问题描述   
　　闲暇时，福尔摩斯和华生玩一个游戏：   
　　在N张卡片上写有N个整数。两人轮流拿走一张卡片。要求下一个人拿的数字一定是前一个人拿的数字的约数或倍数。例如，某次福尔摩斯拿走的卡片上写着数字“6”，则接下来华生可以拿的数字包括：   
　　1，2，3, 6，12，18，24 ….   
　　当轮到某一方拿卡片时，没有满足要求的卡片可选，则该方为输方。   
　　请你利用计算机的优势计算一下，在已知所有卡片上的数字和可选哪些数字的条件下，怎样选择才能保证必胜！   
　　当选多个数字都可以必胜时，输出其中最小的数字。如果无论如何都会输，则输出-1。   
输入格式   
　　输入数据为2行。第一行是若干空格分开的整数（每个整数介于1~100间），表示当前剩余的所有卡片。   
　　第二行也是若干空格分开的整数，表示可以选的数字。当然，第二行的数字必须完全包含在第一行的数字中。   
输出格式   
　　程序则输出必胜的招法！！   
样例输入   
2 3 6   
3 6   
样例输出   
3   
样例输入   
1 2 2 3 3 4 5   
3 4 5   
样例输出   
4

在官网上能过，但在学校oj上超时

**博弈论+dfs**

总之分析下必胜态和必败态，如果我拿了一张牌，并且没有其他的约数或倍数可拿，显然这是我的必胜态。如果在这之前，对方也拿了一张牌，显然对他来说他拿的那张牌就是必败态，所以这种博弈里，必胜态和必败态是交替出现的   
但要确定是必败还是必胜，不抽到最后一张还是不能确定，所以要把所有可能的抽卡情况搜索出来，只有确定抽了这张牌，之后所有的情况都是必败，我才能确定这张牌是必胜

#include <iostream>

#include <cstdio>

#include <cstdlib>

#include <cstring>

#include<vector>

#include<algorithm>

using namespace std;

vector<int> choice;

vector<int> table[110];

int num[110];

int dfs(int x)//必败态和必胜态轮流出现，所以要找取第一个x时到底是必胜还是必败

//-1是必胜态，1是必败态

{

for(int i=table[x].size()-1;i>=0;i--)//从大到小取数，不然会超时，想不出为什么

{

if(num[table[x][i]])

{

num[table[x][i]]--;

int t=dfs(table[x][i]);

num[table[x][i]]++;

if(t==-1)//如果下一个状态是必胜态

return 1;//那当前状态就是必败态，并且容易看出从第一个dfs状态看来，只要后面有一个必胜态，那结果就一定是必败

}

}

return -1;//没有下一个约数或倍数了，取到这个x的人必胜

}

int main()

{

memset(num,0,sizeof(num));

char c;

int x;

while(1)

{

cin>>x;

num[x]++;

c=getchar();

if(c=='\n')

break;

}

while(1)

{

cin>>x;

choice.push\_back(x);

c=getchar();

if(c=='\n')

break;

}

for(int i=1;i<=100;i++)//仍属于预处理

if(num[i])

{

num[i]--;//相同数字如果出现多次也可以用

for(int j=1;j<=100;j++)

if(num[j] && (i%j==0||j%i==0))

table[i].push\_back(j);

num[i]++;

}

sort(choice.begin(),choice.end());//choice是当前可选择的数

bool flag=0;

for(int i=0;i<choice.size();i++)

{

num[choice[i]]--;//num[i]为0是当前的数不可选择,大于0是可以选择

int t=dfs(choice[i]);//判断选这个数是必胜态还是必败态

if(t==-1)

{

cout<<choice[i]<<"\n";

flag=1;

break;

}

num[choice[i]]++;

}

if(flag==0)cout<<"-1\n";

return 0;

}