目录

快读快写

```
inline int read() {
  int x = 0, neg = 1; char op = getchar();
  while (!isdigit(op)) { if (op == '-') neg = -1; op = getchar(); }
  while (isdigit(op)) { x = 10 * x + op - '0'; op = getchar(); }
  return neg * x;
}
inline void print(int x) {
  if (x < 0) { putchar('-'); x = -x; }
  if (x >= 10) print(x / 10);
  putchar(x % 10 + '0');
}
```

使用 istringstream 分割 string 默认以空格作为分隔符

```
string str="1      2        3"
Istringstream temp(str);
While(temp>>str){
      cout<<str;
}
cout<<str<<endl;
```

算法纸质资料

取到某个数的二进制最低位： result=n&（-n);

取到某个数的二进制最高位：

```
n|=(n>>1);  //前 2 位变为 1
n|=(n>>2);  //前 4 位变为 1
n|=(n>>4);  //前 8 位变为 1
n|=(n>>8);  //前 16 位变为 1
n|=(n>>16); //前 32 位变为 1
n|=(n>>32); //前 64 位变为 1
//超过 int 最大位数，足够大，能够保证所有位都变为 1.
n^=(n>>1);  //右移一位异或后 n 取到最高位的大小
```

【STL 用法整合】

一、【Vector】

```
2.基本操作：
v.capacity();           //容器容量
v.size();                    //容器大小
v.push_back();          //尾部插入
v.pop_back();           //尾部删除
v.front();              //获取头部元素
v.back();                //获取尾部元素
v.begin();              //头元素的迭代器
v.end();                     //尾部元素的迭代器
```

二、【Queue】queue 为单端队列，deque 为双端队列。

```
2. 基本操作：
(1) 元素访问：
d[i];
d.front();
d.back();
d.begin();
d.end();
d.push_back();
d.push_front();
d.pop_back();
d.pop_front();
```

优先队列:

```
priority_queue<int>q;
//默认从大到小，需要重载<运算符
struct node{
	int from,to,val;
    friend bool operator<(node a, node b)    //从小到大
	{
		return a.key > b.key;
	}
};
```

三、【Stack】

```
empty() 堆栈为空则返回真
pop() 移除栈顶元素
push() 在栈顶增加元素
size() 返回栈中元素数目
top() 返回栈顶元素
```

单调栈:

一种思想，把栈中的数据按照递增或递减的顺序放置

四、【Map】

map 会以 key 的大小从小到大的顺序自动排序（内部为红黑树，与 set 相同）

```
m[key];
m.find(key)          //存在返回迭代器，不存在返回 m.end()
m.count(key);        //返回 bool，1 存在，0 不存在
m.max_size();        //求算容器最大存储量
m.size();            //容器的大小
m.begin();
m.end();
```

五、【List】 双向链表，同 queue

六、【Set】

set 存储的是一组无重复的元素，而 multiset 允许存储有重复的元素;

如果要修改某一个元素值，必须先删除原有的元素，再插入新的元素。

```
s.find(elem);   //存在返回迭代器，不存在返回 s.end()
s.count(elem); //elem 的个数，要么是 1，要么是 0，multiset 可以大于一
s.empty();        //判断容器是否为空
s.insert(elem);
s.insert(pos, elem);
s.clear();      //清除 a 中所有元素；
```

【字符串】

一、【KMP 算法】模板：

```
int Next[1000000];
void getNext(string p)
{
    Next[0] = -1;              //初始化 next[0]，相当于把前后缀相同长度的表整体
向右移一位。
    int j = 0;
    int k = -1;
    while (j < (int)p.length() - 1)
    {
        //没有匹配的或找到相等
        if (k == -1 || p[j] == p[k])j++,k++,Next[j] = k;
        else k = Next[k];//
    }
}
int KMP(string T, string p)//T 为母串，p 为子串
{
    int i = 0,j=0;
     getNext(p);
    while (i < (int)T.length() && j < (int)p.length()){
        if (j == -1 || T[i] == p[j])i++,j++;
        else j = Next[j];//相当于右移 j-next[j]位
    }
    if (j == (int)p.length())return i - j;
    return -1;//不匹配
}
```

二、【字典树】

```
//其中 MAX_NODE 是 trie 中最大能存储的节点数目，CHARSET 是字符集的大小，k 是当前 trie 中包含有多少个节点。Trie[i][j]的值是 0 表示 trie 树中 i 号节点，并没有一条连出去的边，满足边上的字符标识是字符集中第 j 个字符（从 0 开始）；trie[i][j]的值是正整数 x 表示 trie 树中 i 号节点，有一条连出去的边，满足边上的字符标识是字符集中第 j 个字符，并且这条边的终点是 x 号节点。
//color[p]=1 标记 p 点为终结点，0 则非终结点
const int maxn = 5e5 + 10;//最大节点数
int tire[maxn][26], isend[maxn];
int k = 1;
void build(string str)
{
	int p = 0;
	for (int i = 0; i < str.size(); i++)
	{
		int c = str[i] - 'a';
		if (!tire[p][c])tire[p][c] = k++;
		p = tire[p][c];
	}
	isend[p] = 1;
}
bool search(string str)
{
	int p = 0;
	for (int i = 0; i < str.size(); i++)
	{
		int c = str[i] - 'a';
		if (!tire[p][c])return false;
		p = tire[p][c];
	}
	return isend[p];
}
```

三、【最长回文串（Manacher）】

```
const int maxn=10000010;
char str[maxn];//原字符串
char tmp[maxn<<1];//转换后的字符串
int Len[maxn<<1];//每个点的最长回文子串
//转换原始串
int INIT(char *st)
{
    int i,len=strlen(st);
    tmp[0]='%';//字符串开头增加一个特殊字符，防止越界
    for(i=1;i<=2*len;i+=2)
    {
        tmp[i]='#';
        tmp[i+1]=st[i/2];
    }
    tmp[2*len+1]='#';
    tmp[2*len+2]='$';//字符串结尾加一个字符，防止越界
    tmp[2*len+3]=0;
    return 2*len+1;//返回转换字符串的长度
}
//Manacher 算法计算过程
int MANACHER(char *st,int len)
{
     int mx=0,ans=0,po=0;//mx 即为当前计算回文串最右边字符的最大值
     for(int i=1;i<=len;i++)
     {
         if(mx>i)Len[i]=min(mx-i,Len[2*po-i]);//在 Len[j]和 mx-i 中取个小
         else Len[i]=1;//如果 i>=mx，要从头开始匹配
         while(st[i-Len[i]]==st[i+Len[i]])Len[i]++;
         if(Len[i]+i>mx)//若新计算的回文串右端点位置大于 mx，要更新 po 和
mx 的值
         {
             mx=Len[i]+i;
             po=i;
             ans=max(ans,Len[i]);
         }

     }
     return ans-1;//返回 Len[i]中的最大值-1 即为原串的最长回文子串额长度
  }
```

【动态规划（dp）】

一、【数位 dp】

```
typedef long long ll;
int a[20];
ll dp[20][state];//不同题目状态不同
ll dfs(int pos,/*state 变量*/,bool lead/*前导零*/,bool limit/*数位上界变量*/)
{
	//递归边界，既然是按位枚举，最低位是 0，那么 pos==-1 说明这个数我枚举完了
	if(!pos) return 1;/*这里一般返回 1，表示你枚举的这个数是合法的
	if(!limit && !lead && dp[pos][state]!=-1) return dp[pos][state];
	int up=limit?a[pos]:9;
	ll ans=0;
	for(int i=0;i<=up;i++)
	{
		if() ...
		else if()...
		ans+=dfs(pos-1,,lead && i==0,limit && i==a[pos])
	}
	if(!limit && !lead) dp[pos][state]=ans;
	return ans;
}
ll solve(ll x)
{
	int pos=0;
	while(x)
	{
		a[++pos]=x%10;//取出低位到高位的所有,a[2]=3 表示第二位上的数字为 3
		x/=10;
	}
	return dfs(pos,true,true);
}
int main()
{
	ll le,ri;
	while(~scanf("%lld%lld",&le,&ri))
	{
		printf("%lld\n",solve(ri)-solve(le-1));
	}
```

二、【01 背包】

```
//可改为一维数组进行滚动，也要逆序
typedef long long ll;
int cost[101], val[101];
int dp[101][1010];
//dp[i][j],当容量为 j 时，放入前 i 个物品价值最大；
//dp[i][j]=max(dp[i-1][j],dp[i-1][j-cost[i]]+val[i]);
int main() {
	ios::sync_with_stdio(false); cin.tie(0); cout.tie(0);
	int t, num;
	cin >> t >> num;
	for (int i = 1; i <= num; i++)
	{
		cin >> cost[i] >> val[i];
	}
	for (int i = 1; i <= num; i++)
	{
		for (int j = t; j >= 0; j--)
		{
			if (j >= cost[i])
				dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - cost[i]] + val[i]);
			else dp[i][j] = dp[i - 1][j];
		}
	}
	cout << dp[num][t] << endl;
	return 0;
}
```

三、【完全背包】

注意事项:在变成一维进行滚动时，第二层循环要顺序，这样才能确保重复取，符合完全背包的定义。

```
for (int i = 1; i <= num; i++)
		for (int j = cost[i]; j <= t; j++)
			dp[j] = max(dp[j], dp[j - cost[i]] + val[i]);
```

五、【LCS 最长公共子序列】

```
string S,T;
int dp[maxn+1][maxn+1];
int LCS(string S,string T){
	int n=S.size(),m=T.size();
	for(int i=0;i<n;i++){
		for(int j=0;j<m;j++){
			if(S[i]==T[j])dp[i][j]=dp[i-1][j-1]+1;
			else dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
		}
	}
	return dp[n][m]
}
```

六、【LIS 最长上升子序列】

```
int n;//原序列长度
int a[maxn],dp[maxn];//dp[i]:以 i 结尾的最长上升子序列
int LIS(){
	int res=0;
	for(int i=0;i<n;i++) {
		dp[i]=1;
		for(int j=0;j<i;j++){
			if(a[j]<a[i]) dp[i]=max(dp[i],dp[j]+1);
			res=max(res,dp[i]);
		}
	}
	return res;
}
```

【数论】

一、【基本的取模运算】

(a+b)%c=(a%c+b%c)%c

(a-b)%c=(a-b+c)%c

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

(a/b)%c=(a*b-1)%C

二、【GCD(a,b)、LCM(a,b)】

```
int GCD(int a,int b){
	if(b==0)return a;
	return GCD(b,a%b);
}
int LCM(int a,int b){
	int t=GCD(a,b);
	return a*b/t;
}
```

三、【扩展欧几里得算法】

```
int extgcd(int a,int b,int &x,int &y)
{
	if(b==0)
	{
		x=1;
		y=0;
		return a;
	}
	int gcd=exgcd(b,a%b,x,y);
	int temp=x;
	x=y;
	y=temp-a/b*y;
	return gcd;
}
```

乘法逆元定义：(a*x)%c=1%c

salution 1：扩展欧几里得算法实现（条件弱，无限制）

```
LL cal(LL a,LL b,LL p)
{
	LL x,y;
	LL gcd=extgcd(a,b,x,y);
	if(p%gcd!=0) return -1;    //gcd(a,b)=1 时才有解
	x*=p/gcd;    b/=gcd;
	if(b<0) b=-b;
	LL ans=x%b;
	if(ans<=0) ans+=b;
	return ans;
}
```

salution 2:费马小定理实现(条件强，要求 c 为质数)

-> a*x=1 (mod c)

-> a*ac-2=1 (mod c)

则 ac-2 即为所要求的的 x,使用快速幂可快速得到解。

四、【快速幂、快速乘】

快速幂

```
ll ksm(ll base, ll power) {
	ll result = 1;
	while (power > 0) {
		if (power & 1) {//此处等价于 if(power%2==1)
			result = result * base % 1000;
		}
		power >>= 1;//此处等价于 power=power/2
		base = (base * base) % 1000;
	}
	return result;
}
```

快速乘

```
ll ksc(ll x,ll y,ll mod){
	return (x*y-(ll)((long double)x/mod*y)*mod+mod)%mod;
}
```

五、【欧拉筛】

```
int prime[maxn];//记录所有素数的数组  prime[0]存放范围内素数数量
bool visit[maxn];//记录所有数，并将其标记合数为 1,素数为 0
void Prime(){
	memset(visit,0,sizeof(visit));//初始化全为素数
	memset(prime, 0,sizeof(prime));
	for (int i = 2;i <= maxn; i++) {
		cout<<" i = "<<i<<endl;
		if (!visit[i]) {
			prime[++prime[0]] = i;
		}
  for (int j = 1; j <=prime[0] && i*prime[j] <= maxn; j++) {
			visit[i*prime[j]] = 1;//标记合数为 1
			if (i % prime[j] == 0) {
				break;
			}
		}
	}
}
```

七、【数论分块】

对于计算带有下列特征的结果可采取数论分块的思想： $\sum_{i=1}^{n} \frac{n}{i}$

```
int sum=0;
for(int l=1,r;l<=n;l=r+1){
	r=n/n/l;
	sum+=(n/l)*(r-l+1);
}
```

八、【中国剩余定理】

在《孙子算经》中有这样一个问题："今有物不知其数，三三数之剩二（除以 3 余 2），五五数之剩三（除以 5 余 3），七七数之剩二（除以 7 余 2），问物几何？

$$x \equiv 2 \pmod{3} x \equiv 3 \pmod{5} x \equiv 2 \pmod{7}$$

具体解法分三步：

> 1、找出三个数：从 3 和 5 的公倍数中找出被 7 除余 1 的最小数 15，从 3 和 7 的公倍数中找出被 5 除余 1 的最小数 21，最后从 5 和 7 的公倍数中找出除 3 余 1 的最小数 70。
>
> 2、用 15 乘以 2（2 为最终结果除以 7 的余数），用 21 乘以 3（3 为最终结果除以 5 的余数），同理，用 70 乘以 2（2 为最终结果除以 3 的余数），然后把三个乘积相加 15* 2+21* 3+70* 2 得到和 233。
>
> 3、用 233 除以 3，5，7 三个数的最小公倍数 105，得到余数 23，即 233%105=23
>
> 这个余数 23 就是符合条件的最小数。

九、【高维前缀和（1、2、3....维）】

二维：

$$DP[i][j] = DP[i][j] + DP[i][j] - DP[i-1][j-1] + num[i][j]$$

或：

```
//假设为n*m的矩阵
for(int i=1;i<=n;i++)
    for(int j=1;j<=m;j++)
        DP[i][j]+=DP[i-1][j];
for(int i=1;i<=n;i++)
    for(int j=1;j<=m;j++)
        DP[i][j]+=DP[i][j-1];
```

十、【大数乘法】

10.1【求 A^BmodC 】

A^B%C = A^(B%phi(C)+phi(C))%C

$$A^{x} = A^{(x \bmod Phi(C) + Phi(C))} \bmod C \ (x \geq Phi(C))$$

```
#include<bits/stdc++.h>
#define put putchar('\n')
#define re register
using namespace std;
typedef long long ll;
const int maxn =207;
const int mod = 998244353;
const int inf = 1e9 + 7;
inline int read() {
	char c = getchar(); int tot = 1; while ((c < '0' || c>'9') && c != '-') c = getchar(); if (c == '-') { tot = -1; c = getchar(); }
	int sum = 0; while (c >= '0' && c <= '9') { sum = sum * 10 + c - '0'; c = getchar(); }return sum * tot;
}
inline void wr(int x) { if (x < 0) { putchar('-'); wr(-x); return; }if (x >= 10)wr(x / 10); putchar(x % 10 + '0'); }
inline void wrn(int x) { wr(x); put; }
inline void wri(int x) { wr(x); putchar(' '); }
int arr[maxn];
int te[maxn];
ll	phi(ll n) {
	ll s = n;
	for (ll i = 2; i * i <= n; i++) {
		if (n % i == 0)	s = s / i * (i - 1);
		while (n % i == 0) n /= i;
	}
	if (n != 1)	s = s / n * (n - 1);
	return s;
}
```

```
ll    Pow(ll x, ll y, ll m) {
        ll s = 1;
        for (; y; y >>= 1) {
                if (y & 1) { s *= x; s %= m; }
                x *= x; x %= m;
        }
        return s;
}

ll      Pow(ll x, char* y, ll m) {
        ll phim = phi(m);
        ll s = 0;
        for (int i = 0; y[i] != '\0'; i++) {
                s = s * 10 + y[i] - '0';
                if (s >= m) break;
        }
        if (s >= m) {
                s = 0;
                for (int i = 0; y[i] != '\0'; i++) {
                        s = s * 10 + y[i] - '0';
                        if (s >= phim) s %= phim;
                }
                s += phim;
                return Pow(x, s, m);
        }
        else return Pow(x, s, m);
}

char y[1000008];

int main()
{
        ios::sync_with_stdio(false); cin.tie(0); cout.tie(0);
        int t;
        cin >> t;
        while (t--) {
                ll x, m;
                cin >> x >> y >> m;
                cout << Pow(x, y, m) << endl;

        }
        return 0;
}
```

十一、【组合数学】

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
const int maxn=3e6+10;
const int mod=1e9+7;
int inv[maxn],fac[maxn];
int C(int n,int m){return fac[n]*inv[n-m]%mod*inv[m]%mod;}
int A(int n,int m){return fac[n]*inv[n-m]%mod;}
int ksm(int x,int k){
	int res=1;
	while(k){
		if(k&1)res=res*x%mod;
		x=x*x%mod;
		k/=2;
	}
	return res;
}
int ny(int x){
	return ksm(x,mod-2);
}
void init(){
	inv[0]=fac[0]=1;
	inv[1]=1;
	for(int i=1;i<maxn;i++){
		fac[i]=fac[i-1]*i%mod;
	}
	//求每个数的逆元推导公式：i 逆元≡-⌊ p/i⌋ * (p mod i)的逆元(mod p)
	inv[1]=1;
	for(int i=2;i<maxn;i++){
		inv[i]=(int)(mod-mod/i)*inv[mod%i]%mod;
	}
	//求阶乘的逆元
	inv[0]=1;
	for(int i=1;i<maxn;i++){
		inv[i]=inv[i-1]*inv[i]%mod;
	}
}
```

【图论】

一、【前向星式邻接表】

```
const int MAX = 1e6;      //数组大小初始化
struct Edges {
    int to, w, next; //（边的终点编号、权重、同起点的下一条边）
};
Edges edge[MAX];    //记录所有点的信息
ll head[MAX];         //例：head[1]=5;记录从 1 点出发的最后输入的一条边 5
ll cnt;                    //记录每组数据的编号
void init()              //初始化
{
    memset(head, -1, sizeof(head));
    cnt = 0;
}
void add(ll from, ll to, ll w) //添加边
{
    edge[cnt].to = to;
    edge[cnt].w = w;
    edge[cnt].next = head[from];
    head[from] = cnt++;
}
void visit(int from)
{
     for(int i=head[from];i!=-1;i=edge[i].next)
     {
          //遍历以 from 为起点的所有边
          //cout<<edge[i].to<<endl;
          //cout<<edge[i].w<<endl;
     }
}
```

二、【最短路问题】

1、SPFA 算法

适用范围：单源最短路问题，可判断是否存在负环

```
bool vis[]                              //标记是否在队列内
int dis[]                               //起点到各点的最短路
int c[]             //判断各个点更新次数，用来判断是否存在负环，当 c[i]>n 时存在负环
bool flag=0;                            //标记是否存在负环，0 为不存在
void spfa(int start，int n)             //start 是源点,n 为顶点数，用于判断负环
{
    memset(vis, false, sizeof(vis));
    memset(dis, inf, sizeof(dis));      //距离源点的距离初始化成 inf(很大的数)
    queue<int>q;
    dis[start] = 0;                     //源点的距离默认成 0
    vis[start] = true;                  //标记源点
    q.push(start);                      //放入队列
    while (!q.empty())
    {
        int u = q.front();                  //取出下一个节点
        q.pop();
        vis[u] = false;                     //标记为不在队列内
        for (int i = head[u]; i != -1; i = edge[i].next)
        {
            int v = edge[i].v;
            int w = edge[i].w;
            if (dis[v] > dis[u] + w)        //dis[v]为当前与 u 点相连的节点的最短路径（初
始为 INF）
            {
                dis[v] = dis[u] + w;
                //c[v]++;
               // if(c[v]>n)flag=1 return;//存在负环则返回，flag 标记为 1
                if (!vis[v])            //如果改点不在队列内，则标记并加入队列
                {
                    vis[v] = true;
                    q.push(v);
                }
            }
        }
    }
}
```

1、Dijkstra 算法

```
#define max 110    //最大顶点个数
#define INF 0xfffff       //权值上限
int n;    //顶点数
struct node         //顶点节点
{
	int vlue,key,parent;
	//按升序排列，以小为先
	friend bool operator<(node a, node b) {
		return a.key > b.key;
	}
};
void Dijkstra(int **G, int s) {
	priority_queue<node> q; //最小优先队列
	node d[max];           //结点集
	bool visited[max]; //用于判断顶点是否已经在最短路径树中
	for (int i = 1; i <= n; i++) {
		d[i].vlue = i;
		d[i].key = INF;                    //估算距离置 INF
		d[i].parent = -1;                //每个顶点都无父亲节点
		visited[i] = false;         //都未找到最短路
	}
	d[s].key = 0;                               //源点到源点最短路权值为 0
	q.push(d[s]);                           //压入队列中
	while (!q.empty()){
		node cd = q.top();           //取最小顶点
		q.pop();
		int u = cd.vlue;
		if (visited[u])
			continue;
		visited[u] = true;
		for (int i = 1; i < n;i++) {
			if (!visited[i] &&d[i].key>d[u].key+G[d[u].vlue][i])
			{
				d[i].parent = u;
				d[i].key = d[u].key + G[d[u].vlue][i];
				q.push(d[i]);
			}
		}
	}
}
```

三、【最小生成树】

1、【Kruskal 算法】适用范围：稀疏图

```
using namespace std;
typedef long long ll;
struct Edges {
	ll from,to, w;
};
Edges edge[2000005];
ll   cnt,n, m,ans,e1,e2;//n 为顶点个数，m 为边的个数
ll fa[5005];//并查集需要
//sort 结构体权值从小到大
bool cmp(Edges a, Edges b)
{
	return a.w < b.w;
}
//并查集
ll find_set(ll x)
{
	if (x != fa[x])return fa[x] = find_set(fa[x]);
	return x;
}
void Kruskal()
{
	sort(edge, edge + m, cmp);
	for (ll i = 0; i < m; i++)
	{
		 e1 = find_set(edge[i].from);
		 e2 = find_set(edge[i].to);
		if (e1 == e2)continue;
		fa[e2] = e1;
		ans += edge[i].w;
		cnt++;//记录边数
		if (cnt == n - 1) break;//边数为顶点数-1 时退出
	}
}
```

2、【Prim 算法】适用范围：稠密图

```
il int prim()
{
	for (re int i = 0; i <= n; i++)
	{
		dis[i] = inf;
	}
	for (re int i = head[1]; i!=-1; i = e[i].next)
	{
		dis[e[i].to] = min(dis[e[i].to], e[i].w);
	}
	while (tot != n-1)//最小生成树边数等于点数-1
	{
		re int minn = inf;//把 minn 置为极大值
		vis[now] = 1;//标记点已经走过
		for (re int i = 0; i <= n; i++)
		{
			if (!vis[i] && minn > dis[i])
			{
				minn = dis[i];
				now = i;
			}
		}
		ans += minn;
		//枚举 now 的所有连边，更新 dis 数组
		for (re int i = head[now]; i!=-1; i = e[i].next)
		{
			re int to = e[i].to;
			if (dis[to] > e[i].w && !vis[to])
			{
				dis[to] = e[i].w;
			}
		}
		tot++;
	}
	return ans;
}
```

四、【拓扑排序】

```
const int MAX = 1e6;      //数组大小初始化
struct Edges {
      int to, w, next; //（边的终点编号、权重、同起点的下一条边）
};
Edges edge[MAX];   //记录所有点的信息
ll head[MAX];        //例：head[1]=5;记录从 1 点出发的最后输入的一条边 5
queue<int>q;
int in[maxn];//入度
vector<int>ans;//存放结果
void solve(){
       for(int i=0;i<n;i++)if(!in[i])q.push(i);
       while(!q.empty()){
               int from=q.front();q.pop();
               ans.push_back(from)
               for(int i=head[from];i!=-1;i=edge[i].next)
               {
                      int point=edge[i].to;
                      in[point]--;
                      if(!in[point])
                             q.push(point);
               }
       }
}
```

五、【差分约束系统】

差分约束系统的解法如下：

1、 根据条件把题意通过变量组表达出来得到不等式组，注意要发掘出隐含的不等式，比如说前后两个变量之间隐含的不等式关系。

2、 进行建图：

首先根据题目的要求进行不等式组的标准化。

(1)、如果要求取最小值，那么求出最长路，那么将不等式全部化成 xi – xj >= k 的形式，这样建立 j->i 的边，权值为 k 的边，如果不等式组中有 xi – xj > k，因为一般题目都是对整形变量的约束，化为 xi – xj >= k+1 即可，如果 xi – xj = k 呢，那么可以变为如下两个：xi – xj >= k, xi – xj <= k,进一步变为 xj – xi >= -k，建立两条边即可。

(2)、如果求取的是最大值，那么求取最短路，将不等式全部化成 xi – xj <= k 的形式, 这样建立 j->i 的边，权值为 k 的边，如果像上面的两种情况，那么同样地标准化就行了。

(3)、如果要判断差分约束系统是否存在解，一般都是判断环，选择求最短路或者最长路求解都行，只是不等式标准化时候不同，判环地话，用 spfa 即可，n 个点中如果同一个点入队超过 n 次，那么即存在环。

值得注意的一点是：建立的图可能不联通，我们只需要加入一个超级源点，比如说求取最长路时图不联通的话，我们只需要加入一个点 S，对其他的每个点建立一条权值为 0 的边图就联通了，然后从 S 点开始进行 spfa 判环。最短路类似。

六、【TarJan】

无向图：

```
//核心模板
//链式前向星加边
//由于是无向图，加边时要加两条边 from->to,to->from
int head[maxn],dfn[maxn], low[maxn];
int ans,d;//存放割点答案数量
bool cut[maxn];
void tarjan(int from, int fa) {
	dfn[from] = low[from] = ++d;
	int child = 0;
	for (int i = head[from]; i != -1; i = edge[i].next){
		int to = edge[i].to;
		if (!dfn[to]){
			tarjan(to, fa);
			low[from] = min(low[from], low[to]);
			if (low[to] >= dfn[from] && from != fa) cut[from] = 1;
			if (from == fa)child++;
		}
		low[from] = min(low[from], dfn[to]);
	}
		if (from == fa && child >= 2)cut[from] = 1;
}
//由于 tarjan 图各个连通块之间可能不连通，主函数内要这样写
for (int i = 1; i <= n; i++)
		if (!dfn[i])
			tarjan(i, i);
```

有向图：

```
int color[maxn],dfn[maxn], low[maxn],stack_[maxn];
int num[maxn];//同属于同一颜色连通块的节点个数
bool exist[maxn];
int d,id,top;
void tarjan(int x) {
    dfn[x] = low[x] = ++d;
    stack_[++top]=x;
     exist[x]=1;
    for (int i = head[x]; i != -1; i = edge[i].next){
        int to = edge[i].to;
        if (!dfn[to]){
            tarjan(to);
            low[x] = min(low[x], low[to]);
        }
          else if(exist[to]){
            low[x] = min(low[x], low[to]);
          }
    }
        if(low[x]==dfn[x])
          {
            id++;
            do{
                color[stack_[top]]=id;
                  num[id]++;
                  exist[stack_[top]]=0;
            }while(x!=stack_[top--]);
          }
}
```

七、【网络流】dini 算法

```
const int maxn = 2050;
const int INF = 1e9 + 7;
struct Edges {
	int to, w, next;
};
Edges edge[maxn<<1];
int head[maxn], deep[maxn];
int cnt;
ll n, m;//n 为汇点，源点默认为 1，m 是边的数量
void add(int from, int to,int w) {
	edge[cnt].to = to;
	edge[cnt].w = w;
	edge[cnt].next = head[from];
	head[from] = cnt++;
}
void init()
{
	memset(head, -1, sizeof(head));
	cnt = 0;
}
bool bfs()
{
	memset(deep, -1, sizeof(deep));
	deep[1] = 0;
	queue<int>q;
	q.push(1);
	while (!q.empty())
	{
		int from = q.front();
		q.pop();
		for (int i = head[from]; i != -1; i = edge[i].next)
		{
			int to = edge[i].to;
			if (deep[to] == -1 && edge[i].w)//子节点没被搜索过并且子节点的剩余流量不为 0。
			{
				deep[to] = deep[from] + 1;
				q.push(to);
			}
		}
	}
	return deep[n] != -1;
}
```

```
int dfs(int from, int rest)
{

	if (from == n)return rest;
	int restnow = rest;
	for (int i = head[from]; i != -1; i = edge[i].next)
	{
		int to = edge[i].to;
		if (deep[to] == (deep[from] + 1) && edge[i].w > 0)
		{
			int allow = dfs(to, min(restnow, edge[i].w));
			edge[i].w -= allow, edge[i ^ 1].w += allow;
			restnow -= allow;
			if (restnow == 0)break;
		}
	}
	return rest - restnow;
}
int dini()
{
	int ans = 0;
	while (bfs())ans += dfs(1, INF);
	return ans;
}
int main() {
	ios::sync_with_stdio(false); cin.tie(0); cout.tie(0);
	init();
	int from, to, w;
	cin >> n >> m >> x;
	for (int i = 0; i < m; i++)
	{
		cin >> from >> to >> w;
		add(from, to, w), add(to, from, 0);
	}
	int ans = dini();
	cout<<ans<<endl;
	return 0;
}
```

八、【并查集】

```
//并查集_未优化
ll find_set(ll x)
{
	return x!=fa[x]?find_set(fa[x]):x;
}

//并查集_路径压缩优化
ll find_set(ll x)
{
	if (x != fa[x])return fa[x] = find_set(fa[x]);
	return x;
}

//并查集_启发式合并
//rank[maxn]代表深度，把深度小的合并到深度大的上
void merge(int u,int v){
	int t1=find(u);
	int t2=find(v);
	if(t1!=t2) {//启发式算法的核心代码，将节点深度小的并到节点深度大的上面。
		if(rank[t1]>rank[t2]){
			f[t2]=t1;
		}
		else if(rank[t1]<rank[t2]){
			f[t1]=t2;
		}
		else {//深度相等时将其中一个的深度增加
			f[t2]=t1;
			rank[t1]++;
		}
	}

}
```

【树】

一、【线段树】

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int maxn = 1e6;
const int inf = 1e9 + 5;
#define ls p<<1
#define rs p<<1|1
struct node {
	int l, r,tmax, cnt, cd, lz;
	ll sum;
}T[maxn << 2];
void pushup(int p) {
	T[p].sum = T[ls].sum + T[rs].sum;
	T[p].tmax = max(T[ls].tmax,T[rs].tmax);
	if (T[ls].tmax > T[rs].tmax) {
		T[p].cnt = T[ls].cnt;
		T[p].cd = max(T[ls].cd, T[rs].tmax);
	}
	else if (T[ls].tmax == T[rs].tmax) {
		T[p].cnt = T[rs].cnt + T[ls].cnt;
		T[p].cd = max(T[rs].cd, T[ls].cd);
	}
	else {
		T[p].cnt = T[rs].cnt;
		T[p].cd = max(T[ls].tmax, T[rs].cd);
	}
}
int a[maxn];
void build(int p, int l, int r) {
	T[p].l = l; T[p].r = r;
	T[p].cd = T[p].lz = -1;
	if (l == r) {
		T[p].tmax = T[p].sum=a[l];
		T[p].cnt = 1;
		return;
	}
```

```
  int mid = (l + r) >> 1;
	build(ls, l, mid);
	build(rs, mid + 1, r);
	pushup(p);
}
void pushdown(int p) {
	if (T[p].lz != -1) {
		if (T[p].lz < T[ls].tmax) {
			T[ls].sum += 1ll * (T[p].lz - T[ls].tmax) * T[ls].cnt;
			T[ls].tmax = T[ls].lz = T[p].lz;
		}
		if (T[p].lz < T[rs].tmax) {
			T[rs].sum += 1ll * (T[p].lz - T[rs].tmax) * T[rs].cnt;
			T[rs].tmax = T[rs].lz = T[p].lz;
		}
		T[p].lz = -1;
	}
}
void update(int p, int l, int r, int v) {
	if (v >= T[p].tmax) return;
	if (l <= T[p].l && T[p].r <= r) {
		if (v > T[p].cd) {
			T[p].lz = v;
			T[p].sum += 1ll*(v - T[p].tmax) * T[p].cnt;
			T[p].tmax = v;
			return;
		}
	}
	pushdown(p);
	int mid = (T[p].l + T[p].r) >> 1;
	if (l <= mid) update(ls,l,r,v);
	if (r > mid) update(rs,l,r,v);
	pushup(p);
}
int q1(int p, int l, int r) {
	if (T[p].l >= l && T[p].r <= r) return T[p].tmax;
	pushdown(p);
	int mid = (T[p].l + T[p].r) >> 1;
	int ans = 0;
	if (l <= mid) ans = max(ans, q1(ls, l, r));
	if (r > mid) ans = max(ans, q1(rs, l, r));
	return ans;
}
```

```
ll q2(int p, int l, int r) {
	if (T[p].l >= l && T[p].r <= r) return T[p].sum;
	pushdown(p);
	int mid = (T[p].l + T[p].r) >> 1;
	ll ans = 0;
	if (l <= mid) ans += q2(ls, l, r);
	if (r > mid) ans += q2(rs, l, r);
	return ans;
}
int main() {
	int t;
	scanf("%d", &t);
	while (t--) {
		int n, m;
		scanf("%d%d", &n, &m);
		for(int i=1;i<=n;++i) scanf("%d",a+i);
		build(1, 1, n);
		while(m--){
			int op, x, y, z;
			scanf("%d%d%d",&op,&x,&y);
			if (op == 0) {
				scanf("%d", &z);
				update(1, x, y, z);
			}
			else if (op == 1) printf("%d\n", q1(1, x, y));
			else printf("%lld\n", q2(1, x, y));
		}
	}
	return 0;
}
```

二、树链剖分+边权

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int maxn = 2e5+15;
const int inf = 2e9+5;
int n,m;
struct edge{
	int v,w,ne;
}e[maxn<<1];
int h[maxn],cnt=0;
void add(int u,int v,int w){
	e[++cnt]={v,w,h[u]};h[u]=cnt;
}
int num=0;
int deep[maxn],tot[maxn],fa[maxn],son[maxn],top[maxn],idx[maxn];
int tmp[maxn],a[maxn];
void dfs1(int u,int f){
	deep[u]=deep[f]+1;
	fa[u]=f;
	tot[u]=1;
	for(int i=h[u];i;i=e[i].ne){
		int v=e[i].v;
		if(v==fa[u]) continue;
		dfs1(v,u);
		tmp[v]=e[i].w;
		tot[u]+=tot[v];
		if(tot[v]>tot[son[u]]) son[u]=v;
	}
}
void dfs2(int u,int tpf){
	top[u]=tpf;
	idx[u]=++num;
	a[num]=tmp[u];
	if(son[u]) dfs2(son[u],tpf);
	for(int i=h[u];i;i=e[i].ne){
		int v=e[i].v;
		if(v==fa[u]||v==son[u]) continue;
		dfs2(v,v);
	}
}
int sum[maxn<<2],ma[maxn<<2],mi[maxn<<2];
```

```
bool lz[maxn<<2];
struct pos{
	int x,y;
}T[maxn];
#define ls p<<1
#define rs p<<1|1
#define mid ((l+r)>>1)
void pushup(int p){
	sum[p]=sum[ls]+sum[rs];
	ma[p]=max(ma[ls],ma[rs]);
	mi[p]=min(mi[ls],mi[rs]);
}
void build(int p,int l,int r){
	if(l==r){
		sum[p]=ma[p]=mi[p]=a[l];
		return ;
	}
	build(ls,l,mid);
	build(rs,mid+1,r);
	pushup(p);
}
void qf(int p){
	lz[p]^=1;
	sum[p]*=-1;
	ma[p]*=-1;
	mi[p]*=-1;
	swap(ma[p],mi[p]);
}
void pushdown(int p){
	if(lz[p]){
		qf(ls);
		qf(rs);
		lz[p]=0;
	}
}
void update(int p,int l,int r,int q,int val){
	if(l==r){
		sum[p]=ma[p]=mi[p]=val;
		return;
	}
	pushdown(p);
	if(q<=mid) update(ls,l,mid,q,val);
	else update(rs,mid+1,r,q,val);
	pushup(p);
	}
```

```
void change(int p,int l,int r,int L,int R){
	if(L<=l&&R>=r){
		qf(p);
		return ;
	}
	pushdown(p);
	if(L<=mid) change(ls,l,mid,L,R);
	if(R>mid) change(rs,mid+1,r,L,R);
	pushup(p);
}
int qsum(int p,int l,int r,int L,int R){
	if(L<=l&&R>=r) return sum[p];
	pushdown(p);
	int ans=0;
	if(L<=mid) ans+=qsum(ls,l,mid,L,R);
	if(R>mid) ans+=qsum(rs,mid+1,r,L,R);
	return ans;
}
int qmax(int p,int l,int r,int L,int R){
	if(L<=l&&R>=r) return ma[p];
	pushdown(p);
	int ans=-inf;
	if(L<=mid) ans=max(ans,qmax(ls,l,mid,L,R));
	if(R>mid) ans=max(ans,qmax(rs,mid+1,r,L,R));
	return ans;
}
int qmin(int p,int l,int r,int L,int R){
	if(L<=l&&R>=r) return mi[p];
	pushdown(p);
	int ans=inf;
	if(L<=mid) ans=min(ans,qmin(ls,l,mid,L,R));
	if(R>mid) ans=min(ans,qmin(rs,mid+1,r,L,R));
	return ans;
}
void update(int x,int y){
	while(top[x]!=top[y]){
		if(deep[top[x]]<deep[top[y]]) swap(x,y);
		change(1,1,n,idx[top[x]],idx[x]);
		x=fa[top[x]];
	}
	if(deep[x]>deep[y]) swap(x,y);
	if(x!=y) change(1,1,n,idx[x]+1,idx[y]);
}
```

```
int qqsum(int x,int y){
	int ans=0;
	while(top[x]!=top[y]){
		if(deep[top[x]]<deep[top[y]]) swap(x,y);
		ans+=qsum(1,1,n,idx[top[x]],idx[x]);
		x=fa[top[x]];
	}
	if(deep[x]>deep[y]) swap(x,y);
	ans+=qsum(1,1,n,idx[x]+1,idx[y]);
	return ans;
}
int qqmax(int x,int y){
	int ans=-inf;
	while(top[x]!=top[y]){
		if(deep[top[x]]<deep[top[y]]) swap(x,y);
		ans=max(ans,qmax(1,1,n,idx[top[x]],idx[x]));
		x=fa[top[x]];
	}
	if(deep[x]>deep[y]) swap(x,y);
	ans=max(ans,qmax(1,1,n,idx[x]+1,idx[y]));
	return ans;
}
int qqmin(int x,int y){
	int ans=inf;
	while(top[x]!=top[y]){
		if(deep[top[x]]<deep[top[y]]) swap(x,y);
		ans=min(ans,qmin(1,1,n,idx[top[x]],idx[x]));
		x=fa[top[x]];
	}
	if(deep[x]>deep[y]) swap(x,y);
	ans=min(ans,qmin(1,1,n,idx[x]+1,idx[y]));
	return ans;
}
int lca(int x,int y){
	while(top[x]!=top[y]){
		if(deep[top[x]]<deep[top[y]]) swap(x,y);
		x=fa[top[x]];
	}
	if (deep[x]<deep[y])swap(x,y);
	return y;
}
int main(){
	scanf("%d",&n);
```

```
    int u,v,w;
    for(int i=1;i<n;++i){
        scanf("%d%d%d",&u,&v,&w);
        u++,v++;
        add(u,v,w);add(v,u,w);
        T[i].x=u;T[i].y=v;
    }
    dfs1(1,0);
    dfs2(1,1);
    build(1,1,n);
    scanf("%d",&m);
    while(m--){
        char op[10];int x,y;
        scanf("%s",op);
        scanf("%d%d",&x,&y);
        if(op[0]=='S'){
            x++,y++;
            printf("%d\n",qqsum(x,y));
        }
        else if(op[0]=='N'){
            x++,y++;
            update(x,y);
        }
        else if(op[0]=='C'){
            int tmpp=(deep[T[x].x]>deep[T[x].y])?T[x].x:T[x].y;
            update(1,1,n,idx[tmpp],y);
        }
        else if(op[0]=='M'&&op[1]=='A'){
            x++,y++;
            printf("%d\n",qqmax(x,y));
        }
        else if(op[0]=='M'&&op[1]=='I'){
            x++,y++;
            printf("%d\n",qqmin(x,y));
        }
    }
    return 0;
}
```

三、可持续化线段树

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int maxn=1e6+5;
int node_cnt,n,m;
int sum[maxn<<5],rt[maxn],ls[maxn<<5],rs[maxn<<5];
int a[maxn],b[maxn];
int p;//修改点
int build(int l,int r){
	int root=++node_cnt;
	if(l==r) return root;
	int mid=(l+r)>>1;
	ls[root]=build(l,mid);
	rs[root]=build(mid+1,r);
	return root;
}
int update(int l,int r,int root){
	int dir=++node_cnt;
	ls[dir]=ls[root];rs[dir]=rs[root];sum[dir]=sum[root]+1;
	if(l==r) return dir;
	int mid=(l+r)>>1;
	if(p<=mid) ls[dir]=update(l,mid,ls[dir]);
	else rs[dir]=update(mid+1,r,rs[dir]);
      return dir;
}
int query(int u,int v,int l,int r,int k){
	int mid=(l+r)>>1,
	x=sum[ls[v]]-sum[ls[u]];
	if(l == r)
		return l;
	if(x>=k) return query(ls[u],ls[v],l,mid,k);
	else return query(rs[u],rs[v],mid+1,r,k-x);
}
	int l,r,k;
```

四、【树状数组】

```
int a[1005],c[1005]; //对应原数组和树状数组
int lowbit(int x){
	return x&(-x);
}
void updata(int i,int k){	//在 i 位置加上 k
	while(i <= n){
		c[i] += k;
		i += lowbit(i);
	}
}
int getsum(int i){		//求 A[1 - i]的和
	int res = 0;
	while(i > 0){
		res += c[i];
		i -= lowbit(i);
	}
	return res;
}
```

区间加可以采用差分的方式，维护一个差分数组 b 和 c[i]=b[i]*i

$$
\begin{aligned}
&\sum_{i=1}^{r} a_i \\
&=\sum_{i=1}^{r}\sum_{j=1}^{i} b_j \\
&=\sum_{i=1}^{r} b_i \times (r-i+1) \\
&=\sum_{i=1}^{r} b_i \times (r+1) - \sum_{i=1}^{r} b_i \times i
\end{aligned}
$$

```
Int b[MAXN], c[MAXN], n;
inline int lowbit(int x) { return x & (-x); }
void add(int pos, int v) {
		int cv = pos * v;
		while (pos <= n) {	b[pos] += v, c[pos] += vc;	pos += lowbit(pos);	}
}
int getsum(int *t, int k) {
		int ret = 0;
		while (k) {	ret += t[k];	k -= lowbit(k);	}
		return ret;
}
 // 将区间加差分为两个前缀加
void add1(int l, int r, int v) {	add(l, v), add(r + 1, -v); }
int getsum1(int l, int r) {
		return (r + 1) * getsum(b, r) - l * getsum(b, l - 1) - (getsum(c, r) -getsum(c, l - 1));
}
```

五、【LCA】

```
struct zzz {
	int t, nex;
}e[500010 << 1]; int head[500010], tot;
void add(int x, int y) {
	e[++tot].t = y;
	e[tot].nex = head[x];
	head[x] = tot;
}
int depth[500001], fa[500001][22], lg[500001];
void dfs(int now, int fath) {    //now 表示当前节点，fath 表示它的父亲节点
	fa[now][0] = fath; depth[now] = depth[fath] + 1;
	for(int i = 1; i <= lg[depth[now]]; ++i)
		fa[now][i] = fa[fa[now][i-1]][i-1];
	for(int i = head[now]; i; i = e[i].nex)
		if(e[i].t != fath) dfs(e[i].t, now);
}
int LCA(int x, int y) {
	if(depth[x] < depth[y]) //用数学语言来说就是：不妨设 x 的深度 >= y 的深度
		swap(x, y);
	while(depth[x] > depth[y])
		x = fa[x][lg[depth[x]-depth[y]] - 1]; //先跳到同一深度
	if(x == y)    //如果 x 是 y 的祖先，那他们的 LCA 肯定就是 x 了
		return x;
	for(int k = lg[depth[x]] - 1; k >= 0; --k) //不断向上跳（lg 就是之前说的常数优化）
		if(fa[x][k] != fa[y][k])
			x = fa[x][k], y = fa[y][k];
	return fa[x][0];
}
int main() {
	int n, m, s; scanf("%d%d%d", &n, &m, &s);
	for(int i = 1; i <= n-1; ++i) {
		int x, y; scanf("%d%d", &x, &y);add(x, y); add(y, x);
	}
for(int i = 1; i <= n; ++i) //预先算出 log_2(i)+1 的值，用的时候直接调用就可以了
	  lg[i] = lg[i-1] + (1 << lg[i-1] == i);    //看不懂的可以手推一下
	dfs(s, 0);
	int x,y;
	Cout<<LCA(x,y);
}
```

六、【树上差分】

```
ll t, n, k, m;
struct Edges {
	ll to, w, next; //（边的终点编号、权重、同起点的下一条边）
};
Edges edge[maxn*2];   //记录所有点的信息
ll head[maxn];        //例：head[1]=5;记录从 1 点出发的最后输入的一条边 5
ll cnt;               //记录每组数据的编号
ll faver[maxn];
void init(){
	memset(head, -1, sizeof(head));
	cnt = 0;
}
void add(ll from, ll to, ll w) //添加边{
	edge[cnt].to = to;
	edge[cnt].w = w;
	edge[cnt].next = head[from];
	head[from] = cnt++;
}
ll cf[maxn];      //差分数组
ll st[maxn];      //当前链上深度为 i 的节点
ll deep;
void dfs(ll u, ll fa) {
	st[++deep] = u;
	cf[u] += 1;
	int p = faver[u]+1;
	if (p >= deep)p = deep;
	cf[st[deep - p]] -= 1;
	for (ll i = head[u]; i != -1; i = edge[i].next) {
		if (edge[i].to == fa)continue;
		dfs(edge[i].to, u);
		cf[u] += cf[edge[i].to];
	}
	deep--;
}
```

【杂项】

一、【高精度】

加法：算法复杂度：o(n)

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

减法：算法复杂度：o(n)

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

乘法：算法复杂度：o(n^2)

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

除法：

```
int sub(int *a,int *b,int La,int Lb){
	if(La<Lb) return -1;//如果 a 小于 b，则返回-1
	if(La==Lb){
		for(int i=La-1;i>=0;i--)
			if(a[i]>b[i]) break;
			else if(a[i]<b[i]) return -1;//如果 a 小于 b，则返回-1
	}
	for(int i=0;i<La;i++){
		a[i]-=b[i];
		if(a[i]<0) a[i]+=10,a[i+1]--;
	}
	for(int i=La-1;i>=0;i--)
		if(a[i]) return i+1;//返回差的位数
	return 0;//返回差的位数
}
string div(string n1,string n2,int nn){
	const int L=1e5;
	string s,v;//s 存商,v 存余数
	  int a[L],b[L],r[L],La=n1.size(),Lb=n2.size(),i,tp=La;
	  fill(a,a+L,0);fill(b,b+L,0);fill(r,r+L,0);//数组元素都置为 0
	  for(i=La-1;i>=0;i--) a[La-1-i]=n1[i]-'0';
	  for(i=Lb-1;i>=0;i--) b[Lb-1-i]=n2[i]-'0';
	  if(La<Lb || (La==Lb && n1<n2)) return n1;
	  int t=La-Lb;//除被数和除数的位数之差
	  for(int i=La-1;i>=0;i--)//将除数扩大 10^t 倍
		if(i>=t) b[i]=b[i-t];
		else b[i]=0;
	     Lb=La;
   for(int j=0;j<=t;j++){
		  int temp;
		  while((temp=sub(a,b+j,La,Lb-j))>=0){
			La=temp;
			r[t-j]++;
		  }
	  }
for(i=0;i<L-10;i++) r[i+1]+=r[i]/10,r[i]%=10;//统一处理进位
	  while(!r[i]) i--;//将整形数组表示的商转化成字符串表示的
	  while(i>=0) s+=r[i--]+'0';
	    i=tp;
	  while(!a[i]) i--;//将整形数组表示的余数转化成字符串表示的</span>
	  while(i>=0) v+=a[i--]+'0';
	  if(v.empty()) v="0";
	  //cout<<v<<endl;
	  if(nn==1) return s;//返回商
	  if(nn==2) return v;//返回余数
}
```