出生年：

拆分每一位放到集合里，使用集合的size来判断是否满足要求

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

int main()
{
    int n, cnt;
    set<int> s;
    cin >> n >> cnt;
    int x = n;
    while(1)
    {
        s.clear();
        s.insert(x/1000);//千位
        s.insert(x%1000/100);//百位
        s.insert(x%100/10);//十位
        s.insert(x%10);//个位
        if(s.size()==cnt) break;
        x++;
    }
    cout << x-n << " ";
    printf("%04d", x);//输出宽度为4为，左边填充0

}
```

吉老师的回归：

使用字符串查找函数即可
注意cin和getline的使用

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

int main()
{
    int n, m;
    cin >> n >> m;
    getchar();//因为后面要使用getline，其会读入空格和回车，所以要把本次cin之后的回车吸收掉
    int cnt = 0, i;
    for(i=0; i<n; i++)
    {
        string s;
        getline(cin, s);
        if(s.find("qiandao")==string::npos && s.find("easy")==string::npos)
        {
            cnt++;
            if(cnt==m+1)
            {
                cout << s;
```

```
                break;
            }
        }
    }
    if(i==n) cout << "Wo AK le";
    return 0;
}
```

倒数第N个字符串：

每一位都是小写字母，所以可以它们看作是26进制的运算。
先算出len位26进制数对应的十进制数（因为要与N运算，而N是十进制），然后算出十进制的倒数第N个是多少
pow(26, len)-1：len位26进制数最大值对应的十进制数，也是倒数第一个。所以倒数第N个就是pow(26, len)-N
算出十进制的倒数第N个后，再转成26进制数，每一位数转成相应英文字母就行
注意转换后不足len位的话，前面补'a'

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

int main()
{
    int len, n;
    cin >> len >> n;
    int num = pow(26, len) - n;//求倒数第n个数对应的十进制数
    string res = "";
    while(num)//把num转为26进制对应的英文字符
    {
        res = char(num%26+97)+res;
        num /= 26;
    }
    res = string(len-res.size(), 'a') + res;//若res不够len位，在前面补'a'
    cout << res;
}
```

前世档案：

是为0，否为1，把从上往下的路径看作是二进制数，对应的编号就是相应十进制数+1

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

int main()
{
    int n, m;
    int num, cur;
    char c;
    cin >> n >> m;
    for(int i=0; i<m; i++)
    {
        int res = 0;
        for(int j=0; j<n; j++)
        {
```

```
            cin >> c;
            res *= 2;
            if(c=='n') res++;
        }
        cout << res+1 << endl;
    }
}
```

天梯赛座位分配：

a数组记录每一个高校的参赛人数
cur是当前分配到的座位号，每次cur++ 或 cur+=2（只剩一个高校时）
用一个队列保存当前待分配的高校编号，高校们轮流出队再排队，每次给出队的高校分配一个座位号，同时a数组相应的高校人数减1，如果高校人数变为0，该高校不再去排队。

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

int main()
{
    int n;
    int a[100];
    queue<int> s;
    vector<int> res[100];
    cin >> n;
    for(int i=0; i<n; i++)
    {
        cin >> a[i];
        a[i] *= 10;//i号高校的人数
        s.push(i);//i号高校进入待分配座位的队列
    }
    int cur = 1;//当前分配的座位号
    int t;
    while(1)
    {
        t = s.front();
        s.pop();
        if(s.size()==0)//只剩一个高校
        {
            while(a[t]--)
            {
                res[t].push_back(cur);
                cur += 2;
            }
            break;
        }
        res[t].push_back(cur++);//t号高校分配座位号
        a[t]--;// t号高校待分配人数减1
        if(a[t]) s.push(t);// t号高校还没分配完
    }
    for(int i=0; i<n; i++)
    {
        cout << "#" << i+1 << endl;
        int j = 1;
        for(int x:res[i])
        {
```

```
            cout << x;
            if(j%10==0) cout << "\n";
            else cout << " ";
            j++;
        }
    }
}
```

猜数字：

挺简单的，不解释

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

int main()
{
    int n;
    cin >> n;
    string name[10000];
    int num[10000];
    int i;
    int sum = 0;
    for(i=0; i<n; i++)
    {
        cin >> name[i] >> num[i];
        sum += num[i];
    }
    int avg = sum / n / 2;
    int min = abs(num[0]-avg);
    int id = 0;
    for(i=1; i<n; i++)
    {
        if(abs(num[i]-avg)<min)
        {
            min = abs(num[i]-avg);
            id = i;
        }
    }
    cout << avg << " " << name[id];
}
```

玩转二叉树：

针对当前指定的先序序列范围，保存当前子树的根，然后递归遍历右子树和左子树
但是遍历子树所得到的根顺序，与层次顺序不一致
解决办法是 利用层次遍历的原理，当前子树的根是层次顺序中第i个结点的话，当前的左右子树的根一定是层次顺序的第2i和第2i+1个结点。
注意，前面的方法是基于满二叉树的编号规律。因为有些编号其实是空结点，所以在输出时要进行判断

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

int a[31], b[31];
```

```
int res[1000];

void dfs(int l, int r, int rid, int num)
{//l~r: 先序遍历的子树范围, rid: 当前子树的根在b数组的下标，num: 当前子树的根在层次遍历中的序号
    //cout << l << " " << r << " " << num << " " << rid << endl;
    if(l>r) return;
    res[num] = b[rid];
    int i;
    for(i=l; i<=r && a[i]!=b[rid]; i++);//在中序里找左右子树的分界点(就是找根)
    if(i!=r)//有右子树
        dfs(i+1, r, rid+i-l+1, 2*num);//因为是镜面反转，所以先dfs右子树
    if(i!=l)//有左子树
        dfs(l, i-1, rid+1, 2*num+1);
}

int main()
{
    int n;
    cin >> n;
    for(int i=0; i<n; i++) cin >> a[i];
    for(int i=0; i<n; i++) cin >> b[i];
    dfs(0, n-1, 0, 1);
    int cnt = 0;
    for(int i=1; cnt<n; i++)
    {
        if(res[i]!=0 && cnt>0)
        {
            cout << " " << res[i];
            cnt++;
        }
        else if(res[i]!=0)
        {
            cout << res[i];
            cnt++;
        }
    }
    return 0;
}
```

抢红包：

简单，结构体+sort

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

struct node
{
    int id, cnt = 0;
    int rest = 0;
};

bool cmp(node x, node y)
{
    if(x.rest!=y.rest) return x.rest>y.rest;
    else if(x.cnt!=y.cnt) return x.cnt>y.cnt;
```

```
    else return x.id<y.id;
}

int main()
{
    int n;
    cin >> n;
    node a[10001];
    for(int i=1; i<=n; i++)
    {
        a[i].id = i;
        int k;
        cin >> k;
        int id, x;
        while(k--)
        {
            cin >> id >> x;
            a[id].rest += x;
            a[id].cnt++;
            a[i].rest -= x;
        }
    }
    sort(a+1, a+n+1, cmp);
    for(int i=1; i<=n; i++)
        printf("%d %.2f\n", a[i].id, a[i].rest/100.0);

    return 0;
}
```

多项式A/B：

没啥技巧，直接模拟多项式相除的过程。
希望你写得比我好

```
//测试点2是比较大的指数
#include<bits/stdc++.h>
using namespace std;

#define LEN 5000//2000会出错

void print(double a[], int t)
{
    int cnt = 0;
    for(int i=t; i>=0; i--)
    {
        //a[i] = round(a[i]*10)/10.0;
        if(a[i]<0.05 && a[i]>=-0.05) a[i] = 0;
        if(a[i]!=0) cnt++;
    }
    if(cnt==0) cout << "0 0 0.0";
    else
    {
        cout << cnt;
        for(int i=t; i>=0; i--)
            if(a[i]!=0) printf(" %d %.1lf", i, a[i]);
    }
    cout << endl;
```

```
}

int main()
{
    int n;
    double a[LEN] = {0};
    double b[LEN] = {0};
    int maxa, maxb;
    cin >> n;
    for(int i=0; i<n; i++)
    {
        int x;
        cin >> x;
        cin >> a[x];
        if(i==0) maxa = x;//多项式a的最高指数
    }
    cin >> n;
    for(int i=0; i<n; i++)
    {
        int x;
        cin >> x;
        cin >> b[x];
        if(i==0) maxb = x;//多项式b的最高指数
    }
    double res[LEN] = {0};
    for(int k=maxa; k>=maxb; k--)//从a的最高指数开始除
    {
        if(a[k]==0) continue;
        int x = k-maxb;
        double y = a[k]/b[maxb];
        res[x] = y;
        double tmp[LEN] = {0};
        for(int i=maxb; i>=0; i--)//两式相减，生成新的被除式
            a[i+x] -= y*b[i];
    }
    print(res, maxa);
    print(a, maxb-1);
    return 0;
}
```

冰岛人：

要搞清楚题目，还是得花一些时间。论阅读理解的重要性。
读入姓名同时存入mp字典，具体看注释
使用find函数查询两异性是否五服之内

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

map<string, pair<string, int>> mp;
//维京人：mp[名]={姓，男0女1}
//非维京人：mp[名]={""，男0女1}

bool find(string x, string y)
{
    int i=1;//第一个人的第几服
    for(string s=x; s!=""; s=mp[s].first)//第一个人网上找祖先
```

```
    {
        int j=1;//第二个人的第几服
        for(string t=y; t!=""; t=mp[t].first)//循环条件使用mp.find(t)!=mp.end()也会超时
        {
            if(i>=5 && j>=5) break;//没有这个条件判断会超时
            if(s==t && (i<5 || j<5)) return 1; //祖先相等且都在五服之内
            j++;
        }
        i++;
    }
    return 0;
}

int main()
{
    int n;
    cin >> n;
    while(n--)
    {
        string s, t;
        cin >> s >> t;
        if(t.back()=='m' || t.back()=='f')//非维京人
        {
            mp[s].first = "";
            mp[s].second = t.back()=='m' ? 0:1;
        }
        else if(t.back()=='n')//维京儿子
        {
            mp[s].first = t.substr(0, t.size()-4);
            mp[s].second = 0;
        }
        else//维京女儿
        {
            mp[s].first = t.substr(0, t.size()-7);
            mp[s].second = 1;
        }
    }
    cin >> n;
    while(n--)
    {
        string s1, t1, s2, t2;
        cin >> s1 >> t1 >> s2 >> t2;
        if(mp.find(s1)==mp.end()||mp.find(s2)==mp.end()) cout << "NA\n";
        else if(mp[s1].second==mp[s2].second) cout << "Whatever\n";
        else if(find(s1, s2)==0) cout << "Yes\n";
        else cout << "No\n";
    }
    return 0;
}
```

口罩发放：

测试点1、6是当天口罩额度为0 sort的底层是快排，是不稳定的排序算法。 因此，要获得稳定的排序结果，要以出现顺序为第二关键字进行排序.
没啥技巧，正确模拟发放流程即可

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

struct node
{
    int num;
    string na, id, ti;
    int si;
};

vector<pair<string, string>> sick;//保存生病记录
set<string> ssi;//保存生病的身份证号

map<string, int> mp;//保存发放过口罩的id号和第几天发放


bool cmp(node x, node y)
{
    if(x.ti==y.ti) return x.num<y.num;
    else return x.ti<y.ti;
}

bool ju(string s)
{
    for(auto x:s)
        if(!isdigit(x)) return 0;
    return 1;
}

int main()
{
    int d, p;
    cin >> d >> p;
    for(int i=1; i<=d; i++)//第i天
    {
        vector<node> v;
        int t, s;
        cin >> t >> s;
        int cnt = 0;
        for(int j=0; j<t; j++)//处理t个申请
        {
            node tmp;
            cin >> tmp.na >> tmp.id >> tmp.si >> tmp.ti;
            if(tmp.id.size()!=18 || ju(tmp.id)==0) continue;
            if(tmp.si==1 && ssi.find(tmp.id)==ssi.end())
            {
                sick.push_back({tmp.na, tmp.id});
                ssi.insert(tmp.id);
            }
            tmp.num = cnt++;
            v.push_back(tmp);//有资格申请口罩
```

```
        }
        sort(v.begin(), v.end(), cmp);
        int k = 0;
        for(auto x:v)
        {
            if(mp[x.id]==0 || mp[x.id]+p<i)
            {
                if(k==s) break;
                printf("%s %s\n", x.na.c_str(), x.id.c_str());
                mp[x.id] = i;
                k++;
            }
        }
    }
    for(auto x:sick)
        printf("%s %s\n", x.first.c_str(), x.second.c_str());

    return 0;
}
```

包装机：

好像没啥，挺简单的

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

int main()
{
    int n, m, max;
    cin >> n >> m >> max;
    queue<char> v[n];
    for(int i=0; i<n; i++)
    {
        string s;
        cin >> s;
        for(int j=0; j<m; j++) v[i].push(s[j]);
    }
    stack<char> s;
    int x;
    while(cin>>x)
    {
        if(x==-1) break;
        if(x==0)//取筐
        {
            if(s.size()!=0)
            {
                cout << s.top();
                s.pop();
            }
        }
        else if(v[x-1].size()>0) //取轨道，轨道非空
        {
            if(s.size()==max)//筐满
            {
```

```
                cout << s.top();
                s.pop();
            }
            s.push(v[x-1].front());
            v[x-1].pop();
        }
    }
    return 0;
}
```