A 石头剪刀布

纯签到，wa 一发的同学绝大多数错误是没想到三个同样的出拳也能平局。
参考代码：

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

int main() {
        int x, y;
        scanf("%d%d", &x, &y);
        printf("%d\n", x == y? x: 3 - x - y);
        return 0;
}
```

B 移除子串

没 A 那么纯的签到，想要找到一组操作满足题目条件，如果字符串第一个字符和最后一个字符不同，那么选择首尾一次操作就能清空字符串，否则需要找到一个 i 满足 $1 \le i \le n-2$ 使得 $S_0 = S_i$ 且 $S_{i+1} = S_{N-1}$，那么两次操作就能清空字符串。如果不能找到这样一个 i，那么容易证明无论如何最后必定会留下不能清除的字符。
参考代码：

```
#include <iostream>
using namespace std;

const int N = 100005;
int n;
char c[N];
int main() {
    cin >> n;
    cin >> c;
    if (c[0] != c[n - 1]) {
        cout << 1 << endl;
        return 0;
    }
    for (int i = 1; i <= n - 2; i ++ )
        if (c[i] != c[0] && c[i + 1] != c[0]) {
            cout << 2 << endl;
            return 0;
        }
    cout << "-1" << "\n";
    return 0;
}
```

C 均匀分布

只有对所有 (i,j) 满足 $(i+1,j)$ 和 $(i,j+1)$ 颜色相同时（当然最右边一列和最下面一行例外），才能满足题设要求的 “无论所走的路径如何，途径过的红色格子数

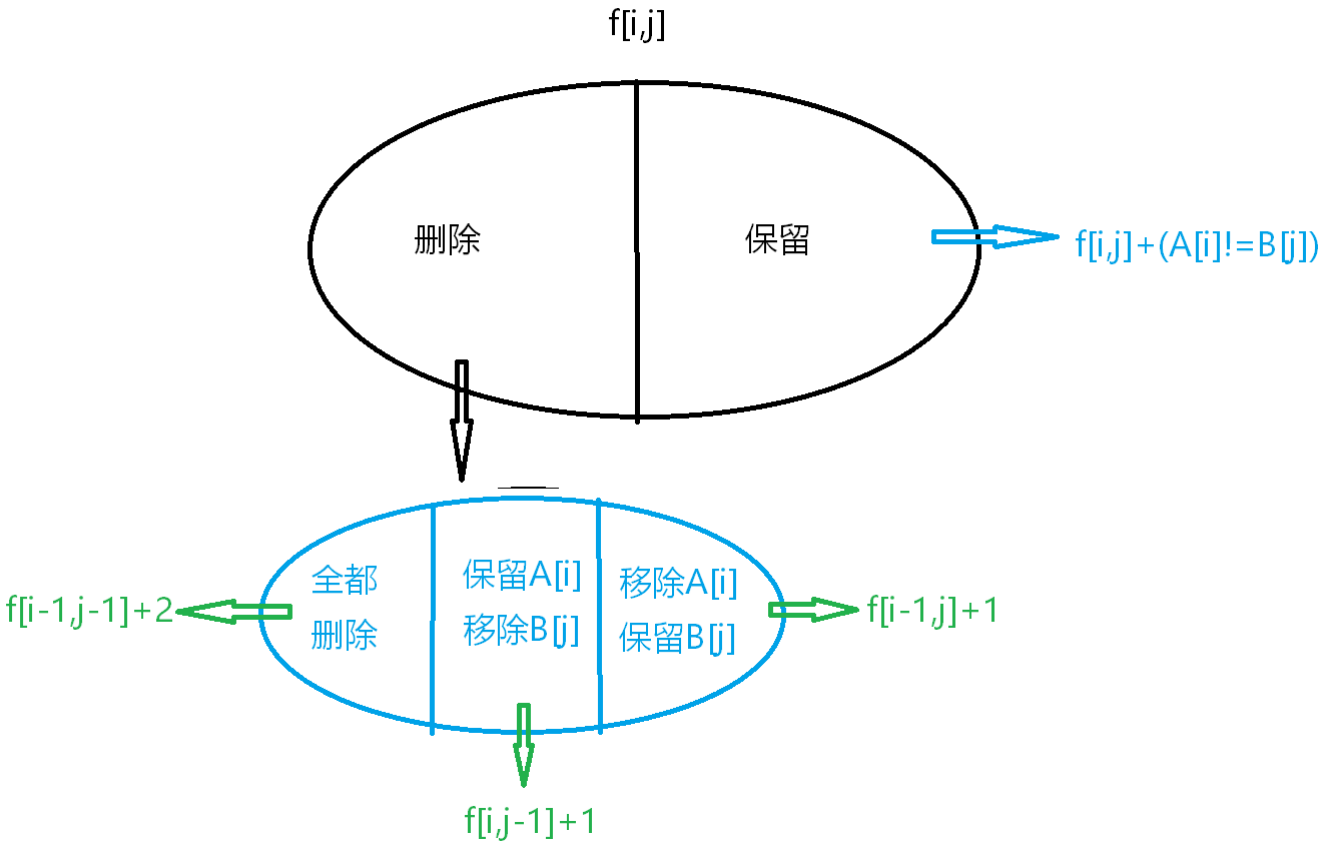
量都相同”。
所以对于每一个 $i+j$ 统计颜色分布，如果全是 `.` 答案乘 2，如果既有 `B` 又有 `W` 答案为 0，否则答案不变。
由于题目描述为：在把这些格子上色成红色或蓝色的 2^K 种方法中。因此没有 `.` 的情况被认为有 1 种方法，这在样例中有体现，并不是特判。
参考代码：

```
#include <iostream>
using namespace std;

const int M = 505;
const int MOD = 998244353;
int n,m,ans=1;char s[M][M];
int main()
{
    cin >> n >> m;
        for(int i=1;i<=n;i++)
        cin >> s[i] + 1;
        for(int i=2;i<=n+m;i++)
        {
                int c1=0,c2=0;
                for(int j=1;j<=n;j++)
                {
                        if(i-j<=0 || i-j>m) continue;
                        if(s[j][i-j]=='R') c1++;
                        if(s[j][i-j]=='B') c2++;
                }
                if(c1 && c2) ans=0;
                if(!c1 && !c2) ans=2*ans%MOD;
        }
    cout << ans << '\n';
    return 0;
}
```

D 序列匹配

容易想到的是搜索所有可能性，对于一个元素来说是保留还是直接删除，由于还有可能相同个数的情况下还需要删除元素才能得到最小值的情况，因此暴力求解的做法实现困难并且时间复杂度高。我们考虑使用动态规划的方式来计算。
定义数组 $dp[i][j]$：A 的前 i 个元素与 B 的前 j 个元素的 $x+y$ 的最小值。
现在对状态集合进行划分：



（在删除集合中，全都删除的集合被另外两个集合覆盖，可以不用进入比较。）
还需要考虑到边界情况，$f[i][j] = i + j$ $(i = 0 \ || \ j = 0)$。
参考代码：

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

const int N = 1010;
int a[N], b[N];
int dp[N][N];
int main() {
    int n, m;
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    for (int i = 1; i <= m; i++)
        cin >> b[i];
    for (int i = 0; i <= m; i++)
        dp[0][i] = i;
    for (int i = 0; i <= n; i++)
```

```
        dp[i][0] = i;

    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            dp[i][j] = min(dp[i - 1][j], dp[i][j - 1]) + 1;
            dp[i][j] = min(dp[i][j], dp[i - 1][j - 1] + (a[i] != b[j]));
        }
    }
    cout << dp[n][m] << "\n";
    return 0;
}
```

E 交换

交换两个数时，向左移动的数加上 1，向右移动的数减去 1。因此每个数的大小会和它的下标有一定关联。可以把每个数加上它自己的下标，这样就不需要在交换时增加或减少 1 了。
例如输入数据：

```
3
3 1 4
6 2 0
```

将 A 序列和 B 序列每个数都加上自己的下标：

```
3 2 6
6 3 2
```

会发现如果可以通过交换来满足 A 序列和 B 序列相同，那么处理后两个序列出现的数将会完全一样，接下来求出交换次数即可。
可以通过树状数组来维护每个数移动的距离。
参考代码：

```
#include <cstdio>
#include <iostream>
#include <set>
using namespace std;
const int M = 200005;
long long n, ans, bit[M];
set<pair<int,int>> s;
int lowbit(int x) {
    return x & (-x);
}
void add(int x, int f) {
    for (int i = x; i <= n; i += lowbit(i))
        bit[i] += f;
}
int ask(int x) {
    int r = 0;
    for (int i = x; i > 0; i -= lowbit(i))
        r += bit[i];
    return r;
}
```

```
int main() {
    cin >> n;
    for (int i = 1; i <= n; i++) {
        int x;
        cin >> x;
        s.insert(make_pair(x + i, i));
    }
    for (int i = 1; i <= n; i++) {
                int x;
                cin >> x;
        set<pair<int, int>>::iterator it = s.lower_bound(make_pair(x + i, 0));
        if (it == s.end() || (*it).first != x + i) {
            puts("-1");
            return 0;
        }
        int t = ask((*it).second);
        add(1, 1);
        add((*it).second, -1);
        ans += (*it).second + t - i;
        s.erase(it);
    }
    printf("%lld\n", ans);
    return 0;
}
```

F 好集合

简单题，放在后面是我排序失误了。
对于 1 个 1 位数，它显然会成为 17 个 2 位数的子序列（对于 0 只会成为 9 个 2 位数的子序列）。这就是说，如果可选范围内同时有 1 位数和 2 位数，将不会优先选择 1 位数。
同理，只要有更多位的数，应当优先选择位数多的数作为集合元素，而非位数少的。在最大的数中，如果首位不是 1，说明位数比它少的数都没有机会被选为集合元素。如果首位是 1，则还需要从位数少 1 位的数中寻找一些满足条件的作为元素。
参考代码：

```
#include <cstdio>
#include <algorithm>
using namespace std;

int lg[15];
int main() {
    int t;
    scanf("%d", &t);
    lg[1] = 1;
    for(int i = 2; i <= 12; ++ i)
        lg[i] = lg[i - 1] * 10;
    for(int i = 1; i <= t; ++ i) {
        long long l, r;
        scanf("%lld%lld", &l, &r);
        int x = l, y = r, s1 = 0, s2 = 0;
        while(x) s1 ++ , x /= 10;
        while(y) s2 ++ , y /= 10;
        if(s1 == s2) printf("%lld\n", r - l + 1);
```

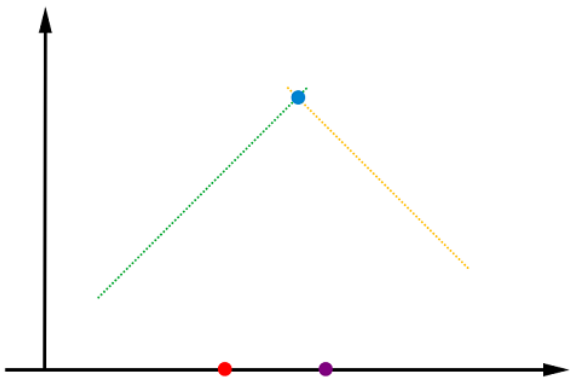
```
        else {
            if(r / lg[s2] == 1)
                printf("%lld\n", r - max(r % lg[s2], max(r / 10, l - 1)));
            else printf("%lld\n", r - lg[s2] + 1);
        }
    }
}
```

G 一维趴体

首先可以有如下观察：每个人时时刻刻都是在动的，i 一开始会一直往右边走直接碰到 $i+1$，然后一直往左走，反之亦然。
然后我们考虑如果 i 和 $i+1$ 是相向而行的，他们碰面了之后不反向，而是照着原来的方向继续行进，只是他们需要交换各自的任务。

如图，我们可以转化题意：横轴代表现在每个人的坐标，纵轴代表时间。那么我们钦定两个人同向走就能让中间的点都能满足条件（注意端点还是不满足条件的），我们称这个过程为**配对**，配对的过程中有如下限制：

- 每个点都必须被包含在配对点连成的三角形中。
- 每个点至多被配对一次。
- 相邻两个配对三角形直接必须有交。



那么可以用 dp 来配对，设 $f[i]$ 表示前 i 个点均合法的方案数，转移我们让 $i+1$ 和 $j-1$ 配对就可以得到区间 $[j,i]$ 的合法方案数，但是注意 $i=j$ 的转移会出问题，因为连续四个点让它们两两配对会导致不满足性质三，所以转移如下：

$$f[i] \leftarrow \max(f[j-1], a[i-1]-a[j-1]), j \in [1, i-1]$$

看上去是 $O(n^2)$ 的，其实 j 的取值范围很有限因为根本不可能配对得那么远，所以循环到 $j=i-3$ 即可。
参考代码：

```
#include <iostream>
using namespace std;

const int M = 200005;
int n, a[M], f[M];
int main() {
    cin >> n;
    for (int i = 1; i <= n; i++)
```

```
        cin >> a[i];
    a[0] = a[1];
    a[n + 1] = a[n];
    for (int i = 2; i <= n; i++) {
        f[i] = a[i + 1] - a[1];
        for (int j = max(i - 3, 1); j <= i - 1; j++)
            f[i] = min(f[i], max(f[j - 1], a[i + 1] - a[j - 1]));
    }
    cout << f[n] / 2 << '\n';
    return 0;
}
```

H 排序

如果一个排列每个位置上的逆序对个数都 $\leq k$，那么它是好排列。假设你有排列 P，每次可以交换两个相邻元素，用最小的步数得到好排列 P' 现给定 P' 和 k，求可能的 P 有多少个。

首先考虑知道了排列 P 怎么求出排列 P'，设第 i 个位置的逆序对个数是 x_i，因为一次交换最多让逆序对减一，所以答案下界是 $\sum_i \max(x_i - k, 0)$，构造方法是每次找到最小的位置 i 满足 $p_{i-1} > p_i$，并且 $x_i > k$，把这两个位置交换一下，可以证明如果排列不合法一定能找到这样的位置。

那么考虑用排列 P' 还原出可能的 P，首先我声明：如果我们知道了每个位置上的逆序对个数，那么对应的排列是唯一的。所以我们考察调整后的逆序对个数是否合法即可。

对于 $x_i < k$ 的位置是不能主动换的，因为交换让逆序对增加或减小都是不合法的，让他们的逆序对固定即可。对于 $x_i = k$ 的位置逆序对是可以任意增加的，但就是不能减小，所以如果我们按照 $i = n...1$ 的顺序考虑，我们可以把它右移任意步数，那么乘上 $(n - i + 1)$ 即可。
参考代码：

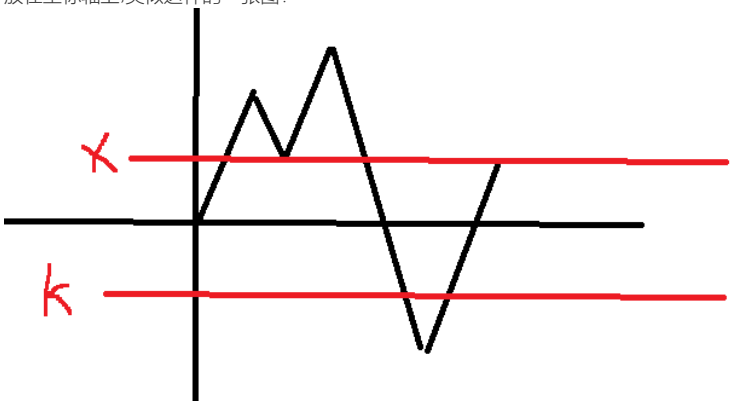
```
#include <iostream>
using namespace std;

const int M = 5005;
const long long MOD = 998244353;
long long n, k, ans = 1, p[M], x[M];
int main() {
    cin >> n >> k;
    for (int i = 1; i <= n; i++)
        cin >> p[i];
    for (int i = 1; i <= n; i++)
        for (int j = 1; j < i; j++)
            x[i] += (p[j] > p[i]);
    for (int i = n; i >= 1; i--)
        if (x[i] == k) ans = ans * (n - i + 1) % MOD;
    cout << ans << '\n';
    return 0;
}
```

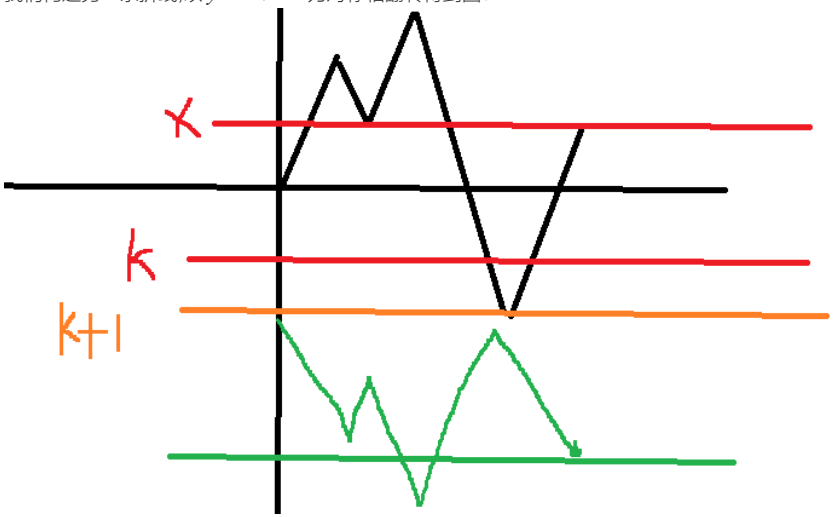
I 黑白球

考虑以 $(0, 0)$ 为起点,选白球相当于往右下走一步,黑球相当于向右上走一步.
白球 n 个,黑球 m 个,设 $x = m - n$ (先设 $m >= n$)。

放在坐标轴上,类似这样的一张图:



不管怎么走,最后终点都是 $(n+m,x)$，总的方案是 C_n^{n+m},我们需要知道不合法的方案是多少。
容易看出,只要上面的折线越过了 $y=-k$ 这根直线就不合法。
我们构造另一条折线,以 $y=-k-1$ 为对称轴翻转得到图:



发现翻转后折线的终点变为 $(n+m,2k+2+x)$，也就是不合法的方案数相当于向下走 z 步,满足 $z-(n+m-z)=2k+2+x$。
于是真正的方案数为 $C_n^{n+m}-C_z^{n+m}$。
当 $n<m$ 时同理。
参考代码:

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

const long long mod = 1e9 + 7;
const long long maxn = 2e6 + 10;
long long n, m, k, fac[maxn];
long long quick(long long x, long long n) {
    long long ans = 1;
    for ( ; n ; n >>= 1, x = x * x % mod )
        if ( n & 1 )   ans = ans * x % mod;
    return ans;
```

```
}
long long C(long long n, long long m) {
    if ( n < m )   return 0ll;
    return fac[n] * quick( fac[m] * fac[n - m] % mod, mod - 2 ) % mod;
}
int main() {
    fac[0] = 1;
    for (int i = 1; i <= 2000000; i++) fac[i] = fac[i - 1] * i % mod;
    cin >> n >> m >> k;
    if ( m + k < n ) {
        puts("0");
        return 0;
    }
    long long ans = 0;
    long long z = k + m + 1;
    ans = C(n + m, m) - C(n + m, z);
    cout << ( ans % mod + mod ) % mod;
    return 0;
}
```