 0)
                q.push(v);
        }
    }
    for(int i = 1; i <= n; ++ i)
        std::cout << ans[i] << " ";
    std::cout << "\n";
    return 0;
}
```