A、你看这只兔子 HDU1406

> 注意num1,num2有可能num1>num2,不然提交是错的，大水题，按照题意模拟即可

```
#include<iostream>

using namespace std;

int main() {
    int k,i,T;
    cin>>T;
    while(T--){
        int num1,num2;
        cin>>num1>>num2;
        num1=min(num1,num2);
        num2=max(num1,num2);
        int count=0,p;
        for(p=num1; p<=num2; p++) {
            int j,sum = 0;
            for(j=1; j<=p/2; j++) {
                if(p%j == 0) {
                    sum += j;
                }
            }
            if(sum == p)
                count ++;
        }
        cout<<count<<endl;
    }
    return 0;
}
```

B、不如做成HDU1009

> 简单的分数背包问题，要使结果最优只需是每份前可以买到的串数最多即可

```
#include <iostream>
#include <algorithm>
#include <cstdio>
using namespace std;
struct node {
    double w;
    double v;
    double k;
};
int cmp(node c, node d) {
    return c.k > d.k;
}
int main() {
    int i, j, n;
    double m;
    node a[10005];
    double sum;
    while ((scanf("%lf%d", &m, &n) == 2) && ((n != -1) || (m != -1))) {
        sum = 0;
```

```
        for (i = 0; i < n; i++) {
            scanf("%lf%lf", &a[i].w, &a[i].v);
            a[i].k = a[i].w * 1.0 / a[i].v;
        }
        sort(a, a + n, cmp);
        j = 0;
        while (1) {
            if (j == n)
                break;
            if (m >= a[j].v) {
                sum = sum + a[j].w;
                m = m - a[j].v;
                j++;
            } else {
                sum = sum + a[j].w * 1.0 / a[j].v * m;
                break;
            }
        }
        printf("%.3f\n", sum);
    }
    return 0;
}
```

C 、兔子这么可爱为啥丢掉呀 cf122A

> 方法一：找到所有符合的数打个表然后保存，直接判断在不在表里
>
> 方法二：遍历一个数所有的因子一旦找到不包含除4、7外其他数字的因子弹出并输出YES，如果遍历完也无法找到，输出NO即可

```
#include<iostream>
#include <algorithm>
#include <iomanip>
#include<cmath>
#include<cstring>
using namespace std;
bool check(int a){
    while (a>0){
        if(not (a%10==7||a%10==4))return false;
        a=a/10;
    }
    return true;
}
int main(){
    int a;
    int x,y;
    cin>>a;
    for(int i=1;i<a/2+1;i++){
        if(a%i==0){
            x=a/i;
            y=i;

            if(check(x)||check(y)) {
                cout<<"YES";
                return 0;
            }
```

```
        }
    }
    cout<<"NO";
}
```

D、阅读理解题 cf523B

```
#include <bits/stdc++.h>
#define ll long long
using namespace std;
int n, T;
double c;
int ans[200010];
int m, p;
int main() {
    scanf("%d%d%lf", &n, &T, &c);
    int i;
    for (i = 1; i <= n; ++i) {
        scanf("%d", &ans[i]);
    }
    scanf("%d", &m);
    i = 1;
    ll sum = 0;
    double mean = 0;
    int j;
    for (j = 1; j <= m; ++j) {
        scanf("%d", &p);
        for ( ; i <= p; ++i) {
            sum += ans[i];
            if (i > T)
                sum -= ans[i - T];
            mean = (mean + (double)ans[i] / T) / c;
        }
        double real = (double)sum / T;
        printf("%f %f %f\n", real, mean, (double)fabs(mean - real) / real);
    }
}
```

E、千万不要莽贪心

> 这是一道简单的dp问题，为什么不能用贪心，因为使用贪心的条件是无后效性，也就是说当前选择对后面的选择没有任何影响时才可以用贪心算法进行解决。
>
> 这题采用动态规划解决对于（0-N）的每个（A[j]-A[N]），都做一次累加，将每次累加的结果做一次判断（小于零则舍弃），大于MaxSum则统计到MaxSum里，则N次循环内，必然会有一个最优解被更新为MaxSum的值。
>
> 状态转移方程为：
>
> dp[i] = max{ dp[i-1]+A[i] , A[i] }
> 考虑到结尾并不需要每一个dp[i] , 其实用两个变量代替就可以:
>
> ThisSum = max{LastSum+A[i] , A[i]}
> 本题的要求同时输出最大子序列的上界和下界，需要几个变量来统计。

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

```
#define ll long long
using namespace std;
ll a[100005];
int main() {
    memset(a,0,sizeof(a));
    ll n,m,i,j,z=0;
    cin>>n;
    while(n--) {
        z++;
        cin>>m;
        for(i=0; i<m; i++)
            cin>>a[i];
        ll Max=-32767,sum=0,ei=-1,bi=0,b1,b2,k;//MAX最大值   sum记录临时最大值
        ll l=0;
        for(i=0; i<m; i++) {
            if(sum<0){
                sum=a[i];
                bi=i;
                ei=i;
            }else{
                sum+=a[i];
                ei++;
            }
            if(sum>Max){
                Max=sum;
                b1=bi;
                b2=ei;
            }
        }
        cout<<"Case "<<z<<":"<<endl;
        cout<<Max<<" "<<b1+1<<" "<<b2+1<<endl;
        if(n)cout<<endl;
        sum=0;
        Max=0;
        memset(a,0,sizeof(a));
    }
    return 0;
}
```

F、工具人cpx

> 给出一个长度为n的序列和一个整数k，求最少需要多少次操作可以把序列的中位数变成k。一次操作定义为将序列中的任意一个数+1或-1。
>
> 经过观察可以知道，不改变序列元素之间的相对大小是更优的选择。
>
> 运用贪心策略
>
> 先对序列进行排序。
>
> 如果当前中位数小于k，那么就把它加到k，为了保证序列的相对大小不变，需要把它后面小于k的数都加到k。大于k时同理

```
#include<iostream>
#include<algorithm>
using namespace std;
int a[200001];
```

```
int cmd(int a,int b){
    return a<b;
}
int main(){
    long long sum=0,n,s;
    cin>>n>>s;
    for(int i=0;i<n;i++)
        cin>>a[i];
    sort(a,a+n);
    if(a[n/2]<=s){
        for(int i=n/2;i<n;i++)
        {
            if(a[i]>=s) break;
            else {
                sum=sum+s-a[i];
            }
        }
    }
    else if(a[n/2]>=s){
        for(int i=n/2;i>=0;i--){
            if(a[i]<=s) break;
            else{
                sum=sum+a[i]-s;
            }
        }
    }

    cout<<sum<<endl;
}
```

G、工具人wyh

> 看图可知，由于蜜蜂每次只能从前1个蜂房前2个蜂房过来，那么f(n)=f(n-2)+f(n-1)
>
> 如果n<3时的情况特判一下即可
>
> f(1)=0，因为蜜蜂开始在第1个蜂房；
>
> f(2)=1，蜜蜂只能从第1个蜂房来到第2个蜂房；
>
> f(3)=2，蜜蜂可以从第1个蜂房过来，也可以从第2个蜂房过来
>
> f(n)=f(n-2)+f(n-1)，n>3
>
> 当然需要打个表记录一下路线数量

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

using namespace std;


int main(){
    int n,a,b;
    cin>>n;
    double way[51]={0};
    way[2]=1;
    way[3]=2;
```

```
    for(int i=4;i<=50;i++)
        way[i]=way[i-1]+way[i-2];

    while(n--){
        cin>>a>>b;
        printf("%.0f\n",way[b-a+1]);
    }
}
```

H、工具人cpx2

> 和上面一题一样找到一个递推关系式即可做出
>
> 当n==1时 只有一种情况
>
> 当n==2时 有两种情况
>
> 当n>3时的最后一块格子，摆放只有两种方式，一种竖排，那么方案总数为f(n-1)，一种是横着排，那么方案总数为f(n-2)，则f(n)=f(n-1)+f(n-2)
>
> 打个表储存f(n)

```
#include<bits/stdc++.h>
using namespace std;
long long a[100];
int main(){
    a[0]=0;
    a[1]=1;
    a[2]=2;
    int n;
    for(int i=3;i<=51;i++)
        a[i]=a[i-1]+a[i-2];
    while(cin>>n)
        cout<<a[n]<<endl;
}
```

I、全员莽夫的暑期EOF串串

> 从最后一个向前递推
> 当第n个是O时，那么第n-1个不能为O，此种情况有f(n-2) * 2个
> 当第n个为非O时，有f(n-1) * 2个；
> 综上得递推公式：f(n)=f(n-2) * 2+f(n-1) * 2

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

using namespace std;
long long a[51];
int main(){
    int n;
    a[0]=1;
    a[1]=3;
    for(int i=2;i<51;i++){
        a[i]=(a[i-1]+a[i-2])*2;
    }
```

```
    while(cin>>n){
        cout<<a[n]<<endl;
    }
}
```

J、卡时间的贪心

> 注意n的范围时3*10^5如果直接进行排序的贪心策略时间复杂度为n^2 log n 那么2s的时间时跑不出来的，所以算法必须在n log n 的时间范围内，这时候就需要采用优先队列的stl库
>
> emmmmm用了就可以用正常的贪心思想做，每次弹出最大的两个匹配后
>
> 按照美丽值b从大到小排序
> 用优先队列存歌曲长度t
> 当选择的曲子数量超过k时， 弹出优先队列中最小的歌曲长度t即可

```
#include<bits/stdc++.h>
#define INF 0x3F3F3F3F
#define INF_LL 0x7FFFFFFF
#define ll long long
#define pb(x) push_back(x)
const double Pi = acos(-1.0);
using namespace std;
const int maxn=300005;
struct node{
    int len;
    int beu;
    friend bool operator < (node p,node q){
        return p.beu>q.beu;
    }
}a[maxn];
priority_queue<int,vector<int>,greater<int> >h;
int n,k;
int main(){
    FAST_IO;
    cin>>n>>k;
    for(int i=1;i<=n;i++){
        cin>>a[i].len>>a[i].beu;
    }
    sort(a+1,a+1+n);
    long long sum=0,ans=0;
    for(int i=1;i<=n;i++){
        h.push(a[i].len);
        sum+=a[i].len;
        while(h.size()>k){
            sum-=h.top();
            h.pop();
        }
        ans=max(ans,sum*a[i].beu);
    }
    cout<<ans<<endl;;
}
```