待打印的

模运算与基本四则运算有些相似，但是除法例外。其规则如下：

1. (a + b) % p = (a % p + b % p) % p （1）
2. (a - b) % p = (a % p - b % p) % p （2）
3. (a * b) % p = (a % p * b % p) % p （3）
4. a ^ b % p = ((a % p)^b) % p （4）

- 结合律：
  ((a+b) % p + c) % p = (a + (b+c) % p) % p （5）

((a*b) % p * c)% p = (a * (b*c) % p) % p （6）

- 交换律：
  (a + b) % p = (b+a) % p （7）

(a * b) % p = (b * a) % p （8）

- 分配律：
  (a+b) % p = (a % p + b % p) % p （9）
  ((a +b)% p * c) % p = ((a * c) % p + (b * c) % p) % p （10）

重要定理

- 若a≡b (% p)，则对于任意的c，都有(a + c) ≡ (b + c) (%p)； （11）
- 若a≡b (% p)，则对于任意的c，都有(a * c) ≡ (b * c) (%p)； （12）
- 若a≡b (% p)，c≡d (% p)，则 (a + c) ≡ (b + d) (%p)，(a - c) ≡ (b - d) (%p)，(a * c) ≡ (b * d) (%p)，(a / c) ≡ (b / d) (%p)； （13）

2、头文件：

```
#include<iostream>
#include<cstring>
#include<cstdio>
#include<string>
#include<map>
#include<stack>
#include<queue>
#include<set>
#include<sstream>
#include<ctime>
#include<algorithm>
#include<bits/stdc++.h>
#include<sstream>
using namespace std;
const int maxn = 10086;
#define inf 0x3f3f3f3f
#define eps 1e-8
#define pi acos(-1.0)
typedef long long LL;
void anhangduru(){
  string line;
```

```
while(getline(cin,line)){
int sum=0;
int x;
stringstream ss(line);
while(ss>>x){sum+=x;}//按空格读入
}
}//按行读入
//加上符号重载
int main()
{
ios::sync_with_stdio(0);//输入输出挂
return 0;
}
```

3、翻转问题模板题：

Description

在一个n*m的网格内，每个格子都可以翻转正反面。
当翻转一个格子时，与他相邻的格子（上下左右）也会被同时翻转。
现在给定初始时每个格子的状态，你需要确定是否可以通过一系列翻转操作使得所有格子都正面朝上，若可以，求出最少的翻转次数。

Input

多组数据

第一行有一个整数T(T<=20)，代表数据组数

对每组数据，第一行有两个由空格分割的整数n，m(2<=n, m <= 15)，表示网格的规模

接下来n行，每行有一个长度为m的01字符串，描述了网格的初始状态。0表示反面，1表示正面。

Output

对每组数据，若可以使得所有格子都翻转正面，输出Yes以及最少需要的翻转次数，由一个空格隔开；否则输出No

Sample Input

```
2
4 2
10
00
01
11
4 4
0100
1011
0010
1100
```

Sample Output

```
Yes 3
No
```

代码加注释：

```
#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>
using namespace std;
typedef long long LL;
const int N = 15 + 5;
const int inf = 0x3f3f3f3f;
int T;
int n, m;
char s[N][N];
int flip[N][N];
int dx[] = {0, 0, 0, -1};
int dy[] = {0, 1, -1, 0};
```

```
inline int in(int x, int y) {
    return x >= 0 && x < n && y >= 0 && y < m;
}
int get(int x, int y) {
    int ret = s[x][y] - '0';
    for(int i = 0; i < 4; i++) {
        int nx = x + dx[i], ny = y + dy[i];
        if(in(nx, ny)) ret ^= flip[nx][ny];
    }
    return ret;
}//一个位置[x][y]当前的情况由它本身、它的上面[x-1][y]、它的左面[x][y-1]和它的右面[x][y+1]的翻转情况决定，因为这些位置的翻转情况都已经确定了，只有
//[x-1][y]的不确定，所以不用写。之所以写异或，因为0异或1等于1,1异或1等于0，刚好可以用来模拟翻转的那个过程
//这个函数的作用就是返回[x][y]的当前状态
int solve(int S) {//这个S指的是翻转情况，而不是第一行现在面朝上的情况
    int cnt = __builtin_popcount(S);//初始化为第一行的翻转数目。
    for(int i = 0; i < m; i++) {
        flip[0][i] = (S>>i)&1;
    }//获得第一行的翻转情况
    for(int i = 1; i < n; i++) {
        for(int j = 0; j < m; j++) {
            if(get(i-1, j) == 0) {
                flip[i][j] = 1;//用来表示格子[i][j]需不需要翻转，等于一表示需要翻转，等于0表示不需要翻转
                cnt++;
            }else flip[i][j] = 0;
        }
    }
    for(int j = 0; j < m; j++) {
        if(get(n-1, j) == 0) return inf;
    }//判断最后一行的情况 ，如果最后一行不满足要求的全为状态，就返回inf无穷大。
    return cnt;
}
int main() {
    scanf("%d", &T);
    while(T--) {
        scanf("%d%d", &n, &m);
        for(int i = 0; i < n; i++) {
            scanf("%s", s[i]);
        }
        int ans = inf;
        for(int i = 0; i < 1<<m; i++) {
            ans = min(solve(i), ans);//用状态压缩的思想枚举第一行每一种翻转情况，找到最小的结果
        }
        if(ans == inf) puts("No");
        else printf("Yes %d\n", ans);
    }
    return 0;
}
```

4、

Submit: 198 Solved: 13
[Submit][Status][Web Board]

Description

记gcd(a,b)为正整数a、b的最大公因数。
计算gcd(1,1)*gcd(1,2)*...*gcd(1,m)*gcd(2,1)*gcd(2,2)*...*gcd(2,m)*...*gcd(n,1)*gcd(n,2)*...*gcd(n,m)

Input

多组数据

第一行有一个整数T(T<=20)，代表数据组数

对每组数据，有两个由空格分隔的整数n，m(1<=n, m <= 10^7)

Output

对每组数据，输出答案除以998244353的余数

Sample Input

```
2
2 2
4 4
```

Sample Output

```
2
96
```

HINT

代码：

```
#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>
using namespace std;
typedef long long LL;
const int N = 1e7 + 5;
const int mod = 998244353;
using namespace std;
int T;
int p[N], vis[N], C;
void init() {
    for(int i = 2; i < N; i++) {
        if(!vis[i]) p[C++] = i;
        for(int j = 0; j < C && 1LL*i*p[j] < N; j++) {
            vis[i*p[j]] = 1;
            if(i%p[j] == 0) break;
        }//O(nlog(n))
    }
}
LL powMod(LL a, LL b) {
    LL r = 1;
    for(; b; b >>= 1) {
        if(b&1) r = r*a%mod;
        a = a*a%mod;
    }
    return r;
}//O(log(n))
int n, m;
LL get(int x) {
    LL ret = 0;
```

```
    for(LL i = x; i <= n && i <= m; i *= x) {
        ret += (n/i)*(m/i);
    }
    return ret;
}

int main() {
    //freopen("data/in0.txt", "r", stdin);
    //freopen("data/out0.txt", "w", stdout);
    init();
    scanf("%d", &T);
    while(T--) {
        scanf("%d%d", &n, &m);
        LL ans = 1;
        for(int i = 0; i < C && p[i] <= n && p[i] <= m; i++) {
            ans = ans*powMod(p[i], get(p[i]))%mod;
        }
        printf("%lld\n", ans);
    }
    return 0;
}
```

5、置换群循环节：

解题思路：

题目要求为给定一个排列 4 1 5 2 3，问经过多少次置换可以重新回到这个排列。即1、2、3、4、5。

定理：

在置换群中有一个定理：设$T^k = e$，（T 为一置换，e 为单位置换（映射函数为$f(x) = x$的置换）），那么 k 的最小正整数解是 T 的拆分的所有循环长度的最小公倍数。

什么是循环节。。比如

1 2 3 4 5　从上向下看

4 1 5 2 3

1->4->2->1 （1,4,2）为一个循环节，长度为3　3->5->3（3,5）为一个循环节，长度为2　则所求的次数即定理中的k

下面的样例解释转于博客　http://blog.csdn.net/cqlf__/article/details/7910849

分析下样例

1 2 3 4 5

原始序列: 4 1 5 2 3

2 4 3 1 5

p(p(1))=p(4)=2;

p(p(2))=p(1)=4;

p(p(3))=p(5)=3;

...

...

1 2 5 4 3

p(p(p(1)))=p(2)=1;

p(p(p(2)))=p(4)=2;

p(p(p(3)))=p(3)=5;

...

...

4 1 3 2 5

p(p(p(p(1))))=p(1)=4;

p(p(p(p(2))))=p(2)=1;

p(p(p(p(3))))=p(5)=3;

2 4 5 1 3

1 2 3 4 5

就是两个循环节(1,2,4) 和(3,5) 我们只要求出lcm(循环节长度)便是其需要移动的次数

代码：

```
#include <iostream>
#include <string.h>
```

```
using namespace std;
int gcd(int a,int b)
{
    return b==0?a:gcd(b,a%b);
}
int lcm(int a,int b)//求最小公倍数
{
    return a/gcd(a,b)*b;
}
const int maxn=1005;
int num[maxn];
int visit[maxn];
int main()
{
    int n;
    while(cin>>n)
    {
        for(int i=1;i<=n;i++)
            cin>>num[i];
        memset(visit,0,sizeof(visit));
        int res=1,flag;
        for(int i=1;i<=n;i++)
        {
            int count=0;//用来记录每个循环节的长度
            flag=i;//记录下标
            while(!visit[flag])//没有被访问过，不存在于之前的循环节中
            {
                count++;
                visit[flag]=1;
                flag=num[flag];
            }
            if(count)//注意有可能count是0的情况，即外层循环循环到第i个数，而此时第i个数在之前寻找循环节中已经被访问过
                res=lcm(count,res);
        }
        cout<<res<<endl;
    }
    return 0;
}
```

6、嵌套大模拟：

嵌套字符串进行处理现在看起来就是一棵树：

问题描述

JSON (JavaScript Object Notation) 是一种轻量级的数据交换格式，可以用来描述半结构化的数据。JSON 格式中的基本单元是值 (value)，出于简化的目的本题只涉及 2 种类型的值：

* 字符串 (string)：字符串是由双引号 " 括起来的一组字符（可以为空）。如果字符串的内容中出现双引号 "，在双引号前面加反斜杠，也就是用 \" 表示；如果出现反斜杠 \，则用两个反斜杠 \\ 表示。反斜杠后面不能出现 " 和 \ 以外的字符。例如：""、"hello"、"\"\\"。

* 对象 (object)：对象是一组键值对的无序集合（可以为空）。键值对表示对象的属性，键是属性名，值是属性的内容。对象以左花括号 { 开始，右花括号 } 结束，键值对之间以逗号 , 分隔。一个键值对的键和值之间以冒号 : 分隔。键必须是字符串，同一个对象所有键值对的键必须两两都不相同；值可以是字符串，也可以是另一个对象。例如：{}、{"foo": "bar"}、{"Mon": "weekday", "Tue": "weekday", "Sun": "weekend"}。

除了字符串内部的位置，其他位置都可以插入一个或多个空格使得 JSON 的呈现更加美观，也可以在一些地方换行，不会影响所表示的数据内容。例如，上面举例的最后一个 JSON 数据也可以写成如下形式。

```
{
"Mon": "weekday",
"Tue": "weekday",
"Sun": "weekend"
}
```

给出一个 JSON 格式描述的数据，以及若干查询，编程返回这些查询的结果。

输入格式

第一行是两个正整数 n 和 m，分别表示 JSON 数据的行数和查询的个数。

接下来 n 行，描述一个 JSON 数据，保证输入是一个合法的 JSON 对象。

接下来 m 行，每行描述一个查询。给出要查询的属性名，要求返回对应属性的内容。需要支持多层查询，各层的属性名之间用小数点 . 连接。保证查询的格式都是合法的。

输出格式

对于输入的每一个查询，按顺序输出查询结果，每个结果占一行。

如果查询结果是一个字符串，则输出 STRING <string>，其中 <string> 是字符串的值，中间用一个空格分隔。

如果查询结果是一个对象，则输出 OBJECT，不需要输出对象的内容。

如果查询结果不存在，则输出 NOTEXIST。

如果查询结果不存在，则输出 NOTEXIST。

样例输入

```
10 5
{
"firstName": "John",
"lastName": "Smith",
"address": {
"streetAddress": "2ndStreet",
"city": "NewYork",
"state": "NY"
},
"esc\\aped": "\"hello\""
}
firstName
address
address.city
address.postal
esc\aped
```

样例输出

```
STRING John
OBJECT
STRING NewYork
NOTEXIST
STRING "hello"
```

评测用例规模与约定

$n \leqslant 100$，每行不超过 80 个字符。

$m \leqslant 100$，每个查询的长度不超过 80 个字符。

字符串中的字符均为 ASCII 码 33-126 的可打印字符，不会出现空格。所有字符串都不是空串。

所有作为键的字符串不会包含小数点 .。查询时键的大小写敏感。

50%的评测用例输入的对象只有 1 层结构，80%的评测用例输入的对象结构层数不超过 2 层。举例来说，{"a": "b"} 是一层结构的对象，{"a": {"b": "c"}} 是二层结构的对象，以此类推。

代码：

```
#include<cstdio>
#include<cstring>
#include<cstdlib>
#include<cctype>
#include<cmath>
```

```
#include<iostream>
#include<sstream>
#include<iterator>
#include<algorithm>
#include<string>
#include<vector>
#include<set>
#include<map>
#include<stack>
#include<deque>
#include<queue>
#include<list>
#define lowbit(x) (x & (-x))
const double eps = 1e-8;
inline int dcmp(double a, double b){
    if(fabs(a - b) < eps) return 0;
    return a > b ? 1 : -1;
}
typedef long long LL;
typedef unsigned long long ULL;
const int INT_INF = 0x3f3f3f3f;
const int INT_M_INF = 0x7f7f7f7f;
const LL LL_INF = 0x3f3f3f3f3f3f3f3f;
const LL LL_M_INF = 0x7f7f7f7f7f7f7f7f;
const int dr[] = {0, 0, -1, 1, -1, -1, 1, 1};
const int dc[] = {-1, 1, 0, 0, -1, 1, -1, 1};
const int MOD = 1e9 + 7;
const double pi = acos(-1.0);
const int MAXN = 8000 + 10;
const int MAXT = 10000 + 10;
using namespace std;
map<string, int> mp[MAXN];
map<int, int> ma[MAXN];
map<int, string> ans[MAXN];
int len;
string s;
int k;
vector<string> tmp;
bool vis[MAXN];
void solve(int fa, int &pos, int num);
void deal(int &i, int num, int cnt);
void deal(int &i, int num, int cnt){
    int len = s.size();
    int j, et, st;
    string x;
    for(j = i - 1; j >= 0; --j){
        if(s[j] == '\"'){
            et = j;
            break;
        }
    }
    for(j = et - 1; j >= 0; --j){
        if(s[j] == ',' || s[j] == '{'){
            break;
        }
        x += s[j];
    }
    x.resize(x.size() - 1);
```

```
    reverse(x.begin(), x.end());
    mp[num][x] = cnt;
    int id = mp[num][x];
    j = i + 1;
    if(s[j] == '\"'){
        st = j;
        x = "";
        for(j = st + 1; j < len; ++j){
            if(s[j] == ',' || s[j] == '}') break;
            x += s[j];
        }
        x.resize(x.size() - 1);
        ans[num][id] = "STRING " + x;
    }
    else if(s[j] == '{'){
        ans[num][id] = "OBJECT";
        ma[num][id] = ++k;
        solve(id, j, k);
    }
    i = j;
}
void solve(int fa, int &pos, int num){
    int cnt = 0;
    for(int i = pos + 1; i < len; ++i){
        if(s[i] == ':'){
            ++cnt;
            deal(i, num, cnt);
            if(s[i] == '}'){
                pos = ++i;
                break;
            }
        }
    }
}
bool Find(int cur, int num, int l, string& res, int id){
    if(num == l){
        res = ans[cur][id];
        return true;
    }
    else{
        cur = ma[cur][id];
        if(!mp[cur].count(tmp[num])) return false;
        return Find(cur, num + 1, l, res, mp[cur][tmp[num]]);
    }
}
int main(){
    int n, m;
    scanf("%d%d", &n, &m);
    getchar();
    while(n--){
        string str;
        getline(cin, str);
        int l = str.size();
        for(int i = 0; i < l; ++i){
            if(str[i] == ' ') continue;
            if(str[i] == '\\'){
                s += str[i + 1];
                ++i;
```

```
                continue;
            }
            s += str[i];
        }
    }
    len = s.size();
    for(int i = 0; i < len; ++i){
        if(s[i] == '{'){
            ++k;
            vis[k] = true;
            solve(-1, i, k);
        }
    }
    string x;
    while(m--){
        cin >> x;
        int l = x.size();
        tmp.clear();
        int id = 0;
        for(int i = 0; i < l; ++i){
            if(x[i] == '.'){
                tmp.push_back(x.substr(id, i - id));
                id = i + 1;
            }
        }
        tmp.push_back(x.substr(id, l - id));
        l = tmp.size();
        string res;
        bool ok = false;
        for(int i = 1; i <= k; ++i){
            if(!vis[i]) continue;
            if(mp[i].count(tmp[0])){
                int id = mp[i][tmp[0]];
                if(Find(i, 1, l, res, id)){
                    ok = true;
                    printf("%s\n", res.c_str());
                    break;
                }
            }
        }
        if(!ok){
            printf("NOTEXIST\n");
        }
    }
    return 0;
}
```
校赛1483：

[Submit][Status][Web

Description

mex是施加于一个集合S的运算，运算结果为 不属于这个集合的最小的非负整数。
例如，mex{0, 1, 3, 4} = 2, mex{1, 2, 3} = 0
在集合S上，再定义两种操作：
操作1：在S中添加一个数x
操作2：删除S中的某个数x

现在，给定一个初始为空的集合S和一系列操作，你需要在每次操作后，计算出S的mex运算的结果
忽略非法操作（例如添加已经存在的元素或者删除不存在的元素）

Input

多组数据

第一行有一个整数T(T<=20)，代表数据组数

对每组数据，第一行为一个整数n(0<n<=10^5)，表示操作次数

接下来n行，每一行有两个由空格隔开的整数o、x(o为1或2，0<=x<=10^9)，o为1表示添加x，o为2表示删除x

Output

对每一次操作，输出操作后集合S的mex运算结果（若遇到非法操作，忽略该操作，但仍要输出结果）

Sample Input

```
1
4
1 1
1 2
1 0
2 1
```

可以证明我们的结果肯定小于等于操作数n：每次最多向里面加入不相同的小于等于n的数一个，而0到n-1一共有n+1个数，所以最终的结果肯定是在0到n之中。所以就可以使用set将所有n以内的数全部加进去。对于每次操作，1的时候就删去x，2的时候就加入x。每次输出第一个就可以了。另外这里还有一个知识点，就是set的遍历方法，*begin到*end。而且它也是排好序了的。下面来说一下遍历map,set的方法。

set:

set，顾名思义是“集合”的意思，在set中元素都是唯一的，而且默认情况下会对元素自动进行升序排列，支持集合的交(set_intersection),差(set_difference) 并(set_union)，对称差(set_symmetric_difference)

```
set<int>:: iterator it;
for(it = s.begin(); it!=s.end(); it++){
    cout<<(*it)<<endl;
}
```

map:

```
map<int,string> m;
map<int,string>::iterator it;
it = m.find(112);
if(it == m.end()){ exit;}
else m.erase(it);
//类似的也可以通过赋值为空来将一个元素删除
```

生成全排列：

a、正序打印

```
int a[] = {3,1,2};
do{
cout << a[0] << " " << a1 << " " << a2 <<
 endl;
}
while (next_permutation(a,a+3));
```

b、逆序打印

```
int a[] = {3,2,1};
do{
cout << a[0] << " " << a1 << " " << a2 <<
 endl;
}
```

while (prev_permutation(a,a+3));

ac代码：

```
set<int>s;
int main()
{
    ios::sync_with_stdio(0);//输入输出挂
    int t;
    cin>>t;
    while(t--){
        int n;
        cin>>n;
        for(int i=0;i<=n;i++){
            s.insert(i);
        }
        for(int i=0;i<n;i++){
            int p,q;
            cin>>p>>q;
            if(p==1){
                s.erase(q);
            }else{
                s.insert(q);
            }
            printf("%d\n",*s.begin());
        }
    }
    return 0;
}
```

注意：如果全部使用scanf()将会快很多。所以尽量用scanf。

1483、翻转游戏

Submit: 92 Solved: 15
[Submit][Status][Web Board]

Description

在一个n*m的网格内，每个格子都可以翻转正反面。
当翻转一个格子时，与他相邻的格子（上下左右）也会被同时翻转。
现在给定初始时每个格子的状态，你需要确定是否可以通过一系列翻转操作使得所有格子都正面朝上，若可以，求出最少的翻转次数。

Input

多组数据

第一行有一个整数T(T<=20)，代表数据组数

对每组数据，第一行有两个由空格分割的整数n，m(2<=n, m <= 15)，表示网格的规模

接下来n行，每行有一个长度为m的01字符串，描述了网格的初始状态。0表示反面，1表示正面。

Output

对每组数据，若可以使得所有格子都翻转正面，输出Yes以及最少需要的翻转次数，由一个空格隔开；否则输出No

Sample Input

```
2
4 2
10
00
01
11
4 4
0100
1011
0010
1100
```

这个就是枚举第一行的所有情况，一旦第一行的所有情况都确定了，那么下面的情况也都是确定的。比如我第二行的第i个翻不翻取决于第一行的第i个是什么情况，而且这样的影响是递推的。所以这样递推下去，最后判断最后一行是否全为想要的结果就可以了。然后在所有满足的情况下，求最小的次数就可以了。现在需要了解的就是应该如何快速的解决查找最后一排的状态。如果一个一个枚举的话，时间复杂度也是会爆炸的。

知识点：

1、**__builtin_popcount():** 计算一个 32 位无符号整数有多少个位为1

2、枚举加遍历状态的模板：

```
inline int in(int x, int y) {
    return x >= 0 && x < n && y >= 0 && y < m;
}
int get(int x, int y) {
    int ret = s[x][y] - '0';
    for(int i = 0; i < 4; i++) {
        int nx = x + dx[i], ny = y + dy[i];
        if(in(nx, ny)) ret ^= flip[nx][ny];
    }
    return ret;
}//一个位置[x][y]当前的情况由它本身、它的上面[x-1][y]、它的左面[x][y-1]和它的右面[x][y+1]的翻转情况决定，因为这些位置的翻转情况都已经确定了，只有
//[x-1][y]的不确定，所以不用写。之所以写异或，因为0异或1等于1,1异或1等于0，刚好可以用来模拟翻转的那个过程
//这个函数的作用就是返回[x][y]的当前状态
int solve(int S) {//这个S指的是翻转情况，而不是第一行现在面朝上的情况
    int cnt = __builtin_popcount(S);//初始化为第一行的翻转数目。
    for(int i = 0; i < m; i++) {
        flip[0][i] = (S>>i)&1;
    }//获得第一行的翻转情况
    for(int i = 1; i < n; i++) {
        for(int j = 0; j < m; j++) {
            if(get(i-1, j) == 0) {
                flip[i][j] = 1;//用来表示格子[i][j]需不需要翻转，等于一表示需要翻转，等于0表示不需要翻转
                cnt++;
            }else flip[i][j] = 0;
        }
    }
    for(int j = 0; j < m; j++) {
        if(get(n-1, j) == 0) return inf;
    }//判断最后一行的情况 ，如果最后一行不满足要求的全为状态，就返回inf无穷大。
    return cnt;
}
```

我对mz的代码进行了注释，还是很好理解的。

```
int main() {
    scanf("%d", &T);
    while(T--) {
        scanf("%d%d", &n, &m);
        for(int i = 0; i < n; i++) {
            scanf("%s", s[i]);
        }
        int ans = inf;
        for(int i = 0; i < 1<<m; i++) {
            ans = min(solve(i), ans);//用状态压缩的思想枚举第一行每一种翻转情况，找到最小的结果
        }
        if(ans == inf) puts("No");
        else printf("Yes %d\n", ans);
    }
    return 0;
}
```