牛客网NOIP赛前集训营-普及组&提高组（第一场）解题报告

PJ-A 绩点

给你两个数组 gpa 和 sc ，要你求出 $\frac{\sum_{i=1}^{n} gpa_i \times sc_i}{\sum_{i=1}^{n} sc_i}$ 。

签到题，直接模拟即可（甚至不需要数组）。

```
#include<cstdio>
#include<iostream>
using namespace std;
const int MAXN=55;
int n,sum1=0,sc;
double sum2=0,gpa;
int main(){
    scanf("%d",&n);
    for(int i=1;i<=n;++i){
        scanf("%lf%d",&gpa,&sc);
        sum1+=sc;sum2+=gpa*sc;
    }
    printf("%.1lf",sum2/sum1);
    return 0;
}
```

PJ-B 巨大的棋盘

给你一个棋盘，一开始你在一个位置，然后你会按照指令向上、下、左、右四个方向走，指令会重复若干次，有若干次询问，给出初始位置，求末位置（走到下边界从同一列上边界继续，其他方向与此相同）。

我们发现，向上走一次和向下走一次抵消，向左走一次和向右走一次抵消，因此，我们可以分别用 sum_1 和 sum_2 表示执行完一遍指令后，我们总共向下和向右走了多少格，重复 k 次则乘以 k 即可，超过边界就取模，算出负数的话先把他变成正数再取模。

最后，还想说一句，**注意** $longlong$ ，一开始因为没有注意丢了 60 分。。。

```
#include<cstdio>
#include<cstring>
#include<iostream>
using namespace std;
typedef long long int ll;
const int MAXN=1e5+5;
char s[MAXN];
ll n,m,T,q,x,y,len;
ll sum1=0,sum2=0;
ll read(){
    ll x=0,f=1;char c=getchar();
    for(;c<'0'||c>'9';c=getchar()) if(c=='-') f=-1;
    for(;c>='0'&&c<='9';c=getchar()) x=(x<<3)+(x<<1)+c-'0';
```

```
    return x*f;
}
int main(){
    n=read();m=read();T=read();
    scanf("%s",s+1);len=strlen(s+1);
    for(int i=1;i<=len;++i){
        if(s[i]=='U') --sum1;
        if(s[i]=='D') ++sum1;
        if(s[i]=='L') --sum2;
        if(s[i]=='R') ++sum2;
    }
    while(sum1<0) sum1+=n;sum1=(sum1*T)%n;
    while(sum2<0) sum2+=m;sum2=(sum2*T)%m;
    q=read();
    while(q--){
        x=read()+sum1;y=read()+sum2;
        if(x>n) x%=n;
        if(y>m) y%=m;
        printf("%lld %lld\n",x,y);
    }
    return 0;
}
```

PJ-C 括号

给你一个括号序列，你可以任意删除里面的左括号和右括号，但不能将这个序列删空，问你有多少种方案，使得剩下的括号序列里的括号是匹配的。

想到我们在判断一个括号序列匹不匹配的时候对栈的使用方法，我们发现序列合不合法与已判断部分有多少个左括号有关，于是我们可以通过 dp 来解决这道题。

我们假设 $dp[i][j]$ 表示判断到了第 i 位，前面有 j 个左括号没有匹配的方案数。

因为我们每考虑一位，没有匹配的左括号数就会增加一（增加一个左括号）或者减少一（增加一个右括号）。

因此，我们有:

$$dp[i][j]=\begin{cases} dp[i][j]+dp[i-1][j-1]+dp[i-1][j] & \text{括号序列的第 } i \text{ 位为左括号} \\ dp[i][j]+dp[i-1][j+1]+dp[i-1][j] & \text{括号序列的第 } i \text{ 位为右括号} \end{cases}$$

最后面加上 $dp[i-1][j]$ 是考虑到第 i 位的括号可以不选。

然后初始条件为 $dp[0][0]=1$ ，答案为 $\sum_{i=1}^{n} dp[i][0]$ 。

考虑到 $n \le 10000$ ，若果直接开一个二维数组空间会爆掉，并且我们在转移的时候永远只会从 i 转移到 $i+1$ ，所以我们可以删去 i 这一维，使用滚动数组来优化空间，但是，这个时候就导致，我们最终算出的 $dp[0]$ 还包含了原数组中的 $dp[0][0]$ ，因为不能删空，所以最终答案要减一。

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

```
const int mod=1e9+7;
const int MAXN=1e4+5;
char s[MAXN];
int n,now=1;
int dp[2][MAXN];
int main(){
    scanf("%d%s",&n,s+1);
    dp[0][0]=1;
    for(int i=1;i<=n;++i,now^=1){
        memset(dp[now],0,sizeof(dp[now]));
        for(int j=0;j<=i;++j){
            dp[now][j]=(dp[now][j]+dp[now^1][j])%mod;
            if(s[i]=='('&&j) dp[now][j]=(dp[now][j]+dp[now^1][j-1])%mod;
            if(s[i]==')') dp[now][j]=(dp[now][j]+dp[now^1][j+1])%mod;
        }
    }
    printf("%d",(dp[now^1][0]-1)%mod);
    return 0;
}
```

PJ-D 配对

G-A　中位数

给你一个序列，让你找出所有长度大于等于 len 的子序列中，中位数最大的那个子序列，并求出这个中位数。

补一个套路：假设我们要找一个序列中最大的中位数，那么这个答案是满足二分性质的，因为我们每往一个序列中加入一个数或者删除一个数，最多会导致子序列的中位数在排完了序的序列中移动一位，所以它会存在一个最大值和一个最小值，于是，我们就可以二分中位数，假设枚举的中位数为 key ，那么，我们可以将原序列中所有的小于 key 的数变成 -1 ，将所有等于 key 的数变成 0 ，将所有大于 key 的数变成 1 ，然后，我们便只要判断这个新序列存不存在一个长度大于等于 lem 的子序列，使得所有元素的和大于 0 ，而这明显可以通过前缀和加上一遍扫描找到，总复杂度 $O(nlogn)$ 。

```
#include<cstdio>
#include<cstdlib>
#include<iostream>
#include<algorithm>
using namespace std;
const int MAXN=1e5+5;
int n,len,ans=0;
int arr[MAXN],cparr[MAXN];
int change[MAXN],sum[MAXN];
bool judge(int key){
    for(int i=1;i<=n;++i){
        if(arr[i]<key) change[i]=-1;
        if(arr[i]==key) change[i]=0;
        if(arr[i]>key) change[i]=1;
        sum[i]=sum[i-1]+change[i];
    }
    int minsum=0;
    for(int i=len;i<=n;++i){
        minsum=min(minsum,sum[i-len]);
```

```
        if(sum[i]>=minsum) return true;
    }
    return false;
}
int read(){
    int x=0,f=1;char c=getchar();
    for(;c<'0'||c>'9';c=getchar()) if(c=='-') f=-1;
    for(;c>='0'&&c<='9';c=getchar()) x=(x<<3)+(x<<1)+c-'0';
    return x*f;
}
int main(){
    n=read();len=read();
    for(int i=1;i<=n;++i) cparr[i]=arr[i]=read();
    sort(cparr+1,cparr+n+1);
    int l=1,r=n;
    while(l<=r){
        int mid=(l+r)>>1;
        if(judge(cparr[mid])) {ans=mid;l=mid+1;}
        else r=mid-1;
    }
    printf("%d",cparr[ans]);
    return 0;
}
```

TG-B　数数字

TG-C 保护