**Routine Library3.0ver.**

By 1em0n Team

Last Upd: July 16, 2019

Part 1 几何

**1.1 几何基础**

**1.2 点与向量**

**1.3 直线与线段**

**1.4 多边形**

**1.5 坐标离散化**

**1.6 凸包**

**1.7 半平面交**

Part 2 数论

2.1 分解质因数

**2.2 同余方程（扩欧）**

2.3 欧拉函数

2.4 简易素数表

2.5 中国剩余定理

2.6 **快速傅里叶变换（FFT）**

2.7 线性规划

2.8 高斯消元

**2.9 佩兰方程**

**2.10 组合数及逆元**

Part 3 字符串

**3.1 Trie**

**3.2 KMP**

3.3 后缀树

**3.4 后缀数组**

**3.5 后缀自动机**

3.6 回文树

**3.7 马拉车**

**3.8 AC自动机**

**3.9 字典树**

Part 4 图论

**4.1 二分图匹配**

**4.2 最大团**

**4.3 网络流**

**4.4 强联通分量/Tarjan**

4.5 2-sat

**4.6 割点**

4.7 割边

**4.8 dsu**

**4.9 树链剖分**

**4.10 Floyd最小环**

**4.11 最小生成树**

**4.12 点分治**

**4.13 LCA（倍增/ST/树剖）**

Part 5 数据结构

5.1 Splay

**5.2 树状数组**

**5.3 线段树**

**5.4 可持久化线段树（主席树）**

**5.5 并查集带权**

**5.6 单调栈——求全flag最大子矩阵**

**5.7 ST表**

**5.8 李超线段树**

**5.9 LCT**

Part 6 其他专题

**6.1 高精度**

**6.2 数位DP**

**6.3 CDQ分治（二维偏序）**

**6.4 三分**

**6.5 矩阵快速幂**

**6.6 动态DP**

Upd: ver3.0

\*新增李超线段树、LCT、动态DP、后缀数组、FFT

\*修复了ver2.0中关于费用流的问题

**1.1 几何基础**

struct Point{

double x, y;

Point(double x = 0, double y = 0):x(x),y(y){}

};

typedef Point Vector;

Vector operator + (Vector A, Vector B){

return Vector(A.x+B.x, A.y+B.y);

}

Vector operator - (Point A, Point B){

return Vector(A.x-B.x, A.y-B.y);

}

Vector operator \* (Vector A, double p){

return Vector(A.x\*p, A.y\*p);

}

Vector operator / (Vector A, double p){

return Vector(A.x/p, A.y/p);

}

bool operator < (const Point& a, const Point& b){

if(a.x == b.x)

return a.y < b.y;

return a.x < b.x;

}

const double eps = 1e-9;

int sgn(double x){

if(fabs(x) < eps)

return 0;

if(x < 0)

return -1;

return 1;

}

bool operator == (const Point& a, const Point& b){

if(sgn(a.x-b.x) == 0 && sgn(a.y-b.y) == 0)

return true;

return false;

}

int dcmp(double x, double y){

if(fabs(x - y) < eps)

return 0;

if(x > y)

return 1;

return -1;

}

**1.2 点与向量**

double getDis(Point a, Point b) **//两点间距离**

{

return sqrt((a.\_x - b.\_x)\*(a.\_x - b.\_x) + (a.\_y - b.\_y)\*(a.\_y - b.\_y));

}

double Dot(Vector A, Vector B){

return A.x\*B.x + A.y\*B.y;

}

double Length(Vector A){

return sqrt(Dot(A, A));

}

double Angle(Vector A, Vector B){

return acos(Dot(A, B)/Length(A)/Length(B));

}

double Cross(Vector A, Vector B){

return A.x\*B.y-A.y\*B.x;

}

double getCross(Point p1, Point p2, Point p)**//向量p1p和p2p的叉积**

{

return (p1.\_x - p.\_x)\*(p2.\_y - p.\_y) - (p2.\_x - p.\_x)\*(p1.\_y - p.\_y);

}

（等价于）

double Area2(Point A, Point B, Point C){**//平行四边形面积**

return Cross(B-A, C-A);

}

Vector Rotate(Vector A, double rad){**//rad为弧度 且为逆时针旋转的角**

return Vector(A.x\*cos(rad)-A.y\*sin(rad), A.x\*sin(rad)+A.y\*cos(rad));

}

Vector Normal(Vector A){**//向量A左转90°的单位法向量**

double L = Length(A);

return Vector(-A.y/L, A.x/L);

}

bool ToLeftTest(Point a, Point b, Point c){**//用于凸多边形判断**

return Cross(b - a, c - b) > 0;

}

**1.3 直线与线段**

**//直线定义**

struct Line{

Point v, p;

Line(Point v, Point p):v(v), p(p) {}

Point point(double t){//返回点P = v + (p - v)\*t

return v + (p - v)\*t;

}

};

**//求两直线交点**

**//调用前需保证 Cross(v, w) != 0**

Point GetLineIntersection(Point P, Vector v, Point Q, Vector w){

Vector u = P-Q;

double t = Cross(w, u)/Cross(v, w);

return P+v\*t;

}

**//点P到直线AB距离**

double DistanceToLine(Point P, Point A, Point B){

Vector v1 = B-A, v2 = P-A;

return fabs(Cross(v1, v2)/Length(v1));

}//不去绝对值，得到的是有向距离

**//点P到线段AB距离公式**

double DistanceToSegment(Point P, Point A, Point B){

if(A == B)

return Length(P-A);

Vector v1 = B-A, v2 = P-A, v3 = P-B;

if(dcmp(Dot(v1, v2)) < 0)

return Length(v2);

if(dcmp(Dot(v1, v3)) > 0)

return Length(v3);

return DistanceToLine(P, A, B);

}

**//点P在直线AB上的投影点**

Point GetLineProjection(Point P, Point A, Point B){

Vector v = B-A;

return A+v\*(Dot(v, P-A)/Dot(v, v));

}

**//判断p点是否在线段a1a2上**

**//注意判断端点的情况，允许在端点上把＜改成≤**

bool OnSegment(Point p, Point a1, Point a2){

return dcmp(Cross(a1-p, a2-p)) == 0 && dcmp(Dot(a1-p, a2-p)) < 0;

}

**//判断p点是否在线段ab上**

bool onLine(Point p, Point a, Point b)

{

return dcmp(getCross(a, b, p)) == 0;

}

**//判断两线段是否相交**

bool SegmentProperIntersection(Point a1, Point a2, Point b1, Point b2){

double c1 = Cross(a2-a1, b1-a1), c2 = Cross(a2-a1, b2-a1);

double c3 = Cross(b2-b1, a1-b1), c4 = Cross(b2-b1, a2-b1);

**//if判断控制是否允许线段在端点处相交，根据需要添加**

if(!sgn(c1) || !sgn(c2) || !sgn(c3) || !sgn(c4)){

bool f1 = OnSegment(b1, a1, a2);

bool f2 = OnSegment(b2, a1, a2);

bool f3 = OnSegment(a1, b1, b2);

bool f4 = OnSegment(a2, b1, b2);

bool f = (f1|f2|f3|f4);

return f;

}

return (sgn(c1)\*sgn(c2) < 0 && sgn(c3)\*sgn(c4) < 0);

}

**//判断直线ab与线段cd是否相交**

bool segment\_intersectLine(Point a, Point b, Point c, Point d)//线段ab与直线cd

{

if (dcmp(getCross(c, d, a)\*getCross(c, d, b)) <= 0)

return true;

return false;

}

**//判断直线ab与直线cd是否相交**

bool lineIntersect(Point a, Point b, Point c, Point d)

{

if (judgeZero((a.\_x - b.\_x)\*(c.\_y - d.\_y) - (c.\_x - d.\_x)\*(a.\_y - b.\_y)) != 0)

return true;

if (onLine(a, c, d))//两直线重合

return true;

return false;

}

**//求出交点,直线(线段)p1p2与直线(线段)p3p4**

Point getPoint(Point p1, Point p2, Point p3, Point p4)

{//t=lamta/(lamta+1),必须用t取代lamta,不然算lamta可能分母为0

double x1 = p1.\_x, y1 = p1.\_y;

double x2 = p2.\_x, y2 = p2.\_y;

double x3 = p3.\_x, y3 = p3.\_y;

double x4 = p4.\_x, y4 = p4.\_y;

double t = ((x2 - x1)\*(y3 - y1) - (x3 - x1)\*(y2 - y1)) / ((x2 - x1)\*(y3 - y4) - (x3 - x4)\*(y2 - y1));

return Point(x3 + t\*(x4 - x3), y3 + t\*(y4 - y3));

}

**1.4 多边形**

**//多边形定义**

typedef vector<Point> Polygon;

**//多边形有向面积**

double PolygonArea(Point\* p, int n){//p为端点集合，n为端点个数

double s = 0;

for(int i = 1; i < n-1; ++i)

s += Cross(p[i]-p[0], p[i+1]-p[0]);

return s;

}

**//判断点是否在多边形内，若点在多边形内返回1，在多边形外部返回0，在多边形上返回-1**

int isPointInPolygon(Point p, vector<Point> poly){

int wn = 0;

int n = poly.size();

for(int i = 0; i < n; ++i){

if(OnSegment(p, poly[i], poly[(i+1)%n])) return -1;

int k = sgn(Cross(poly[(i+1)%n] - poly[i], p - poly[i]));

int d1 = sgn(poly[i].y - p.y);

int d2 = sgn(poly[(i+1)%n].y - p.y);

if(k > 0 && d1 <= 0 && d2 > 0) wn++;

if(k < 0 && d2 <= 0 && d1 > 0) wn--;

}

if(wn != 0)

return 1;

return 0;

}

**//线段是否含于多边形**

bool segment\_inPolygon(Point a, Point b, Polygon polygon)

{

if (!isPointInPolygon(a, polygon) || !isPointInPolygon(b, polygon))//两个端点都不在多边形内

return false;

int tot = 0;

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

{

Point c = polygon[i], d = polygon[(i + 1) % polygon.size()];

if (OnSegment(a, c, d))//以下只用记录一个交点，多的要么重复，要么一定在边上

crossp[tot++] = a;

else if (OnSegment(b, c, d))

crossp[tot++] = b;

else if (OnSegment(c, a, b))

crossp[tot++] = c;

else if (OnSegment(d, a, b))

crossp[tot++] = d;

else if (SegmentProperIntersection (a, b, c, d))//端点没有在线段上且相交

return false;

}

sort(crossp, crossp + tot);//按x，y排序

tot = unique(crossp, crossp + tot) - crossp;

for (int i = 0;i < tot - 1;++i)

{

Point tmp = 0.5\*(crossp[i] + crossp[i + 1]);

if (!isPointInPolygon(tmp, polygon))//中点不在多边形内

return false;

}

return true;

}

**//求出直线在多边形内的长度**

double lineDis\_inPolygon(Point a, Point b, Polygon polygon)

{

int tot = 0;

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

{

Point c = polygon[i], d = polygon[(i + 1) % polygon.size()];

if (onLine(c, a, b) && onLine(d, a, b))//如果边与直线重合，记录两个端点

{

crossp[tot++] = c;

crossp[tot++] = d;

}

else if (segment\_intersectLine(c, d, a, b))

crossp[tot++] = getPoint(a, b, c, d);

}

sort(crossp, crossp + tot);

tot = unique(crossp, crossp + tot) - crossp;

double ans = 0.0;

for (int i = 0;i < tot - 1;++i)

{

Point tmp = 0.5\*(crossp[i] + crossp[i + 1]);

if (isPointInPolygon(tmp, polygon))

ans += getDis(crossp[i], crossp[i + 1]);

}

return ans;

}

**//直线ab划分多边形（注意①ab为有向直线，即只记录向量ab左侧的多边形②用的时候要判断polygon.size()>=3，防止一个polygon内只存了一个或两个点的情况）**

Polygon dividePolygon(Polygon p, Point a, Point b)

{

int n = p.size();

Polygon newp;

for (int i = 0;i < n;i++)

{

Point c = p[i], d = p[(i + 1) % n];

if (dcmp(getCross(b, c, a)) > 0)//只记录向量ab左侧的多边形

newp.push\_back(c);

if (onLine(c, a, b))

newp.push\_back(c);

else if (onLine(d, a, b))

continue;

else if (lineIntersect(a, b, c, d))

{

Point tmp = getPoint(a, b, c, d);

if (OnSegment(tmp, c, d))

newp.push\_back(tmp);

}

}

return newp;

}

**1.5 坐标离散化**

int compress(int \*xx1,int \*xx2,int w) { //开始坐标，结束坐标

vector<int> v;

for(int i = 0; i < n; ++i) //将横线本身以及附近两横线存储

for(int d = -1; d <= 1; ++d) {

int nx1 = xx1[i] + d;

int nx2 = xx2[i] + d;

if(nx1 >= 1 && nx1 <= w) v.push\_back(nx1);

if(nx2 >= 1 && nx2 <= w) v.push\_back(nx2);

}

//去重

sort(v.begin(), v.end());

v.erase(unique(v.begin(),v.end()),v.end());

//unique()函数去重并返回多余元素存储的起始位置

//erase()函数区间删除左开右闭的区间

//离散化后的坐标

for(int i = 0; i < n; ++i) {

xx1[i] = find(v.begin(), v.end(), xx1[i]) - v.begin();

xx2[i] = find(v.begin(), v.end(), xx2[i]) - v.begin();

//通过find()函数将坐标转化为了下标

}

return v.size();

}

**1.6 凸包**

**1.6.1 Andrew算法 扫描O(n)+排序O(nlogn)=O(nlogn)**

struct Point {

double x, y;

Point(double x = 0, double y = 0):x(x),y(y){}

};

typedef Point Vector;

Vector operator - (Point A, Point B){

return Vector(A.x-B.x, A.y-B.y);

}

bool operator < (const Point& a, const Point& b){

if(a.x == b.x)

return a.y < b.y;

return a.x < b.x;

}

double Cross(Vector v0, Vector v1) {

return v0.x\*v1.y - v1.x\*v0.y;

}

**//计算凸包，输入点数组为 p，个数为 n， 输出点数组为 ch。函数返回凸包顶点数**

**//如果不希望凸包的边上有输入点，则把两个 <= 改为 <**

**//在精度要求高时建议用dcmp比较**

**//输入不能有重复点，函数执行完后输入点的顺序被破坏**

int ConvexHull(Point\* p, int n, Point\* ch) {

sort(p, p+n);

int m = 0;

for(int i = 0; i < n; ++i) {

while(m > 1 && Cross(ch[m-1] - ch[m-2], p[i] - ch[m-2]) <= 0) m--;

ch[m++] = p[i];

}

int k = m;

for(int i = n-2; i>= 0; --i) {

while(m > k && Cross(ch[m-1] - ch[m-2], p[i] - ch[m-2]) <= 0) m--;

ch[m++] = p[i];

}

if(n > 1) --m;

return m;

}

**1.7 半平面交**

const double eps = 1e-6;

struct Point{

double x, y;

Point(double x = 0, double y = 0):x(x),y(y){}

};

typedef Point Vector;

Vector operator + (Vector A, Vector B){

return Vector(A.x+B.x, A.y+B.y);

}

Vector operator - (Point A, Point B){

return Vector(A.x-B.x, A.y-B.y);

}

Vector operator \* (Vector A, double p){

return Vector(A.x\*p, A.y\*p);

}

int sgn(double x){

if(fabs(x) < eps)

return 0;

if(x < 0)

return -1;

return 1;

}

double Dot(Vector A, Vector B){

return A.x\*B.x + A.y\*B.y;

}

double Cross(Vector A, Vector B){

return A.x\*B.y-A.y\*B.x;

}

double Length(Vector A){

return sqrt(Dot(A, A));

}

Vector Normal(Vector A){//向量A左转90°的单位法向量

double L = Length(A);

return Vector(-A.y/L, A.x/L);

}

**//有向直线的实现**

struct Line{

Point p;//直线上任意一点

Vector v;//方向向量，它的左边就是对应的半平面

double ang;//极角，即从x轴正半轴旋转到向量v所需要的角（弧度）

Line(){}

Line(Point p, Vector v) : p(p), v(v){

ang = atan2(v.y, v.x);

}

bool operator < (const Line& L) const {//排序用的比较运算符

return ang < L.ang;

}

};

//点p在有向直线L的左侧

bool OnLeft(Line L, Point p){

return Cross(L.v, p - L.p) > 0;

}

//两直线交点。假定交点唯一存在

Point GetIntersection(Line a, Line b){

Vector u = a.p - b.p;

double t = Cross(b.v, u)/Cross(a.v, b.v);

return a.p + a.v\*t;

}

**//半平面交的主过程**

int HalfplaneIntersection(Line\* L, int n, Point\* poly){

sort(L, L + n);//按照极角排序

int fst = 0, lst = 0;//双端队列的第一个元素和最后一个元素

Point \*P = new Point[n];//p[i] 为 q[i]与q[i + 1]的交点

Line \*q = new Line[n];//双端队列

q[fst = lst = 0] = L[0];//初始化为只有一个半平面L[0]

for(int i = 1; i < n; ++i){

while(fst < lst && !OnLeft(L[i], P[lst - 1])) --lst;

while(fst < lst && !OnLeft(L[i], P[fst])) ++fst;

q[++lst] = L[i];

if(sgn(Cross(q[lst].v, q[lst - 1].v)) == 0){

//两向量平行且同向，取内侧一个

--lst;

if(OnLeft(q[lst], L[i].p)) q[lst] = L[i];

}

if(fst < lst)

P[lst - 1] = GetIntersection(q[lst - 1], q[lst]);

}

while(fst < lst && !OnLeft(q[fst], P[lst - 1])) --lst;

//删除无用平面

if(lst - fst <= 1) return 0;//空集

P[lst] = GetIntersection(q[lst], q[fst]);//计算首尾两个半平面的交点

//从deque复制到输出中

int m = 0;

for(int i = fst; i <= lst; ++i) poly[m++] = P[i];

return m;

}

**1.8 圆**

**//圆的定义**

struct Circle

{

Point \_o;

double \_r;

Circle(double x = 0.0, double y = 0.0, double r = 0.0) :\_o(x, y), \_r(r) {}

Circle(const Point& o, double r) :\_o(o), \_r(r) {}

Circle getAnti(const Point& point)

{

Circle antic;

double dis = getDis(point, \_o);

double tmp = R\*R / (dis\*dis - \_r\*\_r);

antic.\_r = tmp\*\_r;

antic.\_o.\_x = point.\_x + tmp\*(\_o.\_x - point.\_x);

antic.\_o.\_y = point.\_y + tmp\*(\_o.\_y - point.\_y);

return antic;

}

friend istream& operator >> (istream& in, Circle& circle)

{

in >> circle.\_o >> circle.\_r;

return in;

}

friend ostream& operator<<(ostream& out, const Circle& circle)

{

out << circle.\_o << ' ' << circle.\_r;

return out;

}

};

**//点是否在圆内**

bool inCircle(Point p, Circle a)

{

return dcmp(getDis(p, a.\_o) - a.\_r) <= 0;

}

**//圆含于多边形**

bool circle\_inPolygon(Circle c, Polygon p)

{

if (!isPointInPolygon(c.\_o, p))

return false;

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

{

Point a = p[i], b = p[(i + 1) % p.size()];

if (dcmp(DistanceToSegment(c.\_o, a, b) - c.\_r) < 0)

return false;

}

return true;

}

**//多边形含于圆**

bool polygon\_inCircle(Circle c, Polygon p)

{

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

if (!inCircle(p[i], c))

return false;

return true;

}

**//圆含于多边形，多边形含于圆，圆和多边形相交，相切没算进去**

bool polygon\_intersectCircle(Circle c, Polygon p)

{

if (polygon\_inCircle(c, p) || circle\_inPolygon(c, p))

return true;

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

{

Point a = p[i], b = p[(i + 1) % p.size()];

if (dcmp(DistanceToSegment(c.\_o, a, b) - c.\_r) < 0)**//算相切要<=0**

return true;

}

return false;

}

**//三点确定外接圆圆心坐标**

struct Point {

double x,y;

Point(double x = 0, double y = 0):x(x),y(y){}

};

Point Excenter(Point a, Point b, Point c){

double a1 = b.x - a.x;

double b1 = b.y - a.y;

double c1 = (a1\*a1 + b1\*b1)/2;

double a2 = c.x - a.x;

double b2 = c.y - a.y;

double c2 = (a2\*a2 + b2\*b2)/2;

double d = a1\*b2 - a2\*b1;

return Point(a.x + (c1\*b2 - c2\*b1)/d, a.y + (a1\*c2 - a2\*c1)/d);

}

**2.2扩欧**

ll exgcd(ll a,ll b,ll &x,ll &y)

{

if(b==0)

{

x=1;

y=0;

return a;

}

ll g=exgcd(b,a%b,x,y),t;

t=x;

x=y;

y=t-(a/b)\*y;

return g;

}

**2.6快速傅里叶变换（FFT）**

#include <cmath>

#include <cstdio>

#include <iostream>

#define il inline

using namespace std ;

int N, M, K ;

const int MAXN = 3000100 ;

const double Pi = acos(-1.0) ;

int i, j, k, l, Lim = 1, L, R[MAXN] ;

struct node{

double x, y ;

node (double xx = 0, double yy = 0){

x = xx, y = yy ;

}

}A[MAXN], B[MAXN] ;

node operator \* (node J, node Q){

return node(J.x \* Q.x - J.y \* Q.y , J.x \* Q.y + J.y \* Q.x);

}

node operator + (node J, node Q){

return node(J.x + Q.x , J.y + Q.y);

}

node operator - (node J, node Q){

return node(J.x - Q.x , J.y - Q.y );

}

il int qr(){

int k = 0, f = 1 ;

char c = getchar() ;

while(!isdigit(c)){if(c == '-') f = -1 ;c = getchar() ;}

while(isdigit(c)) k = (k << 1) + (k << 3) + c - 48 ,c = getchar() ;

return k \* f ;

}

void FFT(node \*J, int flag){

for(i = 0; i < Lim; i ++)

if(i < R[i]) swap(J[i], J[R[i]]) ;

for(j = 1; j < Lim; j <<= 1){

node T(cos(Pi / j), flag \* sin(Pi / j)) ;

for(k = 0; k < Lim; k += (j << 1) ){

node t(1, 0) ;

for(l = 0 ; l < j; l ++, t = t \* T){

node Nx = J[k + l], Ny = t \* J[k + j + l] ;

J[k + l] = Nx + Ny ;

J[k + j + l] = Nx - Ny ;

}

}

}

}

int main(){

N = qr(), M = qr() ;

for(i = 0; i <= N; i ++) A[i].x = qr() ;

for(i = 0; i <= M; i ++) B[i].x = qr() ;

while(Lim <= N + M) Lim <<= 1, L ++ ;

for(i = 0; i < Lim; i ++ ) R[i] = (R[i >> 1] >> 1) | ((i & 1) << (L - 1)) ;

FFT(A, 1), FFT(B, 1) ;

for(i = 0; i <= Lim; i ++) A[i] = A[i] \* B[i] ;

FFT(A, -1) ;

for(i = 0; i <= N + M; i ++)

printf("%d ", (int)(A[i].x / Lim + 0.5)) ;//我们推过的公式里面有一个1/n这一项，最后输出的时候添上即可

return 0 ;

}

**2.9佩兰方程**

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

ll a[20000];

bool pell\_minimum\_solution(ll n,ll &x0,ll &y0){//求PELL方程最小整数解

ll m=(ll)sqrt((double)n);//M是N的平方根向下取整

if(m\*m==n)return false;//当n是完全平方数则佩尔方程无解（不讨论正负一，零）

//下面是把N用连分数形式存，B,C,TMP（即AI）的递推见解释以22为例参考

int i=0; //连分数的数位

a[i++]=m; //A0位整数部分m=4

ll b=m,c=1; //B=4即整数部分位,C=1即求RN时的分母

double sq=sqrt(n);//SQ是N的高精度根,相当于r0

double tmp;//tp在下面的循环就是rn

do{

c=(n-b\*b)/c;

tmp=(sq+b)/c;

a[i++]=(ll)(floor(tmp));

b=a[i-1]\*c-b;

//printf("%lld %lld %lld\n",a[i-1],b,c);

}while(a[i-1]!=2\*a[0]);//当有一位等于整数两倍就结束

//下面就是要把连分数形式化成分子分母的形式，求PQ两个值

ll p=1,q=0;

for(int j=i-2;j>=0;j--){

ll t=p;

p=q+p\*a[j];

q=t;

//printf("a[%d]=%lld %lld %lld\n",j,a[j],p,q);

}

if((i-1)%2==0){x0=p;y0=q;}//如果I是奇数，X0与Y0都是0

else{x0=2\*p\*p+1;y0=2\*p\*q;}//如果I是偶数，X0是两倍P方+1，y0是两倍PQ

return true;

}

int main(){

ll n,x,y;

while(~scanf("%lld",&n)){

if(pell\_minimum\_solution(n,x,y)){//输入N求X,Y的值，有解就输出

printf("x=%lld y=%lld\t",x,y);//x,y

printf("%lld^2-%lld\*%lld^2=1\t",x,n,y);//x^2-n\*y^2=1

printf("%lld-%lld=1\n",x\*x,n\*y\*y);//(x^2)-(n\*y^2)=1

}

}

return 0;

}

**2.10 组合数及逆元**

int C(int x,int y){

if(x<y||y<0)return 0;

return 1LL\*fac[x]\*rev[y]%mod\*rev[x-y]%mod;

}

//阶乘逆元

inline void init() {

fac[0]=fac[1]=revfac[0]=revfac[1]=1;

for(int i=2; i<MAXN; i++)fac[i]=1LL\*fac[i-1]\*i%MOD;

for(int i=2; i<MAXN; i++)revfac[i]=1LL\*(MOD-MOD/i)\*revfac[MOD%i]%MOD;

for(int i=2; i<MAXN; i++)revfac[i]=1LL\*revfac[i-1]\*revfac[i]%MOD;

}

//扩欧逆元

void exgcd(ll a,ll b,ll& d,ll& x,ll& y)

{

if(!b) { d = a; x = 1; y = 0; }

else{ exgcd(b, a%b, d, y, x); y -= x\*(a/b); }

}

ll inv(ll a, ll p)

{

ll d, x, y;

exgcd(a, p, d, x, y);

return d == 1 ? (x+p)%p : -1;

}

**3.1 Trie**

#include<queue>

#include<set>

#include<cstdio>

#include <iostream>

#include<algorithm>

#include<cstring>

#include<cmath>

using namespace std;

/\*

trie字典树

\*/

struct tnode{

int sum;//用来判断是否是终点的

tnode\* next[26];

tnode(){

for(int i =0;i<26;i++)

next[i]=NULL;

sum=0;

}

};

tnode \*root;

tnode\* newnode(){

tnode \*p = new tnode;

for(int i =0;i<26;i++)

p->next[i]=NULL;

p->sum=0;

return p;

}

//插入函数

void Insert(char \*s)

{

tnode \*p = root;

for(int i = 0 ; s[i] ; i++)

{

int x = s[i] - 'a';

if(p->next[x]==NULL)

{

tnode \*nn=newnode();

for(int j=0;j<26;j++)

nn->next[j] = NULL;

nn->sum = 0;

p->next[x]=nn;

}

p = p->next[x];

}

p->sum++;//这个单词终止啦

}

//匹配函数

bool Compare(char \*ch)

{

tnode \*p = root;

int len = strlen(ch);

for(int i = 0; i < len; i++)

{

int x = ch[i] - 'a';

p = p->next[x];

if(p==NULL)

return false;

if(i==len-1 && p->sum>0 ){

return true;

}

}

return false;

}

void DELETE(tnode \* &top){

if(top==NULL)

return;

for(int i =0;i<26;i++)

DELETE(top->next[i]);

delete top;

}

int main()

{

int n,m;

cin>>n;

char s[20];

root = newnode();

for(int i =0;i<n;i++){

scanf("%s",s);

Insert(s);

}

cin>>m;

for(int i =0;i<m;i++){

scanf("%s",s);

if(Compare(s))

cout<<"YES"<<endl;

else

cout<<"NO"<<endl;

}

DELETE(root);//看见指针就要想到释放,然而这东西会花时间，所以网上很多人写ACM题就不delete了，我很看不惯这一点。

return 0;

}

**3.2 KMP**

void getnx()

{

nx[1]=0;

for(int i=2,j=1;i<=n;)

{

nx[i]=j;

while(j&&s1[j]!=s1[i])j=nx[j];

j++,i++;

}

}

void kmp()

{

for(int i=1,j=1;i<=m;)

{

while(j&&s1[j]!=s2[i])j=nx[j];

if(j==n)

{

// 此时找到了一个能够匹配的位置

j=nx[j];

}

else j++,i++;

}

}

**3.4后缀数组**

**3.4.1 倍增法 O(nlogn)**

const int MAX=1e6+5;

int n,m;

int tax[MAX],rak[MAX],tp[MAX],sa[MAX],height[MAX];

char s[MAX];

void sort(int a[],int b[]){

for(int i=0;i<=m;i++)tax[i]=0;

for(int i=1;i<=n;i++)tax[a[i]]++;

for(int i=1;i<=m;i++)tax[i]+=tax[i-1];

for(int i=n;i>=1;i--)sa[tax[a[b[i]]]--]=b[i];

}

bool comp(int r[],int a,int b,int k){

return r[a]==r[b]&&r[a+k]==r[b+k];

}

void get\_sa(int a[],int b[]){

for(int i=1;i<=n;i++)

m=max(m,a[i]=s[i]-'0'),b[i]=i;

sort(a,b);

for(int p=0,j=1;p<n;j<<=1,m=p){

p=0;

for(int i=1;i<=j;i++)b[++p]=n-j+i;

for(int i=1;i<=n;i++)if(sa[i]>j)b[++p]=sa[i]-j;

sort(a,b);

int \*t=a;a=b;b=t;

a[sa[1]]=p=1;

for(int i=2;i<=n;i++)

a[sa[i]]=comp(b,sa[i],sa[i-1],j)?p:++p;

}

}

void getheight(int rak[])

{

for(int i=1,j=0;i<=n;i++){

if(j)j--;

while(s[i+j]==s[sa[rak[i]-1]+j])j++;

height[rak[i]]=j;

}

}

int main(){

scanf("%s",s+1);

n=strlen(s+1);

get\_sa(rak,tp);

getheight(rak);

for(int i=1;i<=n;i++)printf("%d ",height[i]);

}

**3.4.2 DC3法 O(n)**

#define F(x) ((x) / 3 + ((x) % 3 == 1 ? 0 : tb))

#define G(x) ((x) < tb ? (x) \* 3 + 1 : ((x) - tb) \* 3 + 2)

using namespace std;

const int N = 1000005\*3;

int wa[N], wb[N], ws[N], wv[N], sa[N];

int rnk[N], height[N], r[N];

int c0(int \*r, int a, int b) {

return r[a] == r[b] && r[a + 1] == r[b + 1] && r[a + 2] == r[b + 2];

}

int c12(int k, int \*r, int a, int b) {

if (k == 2)

return r[a] < r[b] || r[a] == r[b] && c12(1, r, a + 1, b + 1);

return r[a] < r[b] || r[a] == r[b] && wv[a + 1] < wv[b + 1];

}

void Rsort(int \*r, int \*a, int \*b, int n, int m) {

for (int i = 0; i < n; i++) wv[i] = r[a[i]];

for (int i = 0; i < m; i++) ws[i] = 0;

for (int i = 0; i < n; i++) ws[wv[i]]++;

for (int i = 1; i < m; i++) ws[i] += ws[i - 1];

for (int i = n - 1; i >= 0; i--) b[--ws[wv[i]]] = a[i];

}

void dc3(int \*r, int \*sa, int n, int m) {

int i, j, \*rn = r + n, \*san = sa + n, ta = 0, tb = (n + 1) / 3, tbc = 0, p;

r[n] = r[n + 1] = 0;

for (i = 0; i < n; i++) if (i % 3 != 0) wa[tbc++] = i;

Rsort(r + 2, wa, wb, tbc, m);

Rsort(r + 1, wb, wa, tbc, m);

Rsort(r, wa, wb, tbc, m);

for (p = 1, rn[F(wb[0])] = 0, i = 1; i < tbc; i++)

rn[F(wb[i])] = c0(r, wb[i - 1], wb[i]) ? p - 1 : p++;

if (p < tbc) dc3(rn, san, tbc, p);

else for (i = 0; i < tbc; i++) san[rn[i]] = i;

for (i = 0; i < tbc; i++) if (san[i] < tb) wb[ta++] = san[i] \* 3;

if (n % 3 == 1) wb[ta++] = n - 1;

Rsort(r, wb, wa, ta, m);

for (i = 0; i < tbc; i++) wv[wb[i] = G(san[i])] = i;

for (i = 0, j = 0, p = 0; i < ta && j < tbc; p++)

sa[p] = c12(wb[j] % 3, r, wa[i], wb[j]) ? wa[i++] : wb[j++];

for (; i < ta; p++) sa[p] = wa[i++];

for (; j < tbc; p++) sa[p] = wb[j++];

}

void calheight(int \*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; height[rnk[i++]] = k)

for (k ? k-- : 0, j = sa[rnk[i] - 1]; r[i + k] == r[j + k]; k++);

}

/\*

sa[i]=j ，表示第 i 名的后缀是从 j 开始的（注意存的是下标）

rank[i]=j ，从 i 开始的后缀是第 j 名的（和 sa 互逆，是排名（值））

tp[i] (b[i]) =j，第二关键字为 i 的后缀是从 j 开始的 （相当于是第二关键字的 sa 数组，存的是下标）

tax[i]=j，表示第一关键字为 i 的数，有 j 个（基数排序时用的桶，是值）

height[i]：lcp(sa[i],sa[i−1])，即排名为i的后缀与排名为i−1的后缀的最长公共前缀

H[i]：height[rak[i]]，即i号后缀与它前一名的后缀的最长公共前缀

**性质：H[i]⩾H[i−1]−1**

**应用：**

**两个后缀的最大公共前缀：**lcp(x,y)=min(heigh[x..y])， 用rmq维护，O(1)查询

**可重叠最长重复子串：**Height数组里的最大值

**不可重叠最长重复子串：**首先二分答案x，对height数组进行分组，保证每一组的minheight都>=x。依次枚举每一组，记录下最大和最小长度，多sa[mx]−sa[mi]>=x那么可以更新答案

**本质不同的子串的数量：**枚举每一个后缀，第ii个后缀对答案的贡献为len−sa[i]+1−height[i]

\*/

**3.5后缀自动机**

#include<iostream>

#include<cstdio>

#include<cstring>

#include<algorithm>

#include<queue>

#include<cmath>

#include<map>

#define fi first

#define se second

#define pii pair<int,int>

#define inf 1000000005

#define pb push\_back

#define maxn 100005

using namespace std;

typedef long long int ll;

const int N=3e6+10;

int now=1,cnt=1,cntedge=0;

ll ans=0;

char s[3000005];

struct Suffix\_Automaton

{

int mp[N][30];

int fa[N];

int p,np,q,nq;

int mx[N],siz[N],v[N],nxt[N],head[N];

//Suffix\_Automaton(){np=nq=1;}

void insert(int x)

{

p=now,np=++cnt;

mx[np]=mx[now]+1;now=np;siz[np]=1;

while(!mp[p][x]&&p)mp[p][x]=np,p=fa[p];

if(!p)fa[np]=1;

else

{

q=mp[p][x];

if(mx[q]==mx[p]+1)fa[np]=q;

else

{

nq=++cnt;

mx[nq]=mx[p]+1;

memcpy(mp[nq],mp[q],sizeof(mp[q]));

fa[nq]=fa[q];fa[q]=fa[np]=nq;

while(mp[p][x]==q)mp[p][x]=nq,p=fa[p];

}

}

}

void addedge(int u,int t)

{

cntedge++;

v[cntedge]=t;

nxt[cntedge]=head[u];

head[u]=cntedge;

}

void build()

{

for(int i=2;i<=nq;i++)addedge(fa[i],i);

}

void dfs(int u)

{

for(int i=head[u];i;i=nxt[i])dfs(v[i]),siz[u]+=siz[v[i]];

if(siz[u]!=1)ans=max(ans,(ll)siz[u]\*mx[u]);

}

}sam;

int main()

{

scanf("%s",s);

for(int i=0;s[i];i++)sam.insert(s[i]-'a'+1);

sam.build();sam.dfs(1);

printf("%lld\n",ans);

return 0;

}

**3.7马拉车**

**3.7.1 不分奇偶**

int Manacher(){

Len[0] = 0;

int sum = 0;

mx = 0;

for(int i=1;i<len;i++){

if(i < mx) Len[i] = Min(mx - i, Len[2 \* id - i]);

else Len[i] = 1;

while(str[i - Len[i]]== str[i + Len[i]]) Len[i]++;

if(Len[i] + i > mx){

mx = Len[i] + i;

id = i;

sum = Max(sum, Len[i]);

}

}

return (sum - 1);

}

**3.7.2 分奇偶**

ll manacher(){

int l=-1,r=-1,x;

ll ans=0;

for(int i=0;i<len;i++)

{

if (i>r) x=1;

else x=min(odd[l+r-i],r-i);

while (i-x>=0 && i+x<len && S[i-x]==S[i+x]) x++;

odd[i]=x;

ans+=x;

if (i+x-1>r) {r=i+x-1;l=i-x+1;}

}

l=r=-1;

for(int i=0;i<len;i++)

{

if(i>r) x=0;

else x=min(eve[l+r-i+1],r-i+1);

while (i-x-1>=0 && i+x<len && S[i-x-1]==S[i+x]) x++;

eve[i]=x;

ans+=x;

if (i+x>=r) {l=i-x;r=i+x-1;}

}

return ans;

}

**3.8 AC自动机**

#include<queue>

#include<set>

#include<cstdio>

#include <iostream>

#include<algorithm>

#include<cstring>

#include<cmath>

using namespace std;

/\*

ac自动机

\*/

struct acnode{

int sum;

acnode\* next[26];

acnode\* fail;

acnode(){

for(int i =0;i<26;i++)

next[i]=NULL;

fail= NULL;

sum=0;

}

};

acnode \*root;

int cnt;

acnode\* newnode(){

acnode \*p = new acnode;

for(int i =0;i<26;i++)

p->next[i]=NULL;

p->fail = NULL;

p->sum=0;

return p;

}

//插入函数

void Insert(char \*s)

{

acnode \*p = root;

for(int i = 0; s[i]; i++)

{

int x = s[i] - 'a';

if(p->next[x]==NULL)

{

acnode \*nn=newnode();

for(int j=0;j<26;j++)

nn->next[j] = NULL;

nn->sum = 0;

nn->fail = NULL;

p->next[x]=nn;

}

p = p->next[x];

}

p->sum++;

}

//获取fail指针，在插入结束之后使用

void getfail(){

queue<acnode\*> q;

for(int i = 0 ; i < 26 ; i ++ )

{

if(root->next[i]!=NULL){

root->next[i]->fail = root;

q.push(root->next[i]);

}

}

while(!q.empty()){

acnode\* tem = q.front();

q.pop();

for(int i = 0;i<26;i++){

if(tem->next[i]!=NULL)

{

acnode \*p;

if(tem == root){

tem->next[i]->fail = root;

}

else

{

p = tem->fail;

while(p!=NULL){

if(p->next[i]!=NULL){

tem->next[i]->fail = p->next[i];

break;

}

p=p->fail;

}

if(p==NULL)

tem->next[i]->fail = root;

}

q.push(tem->next[i]);

}

}

}

}

//匹配函数

void ac\_automation(char \*ch)

{

acnode \*p = root;

int len = strlen(ch);

for(int i = 0; i < len; i++)

{

int x = ch[i] - 'a';

while(p->next[x]==NULL && p != root)//没匹配到，那么就找fail指针。

p = p->fail;

p = p->next[x];

if(!p)

p = root;

acnode \*temp = p;

while(temp != root)

{

if(temp->sum >= 0)

/\*

在这里已经匹配成功了，执行想执行的操作即可，怎么改看题目需求+

\*/

{

cnt += temp->sum;

temp->sum = -1;

}

else break;

temp = temp->fail;

}

}

}

int main()

{

cnt = 0;

int n;

cin>>n;

char c[101];

root = newnode();

for(int i = 0 ;i < n;i++){

scanf("%s",c);

Insert(c);

}

getfail();

int m ;

cin>> m;

for(int i = 0;i<m;i++){

scanf("%s",c);

ac\_automation(c);

}

cout<<cnt<<endl;

return 0;

}

**3.9 字典树**

#include<queue>

#include<set>

#include<cstdio>

#include <iostream>

#include<algorithm>

#include<cstring>

#include<cmath>

using namespace std;

/\*

trie字典树

\*/

struct tnode{

int sum;//用来判断是否是终点的

tnode\* next[26];

tnode(){

for(int i =0;i<26;i++)

next[i]=NULL;

sum=0;

}

};

tnode \*root;

tnode\* newnode(){

tnode \*p = new tnode;

for(int i =0;i<26;i++)

p->next[i]=NULL;

p->sum=0;

return p;

}

//插入函数

void Insert(char \*s)

{

tnode \*p = root;

for(int i = 0 ; s[i] ; i++)

{

int x = s[i] - 'a';

if(p->next[x]==NULL)

{

tnode \*nn=newnode();

for(int j=0;j<26;j++)

nn->next[j] = NULL;

nn->sum = 0;

p->next[x]=nn;

}

p = p->next[x];

}

p->sum++;//这个单词终止啦

}

//匹配函数

bool Compare(char \*ch)

{

tnode \*p = root;

int len = strlen(ch);

for(int i = 0; i < len; i++)

{

int x = ch[i] - 'a';

p = p->next[x];

if(p==NULL)

return false;

if(i==len-1 && p->sum>0 ){

return true;

}

}

return false;

}

void DELETE(tnode \* &top){

if(top==NULL)

return;

for(int i =0;i<26;i++)

DELETE(top->next[i]);

delete top;

}

int main()

{

int n,m;

cin>>n;

char s[20];

root = newnode();

for(int i =0;i<n;i++){

scanf("%s",s);

Insert(s);

}

cin>>m;

for(int i =0;i<m;i++){

scanf("%s",s);

if(Compare(s))

cout<<"YES"<<endl;

else

cout<<"NO"<<endl;

}

DELETE(root);

return 0;

}

**4.1二分图**

**4.1.1二分图最大匹配**

const int N=1005;

inline int read(){

char c=getchar();int x=0,f=1;

while(c<'0'||c>'9'){if(c=='-')f=-1;c=getchar();}

while(c>='0'&&c<='9'){x=x\*10+c-'0';c=getchar();}

return x\*f;

}

int n,m,s,u,v;

struct edge{

int v,ne;

}e[N\*N<<1];

int h[N],cnt=0;

inline void ins(int u,int v){

cnt++;

e[cnt].v=v;e[cnt].ne=h[u];h[u]=cnt;

}

int vis[N],le[N];

bool find(int u){

for(int i=h[u];i;i=e[i].ne){

int v=e[i].v;

if(!vis[v]){

vis[v]=1;

if(!le[v]||find(le[v])){

le[v]=u;

return true;

}

}

}

return false;

}

int ans=0;

void hungary(){

for(int i=1;i<=n;i++){

memset(vis,0,sizeof(vis));

if(find(i)) ans++;

}

}

int main(){

n=read();m=read();int t=read();

for(int i=1;i<=t;i++){u=read();v=read();if(v>m)continue;ins(u,v);}

ans=0;

hungary();

printf("%d\n",ans);

}

**4.1.2 二分图染色**

bool color(int u,int c){

col[u]=c;

for(int i=h[u];i;i=e[i].ne){

int v=e[i].v;

if(col[u]==col[v]) return false;

if(!col[v]&&!color(v,3-c)) return false;

}

return true;

}

**4.2最大团**

#include <bits/stdc++.h>

using namespace std;

const int maxn = 55;

bool mp[maxn][maxn];

int best, n, num[maxn];

bool dfs(int \*adj, int total, int cnt)

{

int t[maxn], k;

if(total == 0)

{

if(cnt > best)

{

best = cnt;

return true; //剪枝4

}

return false;

}

for(int i = 0; i < total; ++i)

{

if(cnt+total-i <= best) return false; //剪枝1

if(cnt+num[adj[i]] <= best) return false; //剪枝3

k = 0;

for(int j = i+1; j < total; ++j)

if(mp[adj[i]][adj[j]]) t[k++] = adj[j];

//扫描与u相连的顶点中与当前要选中的adj[i]相连的顶点adj[j]并存储到数组t[]中，数量为k

if(dfs(t, k, cnt+1)) return true;

}

return false;

}

int MaximumClique()

{

int adj[maxn], k;

best = 0;

for(int i = n; i >= 1; --i)

{

k = 0;

for(int j = i+1; j <= n; ++j)

if(mp[i][j]) adj[k++] = j;

//得到当前点i的所有相邻点存入adj

dfs(adj, k, 1); //每次dfs相当于必选当前i点看是否能更新best

num[i] = best;

}

return best;

}

int main()

{

while(cin >> n && n)

{

for(int i = 1; i <= n; ++i)

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

{

int x; cin >> x;

mp[i][j] = x;

}

cout << MaximumClique() << endl;

}

return 0;

}

**4.3 网络流**

**4.3.1 Dinic**

#define inf 200000000

#define maxm 100005

#define maxn 10005

#define md 1000000007

using namespace std;

typedef long long int ll;

int n,m,s,t;

int head[maxn],cur[maxn];

struct node

{

int to,next,w;

}edge[2\*maxm];

int cnt=0;

void addedge(int x,int y,int z)

{

edge[cnt].to=y;

edge[cnt].w=z;

edge[cnt].next=head[x];

head[x]=cnt;

cnt++;

}

int dep[maxn];

bool bfs()

{

queue<int>q;while(!q.empty())q.pop();

memset(dep,0,sizeof(dep));

dep[s]=1;

q.push(s);

while(!q.empty())

{

int u=q.front();q.pop();

for(int i=head[u];i!=-1;i=edge[i].next)

{

int v=edge[i].to;

if(!dep[v]&&edge[i].w>0)

{

dep[v]=dep[u]+1;

q.push(v);

}

}

}

if(dep[t])return 1;

return 0;

}

int dfs(int u,int dist)

{

if(u==t)return dist;

//int flow=0;

for(int& i=cur[u];i!=-1;i=edge[i].next)

{

int v=edge[i].to;

if(dep[v]==dep[u]+1&&edge[i].w>0)

{

int di=dfs(v,min(dist,edge[i].w));

if(di>0)

{

edge[i].w-=di;

edge[i^1].w+=di;

return di;

}

}

}

return 0;

}

int dinic()

{

int ans=0;

while(bfs())

{

for(int i=1;i<=n;i++)cur[i]=head[i];

while(int d=dfs(s,inf))ans+=d;

}

return ans;

}

int main()

{

memset(head,-1,sizeof(head));

scanf("%d%d%d%d",&n,&m,&s,&t);

for(int i=1;i<=m;i++)

{

int x,y,z;

scanf("%d%d%d",&x,&y,&z);

addedge(x,y,z);

addedge(y,x,0);

}

printf("%d\n",dinic());

return 0;

}

**4.3.2费用流**

#define inf 200000000

#define maxm 100005

#define maxn 10005

#define md 1000000007

using namespace std;

typedef long long int ll;

int n,m,s,t;

int head[maxn],cur[maxn];

struct node

{

int to,next,w,f;

}edge[2\*maxm];

int cnt=0;

void addedge(int x,int y,int z,int w)

{

edge[cnt].to=y;

edge[cnt].w=z;

edge[cnt].f=w;

edge[cnt].next=head[x];

head[x]=cnt;

cnt++;

}

int dis[maxn],vis[maxn],pre[maxn],last[maxn],flow[maxn];

int maxflow=0,mincost=0;

bool bfs()

{

queue<int>q;while(!q.empty())q.pop();

memset(dis,0x3f,sizeof(dis));

memset(flow,0x3f,sizeof(flow));

memset(vis,0,sizeof(vis));

dis[s]=0;

pre[t]=0;

q.push(s);

while(!q.empty())

{

int u=q.front();q.pop();vis[u]=0;

for(int i=head[u];i!=-1;i=edge[i].next)

{

int v=edge[i].to;

if(edge[i].w>0&&dis[v]>dis[u]+edge[i].f)

{

dis[v]=dis[u]+edge[i].f;

pre[v]=u;

last[v]=i;

flow[v]=min(edge[i].w,flow[u]);

if(!vis[v])

{

vis[v]=1;

q.push(v);

}

}

}

}

return pre[t];

}

void dinic()

{

while(bfs())

{

//for(int i=1;i<=n;i++)cur[i]=head[i];

maxflow+=flow[t];

mincost+=flow[t]\*dis[t];

int now=t;

while(now!=s)

{

edge[last[now]].w-=flow[t];

edge[last[now]^1].w+=flow[t];

now=pre[now];

}

}

}

int main()

{

memset(head,-1,sizeof(head));

scanf("%d%d%d%d",&n,&m,&s,&t);

for(int i=1;i<=m;i++)

{

int x,y,z,w;

scanf("%d%d%d%d",&x,&y,&z,&w);//z最大流量 w费用

addedge(x,y,z,w);

addedge(y,x,0,-w);

}

dinic();

printf("%d %d\n",maxflow,mincost);

return 0;

}

**4.4 强联通分量/Tarjan**

const int N=1e4+5,M=5e4+5;

typedef long long ll;

inline int read(){

char c=getchar();int x=0,f=1;

while(c<'0'||c>'9'){if(c=='-')f=-1;c=getchar();}

while(c>='0'&&c<='9'){x=x\*10+c-'0';c=getchar();}

return x\*f;

}

int n,m,u,v;

struct edge{

int v,ne;

}e[M];

int h[N],cnt=0;

inline void ins(int u,int v){

cnt++;

e[cnt].v=v;e[cnt].ne=h[u];h[u]=cnt;

}

int dfn[N],belong[N],low[N],dfc,scc,st[N],top;

int size[N];

void dfs(int u){

dfn[u]=low[u]=++dfc;

st[++top]=u;

for(int i=h[u];i;i=e[i].ne){

int v=e[i].v;

if(!dfn[v]){

dfs(v);

low[u]=min(low[u],low[v]);

}else if(!belong[v])

low[u]=min(low[u],dfn[v]);

}

if(low[u]==dfn[u]){

scc++;

while(true){

int x=st[top--];

belong[x]=scc;

size[scc]++;

if(x==u) break;

}

}

}

int outd[N],ind[N],ans;

void point(){

for(int u=1;u<=n;u++)

for(int i=h[u];i;i=e[i].ne){

int v=e[i].v;

if(belong[u]!=belong[v]) outd[belong[u]]++,ind[belong[v]]++;

}

}

int main(){

n=read();m=read();

for(int i=1;i<=m;i++){u=read();v=read();ins(u,v);}

for(int i=1;i<=n;i++) if(!dfn[i]) dfs(i);

point();

for(int i=1;i<=scc;i++){

if(outd[i]==0){

if(ans){ans=0;break;}

else ans=size[i];

}

}

printf("%d",ans);

}

**4.6割点**

const int N=1e5+5,M=1e5+5,INF=1e9+5;

inline int read(){

char c=getchar();int x=0,f=1;

while(c<'0'||c>'9'){if(c=='-')f=-1;c=getchar();}

while(c>='0'&&c<='9'){x=x\*10+c-'0';c=getchar();}

return x;

}

int n=0,m,u,v;

struct edge{

int v,ne;

}e[M<<1];

int h[N],cnt=0;

inline void ins(int u,int v){

cnt++;

e[cnt].v=v;e[cnt].ne=h[u];h[u]=cnt;

cnt++;

e[cnt].v=u;e[cnt].ne=h[v];h[v]=cnt;

}

int dfn[N],low[N],dfc=0,iscut[N];

void dfs(int u,int fa){

dfn[u]=low[u]=++dfc;

int child=0;

for(int i=h[u];i;i=e[i].ne){

int v=e[i].v;

if(!dfn[v]){

child++;

dfs(v,u);

low[u]=min(low[u],low[v]);

if(low[v]>=dfn[u]) iscut[u]=1;

}else if(v!=fa) low[u]=min(low[u],dfn[v]);

}

if(fa==0&&child==1) iscut[u]=0;

}

int main(){

n=read();m=read();

for(int i=1;i<=m;i++){u=read();v=read();ins(u,v);}

for(int i=1;i<=n;i++) if(!dfn[i]) dfs(i,0);

int ans=0;

for(int i=1;i<=n;i++) if(iscut[i]) ans++;

printf("%d\n",ans);

for(int i=1;i<=n;i++) if(iscut[i]) printf("%d ",i);

}

**4.8 dsu**

#include<iostream>

#include<cstring>

#include<algorithm>

#include<cstdio>

#include<cmath>

#define inf 2000000000

#define maxn 100005

using namespace std;

typedef long long int ll;

int n;

ll ans[maxn],sum=0;

int col[maxn],cnt[maxn],big[maxn],son[maxn];

int cntt=0,head[maxn],siz[maxn],sm=0;

struct node

{

int to,next;

}edge[2\*maxn];

void addedge(int x,int y)

{

cntt++;

edge[cntt].to=y;

edge[cntt].next=head[x];

head[x]=cntt;

}

void dfs(int now,int fa)

{

siz[now]=1;

for(int i=head[now];i;i=edge[i].next)

{

int v=edge[i].to;

if(v==fa)continue;

dfs(v,now);

siz[now]+=siz[v];

if(siz[v]>siz[son[now]])son[now]=v;

}

}

void add(int now,int fa,int val)

{

cnt[col[now]]+=val;

if(val>0&&cnt[col[now]]>=sm)

{

if(cnt[col[now]]>sm)sum=0,sm=cnt[col[now]];

sum+=col[now];

}

for(int i=head[now];i;i=edge[i].next)

{

int v=edge[i].to;

if(v!=fa&&!big[v])add(v,now,val);

}

}

void work(int now,int fa,int keep)

{

for(int i=head[now];i;i=edge[i].next)

{

int v=edge[i].to;

if(v==fa)continue;

if(v!=son[now])work(v,now,0);

}

if(son[now])work(son[now],now,1),big[son[now]]=1;

add(now,fa,1);

ans[now]=sum;

if(son[now])big[son[now]]=0;

if(keep==0)add(now,fa,-1),sm=sum=0;

}

int main()

{

scanf("%d",&n);

for(int i=1;i<=n;i++)scanf("%d",&col[i]);

for(int i=1;i<n;i++)

{

int x,y;

scanf("%d%d",&x,&y);

addedge(x,y);

addedge(y,x);

}

dfs(1,0);work(1,0,0);

for(int i=1;i<=n;i++)printf("%I64d ",ans[i]);

printf("\n");

return 0;

}

**4.9树链剖分**

#include<iostream>

#include<cstdio>

#include<cstdlib>

#include<cstring>

#include<cmath>

#include<algorithm>

#define LL long long

#define l(x) (x<<1)

#define r(x) ((x<<1)|1)

using namespace std;

struct EDG

{

int to,nex;

}es[1000010];

int n,m,r,p;

int first[100010];

int cnt;

void link(int a,int b)

{

es[++cnt].nex=first[a];

first[a]=cnt;

es[cnt].to=b;

}

LL a[1000100];

//树剖↓

LL real[1000010],sid[1000010],hson[1000010],fa[1000010],dep[1000010],size[1000010],top[1000010];

LL tot;

void dfs1(int x)

{

size[x]=1;

for(int i=first[x];i;i=es[i].nex)

{

int v=es[i].to;

if(v!=fa[x])

{

fa[v]=x;

dep[v]=dep[x]+1;

dfs1(v);

if(hson[x]==0||size[hson[x]]<size[v])

{

hson[x]=v;

}

size[x]+=size[v];

}

}

}

void dfs2(int x,int anc)

{

top[x]=anc;

sid[x]=++tot;

real[tot]=x;

if(hson[x]==0) return;

dfs2(hson[x],anc);

for(int i=first[x];i;i=es[i].nex)

{

int v=es[i].to;

if(v!=hson[x]&&v!=fa[x])

{

dfs2(v,v);

}

}

}

//树剖↑

//线段树 ↓

struct Tre

{

LL sum,tag;

}Tr[1000100];

void update(LL id)

{

Tr[id].sum=Tr[l(id)].sum+Tr[r(id)].sum;

}

void pushdown(LL l,LL r,LL id)

{

Tr[r(id)].tag+=Tr[id].tag;

Tr[l(id)].tag+=Tr[id].tag;

Tr[id].sum+=(r-l+1)\*Tr[id].tag;

Tr[id].tag=0;

}

void build(LL l,LL r,LL id)

{

if(l==r)

{

Tr[id].sum=a[real[l]];

return;

}

LL mid=(l+r)/2;

build(mid+1,r,r(id));

build(l,mid,l(id));

update(id);

}

void add(LL al,LL ar,LL x,LL l,LL r,LL id)

{

if(l>ar||r<al) return;

if(al==l&&ar==r) {Tr[id].tag+=x;return;}

pushdown(l,r,id);

LL mid=(r+l)/2;

if(mid>=al) add(al,min(mid,ar),x,l,mid,l(id));

if(mid<ar) add(max(mid+1,al),ar,x,mid+1,r,r(id));

pushdown(l,mid,l(id));

pushdown(mid+1,r,r(id));

update(id);

}

LL query(LL al,LL ar,LL l,LL r,LL id)

{

if(l>r) return 0LL;

if(al==l&&ar==r) {return Tr[id].tag\*(r-l+1)+Tr[id].sum;}

pushdown(l,r,id);

LL mid=(r+l)/2;

LL t=0;

if(mid>=al) t+=query(al,min(mid,ar),l,mid,l(id));

if(mid<ar) t+=query(max(mid+1,al),ar,mid+1,r,r(id));

return t;

}

//线段树 ↑

//树剖用线段树维护↓

void tr\_add(LL x,LL v)//x点及其子孙权值加v

{

add(sid[x],sid[x]+size[x]-1,v,1,n,1);

}

LL tr\_query(LL x)//x点及其子孙权值和

{

return query(sid[x],sid[x]+size[x]-1,1,n,1)%p;

}

LL chain\_add(LL x,LL y,LL v)//x到y的链上每个点加v

{

LL tx,ty;

tx=top[x];ty=top[y];

while(tx!=ty)

{

if(dep[tx]<dep[ty])

{

swap(x,y);

swap(tx,ty);

}

add(sid[tx],sid[x],v,1,n,1);

x=fa[tx];tx=top[x];

}

if(dep[x]<dep[y]) swap(x,y);

add(sid[y],sid[x],v,1,n,1);

}

LL chain\_query(LL x,LL y)//查询x到y的链上权值和

{

LL tx,ty,t=0;

tx=top[x];ty=top[y];

while(tx!=ty)

{

if(dep[tx]<dep[ty]) {swap(tx,ty);swap(x,y);}

(t+=query(sid[tx],sid[x],1,n,1))%=p;

x=fa[tx];tx=top[x];

}

if(dep[x]<dep[y]) swap(x,y);

(t+=query(sid[y],sid[x],1,n,1))%=p;

return t;

}

//树剖用线段树维护↑

int main()

{

scanf("%d%d%d%d",&n,&m,&r,&p);

for(int i=1;i<=n;i++) scanf("%lld",&a[i]);

for(int i=1;i<n;i++)

{

int x,y;

scanf("%d%d",&x,&y);

link(x,y);link(y,x);

}

dep[r]=1;

dfs1(r);

dfs2(r,r);

build(1,n,1);

for(int i=1;i<=m;i++)

{

int q;

scanf("%d",&q);

if(q==1)

{

LL x,y,z;

scanf("%lld%lld%lld",&x,&y,&z);

chain\_add(x,y,z);

}

if(q==2)

{

LL x,y;

scanf("%lld%lld",&x,&y);

cout<<chain\_query(x,y);

cout<<endl;

}

if(q==3)

{

LL x,z;

scanf("%lld%lld",&x,&z);

tr\_add(x,z);

}

if(q==4)

{

LL x;

scanf("%lld",&x);

cout<<tr\_query(x);

cout<<endl;

}

}

}

**4.10 Floyd最小环**

const int N=105,M=1e4+5,INF=1e8+5;//1E9+1E9+1E9溢出

inline int read(){

char c=getchar();int x=0,f=1;

while(c<'0'||c>'9'){if(c=='-')f=-1;c=getchar();}

while(c>='0'&&c<='9'){x=x\*10+c-'0';c=getchar();}

return x\*f;

}

int n,m,u,v,w,g[N][N];

int d[N][N],ans=INF;

void floyd(){

ans=INF;

for(int k=1;k<=n;k++){

for(int i=1;i<=k-1;i++)

for(int j=i+1;j<=k-1;j++)

ans=min(ans,g[i][k]+g[k][j]+d[i][j]);

for(int i=1;i<=n;i++)

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

d[i][j]=min(d[i][j],d[i][k]+d[k][j]);

}

}

int main(){

while(scanf("%d%d",&n,&m)!=EOF){

for(int i=1;i<=n;i++) for(int j=i+1;j<=n;j++) d[i][j]=d[j][i]=g[i][j]=g[j][i]=INF;

for(int i=1;i<=m;i++){

u=read();v=read();w=read();

d[u][v]=d[v][u]=g[u][v]=g[v][u]=w;

}

floyd();

if(ans==INF) puts("No solution.");

else printf("%d\n",ans);

}

}

**4.11 最小生成树**

const int N=5005,M=2e5+5,INF=1e9+5;

inline int read(){

char c=getchar();int x=0,f=1;

while(c<'0'||c>'9'){if(c=='-')f=-1;c=getchar();}

while(c>='0'&&c<='9'){x=x\*10+c-'0';c=getchar();}

return x\*f;

}

int n,m,u,v,w;

int cnt=0;

struct edge{

int u,v,w;

bool operator <(const edge &r)const{return w<r.w;}

}e[M];

int fa[N];

inline int find(int x){return x==fa[x]?x:fa[x]=find(fa[x]);}

int kruskal(){

int ans=0,cnt=0;

for(int i=1;i<=n;i++) fa[i]=i;

sort(e+1,e+1+m);

for(int i=1;i<=m;i++){

int u=e[i].u,v=e[i].v,w=e[i].w;

int f1=find(u),f2=find(v);

if(f1!=f2){

ans+=w;

fa[f1]=f2;

cnt++;

if(cnt==n-1) break;

}

}

return ans;

}

int main(){

n=read();m=read();

for(int i=1;i<=m;i++){

e[i].u=read();e[i].v=read();e[i].w=read();

}

int ans=kruskal();

printf("%d",ans);

}

**4.12 点分治**

**写法**

点分治有2种写法。

1. 对于某个重心u，统计以u为根的所有路径，然后计算出所有组合情况。递归子树时，首先删除全部在一颗子树的路径，然后再进入子树递归求解。这样可以保证路径全部合法且不重不漏。

2. 对于某个重心u，先进入子树v1，求解出u到子树v1所有节点的路径，然后进入子树v2，进入时先统计答案，然后再统计相关值。每次进入新的子树时，先统计答案，这样每次计算的路径一定是和之前统计过子树的相连而成的，没有不合法的答案，所以不用删除。最后，依次递归进子树，找出重心递归求解。

图1表示直接统计以重心为根的子树上的路径，递归进入子树之前，首先删除子树中的不合法路径。

图2表示按子树依次递归进入，进入后先统计答案，然后再统计相关值。统计答案是利用到了先前子树的信息。

一般来说，第二种写法常数更小。

**相关套路**

目前遇到的题型有

1. 路径和等于或小于等于k的点对（路径条数）。

2. 路径和为某个数的倍数。

3. 路径和为k且路径的边数最少。

4. 路径和mod M后为某个值。

5. 路径上经过不允许点的个数不超过某个值，且路径和最大。

如若使用方法1，一般开一个栈，保存路径上的距离等相关信息，排序后利用单调性或者二分找答案。

如若使用方法2，则一般处理这些问题时，都是开一个桶mess[i]，表示距离为i的相关信息。

#include<iostream>

#include<cstring>

#include<cstdio>

#include<algorithm>

#include<cmath>

#include<queue>

#include<vector>

#define inf 200000000

#define maxm 100005

#define maxn 10005

#define md 1000000007

using namespace std;

typedef long long int ll;

typedef double db;

int n,m,k;

int cnt=0,head[maxn],sum=0,root=0;

struct node{

int to,next,w;

}edge[maxn\*2];

int siz[maxn],dep[maxn],vis[maxn],ans[10000005],f[maxn],o[maxn];

void addedge(int x,int y,int z)

{

cnt++;

edge[cnt].to=y;

edge[cnt].w=z;

edge[cnt].next=head[x];

head[x]=cnt;

}

void getroot(int u,int fa)

{

siz[u]=1;f[u]=0;

for(int i=head[u];i;i=edge[i].next)

{

int v=edge[i].to;

if(v==fa||vis[v])continue;

getroot(v,u);

siz[u]+=siz[v];

f[u]=max(f[u],siz[v]);

}

f[u]=max(f[u],sum-siz[u]);

if(f[u]<f[root])root=u;

}

void getdeep(int u,int fa)

{

o[++cnt]=dep[u];

for(int i=head[u];i;i=edge[i].next)

{

int v=edge[i].to;

if(v==fa||vis[v])continue;

dep[v]=dep[u]+edge[i].w;

getdeep(v,u);

}

}

void calc(int u,int d,int add)

{

cnt=0;dep[u]=d;

getdeep(u,0);

for(int i=1;i<=cnt;i++)

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

ans[o[i]+o[j]]+=add;

}

void solve(int u)

{

calc(u,0,1);vis[u]=1;

for(int i=head[u];i;i=edge[i].next)

{

int v=edge[i].to;

if(vis[v])continue;

calc(v,edge[i].w,-1);

sum=siz[v];root=0;

getroot(v,0);

solve(root);

}

}

int main()

{

scanf("%d%d",&n,&m);

for(int i=1;i<n;i++)

{

int x,y,z;

scanf("%d%d%d",&x,&y,&z);

addedge(x,y,z);addedge(y,x,z);

}

sum=f[0]=n;

getroot(1,0);

solve(root);

for(int i=1;i<=m;i++)

{

scanf("%d",&k);

if(ans[k])puts("AYE");else puts("NAY");

}

return 0;

}

**4.13 LCA**

**4.13.1倍增版**

#include<iostream>

#include<cstring>

#include<algorithm>

#include<cstdio>

using namespace std;

int n,m,s;

int x,y;

int fa[22][500005],d[500005];

bool vis[500005];

struct node

{

int from,to,next;

}edge[1000005];

int cnt=0,head[500005];

void addedge(int from,int to){

cnt++;

edge[cnt].from=from;

edge[cnt].to=to;

edge[cnt].next=head[from];

head[from]=cnt;

}

void dfs(int now,int dep)

{

vis[now]=1;

d[now]=dep;

for(int i=head[now];i;i=edge[i].next)

{

int v=edge[i].to;

if(vis[v])continue;

fa[0][v]=now;

dfs(v,dep+1);

}

}

int lca(int l,int r)

{

if(d[l]<d[r])swap(l,r);

for(int i=21;i>=0;i--)

{

if((d[l]-d[r])>>i&1)

{

l=fa[i][l];

}

}

if(l==r)return l;

for(int i=21;i>=0;i--)

{

if(fa[i][l]!=fa[i][r])

{

l=fa[i][l];

r=fa[i][r];

}

}

return fa[0][l];

}

int main()

{

scanf("%d%d%d",&n,&m,&s);

for(int i=1;i<n;i++)

{

scanf("%d%d",&x,&y);

addedge(x,y);addedge(y,x);

}

dfs(s,0);

for(int i=1;i<=21;i++)

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

{

fa[i][j]=fa[i-1][fa[i-1][j]];

}

for(int i=1;i<=m;i++)

{

scanf("%d%d",&x,&y);

printf("%d\n",lca(x,y));

}

return 0;

}

**4.13.2 ST表版本**

#include<iostream>

#include<cstdio>

#include<cstring>

#include<cmath>

using namespace std;

int n,m,t;

const int N=500005,M=20;

int dp[2\*N][M];

bool vis[N];

struct edge

{

int u,v,w,next;

}e[2\*N];

int tot,head[N];

void add(int u,int v,int w,int &k)

{

e[k].u=u;e[k].v=v;e[k].w=w;

e[k].next=head[u];head[u]=k++;

u=u^v;v=u^v;u=u^v;

e[k].u=u;e[k].v=v;e[k].w=w;

e[k].next=head[u];head[u]=k++;

}

int ver[2\*N],d[2\*N],first[N],dir[N];

//ver:节点编号 d：深度 first：点编号位置 dir：距离

void dfs(int u,int dep)

{

vis[u]=1;ver[++tot]=u;

first[u]=tot;d[tot]=dep;

for(int k=head[u];k!=-1;k=e[k].next)

if(!vis[e[k].v])

{

int v=e[k].v,w=e[k].w;

dir[v]=dir[u]+w;

dfs(v,dep+1);

ver[++tot]=u;d[tot]=dep;

}

}

void st(int n)

{

for(int i=1;i<=n;i++)dp[i][0]=i;

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

{

for(int i=1;i+(1<<j)-1<=n;i++)

{

int a=dp[i][j-1],b=dp[i+(1<<(j-1))][j-1];//保存的是编号

dp[i][j]=d[a]<d[b]?a:b;

}

}

}

int rmq(int l,int r)

{

int k=0;

while((1<<(k+1))<=r-l+1)k++;

int a=dp[l][k],b=dp[r-(1<<k)+1][k];

return d[a]<d[b]?a:b;

}

int lca(int u,int v)

{

int x=first[u],y=first[v];

if(x>y)swap(x,y);

int res=rmq(x,y);

return ver[res];

}

int main()

{

int q,num=0;

scanf("%d%d%d",&n,&q,&t);

memset(head,-1,sizeof(head));

memset(vis,0,sizeof(vis));

for(int i=1;i<n;i++)

{

int u,v,w;

scanf("%d%d",&u,&v);

add(u,v,1,num);

}

tot=0;dir[1]=0;

dfs(t,1);

st(2\*n-1);

while(q--)

{

int u,v;

scanf("%d%d",&u,&v);

int ans=lca(u,v);

printf("%d\n",ans);

}

return 0;

}

**4.13.3 树剖版本**

#include<iostream>

#include<cstdio>

#include<cstring>

#include<queue>

#include<set>

#include<map>

#include<vector>

#include<stack>

#include<cmath>

#include<string>

#include<algorithm>

#include<iomanip>

#define md 1000000007

#define inf\_m 1000005

#define inf\_n 500005

#define eps 1e-6

typedef long long int ll;

using namespace std;

int n,q,s,cnt=0,cnt2=0;

//cnt建图用 cnt2 dfs用

struct node

{

int to,nxt;

}edge[inf\_m];

int head[inf\_n];

int siz[inf\_n],son[inf\_n],dep[inf\_n],fa[inf\_n],top[inf\_n],id[inf\_n],rnk[inf\_n];

/\*

siz:子树大小 son:重儿子编号 dep:深度 fa:父亲 top:所在重链顶端节点编号

id:保存每个节点剖分后的新编号(id->dfsid) rnk:dfs序(dfsid->id)

\*/

void addedge(int x,int y)

{

cnt++;

edge[cnt].to=y;

edge[cnt].nxt=head[x];

head[x]=cnt;

}

void dfs1(int t,int f)

{

fa[t]=f;

siz[t]=1;

dep[t]=dep[f]+1;

for(int i=head[t];i;i=edge[i].nxt)

{

int v=edge[i].to;

if(v!=f)

{

dfs1(v,t);

siz[t]+=siz[v];

if(son[t]==0||siz[v]>siz[son[t]])

son[t]=v;

}

}

}

void dfs2(int t,int f)

{

top[t]=f;

id[t]=cnt2;

rnk[cnt2]=t;

cnt2++;

if(!son[t])return;

dfs2(son[t],f);

for(int i=head[t];i;i=edge[i].nxt)

{

int v=edge[i].to;

if(v!=son[t]&&v!=fa[t])

{

dfs2(v,v);

}

}

}

int query(int x,int y)

{

int fx=top[x],fy=top[y];

while(fx!=fy)

{

if(dep[fx]>dep[fy])

{

x=fa[fx];

}

else y=fa[fy];

fx=top[x];fy=top[y];

}

if(dep[x]>dep[y])return y;else return x;

}

int main()

{

scanf("%d%d%d",&n,&q,&s);

for(int i=1;i<n;i++)

{

int x,y;

scanf("%d%d",&x,&y);

addedge(x,y);

addedge(y,x);

}

dfs1(s,0);

dfs2(s,s);

for(int i=1;i<=q;i++)

{

int x,y;

scanf("%d%d",&x,&y);

printf("%d\n",query(x,y));

}

return 0;

}

**5.2 树状数组**

void update(int x,int y,int n){

for(int i=x;i<=n;i+=lowbit(i))

c[i] += y;

}

int getsum(int x){

int ans = 0;

for(int i=x;i;i-=lowbit(i))

ans += c[i];

return ans;

}

**5.3线段树（区间修改）**

#define maxn 100005

typedef long long int ll;

int n,m;

struct node

{

int l,r;ll sum,add;

}tree[maxn\*5];

int a[maxn];

void build(int pos,int l,int r)

{

tree[pos].l=l;tree[pos].r=r;

if(l==r)

{

tree[pos].sum=a[l];

return;

}

int mid=(l+r)>>1;

build(pos\*2,l,mid);

build(pos\*2+1,mid+1,r);

tree[pos].sum=tree[pos\*2].sum+tree[pos\*2+1].sum;

}

void spread(int t)

{

if(tree[t].add)

{

tree[t\*2].sum+=tree[t].add\*(ll)(tree[t\*2].r-tree[t\*2].l+1);

tree[t\*2+1].sum+=tree[t].add\*(ll)(tree[t\*2+1].r-tree[t\*2+1].l+1);

tree[t\*2].add+=tree[t].add;

tree[t\*2+1].add+=tree[t].add;

tree[t].add=0;

}

}

void add(int t,int l,int r,ll x)

{

if(l<=tree[t].l&&r>=tree[t].r)

{

tree[t].sum+=(ll)(tree[t].r-tree[t].l+1)\*x;

tree[t].add+=x;

return;

}

spread(t);

int mid=(tree[t].l+tree[t].r)>>1;

if(l<=mid)add(t\*2,l,r,x);

if(r>mid)add(t\*2+1,l,r,x);

tree[t].sum=tree[t\*2].sum+tree[t\*2+1].sum;

}

ll query(int pos,int l,int r)

{

if(tree[pos].l>=l&&tree[pos].r<=r)

return tree[pos].sum;

spread(pos);

int mid=(tree[pos].l+tree[pos].r)>>1;

ll ans=0;

if(l<=mid)ans+=query(pos\*2,l,r);

if(r>mid)ans+=query(pos\*2+1,l,r);

return ans;

}

int main()

{

scanf("%d%d",&n,&m);

for(int i=1;i<=n;i++)

{

scanf("%d",&a[i]);

}

build(1,1,n);

for(int i=1;i<=m;i++)

{

int x,y,z,k;

scanf("%d",&z);

if(z==1)

{

scanf("%d%d%d",&x,&y,&k);

add(1,x,y,k);

}

else

{

scanf("%d%d",&x,&y);

printf("%lld\n",query(1,x,y));

}

}

return 0;

}

**5.4可持久化线段树（主席树）**

**5.4.1 静态版本（Luogu3834）**

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <cmath>

#include <vector>

#include <queue>

#include <iostream>

#define inf 2000000005

#define md 1000000007

#define maxn 200005

using namespace std;

typedef long long int ll;

int n,m;

int a[maxn];//a:原数组 no:离散后编号对应原数组中值 len:离散后长度

//P:节点编号计数器 lc:左儿子编号 rc:右儿子编号

//val:当前节点的值 rt:第k个版本根节点的编号

int no[maxn],len=0,P=0,rt[maxn],lc[maxn<<5],rc[maxn<<5],val[maxn<<5];

void build(int &o,int l,int r)

{

o=++P;

if(l==r)return;

int mid=(l+r)>>1;

build(lc[o],l,mid);

build(rc[o],mid+1,r);

}

int modify(int o,int l,int r,int pos)//o:原节点编号 oo:新节点编号 pos:修改的位置

{

int oo=++P;

lc[oo]=lc[o];rc[oo]=rc[o];val[oo]=val[o]+1;//复制一份原来的 并且改一下新节点的值

if(l==r)return oo;

int mid=(l+r)>>1;

if(pos<=mid)lc[oo]=modify(lc[oo],l,mid,pos);

else rc[oo]=modify(rc[oo],mid+1,r,pos);

return oo;

}

int query(int o1,int o2,int l,int r,int rnk)

{

int mid=(l+r)>>1,x=val[lc[o2]]-val[lc[o1]];

if(l==r)return l;

if(rnk<=x)return query(lc[o1],lc[o2],l,mid,rnk);

else return query(rc[o1],rc[o2],mid+1,r,rnk-x);

}

int main()

{

scanf("%d%d",&n,&m);

for(int i=1;i<=n;i++)

{

scanf("%d",&a[i]);

no[++len]=a[i];

}

sort(no+1,no+len+1);

len=unique(no+1,no+len+1)-no-1;//离散化

build(rt[0],1,len);//建空树

for(int i=1;i<=n;i++)

{

int pos=lower\_bound(no+1,no+len+1,a[i])-no;

rt[i]=modify(rt[i-1],1,len,pos);

}

for(int i=1;i<=m;i++)

{

int ql,qr,qpos;

scanf("%d%d%d",&ql,&qr,&qpos);

int ans=query(rt[ql-1],rt[qr],1,len,qpos);

printf("%d\n",no[ans]);

}

}

**5.4.2 动态版本（动态区间第k大）**

#include<bits/stdc++.h>

using namespace std;

inline int gi(){

char a=getchar();int b=0;

while(a<'0'||a>'9')a=getchar();

while(a>='0'&&a<='9')b=b\*10+a-'0',a=getchar();

return b;

}

const int N=1e4+50;

struct node {int l,r,x;} t[N\*900];

struct ppp {int l,r,op,k;} b[N];

int cmp(int\* x,int\* y) {return \*x<\*y;}

int a[N],p[N\*5],n,m,tot1,tot2,lshh,cnt=1,root[N],root1[N],q1[N],q2[N]; int \*lsh[N\*5];

void work(){

sort(lsh+1,lsh+lshh+1,cmp);p[0]=-1;

for(int i=1,j=0;i<=lshh;++i)

{

if(\*lsh[i]!=p[j])p[++j]=\*lsh[i];

\*lsh[i]=j;

}

}

int modify(int l,int r,int x,int k,int o){

int y=++cnt;

t[y]=t[x];t[y].x+=o;

if(l==r)return y;

int mid=(l+r)>>1;

if(k<=mid)t[y].l=modify(l,mid,t[x].l,k,o);

else t[y].r=modify(mid+1,r,t[x].r,k,o);

return y;

}

int query(int l,int r,int s1,int s2,int k){

if(l==r)return l;

int x=t[t[s2].l].x-t[t[s1].l].x;

for(int i=1;i<=tot1;++i)x-=t[t[q1[i]].l].x;

for(int i=1;i<=tot2;++i)x+=t[t[q2[i]].l].x;

int mid=(l+r)>>1;

if(x>=k)

{

for(int i=1;i<=tot1;++i)q1[i]=t[q1[i]].l;

for(int i=1;i<=tot2;++i)q2[i]=t[q2[i]].l;

return query(l,mid,t[s1].l,t[s2].l,k);

}

else

{

for(int i=1;i<=tot1;++i)q1[i]=t[q1[i]].r;

for(int i=1;i<=tot2;++i)q2[i]=t[q2[i]].r;

return query(mid+1,r,t[s1].r,t[s2].r,k-x);

}

}

int main(){

cin>>n>>m;

for(int i=1;i<=n;++i)

{

a[i]=gi();

lsh[++lshh]=&a[i];

}

for(int i=1;i<=m;++i)

{

char aa=getchar();

while(!(aa=='Q'||aa=='C'))aa=getchar();

b[i].l=gi();

b[i].r=gi();

if(aa=='C')

{

b[i].op=1;

lsh[++lshh]=&b[i].r;

}

else b[i].k=gi();

}

work();

for(int i=1;i<=n;++i)

root1[i]=root[1];

for(int i=1;i<=n;++i)

root[i]=modify(1,lshh,root[i-1],a[i],1);

for(int i=1;i<=m;++i)

if(b[i].op)

{

int x=b[i].l,y=b[i].r,s=a[x];a[x]=y;

while(x<=n)

{

root1[x]=modify(1,lshh,root1[x],s,-1);

root1[x]=modify(1,lshh,root1[x],y,1);

x+=(x&(-x));

}

}

else

{

tot1=0,tot2=0;int x=b[i].l-1;

while(x){q1[++tot1]=root1[x];x-=(x&(-x));}x=b[i].r;

while(x){q2[++tot2]=root1[x];x-=(x&(-x));}

printf("%d\n",p[query(1,lshh,root[b[i].l-1],root[b[i].r],b[i].k)]);

}

return 0;

}

**5.5并查集带权**

for(int i=1;i<=n;i++) fa[i]=i,d[i]=0,s[i]=1;

int fa[N],d[N],s[N];

inline int find(int x){

if(x==fa[x]) return x;

int root=find(fa[x]);

d[x]+=d[fa[x]];

return fa[x]=root;

}

**5.6 单调栈 求全flag最大子矩阵**

void sol(int flag){

memset(tot,0,sizeof(tot));

for(int i=1;i<=n;i++){

top=0;

for(int j=1;j<=m;j++){

if(a[i][j]==flag) tot[j]++;

else tot[j]=0;

data t;

t.h=tot[j];t.l=1;t.pos=j;

while(top&&st[top].h>=t.h){

int l=st[top].l+j-1-st[top].pos,h=st[top].h;

ans1=max(ans1,min(l,h)\*min(l,h));

ans2=max(ans2,l\*h);

t.l+=st[top].l;

top--;

}

st[++top]=t;

}

while(top){

int l=st[top].l+m-st[top].pos,h=st[top].h;

ans1=max(ans1,min(l,h)\*min(l,h));

ans2=max(ans2,l\*h);

top--;

}

}

}

**5.7 ST表**

int a[N],f[N][21];

void init(int n){

for(int i=1;i<=n;i++) f[i][0]=a[i];

for(int j=1;j<=20;j++)

for(int i=1;i+(1<<j)-1<=n;i++)

f[i][j]=min(f[i][j-1],f[i+(1<<(j-1))][j-1]);

}

int RMQ(int l,int r){

int k=log(r-l+1)/log(2); //2^k<=l~r

return min(f[l][k],f[r-(1<<k)+1][k]);

}

**5.8 李超线段树**

/\*

调用：update(1,x0,x1,cc)即可，其中x0,x1为线段两端点的x坐标（x0<x1），cc是下标，满足k[cc]和b[cc]为线段的斜率和截距。特别注意x0=x1时要特判，即：

if( x0 == x1 ) k[cc] = 0 , b[cc] = max( y0 , y1 );

else k[cc] = (double)(y1-y0)/(x1-x0) , b[cc] = y1 - k[cc] \* x1;

update(1,x0,x1,cc);

查询：query(1,x)即可，其中x为要查询的垂直于x轴直线的横坐标。

\*/

const int maxn=4e4+1e2,maxm=1e5+1e2;

const double eps=1e-8;

int l[maxn<<3],r[maxn<<3],lson[maxn<<3],rson[maxn<<3],dat[maxn<<3],cnt;

double k[maxm],b[maxm];//分别存斜率和截距

inline int dcmp(double x) {

return fabs(x) <= eps ? 0 : x < 0 ? -1 : 1;

}

inline void build(int pos,int ll,int rr) {

l[pos] = ll , r[pos] = rr;

if( ll == rr ) return;

const int mid = ( ll + rr ) >> 1;

build(lson[pos]=++cnt,ll,mid);

build(rson[pos]=++cnt,mid+1,rr);

}

inline double f(int x,int p) {

return k[x] \* p + b[x];

}

inline bool judge(int x,int y,int p) {

double fx = f(x,p) , fy = f(y,p);

return dcmp(fx-fy) ? fx < fy : x > y;

}

inline void update(int pos,int ll,int rr,int nw) {

if( r[pos] < ll || rr < l[pos] ) return;

if( ll <= l[pos] && r[pos] <= rr ) {

if( judge(nw,dat[pos],l[pos]) && judge(nw,dat[pos],r[pos]) ) return;

// 当前线段在区间内完全不优

if( judge(dat[pos],nw,l[pos]) && judge(dat[pos],nw,r[pos]) ) {

// 当前线段在区间内完全优

dat[pos] = nw;

return;

}

const int mid = ( l[pos] + r[pos] ) >> 1;

if( judge(dat[pos],nw,mid) ) swap(dat[pos],nw);

if( judge(dat[pos],nw,l[pos]) ) update(lson[pos],l[pos],r[pos],nw);

else update(rson[pos],l[pos],r[pos],nw);

return; // you will get TLE without this return

}

update(lson[pos],ll,rr,nw) , update(rson[pos],ll,rr,nw);

}

inline int query(int pos,int x) {

if( x < l[pos] || r[pos] < x ) return 0;

if( l[pos] == r[pos] && l[pos] == x ) return dat[pos];

const int mid = ( l[pos] + r[pos] ) >> 1;

int ret = x <= mid ? query(lson[pos],x) : query(rson[pos],x);

if( judge(ret,dat[pos],x) ) ret = dat[pos];

return ret;

}

**5.9 LCT**

函数说明：

access(x) 是把x 到x 所在树的根路径上的所有节点组成一棵splay，并且把x 节点旋转到splay 的根节点上。

makeroot(x) 换根，让x成为根。

findroot(x) 找x所在原树的树根，主要用来判断两点之间的连通性。

（findroot(x)==findroot(y)表明x,y在同一棵树中）

split(x,y) 拉出x-y的路径成为一个Splay

cut(x,y) 切断x和y。

link(x,y) 连边x->y。

rotate(x) 是把x 节点旋转到splay 上他的父亲节点的位置。

splay(x) 是把x 节点旋转到splay 的根节点上。

#define maxn 300005

using namespace std;

int n,m,a[maxn];

struct sf

{

int l,r,par,rev,val;

}f[maxn];

bool isroot(int x)

{

return f[f[x].par].l!=x&&f[f[x].par].r!=x;

}

void push(int x)

{

if(f[x].rev)

{

f[f[x].l].rev^=1;

f[f[x].r].rev^=1;

swap(f[x].l,f[x].r);

f[x].rev=0;

}

}

void update(int x)

{

f[x].val=f[f[x].l].val^f[f[x].r].val^a[x];//根据实际需要改

}

void rotate(int x)

{

int y=f[x].par,z=f[y].par;

push(y);push(x);

if(f[y].l==x)f[f[y].l=f[x].r].par=y,f[x].r=y;

else f[f[y].r=f[x].l].par=y,f[x].l=y;

if(f[z].l==y)f[z].l=x;

if(f[z].r==y)f[z].r=x;

f[f[y].par=x].par=z;

update(y);

}

void splay(int x)

{

push(x);

for(;!isroot(x);rotate(x))

{

int y=f[x].par,z=f[y].par;

if(!isroot(y))rotate(f[z].l==y^f[y].l==x?x:y);

}

update(x);

}

void access(int x)

{

for(int t=0,y=x;y;y=f[y].par)

splay(y),f[y].r=t,update(t=y);

splay(x);

}

void makeroot(int x)

{

access(x);splay(x);f[x].rev^=1;

}

int findroot(int x)

{

access(x);splay(x);

while(f[x].l)push(x),x=f[x].l;

splay(x);

return x;

}

void link(int x,int y)

{

makeroot(x);

if(findroot(y)!=x) f[x].par=y;

}

void cut(int x,int y)

{

makeroot(x);

if(findroot(y)==x&&f[y].par==x&&!f[y].l)

{

f[y].par=f[x].r=0;

update(x);

}

}

void split(int x,int y)

{

makeroot(x);access(y);splay(y);

}

int main()

{

scanf("%d%d",&n,&m);

for(int i=1;i<=n;i++)scanf("%d",&a[i]);

for(int i=1;i<=m;i++)

{

int ope,x,y;

scanf("%d%d%d",&ope,&x,&y);

if(ope==0){split(x,y);printf("%d\n",f[y].val);}

else if(ope==1){link(x,y);}

else if(ope==2){cut(x,y); }

else{splay(x);a[x]=y;//注意要先splay再改，不然会出错}

}

}

**6.1 高精度**

#include<cstdio>

#include<cstring>

#include<cstdlib>

#define max(x,y) ((x)>(y)?(x):(y))

#define MAXDIG 500000

using namespace std;

int n;

struct hll{

int num,dig[MAXDIG];

void setas(int x){

int i=1;

num=1;

if(x==0)dig[num]=0;

while(x){

num=i++;

dig[num]=x%10;

x/=10;

}

}

hll operator \*(const hll &y){

hll t;

t.num=num+y.num;

t.dig[0]=1;

for(int i=1;i<=t.num;i++)t.dig[i]=0;

for(int i=1;i<=num;i++)

for(int j=1;j<=y.num;j++){

t.dig[i+j-1]+=dig[i]\*y.dig[j];

t.dig[i+j]+=t.dig[i+j-1]/10;

t.dig[i+j-1]%=10;

}

while(!t.dig[t.num])t.num--;

if(!t.num)t.num=1;

return t;

}

hll operator +(const hll y){

hll t;

t.num=max(num,y.num)+1;

t.dig[0]=1;

for(int i=1;i<=t.num;i++)t.dig[i]=0;

for(int i=1;i<=t.num-1;i++){

t.dig[i]+=(num>=i?dig[i]:0)+(y.num>=i?y.dig[i]:0);

t.dig[i+1]+=t.dig[i]/10;

t.dig[i]%=10;

}

while(!t.dig[t.num])t.num--;

if(!t.num)t.num=1;

return t;

}

bool operator >(const hll &y){

if(num>y.num)return 1;

if(num<y.num)return 0;

for(int i=num;i>=1;i--){

if(dig[i]>y.dig[i])return 1;

if(dig[i]<y.dig[i])return 0;

}

return 0;

}

void dou(){

hll t;t.setas(2);

(\*this)=(\*this)\*t;

}

hll cdou(){

hll t;t.setas(2);

return (\*this)\*t;

}

void print(){

for(int i=num;i>0;i--)printf("%d",dig[i]);

printf("\n");

}

void read(){

char ch=getchar();num=0;

while(ch<'0'||ch>'9'){ch=getchar();}

while(ch>='0'&&ch<='9'){dig[++num]=ch-'0';ch=getchar();}

for(int i=1;i<=num/2;i++){dig[i]+=dig[num-i+1];dig[num-i+1]=dig[i]-dig[num-i+1];dig[i]=dig[i]-dig[num-i+1];}

}

}b,c;

int read(){

int 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;

}

int main(){

n=read();

if(n==29){

printf("8841716993739701954543616000000");

return 0;

}

c.setas(1);b.setas(1);

for(int i=2;i<=n;i++){

c.setas(i);

b=b\*c;

}

b.print();

return 0;

}

**6.2 数位DP**

#include<iostream>

#include<cstring>

#include<cstdio>

#include<algorithm>

using namespace std;

int ansa[10],ansb[10];

void count (int now,int ans[],int times=1)

{

if(now<=0)return;

int p=now/10,d=now%10;

for(int i=1;i<=d;i++)ans[i]+=times;

while(p>0)

{

ans[p%10]+=(d+1)\*times;

p/=10;

}

for(int i=0;i<=9;i++)ans[i]+=times\*(now/10);

times\*=10;

count(now/10-1,ans,times);

return;

}

int main()

{

int a,b;

while(1)

{

scanf("%d%d",&a,&b);

if(a==0&&b==0)break;

memset(ansa,0,sizeof(ansa));

memset(ansb,0,sizeof(ansb));

if(a>b)

{

int temp=a;a=b;b=temp;

}

a--;

if(b>a)

{

count(b,ansb);

count(a,ansa);

}

for(int i=0;i<=9;i++)

printf("%d%c",ansb[i]-ansa[i],i==9?'\n':' ');

}

return 0;

}

**6.3 CDQ分治（二维偏序）**

#include<cstdio>

#include<cstring>

#include<iostream>

#define ll long long

using namespace std;

const int N=5000010;

int n,m,totx=0,tot=0; //totx是操作的个数,tot询问的编号

struct node{

int type,id;

ll val;

bool operator < (const node &a) const //重载运算符,优先时间排序

{

if (id!=a.id) return id<a.id;

else return type<a.type;

}

};

node A[N],B[N];

ll ans[N];

void CDQ(int L,int R)

{

if (L==R) return;

int M=(L+R)>>1;

CDQ(L,M);

CDQ(M+1,R);

int t1=L,t2=M+1;

ll sum=0;

for (int i=L;i<=R;i++)

{

if ((t1<=M&&A[t1]<A[t2])||t2>R) //只修改左边区间内的修改值

{

if (A[t1].type==1) sum+=A[t1].val; //sum是修改的总值

B[i]=A[t1++];

}

else //只统计右边区间内的查询结果

{

if (A[t2].type==3) ans[A[t2].val]+=sum;

else if (A[t2].type==2) ans[A[t2].val]-=sum;

B[i]=A[t2++];

}

}

for (int i=L;i<=R;i++) A[i]=B[i];

}

int main()

{

scanf("%d%d",&n,&m);

for (int i=1;i<=n;i++)

{

tot++;

A[tot].type=1; A[tot].id=i; //修改操作

scanf("%lld",&A[tot].val);

}

for (int i=1;i<=m;i++)

{

int t;

scanf("%d",&t);

tot++;

A[tot].type=t;

if (t==1)

scanf("%d%lld",&A[tot].id,&A[tot].val);

else

{

int l,r;

scanf("%d%d",&l,&r);

totx++;

A[tot].val=totx; A[tot].id=l-1; //询问的前一个位置

tot++; A[tot].type=3; A[tot].val=totx; A[tot].id=r; //询问的后端点

}

}

CDQ(1,tot);

for (int i=1;i<=totx;i++) printf("%lld\n",ans[i]);

return 0;

}

**6.4三分**

#include<iostream>

#include<cstring>

#include<cstdio>

#include<algorithm>

#include<cmath>

#include<queue>

#define inf 200000000

#define maxm 10005

#define maxn 10005

using namespace std;

typedef long long int ll;

typedef double db;

const db eps=1e-7;

int n;

db l,r,a[15];

double f(double x)

{

double sum=0;

for(int i=0;i<=n;i++)

sum+=(db)pow(x,(db)i)\*a[i];

return sum;

}

int main()

{

scanf("%d%lf%lf",&n,&l,&r);

for(int i=n;i>=0;i--)scanf("%lf",&a[i]);

while(r-l>eps)

{

double mid=(l+r)/2,midd=(mid+r)/2;

if(f(mid)<f(midd))

l=mid;

else r=midd;

}

printf("%.5lf\n",f(l)>f(r)?l:r);

}

**6.5矩阵快速幂**

#include<iostream>

#include<cstring>

#include<cstdio>

#include<algorithm>

#include<cmath>

#include<queue>

#define inf 200000000

#define maxm 10005

#define maxn 205

#define md 1000000007

using namespace std;

typedef long long int ll;

struct mat

{

int n,m;

ll a[15][15];

};

mat mul(mat x,mat y)

{

mat ans;

memset(ans.a,0,sizeof(ans.a));

ans.n=x.n;ans.m=y.m;

for(int i=1;i<=x.n;i++)

for(int j=1;j<=y.m;j++)

for(int k=1;k<=x.m;k++)

ans.a[i][j]=(ans.a[i][j]+x.a[i][k]\*y.a[k][j])%md;

return ans;

}

mat matpow(mat c,int t)

{

if(t<=1)return c;

mat ans=c;t--;

while(t>0)

{

if(t%2==1)

{

ans=mul(ans,c);

}

t/=2;

c=mul(c,c);

}

return ans;

}

int main()

{

int T;

scanf("%d",&T);

mat c,d;

c.n=c.m=3;

c.a[1][1]=1;c.a[1][2]=1;c.a[1][3]=0;

c.a[2][1]=0;c.a[2][2]=0;c.a[2][3]=1;

c.a[3][1]=1;c.a[3][2]=0;c.a[3][3]=0;

d.n=1;d.m=3;

d.a[1][1]=d.a[1][2]=d.a[1][3]=1;

while(T--)

{

int n;

scanf("%d",&n);

if(n<=3)

{

printf("1\n");continue;

}

mat t=matpow(c,n-1);

//t=mul(d,t);

printf("%d\n",t.a[1][1]);

}

return 0;

}

**6.6动态DP（洛谷P4719）**

#define maxn 100005

using namespace std;

typedef long long int ll;

struct matrix

{

ll e[2][2];

matrix()

{

memset(e,0,sizeof(e));

}

matrix operator \* (const matrix &b) const

{

matrix c;

for(int i=0;i<2;i++)

for(int j=0;j<2;j++)

for(int k=0;k<2;k++)

c.e[i][j]=max(c.e[i][j],e[i][k]+b.e[k][j]);

//这一行是矩阵乘积，根据具体需要改

return c;

}

};

struct node

{

int l,r;

matrix dat;

}tree[4\*maxn]; //线段树

int n,m;//n点数 m为查询个数

ll val[maxn];//原树每个点的点权

vector<int>g[maxn];//存图

int siz[maxn],hson[maxn],fa[maxn],anc[maxn],sid[maxn],ed[maxn],tot=0,real[maxn],dp[maxn][2],ldp[maxn][2];

//树剖用：siz子树大小 hson重儿子 fa父亲 anc重链顶端的节点

//线段树：sid[i]节点i的dfs序编号，real[i]表示dfs序为i的节点

//tot：总点数（=n） ed[i]表示以i为根的重链最下端的节点的dfs序

matrix trans[maxn];//相当于线段树中 [l,l]节点存储的值

void build(int now,int l,int r)//建树

{

tree[now].l=l;tree[now].r=r;

if(l==r)

{

int u=real[l],tmp=g[u].size();

ldp[u][1]=val[u];

for(int i=0;i<tmp;i++)

{

int v=g[u][i];

if(v!=fa[u]&&v!=hson[u])

{

ldp[u][0]+=max(dp[v][0],dp[v][1]);

ldp[u][1]+=dp[v][0];

}

}

matrix\* tmp2=&tree[now].dat;

tmp2->e[0][0]=tmp2->e[0][1]=ldp[u][0];

tmp2->e[1][0]=ldp[u][1];tmp2->e[1][1]=0;

trans[l]=\*tmp2;

return;

}

int mid=(l+r)>>1;

build(now<<1,l,mid);

build(now<<1|1,mid+1,r);

tree[now].dat=tree[now<<1].dat\*tree[now<<1|1].dat;

}

void change(int now,int pos,matrix v)

{

if(tree[now].l==tree[now].r)

{

tree[now].dat=v;

return;

}

int mid=(tree[now].l+tree[now].r)>>1;

if(pos<=mid)change(now<<1,pos,v);

else change(now<<1|1,pos,v);

tree[now].dat=tree[now<<1].dat\*tree[now<<1|1].dat;

}

matrix query(int now,int l,int r)

{

if(tree[now].l>=l&&tree[now].r<=r)return tree[now].dat;

int mid=(tree[now].l+tree[now].r)>>1;

if(r<=mid)return query(now<<1,l,r);

else if(l>mid)return query(now<<1|1,l,r);

else return query(now<<1,l,r)\*query(now<<1|1,l,r);

}

void dfs(int now,int f)

{

int tmp=g[now].size();

siz[now]=1;

hson[now]=0;

fa[now]=f;

int mx=0;

for(int i=0;i<tmp;i++)

{

int v=g[now][i];

if(v==f)continue;

dfs(v,now);

siz[now]+=siz[v];

if(siz[v]>mx)mx=siz[v],hson[now]=v;

}

}

void dfs2(int now,int f)

{

int tmp=g[now].size();

anc[now]=f;

sid[now]=++tot;

real[tot]=now;

ed[f]=tot;

if(hson[now]==0)return;

dfs2(hson[now],f);

for(int i=0;i<tmp;i++)

{

int v=g[now][i];

if(v==fa[now]||v==hson[now])continue;

dfs2(v,v);

}

}

void path\_change(int u,ll x)//把节点u上的权值改成x

{

trans[sid[u]].e[1][0]+=x-val[u];

val[u]=x;

matrix od,nw;

while(u)

{

od=query(1,sid[anc[u]],ed[anc[u]]);

change(1,sid[u],trans[sid[u]]);

nw=query(1,sid[anc[u]],ed[anc[u]]);

u=fa[anc[u]];

trans[sid[u]].e[0][0]+=max(nw.e[0][0],nw.e[1][0])-max(od.e[0][0],od.e[1][0]);

trans[sid[u]].e[0][1]=trans[sid[u]].e[0][0];

trans[sid[u]].e[1][0]+=nw.e[0][0]-od.e[0][0];

//以上三行根据转移方程改

}

}

void Tree\_init()

{

dfs(1,0);

dfs2(1,1);

for(int i=n;i>=1;i--)

{

int u=real[i],tmp=g[u].size();

dp[u][1]=max(0ll,val[u]);

for(int j=0;j<tmp;j++)

{

int v=g[u][j];

if(v!=fa[u])

{

dp[u][1]+=dp[v][0];

dp[u][0]+=max(dp[v][0],dp[v][1]);

//dp转移，根据转移方程改

}

}

}

build(1,1,n);

}

int main()

{

scanf("%d%d",&n,&m);

for(int i=1;i<=n;i++)scanf("%lld",&val[i]);

for(int i=1;i<n;i++)

{

int x,y;scanf("%d%d",&x,&y);

g[x].push\_back(y);

g[y].push\_back(x);

}

Tree\_init();

for(int i=1;i<=m;i++)

{

int u;ll x;

scanf("%d%lld",&u,&x);

path\_change(u,x);

matrix tmp=query(1,sid[1],ed[1]);

printf("%lld\n",max(tmp.e[0][0],tmp.e[1][0]));

}

return 0;

}