**数据结构**

**KDTtree**

**双向链表**

void del(int x){

next[prev[x]]=next[x];

prev[next[x]]=prev[x];

}

for (int i=1; i<=n; i++) next[i]=i+1, prev[i]=i-1;

next[0]=1; prev[n+1]=n;

**线段树**

const int MAXN=50010;

int a[MAXN],ans[MAXN<<2],lazy[MAXN<<2];

// a[]为原序列信息，ans[]模拟线段树维护区间和，lazy[]为懒惰标记

// 若只涉及点更新的题，只需用(1)(2)(4)(6)

// 若只涉及区间更新的题，需用(1)(2)(3)(5)(6)

// 若为两种更新都有，则在所有向子区间查询或更新前，都需PushDown()

void PushUp(int rt){

ans[rt]=ans[rt<<1]+ans[rt<<1|1];

}

// (2)建树

void Build(int l,int r,int rt){

if (l==r){

ans[rt]=a[l];

return;

}

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

Build(l,mid,rt<<1);

Build(mid+1,r,rt<<1|1);

PushUp(rt);

}

// (3) 下推懒惰标记

void PushDown(int rt,int ln,int rn){

//ln表示左子树元素结点个数，rn表示右子树结点个数

if (lazy[rt]){

lazy[rt<<1]+=lazy[rt];

lazy[rt<<1|1]+=lazy[rt];

ans[rt<<1]+=lazy[rt]\*ln;

ans[rt<<1|1]+=lazy[rt]\*rn;

lazy[rt]=0;

}

}

// (4)点更新

void Add(int L,int C,int l,int r,int rt){

if (l==r){

ans[rt]+=C;

return;

}

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

//PushDown(rt,mid-l+1,r-mid); 若既有点更新又有区间更新，需要这句话

if (L<=mid) Add(L,C,l,mid,rt<<1);

else Add(L,C,mid+1,r,rt<<1|1);

PushUp(rt);

}

// (5)区间更新

void Update(int L,int R,int C,int l,int r,int rt){

if (L<=l&&r<=R){

ans[rt]+=C\*(r-l+1);

lazy[rt]+=C;

return;

}

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

PushDown(rt,mid-l+1,r-mid);

if (L<=mid) Update(L,R,C,l,mid,rt<<1);

if (R>mid) Update(L,R,C,mid+1,r,rt<<1|1);

PushUp(rt);

}

// (6)区间查询

LL Query(int L,int R,int l,int r,int rt){

if (L<=l&&r<=R)

return ans[rt];

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

PushDown(rt,mid-l+1,r-mid);//若更新只有点更新，不需要这句

LL ANS=0;

if (L<=mid) ANS+=Query(L,R,l,mid,rt<<1);

if (R>mid) ANS+=Query(L,R,mid+1,r,rt<<1|1);

return ANS;

}

//建树

Build(1,n,1);

//点更新

Add(L,C,1,n,1);

//区间修改

Update(L,R,C,1,n,1);

//区间查询

int ANS=Query(L,R,1,n,1);

**树状数组**

// 查询区间是[1, MAX]

#define lowbit(x) ((x)&(-x))

int n, arr[int(5e4)+5];

long long getsum(int idx){

long long sum=0;

for (int i=idx; i>0; i-=lowbit(i))

sum+=arr[i];

return sum;

}

void add(int idx, int num){

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

arr[i]+=num;

}

**图论**

**Dijkstra(Queue) &** **BellmanFord(Stack/Queue)**

const int maxn=3e4+20, maxm=15e4+20, INF=0x3f3f3f3f;

typedef pair<int, int> Node;

struct Cmp{

bool operator () (const Node &a, const Node &b){

return a.first>b.first;

}

};

struct Edge{

int to, dis, next;

}edges[maxm+5];

int head[maxn+5], size=0;

void addEdge(int from, int to, int dis){

edges[size]=Edge{to, dis, head[from]};

head[from]=size++;

}

void init(void){

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

size=0;

}

int Bellman(int n){

int dist[maxn+5], sta[maxn+5], top=0;//cnt[maxn+5];

bool inq[maxn+5]={false};

// queue<int> que;

memset(dist, INF, sizeof(dist)); dist[1]=0;

sta[top++]=1;

while (top!=0){

int from=sta[--top];

inq[from]=false;

for (int i=head[from]; i!=-1; i=edges[i].next){

Edge &e=edges[i];

int &to=e.to, &dis=e.dis;

if (dist[to]<=dist[from]+dis) continue;

dist[to]=dist[from]+dis;

if (inq[to]) continue;

sta[top++]=to; inq[to]=true;

}

}return dist[n];

}

int Dij(int n){

int dist[maxn+5];

priority\_queue<Node, vector<Node>, Cmp> que;

memset(dist, INF, sizeof(dist)); dist[1]=0;

que.push(Node(dist[1], 1));

while (que.size()){

Node x=que.top(); que.pop();

if (x.first!=dist[x.second]) continue;

int &from=x.second;

for (int i=head[from]; i!=-1; i=edges[i].next){

Edge &e=edges[i];

int &to=e.to, &dis=e.dis;

if (dist[to]<=dist[from]+dis) continue;

dist[to]=dist[from]+dis;

que.push(Node(dist[to], to));

}

}return dist[n];

}

int main(void){

int n, m, from, to, dis;

init();

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

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

scanf("%d%d%d", &from, &to, &dis);

addEdge(from, to, dis);

}printf("%d\n", Dij(n));//Bellman(n));

return 0;

}

**Floyd**

// 若用于求最短路，需要把不存在的边权赋为INF  
// 若用于有向图传递闭包(Transitive Closure)，把边权设为1，不存在的边设为0

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

const int maxn=105, INF=0x3f3f3f3f;

int n, dist[maxn+5][maxn+5];

void Floyd(void){

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

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

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

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

if (dist[i][k]<INF && dist[k][j]<INF)

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

}

**无向图查连通**

**#include <cstdio>**

#include <cstring>

#include <queue>

using namespace std;

const int maxn=1000;

struct Node{

int parent, rank;

Node(int parent=0, int rank=0):

parent(parent), rank(rank) {}

}node[maxn+5];

int n;

int find(int x){

return (node[x].parent==x)?x:(node[x].parent=find(node[x].parent));

}

void join(int a, int b){

a=find(a); b=find(b);

if (a==b) return;

if (node[a].rank==node[b].rank) node[a].rank++;

if (node[a].rank>node[b].rank) node[b].parent=a;

else node[a].parent=b;

}

bool connect(void){

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

if (find(1)!=find(i)) return false;

return true;

}

int main(void){

while (scanf("%d", &n)==1 && n){

int m, cnt=0, vis[maxn+5]={0};

bool set[maxn+5]={false};

for (int i=1; i<=n; i++) node[i]=Node(i, 0);

scanf("%d", &m);

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

scanf("%d%d", &a, &b);

join(a, b);

vis[a]++; vis[b]++;

}

int flag=false;

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

if (vis[i]%2) {flag=true; break;}

if (flag==false && !connect()) flag=true;

printf("%d\n", (flag)?0:1);

}

return 0;

}

**二分图**

// 最大匹配O(E\*V)

const int maxn=300+20, maxp=100+20;

bool G[maxp][maxn], vis[maxn];

int match[maxn], n, p;

bool dfs(int u){

for (int i=1; i<=n; i++) if (!vis[i] && G[u][i]){

vis[i]=true;

if (match[i]==-1 || dfs(match[i])){

match[i]=u;

return true;

}

}return false;

}

int solve(void){

int ans=0;

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

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

memset(vis, false, sizeof(vis));

if (dfs(i)) ans++;

}return ans;

}

**dfs序**

int head[maxn], esize;

int tim, st[maxn], siz[maxn], fa[maxn], dfn[maxn];

void dfs(int u, int pre){

dfn[tim]=u;

st[u]=tim++; siz[u]=1; fa[u]=pre;

#define TO edges[i].to

for (int i=head[u]; i!=-1; i=edges[i].nxt)

if (TO!=pre) dfs(TO, u), siz[u]+=siz[TO];

#undef TO

}

int main(void){

int a, b, val;

while (scanf("%d", &n)==1){

init();

dfs(1, -1);

int diff[maxn]={0};

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

vector<pair<int, int> > e;

for (int i=head[u]; i!=-1; i=edges[i].nxt)

e.push\_back(make\_pair(edges[i].val, edges[i].to));

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

int ptr=0, sizes=e.size();

while (ptr<sizes){

int pre=e[ptr].first, tmp=ptr+1;

while (tmp<sizes && pre==e[tmp].first) tmp++;

if (tmp-1==ptr) {ptr++; continue;}

if (pre!=e[tmp-1].first) break;

for (; ptr<=tmp-1; ptr++){

int to=e[ptr].second;

if (to==fa[u]){

diff[st[1]]++;

diff[st[u]]--;

diff[st[u]+siz[u]]++;

}else{

diff[st[to]]++;

diff[st[to]+siz[to]]--;

}

}

}

}

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

diff[i]+=diff[i-1];

}

return 0;

}

**Dinic**

// 初始化dinic.init(n)

const int maxn=400+20, INF=1e8;

struct Edge{

int from,to,cap,flow;

Edge(int u,int v,int c,int f):

from(u), to(v), cap(c), flow(f) {}

};

struct Dinic{

int n, m, s, t;

vector<int> G[maxn];

vector<Edge> edges;

bool vis[maxn];

int dep[maxn], cur[maxn];

void init(int n){

this->n=n;

for (int i=0;i<=n;i++) G[i].clear();

edges.clear();

}

void addEdge(int from, int to, int 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(void){

memset(vis, false, sizeof(vis));

queue<int> Q;

vis[s]=true;

dep[s]=0;

Q.push(s);

while(!Q.empty()){

int x=Q.front(); Q.pop();

for(int 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;

dep[e.to]=dep[x]+1;

Q.push(e.to);

}

}

}

return vis[t];

}

int dfs(int x, int a){

if(x==t || a==0)return a;

int flow=0, f;

for(int &i=cur[x];i<G[x].size();i++) {

Edge &e=edges[G[x][i]];

if(dep[e.to]==dep[x]+1 && (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;

}

int maxFlow(int s, int t){

this->s=s; this->t=t;

int flow=0;

while(bfs()){

memset(cur, 0, sizeof(cur));

flow+=dfs(s, INF);

}

return flow;

}

}dinic;

EK费用流

void addEdge(int from, int to, int cap, double cost){

edges[esiz]=Edge(to, head[from], cap, -log(1-cost));

head[from]=esiz++;

edges[esiz]=Edge(from, head[to], 0, log(1-cost));

head[to]=esiz++;

}

bool equal(double a, double b){

return (a-b)<eps && (b-a)<eps;

}

bool Bellman(int s, int t){

queue<int> que;

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

memset(vis, false, sizeof(vis));

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

dlt[i]=INF, dis[i]=FINF;

dis[s]=0; vis[s]=true;

que.push(s);

while (!que.empty()){

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

// printf("%d -\n", u);

vis[u]=false;

for (int i=head[u]; i!=-1; i=E.nxt){

// printf("%d %d %f+\n", E.to, E.cap, E.cost);

if (E.flow>=E.cap) continue;

if (dis[E.to]<dis[u]+E.cost || equal(dis[E.to], dis[u]+E.cost)) continue;

dis[E.to]=dis[u]+E.cost;

dlt[E.to]=min(dlt[u], E.cap-E.flow);

pre[E.to]=i;

if (!vis[E.to]){

vis[E.to]=true;

que.push(E.to);

}

}

}

if (pre[t]==-1) return false;

return true;

}

pair<int, double> minCost(int s, int t){

int flow=0;

double cost=0;

while (Bellman(s, t)){

int delta=dlt[t];

for (int i=pre[t]; i!=-1; i=pre[NE.to]){

E.flow+=delta;

NE.flow-=delta;

}

cost+=dis[t]\*delta; // ??????

flow+=delta;

}return make\_pair(flow, cost);

}

割点（无向图）

int head[maxn], n, esize;

int pre[maxn], low[maxn], tim;

bool iscut[maxn];

void init(void){

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

esize=0;

}

void addEdge(int from, int to){

edges[esize]=Edge(to, head[from]);

head[from]=esize++;

}

void dfs(int u, int fa){

pre[u]=low[u]= ++tim;

int child=0;

for (int i=head[u]; i!=-1; i=edges[i].nxt){

int &to=edges[i].to;

if (!pre[to]){

child++;

dfs(to, u);

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

if (low[to]>=pre[u]) iscut[u]=true;

}else if (pre[to]<pre[u] && to!=fa)

low[u]=min(low[u], pre[to]);

}

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

}

void findCut(void){

tim=0;

memset(pre, 0, sizeof(pre));

memset(low, 0, sizeof(low));

memset(iscut, false, sizeof(iscut));

dfs(1, -1);

}

桥（无向图）

typedef pair<int, int> Pair;

struct Edge{

int to, nxt;

Edge(int to=0, int nxt=0):

to(to), nxt(nxt) {}

}edges[maxm];

int head[maxn], esize, n;

int pre[maxn], low[maxn], tim;

set<Pair> ans;

void init(void){

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

esize=0;

ans.clear();

}

void addEdge(int from, int to){

edges[esize]=Edge(to, head[from]);

head[from]=esize++;

}

void dfs(int u, int fa){

pre[u]=low[u]=++tim;

// int child=0;

for (int i=head[u]; i!=-1; i=edges[i].nxt){

int &to=edges[i].to;

if (!pre[to]){

// child++;

dfs(to, u);

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

// if (low[to]<=pre[u]) iscut[u]=true;

}else if (pre[to]<pre[u] && to!=fa)

low[u]=min(low[u], pre[to]);

if (low[to]>pre[u]) ans.insert(Pair(min(u, to), max(u, to)));

}

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

}

void getBridge(void){

tim=0;

memset(pre, 0, sizeof(pre));

memset(low, 0, sizeof(low));

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

if (!pre[i]) dfs(i, -1);

}

强连通分量（有向图）&& 双连通分量（无向图） && 缩点

// 边双连通分量求解可转化为强连通分量求解，注意取消反向边即可（i!=prev^1）

int head[maxn], esize, n, m;

int dfn[maxn], low[maxn], sccn[maxn], tim, scnt;

int deg[maxn];

stack<int> sta;

void init(void){

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

esize=0;

}

void addEdge(int from, int to){

edges[esize]=Edge(to, head[from]);

head[from]=esize++;

}

void dfs(int u, int prev){

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

sta.push(u);

for (int i=head[u]; i!=-1; i=edges[i].nxt){

int &v=edges[i].to;

if (i==(prev^1)) continue;// if (v==fa) continue; // undirect graph -> direct

if (!dfn[v]){

dfs(v, i);

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

}else if (!sccn[v])

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

}

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

scnt++;

while (true){

int x=sta.top(); sta.pop();

sccn[x]=scnt; // ssize[scnt]++;

if (x==u) break;

}

}

}

void getSCC(void){

scnt=tim=0;

memset(dfn, 0, sizeof(dfn));

memset(low, 0, sizeof(low));

memset(sccn, 0, sizeof(sccn));

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

if (!dfn[i]) dfs(i, -1);

}

int main(void){

getSSC();

memset(in, 0, sizeof(in));

memset(out, 0, sizeof(out));

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

for (int i=head[u]; i!=-1; i=edges[i].nxt){

int &v=edges[i].to;

if (sccn[u]!=sccn[v]) out[sccn[u]]++, in[sccn[v]]++;

}

}

// u is from 1 to n;

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

if (in[u]==0) …;

}

生成树 && 最小树形图

LCA

ISAP（注意设置N）

void bfs(int t){

queue<int> que;

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

memset(gap, 0, sizeof(gap));

memcpy(cur, head, sizeof(head));

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

que.push(t);

while (!que.empty()){

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

for (int i=head[u]; i!=-1; i=E.nxt) if (!dep[E.to]){

++gap[dep[E.to]=dep[u]+1];

que.push(E.to);

}

}

}

int dfs(int u, int dlt, int s, int t){

if (u==t) return dlt;

int flow=0;

for (int &i=cur[u]; i!=-1; i=E.nxt) if (dep[u]==dep[E.to]+1){

int res=dfs(E.to, min(dlt, E.cap-E.flow), s, t);

flow+=res, dlt-=res;

E.flow+=res, NE.flow-=res;

if (!dlt) return flow;

}

if (!(--gap[dep[u]])) dep[s]=N+1;

++gap[++dep[u]];

cur[u]=head[u];

return flow;

}

int maxflow(int s, int t){

bfs(t);

int flow=0;

while (dep[s]<=N) flow+=dfs(s, INF, s, t);

return flow;

}

**数学**

高斯消元

Polya

Catalan

h[n]=h[1]h[n-1]+h[2]\*h[n-2]+...+h[n-1]h[1];

=>h[i]=h[i-1]\*(4\*i-2)/(i+1)

FFT

struct Complex{

double a,b;

Complex(){}

Complex(double \_a,double \_b):a(\_a),b(\_b){}

Complex(double \_a):a(\_a),b(0.0){}

inline Complex operator + (const Complex &z)const{return Complex(a+z.a,b+z.b);}

inline Complex operator - (const Complex &z)const{return Complex(a-z.a,b-z.b);}

inline Complex operator \* (const Complex &z)const{return Complex(a\*z.a-b\*z.b,a\*z.b+b\*z.a);}

};

Complex a[MAXN],b[MAXN];

int n,m,len,R,L,rev[MAXN];

inline void FFT(Complex c[],int n,int f){

Complex wn,w,x,y;

for (int i=0;i<n;i++) if (i<rev[i])

swap(c[i],c[rev[i]]);

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

wn=Complex(cos(pi/i),sin(pi/i)\*f);

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

w=Complex(1,0);

for (int k=0;k<i;k++,w=w\*wn){

x=c[j+k];y=w\*c[j+k+i];

c[j+k]=x+y;c[j+k+i]=x-y;

}

}

}

if (!~f) for (int i=0;i<n;i++)

c[i].a/=(double)n;

}

int main()

{

n=read();m=read();len=n+m+2;

for (int i=0;i<=n;i++) a[i].a=read();

for (int i=0;i<=m;i++) b[i].a=read();

for (R=1;R<len;R<<=1,L++);

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

FFT(a,R,1);FFT(b,R,1); // DFT

for (int i=0;i<R;i++) a[i]=a[i]\*b[i];

FFT(a,R,-1);// IDFT

for (int i=0;i<len-1;i++) printf("%d%c",(int)(a[i].a+0.5),i==len-2?'\n':' ');

return 0;

}

扩展欧几里德求逆元 **O(logn)**

int exgcd(int a, int b, int &x, int &y){

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

int gcd=exgcd(b, a%b, y, x);

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

return gcd;

}

int inv(int a, int p){

int x, y, gcd=exgcd(a, p, x, y);

if (gcd==1) return (x%p+p)%p;

return -1;

}

求逆元表 O(n)  
对于不能求逆元的情况(A/B)%mod = (A%(B\*mod))/B%mod

long long inv[mod];

void init(void){

inv[1]=1;

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

inv[i]=(long long)(mod-mod/i)\*inv[mod%i]%mod;

}

埃氏筛 **O(nlog(logn))**

const int maxn=1e7, maxp=7e5;

int primes[maxp+5], psize;

bool isprime[maxn+5];

void initPrime(void){

memset(isprime, true, sizeof(isprime));

for (int i=2; i<=maxn; i++) if (isprime[i]){

for (int j=i\*2; j<=maxn; j+=i)

isprime[j]=false;

primes[psize++]=i;

}

}

欧拉筛 O(n)

const int maxn=1e5+20;

int primes[maxn/10], psize;

bool isprime[maxn];

void initPrimes(void){

memset(isprime, true, sizeof(isprime));

isprime[0]=isprime[1]=false;

for (int i=2; i<=maxn; i++){

if(isprime[i]) primes[psize++]=i;

for (int j=0; j<psize && i\*primes[j]<=maxn; j++){

isprime[primes[j]\*i]=false;

if (i%primes[j]==0) break;

}

}

}

质因数分解

利用素数表 **O(sqrt(n)/logn) + O(n)**

const int maxn=1e5+20;

int factors[100][2], fsize, primes[maxn/10], psize;

void getFactors(long long n){

fsize=0;

for (int i=0; i<psize && primes[i]<=n/primes[i]; i++){

if (n%primes[i]==0){

factors[fsize][0]=primes[i];

factors[fsize][1]=0;

while (n%primes[i]==0) factors[fsize][1]++, n/=primes[i];

fsize++;

}

}

if (n>1){

factors[fsize][0]=n;

factors[fsize++][1]=1;

}

}

不用素数表 O(sqrt(n))

int factors[100][2], fsize;

void getFactors(long long n){

fsize=0;

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

if (n%i==0){

factors[fsize][0]=i;

factors[fsize][1]=0;

while (n%i==0) factors[fsize][1]++, n/=i

fsize++;

}

}

if (n>1){

factors[fsize][0]=n;

factors[fsize++][1]=1;

}

}

### **最大公因数&最小公倍数**

// gcd(a, b, c)==gcd(gcd(a, b), c)

// lcm(a, b, c)==lcm(lcm(a, b), c)

long long gcd(long long a, long long b){

return (b==0)?a:gcd(b, a%b);

}

long long exgcd(long long a, long long b, long long &x, long long &y){

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

long long gcd=exgcd(b, a%b, y, x);

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

return gcd;

}

**分块打表**

**const int maxn=1e6;**

double h[maxn+5];

void init(void){

h[0]=h[1]=0;

for (int i=1, ptr=1; i<=1e8; i++){

if (i%100==0) {ptr++; h[ptr]=h[ptr-1];}

h[ptr]+=1/(double)i;

}

}

double calc(int n){

double sum=0;

for (int i=(n/100)\*100; i<=n; i++)

if (i!=0) sum+=1/(double)i;

return sum+h[n/100];

}

**乘法快速幂**

**long long quickMult(long long a, long long n, long long mod){**

long long ans=0, tmp=a;

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

if ((1<<i)&n) ans=(ans+tmp)%mod;

tmp=(long long)(tmp+tmp)%mod;

}return ans;

}

long long quickPow(long long a, long long n, long long mod){

int ans=1, tmp=a;

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

if ((1<<i)&n) ans=quickMult(ans, tmp)%mod;

tmp=quickMult(tmp, tmp)%mod;

}return ans;

}

矩阵快速幂

const int maxn=20;

const long long mod=1000000007;

struct Matrix{

int r, c;

long long mat[maxn][maxn];

Matrix(int r, int c):r(r), c(c) {}

void clear(void){memset(mat, 0, sizeof(mat));}

};

Matrix operator + (Matrix a, Matrix b){

Matrix s(a.r, a.c);

for(int i = 0; i < a.r; i++)

for(int j = 0; j < a.c; j++)

s.mat[i][j]=(a.mat[i][j]+b.mat[i][j])%mod;

return s;

}

Matrix operator \* (Matrix a, Matrix b){

Matrix s(a.r, b.c);

for(int i = 0; i < a.r; i++)

for(int k = 0; k < a.c; k++)

for(int j = 0; j < b.c; j++)

s.mat[i][j]=(s.mat[i][j]+a.mat[i][k]\*b.mat[k][j])%mod;

return s;

}

Matrix pow(Matrix a, long long n){

Matrix ret(a.r, a.c), tmp(a);

for(int i = 0; i < a.r; i++)

ret.mat[i][i]=1;

while(n){

if(n&1) ret=ret\*tmp;

tmp=tmp\*tmp;

n>>=1;

}return ret;

}

解一元线性同余方程

// 求ax=b (mod m)最小解

long long solve(long long a, long long b, long long m){

long long x, y, gcd=exgcd(a, m, x, y);

if(b % d == 0) {

x = x \* (b /gcd);

x = (x%(m/gcd) + (m/gcd)) % (m/gcd);

return x;

}

return -1;

}

迭代法解同余方程组  
为什么不用中国剩余定理？迭代法不要求互素

long long a[maxn], m[maxn];

bool solve(long long &m0, long long &a0, int n){

m0 = 1; a0 = 0;

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

long long t, s, t0;

long long d = exgcd(m0, m[i], t, s);

if((a[i] - a0) % d != 0) return false;

t \*= (a[i] - a0) / d;

t0 = (t % (m[i] / d) + (m[i] / d)) % (m[i] / d);

a0 += m0 \* t0;

m0 \*= (m[i] / d);

a0 %= m0;

}

return true;

}

费马小定理p为素数，则a^p=a (mod p)  
若a, p互素，则a^(p-1)=1 (mod p)

欧拉定理  
任意正整数n，a^(phi(n)+1)=a (mod n)  
若a, n互素，a^phi(n)=1 (mod n)  
欧拉函数、莫比乌斯函数是积性函数( f(nm)=f(n)\*f(m) )

应用  
求逆元：a,n互素时，inv(a, p)==a^(phi(n)-1)  
降幂：当b>phi(n)时（即指数超大），a^b=a^( b%phi(n)+phi(n) ) (mod n)注意a, n不需要互素

分解质因数方法 O(sqrt(n))

long long eular(long long n) {

getFactors(n);

long long ret = n;

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

ret=ret/factor[i][0]\*(factor[i][0]-1);

return ret;

}

筛法

const int maxn=1e6;

int phi[maxn+5];

long long sum[maxn+5];

void initPhi(void){

memset(phi, 0, sizeof(phi));

phi[1]=1;

for (int i=2; i<=maxn; i++) if (!phi[i])

for (int j=i; j<=maxn; j+=i){

if (!phi[j]) phi[j]=j;

phi[j]=phi[j]/i\*(i-1);

}

}

毕达哥拉斯三元组x^2+y^2=z^2  
当gcd(x, y, z)=1时，称此三元组是本原的  
存在互素且奇偶性不同的正整数n, m  
x=m^2-n^2, y=2mn, z=m^2+n^2

费马大定理  
x^n+y^n=z^n在n>2时无非零整数解

佩尔方程  
x^2-Dy^2=1  
最小整数解，D<30时暴力枚举  
递推法求通解

对于 x2 - Dy2 = M，其中 M = ±1, ±2, ±4，若方程存在基本解 (x1, y1)，则有

xn = C(xn-1 - xn-2) + xn-3，

yn = C(yn-1 - yn-2) + yn-3，

当 M 为 1, 2, 4, -1, -2, -4 时，

C 分别为 2x1+1, 2x12-1, x1+1, 4x12+3, 2x12+3, x12+3，

**字符串**

### **KMP算法**

应用于单个模式串匹配，求最小循环节等  
我的观点来看KMP的fail数组  
就是值域和定义域都是串的长度，返回值是这个串能够匹配后缀的最大前缀串长度  
但是纯循环节构成的串中，这个返回值不包括第一个循环节  
比如aabaabaab  
fail[9]==6 fail[6]==3

const int maxn=1e6+20, maxm=1e4+20;

char P[maxm], T[maxn];

int fail[maxm];

void getFail(int m){

fail[0]=fail[1]=0;

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

int j=fail[i];

while (j && P[j]!=P[i]) j=fail[j];

fail[i+1]=((P[i]==P[j])?j+1:0);

}

}

int count(int n, int m){

int cnt=0;

getFail(m);

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

while (j && T[i]!=P[j]) j=fail[j];

if (P[j]==T[i]) j++;

if (j==m){

// cnt++; j=0; // 匹配成功。这里随便改，j是模式串下标，i是匹配串下标

}

}return cnt;

}

### AC自动机

const int maxn=1000+20, maxw=50+20, maxl=2000000+20;

const int ACSize=maxn\*maxw, maxitem=128; // 注意此处ACSize最大的情况不超过 总词数\*词长度，maxitem指分支数量，所有可见ASCII码个数128(in HDU)

struct ACauto{

int next[ACSize][maxitem], fail[ACSize], cnt[ACSize];

int root, total;

int newnode(void){

for(int pos=0; pos<maxitem; pos++)

next[total][pos]=-1;

cnt[total]=0; // 词尾节点的初始化，不是词尾就赋0或-1

return total++;

}

void init(void){

total=0;

root=newnode();

}

int getPos(char ch){

return ch;

}

void insert(char buf[], int idx){

int now=root;

for(int i=0; buf[i]; i++){

int pos=getPos(buf[i]);

if(next[now][pos]==-1)

next[now][pos]=newnode();

now=next[now][pos];

}

cnt[now]=idx; // 这里维护词尾节点的值

}

void build(void){ // 一般不会改GetFail算法

queue<int> que;

fail[root]=root;

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

if(next[root][i]==-1)

next[root][i]=root;

else{

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

que.push(next[root][i]);

}

while(!que.empty()){

int now=que.front(); que.pop();

for(int pos=0; pos<maxitem; pos++)

if(next[now][pos]==-1)

next[now][pos]=next[fail[now]][pos];

else{

fail[next[now][pos]]=next[fail[now]][pos];

que.push(next[now][pos]);

}

}

}

void query(char buf[], int counter[]){

int now=root;

for(int i=0; buf[i]; i++){

int pos=getPos(buf[i]);

now=next[now][pos];

for (int tmp=now; tmp!=root; tmp=fail[tmp]) if (cnt[tmp])

counter[cnt[tmp]]++; // 匹配成功。这里随便改

}

}

}AC;

// 注意使用前AC.init()，插入后AC.build()

**后缀数组**

const int N = 200005;

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

int rank[N], height[N];

char ss[N];

bool cmp(int r[], int a, int b, int l){

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

}

void da(int r[], int sa[], int n, int m){

int i, j, p, \*x = wa, \*y = wb;

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

for (i = 0; i < n; ++i) ws[x[i]=r[i]]++;

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

for (i = n-1; i >= 0; --i) sa[--ws[x[i]]] = i;

for (j = 1, p = 1; p < n; j \*= 2, m = p){

for (p = 0, i = n - j; i < n; ++i) y[p++] = i;

for (i = 0; i < n; ++i) if (sa[i] >= j) y[p++] = sa[i] - j;

for (i = 0; i < n; ++i) wv[i] = x[y[i]];

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

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

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

for (i = n-1; i >= 0; --i) sa[--ws[wv[i]]] = y[i];

for (std::swap(x, y), p = 1, x[sa[0]] = 0, i = 1; i < n; ++i)

x[sa[i]] = cmp(y, sa[i-1], sa[i], j) ? p-1 : p++;

}

}

void calheight(int r[], int sa[], int n){

int i, j, k = 0;

for (i = 1; i <= n; ++i) rank[sa[i]] = i;

for (i = 0; i < n; height[rank[i++]] = k)

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

}

struct RMQ{

int log2[N],mi[N][25];

void init(int n){

for(int i = 0; i <= n; i ++)log2[i] = (i == 0 ? -1 : log2[i >> 1] + 1);

for(int j = 1; j < log2[n]; j ++)

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

mi[i][j] = std::min(mi[i][j - 1], mi[i + (1 << j - 1)][j - 1]);

}

int query(int ql, int qr){

int k = log2[qr - ql + 1];

return std::min(mi[ql][k], mi[qr - (1 << k) + 1][k]);

}

} rmq;

int sc(int i,int k,int len){

if(k==1) return len-sa[i];

return rmq.query(i+1,i+k-1);

}

int main(){

int t,k;std::cin>>t;

while(t--)

{

int ans=0;

scanf("%d%s",&k,ss);

int len=strlen(ss);

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

s[i]=ss[i]-'a'+1;

s[len]=0;

da(s,sa,len+1,28);

calheight(s,sa,len);

height[len+1]=0;

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

rmq.mi[i][0]=height[i];

rmq.init(len);

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

ans+=std::max(sc(i,k,len)-std::max(height[i],height[i+k]),0);

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

}

return 0;

}

**计算几何**

#include <bits/stdc++.h>

#define LL long long

#define PI 3.1415926535897932384626

const double eps = 1e-6;

using namespace std;

#define Vector Point

#define ChongHe 0

#define NeiHan 1

#define NeiQie 2

#define INTERSECTING 3

#define WaiQie 4

#define XiangLi 5

int dcmp(double x) { return fabs(x) < eps ? 0 : (x < 0 ? -1 : 1); }

struct Point {

double x, y;

Point(const Point& rhs): x(rhs.x), y(rhs.y) { } //拷贝构造函数

Point(double x = 0.0, double y = 0.0): x(x), y(y) { } //构造函数

friend istream& operator >> (istream& in, Point& P) { return in >> P.x >> P.y; }

friend ostream& operator << (ostream& out, const Point& P) { return out << P.x << ' ' << P.y; }

friend Vector operator + (const Vector& A, const Vector& B) { return Vector(A.x+B.x, A.y+B.y); }

friend Vector operator - (const Point& A, const Point& B) { return Vector(A.x-B.x, A.y-B.y); }

friend Vector operator \* (const Vector& A, const double& p) { return Vector(A.x\*p, A.y\*p); }

friend Vector operator / (const Vector& A, const double& p) { return Vector(A.x/p, A.y/p); }

friend bool operator == (const Point& A, const Point& B) { return dcmp(A.x-B.x) == 0 && dcmp(A.y-B.y) == 0; }

friend bool operator < (const Point& A, const Point& B) { return A.x < B.x || (A.x == B.x && A.y < B.y); }

void in(void) { scanf("%lf%lf", &x, &y); }

void out(void) { printf("%lf %lf", x, y); }

};

struct Line {

Point P; //直线上一点

Vector dir; //方向向量(半平面交中该向量左侧表示相应的半平面)

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

Line() { } //构造函数

Line(const Line& L): P(L.P), dir(L.dir), ang(L.ang) { }

Line(const Point& P, const Vector& dir): P(P), dir(dir) { ang = atan2(dir.y, dir.x); }

bool operator < (const Line& L) const { //极角排序

return ang < L.ang;

}

Point point(double t) { return P + dir\*t; }

};

typedef vector<Point> Polygon;

struct Circle {

Point c; //圆心

double r; //半径

Circle() { }

Circle(const Circle& rhs): c(rhs.c), r(rhs.r) { }

Circle(const Point& c, const double& r): c(c), r(r) { }

Point point(double ang) const { return Point(c.x + cos(ang)\*r, c.y + sin(ang)\*r); } //圆心角所对应的点

double area(void) const { return PI \* r \* r; }

};

double Dot(const Vector& A, const Vector& B) { return A.x\*B.x + A.y\*B.y; } //点积

double Length(const Vector& A){ return sqrt(Dot(A, A)); }

double Angle(const Vector& A, const Vector& B) { return acos(Dot(A, B)/Length(A)/Length(B)); } //向量夹角

double Cross(const Vector& A, const Vector& B) { return A.x\*B.y - A.y\*B.x; } //叉积

double Area(const Point& A, const Point& B, const Point& C) { return fabs(Cross(B-A, C-A)); }

//三边构成三角形的判定

bool check\_length(double a, double b, double c) {

return dcmp(a+b-c) > 0 && dcmp(fabs(a-b)-c) < 0;

}

bool isTriangle(double a, double b, double c) {

return check\_length(a, b, c) && check\_length(a, c, b) && check\_length(b, c, a);

}

//平行四边形的判定（保证四边形顶点按顺序给出）

bool isParallelogram(Polygon p) {

if (dcmp(Length(p[0]-p[1]) - Length(p[2]-p[3])) || dcmp(Length(p[0]-p[3]) - Length(p[2]-p[1]))) return false;

Line a = Line(p[0], p[1]-p[0]);

Line b = Line(p[1], p[2]-p[1]);

Line c = Line(p[3], p[2]-p[3]);

Line d = Line(p[0], p[3]-p[0]);

return dcmp(a.ang - c.ang) == 0 && dcmp(b.ang - d.ang) == 0;

}

//梯形的判定

bool isTrapezium(Polygon p) {

Line a = Line(p[0], p[1]-p[0]);

Line b = Line(p[1], p[2]-p[1]);

Line c = Line(p[3], p[2]-p[3]);

Line d = Line(p[0], p[3]-p[0]);

return (dcmp(a.ang - c.ang) == 0 && dcmp(b.ang - d.ang)) || (dcmp(a.ang - c.ang) && dcmp(b.ang - d.ang) == 0);

}

//菱形的判定

bool isRhombus(Polygon p) {

if (!isParallelogram(p)) return false;

return dcmp(Length(p[1]-p[0]) - Length(p[2]-p[1])) == 0;

}

//矩形的判定

bool isRectangle(Polygon p) {

if (!isParallelogram(p)) return false;

return dcmp(Length(p[2]-p[0]) - Length(p[3]-p[1])) == 0;

}

//正方形的判定

bool isSquare(Polygon p) {

return isRectangle(p) && isRhombus(p);

}

//三点共线的判定

bool isCollinear(Point A, Point B, Point C) {

return dcmp(Cross(B-A, C-B)) == 0;

}

//向量绕起点旋转

Vector Rotate(const Vector& A, const double& rad) { return Vector(A.x\*cos(rad)-A.y\*sin(rad), A.x\*sin(rad)+A.y\*cos(rad)); }

//向量的单位法线(调用前请确保A 不是零向量)

Vector Normal(const Vector& A) {

double len = Length(A);

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

}

//两直线交点(用前确保两直线有唯一交点，当且仅当Cross(A.dir, B.dir)非0)

Point GetLineIntersection(const Line& A, const Line& B) {

Vector u = A.P - B.P;

double t = Cross(B.dir, u) / Cross(A.dir, B.dir);

return A.P + A.dir\*t;

}

//点到直线距离

double DistanceToLine(const Point& P, const Line& L) {

Vector v1 = L.dir, v2 = P - L.P;

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

}

//点到线段距离

double DistanceToSegment(const Point& P, const Point& A, const 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 fabs(Cross(v1, v2)) / Length(v1);

}

//点在直线上的投影

Point GetLineProjection(const Point& P, const Line& L) { return L.P + L.dir\*(Dot(L.dir, P - L.P)/Dot(L.dir, L.dir)); }

//点在线段上的判定

bool isOnSegment(const Point& P, const Point& A, const Point& B) {

//若允许点与端点重合，可关闭下面的注释

//if (P == A || P == B) return true;

// return dcmp(Cross(A-P, B-P)) == 0 && dcmp(Dot(A-P, B-P)) < 0;

return dcmp(Length(P-A) + Length(B-P) - Length(A-B)) == 0;

}

//线段相交判定

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

//若允许在端点处相交，可适当关闭下面的注释

//if (isOnSegment(a1, b1, b2) || isOnSegment(a2, b1, b2) || isOnSegment(b1, a1, a2) || isOnSegment(b2, a1, a2)) return true;

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);

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

}

//多边形的有向面积

double PolygonArea(Polygon po) {

int n = po.size();

double area = 0.0;

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

area += Cross(po[i]-po[0], po[i+1]-po[0]);

}

return area \* 0.5;

}

//点在多边形内的判定(多边形顶点需按逆时针排列)

bool isInPolygon(const Point& p, const Polygon& poly) {

int n = poly.size();

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

//若允许点在多边形边上，可关闭下行注释

// if (isOnSegment(p, poly[(i+1)%n], poly[i])) return true;

if (Cross(poly[(i+1)%n]-poly[i], p-poly[i]) < 0) return false;

}

return true;

}

//过定点作圆的切线

int getTangents(const Point& P, const Circle& C, std::vector<Line>& L) {

Vector u = C.c - P;

double dis = Length(u);

if (dcmp(dis - C.r) < 0) return 0;

if (dcmp(dis - C.r) == 0) {

L.push\_back(Line(P, Rotate(u, PI / 2.0)));

return 1;

}

double ang = asin(C.r / dis);

L.push\_back(Line(P, Rotate(u, ang)));

L.push\_back(Line(P, Rotate(u, -ang)));

return 2;

}

//直线和圆的交点

int GetLineCircleIntersection(Line& L, const Circle& C, vector<Point>& sol) {

double t1, t2;

double a = L.dir.x, b = L.P.x - C.c.x, c = L.dir.y, d = L.P.y - C.c.y;

double e = a\*a + c\*c, f = 2.0\*(a\*b + c\*d), g = b\*b + d\*d - C.r\*C.r;

double delta = f\*f - 4\*e\*g; //判别式

if (dcmp(delta) < 0) return 0; //相离

if (dcmp(delta) == 0) { //相切

t1 = t2 = -f / (2 \* e);

sol.push\_back(L.point(t1));

return 1;

}

t1 = (-f - sqrt(delta)) / (2.0 \* e); sol.push\_back(L.point(t1)); // 相交

t2 = (-f + sqrt(delta)) / (2.0 \* e); sol.push\_back(L.point(t2));

return 2;

}

//两圆位置关系判定

int GetCircleLocationRelation(const Circle& A, const Circle& B) {

double d = Length(A.c-B.c);

double sum = A.r + B.r;

double sub = fabs(A.r - B.r);

if (dcmp(d) == 0) return dcmp(sub) != 0;

if (dcmp(d - sum) > 0) return XiangLi;

if (dcmp(d - sum) == 0) return WaiQie;

if (dcmp(d - sub) > 0 && dcmp(d - sum) < 0) return INTERSECTING;

if (dcmp(d - sub) == 0) return NeiQie;

if (dcmp(d - sub) < 0) return NeiHan;

return 0;

}

//两圆相交的面积

double GetCircleIntersectionArea(const Circle& A, const Circle& B) {

int rel = GetCircleLocationRelation(A, B);

if (rel < INTERSECTING) return min(A.area(), B.area());

if (rel > INTERSECTING) return 0;

double dis = Length(A.c - B.c);

double ang1 = acos((A.r\*A.r + dis\*dis - B.r\*B.r) / (2.0\*A.r\*dis));

double ang2 = acos((B.r\*B.r + dis\*dis - A.r\*A.r) / (2.0\*B.r\*dis));

return ang1\*A.r\*A.r + ang2\*B.r\*B.r - A.r\*dis\*sin(ang1);

}

//凸包(Andrew算法)

//如果不希望在凸包的边上有输入点，把两个 <= 改成 <

//如果不介意点集被修改，可以改成传递引用

Polygon ConvexHull(vector<Point> p) {

//预处理，删除重复点

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

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

int n = p.size(), m = 0;

Polygon res(n+1);

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

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

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

}

int k = m;

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

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

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

}

m -= n > 1;

res.resize(m);

return res;

}

//点P在有向直线L左边的判定(线上不算)

bool isOnLeft(const Line& L, const Point& P) {

return Cross(L.dir, P-L.P) > 0;

}

//半平面交主过程

//如果不介意点集被修改，可以改成传递引用

Polygon HalfPlaneIntersection(vector<Line> L) {

int n = L.size();

int head, rear; //双端队列的第一个元素和最后一个元素的下标

vector<Point> p(n); //p[i]为q[i]和q[i+1]的交点

vector<Line> q(n); //双端队列

Polygon ans;

sort(L.begin(), L.end()); //按极角排序

q[head=rear=0] = L[0]; //双端队列初始化为只有一个半平面L[0]

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

while(head < rear && !isOnLeft(L[i], p[rear-1])) rear--;

while(head < rear && !isOnLeft(L[i], p[head])) head++;

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

if (fabs(Cross(q[rear].dir, q[rear-1].dir)) < eps) { //两向量平行且同向，取内侧的一个

rear--;

if (isOnLeft(q[rear], L[i].P)) q[rear] = L[i];

}

if (head < rear) p[rear-1] = GetLineIntersection(q[rear-1], q[rear]);

}

while(head < rear && !isOnLeft(q[head], p[rear-1])) rear--; //删除无用平面

if (rear - head <= 1) return ans; //空集

p[rear] = GetLineIntersection(q[rear], q[head]); //计算首尾两个半平面的交点

for(int i = head; i <= rear; i++) { //从deque复制到输出中

ans.push\_back(p[i]);

}

return ans;

}

**DP**

**数位DP**

long long data[maxn][2];

char num[maxn];

long long dp(int pos, int pre, bool lim){

if (pos==-1) return 1;

if (!lim && data[pos][pre==4]>0)

return data[pos][pre==4];

long long ans=0;

int up=lim?(num[pos]-'0'):9;

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

if (pre==4 && i==9) continue;// ans+=pow10[pos];

ans+=dp(pos-1, i, lim && i==up);

}

if (!lim) data[pos][pre==4]=ans;

return ans;

}

long long solve(long long n){

sprintf(num, "%lld", n);

int len=strlen(num);

reverse(num, num+len);

return n-dp(len-1, -1, true)+1;

}

**多重背包 二进制优化O(V∑logNi)**

void zeroKnap(int cost, int val){

for (int i=m; i>=cost; i--)

dp[i]=max(dp[i], dp[i-cost]+val);

}

void compKnap(int cost, int val){

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

dp[i]=max(dp[i], dp[i-cost]+val);

}

int main(void){

while (scanf("%d%d", &n, &m)==2 && n+m){

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

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

memset(dp, 0, sizeof(dp));

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

if (num[i]\*cost[i]>=m)

compKnap(cost[i], cost[i]);

else{

for (int k=1; k<num[i]; k\*=2){

zeroKnap(cost[i]\*k, cost[i]\*k);

num[i]-=k;

}

zeroKnap(cost[i]\*num[i], cost[i]\*num[i]);

}

}

int ans=0;

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

if (dp[i]==i) ans++;

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

}

return 0;

}

**最长上升子序列的长度==最长不上升子序列的个数**

int last[int(1e5)+5], size=0, ans=1; last[size++]=hei[0];

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

if (last[size-1]<hei[i]){

last[size++]=hei[i];

ans++;

}else{

int idx=lower\_bound(last, last+size, hei[i])-last; // upper

last[idx]=hei[i];

}

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

}

**杂项**

**Java高精度**

// Main.java

import java.math.BigInteger;

import java.util.Scanner;

public class Main{

public static void main(String[] args){

Scanner cin=new Scanner(System.in);

BigInteger[] num=new BigInteger[100+5];

num[1]=new BigInteger("1");

for (int i=2; i<=100; i++){

Integer tmp1=4\*i-2, tmp2=i+1;

BigInteger a=new BigInteger(tmp1.toString());

BigInteger b=new BigInteger(tmp2.toString());

num[i]=num[i-1].multiply(a).divide(b);

}

while (cin.hasNext()){

int n=cin.nextInt();

System.out.println(num[n]);

}

}

}

BigInteger a=new BigInteger("123");

BigInteger d = new BigInteger("3", 8); //n进制字符串 转 BigInteger

a=BigInteger.valueOf(10); //int 转 BigInteger

a.add(b);

a.subtract(b);

a.multiply(b);

a.divide(b);

a.mod(b); //取模a%b，b需大于0，5mod3=2 -5mod3=1

a.abs();

a.equals(b); // a==b

a.signum(); //正为1 0为0 负为-1

a.compareTo(b); //比较a>b返回1 a==b返回0 a<b返回-1

a.max(b);

a.min(b);

a.pow(n);