ACM 模板 *for 北湖咸鱼*

三条咸鱼——zhber、strawberry、miamiao

目录

一、头文件

```
#include <bits/stdc++.h>
using namespace std;
#define fi first
#define se second
#define pb push_back
#define mp make_pair
#define SZ(x) ((int)(x).size())
#define All(x) (x).begin(),(x).end()
#define rep(i,a,n) for (int i=a;i<n;i++)
#define per(i,a,n) for (int i=n-1;i>=a;i--)
#define Close() ios::sync_with_stdio(0),cin.tie(0)
typedef vector<int> VI;
typedef pair<int, int> PII;
typedef long long ll;
typedef long double ld;
typedef unsigned long long ull;
const int maxn = 1e5 + 10;
const int maxm = 3e3 + 5;
const int mod = 1e9 + 7;
```

注：①部分模板可能会重新定义新类型；②不同模板的 maxn、maxm 大小取决于题目要求

二、算法

1.快速读入

①普通快速读

```
inline ll read() {
	ll x=0,f=1;char ch=getchar();
	while(ch<'0'||ch>'9'){if(ch=='-')f=-1;ch=getchar();}
	while(ch>='0'&&ch<='9'){x=x*10+ch-'0';ch=getchar();}
	return x*f;
}
```

②文件读入流

```
namespace fastIO {
#define BUF_SIZE 100000
	//fread -> read
	bool IOerror = 0;
	inline char nc() {

```

```
7                 static char buf[BUF_SIZE], *p1 = buf + BUF_SIZE, *pend = buf
8   + BUF_SIZE;
9                 if(p1 == pend) {
10                        p1 = buf;
11                        pend = buf + fread(buf, 1, BUF_SIZE, stdin);
12                        if(pend == p1) {
13                                IOerror = 1;
14                                return -1;
15                        }
16                } return *p1++;
17        }
18        inline bool blank(char ch) {
19                return ch == ' ' || ch == '\n' || ch == '\r' || ch == '\t';
20        }
21        inline void read(ll &x) {
22                char ch;
23                ll f = 1;
24                while(blank(ch = nc()));
25                while (ch == '-')f = -f, ch = nc();
26                if(IOerror)return;
27                for(x = ch - '0'; (ch = nc()) >= '0' && ch <= '9'; x = x * 10
28  + ch - '0');
29                x *= f;
30        }
31        inline void read(int &x) {
32                char ch;
33                int f = 1;
34                while(blank(ch = nc()));
35                while (ch == '-')f = -f, ch = nc();
36                if(IOerror)return;
37                for(x = ch - '0'; (ch = nc()) >= '0' && ch <= '9'; x = x * 10
38  + ch - '0');
39                x *= f;
40        }
41        inline void read(char *s) {
42                char ch;
43                while(blank(ch = nc()));
44                if(IOerror)return;
45                int i = 0;
46                for (; !blank(ch); s[i++] = ch, ch = nc());
47                s[i] = '\0';
48        }
49  #undef BUF_SIZE
50  };
    using namespace fastIO;
```

2.LCA 问题

①RMQ 在线求 LCA，可以配合 ST

```
void RMQ() {
	for (int j = 1; (1 << j) <= n; j++)
		for (int i = 1; i <= n; i++)
			if (~pa[i][j - 1])
				pa[i][j] = pa[pa[i][j - 1]][j - 1];
}
int LCA(int x, int y) {
	if (deep[x] < deep[y]) swap(x, y);
	int i, j;
	for (i = 0; (1 << i) <= deep[x]; i++); i--;
	for (j = i; j >= 0; j--) // 把深度统一
		if (deep[x] - (1 << j) >= deep[y])
			x = pa[x][j];
	if (x == y) return x;
	for (j = i; j >= 0; j--) //去找 LCA
		if (pa[x][j] != -1 && pa[x][j] != pa[y][j]) {
			x = pa[x][j];
			y = pa[y][j];
		}
	return pa[x][0];
}
```

②Tarjan 离线求 LCA

3.STL 技巧

①string

```
//1.string 类重载运算符 operator>>用于输入，同样重载运算符 operator<<用于输出操作
string str1;
cin >> str1;//当用 cin>>进行字符串的输入的时候，遇到空格的地方就停止字符串的读取输入
cout << str1 << endl;
cin.get();//这个的作用就是读取 cin>>输入的结束符，不用对 getline 的输入产生影响！
getline(cin, str1);//字符串的行输入
cout << str1 << endl;


//2.string 类的构造函数
string str2 = "aaaaa";//最简单的字符串初始化
cout << str2 << endl;

char *s = "bbbbb";
string str3(s);//用 c 字符串 s 初始化
cout << str3 << endl;

char ch = 'c';
string str4(5, ch);//用 n 个字符 ch 初始化
cout << str4 << endl;

```

```
24  //3.string 类的字符操作
25  string str5 = "abcde";
26  ch = str5[3];//operator[]返回当前字符串中第 n 个字符的位置
27  cout << ch << endl;
28
29  string str6 = "abcde";
30  ch = str6.at(4);//at()返回当前字符串中第 n 个字符的位置,并且提供范围检查，当越界
31  时会抛出异常!
32  cout << ch << endl;
33
34  //4.string 的特性描述
35  string str7 = "abcdefgh";
36  int size;
37  size = str7.capacity();//返回当前容量
38  cout << size << endl;
39  size = str7.max_size();//返回 string 对象中可存放的最大字符串的长度
40  cout << size << endl;
41  size = str7.size();//返回当前字符串的大小
42  cout << size << endl;
43  size = str7.length();//返回当前字符串的长度
44  cout << size << endl;
45  bool flag;
46  flag = str7.empty();//判断当前字符串是否为空
47  cout << flag << endl;
48  int len = 10;
49  str7.resize(len, ch);//把字符串当前大小置为 len，并用字符 ch 填充不足的部分
50  cout << str7 << endl;
51
52  //5.string 的赋值
53  string str8;
54  str8 = str7;//把字符串 str7 赋给当前字符串
55  cout << str8 << endl;
56  str8.assign(str7);//把字符串 str7 赋给当前字符串
57  cout << str8 << endl;
58  str8.assign(s);//用 c 类型字符串 s 赋值
59  cout << str8 << endl;
60  str8.assign(s, 2);//用 c 类型字符串 s 开始的 n 个字符赋值
61  cout << str8 << endl;
62  str8.assign(len, ch);//用 len 个字符 ch 赋值给当前字符串
63  cout << str8 << endl;
64  str8.assign(str7, 0, 3);//把字符串 str7 中从 0 开始的 3 个字符赋给当前字符串
65  cout << str8 << endl;
66  string str9 = "0123456789";
67  str8.assign(str9.begin(), str9.end());//把迭代器之间的字符赋给字符串
68  cout << str8 << endl;
69
70  //6.string 的连接
71  string str10;
72  str10 += str9;//把字符串 str9 连接到当前字符串的结尾
```

```
73  cout << str10 << endl;
74  str10.append(s);//把 c 类型字符串 s 连接到当前字符串的结尾
75  cout << str10 << endl;
76  str10.append(s, 2);//把 c 类型字符串 s 的前 2 个字符连接到当前字符串的结尾
77  cout << str10 << endl;
78  str10.append(str9.begin(), str9.end());//把迭代器之间的一段字符连接到当前字符
79  串的结尾
80  cout << str10 << endl;
81  str10.push_back('k');//把一个字符连接到当前字符串的结尾
82  cout << str10 << endl;
83
84  //7.string 的比较
85  flag = (str9 == str10);//判断两个字符串是否相等
86  cout << flag << endl;
87  flag = (str9 != str10);//判断两个字符串是否不相等
88  cout << flag << endl;
89  flag = (str9 > str10);//判断两个字符串是否大于关系
90  cout << flag << endl;
91  flag = (str9 < str10);//判断两个字符串是否为小于关系
92  cout << flag << endl;
93  flag = (str9 >= str10);//判断两个字符串是否为大于等于关系
94  cout << flag << endl;
95  flag = (str9 <= str10);//判断两个字符串否为小于等于关系
96  cout << flag << endl;
97
98  //以下的 3 个函数同样适用于 c 类型的字符串，在 compare 函数中>时返回 1，<时返回-1，=
99  时返回 0
100 flag = str10.compare(str9);//比较两个字符串的大小，通过 ASCII 的相减得出!
101 cout << flag << endl;
102 flag = str10.compare(6, 12, str9);//比较 str10 字符串从 6 开始的 12 个字符组成的
103 字符串与 str9 的大小
104 cout << flag << endl;
105 flag = str10.compare(6, 12, str9, 3, 5);//比较 str10 字符串从 6 开始的 12 个字符
106 组成的字符串与 str9 字符串从 3 开始的 5 个字符组成的字符串的大小
107 cout << flag << endl;
108
109 //8.string 的字串
110 string str11;
111 str11 = str10.substr(10, 15);//返回从下标 10 开始的 15 个字符组成的字符串
112 cout << str11 << endl;
113
114 //9.string 的交换
115 str11.swap(str10);//交换 str11 与 str10 的值
116 cout << str11 << endl;
117
118 //10.string 的查找，查找成功时返回所在位置，失败时返回 string::npos 的值，即是-1
119 string str12 = "abcdefghijklmnopqrstuvwxyz";
120 int pos;
121 pos = str12.find('i', 0);//从位置 0 开始查找字符 i 在当前字符串的位置
```

```
122 cout << pos << endl;
123 pos = str12.find("ghijk", 0);//从位置 0 开始查找字符串“ghijk”在当前字符串的位置
124 cout << pos << endl;
125 pos = str12.find("opqrstuvw", 0, 4);//从位置 0 开始查找字符串“opqrstuvw”前 4 个
126 字符组成的字符串在当前字符串中的位置
127 cout << pos << endl;
128 pos = str12.rfind('s', string::npos);//从字符串 str12 反向开始查找字符 s 在字符
129 串中的位置
130 cout << pos << endl;
131 pos = str12.rfind("klmn", string::npos);//从字符串 str12 反向开始查找字符串
132 “klmn”在字符串中的位置
133 cout << pos << endl;
134 pos = str12.rfind("opqrstuvw", string::npos, 3);//从 string::pos 开始从后向前
135 查找字符串 s 中前 n 个字符组成的字符串在当前串中的位置
136 cout << pos << endl;
137
138 string str13 =
139 "aaaabbbbccccdddeeefffggghhhiiijjjkkllmmmandjfaklsdfpopdtwptioczx";
140 pos = str13.find_first_of('d', 0);//从位置 0 开始查找字符 d 在当前字符串第一次出
141 现的位置
142 cout << pos << endl;
143 pos = str13.find_first_of("eefff", 0);//从位置 0 开始查找字符串“eeefff“在当前
144 字符串中第一次出现的位置
145 cout << pos << endl;
146 pos = str13.find_first_of("efff", 0, 3);//从位置 0 开始查找当前串中第一个在字
147 符串”efff“的前 3 个字符组成的数组里的字符的位置
148 cout << pos << endl;
149 pos = str13.find_first_not_of('b', 0);//从当前串中查找第一个不在串 s 中的字符出
150 现的位置
151 cout << pos << endl;
152 pos = str13.find_first_not_of("abcdefghij", 0);//从当前串中查找第一个不在串 s
153 中的字符出现的位置
154 cout << pos << endl;
155 pos = str13.find_first_not_of("abcdefghij", 0, 3);//从当前串中查找第一个不在
156 由字符串”abcdefghij”的前 3 个字符所组成的字符串中的字符出现的位置
157 cout << pos << endl;
158 //下面的 last 的格式和 first 的一致，只是它从后面检索！
159 pos = str13.find_last_of('b', string::npos);
160 cout << pos << endl;
161 pos = str13.find_last_of("abcdef", string::npos);
162 cout << pos << endl;
163 pos = str13.find_last_of("abcdef", string::npos, 2);
164 cout << pos << endl;
165 pos = str13.find_last_not_of('a', string::npos);
166 cout << pos << endl;
167 pos = str13.find_last_not_of("abcdef", string::npos);
168 cout << pos << endl;
169 pos = str13.find_last_not_of("abcdef", string::npos, 3);
170 cout << pos << endl;
```

```
171
172 //11.string 的替换
173 string str14 = "abcdefghijklmn";
174 str14.replace(0, 3, "qqqq");//删除从 0 开始的 3 个字符，然后在 0 处插入字符串
175 “qqqq”
176 cout << str14 << endl;
177 str14.replace(0, 3, "vvvv", 2);//删除从 0 开始的 3 个字符，然后在 0 处插入字符串
178 “vvvv”的前 2 个字符
179 cout << str14 << endl;
180 str14.replace(0, 3, "opqrstuvw", 2, 4);//删除从 0 开始的 3 个字符，然后在 0 处插
181 入字符串“opqrstuvw”从位置 2 开始的 4 个字符
182 cout << str14 << endl;
183 str14.replace(0, 3, 8, 'c');//删除从 0 开始的 3 个字符，然后在 0 处插入 8 个字符 c
184 cout << str14 << endl;
185 //上面的位置可以换为迭代器的位置，操作是一样的，在这里就不再重复了！
186
187 //12.string 的插入，下面的位置处亦可以用迭代器的指针表示，操作是一样的
188 string str15 = "abcdefg";
189 str15.insert(0, "mnop");//在字符串的 0 位置开始处，插入字符串“mnop”
190 cout << str15 << endl;
191 str15.insert(0, 2, 'm');//在字符串的 0 位置开始处，插入 2 个字符 m
192 cout << str15 << endl;
193 str15.insert(0, "uvwxy", 3);//在字符串的 0 位置开始处，插入字符串“uvwxy”中的前
194 3 个字符
195 cout << str15 << endl;
196 str15.insert(0, "uvwxy", 1, 2);//在字符串的 0 位置开始处，插入从字符串“uvwxy”的
197 1 位置开始的 2 个字符
198 cout << str15 << endl;
199
200 //13.string 的删除
201 string str16 = "gfedcba";
202 string::iterator it;
203 it = str16.begin();
204 it++;
205 str16.erase(it);//删除 it 指向的字符，返回删除后迭代器的位置
206 cout << str16 << endl;
207 str16.erase(it, it + 3); //删除 it 和 it+3 之间的所有字符，返回删除后迭代器的位
208 置
209 cout << str16 << endl;
210 str16.erase(2);//删除从字符串位置 3 以后的所有字符，返回位置 3 前面的字符
211 cout << str16 << endl;
212
213 //14.字符串的流处理
214 string str17("hello,this is a test");
215 istringstream is(str17);
216 string s1, s2, s3, s4;
217 is >> s1 >> s2 >> s3 >> s4; //s1="hello,this",s2="is",s3="a",s4="test"
218 ostringstream os;
219 os << s1 << s2 << s3 << s4;
```

```
220  cout << os.str() << endl;
```

②bitset

```
1   //bitset 使用整数初始化 bitset
2   bitset<3> bs(7);
3   //输出 bs 各个位的值
4   cout << "bs[0] is " << bs[0] << endl;
5   cout << "bs[1] is " << bs[1] << endl;
6   cout << "bs[2] is " << bs[2] << endl;
7   //下面的语句会抛出 outofindexexception
8   //cout<<"bs[3] is "<<bs[3]<<endl;
9
10  //使用字符串初始化 bitset
11  //注意：使用 string 初始化时从右向左处理，如下初始化的各个位的值将是 110，而非 011
12  string strVal("011");
13  bitset<3> bs1(strVal);
14  //输出各位
15  cout << "bs1[0] is " << bs1[0] << endl;
16  cout << "bs1[1] is " << bs1[1] << endl;
17  cout << "bs1[2] is " << bs1[2] << endl;
18  //cout 输出时也是从右边向左边输出
19  cout << bs1 << endl;
20
21  //bitset 的方法
22  //any()方法如果有一位为 1，则返回 1
23  cout << "bs1.any() = " << bs1.any() << endl;
24
25  //none()方法，如果有一个为 1none 则返回 0，如果全为 0 则返回 1
26  bitset<3> bsNone;
27  cout << "bsNone.none() = " << bsNone.none() << endl;
28
29  //count()返回几个位为 1
30  cout << "bs1.count() = " << bs1.count() << endl;
31
32  //size()返回位数
33  cout << "bs1.size() = " << bs1.size() << endl;
34
35  //test()返回某一位是否为 1
36  //flip()诸位取反
37  bitset<3> bsFlip = bs1.flip();
38  cout << "bsFlip = " << bsFlip << endl;
39
40  //to_ulong
41  unsigned long val = bs1.to_ulong();
42  cout << val;
```

4.二维 RMQ

```
1   int num[maxn][maxn];
2   int st[maxn][maxn][9][9];
3   void build_st(int n, int m) {
```

```
        for (int i = 1; i <= n ; ++i)
                for (int j = 1; j <= m ; ++j)
                        st[i][j][0][0] = num[i][j];
        for (int g = 0; g < 10 ; ++g) {
                for (int t = 0; t < 10 ; ++t) {
                        if(g == 0 && t == 0)
                                continue;
                        for (int i = 1; i <= n; ++i) {
                                for (int j = 1; j <= m ; ++j) {
                                        if(i + (1 << g) - 1 > n || j + (1 << t)
- 1 > m)
                                                break;
                                        if(g == 0)
                                                st[i][j][g][t] =
max(st[i][j][g][t - 1], st[i][j + (1 << (t - 1))][g][t - 1]);
                                        else
                                                st[i][j][g][t] = max(st[i][j][g -
1][t], st[i + (1 << (g - 1))][j][g - 1][t]);
                                }
                        }
                }
        }
}
int query(int a, int b, int c, int d) {
        int k = (c - a + 1);
        int t = (d - b + 1);
        k = log2(1.0 * k) + eps;
        t = log2(1.0 * t) + eps;
        return max(max(st[a][b][k][t], st[c - (1 << k) + 1][b][k][t]),
max(st[a][d - (1 << t) + 1][k][t], st[c - (1 << k) + 1][d - (1 << t) +
1][k][t]));
}
```

5.背包

```
int dp[maxn];
//0-1 背包，代价为 cost,获得的价值为 weight
void ZeroOnePack(int cost, int weight, int nValue) {
        for (int i = nValue; i >= cost; i--)
                dp[i] = max(dp[i], dp[i - cost] + weight);
}
//完全背包，代价为 cost,获得的价值为 weight
void CompletePack(int cost, int weight, int nValue) {
        for (int i = cost; i <= nValue; i++)
                dp[i] = max(dp[i], dp[i - cost] + weight);
}
//多重背包
void MultiplePack(int cost, int weight, int amount, int nValue) {
        if (cost * amount >= nValue)
                CompletePack(cost, weight, nValue);
```

```
        else {
                int k = 1;
                while (k < amount) {
                        ZeroOnePack(k * cost, k * weight, nValue);
                        amount -= k;
                        k <<= 1;
                }
                ZeroOnePack(amount * cost, amount * weight, nValue);
        }
}
```

6.莫队

①线性莫队

```
int a[maxn], pos[maxn], cnt[maxm];
ll ans;
struct Query
{
        int l, r, id;
        ll ans;
} Q[maxn];
bool cmp1(const Query& a, const Query& b) {
        return a.id < b.id;
}
bool cmp2(const Query& a, const Query& b) {
        return pos[a.l] < pos[b.l] || (pos[a.l] == pos[b.l] && a.r < b.r);
}
inline void modify(int p, bool op) {
        if (op == 1) {
                cnt[a[p]]++;
                ans += (2LL * cnt[a[p]] - 1) * a[p];
        } else if (op == 0) {
                cnt[a[p]]--;
                ans -= (2LL * cnt[a[p]] + 1) * a[p];
        }
}
inline void solve() {
        int n, t;
        scanf("%d%d", &n, &t);
        rep(i, 1, n + 1) scanf("%d", &a[i]);
        memset(cnt, 0, sizeof cnt);
        int block = sqrt(n);
        rep(i, 1, n + 1) pos[i] = (i - 1) / block + 1;
        rep(i, 0, t) {
                scanf("%d%d", &Q[i].l, &Q[i].r);
                Q[i].id = i;
        }
        sort(Q, Q + t, cmp2);
        ans = 0;
        int L = 1, R = 1;
```

```
        rep(i, 0, t) {
                int l = Q[i].l, r = Q[i].r;
                while (L < l) modify(L, 0), L++;
                while (L > l) modify(L - 1, 1), L--;
                while (R <= r) modify(R, 1), R++;
                while (R > r + 1) modify(R - 1, 0), R--;
                Q[i].ans = ans;
        }
        sort(Q, Q + t, cmp1);
        rep(i, 0, t) printf("%lld\n", Q[i].ans);
}
```

②树上莫队

7.搜索

①A*

②IDA*

③DancingLinks

三、数据结构

1.线段树

注意：这里每个节点存了左和右边界，在**极限情况**可以省略。

①结构体版本

```
//这是一个区间加和 RMQ
class SegmentTree {
public:
#define lson (root << 1)
#define rson (root << 1 | 1)
#define lent (t[root].r - t[root].l + 1)
#define lenl (t[lson].r - t[lson].l + 1)
#define lenr (t[rson].r - t[rson].l + 1)
        struct Tree {
                int l, r, val, lazy;
        } t[maxn << 4];

        void pushup(int root) {
                t[root].val = t[lson].val + t[rson].val;
        }

        void pushdown(int root) {
                if (t[root].lazy) {
                        t[lson].lazy += t[root].lazy;
                        t[rson].lazy += t[root].lazy;
                        t[lson].val += lenl * t[root].lazy;
                        t[rson].val += lenr * t[root].lazy;
                        t[root].lazy = 0;
                }
        }
```

```
26
27          void build(int l, int r, int root) {
28                  t[root].l = l;
29                  t[root].r = r;
30                  t[root].lazy = 0;
31                  if (l == r) {
32                          t[root].val = 0;
33                          return;
34                  }
35                  int mid = l + r >> 1;
36                  build(l, mid, lson);
37                  build(mid + 1, r, rson);
38                  pushup(root);
39          }
40
41          void update(int l, int r, int val, int root) {
42                  if (l <= t[root].l && t[root].r <= r) {
43                          t[root].val += lent * val;
44                          t[root].lazy += val;
45                          return;
46                  }
47                  pushdown(root);
48                  int mid = t[root].l + t[root].r >> 1;
49                  if (l <= mid) update(l, r, val, lson);
50                  if (r > mid) update(l, r, val, rson);
51                  pushup(root);
52          }
53
54          int query(int l, int r, int root) {
55                  if (l <= t[root].l && t[root].r <= r)
56                          return t[root].val;
57                  pushdown(root);
58                  int mid = t[root].l + t[root].r >> 1;
59                  int ans = 0;
60                  if (l <= mid) ans += query(l, r, lson);
61                  if (r > mid) ans += query(l, r, rson);
62                  return ans;
63          }
64  #undef lenr
65  #undef lenl
66  #undef lent
67  #undef rson
68  #undef lson
69  };
```

②数组版本（占坑）

2.树状数组

①一维树状数组

```
1   int c[maxn];
```

```
inline int lowbit(int x) { return x & (-x); }
inline void modify(int x, int y, int data) {
	while (x <= y) {
		c[x] += data;
		x += lowbit(x);
	}
}
inline int query(int x) {
	int sum = 0;
	while (x > 0) {
		sum += c[x];
		x -= lowbit(x);
	}
	return sum;
}
```

②二维树状数组——记得更新四个点

```
class BTree {
	ll a[N][N];
	int n;
	void init() {
		n = 0;
		memset(a, 0, sizeof a);
	}
	void lowbit(int x) {
		return x & (-x);
	}
	void modify(int x, int y, int d) {
		for (int i = x; i <= n; i += lowbit(i))
			for (int j = y; j <= n; j += lowbit(j))
				a[i][j] += d;
	}
	ll query(int x, int y) {
		ll res = 0;
		for (int i = x; i >= 0; i -= lowbit(i))
			for (int j = y; j >= 0; j -= lowbit(j))
				res += a[i][j];
		return res;
	}
};
```

3.树链剖分

①点权式

```
// 点权式 - 配合线段树的单点修改+区间查询
struct Edge {
	int to, next;
	Edge() {}
	Edge(int a, int b) { to = a; next = b; }
} E[maxn << 1];
int head[maxn], cnt, tot;
```

```
int top[maxn], son[maxn], size[maxn], deep[maxn], pa[maxn], id[maxn];
int a[maxn];
void init() {
	memset(head, -1, sizeof head);
	tot = cnt = 0;
}
void addedge(int u, int v) {
	E[cnt].to = v;
	E[cnt].next = head[u];
	head[u] = cnt++;
}
void dfs1(int u, int fa, int d) {
	size[u] = 1; deep[u] = d; son[u] = 0;
	for (int i = head[u]; ~i; i = E[i].next) {
		int v = E[i].to;
		if (v != fa) {
			dfs1(v, u, d + 1);
			pa[v] = u;
			size[u] += size[v];
			if (size[v] > size[son[u]]) son[u] = v;
		}
	}
}
void dfs2(int u, int first) {
	top[u] = first;
	id[u] = ++tot;
	if (son[u]) dfs2(son[u], first);
	for (int i = head[u]; ~i; i = E[i].next) {
		int v = E[i].to;
		if (v != pa[u] && v != son[u]) dfs2(v, v);
	}
}
void solve(int u, int v) { //注意改的是这里
	int x = top[u], y = top[v], res = 0;
	while (x != y) {
		if (deep[x] < deep[y]) {
			swap(u, v);
			swap(x, y);
		}
		res += T.query(id[x], id[u], 1);
		u = pa[x];
		x = top[u];
	}
	if (deep[u] > deep[v]) swap(u, v);
	res += T.query(id[u], id[v], 1);
	printf("%d\n", res);
}
```

②边权式

```
// 边权式：把边权偏移到深度较大的点，化成点权式
```

```
struct Point {
	int from, to, val;
} p[maxn];
struct Edge {
	int to, next;
} E[maxn << 1];
int head[maxn], cnt, tot;
int top[maxn], son[maxn], size[maxn], deep[maxn], pa[maxn], id[maxn];
int a[maxn];
void init() {
	memset(head, -1, sizeof head);
	tot = cnt = 0;
}
void addedge(int u, int v) {
	E[cnt].to = v;
	E[cnt].next = head[u];
	head[u] = cnt++;
}
void dfs1(int u, int fa, int d) {
	size[u] = 1; deep[u] = d; son[u] = 0;
	for (int i = head[u]; ~i; i = E[i].next) {
		int v = E[i].to;
		if (v != fa) {
			dfs1(v, u, d + 1);
			pa[v] = u;
			size[u] += size[v];
			if (size[v] > size[son[u]]) son[u] = v;
		}
	}
}
void dfs2(int u, int first) {
	top[u] = first;
	id[u] = ++tot;
	if (son[u]) dfs2(son[u], first);
	for (int i = head[u]; ~i; i = E[i].next) {
		int v = E[i].to;
		if (v != pa[u] && v != son[u]) dfs2(v, v);
	}
}
int solve(int u, int v) {
	int x = top[u], y = top[v];
	int ans = 0;
	while (x != y) {
		if (deep[x] < deep[y]) {
			swap(u, v);
			swap(x, y);
		}
		ans += T.query(id[x], id[u], 1);
		u = pa[x];
```

```
                x = top[u];
        }
        //      printf("%d : %d - %d : ", ans, u, v);
        if (deep[u] > deep[v])
                swap(u, v);
        if (u != v) ans += T.query(id[son[u]], id[v], 1);
        return ans;
}
```

5.Treap

```
class Treap{
public:
    struct treap_point{
        int l,r,x,rep,son,rnd;
    }t[N];
    int treesz,root;
    inline void update(int k){t[k].son=t[t[k].l].son+t[t[k].r].son+t[k].rep;}
    inline void right_rotate(int &k){int
tt=t[k].l;t[k].l=t[tt].r;t[tt].r=k;update(k);update(tt);k=tt;}
    inline void left_rotate(int &k){int
tt=t[k].r;t[k].r=t[tt].l;t[tt].l=k;update(k);update(tt);k=tt;}
    inline void insert_to_tree(int &k,int x)
    {
        if
(!k){k=++treesz;t[k].x=x;t[k].rnd=rand();t[k].son=t[k].rep=1;t[k].l=t[k].r=0;ret
        t[k].son++;
        if (x==t[k].x){t[k].rep++;return;}
        if (x<t[k].x){insert_to_tree(t[k].l,x);if
(t[t[k].l].rnd<t[k].rnd)right_rotate(k);}
        if (x>t[k].x){insert_to_tree(t[k].r,x);if
(t[t[k].r].rnd<t[k].rnd)left_rotate(k);}
    }
    inline bool delete_from_tree(int &k,int x)
    {
        if (!k)return 0;
        if (x==t[k].x)
        {
            if (t[k].rep>1){t[k].rep--;t[k].son--;return 1;}
            if (!t[k].l||!t[k].r){k=t[k].l+t[k].r;return 1;}
            if (t[t[k].l].rnd<t[t[k].r].rnd){right_rotate(k);return
delete_from_tree(k,x);}
            left_rotate(k);return delete_from_tree(k,x);
        }
        int res;
        if (x<t[k].x){res=delete_from_tree(t[k].l,x);if (res)t[k].son--;return re
        res=delete_from_tree(t[k].r,x);if (res)t[k].son--;return res;
    }
    inline int get_succ_in_tree(int k,int x)
    {
```

```
        if (!k)return -1;
        int sv;
        if (x<t[k].x){sv=get_succ_in_tree(t[k].l,x);return sv==-1?t[k].x:sv;}
        return get_succ_in_tree(t[k].r,x);
    }
    inline int get_pred_in_tree(int k,int x)
    {
        if(!k)return -1;
        int sv;
        if (x>t[k].x){sv=get_pred_in_tree(t[k].r,x);return sv==-1?t[k].x:sv;}
        return get_pred_in_tree(t[k].l,x);
    }
    inline int ask_kth_in_tree(int k,int x)
    {
        if (!k)return -1;
        if (x<=t[t[k].l].son)return ask_kth_in_tree(t[k].l,x);
        if (x>t[t[k].l].son+t[k].rep)return ask_kth_in_tree(t[k].r,x-t[t[k].l].so
t[k].rep);
        return t[k].x;
    }
    inline int ask_rank_in_tree(int k,int x)
    {
        if (!k)return -1;
        if (x==t[k].x)return t[t[k].l].son+1;
        if (x<t[k].x)return ask_rank_in_tree(t[k].l,x);
        return t[t[k].l].son+t[k].rep+ask_rank_in_tree(t[k].r,x);
    }
    /*
    inline void bianli(int k)//for debug
    {
        if (!k)return;
        bianli(t[k].l);
        printf("i=%d l=%d r=%d x=%d rep=%d
son=%d\n",k,t[k].l,t[k].r,t[k].x,t[k].rep,t[k].son);
        bianli(t[k].r);
    }
    inline void put(){puts("\n----------------------------------------
");bianli(root);puts("----------------------------------------\n");}//for debug
    */
    inline void init(){treesz=root=0;}
    inline void insert(int x){insert_to_tree(root,x);}
    inline int del(int x){return delete_from_tree(root,x);}
    inline int succ(int x){return get_succ_in_tree(root,x);}
    inline int pred(int x){return get_pred_in_tree(root,x);}
    inline int ask_kth(int x){return ask_kth_in_tree(root,x);}
    inline int ask_rank(int x){return ask_rank_in_tree(root,x);}
}T;
int main()
{
```

```
    int n,x,op;read(n);
    T.init();
    for (int i=1;i<=n;i++)
    {
        read(op);read(x);
        if (op==1)T.insert(x);
        if (op==2)T.del(x);
        if (op==3)printf("%d\n",T.ask_rank(x));
        if (op==4)printf("%d\n",T.ask_kth(x));
        if (op==5)printf("%d\n",T.pred(x));
        if (op==6)printf("%d\n",T.succ(x));
    }
};
```

6.Splay

7.分块

8.块状链表

9.可持久化数据结构

①可持久化线段树
②可持久化并查集

四、字符串

1.Kmp

①Char 版本

```
char n[N], m[N];
int f[N];
void compute(char *P) {
        int len = strlen(P);
        memset(f, 0, sizeof(f));
        for (int i = 1; i < len; i++) {
                int j = f[i];
                while (j && P[i] != P[j])
                        j = f[j];
                f[i + 1] = (P[i] == P[j]) ? j + 1 : 0;
        }
}
bool kmp(char *T, char *P) {
        int n = strlen(T);
        int m = strlen(P);
        compute(P);
        int q = 0;
        for (int i = 0; i < n; i++) {
                while (q && P[q] != T[i])
                        q = f[q];
```

```
                if (P[q] == T[i])
                        q++;
                if (q == m)
                        return 1;       //修改这里可以实现很多功能
        }
        return 0;
}
```

②String 版本

```
string n, m;
int f[maxn];
void compute(string &P) {
        int len = P.length();
        memset(f, 0, sizeof(f));
        for (int i = 1; i < len; i++) {
                int j = f[i];
                while (j && P[i] != P[j])
                        j = f[j];
                f[i + 1] = (P[i] == P[j]) ? j + 1 : 0;
        }
}
bool kmp(string &T, string &P) {
        int n = T.length();
        int m = P.length();
        compute(P);
        int q = 0;
        for (int i = 0; i < n; i++) {
                while (q && P[q] != T[i])
                        q = f[q];
                if (P[q] == T[i])
                        q++;
                if (q == m)
                        return 1;
        }
        return 0;
}
```

③扩展 kmp

2.Trie

```
struct Trie {
#define sigma_size 26
        int ch[maxn][sigma_size];
        int val[maxn];
        int sz;

        void init() {
                sz = 1;
                memset(ch[0], 0, sizeof ch[0]);
```

```
        }

        int idx(char c) {
                return c - 'a';
        }

        void insert(char *s, int v) {
                int u = 0, len = strlen(s);
                rep(i, 0, len) {
                        int c = idx(s[i]);
                        if (!ch[u][c]) {
                                memset(ch[sz], 0, sizeof ch[sz]);
                                val[sz] = 0;
                                ch[u][c] = sz++;
                        }
                        u = ch[u][c];
                }
                val[u] = v;
        }
#undef sigma_size
};
```

3.Manacher

```
/*
 *以 char 方式储存字符串，变更字符串为$#*#*#*#*#*#形式，所以计算应该从 1 开始
 *每个字符的 p 值大小代表该字符可以扩展的大小也打表回文串长度+1
 *如果 p 值为奇数，代表此字符左右为中心的字符串为回文串的长度+1
 *如果 p 值为偶数，代表此字符为中心扩展的字符的长度
 */
int p[maxn * 2 + 10];
char re[maxn * 2];
char s[maxn];
void manacher(char str[], int len) {
        //memset(p,0,sizeof(p));
        int mx = 0, id = 0;
        int ans = 1;
        for(int i = 1; i <= len; i++) {
                if(mx > i)
                        p[i] = (p[2 * id - i] < (mx - i)) ? p[2 * id - i] :
(mx - i);
                else p[i] = 1;
                while(str[i - p[i]] == str[i + p[i]]) p[i]++;
                if(i + p[i] > mx) {
                        mx = i + p[i];
                        id = i;
                }
                ans = max(ans, p[i] - 1);
        }
        printf("%d\n", ans);
```

```
}
// 字符串变更
void solve(char str[]) {
	re[0] = '$'; re[1] = '#';
	int len = strlen(str + 1);
	for(int i = 1 ; i <= len; i++) {
		re[i * 2] = str[i];
		re[i * 2 + 1] = '#';
	}
	manacher(re, len * 2 + 1);
}
```

4.AC 自动机

```
const int AC_NOD = 28, AC_N = 5e5 + 10;
struct Tire {
	int nxt[AC_N][AC_NOD], fail[AC_N], cnt[AC_N];
	int root , tot;
	int id[AC_NOD];
	int end[AC_N];
	int newnode() {
		for(int i = 0; i < AC_NOD ; i++)nxt[tot][i] = -1;
		fail[tot] = -1;
		cnt[tot] = 0;
		return tot++;
	}
	void init() {
		tot = 0;
		memset(end, false, sizeof end);
		root = newnode();
	}
	int getid(char x) {
		if(x <= 'z' && x >= 'a') {
			return x - 'a' + 10;
		} else if(x <= 'Z' && x >= 'A') {
			return 36 + x - 'A';
		} else return x - '0';
	}


	void insert(string s) {
		int p = root, len = s.length();
		for(int i = 0; i < len; i++) {
			int cur = getid(s[i]);
			if(nxt[p][cur] == -1)nxt[p][cur] = newnode();
			p = nxt[p][cur];
		}
		cnt[p]++; end[p] = true;
	}
	void build() {
```

```
37                 queue<int> que;
38                 que.push(root);
39                 int now;
40 
41                 while(!que.empty()) {
42                         now = que.front(); que.pop();
43                         if(end[fail[now]])end[now] = true;
44                         for(int i = 0; i < AC_NOD; i++) {
45                                 if(~nxt[now][i]) {
46                                         if(now == root) fail[nxt[now][i]] =
47 root;
48                                         else fail[nxt[now][i]] =
49 nxt[fail[now]][i];
50                                         que.push(nxt[now][i]);
51                                 } else {
52                                         if(now == root) nxt[now][i] = root;
53                                         else nxt[now][i] = nxt[fail[now]][i];
54                                 }
55                         }
56                 }
57         }
58         int count(string t) {
59                 int now = root;
60                 int len = t.length();
61                 int p;
62                 int ans = 0;
63                 for(int i = 0; i < len; i++) {
64                         p = getid(t[i]);
65                         now = nxt[now][p];
66                         for(int tmp = now; tmp != root; tmp = fail[tmp]) {
67                                 ans += cnt[tmp];
68                         }
69                 }
70                 return ans;
71         }
72 } T;
73 int main() {
74         int t;
75         cin.sync_with_stdio(0);
76         cin >> t;
77         while(t--) {
78                 T.init();
79                 int n; cin >> n;
80                 string q; for(int i = 1; i <= n; i++)cin >> q, T.insert(q);
81                 T.build();
82                 cin >> q; cout << T.count(q) << endl;
83         }
84         return 0;
85 }
```

5.后缀数组

①Da

```
#include <bits/stdc++.h>
using namespace std;
/*n 为总字符串的长度
 *r 数组为被操作的字符串，索引从 0 开始，最后一位(r[n-1])一定是'$'(char 数组)||'0'(int 数组)
 *sa 为生成的后缀数组，从 0 开始到 n-1 结束，保证 sa[0] = n-1
 *m 代表字符集的大小或者说范围。
 *height[n-1]的值无效因为其没有前驱字符串
 *查询两个后缀的 lcp 应该查询 query(sa[l]+1,sa[r]);
 */
const int maxn = 500005;
char a[maxn];
int wa[maxn],wb[maxn],wv[maxn],wts[maxn];
int cmp(int *r,int a,int b,int l)
{
	return r[a] == r[b]&&r[a+l] == r[b+l];
}
void init_sa(char *r,int *sa,int n,int m)
{
	int i,j,*x = wa,*y = wb,*t;
	int p;
	for(i = 0;i < m;i ++) wts[i] = 0;
	for(i = 0;i < n;i ++) wts[ x[i] = r[i]]++;
	for(i = 0;i < m;i ++) wts[i] += wts[i-1];
	for(i = n-1;i >= 0; --i) sa[--wts[x[i]]] = i;
	for(j = 1,p = 1; p < n ; j <<= 1 , m = p )
	{

	for(p = 0,i = n - j; i < n; i ++) y[p++] = i;
		for(i = 0;i < n; i ++) if(sa[i] >= j) y[p++] = sa[i] - j;
		for(i = 0;i < n; i ++) wv[i] = x[y[i]];
		for(i = 0;i < m; i ++) wts[i] = 0;
		for(i = 0;i < n; i ++) wts[wv[i]]++;
	for(i = 1;i < m; i ++) wts[i] += wts[i-1];
		for(i = n-1;i >= 0;i--) sa[--wts[wv[i]]] = y[i];
	for(t = x,x = y,y = t,p = 1,x[sa[0]] = 0,i = 1;i < n; i++)
			x[sa[i]] = cmp(y,sa[i-1],sa[i],j)?p-1:p++;
	}

	return;
}
int rnk[maxn],height[maxn];
void calheight(char *r,int *sa,int n)
{
```

```
        int i,j,k = 0;
        for(i = 1;i < n;i++)rnk[sa[i]] = i;
        for(i = 0;i < n-1;height[rnk[i++]] = k)
        for(k?k--:0,j = sa[rnk[i]-1];r[i+k] == r[j+k];k++);
}
int sa[maxn];
void solve(int cur)
{
    init_sa(a,sa,len,m);
    calheight(a,sa,len);
}
```

②Dc3

6.后缀自动机

五、图论

1.最短路

①Floyd

```
#define inf 0x3fffffff      //注意，太大会溢出
#define M                           //最大点数
int n, dist[M][M];          //n：实际点数

void init ()                //有时候需要初始化
{
    int i, j;
    for (i = 1; i <= n; i++)
        for (j = i + 1; j <= n; j++)
            dist[i][j] = dist[j][i] = inf;
}

void floyd ()
{
    int i, j, k;
    for (k = 1; k <= n; k++)
        for (i = 1; i <= n; i++)
            for (j = 1; j <= n; j++)    //有的题目会溢出就要自己变通了
                if (dist[i][k] + dist[k][j] < dist[i][j])
                            dist[i][j] = dist[i][k] + dist[k][j];
}
```

②Dijkstra

- 普通：

```
#define inf 0x3fffffff
#define M 105
int dist[M], map[M][M], n;
bool mark[M];
```

```
void init () {
	int i, j;
	for (i = 1; i <= n; i++) //i==j 的时候也可以初始化为 0，只是有时候不合适
		for (j = 1; j <= n; j++)
			map[i][j] = inf;
}
void dijkstra (int u) {
	int i, j, mins, v;
	for (i = 1; i <= n; i++) {
		dist[i] = map[u][i];
		mark[i] = false;
	}
	mark[u] = true;
	dist[u] = 0;	//既然上面的 map 当 i==j 时不是 0，就要这句
	while (1) {
		mins = inf;
		for (j = 1; j <= n; j++)
			if (!mark[j] && dist[j] < mins)
				mins = dist[j], v = j;
		if (mins == inf)
			break;
		mark[v] = true;
		for (j = 1; j <= n; j++)
			if (!mark[j] && dist[v] + map[v][j] < dist[j])
				dist[j] = dist[v] + map[v][j];
	}
}
```

- 堆优化：

```
struct node {
	int to, val;
	friend bool operator< (const node &a, const node &b) {
		return a.val > b.val;
	}
};
vector<node> g[maxn];
int dis[maxn];
int dijkstra(int s, int t, int n) {
	fill(dis, dis + n + 1, INF);
	priority_queue<node> q;
	dis[s] = 0;
	q.push({s, 0});
	while (!q.empty()) {
		int u = q.top().to; q.pop();
		int sz = SZ(g[u]);
		rep(i, 0, sz) {
			int v = g[u][i].to, w = g[u][i].val;
			if (dis[v] > dis[u] + w)
				q.push({v, dis[v] = dis[u] + w});
```

```
                }
        }
        return dis[t];
}
```

③Spfa

- 普通队列:

```
bool vis[maxn];
vector<PII> g[maxn];
int dis[maxn];
inline void spfa(int s, int n) {
        queue<int> q;
        memset(vis, 0, sizeof vis);
        fill(dis, dis + n + 1, INF);
        dis[s] = 0; vis[s] = 1;
        q.push(s);
        while (!q.empty()) {
                int u = q.front(); q.pop();
                vis[u] = 0;
                rep(i, 0, SZ(g[u])) {
                        int v = g[u][i].fi, w = g[u][i].se;
                        if (dis[v] > dis[u] + w) {
                                dis[v] = dis[u] + w;
                                if (!vis[v]) {
                                        vis[v] = 1;
                                        q.push(v);
                                }
                        }
                }
        }
}
```

- 双端队列:

```
bool vis[maxn];
vector<PII> g[maxn];
int dis[maxn];
inline void spfa(int s, int n) {
        deque<int> q;
        memset(vis, 0, sizeof vis);
        fill(dis, dis + n + 1, INF);
        dis[s] = 0; vis[s] = 1;
        q.pb(s);
        while (!q.empty()) {
                int u = q.front(); q.pop_front();
                vis[u] = 0;
                rep(i, 0, SZ(g[u])) {
                        int v = g[u][i].fi, w = g[u][i].se;
                        if (dis[v] > dis[u] + w) {
                                dis[v] = dis[u] + w;
                                if (!vis[v]) {
```

```
                                        vis[v] = 1;
                                        if (!q.empty() && dis[v] <=
dis[q.front()]) q.push_front(v);
                                        else q.pb(v);
                                }
                        }
                }
        }
}
```

2.K 短路

3.差分约束

4.次小生成树

5.曼哈顿最小生成树

6.最小树形图

7.2-SAT

8.二分图匹配

①匈牙利算法

- 链表：

```
vector<int> g[maxn];
int match[maxn];
bool vis[maxn];
int n;
bool dfs(int u) {
        vis[u] = 1;
        int sz = g[u].size();
        rep(i, 0, sz) {
                int v = g[u][i];
                if (match[v] == -1 || !vis[v] && dfs(v)) {
                        match[v] = u;
                        match[u] = v;
                        return 1;
                }
        }
        return 0;
}
int bipartite_matching() {
        int ans = 0;
        memset(match, -1, sizeof match);
        rep(u, 1, n + 1) {
                if (match[u] == -1) {
                        memset(vis, 0, sizeof vis);
                        if (dfs(u))
```

```
                                ans++;
                        }
                }
        return ans;
}
```

- 邻接矩阵：

```
int g[N][N], match[N];
bool vis[N];
int n, m; // n 个点 m 个边
bool dfs(int u) {
        rep(v, 1, n + 1) {
                if (g[u][v] && !vis[v]) {
                        vis[v] = 1;
                        if (match[v] == -1 || dfs(match[v])) {
                                match[v] = u;
                                return 1;
                        }
                }
        }
        return 0;
}
int hungary() {
        int ans = 0;
        memset(match, -1, sizeof match);
        rep(u, 1, m + 1) {
                memset(vis, 0, sizeof vis);
                if (dfs(u))
                        ans++;
        }
        return ans;
}
```

9.二分图最大权匹配

```
int nx, ny; //两边的点数
int g[N][N];//二分图描述
int match[N], lx[N], ly[N]; //y 中各点匹配状态，x,y 中的点标号
int slack[N];
bool visx[N], visy[N];

bool DFS(int x) {
        visx[x] = true;
        for(int y = 0; y < ny; y++) {
                if(visy[y])continue;
                int tmp = lx[x] + ly[y] - g[x][y];
                if(tmp == 0) {
                        visy[y] = true;
                        if(match[y] == -1 || DFS(match[y])) {
                                match[y] = x;
```

```
                        return true;
                }
            } else if(slack[y] > tmp)
                slack[y] = tmp;
        }
        return false;
}
int KM() {
        memset(match, -1, sizeof(match));
        memset(ly, 0, sizeof(ly));
        for(int i = 0; i < nx; i++) {
            lx[i] = -INF;
            for(int j = 0; j < ny; j++)
                if(g[i][j] > lx[i])
                    lx[i] = g[i][j];
        }
        for(int x = 0; x < nx; x++) {
            for(int i = 0; i < ny; i++)
                slack[i] = INF;
            while(true) {
                memset(visx, false, sizeof(visx));
                memset(visy, false, sizeof(visy));
                if(DFS(x))break;
                int d = INF;
                for(int i = 0; i < ny; i++)
                    if(!visy[i] && d > slack[i])
                        d = slack[i];
                for(int i = 0; i < nx; i++)
                    if(visx[i])
                        lx[i] -= d;
                for(int i = 0; i < ny; i++) {
                    if(visy[i])ly[i] += d;
                    else slack[i] -= d;
                }
            }
        }
        int res = 0;
        for(int i = 0; i < ny; i++)
            if(match[i] != -1)
                res += g[match[i]][i];
        return res;
}
```

10.连通性

①强连通分量

```
/*
* tarjan:
* scnt——缩的点
* belong——每个点属于那个块
```

```
*/
struct Edge {
    int to, next;
} E[maxn << 1];
int head[maxn], cnt;
int dfn[maxn], low[maxn], s[maxn], belong[maxn], sec, top, scnt;
bool inS[maxn];
void init() {
    memset(head, -1, sizeof head);
    memset(dfn, 0, sizeof dfn);
    memset(low, 0, sizeof low);
    cnt = scnt = sec = top = 0;
}
void addedge(int u, int v) {
    E[cnt].to = v;
    E[cnt].next = head[u];
    head[u] = cnt++;
}
void tarjan(int u) {
    dfn[u] = low[u] = ++sec;
    inS[u] = 1;
    s[top++] = u;
    for (int i = head[u]; ~i; i = E[i].next) {
        int v = E[i].to;
        if (!dfn[v]) {
            tarjan(v);
            low[u] = min(low[u], low[v]);
        } else if (inS[v] && low[u] > dfn[v]) low[u] = dfn[v];
    }
    if (dfn[u] == low[u]) {
        ++scnt;
        int tmp;
        do {
            tmp = s[--top];
            inS[tmp] = 0;
            belong[tmp] = scnt;
        } while (tmp != u);
    }
}
```

②双连通分量

- 边双连通分量

```
/*
* tarjan
* block 边连通分量的数量
* belong 标记该点属于哪个连通分量
*/
int low[maxn], dfn[maxn];
bool inStack[maxn];
int cnt = 0;
```

```
stack<int> stk;
int bridge = 0;
int block = 0;
int belong[maxn];
struct edge {
	int to, nxt;
	bool cut;
} e[maxm];
int head[maxn], tot = 0;
void addEdge(int u, int v) {
	e[tot].to = v;
	e[tot].nxt = head[u];
	e[tot].cut = false;
	head[u] = tot++;
}
void tarjan(int u, int pre) {
	int v;
	low[u] = dfn[u] = ++cnt;
	stk.push(u);
	inStack[u] = true;
	for(int i = head[u]; i != -1; i = e[i].nxt) {
		int v = e[i].to;
		if(v == pre) continue;
		if(!dfn[v]) {
			tarjan(v, u);
			low[u] = min(low[u], low[v]);
			// bridge
			if(low[v] > dfn[u]) {
				bridge++;
				e[i].cut = true;
				e[i ^ 1].cut = true;
			}
		} else if(inStack[v] && low[u] > dfn[v]) {
			low[u] = dfn[v];
		}
	}
	if(low[u] == dfn[u]) {
		block++;
		do {
			v = stk.top();
			inStack[v] = false;
			stk.pop();
			belong[v] = block;
		} while(v != u);
	}
}
void gao(int n) {
	mem(dfn, 0); mem(low, 0);
	mem(inStack, false);
```

```
        cnt = 0;
        bridge = 0;
        block = 0;
        rep(i, 1, n + 1) {
                if(!dfn[i]) {
                        tarjan(i, i);
                }
        }
}
// multi-edge
map<int, int> mp;
inline bool isHash(int u, int v) {
        if(mp[u * maxn + v] || mp[v * maxn + u]) return true;
        mp[u * maxn + v] = mp[v * maxn + u] = 1;
        return false;
}
```

- 点双连通分量

```
/*
* tarjan
* block 点双连通分量数目
* belong 属于哪个分量 割点可能属于多个连通分量 所以割点要特殊判断
*/
int low[maxn], dfn[maxn];
bool inStack[maxn];
int cnt = 0;
stack<int> stk;
int bridge = 0;
int block = 0;
int belong[maxn];
struct edge {
        int to, nxt;
        bool cut;
} e[maxm];
int head[maxn], tot = 0;
void addEdge(int u, int v) {
        e[tot].to = v;
        e[tot].nxt = head[u];
        e[tot].cut = false;
        head[u] = tot++;
}
void tarjan(int u, int pre) {
        int v;
        low[u] = dfn[u] = ++cnt;
        stk.push(u);
        inStack[u] = true;
        for(int i = head[u]; i != -1; i = e[i].nxt) {
                int v = e[i].to;
                if(v == pre) continue;
                if(!dfn[v]) {
```

```
                        tarjan(v, u);
                        low[u] = min(low[u], low[v]);
                        // bridge
                        if(low[v] >= dfn[u]) {
                                block++;
                                int vv;
                                do {
                                        vv = stk.top();
                                        belong[vv] = block;
                                        inStack[vv] = false;
                                } while(vv != v);
                        }
                } else if(inStack[v] && low[u] > dfn[v]) {
                        low[u] = dfn[v];
                }
        }
}
void gao(int n) {
        mem(dfn, 0); mem(low, 0);
        mem(inStack, false);
        cnt = 0;
        bridge = 0;
        block = 0;
        rep(i, 1, n + 1) {
                if(!dfn[i]) {
                        tarjan(i, i);
                }
        }
}
// multi-edge
map<int, int> mp;
inline bool isHash(int u, int v) {
        if(mp[u * maxn + v] || mp[v * maxn + u]) return true;
        mp[u * maxn + v] = mp[v * maxn + u] = 1;
        return false;
}
```

③割点和桥：low[v]>dfn[u]

```
/*
* Tarjan O(n + m)
* bridge 桥的数量
* addBlock 删除该点后增加的连通块
*/
int low[maxn], dfn[maxn];
bool inStack[maxn];
int cnt = 0;
stack<int> stk;
int bridge = 0;
bool cut[maxn];
int addBlock[maxn];
```

```cpp
struct edge {
	int to, nxt;
	bool cut;
} e[maxm];
int head[maxn], tot = 0;
void addEdge(int u, int v) {
	e[tot].to = v;
	e[tot].nxt = head[u];
	e[tot].cut = false;
	head[u] = tot++;
}
void tarjan(int u, int pre) {
	int v;
	low[u] = dfn[u] = ++cnt;
	stk.push(u);
	inStack[u] = true;
	int son = 0;
	for(int i = head[u]; i != -1; i = e[i].nxt) {
		int v = e[i].to;
		if(v == pre) continue;
		if(!dfn[v]) {
			son++;
			tarjan(v, u);
			low[u] = min(low[u], low[v]);
			// bridge
			if(low[v] > dfn[u]) {
				bridge++;
				e[i].cut = true;
				e[i ^ 1].cut = true;
			}
			// cut point
			if(u != pre && low[v] >= dfn[u]) {
				cut[u] = true;
				addBlock[u]++;
			}
		} else if(low[u] > dfn[v]) {
			low[u] = dfn[v];
		}
	}
	// root
	if(u == pre && son > 1) cut[u] = true;
	if(u == pre) addBlock[u] = son - 1;
	inStack[u] = false;
	stk.pop();
}
void gao(int n) {
	mem(dfn, 0); mem(low, 0);
	mem(inStack, false);
	mem(addBlock, 0);
```

```
        mem(cut, false);
        cnt = 0;
        bridge = 0;
        rep(i, 1, n + 1) {
                if(!dfn[i]) {
                        tarjan(i, i);
                }
        }
}
// multi-edge
map<int, int> mp;
inline bool isHash(int u, int v) {
        if(mp[u * maxn + v] || mp[v * maxn + u]) return true;
        mp[u * maxn + v] = mp[v * maxn + u] = 1;
        return false;
}
```

11.网络流

①Dinic

```
struct Edge {
        int to, cap, next;
} E[maxn];
int head[N], pa[N], vis[N], cnt;
void init() {
        memset(head, -1, sizeof head);
        cnt = 0;
}
void addedge(int u, int v, int w) {
        E[cnt].to = v; E[cnt].cap = w; E[cnt].next = head[u]; head[u] =
cnt++;
        E[cnt].to = u; E[cnt].cap = 0; E[cnt].next = head[v]; head[v] =
cnt++; // 0 or w
}
bool bfs(int s, int t) {
        memset(vis, -1, sizeof vis);
        queue<int> q;
        vis[s] = 0;
        q.push(s);
        while (!q.empty()) {
                int u = q.front();
                q.pop();
                for (int i = head[u]; i != -1; i = E[i].next) {
                        int v = E[i].to;
                        if (E[i].cap && vis[v] == -1) {
                                vis[v] = vis[u] + 1;
                                q.push(v);
                        }
                }
        }
```

```
        return vis[t] != -1;
}
int dfs(int u, int t, int flow) {
        if (u == t) return flow;
        for (int &i = pa[u]; i != -1; i = E[i].next) {
                int v = E[i].to;
                if (E[i].cap && vis[v] == vis[u] + 1) {
                        int res = dfs(v, t, min(flow, E[i].cap));
                        if (res) {
                                E[i].cap -= res;
                                E[i ^ 1].cap += res;
                                return res;
                        }
                }
        }
        return 0;
}
int Dinic(int s, int t) {
        int max_flow = 0;
        while (bfs(s, t)) {
                memcpy(pa, head, sizeof head);
                int res;
                do {
                        res = dfs(s, t, INF);
                        max_flow += res;
                } while (res);
        }
        return max_flow;
}

```

②ISAP

```
 struct Edge {
        int from, to, cap, flow;
 };
 class ISAP {
 public:
 #define max_V 202
        int n, m, s, t;
        vector<Edge> edges;
        vector<int> g[max_V];
        bool vis[max_V];
        int d[max_V], cur[max_V], p[max_V], num[max_V];

        void Addedge(int u, int v, int w) {
                edges.pb((Edge) {
                        u, v, w, 0
                });
                edges.pb((Edge) {
                        v, u, 0, 0
```

```
19              });
20              m = SZ(edges);
21              g[u].pb(m - 2);
22              g[v].pb(m - 1);
23      }
24
25      bool Bfs() {
26              memset(vis, 0, sizeof vis);
27              queue<int> q;
28              q.push(t);
29              vis[t] = 1;
30              d[t] = 0;
31              while (!q.empty()) {
32                      int u = q.front(); q.pop();
33                      int sz = SZ(g[u]);
34                      rep(i, 0, sz) {
35                              Edge &E = edges[g[u][i] ^ 1];
36                              if (!vis[E.from] && E.cap > E.flow) {
37                                      vis[E.from] = 1;
38                                      d[E.from] = d[u] + 1;
39                                      q.push(E.from);
40                              }
41                      }
42              }
43              return vis[s];
44      }
45
46      void ClearAll(int n) {
47              this->n = n;
48              rep(i, 0, n) g[i].clear();
49              edges.clear();
50      }
51
52      void ClearFlow() {
53              int sz = SZ(edges);
54              rep(i, 0, sz) edges[i].flow = 0;
55      }
56
57      int Augment() {
58              int x = t, a = INF;
59              while (x != s) {
60                      Edge &E = edges[p[x]];
61                      a = min(a, E.cap - E.flow);
62                      x = edges[p[x]].from;
63              }
64              x = t;
65              while (x != s) {
66                      edges[p[x]].flow += a;
67                      edges[p[x] ^ 1].flow -= a;
```

```
                x = edges[p[x]].from;
        }
        return a;
    }

    int MaxFlow(int s, int t) {
        this->s = s; this->t = t;
        int flow = 0;
        Bfs();
        memset(num, 0, sizeof num);
        rep(i, 0, n) num[d[i]]++;
        int x = s;
        memset(cur, 0, sizeof cur);
        while (d[s] < n) {
            if (x == t) {
                flow += Augment();
                x = s;
            }
            bool ok = 0;
            int sz = SZ(g[x]);
            rep(i, cur[x], sz) {
                Edge &E = edges[g[x][i]];
                if (E.cap > E.flow && d[x] == d[E.to] + 1) {
                    ok = 1;
                    p[E.to] = g[x][i];
                    cur[x] = i;
                    x = E.to;
                    break;
                }
            }
            if (!ok) {
                int m = n - 1;
                rep(i, 0, sz) {
                    Edge &E = edges[g[x][i]];
                    if (E.cap > E.flow)
                        m = min(m, d[E.to]);
                }
                if (--num[d[x]] == 0) break; // gap
                num[d[x] = m + 1]++;
                cur[x] = 0;
                if (x != s) x = edges[p[x]].from;
            }
        }
        return flow;
    }

    vector<int> Mincut() {
        Bfs();
        vector<int> ans;
```

```
                int sz = SZ(edges);
                rep(i, 0, sz) {
                        Edge &E = edges[i];
                        if (!vis[E.from] && vis[E.to] && E.cap > 0)
                                ans.pb(i);
                }
                return ans;
        }

        void Reduce() {
                int sz = SZ(edges);
                rep(i, 0, sz) {
                        edges[i].cap -= edges[i].flow;
                }
        }

        void Print() {
                puts("Graph:");
                int sz = SZ(edges);
                rep(i, 0, sz) {
                        printf("%d->%d, %d, %d\n", edges[i].from,
edges[i].to, edges[i].cap, edges[i].flow);
                }
        }
#undef max_V
};
```

12.费用流

```
struct Edge {
        int from, to, cap, flow, cost;
        Edge(int a, int b, int c, int d, int E) { from = a; to = b; cap = c;
flow = d; cost = E; }
};
class MCMF {
public:
#define max_V 202
        int n, m, s, t;
        vector<Edge> edges;
        vector<int> g[max_V];
        bool vis[max_V];
        int d[max_V], p[max_V], a[max_V];

        void Init(int n) {
                this->n = n;
                rep(i, 0, n) g[i].clear();
                edges.clear();
        }

        void Addedge(int u, int v, int c, int w) {
```

```
22                edges.pb(Edge(u, v, c, 0, w));
23                edges.pb(Edge(v, u, 0, 0, -w));
24                int m = SZ(edges);
25                g[u].pb(m - 2);
26                g[v].pb(m - 1);
27        }
28
29        bool Spfa(int s, int t, int &flow, int &cost) {
30                fill(d, d + n, INF);
31                memset(vis, 0, sizeof vis);
32                d[s] = 0; vis[s] = 1; p[s] = 0; a[s] = INF;
33                queue<int> q;
34                q.push(s);
35                while (!q.empty()) {
36                        int u = q.front(); q.pop();
37                        vis[u] = 0;
38                        int sz = SZ(g[u]);
39                        rep(i, 0, sz) {
40                                Edge& E = edges[g[u][i]];
41                                if (E.cap > E.flow && d[E.to] > d[u] + E.cost)
42 {
43                                        d[E.to] = d[u] + E.cost;
44                                        p[E.to] = g[u][i];
45                                        a[E.to] = min(a[u], E.cap - E.flow);
46                                        if (!vis[E.to]) {
47                                                vis[E.to] = 1;
48                                                q.push(E.to);
49                                        }
50                                }
51                        }
52                }
53                if (d[t] == INF) return 0;
54                flow += a[t];
55                cost += d[t] * a[t];
56                int u = t;
57                while (u != s) {
58                        edges[p[u]].flow += a[t];
59                        edges[p[u] ^ 1].flow -= a[t];
60                        u = edges[p[u]].from;
61                }
62                return 1;
63        }
64
65        int MincostMaxflow(int s, int t) {
66                int flow = 0, cost = 0;
67                while (Spfa(s, t, flow, cost));
68                return cost;
69        }
70 #undef max_V
```

```
};
```

六、数论、组合数学

1.基础数论

①extend gcd

注意：解只是一组，(x+k*b/g,y-k*a/g)也是一组

```
int extend_gcd(int a, int b, int& x, int& y) {
	int d = a;
	if (b != 0) {
		d = extend_gcd(b, a % b, y, x);
		y -= (a / b) * x;
	} else {
		x = 1;
		y = 0;
	}
	return d;
} // (x+k*b/g,y-k*a/g)
```

②欧拉函数

③一元线性同余方程

④中国剩余定理

⑤逆元

2.质因子分解

①线性筛法

②Miller Rabin and Pollard Rho

```
//质因子分解，小数据用筛法直接判，大数据用 pollard_rho
//map[i]是含有多少个质因子 i
//map<LL,int>::iterator c,c->first 表示质因子，c->second 表示次方

#define maxn_for_division 1000001
bool is_prime[maxn_for_division];
vector<int>prime;
map<LL, int>factor;
inline void get_prime() {
	for(int i = 0; i < maxn_for_division; i++)is_prime[i] = 1;
	is_prime[0] = is_prime[1] = 0;
	for(int i = 2; i < maxn_for_division; i++)
		if (is_prime[i]) {
			prime.push_back(i);
			for (int j = i; j < maxn_for_division; j += i)is_prime[j] = 0;
		}
}
inline LL gcd(LL a, LL b) {
	if (!b)return a;
	return gcd(b, a % b);
```

```
}
inline LL mod_mul(LL a, LL b, LL p) {
	LL ans = 0ll;
	a %= p, b %= p;
	if (a > b)swap(a, b);
	while (b) {
		if (b & 1)ans = (ans + a) % p;
		a = (a + a) % p;
		b >>= 1;
	}
	return ans;
}
LL mod_pow(LL a, LL b, LL p) {
	LL ans = 1ll;
	a %= p;
	while (b) {
		if (b & 1)ans = mod_mul(ans, a, p);
		a = mod_mul(a, a, p);
		b >>= 1;
	}
	return ans;
}
bool witness(LL a, LL n) {
	LL m = n - 1;
	int j = 0;
	while(!(m & 1))j++, m >>= 1;
	LL x = mod_pow(a, m, n);
	if (x == 1 || x == n - 1)return 0;
	while(j--) {
		x = mod_mul(x, x, n);
		if(x == n - 1)return 0;
	}
	return 1;
}
#define rep_times 20
bool Miller_Rabin(LL n) { //判断 n 是否为素数
	srand(time(0));
	if(n < 2)return 0;
	if(n == 2)return 1;
	if (!(n & 1))return 0;
	for(int i = 0; i < rep_times; i++) {
		LL a = rand() % (n - 1) + 1;
		if (witness(a, n))return 0;
	}
	return 1;
}
#undef rep_times
LL Pollard_Rho(LL n, int c) {
	LL x = 2, y = 2, d = 1;
```

```
        while (d == 1) {
                x = mod_mul(x, x, n) + c;
                y = mod_mul(y, y, n) + c;
                y = mod_mul(y, y, n) + c;
                d = gcd((x - y >= 0 ? x - y : y - x), n);
        }
        if (d == n)return Pollard_Rho(n, c + 1);
        return d;
}
bool Is_Prime(LL n) {
        return n < maxn_for_division && is_prime[n] || n >=
maxn_for_division && Miller_Rabin(n);
}
void Find_Factor(LL n) {
        if (Is_Prime(n)) {
                factor[n]++;
                return;
        }
        for (int i = 0; i < prime.size() && prime[i] <= n; i++)
                if (n % prime[i] == 0) {
                        while (n % prime[i] == 0) {
                                factor[prime[i]]++;
                                n /= prime[i];
                        }
                }
        if (n != 1) {
                if (Is_Prime(n))factor[n]++;
                else {
                        LL p = Pollard_Rho(n, 1);
                        Find_Factor(p);
                        Find_Factor(n / p);
                }
        }
}
int main() {
        LL n;
        get_prime();
        while(read(n), !IOerror) {
                factor.clear();
                Find_Factor(n);
                //----------------------------output-----------------------
-------
                for(map<LL, int>::iterator c = factor.begin(); c !=
factor.end();) {
                        printf("%lld^%d", c->first, c->second);
                        if((++c) != factor.end())printf("*");
                }
                //-----------------------------------------------------------
-------
```

```
                puts("");
        }
}
```

3.卷积

①FFT

```
class Fft {
public:
#define MAX_LEN 120000
        const double pi = acos(-1.0);
        struct cp {
                double a, b;
                cp operator+ (const cp &o) const { return (cp){a + o.a, b + o.b}; }
                cp operator- (const cp &o) const { return (cp){a - o.a, b - o.b}; }
                cp operator* (const cp &o) const { return (cp){a * o.a - b * o.b, b * o.a + a * o.b}; }
                cp operator* (const double &o) const { return (cp){a * o, b * o}; }
                cp operator!() const {return (cp){a, -b}; }
        } w[MAX_LEN];
        int pos[MAX_LEN];
        cp x[MAX_LEN],y[MAX_LEN],z[MAX_LEN];

        void fft_init(int len) {
                int j = 0;
                while ((1 << j) < len) ++j;
                --j;
                rep(i, 0, len) pos[i] = pos[i >> 1] >> 1 | ((i & 1) << j);
        }

        void fft(cp *x, int len, int sta) {
                rep(i, 0, len) if (i < pos[i]) swap(x[i], x[pos[i]]);
                w[0] = {1, 0};
                for (int i = 2; i <= len; i <<= 1) {
                        cp g = {cos(2 * pi / i), sin(2 * pi / i) *sta};
                        for (int j = i >> 1; j >= 0; j -= 2) w[j] = w[j >> 1];
                        for (int j = 1; j < i >> 1; j += 2) w[j] = w[j - 1] * g;
                        for (int j = 0; j < len; j += i) {
                                cp *a = x + j, *b = a + (i >> 1);
                                for (int k = 0; k < i >> 1; k++) {
                                        cp o = b[k] * w[k];
                                        b[k] = a[k] - o; a[k] = a[k] + o;
                                }
                        }
                }
                if (sta == -1) rep(i, 0, len) x[i].a /= len, x[i].b /= len;
```

```
	}

	void FFT(int *a, int *b, int n, int m, int *c) {
		int len = 1;
		while (len <= (n + m) >> 1) len <<= 1;
		fft_init(len);
		rep(i, (n >> 1), len) x[i].a = x[i].b = 0;
		rep(i, (m >> 1), len) y[i].a = y[i].b = 0;
		rep(i, 0, n) (i & 1 ? x[i >> 1].b : x[i >> 1].a) = a[i];
		rep(i, 0, m) (i & 1 ? y[i >> 1].b : y[i >> 1].a) = b[i];
		fft(x, len, 1); fft(y, len, 1);
		rep(i, 0, (len >> 1)) {
			int j = len - 1 & len - i;
			z[i] = x[i] * y[i] - (x[i] - !x[j]) * (y[i] - !y[j]) *
(w[i] + (cp){1, 0}) * 0.25;
		}
		rep(i, (len >> 1), len) {
			int j = len - 1 & len - i;
			z[i] = x[i] * y[i] - (x[i] - !x[j]) * (y[i] - !y[j]) *
((cp){1, 0} - w[i ^ len >> 1]) * 0.25;
		}
		fft(z, len, -1);
		rep(i, 0, n + m) {
			if (i & 1) c[i] = (int)(z[i >> 1].b + 0.5);
			else c[i] = (int)(z[i >> 1].a + 0.5);
		}
	}
#undef MAX_LEN
} f;
char s1[maxm], s2[maxm];
int a[maxm], b[maxm], c[maxm << 1];
inline void solve() {
	int n;
	while (~scanf("%s%s", s1, s2)) {
		int len1 = strlen(s1);
		int len2 = strlen(s2);
		memset(c, 0, sizeof c);
		rep(i, 0, len1) a[i] = s1[len1 - i - 1] - '0';
		rep(i, 0, len2) b[i] = s2[len2 - i - 1] - '0';
		f.FFT(a, b, len1, len2, c);
		rep(i, 0, len1 + len2 - 1) {
			c[i + 1] += c[i] / 10;
			c[i] %= 10;
		}
		int len = len1 + len2 - 1;
		while (c[len] == 0 && len > 0) --len;
		per(i, 0, len + 1) putchar(c[i] + '0');
		puts("");
	}
```

```
93  }
```

②FWT

③NTT

4.积性函数

①莫比乌斯反演

5.组合数

①线性递推

```
1  C[1][0] = C[1][1] = 1;
2  for (int i = 2; i < maxn; i++) {
3         C[i][0] = 1;
4         for (int j = 1; j < maxn; j++) {
5                C[i][j] = (C[i - 1][j] + C[i - 1][j - 1]) % mod;
6         }
7  }
```

②错排列

```
1  rev[1] = 0;rev[2] = 1;
2  for (int i = 3; i <= maxn; ++i)
3       rev[i] =(i-1)*(rev[i-1]+rev[i-2]);
```

③第二类斯特林数

```
1  for (int i = 0; i < MX ; ++i) {
2         S[i][0] = 0; S[i][i] = 1;
3         for (int j = 1; j < i ; ++j) {
4                S[i][j] = (S[i - 1][j - 1] + ((i - 1) * S[i - 1][j]));;
5         }
6  }
```

④卡特兰数

```
1  catalan[0] = 1;
2  for (int i = 1; i <= 100 ; i++ ) {
3         catalan[i] = 0;
4         for (int j = 0; j < i; j++ ) {
5                catalan[i] = catalan[i] + catalan[j] * catalan[i - j - 1];
6         }
7  }
```

6.Polya

7.组合游戏

8.数值分析

七、计算几何

1.基本函数

2.二维、三维最近点对

3.凸包

4.半平面交

5.旋转卡壳

八、代数

1.矩阵快速幂

```
class Matrix {
#define N 2
public:
	int a[N][N];
	int n;

	void Init(int key) {
		memset(a, 0, sizeof a);
		if (key)
			rep(i, 0, n) a[i][i] = 1;
	}

	Matrix operator+ (const Matrix &b) {
		Matrix c;
		c.n = n;
		rep(i, 0, n) rep(j, 0, n) c.a[i][j] = (a[i][j] + b.a[i][j]) %
mod;
		return c;
	}

	Matrix operator+ (int x) {
		Matrix p = *this;
		rep(i, 0, n) p.a[i][i] = (p.a[i][i] + x % mod) % mod;
		return p;
	}
```

```
26
27          Matrix operator- (const Matrix &b) {
28                  Matrix c;
29                  c.n = n;
30                  rep(i, 0, n) rep(j, 0, n) c.a[i][j] = (a[i][j] - b.a[i][j] +
31  mod) % mod;
32                  return c;
33          }
34
35          Matrix operator* (const Matrix &b) {
36                  Matrix c;
37                  c.n = n;
38                  c.Init(0);
39                  rep(i, 0, n) rep(j, 0, n) rep(k, 0, n)
40                          c.a[i][j] = (c.a[i][j] + a[i][k] * b.a[k][j] % mod) %
41  mod;
42                  return c;
43          }
44
45          Matrix Power(int t) {
46                  Matrix ans, p = *this;
47                  ans.n = p.n;
48                  ans.Init(1);
49                  while (t) {
50                          if (t & 1) ans = ans * p;
51                          p = p * p;
52                          t >>= 1;
53                  }
54                  return ans;
55          }
56
57          void Print() {
58                  rep(i, 0, n) {
59                          rep(j, 0, n) {
60                                  if (j == 0) printf("%d", a[i][j]);
61                                  else printf(" %d", a[i][j]);
62                          }
63                          puts("");
64                  }
65          }
66  #undef N
67  };
```

2.高斯消元

①普通消元法

```
1   bool gauss(int n) {
2           rep(i, 1, n + 1) {
3                   int r = i;
4                   rep(j, i + 1, n + 1)
```

```
                if (fabs(a[j][i]) > fabs(a[r][i])) r = j;
                if (r != i) rep(j, 1, n + 2) swap(a[r][j], a[i][j]);
                rep(k, i + 1, n + 1) {
                        double y = a[k][i] / a[i][i];
                        rep(j, i, n + 2) a[k][j] -= y * a[i][j];
                }
        }
        per(i, 1, n + 1) {
                rep(j, i + 1, n + 1) a[i][n + 1] -= a[j][n + 1] * a[i][j];
                if (fabs(a[i][i]) <= eps  && fabs(a[i][n + 1]) > eps)
                        return 0;
                a[i][n + 1] /= a[i][i];
        }
        return 1;
}
```

Process —— 改一改高斯消元

```
per(j, i, n + 2) {
        rep(k, i + 1, n + 1) {
                a[k][j] -= a[k][i] / a[i][i] * a[i][j];
        }
}
```

②异或消元法

```
ll V[maxn << 2];
int cir, tot;
void XORgauss() {
        for(ll i = b[61]; i; i >>= 1) {
                int j = tot + 1;
                while(j <= cir && !(V[j]&i))j++;
                if(j == cir + 1) continue;
                tot++;
                swap(V[tot], V[j]);
                rep(k, 1, cir + 1)
                        if(k != tot && (V[k]&i))
                                V[k] ^= V[tot];
        }
}
```

九、JAVA 类

1.基础语法

头文件：

```
import java.io.*;
import java.util.*;
import java.math.*;
```

读入：

```
Scanner cin = Scanner (System.in);
while(cin.hasNext())//等价于!=EOF
n = cin.nextInt();//读入一个int型的数
```

```
n = cin.nextBigInteger();//读入一个大整数
```
输出：
```
System.out.print(n);//打印n
System.out.println();//换行
System.out.printf("%d\n",n);//也可以类似 c++里的输出方式
```
定义变量：
```
int i,j,k,a[];
a = new int[100];
BigInteger n,m;
String s;
```
数据类型：

布尔型 **boolean** 1 **true**,**false** **false**

字节型 **byte** 8 -128-127 0

字符型 **char** 16 ‘\u000’-\uffff ‘\u0000’

短整型 **short** 16 -32768-32767 0

整型 **int** 32 -2147483648,2147483647 0

长整型 **long** 64 -9.22E18,9.22E18 0

浮点型 **float** 32 1.4E-45-3.4028E+38 0.0

双精度型 **double** 64 4.9E-324,1.7977E+308 0.0

2.BigInteger

基本函数：

valueOf(parament); 将参数转换为制定的类型
```
BigInteger b = BigInteger.valueOf(a);
String s = ”12345”;
BigInteger c = BigInteger.valueOf(s);
```
add(); 大整数相加
```
BigInteger a = new BigInteger(“23”);
BigInteger b = new BigInteger(“34”);
a. add(b);
```
subtract(); 相减

multiply(); 相乘

divide(); 相除取整

remainder(); 取余

pow();
```
a.pow(b) = a ^ b
```
gcd(); 最大公约数

abs(); 绝对值

negate(); 取反数

mod();
```
a.mod(b) = a % b = a.remainder(b);
```
max(); **min**();

int comareTo();

boolean equals(); 是否相等

构造函数：

BigInteger(String val); 将指定字符串转换为十进制表示形式；

BigInteger(String val, **int** radix); 将指定基数的 BigInteger 的字符串表示形式转换为 BigInteger

基本常量：
```
A = BigInteger.ONE 1
```

```
B = BigInteger.TEN 10
C = BigInteger.ZERO 0
```

基本操作：

```
Scanner cin = new Scanner(System.in); // 读入
while(cin.hasNext())  { //等同于!=EOF
    int n;
    BigInteger m;
    n = cin.nextInt(); //读入一个int;
    m = cin.BigInteger(); //读入一个BigInteger;
    System.out.print(m.toString());
}
if( a.compareTo(b) == 0 ) System.out.println("a == b"); //大整数a==b
else if( a.compareTo(b) > 0 ) System.out.println("a > b"); //大整数a>b
else if( a.compareTo(b) < 0 ) System.out.println("a < b"); //大整数a<b
//大整数绝对值
System.out.println(a.abs()); //大整数a的绝对值
//大整数的幂
int exponent = 10;
System.out.println(a.pow(exponent)); //大整数a的exponent次幂
//返回大整数十进制的字符串表示
System.out.println(a.toString());
//返回大整数p进制的字符串表示
int p = 8;
System.out.println(a.toString(p));
```

3.BigDecimal

基本介绍：

```
BigDecimal BigDecimal(double d); //不允许使用
BigDecimal BigDecimal(String s); //常用,推荐使用
static BigDecimal valueOf(double d); //常用,推荐使用
```

常用函数介绍：

```
BigDecimal remainder(BigDecimal divisor); //求余数
BigDecimal negate(); //求相反数
int compareTo(BigDecimal val);
/* compareTo:
 * 将此 BigDecimal 与指定的 BigDecimal 比较
 * 根据此方法,值相等但具有不同标度的两个 BigDecimal 对象（如，2.0 和 2.00）被认为是相等的;
 * 相对六个 boolean 比较运算符 (<, ==, >, >=, !=, <=) 中每一个运算符的各个方法,优先提供此方法;
 * 建议使用以下语句执行上述比较：(x.compareTo(y) <op> 0)，其中 <op> 是六个比较运算符之一;
 *
 * 指定者：接口 Comparable<BigDecimal> 中的 compareTo
 * 返回：当此 BigDecimal 在数字上小于、等于或大于 val 时，返回 -1、0 或 1
 */
```

样例：

```
1   package com.util;
2   import java.math.BigDecimal;
3   /**
4    * 提供精确的浮点数运算(包括加、减、乘、除、四舍五入)工具类
5    */
```

```java
public class ArithUtil {
    // 除法运算默认精度
    private static final int DEF_DIV_SCALE = 10;
    private ArithUtil() {}

    /**
     * 精确加法
     */
    public static double add(double value1, double value2) {
        BigDecimal b1 = BigDecimal.valueOf(value1);
        BigDecimal b2 = BigDecimal.valueOf(value2);
        return b1.add(b2).doubleValue();
    }

    /**
     * 精确减法
     */
    public static double sub(double value1, double value2) {
        BigDecimal b1 = BigDecimal.valueOf(value1);
        BigDecimal b2 = BigDecimal.valueOf(value2);
        return b1.subtract(b2).doubleValue();
    }

    /**
     * 精确乘法
     */
    public static double mul(double value1, double value2) {
        BigDecimal b1 = BigDecimal.valueOf(value1);
        BigDecimal b2 = BigDecimal.valueOf(value2);
        return b1.multiply(b2).doubleValue();
    }

    /**
     * 精确除法 使用默认精度
     */
    public static double div(double value1, double value2) throws IllegalAccessException {
        return div(value1, value2, DEF_DIV_SCALE);
    }

    /**
     * 精确除法
     * @param scale 精度
     */
    public static double div(double value1, double value2, int scale) throws
IllegalAccessException {
        if(scale < 0) {
            throw new IllegalAccessException("精确度不能小于0");
        }
        BigDecimal b1 = BigDecimal.valueOf(value1);
```

```
55          BigDecimal b2 = BigDecimal.valueOf(value2);
56          // return b1.divide(b2, scale).doubleValue();
57          return b1.divide(b2, scale, BigDecimal.ROUND_HALF_UP).doubleValue();
58      }
59
60      /**
61       * 四舍五入
62       * @param scale 小数点后保留几位
63       */
64      public static double round(double v, int scale) throws IllegalAccessException {
65          return div(v, 1, scale);
66      }
67
68      /**
69       * 比较大小
70       */
71      public static boolean equalTo(BigDecimal b1, BigDecimal b2) {
72          if(b1 == null || b2 == null) {
73              return false;
74          }
75          return 0 == b1.compareTo(b2);
76      }
77  }
```