目录

[板子汇总 1](#_Toc14838)

[注意 1](#_Toc15557)

[定义 2](#_Toc22530)

[计算几何 2](#_Toc18559)

[数据结构 2](#_Toc26846)

[LCT 2](#_Toc18670)

[Splay 3](#_Toc24869)

[Treap 4](#_Toc15231)

[舞蹈链（多重覆盖） 4](#_Toc7903)

[舞蹈链（精确覆盖） 6](#_Toc19233)

[数论 7](#_Toc31273)

[数学 7](#_Toc27865)

[卡特兰数 7](#_Toc990)

[快速幂 7](#_Toc26174)

[博弈 7](#_Toc10943)

[高精度GCD 7](#_Toc13450)

[高精度乘法（FFT） 8](#_Toc688)

[高精度乘法（乘单精度 9](#_Toc16728)

[高精度乘法（朴素） 10](#_Toc1793)

[高精度除法（除单精度） 10](#_Toc31317)

[高精度除法（除高精度） 10](#_Toc14301)

[高精度加法 11](#_Toc19303)

[高精度减法 11](#_Toc13397)

[高精度阶乘 11](#_Toc14102)

[高精度进制转换 11](#_Toc15780)

[高精度幂 12](#_Toc26429)

[高精度平方根 13](#_Toc13717)

[高精度取模（对单精度） 14](#_Toc8274)

[欧拉筛 14](#_Toc31039)

[组合数（逆元线性递推 15](#_Toc5052)

[图论 15](#_Toc17005)

[二分图匹配（匈牙利） 15](#_Toc12818)

[强连通（kosaraju 16](#_Toc19816)

[强连通（tarjan 16](#_Toc25669)

[强连通（tarjan无vector 17](#_Toc11780)

[最大流 17](#_Toc2802)

[最小费用最大流 18](#_Toc22043)

[树分治 19](#_Toc23923)

[拓扑排序 19](#_Toc8414)

[最近公共祖先（倍增） 20](#_Toc20085)

[最近公共祖先（线段树） 20](#_Toc4510)

[线性代数 21](#_Toc2570)

[高斯消元 21](#_Toc10753)

[矩阵行列式 21](#_Toc411)

[线性基 22](#_Toc6847)

[矩阵（加减乘快速幂 22](#_Toc23594)

[稀疏矩阵乘法 22](#_Toc27478)

[杂项 23](#_Toc5558)

[快读 23](#_Toc22621)

[字符串 23](#_Toc11977)

[马拉车 23](#_Toc25939)

[AC自动机 23](#_Toc5497)

[KMP 24](#_Toc17961)

[KMP 2 24](#_Toc31584)

[Tire 25](#_Toc9687)

[后缀数组 25](#_Toc10322)

[对拍 25](#_Toc15022)

[windows环境下bat对拍 25](#_Toc6115)

# 板子汇总

## 注意

优先队列是大的在前面 如果要小的 要重载  
long long 二分答案的时候..精度 也有可能 爆int（?

哈希 自然溢出 yyds 双哈希  
输出限制..

匈牙利的复杂度常数非常小（..

递归爆栈 re

for i 进行计算的时候 （ i 开 long long ）

边界问题 各种01的特判

模 多模一点 都可以模（

char数组开小了也可能报错tle 和 wa （

图是否连通 是否重边 是否自环

读题！！与 或（

重点 重边

当保证n的总和不会很大，但数据组数可能很多的时候，注意初始化造成的tle问题（

pow() 的精度问题

unique erase 先排序

## 定义

**()** (a,b)=1 最大公约数 即a，b互质

**|** 整除 a|b b%a==0

## 计算几何

## 数据结构

### LCT

#include <bits/stdc++.h>  
using namespace std;  
  
struct Link\_Cut\_Tree {  
#define ls ch[p][0]  
#define rs ch[p][1]  
 static const int N = 200005;  
 int ch[N][2], f[N], sum[N], val[N], tag[N], laz[N], siz[N];  
  
 inline void pushup(int p) {  
 // maintain other variables  
 siz[p] = siz[ls] + siz[rs];  
 }  
  
 inline void pushdown(int p) {}  
  
 int get(int x) { return ch[f[x]][1] == x; }  
  
 bool isroot(int x) { return ch[f[x]][0] != x && ch[f[x]][1] != x; }  
  
 inline void rotate(int x) {  
 int y = f[x], z = f[y], k = get(x);  
 if (!isroot(y)) ch[z][ch[z][1] == y] = x;  
 // 上面这句一定要写在前面，普通的Splay是不用的，因为 isRoot (后面会讲)  
 ch[y][k] = ch[x][!k], f[ch[x][!k]] = y;  
 ch[x][!k] = y, f[y] = x, f[x] = z;  
 pushup(x), pushup(y);  
 }  
  
 // 从上到下一层一层 pushDown 即可  
 void update(int p) {  
 if (!isroot(p)) update(f[p]);  
 pushdown(p);  
 }  
  
 inline void splay(int x) {  
 update(x); // 马上就能看到啦。 在  
 // Splay之前要把旋转会经过的路径上的点都PushDown  
 for (int fa; fa = f[x], !isroot(x); rotate(x)) {  
 if (!isroot(fa)) rotate(get(fa) == get(x) ? fa : x);  
 }  
 }  
  
 // 回顾一下代码  
 inline int access(int x) {  
 int p;  
 for (p = 0; x; p = x, x = f[x]) {  
 splay(x), ch[x][1] = p, pushup(x);  
 }  
 return p;  
 }  
  
 inline void makeroot(int p) {  
 p = access(p);  
 swap(ch[p][0], ch[p][1]);  
 tag[p] ^= 1;  
 }  
  
 inline void link(int x, int p) {  
 makeroot(x);  
 splay(x);  
 f[x] = p;  
 }  
  
 inline void cut(int x, int p) {  
 makeroot(x), access(p), splay(p), ls = f[x] = 0;  
 }  
  
 inline int find(int p) {  
 access(p), splay(p);  
 while (ls) pushdown(p), p = ls;  
 splay(p);  
 return p;  
 }  
};

### Splay

#include <bits/stdc++.h>  
using namespace std;  
  
struct Splay {  
 static const int N = 100005;  
 int rt, tot, fa[N], ch[N][2], val[N], cnt[N], sz[N];  
 // rt=根编号，tot=总节点，fa=父节点编号，ch=左/右儿子编号，val=节点的值，cnt=权值出现次数，sz=子树大小  
 void maintain(int x) { //更新x节点字数大小  
 sz[x] = sz[ch[x][0]] + sz[ch[x][1]] + cnt[x];  
 }  
  
 bool get(int x) {  
 return x == ch[fa[x]][1];  
 } //返回节点是父亲的0/1-左/右儿子  
  
 void clear(int x) { //销毁节点x  
 ch[x][0] = ch[x][1] = fa[x] = val[x] = sz[x] = cnt[x] = 0;  
 }  
  
 void rotate(int x) { //旋转  
 int y = fa[x], z = fa[y], chk = get(x);  
 ch[y][chk] = ch[x][chk ^ 1];  
 fa[ch[x][chk ^ 1]] = y;  
 ch[x][chk ^ 1] = y;  
 fa[y] = x;  
 fa[x] = z;  
 if (z) ch[z][y == ch[z][1]] = x;  
 maintain(x);  
 maintain(y);  
 }  
  
 void splay(int x) { //将x节点移动到根  
 for (int f = fa[x]; f = fa[x], f; rotate(x))  
 if (fa[f]) rotate(get(x) == get(f) ? f : x);  
 rt = x;  
 }  
  
 void ins(int k) { //插入  
 if (!rt) {  
 val[++tot] = k;  
 cnt[tot]++;  
 rt = tot;  
 maintain(rt);  
 return;  
 }  
 int cnr = rt, f = 0;  
 while (1) {  
 if (val[cnr] == k) {  
 cnt[cnr]++;  
 maintain(cnr);  
 maintain(f);  
 splay(cnr);  
 break;  
 }  
 f = cnr;  
 cnr = ch[cnr][val[cnr] < k];  
 if (!cnr) {  
 val[++tot] = k;  
 cnt[tot]++;  
 fa[tot] = f;  
 ch[f][val[f] < k] = tot;  
 maintain(tot);  
 maintain(f);  
 splay(tot);  
 break;  
 }  
 }  
 }  
  
 int rk(int k) { // k权值的排名  
 int res = 0, cnr = rt;  
 while (1) {  
 if (k < val[cnr]) {  
 cnr = ch[cnr][0];  
 } else {  
 res += sz[ch[cnr][0]];  
 if (k == val[cnr]) {  
 splay(cnr);  
 return res + 1;  
 }  
 res += cnt[cnr];  
 cnr = ch[cnr][1];  
 }  
 }  
 }  
  
 int kth(int k) { //第k名的权值  
 int cnr = rt;  
 while (1) {  
 if (ch[cnr][0] && k <= sz[ch[cnr][0]]) {  
 cnr = ch[cnr][0];  
 } else {  
 k -= cnt[cnr] + sz[ch[cnr][0]];  
 if (k <= 0) {  
 splay(cnr);  
 return val[cnr];  
 }  
 cnr = ch[cnr][1];  
 }  
 }  
 }  
  
 int pre() { //前驱节点编号  
 int cnr = ch[rt][0];  
 while (ch[cnr][1]) cnr = ch[cnr][1];  
 splay(cnr);  
 return cnr;  
 } // 若需要得到前驱 tree.ins(x), printf("%d\n", tree.val[tree.pre()]),  
 // tree.del(x);  
  
 int nxt() { //后驱节点编号  
 int cnr = ch[rt][1];  
 while (ch[cnr][0]) cnr = ch[cnr][0];  
 splay(cnr);  
 return cnr;  
 } // 若需要得到后驱 tree.ins(x), printf("%d\n", tree.val[tree.pre()]),  
 // tree.del(x);  
  
 void del(int k) { //删除k值  
 rk(k);  
 if (cnt[rt] > 1) {  
 cnt[rt]--;  
 maintain(rt);  
 return;  
 }  
 if (!ch[rt][0] && !ch[rt][1]) {  
 clear(rt);  
 rt = 0;  
 return;  
 }  
 if (!ch[rt][0]) {  
 int cnr = rt;  
 rt = ch[rt][1];  
 fa[rt] = 0;  
 clear(cnr);  
 return;  
 }  
 if (!ch[rt][1]) {  
 int cnr = rt;  
 rt = ch[rt][0];  
 fa[rt] = 0;  
 clear(cnr);  
 return;  
 }  
 int cnr = rt;  
 int x = pre();  
 splay(x);  
 fa[ch[cnr][1]] = x;  
 ch[x][1] = ch[cnr][1];  
 clear(cnr);  
 maintain(rt);  
 }  
} tree;

### Treap

#include <bits/stdc++.h>  
using namespace std;  
struct node {  
 node\* ch[2];  
 int r;  
 int v;  
 int cmp(int const& a) const {  
 if (v == a) return -a;  
 return a > v ? 1 : 0;  
 }  
};  
void rotate(node\*& a, int d) {  
 node\* k = a->ch[d ^ 1];  
 a->ch[d ^ 1] = k->ch[d];  
 k->ch[d] = a;  
 a = k;  
}  
void insert(node\*& a, int x) {  
 if (a == NULL) {  
 a = new node;  
 a->ch[0] = a->ch[1] = NULL;  
 a->v = x;  
 a->r = rand();  
 } else {  
 int d = a->cmp(x);  
 insert(a->ch[d], x);  
 if (a->ch[d]->r > a->r) rotate(a, d ^ 1);  
 }  
}  
void remove(node\*& a, int x) {  
 int d = a->cmp(x);  
 if (d == -1) {  
 if (a->ch[0] == NULL)  
 a = a->ch[1];  
 else if (a->ch[1] == NULL)  
 a = a->ch[0];  
 else {  
 int d2 = a->ch[1]->r > a->ch[0]->r ? 0 : 1;  
 rotate(a, d2);  
 remove(a->ch[d2], x);  
 }  
 } else {  
 remove(a->ch[d], x);  
 }  
}  
int find(node\*& a, int x) {  
 if (a == NULL)  
 return 0;  
 else if (a->v == x)  
 return 1;  
 else {  
 int d = a->cmp(x);  
 return find(a->ch[d], x);  
 }  
}  
int main() {  
 node\* a = NULL;  
 int k, l;  
 while (cin >> k >> l) {  
 if (k == 1)  
 insert(a, l);  
 else if (k == 2)  
 remove(a, l);  
 else {  
 cout << find(a, l) << endl;  
 }  
 }  
}

### 舞蹈链（多重覆盖）

#include <bits/stdc++.h>  
using namespace std;  
struct DLX {  
 static const int maxn = 1000; //列的上限  
 static const int maxr = 1000; //解的上限  
 static const int maxnode = 5000; //总结点数上限  
 static const int INF = 1000000000;  
 int n, sz;  
 int S[maxn];  
  
 int row[maxnode], col[maxnode];  
 int L[maxnode], R[maxnode], U[maxnode], D[maxnode];  
  
 int ansd, ans[maxr];  
  
 int vis[maxnode];  
  
 void init(int n) {  
 this->n = n;  
  
 //虚拟节点  
 for (int i = 0; i <= n; i++) {  
 U[i] = i;  
 D[i] = i;  
 L[i] = i - 1;  
 R[i] = i + 1;  
 }  
 R[n] = 0;  
 L[0] = n;  
  
 sz = n + 1;  
 memset(S, 0, sizeof(S));  
 }  
  
 void addRow(int r, vector<int> columns) {  
 int first = sz;  
 for (int i = 0; i < columns.size(); i++) {  
 int c = columns[i];  
 L[sz] = sz - 1;  
 R[sz] = sz + 1;  
 D[sz] = c;  
 U[sz] = U[c];  
 D[U[c]] = sz;  
 U[c] = sz;  
 row[sz] = r;  
 col[sz] = c;  
 S[c]++;  
 sz++;  
 }  
 R[sz - 1] = first;  
 L[first] = sz - 1;  
 }  
#define FOR(i, A, s) for (int i = A[s]; i != s; i = A[i])  
 void remove(int c) {  
 FOR(i, D, c) { L[R[i]] = L[i], R[L[i]] = R[i]; }  
 }  
  
 void restore(int c) {  
 FOR(i, U, c) { L[R[i]] = i, R[L[i]] = i; }  
 }  
 int f\_check() //精确覆盖区估算剪枝  
 {  
 /\*  
 强剪枝。这个  
 剪枝利用的思想是A\*搜索中的估价函数。即，对于当前的递归深度K下的矩阵，估计其最好情况下（即最少还需要多少步）才能出解。也就是，如果将能够覆盖当  
 前列的所有行全部选中，去掉这些行能够覆盖到的列，将这个操作作为一层深度。重复此操作直到所有列全部出解的深度是多少。如果当前深度加上这个估价函数返  
 回值，其和已然不能更优（也就是已经超过当前最优解），则直接返回，不必再搜。  
 \*/  
  
 int ret = 0;  
 FOR(c, R, 0) vis[c] = true;  
 FOR(c, R, 0)  
 if (vis[c]) {  
 ret++;  
 vis[c] = false;  
 FOR(i, D, c)  
 FOR(j, R, i) vis[col[j]] = false;  
 }  
 return ret;  
 }  
 // d为递归深度  
 void dfs(int d, vector<int>& v) {  
 if (d + f\_check() >= ansd) return;  
 if (R[0] == 0) {  
 if (d < ansd) {  
 ansd = d;  
 v.clear();  
 for (int i = 0; i < ansd; i++) {  
 v.push\_back(ans[i]);  
 }  
 } //找到解  
 return; //记录解的长度  
 }  
  
 //找到S最小的列c  
 int c = R[0];  
 FOR(i, R, 0)  
 if (S[i] < S[c])  
 c = i; //第一个未删除的列  
 //删除第c列  
 FOR(i, D, c) { //用结点i所在的行能覆盖的所有其他列  
 ans[d] = row[i];  
 remove(i);  
 FOR(j, R, i) remove(j); //删除结点i所在的能覆的所有其他列  
 dfs(d + 1, v);  
 FOR(j, L, i) restore(j);  
 restore(i); //恢复结点i所在的行能覆盖的所有其他列  
 } //恢复第c列  
 }  
  
 bool solve(vector<int>& v) {  
 v.clear();  
 ansd = INF;  
 dfs(0, v);  
 return !v.empty();  
 }  
};  
//使用时init初始化，vector中存入r行结点列表用addRow加行，solve(ans)后答案按行的选择在ans中  
DLX dlx;  
int main() {  
 int n, m;  
 cin >> n >> m;  
 dlx.init(m);  
 for (int i = 1; i <= n; i++) {  
 vector<int> v;  
 for (int j = 1; j <= m; j++) {  
 int a;  
 cin >> a;  
 if (a == 1) v.push\_back(j);  
 }  
 dlx.addRow(i, v);  
 }  
 vector<int> ans;  
 dlx.solve(ans);  
 for (int i = 0; i < ans.size(); i++) cout << ans[i];  
}

### 舞蹈链（精确覆盖）

#include <bits/stdc++.h>  
using namespace std;  
struct DLX {  
 static const int maxn = 1000; //列的上限  
 static const int maxr = 1000; //解的上限  
 static const int maxnode = 5000; //总结点数上限  
 int n, sz;  
 int S[maxn];  
  
 int row[maxnode], col[maxnode];  
 int L[maxnode], R[maxnode], U[maxnode], D[maxnode];  
  
 int ansd, ans[maxr];  
  
 void init(int n) {  
 this->n = n;  
  
 //虚拟节点  
 for (int i = 0; i <= n; i++) {  
 U[i] = i;  
 D[i] = i;  
 L[i] = i - 1;  
 R[i] = i + 1;  
 }  
 R[n] = 0;  
 L[0] = n;  
  
 sz = n + 1;  
 memset(S, 0, sizeof(S));  
 }  
  
 void addRow(int r, vector<int> columns) {  
 int first = sz;  
 for (int i = 0; i < columns.size(); i++) {  
 int c = columns[i];  
 L[sz] = sz - 1;  
 R[sz] = sz + 1;  
 D[sz] = c;  
 U[sz] = U[c];  
 D[U[c]] = sz;  
 U[c] = sz;  
 row[sz] = r;  
 col[sz] = c;  
 S[c]++;  
 sz++;  
 }  
 R[sz - 1] = first;  
 L[first] = sz - 1;  
 }  
#define FOR(i, A, s) for (int i = A[s]; i != s; i = A[i])  
 void remove(int c) {  
 L[R[c]] = L[c];  
 R[L[c]] = R[c];  
 FOR(i, D, c)  
 FOR(j, R, i) {  
 U[D[j]] = U[j];  
 D[U[j]] = D[j];  
 --S[col[j]];  
 }  
 }  
  
 void restore(int c) {  
 FOR(i, U, c)  
 FOR(j, L, i) {  
 ++S[col[j]];  
 U[D[j]] = j;  
 D[U[j]] = j;  
 }  
 L[R[c]] = c;  
 R[L[c]] = c;  
 }  
  
 // d为递归深度  
 bool dfs(int d) {  
 if (R[0] == 0) {  
 ansd = d; //找到解  
 return true; //记录解的长度  
 }  
  
 //找到S最小的列c  
 int c = R[0];  
 FOR(i, R, 0) if (S[i] < S[c]) c = i; //第一个未删除的列  
  
 remove(c); //删除第c列  
 FOR(i, D, c) { //用结点i所在的行能覆盖的所有其他列  
 ans[d] = row[i];  
 FOR(j, R, i) remove(col[j]); //删除结点i所在的能覆的所有其他列  
 if (dfs(d + 1)) return true;  
 FOR(j, L, i) restore(col[j]); //恢复结点i所在的行能覆盖的所有其他列  
 }  
 restore(c); //恢复第c列  
  
 return false;  
 }  
  
 bool solve(vector<int>& v) {  
 v.clear();  
 if (!dfs(0)) return false;  
 for (int i = 0; i < ansd; i++) v.push\_back(ans[i]);  
 return true;  
 }  
};  
//使用时init初始化，vector中存入r行结点列表用addRow加行，solve(ans)后答案按行的选择在ans中

## 数论

## 数学

### 卡特兰数

卡特兰数1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012,...

超级卡特兰数1, 1, 3, 11, 45, 197, 903, 4279, 20793, 103049,...（从第0项开始）

大施罗德数(OEIS A006318)1, 2, 6, 22, 90, 394, 1806, 8558, 41586, 206098,...

超级卡特兰数的两倍（除第一项）

### 快速幂

ll qpow(ll a, ll b) {  
 ll ans = 1;  
 while (b) {  
 if (b & 1) ans = (ans \* a) % mod;  
 a = (a \* a) % mod;  
 b >>= 1;  
 }  
 return ans;  
}

### 博弈

**SG定理：**

mex(minimal excludant)运算，表示最小的不属于这个集合的非负整数。例如mex{0,1,2,4}=3、mex{2,3,5}=0、mex{}=0。  
Sprague-Grundy定理（SG定理）：游戏和的SG函数等于各个游戏SG函数的Nim和。这样就可以将每一个子游戏分而治之，从而简化了问题。而Bouton定理就是Sprague-Grundy定理在Nim游戏中的直接应用，因为单堆的Nim游戏 SG函数满足 SG(x) = x。

**Nimk：**

普通的NIM游戏是在n堆石子中每次选一堆，取任意个石子，而NIMK游戏是在n堆石子中每次选择k堆，1<=k<=n，从这k堆中每堆里都取出任意数目的石子，取的石子数可以不同，其他规则相同。  
对于普通的NIM游戏，我们采取的是对每堆的SG值进行异或，异或其实就是对每一个SG值二进制位上的数求和然后模2，比如说3^5就是011+101=112，然后对每一位都模2就变成了110，所以3^5=6。而NIMK游戏和NIM游戏的区别就在于模的不是2，如果是取k堆，就模k+1，所以取1堆的普通NIM游戏是模2。当k=2时,3^5→011+101=112，对每一位都模3之后三位二进制位上对应的数仍然是1，1，2。那么当且仅当每一位二进制位上的数都是0的时候，先手必败，否则先手必胜。

### 高精度GCD

#include <bits/stdc++.h>  
using namespace std;  
string add(string a, string b) {  
 const int L = 1e5;  
 string ans;  
 int na[L] = {0}, nb[L] = {0};  
 int la = a.size(), lb = b.size();  
 for (int i = 0; i < la; i++) na[la - 1 - i] = a[i] - '0';  
 for (int i = 0; i < lb; i++) nb[lb - 1 - i] = b[i] - '0';  
 int lmax = la > lb ? la : lb;  
 for (int i = 0; i < lmax; i++)  
 na[i] += nb[i], na[i + 1] += na[i] / 10, na[i] %= 10;  
 if (na[lmax]) lmax++;  
 for (int i = lmax - 1; i >= 0; i--) ans += na[i] + '0';  
 return ans;  
}  
string mul(string a, string b) {  
 const int L = 1e5;  
 string s;  
 int na[L], nb[L], nc[L],  
 La = a.size(), Lb = b.size(); // na存储被乘数，nb存储乘数，nc存储积  
 fill(na, na + L, 0);  
 fill(nb, nb + L, 0);  
 fill(nc, nc + L, 0); //将na,nb,nc都置为0  
 for (int i = La - 1; i >= 0; i--)  
 na[La - i] =  
 a[i] - '0'; //将字符串表示的大整形数转成i整形数组表示的大整形数  
 for (int i = Lb - 1; i >= 0; i--) nb[Lb - i] = b[i] - '0';  
 for (int i = 1; i <= La; i++)  
 for (int j = 1; j <= Lb; j++)  
 nc[i + j - 1] +=  
 na[i] \*  
 nb[j]; // a的第i位乘以b的第j位为积的第i+j-1位（先不考虑进位）  
 for (int i = 1; i <= La + Lb; i++)  
 nc[i + 1] += nc[i] / 10, nc[i] %= 10; //统一处理进位  
 if (nc[La + Lb]) s += nc[La + Lb] + '0'; //判断第i+j位上的数字是不是0  
 for (int i = La + Lb - 1; i >= 1; i--)  
 s += nc[i] + '0'; //将整形数组转成字符串  
 return s;  
}  
int sub(int \*a, int \*b, int La, int Lb) {  
 if (La < Lb) return -1; //如果a小于b，则返回-1  
 if (La == Lb) {  
 for (int i = La - 1; i >= 0; i--)  
 if (a[i] > b[i])  
 break;  
 else if (a[i] < b[i])  
 return -1; //如果a小于b，则返回-1  
 }  
 for (int i = 0; i < La; i++) //高精度减法  
 {  
 a[i] -= b[i];  
 if (a[i] < 0) a[i] += 10, a[i + 1]--;  
 }  
 for (int i = La - 1; i >= 0; i--)  
 if (a[i]) return i + 1; //返回差的位数  
 return 0; //返回差的位数  
}  
string div(string n1, string n2,  
 int nn) // n1,n2是字符串表示的被除数，除数,nn是选择返回商还是余数  
{  
 const int L = 1e5;  
 string s, v; // s存商,v存余数  
 int a[L], b[L], r[L],  
 La = n1.size(), Lb = n2.size(), i,  
 tp = La; // a，b是整形数组表示被除数，除数，tp保存被除数的长度  
 fill(a, a + L, 0);  
 fill(b, b + L, 0);  
 fill(r, r + L, 0); //数组元素都置为0  
 for (i = La - 1; i >= 0; i--) a[La - 1 - i] = n1[i] - '0';  
 for (i = Lb - 1; i >= 0; i--) b[Lb - 1 - i] = n2[i] - '0';  
 if (La < Lb || (La == Lb && n1 < n2)) {  
 // cout<<0<<endl;  
 return n1;  
 } //如果a<b,则商为0，余数为被除数  
 int t = La - Lb; //除被数和除数的位数之差  
 for (int i = La - 1; i >= 0; i--) //将除数扩大10^t倍  
 if (i >= t)  
 b[i] = b[i - t];  
 else  
 b[i] = 0;  
 Lb = La;  
 for (int j = 0; j <= t; j++) {  
 int temp;  
 while ((temp = sub(a, b + j, La, Lb - j)) >=  
 0) //如果被除数比除数大继续减  
 {  
 La = temp;  
 r[t - j]++;  
 }  
 }  
 for (i = 0; i < L - 10; i++)  
 r[i + 1] += r[i] / 10, r[i] %= 10; //统一处理进位  
 while (!r[i]) i--; //将整形数组表示的商转化成字符串表示的  
 while (i >= 0) s += r[i--] + '0';  
 // cout<<s<<endl;  
 i = tp;  
 while (!a[i]) i--; //将整形数组表示的余数转化成字符串表示的</span>  
 while (i >= 0) v += a[i--] + '0';  
 if (v.empty()) v = "0";  
 // cout<<v<<endl;  
 if (nn == 1) return s;  
 if (nn == 2) return v;  
}  
bool judge(string s) //判断s是否为全0串  
{  
 for (int i = 0; i < s.size(); i++)  
 if (s[i] != '0') return false;  
 return true;  
}  
string gcd(string a, string b) //求最大公约数  
{  
 string t;  
 while (!judge(b)) //如果余数不为0，继续除  
 {  
 t = a; //保存被除数的值  
 a = b; //用除数替换被除数  
 b = div(t, b, 2); //用余数替换除数  
 }  
 return a;  
}  
  
//o(无法估计)

### 高精度乘法（FFT）

#include <bits/stdc++.h>  
using namespace std;  
#define L(x) (1 << (x))  
const double PI = acos(-1.0);  
const int Maxn = 133015;  
double ax[Maxn], ay[Maxn], bx[Maxn], by[Maxn];  
char sa[Maxn / 2], sb[Maxn / 2];  
int sum[Maxn];  
int x1[Maxn], x2[Maxn];  
int revv(int x, int bits) {  
 int ret = 0;  
 for (int i = 0; i < bits; i++) {  
 ret <<= 1;  
 ret |= x & 1;  
 x >>= 1;  
 }  
 return ret;  
}  
void fft(double\* a, double\* b, int n, bool rev) {  
 int bits = 0;  
 while (1 << bits < n) ++bits;  
 for (int i = 0; i < n; i++) {  
 int j = revv(i, bits);  
 if (i < j) swap(a[i], a[j]), swap(b[i], b[j]);  
 }  
 for (int len = 2; len <= n; len <<= 1) {  
 int half = len >> 1;  
 double wmx = cos(2 \* PI / len), wmy = sin(2 \* PI / len);  
 if (rev) wmy = -wmy;  
 for (int i = 0; i < n; i += len) {  
 double wx = 1, wy = 0;  
 for (int j = 0; j < half; j++) {  
 double cx = a[i + j], cy = b[i + j];  
 double dx = a[i + j + half], dy = b[i + j + half];  
 double ex = dx \* wx - dy \* wy, ey = dx \* wy + dy \* wx;  
 a[i + j] = cx + ex, b[i + j] = cy + ey;  
 a[i + j + half] = cx - ex, b[i + j + half] = cy - ey;  
 double wnx = wx \* wmx - wy \* wmy, wny = wx \* wmy + wy \* wmx;  
 wx = wnx, wy = wny;  
 }  
 }  
 }  
 if (rev) {  
 for (int i = 0; i < n; i++) a[i] /= n, b[i] /= n;  
 }  
}  
int solve(int a[], int na, int b[], int nb, int ans[]) {  
 int len = max(na, nb), ln;  
 for (ln = 0; L(ln) < len; ++ln)  
 ;  
 len = L(++ln);  
 for (int i = 0; i < len; ++i) {  
 if (i >= na)  
 ax[i] = 0, ay[i] = 0;  
 else  
 ax[i] = a[i], ay[i] = 0;  
 }  
 fft(ax, ay, len, 0);  
 for (int i = 0; i < len; ++i) {  
 if (i >= nb)  
 bx[i] = 0, by[i] = 0;  
 else  
 bx[i] = b[i], by[i] = 0;  
 }  
 fft(bx, by, len, 0);  
 for (int i = 0; i < len; ++i) {  
 double cx = ax[i] \* bx[i] - ay[i] \* by[i];  
 double cy = ax[i] \* by[i] + ay[i] \* bx[i];  
 ax[i] = cx, ay[i] = cy;  
 }  
 fft(ax, ay, len, 1);  
 for (int i = 0; i < len; ++i) ans[i] = (int)(ax[i] + 0.5);  
 return len;  
}  
string mul(string sa, string sb) {  
 int l1, l2, l;  
 int i;  
 string ans;  
 memset(sum, 0, sizeof(sum));  
 l1 = sa.size();  
 l2 = sb.size();  
 for (i = 0; i < l1; i++) x1[i] = sa[l1 - i - 1] - '0';  
 for (i = 0; i < l2; i++) x2[i] = sb[l2 - i - 1] - '0';  
 l = solve(x1, l1, x2, l2, sum);  
 for (i = 0; i < l || sum[i] >= 10; i++) // 进位  
 {  
 sum[i + 1] += sum[i] / 10;  
 sum[i] %= 10;  
 }  
 l = i;  
 while (sum[l] <= 0 && l > 0) l--; // 检索最高位  
 for (i = l; i >= 0; i--) ans += sum[i] + '0'; // 倒序输出  
 return ans;  
}  
int main() {  
 cin.sync\_with\_stdio(false);  
 string a, b;  
 while (cin >> a >> b) cout << mul(a, b) << endl;  
 return 0;  
}  
  
//o(nlogn)

### 高精度乘法（乘单精度

#include <bits/stdc++.h>  
using namespace std;  
string mul(string a, int b) //高精度a乘单精度b  
{  
 const int L = 100005;  
 int na[L];  
 string ans;  
 int La = a.size();  
 fill(na, na + L, 0);  
 for (int i = La - 1; i >= 0; i--) na[La - i - 1] = a[i] - '0';  
 int w = 0;  
 for (int i = 0; i < La; i++)  
 na[i] = na[i] \* b + w, w = na[i] / 10, na[i] = na[i] % 10;  
 while (w) na[La++] = w % 10, w /= 10;  
 La--;  
 while (La >= 0) ans += na[La--] + '0';  
 return ans;  
}  
  
//o(n)

### 高精度乘法（朴素）

#include <bits/stdc++.h>  
using namespace std;  
string mul(string a, string b) //高精度乘法a,b,均为非负整数  
{  
 const int L = 1e5;  
 string s;  
 int na[L], nb[L], nc[L],  
 La = a.size(), Lb = b.size(); // na存储被乘数，nb存储乘数，nc存储积  
 fill(na, na + L, 0);  
 fill(nb, nb + L, 0);  
 fill(nc, nc + L, 0); //将na,nb,nc都置为0  
 for (int i = La - 1; i >= 0; i--)  
 na[La - i] =  
 a[i] - '0'; //将字符串表示的大整形数转成i整形数组表示的大整形数  
 for (int i = Lb - 1; i >= 0; i--) nb[Lb - i] = b[i] - '0';  
 for (int i = 1; i <= La; i++)  
 for (int j = 1; j <= Lb; j++)  
 nc[i + j - 1] +=  
 na[i] \*  
 nb[j]; // a的第i位乘以b的第j位为积的第i+j-1位（先不考虑进位）  
 for (int i = 1; i <= La + Lb; i++)  
 nc[i + 1] += nc[i] / 10, nc[i] %= 10; //统一处理进位  
 if (nc[La + Lb]) s += nc[La + Lb] + '0'; //判断第i+j位上的数字是不是0  
 for (int i = La + Lb - 1; i >= 1; i--)  
 s += nc[i] + '0'; //将整形数组转成字符串  
 return s;  
}  
  
//o(n^2)

### 高精度除法（除单精度）

#include <bits/stdc++.h>  
using namespace std;  
string div(string a, int b) //高精度a除以单精度b  
{  
 string r, ans;  
 int d = 0;  
 if (a == "0") return a; //特判  
 for (int i = 0; i < a.size(); i++) {  
 r += (d \* 10 + a[i] - '0') / b + '0'; //求出商  
 d = (d \* 10 + (a[i] - '0')) % b; //求出余数  
 }  
 int p = 0;  
 for (int i = 0; i < r.size(); i++)  
 if (r[i] != '0') {  
 p = i;  
 break;  
 }  
 return r.substr(p);  
}  
  
//o(n)

### 高精度除法（除高精度）

#include <bits/stdc++.h>  
using namespace std;  
int sub(int \*a, int \*b, int La, int Lb) {  
 if (La < Lb) return -1; //如果a小于b，则返回-1  
 if (La == Lb) {  
 for (int i = La - 1; i >= 0; i--)  
 if (a[i] > b[i])  
 break;  
 else if (a[i] < b[i])  
 return -1; //如果a小于b，则返回-1  
 }  
 for (int i = 0; i < La; i++) //高精度减法  
 {  
 a[i] -= b[i];  
 if (a[i] < 0) a[i] += 10, a[i + 1]--;  
 }  
 for (int i = La - 1; i >= 0; i--)  
 if (a[i]) return i + 1; //返回差的位数  
 return 0; //返回差的位数  
}  
string div(string n1, string n2, int nn)  
// n1,n2是字符串表示的被除数，除数,nn是选择返回商还是余数  
{  
 const int L = 1e5;  
 string s, v; // s存商,v存余数  
 int a[L], b[L], r[L], La = n1.size(), Lb = n2.size(), i, tp = La;  
 // a，b是整形数组表示被除数，除数，tp保存被除数的长度  
 fill(a, a + L, 0);  
 fill(b, b + L, 0);  
 fill(r, r + L, 0); //数组元素都置为0  
 for (i = La - 1; i >= 0; i--) a[La - 1 - i] = n1[i] - '0';  
 for (i = Lb - 1; i >= 0; i--) b[Lb - 1 - i] = n2[i] - '0';  
 if (La < Lb || (La == Lb && n1 < n2)) {  
 // cout<<0<<endl;  
 return n1;  
 } //如果a<b,则商为0，余数为被除数  
 int t = La - Lb; //除被数和除数的位数之差  
 for (int i = La - 1; i >= 0; i--) //将除数扩大10^t倍  
 if (i >= t)  
 b[i] = b[i - t];  
 else  
 b[i] = 0;  
 Lb = La;  
 for (int j = 0; j <= t; j++) {  
 int temp;  
 while ((temp = sub(a, b + j, La, Lb - j)) >=  
 0) //如果被除数比除数大继续减  
 {  
 La = temp;  
 r[t - j]++;  
 }  
 }  
 for (i = 0; i < L - 10; i++)  
 r[i + 1] += r[i] / 10, r[i] %= 10; //统一处理进位  
 while (!r[i]) i--; //将整形数组表示的商转化成字符串表示的  
 while (i >= 0) s += r[i--] + '0';  
 // cout<<s<<endl;  
 i = tp;  
 while (!a[i]) i--; //将整形数组表示的余数转化成字符串表示的</span>  
 while (i >= 0) v += a[i--] + '0';  
 if (v.empty()) v = "0";  
 // cout<<v<<endl;  
 if (nn == 1) return s; //返回商  
 if (nn == 2) return v; //返回余数  
}  
  
//o(n^2)

### 高精度加法

#include <bits/stdc++.h>  
using namespace std;  
string add(string a, string b) //只限两个非负整数相加  
{  
 const int L = 1e5;  
 string ans;  
 int na[L] = {0}, nb[L] = {0};  
 int la = a.size(), lb = b.size();  
 for (int i = 0; i < la; i++) na[la - 1 - i] = a[i] - '0';  
 for (int i = 0; i < lb; i++) nb[lb - 1 - i] = b[i] - '0';  
 int lmax = la > lb ? la : lb;  
 for (int i = 0; i < lmax; i++)  
 na[i] += nb[i], na[i + 1] += na[i] / 10, na[i] %= 10;  
 if (na[lmax]) lmax++;  
 for (int i = lmax - 1; i >= 0; i--) ans += na[i] + '0';  
 return ans;  
}  
  
//o(n)

### 高精度减法

#include <bits/stdc++.h>  
using namespace std;  
string sub(string a, string b) //只限大的非负整数减小的非负整数  
{  
 const int L = 1e5;  
 string ans;  
 int na[L] = {0}, nb[L] = {0};  
 int la = a.size(), lb = b.size();  
 for (int i = 0; i < la; i++) na[la - 1 - i] = a[i] - '0';  
 for (int i = 0; i < lb; i++) nb[lb - 1 - i] = b[i] - '0';  
 int lmax = la > lb ? la : lb;  
 for (int i = 0; i < lmax; i++) {  
 na[i] -= nb[i];  
 if (na[i] < 0) na[i] += 10, na[i + 1]--;  
 }  
 while (!na[--lmax] && lmax > 0)  
 ;  
 lmax++;  
 for (int i = lmax - 1; i >= 0; i--) ans += na[i] + '0';  
 return ans;  
}  
  
//o(n)

### 高精度阶乘

#include <bits/stdc++.h>  
using namespace std;  
string fac(int n) {  
 const int L = 100005;  
 int a[L];  
 string ans;  
 if (n == 0) return "1";  
 fill(a, a + L, 0);  
 int s = 0, m = n;  
 while (m) a[++s] = m % 10, m /= 10;  
 for (int i = n - 1; i >= 2; i--) {  
 int w = 0;  
 for (int j = 1; j <= s; j++)  
 a[j] = a[j] \* i + w, w = a[j] / 10, a[j] = a[j] % 10;  
 while (w) a[++s] = w % 10, w /= 10;  
 }  
 while (!a[s]) s--;  
 while (s >= 1) ans += a[s--] + '0';  
 return ans;  
}  
  
//o(n^2)

### 高精度进制转换

#include <bits/stdc++.h>  
using namespace std;  
//将字符串表示的10进制大整数转换为m进制的大整数  
//并返回m进制大整数的字符串  
bool judge(string s) //判断串是否为全零串  
{  
 for (int i = 0; i < s.size(); i++)  
 if (s[i] != '0') return 1;  
 return 0;  
}  
string solve(  
 string s, int n,  
 int m) // n进制转m进制只限0-9进制，若涉及带字母的进制，稍作修改即可  
{  
 string r, ans;  
 int d = 0;  
 if (!judge(s)) return "0"; //特判  
 while (judge(s)) //被除数不为0则继续  
 {  
 for (int i = 0; i < s.size(); i++) {  
 r += (d \* n + s[i] - '0') / m + '0'; //求出商  
 d = (d \* n + (s[i] - '0')) % m; //求出余数  
 }  
 s = r; //把商赋给下一次的被除数  
 r = ""; //把商清空  
 ans += d + '0'; //加上进制转换后数字  
 d = 0; //清空余数  
 }  
 reverse(ans.begin(), ans.end()); //倒置下  
 return ans;  
}  
  
//o(n^2)

### 高精度幂

#include <bits/stdc++.h>  
#define L(x) (1 << (x))  
using namespace std;  
const double PI = acos(-1.0);  
const int Maxn = 133015;  
double ax[Maxn], ay[Maxn], bx[Maxn], by[Maxn];  
char sa[Maxn / 2], sb[Maxn / 2];  
int sum[Maxn];  
int x1[Maxn], x2[Maxn];  
int revv(int x, int bits) {  
 int ret = 0;  
 for (int i = 0; i < bits; i++) {  
 ret <<= 1;  
 ret |= x & 1;  
 x >>= 1;  
 }  
 return ret;  
}  
void fft(double\* a, double\* b, int n, bool rev) {  
 int bits = 0;  
 while (1 << bits < n) ++bits;  
 for (int i = 0; i < n; i++) {  
 int j = revv(i, bits);  
 if (i < j) swap(a[i], a[j]), swap(b[i], b[j]);  
 }  
 for (int len = 2; len <= n; len <<= 1) {  
 int half = len >> 1;  
 double wmx = cos(2 \* PI / len), wmy = sin(2 \* PI / len);  
 if (rev) wmy = -wmy;  
 for (int i = 0; i < n; i += len) {  
 double wx = 1, wy = 0;  
 for (int j = 0; j < half; j++) {  
 double cx = a[i + j], cy = b[i + j];  
 double dx = a[i + j + half], dy = b[i + j + half];  
 double ex = dx \* wx - dy \* wy, ey = dx \* wy + dy \* wx;  
 a[i + j] = cx + ex, b[i + j] = cy + ey;  
 a[i + j + half] = cx - ex, b[i + j + half] = cy - ey;  
 double wnx = wx \* wmx - wy \* wmy, wny = wx \* wmy + wy \* wmx;  
 wx = wnx, wy = wny;  
 }  
 }  
 }  
 if (rev) {  
 for (int i = 0; i < n; i++) a[i] /= n, b[i] /= n;  
 }  
}  
int solve(int a[], int na, int b[], int nb, int ans[]) {  
 int len = max(na, nb), ln;  
 for (ln = 0; L(ln) < len; ++ln)  
 ;  
 len = L(++ln);  
 for (int i = 0; i < len; ++i) {  
 if (i >= na)  
 ax[i] = 0, ay[i] = 0;  
 else  
 ax[i] = a[i], ay[i] = 0;  
 }  
 fft(ax, ay, len, 0);  
 for (int i = 0; i < len; ++i) {  
 if (i >= nb)  
 bx[i] = 0, by[i] = 0;  
 else  
 bx[i] = b[i], by[i] = 0;  
 }  
 fft(bx, by, len, 0);  
 for (int i = 0; i < len; ++i) {  
 double cx = ax[i] \* bx[i] - ay[i] \* by[i];  
 double cy = ax[i] \* by[i] + ay[i] \* bx[i];  
 ax[i] = cx, ay[i] = cy;  
 }  
 fft(ax, ay, len, 1);  
 for (int i = 0; i < len; ++i) ans[i] = (int)(ax[i] + 0.5);  
 return len;  
}  
string mul(string sa, string sb) {  
 int l1, l2, l;  
 int i;  
 string ans;  
 memset(sum, 0, sizeof(sum));  
 l1 = sa.size();  
 l2 = sb.size();  
 for (i = 0; i < l1; i++) x1[i] = sa[l1 - i - 1] - '0';  
 for (i = 0; i < l2; i++) x2[i] = sb[l2 - i - 1] - '0';  
 l = solve(x1, l1, x2, l2, sum);  
 for (i = 0; i < l || sum[i] >= 10; i++) // 进位  
 {  
 sum[i + 1] += sum[i] / 10;  
 sum[i] %= 10;  
 }  
 l = i;  
 while (sum[l] <= 0 && l > 0) l--; // 检索最高位  
 for (i = l; i >= 0; i--) ans += sum[i] + '0'; // 倒序输出  
 return ans;  
}  
string Pow(string a, int n) {  
 if (n == 1) return a;  
 if (n & 1) return mul(Pow(a, n - 1), a);  
 string ans = Pow(a, n / 2);  
 return mul(ans, ans);  
}  
  
//o(nlognlogm)

### 高精度平方根

#include <bits/stdc++.h>  
using namespace std;  
const int L = 2015;  
string add(string a, string b) //只限两个非负整数相加  
{  
 string ans;  
 int na[L] = {0}, nb[L] = {0};  
 int la = a.size(), lb = b.size();  
 for (int i = 0; i < la; i++) na[la - 1 - i] = a[i] - '0';  
 for (int i = 0; i < lb; i++) nb[lb - 1 - i] = b[i] - '0';  
 int lmax = la > lb ? la : lb;  
 for (int i = 0; i < lmax; i++)  
 na[i] += nb[i], na[i + 1] += na[i] / 10, na[i] %= 10;  
 if (na[lmax]) lmax++;  
 for (int i = lmax - 1; i >= 0; i--) ans += na[i] + '0';  
 return ans;  
}  
string sub(string a, string b) //只限大的非负整数减小的非负整数  
{  
 string ans;  
 int na[L] = {0}, nb[L] = {0};  
 int la = a.size(), lb = b.size();  
 for (int i = 0; i < la; i++) na[la - 1 - i] = a[i] - '0';  
 for (int i = 0; i < lb; i++) nb[lb - 1 - i] = b[i] - '0';  
 int lmax = la > lb ? la : lb;  
 for (int i = 0; i < lmax; i++) {  
 na[i] -= nb[i];  
 if (na[i] < 0) na[i] += 10, na[i + 1]--;  
 }  
 while (!na[--lmax] && lmax > 0)  
 ;  
 lmax++;  
 for (int i = lmax - 1; i >= 0; i--) ans += na[i] + '0';  
 return ans;  
}  
string mul(string a, string b) //高精度乘法a,b,均为非负整数  
{  
 string s;  
 int na[L], nb[L], nc[L],  
 La = a.size(), Lb = b.size(); // na存储被乘数，nb存储乘数，nc存储积  
 fill(na, na + L, 0);  
 fill(nb, nb + L, 0);  
 fill(nc, nc + L, 0); //将na,nb,nc都置为0  
 for (int i = La - 1; i >= 0; i--)  
 na[La - i] =  
 a[i] - '0'; //将字符串表示的大整形数转成i整形数组表示的大整形数  
 for (int i = Lb - 1; i >= 0; i--) nb[Lb - i] = b[i] - '0';  
 for (int i = 1; i <= La; i++)  
 for (int j = 1; j <= Lb; j++)  
 nc[i + j - 1] +=  
 na[i] \*  
 nb[j]; // a的第i位乘以b的第j位为积的第i+j-1位（先不考虑进位）  
 for (int i = 1; i <= La + Lb; i++)  
 nc[i + 1] += nc[i] / 10, nc[i] %= 10; //统一处理进位  
 if (nc[La + Lb]) s += nc[La + Lb] + '0'; //判断第i+j位上的数字是不是0  
 for (int i = La + Lb - 1; i >= 1; i--)  
 s += nc[i] + '0'; //将整形数组转成字符串  
 return s;  
}  
int sub(int \*a, int \*b, int La, int Lb) {  
 if (La < Lb) return -1; //如果a小于b，则返回-1  
 if (La == Lb) {  
 for (int i = La - 1; i >= 0; i--)  
 if (a[i] > b[i])  
 break;  
 else if (a[i] < b[i])  
 return -1; //如果a小于b，则返回-1  
 }  
 for (int i = 0; i < La; i++) //高精度减法  
 {  
 a[i] -= b[i];  
 if (a[i] < 0) a[i] += 10, a[i + 1]--;  
 }  
 for (int i = La - 1; i >= 0; i--)  
 if (a[i]) return i + 1; //返回差的位数  
 return 0; //返回差的位数  
}  
string div(string n1, string n2,  
 int nn) // n1,n2是字符串表示的被除数，除数,nn是选择返回商还是余数  
{  
 string s, v; // s存商,v存余数  
 int a[L], b[L], r[L],  
 La = n1.size(), Lb = n2.size(), i,  
 tp = La; // a，b是整形数组表示被除数，除数，tp保存被除数的长度  
 fill(a, a + L, 0);  
 fill(b, b + L, 0);  
 fill(r, r + L, 0); //数组元素都置为0  
 for (i = La - 1; i >= 0; i--) a[La - 1 - i] = n1[i] - '0';  
 for (i = Lb - 1; i >= 0; i--) b[Lb - 1 - i] = n2[i] - '0';  
 if (La < Lb || (La == Lb && n1 < n2)) {  
 // cout<<0<<endl;  
 return n1;  
 } //如果a<b,则商为0，余数为被除数  
 int t = La - Lb; //除被数和除数的位数之差  
 for (int i = La - 1; i >= 0; i--) //将除数扩大10^t倍  
 if (i >= t)  
 b[i] = b[i - t];  
 else  
 b[i] = 0;  
 Lb = La;  
 for (int j = 0; j <= t; j++) {  
 int temp;  
 while ((temp = sub(a, b + j, La, Lb - j)) >=  
 0) //如果被除数比除数大继续减  
 {  
 La = temp;  
 r[t - j]++;  
 }  
 }  
 for (i = 0; i < L - 10; i++)  
 r[i + 1] += r[i] / 10, r[i] %= 10; //统一处理进位  
 while (!r[i]) i--; //将整形数组表示的商转化成字符串表示的  
 while (i >= 0) s += r[i--] + '0';  
 // cout<<s<<endl;  
 i = tp;  
 while (!a[i]) i--; //将整形数组表示的余数转化成字符串表示的</span>  
 while (i >= 0) v += a[i--] + '0';  
 if (v.empty()) v = "0";  
 // cout<<v<<endl;  
 if (nn == 1) return s;  
 if (nn == 2) return v;  
}  
bool cmp(string a, string b) {  
 if (a.size() < b.size()) return 1; // a小于等于b返回真  
 if (a.size() == b.size() && a <= b) return 1;  
 return 0;  
}  
string DeletePreZero(string s) {  
 int i;  
 for (i = 0; i < s.size(); i++)  
 if (s[i] != '0') break;  
 return s.substr(i);  
}  
  
string BigInterSqrt(string n) {  
 n = DeletePreZero(n);  
 string l = "1", r = n, mid, ans;  
 while (cmp(l, r)) {  
 mid = div(add(l, r), "2", 1);  
 if (cmp(mul(mid, mid), n))  
 ans = mid, l = add(mid, "1");  
 else  
 r = sub(mid, "1");  
 }  
 return ans;  
}  
  
// o(n^3)

### 高精度取模（对单精度）

#include <bits/stdc++.h>  
using namespace std;  
int mod(string a,int b)//高精度a除以单精度b  
{  
 int d=0;  
 for(int i=0;i<a.size();i++) d=(d\*10+(a[i]-'0'))%b;//求出余数  
 return d;  
}  
  
//o(n)

### 欧拉筛

#include <bits/stdc++.h>  
using namespace std;  
typedef long long ll;  
const int N = 1000005;  
int phi[N], prime[N], cnt;  
bool st[N];  
  
void get\_eulers() {  
 phi[1] = 1;  
 for (int i = 2; i < N; i++) {  
 if (!st[i]) {  
 prime[cnt++] = i;  
 phi[i] = i - 1;  
 }  
 for (int j = 0; prime[j] \* i < N; j++) {  
 st[prime[j] \* i] = 1;  
 if (i % prime[j] == 0) {  
 phi[prime[j] \* i] = phi[i] \* prime[j];  
 break;  
 }  
 phi[prime[j] \* i] = phi[i] \* (prime[j] - 1);  
 }  
 }  
}  
  
int main() {  
 get\_eulers();  
 ll n;  
 cin >> n;  
 ll ans = 0;  
 for (int i = 1; i <= n; i++) ans += phi[i];  
 cout << ans;  
}

### 组合数（逆元线性递推

#include <bits/stdc++.h>  
  
using namespace std;  
typedef long long ll;  
const ll mod = 1e9 + 7;  
const ll maxn = 3e4 + 5;  
ll inv[maxn], fac[maxn];  
  
ll qpow(ll a, ll b) {  
 ll ans = 1;  
 while (b) {  
 if (b & 1) ans = (ans \* a) % mod;  
 a = (a \* a) % mod;  
 b >>= 1;  
 }  
 return ans;  
}  
  
ll c(ll n, ll m) {  
 if (n < 0 || m < 0 || n < m) return 0;  
 return fac[n] \* inv[n - m] % mod \* inv[m] % mod;  
}  
  
void init() {  
 fac[0] = 1;  
 for (int i = 1; i < maxn; i++) {  
 fac[i] = fac[i - 1] \* i % mod;  
 }  
 inv[maxn - 1] = qpow(fac[maxn - 1], mod - 2);  
 for (ll i = maxn - 2; i >= 0; i--) {  
 inv[i] = (inv[i + 1] \* (i + 1)) % mod;  
 }  
}

## 图论

### 二分图匹配（匈牙利）

//大量使用了memset，但常数貌似很小？HDU6808跑了998ms（限制5000ms），然而这个代int main()不是HDU6808的  
#include<bits/stdc++.h>  
using namespace std;  
  
const int maxn=505;// 最大点数  
const int inf=0x3f3f3f3f;// 距离初始值  
struct HK\_Hungary{//这个板子从1开始，0点不能用,nx为左边点数，ny为右边点数  
 int nx,ny;//左右顶点数量  
 vector<int>bmap[maxn];  
 int cx[maxn];//cx[i]表示左集合i顶点所匹配的右集合的顶点序号  
 int cy[maxn]; //cy[i]表示右集合i顶点所匹配的左集合的顶点序号  
 int dx[maxn];  
 int dy[maxn];  
 int dis;  
 bool bmask[maxn];  
 void init(int a,int b){  
 nx=a,ny=b;  
 for(int i=0;i<=nx;i++){  
 bmap[i].clear();  
 }  
 }  
 void add\_edge(int u,int v){  
 bmap[u].push\_back(v);  
 }  
 bool searchpath(){//寻找 增广路径  
 queue<int>Q;  
 dis=inf;  
 memset(dx,-1,sizeof(dx));  
 memset(dy,-1,sizeof(dy));  
 for(int i=1;i<=nx;i++){//cx[i]表示左集合i顶点所匹配的右集合的顶点序号  
 if(cx[i]==-1){//将未遍历的节点 入队 并初始化次节点距离为0  
 Q.push(i);  
 dx[i]=0;  
 }  
 }//广度搜索增广路径  
 while(!Q.empty()){  
 int u=Q.front();  
 Q.pop();  
 if(dx[u]>dis) break;//取右侧节点  
 for(int i=0;i<bmap[u].size();i++){  
 int v=bmap[u][i];//右侧节点的增广路径的距离  
 if(dy[v]==-1){  
 dy[v]=dx[u]+1;//v对应的距离 为u对应距离加1  
 if(cy[v]==-1)dis=dy[v];  
 else{  
 dx[cy[v]]=dy[v]+1;  
 Q.push(cy[v]);  
 }  
 }  
 }  
 }  
 return dis!=inf;  
 }  
 int findpath(int u){//寻找路径 深度搜索  
 for(int i=0;i<bmap[u].size();i++){  
 int v=bmap[u][i];//如果该点没有被遍历过 并且距离为上一节点+1  
 if(!bmask[v]&&dy[v]==dx[u]+1){//对该点染色  
 bmask[v]=1;  
 if(cy[v]!=-1&&dy[v]==dis)continue;  
 if(cy[v]==-1||findpath(cy[v])){  
 cy[v]=u;cx[u]=v;  
 return 1;  
 }  
 }  
 }  
 return 0;  
 }  
 int MaxMatch(){//得到最大匹配的数目  
 int res=0;  
 memset(cx,-1,sizeof(cx));  
 memset(cy,-1,sizeof(cy));  
 while(searchpath()){  
 memset(bmask,0,sizeof(bmask));  
 for(int i=1;i<=nx;i++){  
 if(cx[i]==-1){  
 res+=findpath(i);  
 }  
 }  
 }  
 return res;  
 }  
}HK;  
  
int main(){  
 int nn,n,m;  
 cin>>nn;  
 while(nn--){  
 scanf("%d%d",&n,&m);  
 HK.init(n,m);//左端点和右端点数量  
 for(int i=1;i<=n;i++){  
 int snum;  
 cin>>snum;  
 int v;  
 for(int j=1;j<=snum;j++){  
 cin>>v;  
 HK.add\_edge(i,v);//连边  
 }  
 }  
 cout<<HK.MaxMatch()<<endl;//求最大匹配  
 }  
 return 0;  
}

### 强连通（kosaraju

#include <bits/stdc++.h>  
using namespace std;  
struct SCC {  
 static const int MAXV = 100000;  
 int V;  
 vector<int> g[MAXV], rg[MAXV], vs;  
 bool used[MAXV];  
 int cmp[MAXV];  
  
 void add\_edge(int from, int to) {  
 g[from].push\_back(to);  
 rg[to].push\_back(from);  
 }  
  
 void dfs(int v) {  
 used[v] = 1;  
 for (int i = 0; i < g[v].size(); i++) {  
 if (!used[g[v][i]]) dfs(g[v][i]);  
 }  
 vs.push\_back(v);  
 }  
  
 void rdfs(int v, int k) {  
 used[v] = 1;  
 cmp[v] = k;  
 for (int i = 0; i < rg[v].size(); i++) {  
 if (!used[rg[v][i]]) rdfs(rg[v][i], k);  
 }  
 }  
  
 int solve() {  
 memset(used, 0, sizeof(used));  
 vs.clear();  
 for (int v = 1; v <= V; v++) {  
 if (!used[v]) dfs(v);  
 }  
 memset(used, 0, sizeof(used));  
 int k = 0;  
 for (int i = (int)vs.size() - 1; i >= 0; i--) {  
 if (!used[vs[i]]) rdfs(vs[i], ++k);  
 }  
 return k;  
 }  
  
 void init(int n) {  
 V = n;  
 vs.clear();  
 for (int i = 0; i < MAXV; i++) {  
 g[i].clear();  
 rg[i].clear();  
 used[i] = 0;  
 cmp[i] = 0;  
 }  
 }  
  
} scc;  
  
//记得调用init()

### 强连通（tarjan

#include <bits/stdc++.h>  
using namespace std;  
  
struct SCC {  
 static const int MAXN = 100000;  
 vector<int> g[MAXN];  
 int dfn[MAXN], lowlink[MAXN], sccno[MAXN], dfs\_clock, scc\_cnt;  
 stack<int> S;  
  
 void dfs(int u) {  
 dfn[u] = lowlink[u] = ++dfs\_clock;  
 S.push(u);  
 for (int i = 0; i < g[u].size(); i++) {  
 int v = g[u][i];  
 if (!dfn[v]) {  
 dfs(v);  
 lowlink[u] = min(lowlink[u], lowlink[v]);  
 } else if (!sccno[v]) {  
 lowlink[u] = min(lowlink[u], dfn[v]);  
 }  
 }  
 if (lowlink[u] == dfn[u]) {  
 ++scc\_cnt;  
 for (;;) {  
 int x = S.top();  
 S.pop();  
 sccno[x] = scc\_cnt;  
 if (x == u) break;  
 }  
 }  
 }  
  
 void solve(int n) {  
 dfs\_clock = scc\_cnt = 0;  
 memset(sccno, 0, sizeof(sccno));  
 memset(dfn, 0, sizeof(dfn));  
 memset(lowlink, 0, sizeof(lowlink));  
 for (int i = 1; i <= n; i++) {  
 if (!dfn[i]) dfs(i);  
 }  
 }  
} scc;  
  
// scc\_cnt为SCC计数器，sccno[i]为i所在SCC的编号  
// vector<int> g[MAXN]中加边  
//之后再补充init()

### 强连通（tarjan无vector

#include <bits/stdc++.h>  
using namespace std;  
struct SCC {  
 static const int MAXN = 5000;  
 static const int MAXM = 2000000;  
 int dfs\_clock, edge\_cnt = 1, scc\_cnt;  
 int head[MAXN];  
 int dfn[MAXN], lowlink[MAXN];  
 int sccno[MAXN];  
 stack<int> s;  
  
 struct edge {  
 int v, next;  
 } e[MAXM];  
  
 void add\_edge(int u, int v) {  
 e[edge\_cnt].v = v;  
 e[edge\_cnt].next = head[u];  
 head[u] = edge\_cnt++;  
 }  
  
 void tarjan(int u) {  
 int v;  
 dfn[u] = lowlink[u] = ++dfs\_clock; //每次dfs，u的次序号增加1  
 s.push(u); //将u入栈  
 for (int i = head[u]; i != -1; i = e[i].next) //访问从u出发的边  
 {  
 v = e[i].v;  
 if (!dfn[v]) //如果v没被处理过  
 {  
 tarjan(v); // dfs(v)  
 lowlink[u] = min(lowlink[u], lowlink[v]);  
 } else if (!sccno[v])  
 lowlink[u] = min(lowlink[u], dfn[v]);  
 }  
 if (dfn[u] == lowlink[u]) {  
 scc\_cnt++;  
 do {  
 v = s.top();  
 s.pop();  
 sccno[v] = scc\_cnt;  
 } while (u != v);  
 }  
 }  
  
 int find\_scc(int n) {  
 for (int i = 1; i <= n; i++)  
 if (!dfn[i]) tarjan(i);  
 return scc\_cnt;  
 }  
  
 void init() {  
 scc\_cnt = dfs\_clock = 0;  
 edge\_cnt = 1; //不用初始化e数组，省时间  
 while (!s.empty()) s.pop();  
 memset(head, -1, sizeof(head));  
 memset(sccno, 0, sizeof(sccno));  
 memset(dfn, 0, sizeof(dfn));  
 memset(lowlink, 0, sizeof(lowlink));  
 }  
} scc;

### 最大流

#include <bits/stdc++.h>  
using namespace std;  
typedef long long ll;  
  
struct Edge {  
 ll from, to, cap, flow;  
 Edge(ll a, ll b, ll c, ll d) : from(a), to(b), cap(c), flow(d) {}  
};  
  
struct Dinic {  
 static const ll maxn = 2000;  
 static const ll inf = 0x3f3f3f3f3f3f3f3f;  
 ll n, m, s, t;  
 vector<Edge> edges;  
 vector<ll> G[maxn];  
 bool vis[maxn];  
 ll d[maxn];  
 ll cur[maxn];  
  
 void AddEdge(ll from, ll to, ll cap) {  
 edges.push\_back(Edge(from, to, cap, 0));  
 edges.push\_back(Edge(to, from, 0, 0));  
 m = edges.size();  
 G[from].push\_back(m - 2);  
 G[to].push\_back(m - 1);  
 }  
  
 bool BFS() {  
 memset(vis, 0, sizeof(vis));  
 queue<ll> Q;  
 Q.push(s);  
 d[s] = 0;  
 vis[s] = 1;  
 while (!Q.empty()) {  
 ll x = Q.front();  
 Q.pop();  
 for (ll i = 0; i < G[x].size(); i++) {  
 Edge& e = edges[G[x][i]];  
 if (!vis[e.to] && e.cap > e.flow) {  
 vis[e.to] = 1;  
 d[e.to] = d[x] + 1;  
 Q.push(e.to);  
 }  
 }  
 }  
 return vis[t];  
 }  
  
 ll DFS(ll x, ll a) {  
 if (x == t || a == 0) return a;  
 ll flow = 0, f;  
 for (ll& i = cur[x]; i < G[x].size(); i++) {  
 Edge& e = edges[G[x][i]];  
 if (d[x] + 1 == d[e.to] &&  
 (f = DFS(e.to, min(a, e.cap - e.flow))) > 0) {  
 e.flow += f;  
 edges[G[x][i] ^ 1].flow -= f;  
 flow += f;  
 a -= f;  
 if (a == 0) break;  
 }  
 }  
 return flow;  
 }  
  
 ll Maxflow(ll s, ll t) {  
 this->s = s, this->t = t;  
 ll flow = 0;  
 while (BFS()) {  
 memset(cur, 0, sizeof(cur));  
 flow += DFS(s, inf);  
 }  
 return flow;  
 }  
} MF;

### 最小费用最大流

#include <bits/stdc++.h>  
using namespace std;  
typedef long long ll;  
  
struct Edge {  
 ll from, to, cap, flow, cost;  
 Edge(ll u, ll v, ll c, ll f, ll w)  
 : from(u), to(v), cap(c), flow(f), cost(w) {}  
};  
  
struct MCMF {  
 static const ll maxn = 6000;  
 static const ll INF = 0x3f3f3f3f3f3f3f;  
 ll n, m;  
 vector<Edge> edges;  
 vector<ll> G[maxn];  
 ll inq[maxn];  
 ll d[maxn];  
 ll p[maxn];  
 ll a[maxn];  
  
 void init(ll n) {  
 this->n = n;  
 for (ll i = 1; i <= n; i++) G[i].clear();  
 edges.clear();  
 }  
  
 void add\_edge(ll from, ll to, ll cap, ll cost) {  
 edges.push\_back(Edge(from, to, cap, 0, cost));  
 edges.push\_back(Edge(to, from, 0, 0, -cost));  
 m = edges.size();  
 G[from].push\_back(m - 2);  
 G[to].push\_back(m - 1);  
 }  
  
 bool BellmanFord(ll s, ll t, ll& flow, ll& cost) {  
 for (ll i = 1; i <= n; ++i) d[i] = INF;  
 memset(inq, 0, sizeof(inq));  
 d[s] = 0, inq[s] = 1, p[s] = 0, a[s] = INF;  
 queue<ll> Q;  
 Q.push(s);  
 while (!Q.empty()) {  
 ll u = Q.front();  
 Q.pop();  
 inq[u] = 0;  
 for (ll i = 0; i < G[u].size(); ++i) {  
 Edge& e = edges[G[u][i]];  
 if (e.cap > e.flow && d[e.to] > d[u] + e.cost) {  
 d[e.to] = d[u] + e.cost;  
 p[e.to] = G[u][i];  
 a[e.to] = min(a[u], e.cap - e.flow);  
 if (!inq[e.to]) {  
 Q.push(e.to);  
 inq[e.to] = 1;  
 }  
 }  
 }  
 }  
 if (d[t] == INF) return false;  
 flow += a[t];  
 cost += (ll)d[t] \* (ll)a[t];  
 for (ll u = t; u != s; u = edges[p[u]].from) {  
 edges[p[u]].flow += a[t];  
 edges[p[u] ^ 1].flow -= a[t];  
 }  
 return true;  
 }  
  
 //需要保证初始网络中没有负权圈  
 ll MincostMaxflow(ll s, ll t, ll& cost) {  
 ll flow = 0;  
 cost = 0;  
 while (BellmanFord(s, t, flow, cost))  
 ;  
 return flow;  
 }  
} mcmf; // 若固定流量k，增广时在flow+a>=k的时候只增广k-flow单位的流量，然后终止程序

### 树分治

#include <bits/stdc++.h>  
using namespace std;  
const int MAXN = 10005;  
const int INF = 1000000000;  
struct edge {  
 int to, length;  
 edge() {}  
 edge(int a, int b) : to(a), length(b) {}  
};  
  
int N, K;  
vector<edge> g[MAXN];  
  
bool centroid[MAXN];  
int subtree\_size[MAXN];  
  
int ans;  
  
//计算子树大小  
int compute\_subtree\_size(int v, int p) {  
 int c = 1;  
 for (int i = 0; i < g[v].size(); i++) {  
 int w = g[v][i].to;  
 if (w == p || centroid[w]) continue;  
 c += compute\_subtree\_size(w, v);  
 }  
 subtree\_size[v] = c;  
 return c;  
}  
  
//查找重心，t为连通分量大小  
// pair（最大子树顶点数，顶点编号）  
pair<int, int> search\_centroid(int v, int p, int t) {  
 pair<int, int> res = pair<int, int>(INF, -1);  
 int s = 1, m = 0;  
 for (int i = 0; i < g[v].size(); i++) {  
 int w = g[v][i].to;  
 if (w == p || centroid[w]) continue;  
 res = min(res, search\_centroid(w, v, t));  
 m = max(m, subtree\_size[w]);  
 s += subtree\_size[w];  
 }  
 m = max(m, t - s);  
 res = min(res, pair<int, int>(m, v));  
 return res;  
}  
  
void init() {  
 memset(centroid, 0, sizeof(centroid));  
 memset(subtree\_size, 0, sizeof(subtree\_size));  
 for (int i = 0; i <= N; i++) g[i].clear();  
 ans = 0;  
}  
  
int solve(int u) {  
 compute\_subtree\_size(u, -1);  
 int s = search\_centroid(u, -1, subtree\_size[u]).second;  
 centroid[s] = 1;  
 int ans;  
 for (int i = 0; i < g[s].size(); i++) {  
 int v = g[s][i].to;  
 if (centroid[v]) continue;  
 /\*solve()\*/  
 }  
 /\*do something\*/  
 centroid[s] = 0;  
 return ans;  
}

### 拓扑排序

#include <bits/stdc++.h>  
using namespace std;  
const int MAXN = 100000;  
  
int c[MAXN];  
int topo[MAXN], t, V;  
vector<int> g[MAXN];  
  
bool dfs(int u) {  
 c[u] = -1;  
 for (int i = 0; i < g[u].size(); i++) {  
 int v = g[u][i];  
 if (c[v] < 0)  
 return false;  
 else if (!c[v] && !dfs(v))  
 return false;  
 }  
 c[u] = 1;  
 topo[t--] = u;  
 return true;  
}  
  
bool toposort(int n) {  
 V = n;  
 t = n;  
 memset(c, 0, sizeof(c));  
 for (int u = 1; u <= V; u++)  
 if (!c[u] && !dfs(u)) return false;  
 return true;  
}

### 最近公共祖先（倍增）

#include <algorithm>  
#include <cstdio>  
#include <cstring>  
#include <iostream>  
using namespace std;  
const int MAX = 600000;  
  
struct edge {  
 int t, nex;  
} e[MAX << 1];  
int head[MAX], tot;  
  
int depth[MAX], fa[MAX][22], lg[MAX];  
  
void add\_edge(int x, int y) {  
 e[++tot].t = y;  
 e[tot].nex = head[x];  
 head[x] = tot;  
  
 e[++tot].t = x;  
 e[tot].nex = head[y];  
 head[y] = tot;  
}  
  
void dfs(int now, int fath) {  
 fa[now][0] = fath;  
 depth[now] = depth[fath] + 1;  
 for (int i = 1; i <= lg[depth[now]]; ++i)  
 fa[now][i] = fa[fa[now][i - 1]][i - 1];  
 for (int i = head[now]; i; i = e[i].nex)  
 if (e[i].t != fath) dfs(e[i].t, now);  
}  
  
int lca(int x, int y) {  
 if (depth[x] < depth[y]) swap(x, y);  
 while (depth[x] > depth[y]) x = fa[x][lg[depth[x] - depth[y]] - 1];  
 if (x == y) return x;  
 for (int k = lg[depth[x]] - 1; k >= 0; --k)  
 if (fa[x][k] != fa[y][k]) x = fa[x][k], y = fa[y][k];  
 return fa[x][0];  
}  
  
void init(int n, int root) {  
 for (int i = 1; i <= n; ++i) lg[i] = lg[i - 1] + (1 << lg[i - 1] == i);  
 dfs(root, 0);  
}

### 最近公共祖先（线段树）

#include <bits/stdc++.h>  
using namespace std;  
int n, m, root;  
const int MAX\_N = 500005;  
const int MAX = 1 << 20;  
vector<int> g[MAX\_N];  
vector<int> vs;  
pair<int, int> tree[MAX \* 2 + 10];  
int fir[MAX\_N];  
int fa[MAX\_N];  
int dep[MAX\_N];  
void dfs(int k, int p, int d) {  
 fa[k] = p;  
 dep[k] = d;  
 vs.push\_back(k);  
 for (int i = 0; i < g[k].size(); i++) {  
 if (g[k][i] != p) {  
 dfs(g[k][i], k, d + 1);  
 vs.push\_back(k);  
 }  
 }  
}  
void build(int k) {  
 if (k >= MAX) return;  
 build(k << 1);  
 build(k << 1 | 1);  
 tree[k] = min(tree[k << 1], tree[k << 1 | 1]);  
}  
pair<int, int> query(int k, int s, int e, int l, int r) {  
 if (e < l || r < s) return pair<int, int>(INT\_MAX, 0);  
 if (l <= s && e <= r) return tree[k];  
 return min(query(k << 1, s, (s + e) >> 1, l, r),  
 query(k << 1 | 1, ((s + e) >> 1) + 1, e, l, r));  
}  
void init() {  
 dfs(root, root, 0);  
 for (int i = 0; i < MAX \* 2 + 10; i++) tree[i] = pair<int, int>(INT\_MAX, 0);  
 for (int i = MAX; i < MAX + vs.size(); i++)  
 tree[i] = pair<int, int>(dep[vs[i - MAX]], vs[i - MAX]);  
 for (int i = 0; i < vs.size(); i++) {  
 if (fir[vs[i]] == 0) fir[vs[i]] = i + 1;  
 }  
 build(1);  
}  
int lca(int a, int b) {  
 return query(1, 1, MAX, min(fir[a], fir[b]), max(fir[a], fir[b])).second;  
}  
int main() {  
 scanf("%d%d%d", &n, &m, &root);  
 for (int i = 1; i < n; i++) {  
 int a, b;  
 scanf("%d%d", &a, &b);  
 g[a].push\_back(b);  
 g[b].push\_back(a);  
 }  
 init();  
 for (int i = 1; i <= m; i++) {  
 int a, b;  
 scanf("%d%d", &a, &b);  
 printf("%d\n", lca(a, b));  
 }  
}

## 线性代数

### 高斯消元

#include <iostream>  
#include <vector>  
using namespace std;  
const double eps = 1e-8;  
void sway(vector<double>& a, vector<double>& b) {  
 vector<double> s;  
 for (int i = 0; i < a.size(); i++) {  
 s.push\_back(a[i]);  
 }  
 a.clear();  
 for (int i = 0; i < b.size(); i++) {  
 a.push\_back(b[i]);  
 }  
 b.clear();  
 for (int i = 0; i < s.size(); i++) {  
 b.push\_back(s[i]);  
 }  
}  
vector<double> gauss\_jordan(const vector<vector<double> >& A,  
 const vector<double>& b) {  
 int n = A.size();  
 vector<vector<double> > B(n, vector<double>(n + 1));  
 for (int i = 0; i < n; i++)  
 for (int j = 0; j < n; j++) B[i][j] = A[i][j];  
 for (int i = 0; i < n; i++) B[i][n] = b[i];  
  
 for (int i = 0; i < n; i++) {  
 int pivot = i;  
 for (int j = i; j < n; j++) {  
 if (abs(B[j][i]) > abs(B[pivot][i])) pivot = j;  
 }  
 swap(B[i], B[pivot]);  
 if (abs(B[i][i]) < eps) return vector<double>();  
 for (int j = i + 1; j <= n; j++) B[i][j] /= B[i][i];  
 for (int j = 0; j < n; j++) {  
 if (i != j) {  
 for (int k = i + 1; k <= n; k++) B[j][k] -= B[j][i] \* B[i][k];  
 }  
 }  
 }  
 vector<double> x(n);  
 for (int i = 0; i < n; i++) x[i] = B[i][n];  
 return x;  
}  
int main() {  
 int n, m;  
 cin >> n >> m;  
 vector<vector<double> > mat(n, vector<double>(m));  
 for (int i = 0; i < n; i++) {  
 for (int j = 0; j < m; j++) {  
 cin >> mat[i][j];  
 }  
 }  
 vector<double> val(n);  
 for (int i = 0; i < n; i++) cin >> val[i];  
 vector<double> ans = gauss\_jordan(mat, val);  
 for (int i = 0; i < ans.size(); i++) cout << ans[i] << ' ';  
}

### 矩阵行列式

#include <bits/stdc++.h>  
using namespace std;  
typedef long long ll;  
const ll mod = 1e9 + 7;  
struct Matrix {  
 static const ll MAXN = 300;  
 ll a[MAXN][MAXN];  
  
 void init() { memset(a, 0, sizeof(a)); }  
  
 ll det(ll n) {  
 for (int i = 0; i < n; i++)  
 for (int j = 0; j < n; j++) a[i][j] = (a[i][j] + mod) % mod;  
 ll res = 1;  
 for (int i = 0; i < n; i++) {  
 if (!a[i][i]) {  
 bool flag = false;  
 for (int j = i + 1; j < n; j++) {  
 if (a[j][i]) {  
 flag = true;  
 for (int k = i; k < n; k++) {  
 swap(a[i][k], a[j][k]);  
 }  
 res = -res;  
 break;  
 }  
 }  
 if (!flag) return 0;  
 }  
  
 for (int j = i + 1; j < n; j++) {  
 while (a[j][i]) {  
 ll t = a[i][i] / a[j][i];  
 for (int k = i; k < n; k++) {  
 a[i][k] = (a[i][k] - t \* a[j][k]) % mod;  
 swap(a[i][k], a[j][k]);  
 }  
 res = -res;  
 }  
 }  
 res \*= a[i][i];  
 res %= mod;  
 }  
 return (res + mod) % mod;  
 }  
} mat;

### 线性基

//  
  
const int maxbit = 62; //maxbit不能太大  
  
struct L\_B{  
 ll lba[maxbit];  
 L\_B(){  
 memset(lba, 0, sizeof(lba));  
 }  
   
 void Insert(ll val){ //插入  
 for(int i = maxbit - 1; i >= 0; -- i) // 从高位向低位扫   
 if(val & (1ll << i)){ //   
 if(!lba[i]){  
 lba[i] = val;  
 break;  
 }  
 val ^= lba[i];  
 }  
 }  
};  
//对原集合的每个数val转为2进制，从高位向低位扫，对于当前位为1的，若lba[i]不存在就令lba[i]=x，否则令val=val`xor`lba[i]  
//使用： 直接insert   
// --------------线性基模板

### 矩阵（加减乘快速幂

//矩阵类模板  
struct Matrix{  
 int n,m;  
 int a[maxn][maxm];  
 void clear(){  
 n=m=0;  
 memset(a,0,sizeof(a));  
 }  
 Matrix operator +(const Matrix &b) const {  
 Matrix tmp;  
 tmp.n=n;tmp.m=m;  
 for (int i=0;i<n;++i)  
 for(int j=0;j<m;++j)  
 tmp.a[i][j]=a[i][j]+b.a[i][j];  
 return tmp;  
 }  
 Matrix operator -(const Matrix &b)const{  
 Matrix tmp;  
 tmp.n=n;tmp.m=m;  
 for (int i=0;i<n;++i){  
 for(int j=0;j<m;++j)  
 tmp.a[i][j]=a[i][j]-b.a[i][j];  
 }  
   
 return tmp;  
 }  
 Matrix operator \* (const Matrix &b) const{  
 Matrix tmp;  
 tmp.clear();  
 tmp.n=n;tmp.m=b.m;  
 for (int i=0;i<n;++i)  
 for(int j=0;j<b.m;++j)  
 for (int k=0;k<m;++k){  
 tmp.a[i][j]+=a[i][k]\*b.a[k][j];  
 tmp.a[i][j]%=mod;  
 }  
 return tmp;  
 }  
 Matrix get(int x){//幂运算  
 Matrix E;  
 E.clear();  
 E.n=E.m=n;  
 for(int i=0;i<n;++i)  
 E.a[i][i]=1;  
 if(x==0) return E;  
 else if(x==1) return \*this;  
 Matrix tmp=get(x/2);  
 tmp=tmp\*tmp;  
 if(x%2) tmp=tmp\*(\*this);  
 return tmp;  
 }  
};  
//矩阵模板结束

### 稀疏矩阵乘法

struct Matrix{  
 int n,m;  
 int a[maxn][maxn];////  
 void clear(){  
 n=m=0;  
 memset(a,0,sizeof(a));  
 }  
 Matrix operator \* (const Matrix &b) const{  
 Matrix tmp;  
 tmp.clear();  
 tmp.n=n;tmp.m=b.m;  
 for (int k=0;k<m;++k){  
 for (int i=0;i<n;++i){  
 if(a[i][k]==0) continue;  
 for(int j=0;j<b.m;++j){  
 if(b.a[k][j]==0) continue;  
 tmp.a[i][j]+=a[i][k]\*b.a[k][j];  
 tmp.a[i][j]%=mod;  
 }   
 }   
 }  
 return tmp;  
 }  
};  
//稀疏矩阵乘法

## 杂项

### 快读

inline int read(){  
 int X=0,w=0;char ch=0;  
 while(!isdigit(ch)){w|=ch=='-';ch=getchar();}  
 while(isdigit(ch))X=(X<<3)+(X<<1)+(ch^48),ch=getchar();  
 return w?-X:X;  
}

## 字符串

### 马拉车

#include <bits/stdc++.h>  
using namespace std;  
const int maxn = 100005;  
char s[maxn];  
char s\_new[maxn \* 2];  
int p[maxn \* 2];  
  
int Manacher(char\* a, int l) {  
 s\_new[0] = '$';  
 s\_new[1] = '#';  
 int len = 2;  
 for (int i = 0; i < l; i++) {  
 s\_new[len++] = a[i];  
 s\_new[len++] = '#';  
 }  
 s\_new[len] = '\0';  
 int id;  
 int mx = 0;  
 int mmax = 0;  
  
 for (int i = 1; i < len; i++) {  
 p[i] = i < mx ? min(p[2 \* id - i], mx - i) : 1;  
 while (s\_new[i + p[i]] == s\_new[i - p[i]]) p[i]++;  
 if (mx < i + p[i]) {  
 id = i;  
 mx = i + p[i];  
 }  
 mmax = max(mmax, p[i] - 1);  
 }  
 return mmax;  
}  
  
int main() {  
 cin >> s;  
 cout << Manacher(s, strlen(s));  
}

### AC自动机

#include <bits/stdc++.h>  
using namespace std;  
struct AC {  
 static const int maxnode = 200005;  
 static const int sigma\_size = 26;  
 char T[maxnode];  
 int ch[maxnode][sigma\_size];  
 int val[maxnode], fail[maxnode], last[maxnode];  
 int sz;  
 vector<pair<int, int> > ans;  
  
 void init() {  
 sz = 1;  
 memset(ch[0], 0, sizeof(ch[0]));  
 ans.clear();  
 }  
  
 int idx(const char &c) { return c - 'a'; }  
  
 void insert(string s, int v) {  
 int u = 0, n = s.length();  
 for (int i = 0; i < n; i++) {  
 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;  
 }  
  
 void get\_fail() {  
 queue<int> que;  
 fail[0] = 0;  
 for (int c = 0; c < sigma\_size; c++) {  
 int u = ch[0][c];  
 if (u) {  
 fail[u] = 0;  
 que.push(u);  
 last[u] = 0;  
 }  
 }  
 while (!que.empty()) {  
 int r = que.front();  
 que.pop();  
 for (int c = 0; c < sigma\_size; c++) {  
 int u = ch[r][c];  
 if (!u) continue;  
 que.push(u);  
 int v = fail[r];  
 while (v && !ch[v][c]) v = fail[v];  
 fail[u] = ch[v][c];  
 last[u] = val[fail[u]] ? fail[u] : last[fail[u]];  
 }  
 }  
 }  
  
 void print(int j) {  
 if (j) {  
 ans.push\_back(pair<int, int>(j, val[j]));  
 print(last[j]);  
 }  
 }  
  
 void find() {  
 int n = strlen(T);  
 int j = 0;  
 for (int i = 0; i < n; i++) {  
 int c = idx(T[i]);  
 while (j && !ch[j][c]) j = fail[j];  
 j = ch[j][c];  
 if (val[j])  
 print(j);  
 else if (last[j])  
 print(last[j]);  
 }  
 }  
} ac; //字符串下标从0开始

### KMP

//next数组等价于前缀函数  
#include<bits/stdc++.h>  
using namespace std;  
typedef long long ll;  
  
int kmp(char \*s1,int \*p1,char \*s2=0,int \*p2=0){//必须先求s1的next数组，即kmp(s1,p1);再kmp(s1,p1,s2,p2);  
 int n=strlen(s1);  
 if(p2==0){  
 p1[0]=0;  
 for(int i=1;s1[i]!='\0';i++){  
 int j=p1[i-1];  
 while(j>0&&s1[i]!=s1[j])j=p1[j-1];  
 if(s1[i]==s1[j])j++;  
 p1[i]=j;  
 }  
 }  
 else{  
 for(int i=0;s2[i]!='\0';i++){  
 int j=i==0?0:p2[i-1];  
 while(j>0&&s2[i]!=s1[j])j=p1[j-1];  
 if(s2[i]==s1[j])j++;  
 p2[i]=j;  
 if(j==n)return i-n+2;//返回位置  
 }  
 }  
 return 0;  
}  
int main(){  
 char s1[15],s2[105];  
 int p1[15],p2[105];  
 cin>>s1>>s2;  
 kmp(s1,p1);  
 cout<<kmp(s1,p1,s2,p2)<<endl;  
 return 0;  
}

### KMP 2

#include <bits/stdc++.h>  
using namespace std;  
struct KMP {  
 static const int MAXN = 1000010;  
 char T[MAXN], P[MAXN];  
 int fail[MAXN];  
 vector<int> ans;  
  
 void init() { ans.clear(); }  
  
 void get\_fail() {  
 int m = strlen(P);  
 fail[0] = fail[1] = 0;  
 for (int i = 1; i < m; i++) {  
 int j = fail[i];  
 while (j && P[i] != P[j]) j = fail[j];  
 fail[i + 1] = (P[i] == P[j] ? j + 1 : 0);  
 }  
 }  
  
 void find() {  
 int n = strlen(T), m = strlen(P);  
 get\_fail();  
 int j = 0;  
 for (int i = 0; i < n; i++) {  
 while (j && P[j] != T[i]) j = fail[j];  
 if (P[j] == T[i]) j++;  
 if (j == m) ans.push\_back(i - m + 1);  
 }  
 }  
} kmp; //P为模式串，下标从0开始，输入后直接调用find()

### Tire

#include <bits/stdc++.h>  
using namespace std;  
struct Trie {  
 static const int maxnode = 200005;  
 static const int sigma\_size = 26;  
 int ch[maxnode][sigma\_size];  
 int val[maxnode];  
 int sz;  
  
 Trie() {  
 sz = 1;  
 memset(ch[0], 0, sizeof(ch[0]));  
 }  
  
 int idx(const char &c) { return c - 'a'; }  
  
 void insert(string s, int v) {  
 int u = 0, n = s.length();  
 for (int i = 0; i < n; i++) {  
 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;  
 }  
  
 int find(string s) {  
 int u = 0, n = s.length();  
 for (int i = 0; i < n; i++) {  
 int c = idx(s[i]);  
 if (!ch[u][c]) return 0;  
 u = ch[u][c];  
 }  
 return val[u];  
 }  
} trie;

### 后缀数组

#include <bits/stdc++.h>  
using namespace std;  
struct SuffixArray {  
 static const int MAXN = 1100000;  
 char s[MAXN];  
 int sa[MAXN], t[MAXN], t1[MAXN], c[MAXN], ra[MAXN], height[MAXN], m;  
 inline void init() { memset(this, 0, sizeof(SuffixArray)); }  
  
 inline void get\_sa(int n) {  
 m = 256;  
 int \*x = t, \*y = t1;  
 for (int i = 1; i <= m; i++) c[i] = 0;  
 for (int i = 1; i <= n; i++) c[x[i] = s[i]]++;  
 for (int i = 1; i <= m; i++) c[i] += c[i - 1];  
 for (int i = n; i >= 1; i--) sa[c[x[i]]--] = i;  
 for (int k = 1; k <= n; k <<= 1) {  
 int p = 0;  
 for (int i = n - k + 1; i <= n; i++) y[++p] = i;  
 for (int i = 1; i <= n; i++)  
 if (sa[i] > k) y[++p] = sa[i] - k;  
 for (int i = 1; i <= m; i++) c[i] = 0;  
 for (int i = 1; i <= n; i++) c[x[y[i]]]++;  
 for (int i = 1; i <= m; i++) c[i] += c[i - 1];  
 for (int i = n; i >= 1; i--) sa[c[x[y[i]]]--] = y[i];  
 std::swap(x, y);  
 p = x[sa[1]] = 1;  
 for (int i = 2; i <= n; i++) {  
 x[sa[i]] = (y[sa[i - 1]] == y[sa[i]] &&  
 y[sa[i - 1] + k] == y[sa[i] + k])  
 ? p  
 : ++p;  
 }  
 if (p >= n) break;  
 m = p;  
 }  
 }  
  
 inline void get\_height(int n) {  
 int i, j, k = 0;  
 for (int i = 1; i <= n; i++) ra[sa[i]] = i;  
 for (int i = 1; i <= n; i++) {  
 if (k) k--;  
 int j = sa[ra[i] - 1];  
 while (s[i + k] == s[j + k]) k++;  
 height[ra[i]] = k;  
 }  
 }  
  
} SA; //字符串下标从一开始

## 对拍

### windows环境下bat对拍

@echo off  
:loop  
 dataa.exe > data.txt  
 biaocheng.exe < data.txt > ac.txt  
 A.exe < data.txt > test.txt  
 fc ac.txt test.txt  
 if not errorlevel 1 goto loop  
pause  
goto loop

**其中要改的部分（标红辽）**：

@echo off  
:loop  
 dataa.exe > data.txt  
 $\color{red}{biaocheng.exe}$ < data.txt > ac.txt  
 $\color{red}{A.exe}$ < data.txt > test.txt  
 fc ac.txt test.txt  
 if not errorlevel 1 goto loop  
pause  
goto loop

文件以.bat作为后缀

将三个程序（数据生成文件（dataa），标程或暴力代码（biaocheng）, 要看的代码（A））放在同一目录下，

记得加 freopen

随机数记得加srand((int)time(0));

随机数生成code

#include <iostream>  
#include <cstdlib>  
#include <ctime>  
using namespace std;  
  
int main(){  
 freopen("data.txt", "w", stdout);  
   
 srand((int)time(0));  
 int T = rand() % 100000;  
 cout << T << endl;  
   
 for (int i = 0; i < T; i++){  
 cout << rand() % 100;  
 }  
}

rand() 似乎只有三万多，需要更大的数的话要乘一下