A 罚时

参考代码：

```
#include<iostream>
using namespace std;
int s[15];
int x[15];
int main(){
    int T,sum=0;
    cin>>T;
    for(int i=1;i<=T;i++){
         cin>>s[i];
         sum+=s[i];
    }
    for(int i=1;i<=T;i++){
        cin>>x[i];
        if(s[i]!=0){
            sum+=(x[i]-1)*20;
        }
    }
    cout<<sum<<endl;
}
```

B 实力等级

参考代码：

```
#include<iostream>
using namespace std;
int main(){
    int T;
    cin>>T;
    while(T--){
        int n;
        cin>>n;
        if(n==10) cout<<"SSS"<<endl;
        else if(n>=8&&n<=9) cout<<"SS"<<endl;
        else if(n>=5&&n<=7) cout<<"S"<<endl;
        else if(n>=1&&n<=4) cout<<"A"<<endl;
        else cout<<"B"<<endl;
    }
}
```

C 盲蛋

法一：暴力（O(n)）

首先肯定要排序的。排完序之后可以发现，从中位数开始，往后直接一个一个累加，并且一旦中位数和后面的数字相等了，就得一起同时累加，直到k无法满足将相等的数都加1为止。此时中位数即为最大中位数。（注意边界是2e9）

参考代码：

```
#include <iostream>
#include <algorithm>
using namespace std;
const int N = 2e5+10;
int n, k;
int a[N];
int main(){
    cin >> n >> k;
    for(int i = 0; i < n; i++) cin >> a[i];
    sort(a, a + n);
    int p = n / 2;
    a[n] = 2e9+10;
    for(int i = p + 1; i <= n && k; i++){
        if(i - p <= k) {
            int t = a[i] - a[p];
            a[p] += min(t, k / (i - p)), k -= (i - p) * t;
        }
        else k = 0;
    }
    cout << a[p] << endl;
    return 0;
}
```

法二：二分（nlogk）

同样，我们只需要对排完后，中位数及其后面数进行操作。可以二分答案mid，将后半段小于mid的全部加到mid，然后判断操作是否小于k。二分出满足条件的最大值，即为最终答案。

```
#include <iostream>
#include <algorithm>
using namespace std;
typedef long long LL;
const int N = 2e5+10;
int n, k;
int a[N];
bool check(int p){
    LL sum = 0;
    for(int i = n / 2; i < n; i++)
        if(a[i] < p) sum += p - a[i];
    if(sum > k) return false;
    else return true;
}
int main(){
    cin >> n >> k;
    for(int i = 0; i < n; i++) cin >> a[i];
    sort(a, a + n);
    int l = 0, r = 2e9;
    while(l < r){
        int mid = (LL)l + r + 1 >> 1;
```

```
        if(check(mid)) l = mid;
        else r = mid - 1;
    }
    cout << l << endl;
    return 0;
}
```

D 然然的烦恼

提炼出题意：求长度为**n**的数组有多少个连续区间的和是**k**的整数倍。

第一步当然是求出前缀和数组`sum[]`

如果区间**[i, j]**满足条件，则`(sum[j] - sum[i - 1]) % k = 0`，

即：`sum[j] % k == sum[i - 1] % k`。

由此可得，所有模**k**相等的两个端点即可构成一个**k**倍区间

参考代码：

```
#include <iostream>
#include <algorithm>
#include <cstring>
#include <cstdio>
#include <map>
using namespace std;
const int N = 1e5+10;
int n, k;
long long ans, s[N];
map<int, long long> nums;
int main(){
    cin >> n >> k;
    for(int i = 1; i <= n; i++){
        cin >> s[i];
        s[i] = (s[i - 1] + s[i]);
    }
    // s[0] = 0,求区间和会涉及到第一个的前一个元素
    nums[0] = 1;
    for(int i = 1; i <= n; i++){
        s[i] = s[i] % k;
        nums[s[i]]++;
    }
    for(auto i : nums)
        ans += i.second * (i.second - 1) / 2;
    cout << ans << endl;
    return 0;
}
```

考虑一种线性的做法（O(n)）：

用`cnt[]`数组记录当前已经出现的模**k**的值的个数，遍历到下标为**i**的数时，计算其前缀和模**k**的值`s[i] = (s[i - 1] + s[i]) % k`,它可以和已经出现的相同值构成**k**倍区间，所以更新答案：`ans += cnt[s[i]]`，更新数量：`cnt[s[i]]++`。

参考代码：

```
#include <iostream>
#include <algorithm>
#include <cstring>
#include <cstdio>
using namespace std;
const int N = 1e5+10;
int n, k;
long long ans, s[N];
//cnt[] : 记录当前已经出现的模k的值的次数
int cnt[N];
int main(){
    cin >> n >> k;
    // s[0] = 0,求区间和会涉及到第一个的前一个元素
    cnt[0] = 1;
    for(int i = 1; i <= n; i++){
        cin >> s[i];
        //计算当前点的模k值
        s[i] = (s[i - 1] + s[i]) % k;
        //加上已经出现的次数（即可以构成的k倍区间数量）
        ans += cnt[s[i]];
        //当前值的数量+1
        cnt[s[i]]++;
    }
    cout << ans << endl;
    return 0;
}
```

E 人生重开模拟器

玩家需要从n个属性中选出x个天赋，只需要选出x个获得属性点最大的天赋即可，这样可分配的属性点才是最多的。

接下来是完全背包问题，dp[i][j]表示前i个属性分配j个属性点时获取的能力值最大，推算出状态转移方程为`dp[i][j]=max(dp[i][j],dp[i-1][j-k]+c[i][k])`(k表示为第i个属性分配了k个属性点)。

参考代码：

```
#include<iostream>
#include<cstring>
#include<algorithm>
using namespace std;
const int N=105;
int dp[N][N],c[N][N],v[N];
int main(){
    int t;
    cin>>t;
    while(t--){
        int n,x,m;
        memset(v, 0, sizeof v);
        memset(c, 0, sizeof c);
        memset(dp, 0, sizeof dp);
        cin>>n>>x>>m;
```

```
        for(int i=0;i<n;i++) cin>>v[i];
        sort(v,v+n);
        int sum=0;
        for(int i=n-1;i>=n-x;i--) sum+=v[i];
        for(int i=1;i<=m;i++){
            for(int j=1;j<=sum;j++){
                cin>>c[i][j];
            }
        }
        for(int i=1;i<=m;i++){
            for(int j=1;j<=sum;j++){
                for(int k=0;k<=j;k++){
                    if(dp[i][j]<dp[i-1][j-k]+c[i][k]){
                        dp[i][j]=dp[i-1][j-k]+c[i][k];
                    }
                }
            }
        }
        cout<<dp[m][sum]<<endl;
    }
}
```

F 城市

并查集板子题

参考代码：

```
#include <iostream>
using namespace std;
const int N = 100010;
int f[N];
int n, m;
int find(int u)
{
    if(u != f[u])   f[u] = find(f[u]);
    return f[u];
}
int main()
{
    cin >> n >> m;
    for(int i = 1; i <= n; i++) f[i] = i;
    while(m--)
    {
        char c;
        int a, b;
        cin >> c >> a >> b;
        if(c == 'M')
        {
            int fa = find(a), fb = find(b);
            if(fa != fb)
                f[fa] = fb;
        }
        else if(c == 'Q')
        {
```

```
            int fa = find(a), fb = find(b);
            if(fa == fb)
                printf("Yes\n");
            else
                printf("No\n");
        }
    }
}
```

G 子字符串翻转

首先分析一下样例3，原字符串为`abcdef`，三次翻转，第一次将整个字符串全部翻转，此时的字符串为`fedcba`，第二次将`edcb`翻转为`bcde`，此时的字符串为`fbcdea`，第三次将`cd`翻转为`dc`，此时的字符串为`fbdcea`。

显而易见，第一次翻转把第1~6位置的字符串翻转了，第二次翻转把2~5位置的字符串翻转了，那前两次相当于只有第1和第6位置的字符互换了位置，第三次把3~4位置的字符翻转了，两个位置的字符翻转了3次，和只翻转1次的效果一样，由此可得，每一对字符是否互换位置，取决于这两个字符之间的字符串有几次需要被翻转，被翻转偶数次的话，则不需要互换。

参考代码：

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
   string s;
   int n,f,a[100005]={};
   cin>>s>>n;
   for(int i=0;i<n;i++){
       cin>>f;
       a[f]++;
   }
   for(int i=1;i<=s.size()/2;i++){
        a[i]+=a[i-1];
        if(a[i]%2==1){
            swap(s[i-1],s[s.size()-i]);
        }
        cout<<s<<endl;
   }
   cout<<s;
   return 0;
}
```

H 好看的序列

暴力枚举,复杂度为$O(n^2)$由题目可以知道必须先算出前n-1个才能算出第n个，所以第一步算出前n-1个
对于前n-1个每次都calc一下就行
calc的思想：
利用双指针每次找数字相同的一段，同时记录个数即可

参考代码:

```
#include<iostream>
#include<algorithm>
#include<cstring>
using namespace std;
string calc(string tmp)
{
    string res;
    for(int i=0;i<tmp.size();)
    {
        int j=i;
        while(j<tmp.size()&&tmp[j]==tmp[i]) j++;
        int cnt=j-i;
        res+=to_string(cnt)+tmp[i];
        i=j;
    }
    return res;
}
string dfs(int n)
{
    if(n==1) return "1";
    string tmp=dfs(n-1);
    string res=calc(tmp);
    return res;
}
int main()
{
    string res;
    int n;
    cin>>n;
    cout<<dfs(n)<<endl;
}
```

I 失去你，或者又如何?

在公主那里跑一遍bfs，然后求出公主与小碘和公主与边界的最短路，两者相加即可。

参考代码:

```
#include <bits/stdc++.h>
using namespace std;
const int M = 1e3+5;
char a[M][M];
int vis[M][M];
int n, m;
struct node{
    int x, y, s;
};
queue<node> q;
int sx, sy, ex, ey;
int ne[4][2] = {{1, 0}, {0, 1}, {-1, 0}, {0, -1}};
int ans1, ans2, flag;
void bfs(){
```

```
    q.push({sx, sy, 0});
    while(q.size()){
        node t = q.front(); q.pop();
        if(vis[t.x][t.y]) continue;
        vis[t.x][t.y] = 1;
        if(t.x == ex && t.y == ey) ans1 = t.s;
        for(int i=0; i<4; i++){
            int tx = t.x + ne[i][0];
            int ty = t.y + ne[i][1];
            if(1<=tx && tx<=n && 1<=ty && ty<=m && a[tx][ty] !='$' && a[tx][ty]
!= '&' && !vis[tx][ty]){
                q.push({tx, ty, t.s+1});
            }
            if(tx<1 || n<tx || ty<1 || m<ty)
                if(!flag){
                    flag = 1;
                    ans2 = t.s + 1;
                }
        }
    }
}
int main(){
    cin>>n>>m;
    for(int i=1; i<=n; i++){
        for(int j=1; j<=m; j++){
            scanf(" %c", a[i]+j);
            if(a[i][j] == 'D') ex = i, ey = j;
            if(a[i][j] == 'P') sx = i, sy = j;
        }
    }
    bfs();
    if(ans1 && ans2){
        printf("You woke up at %d seconds", ans1+ans2);
    }
    else
        puts("What is life without you");
    return 0;
}
```

J 字符串转换

数字的每三位处理的方法是一样的,即将这三位数拆分再根据题意组合成字符串作为这个三位数的处理结果，用一个字符串`res`来拼接最终结果，如果第1~3位上面有数值，则将这三位的处理结果直接拼接到`res`上，如果4~6位上有数值，则需要在这三位处理结果后面拼接一个`Thousand`再接到res后面，第7~9位和第10位的处理方法也一样。

参考代码：

```
#include<iostream>
#include<cstring>
using namespace std;
string s1[20]=
{"Zero","One","Two","Three","Four","Five","Six","Seven","Eight","Nine","Ten","El
even","Twelve","Thir"};
```

```
string s2[20]={"","","Twenty","Thir","Four","Fif","Six","Seven","Eigh","Nine"};
string s3[20]=
{"","one","two","three","four","five","six","seven","eight","nine","ten","eleven
","twelve","thir"};
string calc(int x)
{
    if(!x) return "";
    int bai=x/100;
    int shi=x/10%10;
    int ge=x%10;
    string res;
    if(bai)//三位数
    {
        res+=s1[bai]+" Hundred";
    }
    if(shi)
    {
        res+=" ";
        if(shi>=3)
        {
            if(shi==4) res+="For";
            else res+=s2[shi];
            res+="ty";
            if(ge)res+=" "+s1[ge];
        }
        else if(shi==2)
        {
            res+=s2[shi];
            if(ge)res+=" "+s1[ge];
        }
        else if(shi==1)
        {
            if(ge>2)
            {
                res+=s2[ge]+"teen";
            }
            else if(ge<=2)
            {
                res+=s1[ge+10];
            }
        }
    }
    if(!shi&&ge)
    {
        res+=" ";
        res+=s1[ge];
    }
    while(res.size()&&res[0]==' ') res=res.substr(1);
    return res;
}
int main()
{
    int num;
    cin>>num;
    string res;
    int x1=num/((int)1e9);
    string t1=calc(x1);
    if(x1) res=t1+" Billion ";
```

```
    x1=num/(int)(1e6)%1000;
    if(x1) res+=calc(x1)+" Million ";
    x1=num/(int)(1e3)%1000;
    if(x1) res+=calc(x1)+" Thousand ";
    x1=num%1000;
    if(x1) res+=calc(x1);
    if(res.size()==0) res="Zero";
    while(res.size()&&res[0]==' ') res=res.substr(1);
    while(res.size()&&res.back()==' ') res.pop_back();
    cout<<res<<endl;
}
```