Nantong University ICPC Team Notebook (2019-20)

Kouhai Takes Me Fly

2019 年 11 月 8 日

目录

语言相关

vim 配置

```
syntax on
set nu
set cindent
set tabstop=4
set shiftwidth=4
set noswapfile
set mouse=a

map <C-A> ggVG"+y

func Close(char)
  if getline(".")[col('.')-1]==a:char
    return "\<Right>"
  else return a:char
  endif
endfunc

inoremap { {}<Esc>i
inoremap } <c-r>=Close('}')<CR>
inoremap ( ()<Esc>i
inoremap ) <c-r>=Close(')')<CR>
inoremap [ []<Esc>i
inoremap ] <c-r>=Close(']')<CR>


map <F9> :call Run()<CR>
func! Run()
  exec "w"
  exec "!g++ -std=c++11 -O2 % -o %<"
  exec "!time ./%<"
endfunc
```

模板

```cpp
//author: thirtiseven
#include <iostream>
#include <algorithm>
#include <cstring>
#include <sstream>
#include <cmath>
#include <iomanip>
#include <cctype>
#include <cstdio>
#include <cstdlib>
#include <string>
#include <set>
#include <queue>
#include <map>
#include <vector>
#include <stack>
#include <bitset>

using ll = long long;
#define pb push_back
#define rep(i,n) for(int i=0; i<n; ++i)
#define mp std::make_pair
#define pii std::pair<int, int>

const int maxn = 1e5+7;
const int inf = 0x7f7f7f7f;
const int mod = 1e9+7;
const double pi = acos(-1.0);

int main(int argc, char *argv[]) {
  std::ios::sync_with_stdio(false);
  std::cin.tie(0);

  return 0;
}
```

取消同步

```cpp
std::ios::sync_with_stdio(false);
std::cin.tie(0);
```

浮点数输出格式

```cpp
//include <iomanip>

std::cout << std::fixed << std::setprecision(12) << ans << std::endl;
```

整型快速输入

```cpp
//整型
//若读入不成功，返回 false
//ios::sync_with_stdio(true)
//#include <cctype>
bool quick_in(int &x) {
  char c;
  while((c = getchar()) != EOF && !isdigit(c));
  if(c == EOF) {
```

```
    return false;
  }
  x = 0;
  do {
    x *= 10;
    x += c - '0';
  } while((c = getchar()) != EOF && isdigit(c));
  return true;
}

//带符号整型
//直接 = 返回值
//#include <cctype>
int read() {
  int x = 0, l = 1; char ch = getchar();
  while (!isdigit(ch)) {if (ch=='-') l=-1; ch=getchar();}
  while (isdigit(ch)) x=x*10+(ch^48),ch=getchar();
  return x*1;
}

template <class T>
inline bool Read(T &ret) {
    char c; int sgn;
    if(c=getchar(),c==EOF) return 0; //EOF
    while(c!='-'&&(c<'0'||c>'9')) c=getchar();
    sgn=(c=='-') ?-1:1 ;
    ret=(c=='-') ?0:(c -'0');
    while(c=getchar(),c>='0'&&c<='9')
        ret=ret*10+(c-'0');
    ret*=sgn;
    return 1;
}
```

字符串快速输入

```
bool quick_in(char *p) {
  char c;
  while((c = getchar()) != EOF && (c == ' ' || c == '\n'));
  if(c == EOF) {
    return false;
  }
  do {
    *p++ = c;
  } while((c=getchar()) != EOF && c != ' ' && c != '\n');
  *p = 0;
  return true;
}
```

python 输入

```
a, b, c =map(int,input().split(' '))
```

java 输入

```
Scanner cin=new Scanner(System.in);// 读入
```

int128 输入输出

```
std::ostream& operator<<(std::ostream& os, __int128 T) {
  if (T<0) os<<"-";if (T>=10 ) os<<T/10;if (T<=-10) os<<(-(T/10));
  return os<<( (int) (T%10) >0 ? (int) (T%10) : -(int) (T%10) ) ;
}

void scan(__int128 &x) {
  x = 0;
  int f = 1;
  char ch;
  if((ch = getchar()) == '-') f = -f;
  else x = x*10 + ch-'0';
  while((ch = getchar()) >= '0' && ch <= '9')
    x = x*10 + ch-'0';
  x *= f;
}

void print(__int128 x) {
  if(x < 0) {
    x = -x;
    putchar('-');
  }
  if(x > 9) print(x/10);
  putchar(x%10 + '0');
}
```

进制输出

```
std::cout << bin << x << std::endl;  // 二
std::cout << oct << x << std::endl;  // 八
std::cout << hex << x << std::endl;  // 十六
```

algorithm

```
std::unique(v.begin(), v.end());
//去重
//比较相邻元素 一样的放到后面 用前一般先排序
//返回去重完毕的下一个 iterator
```

```
std::stable_sort(v.begin(), v.end(), cmp);
//stable_sort 和 sort 的区别在于 前者作排序可以使原来的" 相同" 的值在序列中的相对位置不变

std::sort(iter_begin, iter_end, std::greater<int>());
//从大到小排序
```

bitset

```
#include <bitset>
b.any()          b 中是否存在置为 1 的二进制位？
b.none()        b 中不存在置位 1 的二进制位吗？
b.count()        b 中置为 1 的二进制位的个数
b.size()        b 中二进制位的个数
b[pos]          访问 b 中在 pos 处的二进制位
b.test(pos)        b 中在 pos 处的二进制位是否为 1？
b.set()          把 b 中所有二进制位都置为 1
b.set(pos)        把 b 中在 pos 处的二进制位置为 1
b.reset()        把 b 中所有二进制位置为 0
b.reset(pos)      把 b 中在 pos 处的二进制位置为 0
b.flip()        把 b 中所有二进制位逐位取反
b.flip(pos)        把 b 中在 pos 处的二进制位取反
b.to\_ulong()      用 b 中同样的二进制位返回一个 unsigned long 值
os << b          把 b 中的位集中输出到 os 流
```

string

```
std::stoi C++11      //将字符串转化成带符号 (Signed) 整数
std::stol C++11      //将字符串转化成带符号整数
std::stoll C++11    //将字符串转化成带符号整数
std::stoul C++11    //将字符串转化成无符号 (Unsigned) 整数
std::stoull C++11    //将字符串转化成无符号整数
std::stof C++11      //将字符串转化成浮点数
std::stod C++11      //将字符串转化成浮点数
std::stold C++11    //将字符串转化成浮点数
std::to_string C++11  //将一个整数或浮点数转化成字符串
std::to_wstring C++11  //将一个整数或浮点数转化成宽字符串
std::transform(s.begin(), s.end(), s.begin(), ::toupper); // 大写
std::transform(s.begin(),s.end(),s.begin(), ::tolower); //小写
```

位运算

```
//去掉最后一位
 x >> 1
//在最后加一个 0
x << 1
//在最后加一个 1
x << 1 + 1
//把最后一位变成 1
 x | 1
//把最后一位变成 0
x | 1 - 1
//最后一位取反
 x ^ 1
//把右数第 k 位变成 1
 x | (1 <<(k-1))
//把右数第 k 位变成 0
 x & ~ (1 << (k-1))
//右数第 k 位取反
x ^ (1 << (k-1))
//取末三位
x & 7
//取末 k 位
 x & (1 << k-1)
//取右数第 k 位
 x >>(k-1) & 1
//把末 k 位变成 1
x |(1 << k-1)
//末 k 位取反
 x ^ (1 << k-1)
//把右边连续的 1 变成 0
x & (x+1)
//x 个 1
((1<<x-1)
//二进制里 1 的数量
(x>>16)+(x&((1<<16)-1))
```

内置位运算函数

```
— Built-in Function: int __builtin_ffs (unsigned int x)
Returns one plus the index of the least significant 1-bit of x, or if x is zero, returns zero.
返回右起第一个 ‘1’ 的位置。

— Built-in Function: int __builtin_clz (unsigned int x)
Returns the number of leading 0-bits in x, starting at the most significant bit position. If x is 0, the result is undefined.
返回左起第一个 ‘1’ 之前 0 的个数。

— Built-in Function: int __builtin_ctz (unsigned int x)
Returns the number of trailing 0-bits in x, starting at the least significant bit position. If x is 0, the result is undefined.
```

```
返回右起第一个 ‘1’ 之后的 0 的个数。

— Built-in Function: int __builtin_popcount (unsigned int x)
Returns the number of 1-bits in x.
返回 ‘1’ 的个数。

— Built-in Function: int __builtin_parity (unsigned int x)
Returns the parity of x, i.e. the number of 1-bits in x modulo 2.
返回 ‘1’ 的个数的奇偶性。

— Built-in Function: int __builtin_ffsl (unsigned long)
Similar to __builtin_ffs, except the argument type is unsigned long.

— Built-in Function: int __builtin_clzl (unsigned long)
Similar to __builtin_clz, except the argument type is unsigned long.

— Built-in Function: int __builtin_ctzl (unsigned long)
Similar to __builtin_ctz, except the argument type is unsigned long.

— Built-in Function: int __builtin_popcountl (unsigned long)
Similar to __builtin_popcount, except the argument type is unsigned long.

— Built-in Function: int __builtin_parityl (unsigned long)
Similar to __builtin_parity, except the argument type is unsigned long.

— Built-in Function: int __builtin_ffsll (unsigned long long)
Similar to __builtin_ffs, except the argument type is unsigned long long.

— Built-in Function: int __builtin_clzll (unsigned long long)
Similar to __builtin_clz, except the argument type is unsigned long long.

— Built-in Function: int __builtin_ctzll (unsigned long long)
Similar to __builtin_ctz, except the argument type is unsigned long long.

— Built-in Function: int __builtin_popcountll (unsigned long long)
Similar to __builtin_popcount, except the argument type is unsigned long long.

— Built-in Function: int __builtin_parityll (unsigned long long)
Similar to __builtin_parity, except the argument type is unsigned long long.



随机数函数:
default_random_engine: 随机非负数 (不建议单独使用)。
uniform_int_distribution: 指定范围的随机非负数。
uniform_real_distribution: 指定范围的随机实数。
bernoulli_distribution: 指定概率的随机布尔值。

示例:default_random_engine e;
    uniform_int_distribution<int>u(0,9);
    cout<<u(e)<<endl;
```

随机

```cpp
//#include <iostream>
//#include <random>

std::vector<int> permutation(100);
for (int i = 0; i < 100; i++) {
  permutation[i] = i+1;
}
std::mt19937_64 mt1(1); //64 位
std::mt19937 mt2(2); //32 位
shuffle(permutation.begin(), permutation.end(), mt2); // 打乱序列
for (auto it: permutation) {
  std::cout << it << " ";
}
```

归并求有序集合异或

```cpp
//有序数列异或 ab 可为两个 vector
std::vector<int> v_symDifference;
std::set_symmetric_difference(a.begin(), a.end(), b.begin(), b.end(),std::back_inserter(v_symDifference));
```

优先队列

```cpp
std::priority_queue<int> xxx 大根堆
std::priority_queue<int, std::vector<int>, std::greater<int>> xxxx 小根堆
```

动态规划

背包问题

```
const int maxn=100005;
int w[maxn],v[maxn],num[maxn];
int W,n;
int dp[maxn];

void ZOP(int weight, int value) {
  for(int i = W; i >= weight; i--) {
    dp[i]=std::max(dp[i],dp[i-weight]+value);
  }
}

void CP(int weight, int value){
  for(int i = weight; i <= W; i++) {
    dp[i] = std::max(dp[i], dp[i-weight]+value);
  }
}

void MP(int weight, int value, int cnt){
  if(weight*cnt >= W) {
     CP(weight, value);
  } else {
    for(int k = 1; k < cnt; k <<= 1) {
      ZOP(k*weight, k*value), cnt -= k;
    }
    ZOP(cnt*weight, cnt*value);
  }
}
```

树形背包

```
/*
 * Author: Simon
 * 功能: 树形依赖背包问题
 * 定义 dp[u][i] 表示，以 u 为根节点的子树中保留 i 条树枝所获得的最大权值
 * 则转移方程为
   ↪ dp[u][i]=max(dp[u][i],dp[left[u]][i-j-1]+left[u].w+dp[right[u]][j-1]+right[u].w)
 * 表示 u 的右儿子保留 j-1 条边, u 的左儿子保留剩下的 i-j-1 条边, 此时总共
   ↪ 有 i-2 条边, 还要加上 u-left[u],u-right[u] 这两条边。
 * 另外一种转移状态 dp[u][i]=max(dp[u][i],dp[u][i-j]+dp[v][j-1]+w)
 * 跟上面类似, 只不过将 u 与其中一个儿子节点的状态放在一起。此时需要倒序枚
   ↪ 举 i 来保证只选择一次 (类似 01 背包)。
 * 没有访问过的子树不会保存在 dp[u][i] 中, 所以不会出现重复计算的情况。
 */
void dfs(int u,int p=-1){
  sz[u]=1;
  for(auto t:g[u]){
    int &v=t.first,&w=t.second;
    if(v==p) continue;
    dfs(v,u);sz[u]+=sz[v];
    for(int i=min(q,sz[u]);i>=1;i--){
      for(int j=1;j<=min(sz[v],i);j++){
        dp[u][i]=max(dp[u][i],dp[u][i-j]+dp[v][j-1]+w);
      }
    }
  }
}
```

最长单调子序列 (nlogn)

```
int arr[maxn], n;

template<class Cmp>
int LIS (Cmp cmp) {
  static int m, end[maxn];
  m = 0;
  for (int i=0; i<n; i++) {
    int pos = lower_bound(end, end+m, arr[i], cmp)-end;
    end[pos] = arr[i], m += pos==m;
  }
  return m;
}

bool greater1(int value) {
  return value >=1;
}

/*********
  std::cout << LIS(std::less<int>()) << std::endl;          //严格上升
    std::cout << LIS(std::less_equal<int>()) << std::endl;   //非严格上升
    std::cout << LIS(std::greater<int>()) << std::endl;      //严格下降
    std::cout << LIS(std::greater_equal<int>()) << std::endl;//非严格下降
    std::cout << count_if(a,a+7,std::greater1) << std::endl; //计数
**********/
```

最长公共子序列

```
int dp[maxn][maxn];

void LCS(int n1, int n2, int A[], int B[]) {
  for(int i=1; i<=n1; i++) {
    for(int j=1; j<=n2; j++) {
      dp[i][j] = dp[i-1][j];
      if (dp[i][j-1] > dp[i][j]) {
        dp[i][j] = dp[i][j-1];
      }
```

```
      if (A[i] == B[j] && dp[i-1][j-1] + 1 > dp[i][j]) {
        dp[i][j] = dp[i-1][j-1] + 1;
      }
    }
  }
}
```

单调队列优化 DP

```
//单调队列求区间最小值
int a[maxn], q[maxn], num[maxn] = {0};
int Fmin[maxn];
int k, n, head, tail;

void DPmin() {
  head = 1, tail = 0;
  for (int i = 1; i <= n; i++) {
    while (num[head] < i-k+1 && head <= tail) head++;
    while (a[i] <= q[tail] /* 区间最大值此处改为 >=*/ && head <= tail) tail--;
    num[++tail] = i;
    q[tail] = a[i];
    Fmin[i] = q[head];
  }
}
```

区间 DP

```
for (int x = 0; x < n; x++){//枚举长度
  for (int i = 1; i + x <= n; i++){//枚举起点
    dp[i][i] = 1;
    int j = x + i;//终点
    dp[i][j] = dp[i + 1][j] + 1;
    for (int k = i + 1; k <= j; k++) {
      if (a[i] == a[k])
        dp[i][j] = min(dp[i][j], dp[i][k - 1] + dp[k + 1][j]);
    }
  }
}
```

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 */)//不是每个题都要判断前导零
{
  //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==-1) return 1;/* 这里一般返回 1，表示你枚举的这个数是合法的，那么这里就需要你在枚举时必须每一位都要满足题目条件，也就是说当前枚举到 pos 位，一定要保证前面已经枚举的数位是合法的。不过具体题目不同或者写法不同的话不一定要返回 1 */
  //第二个就是记忆化 (在此前可能不同题目还能有一些剪枝)
  if(!limit && !lead && dp[pos][state]!=-1) return dp[pos][state];
  /* 常规写法都是在没有限制的条件记忆化，这里与下面记录状态是对应，具体为什么是有条件的记忆化后面会讲 */
  int up=limit?a[pos]:9;//根据 limit 判断枚举的上界 up; 这个的例子前面用 213 讲过了
  ll ans=0;
  //开始计数
  for(int i=0;i<=up;i++)//枚举，然后把不同情况的个数加到 ans 就可以了
  {
    if() ...
    else if()...
    ans+=dfs(pos-1,/* 状态转移 */,lead && i==0,limit && i==a[pos]) //最后两个变量传参都是这样写的
    /* 这里还算比较灵活，不过做几个题就觉得这里也是套路了
    大概就是说，我当前数位枚举的数是 i，然后根据题目的约束条件分类讨论
    去计算不同情况下的个数，还有要根据 state 变量来保证 i 的合法性，比如题目
    要求数位上不能有 62 连续出现，那么就是 state 就是要保存前一位 pre，然后分类，
    前一位如果是 6 那么这意味就不能是 2，这里一定要保存枚举的这个数是合法*/
  }
  //计算完，记录状态
  if(!limit && !lead) dp[pos][state]=ans;
  /* 这里对应上面的记忆化，在一定条件下时记录，保证一致性，当然如果约束条件不需要考虑 lead，这里就是 lead 就完全不用考虑了 */
  return ans;
}

ll solve(ll x)
{
  int pos=0;
  while(x)//把数位都分解出来
  {
    a[pos++]=x%10;//个人老是喜欢编号为 [0,pos)，看不惯的就按自己习惯来，反正注意数位边界就行
    x/=10;
  }
  return dfs(pos-1/* 从最高位开始枚举 */,/* 一系列状态 */,true,true);//刚开始最高位都是有限制并且有前导零的，显然比最高位还要高的一位视为 0 嘛
}

```

```cpp
int main()
{
  ll le,ri;
  while(~scanf("%lld%lld",&le,&ri))
  {
    //初始化 dp 数组为-1，这里还有更加优美的优化，后面讲
    printf("%lld\n",solve(ri)-solve(le-1));
  }
}


//模板 2
int a[maxn],bit[maxn];//a 为分解整数数组，bit 数组为 10^(i-1)
pair<int,int>dp[maxn][2000];//first= 满足条件的数个数，second= 满足条件的数的和
bool vis[maxn][2000];
pair<int,int> dfs(int pos,int sta,int num,bool lead,bool limit){//求满足条件的所有数的和
  if(pos==0) return make_pair(1,0);//计数
  if(!limit&&!lead&&vis[pos][sta]) return dp[pos][sta];
  if(!limit&&!lead) vis[pos][sta]=1;
  int up=limit?a[pos]:9,t,tt; pair<int,int>tmp,ans;
  for(int i=0;i<=up;i++){
    if(num>=k&&!(sta&(1<<i))) continue;//不满足条件，跳出
    if(lead&&!i) t=0,tt=0;
    else t=(sta|(1<<i)),tt=(sta&(1<<i))?num:num+1;
    tmp=dfs(pos-1,t,tt,lead&&!i,limit&&i==up);
    ans.first+=tmp.first;ans.first%=mod;//满足条件的数的个数
    ans.second+=tmp.first*bit[pos]%mod*i%mod+tmp.second;ans.second%=mod;//满足条件的数的和 10+11+...+19=9*(10*1)+45
  }
  if(!limit&&!lead) dp[pos][sta]=ans;
  return ans;
}
```

SOSDP

```cpp
//https://codeforces.com/blog/entry/45223#

for(int i = 0; i<(1<<N); ++i)
  F[i] = A[i];
for(int i = 0;i < N; ++i) for(int mask = 0; mask < (1<<N); ++mask){
  if(mask & (1<<i))
    F[mask] += F[mask^(1<<i)];
}
```

数论

暴力判素数

```
bool is_prime(int  u) {
  if(u ==  0 || u ==  1) return false;
  if(u ==  2)       return true;
  if(u%2 == 0)       return false;
  for(int i=3; i <=  sqrt(u) ; i+=2)
    if(u%i==0)       return false;
  return true;
}
```

米勒罗宾素性检测

```
using ll = long long;

ll prime[6] = {2, 3, 5, 233, 331};

ll qmul(ll x, ll y, ll mod) { // 乘法防止溢出， 如果 p * p 不爆 ll 的话可以直接乘； O(1) 乘法或者转化成二进制加法
  return (x * y - (long long)(x / (long double)mod * y + 1e-3) *mod + mod) % mod;
}

ll qpow(ll a, ll n, ll mod) {
  ll ret = 1;
  while(n) {
    if(n & 1) ret = qmul(ret, a, mod);
    a = qmul(a, a, mod);
    n >>= 1;
  }
  return ret;
}

bool Miller_Rabin(ll p) {
  if(p < 2) return 0;
  if(p != 2 && p % 2 == 0) return 0;
  ll s = p - 1;
  while(! (s & 1)) s >>= 1;
  for(int i = 0; i < 5; ++i) {
    if(p == prime[i]) return 1;
    ll t = s, m = qpow(prime[i], s, p);
    while(t != p - 1 && m != 1 && m != p - 1) {
      m = qmul(m, m, p);
      t <<= 1;
    }
    if(m != p - 1 && !(t & 1)) return 0;
  }
  return 1;
}
```

埃氏筛

```
bool prime_or_not[maxn];
for (int i = 2; i <= int(sqrt(maxn)); i++) {
  if (!prime_or_not[i]) {
      for (int j = i * i; j <= maxn; j = j+i) {
      prime_or_not[j] = 1;
    }
  }
}
```

欧拉筛

```
const int maxn = "Edit";
int flag[maxn], primes[maxn], totPrimes;

void euler_sieve(int n) {
  totPrimes = 0;
  memset(flag, 0, sizeof(flag));
  for (int i = 2; i <= n; i++) {
    if (!flag[i]) {
      primes[totPrimes++] = i;
    }
    for (int j = 0; i * primes[j] <= n; j++) {
      flag[i * primes[j]] = true;
      if (i % primes[j] == 0)
      break;
    }
  }
}
```

区间筛

```
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
typedef long long ll;
const int MAXN = 1e5;
bool is_prime[MAXN];
bool is_prime_small[MAXN];
ll prime[MAXN];
ll prime_num = 0;

//对区间 [a,b) 内的整数执行筛法，is_prime[i-a]=true <=> 表示 i 是素数 (下标偏移了 a)
```

```
void segment_sieve(ll a, ll b) {
  for (ll i = 0; i * i < b; i++) //对 [2,sqrt(b)) 的初始化全为质数
    is_prime_small[i] = true;
  for (ll i = 0; i < b - a; i++) //对下标偏移后的 [a,b) 进行初始化
    is_prime[i] = true;

  for (ll i = 2; i * i < b; i++) { //筛选 [2,sqrt(b))
    if (is_prime_small[i]) {
      for (ll j = 2 * i; j * j < b; j += i)
        is_prime_small[j] = false;
      //(a+i-1)/i 得到最接近 a 的 i 的倍数，最低是 i 的 2 倍，然后筛选
      for (ll j = max(2LL, (a + i - 1) / i) * i; j < b; j += i)
        is_prime[j - a] = false;
    }
  }
  for (ll i = 0; i < b - a; i++) //统计个数
    if (is_prime[i])
      prime[prime_num++] = i + a;
}

int main() {
  ll a, b;
  while (~scanf("%lld%lld", &a, &b)) {
    prime_num = 0;
    memset(prime, 0, sizeof(prime));
    segment_sieve(a, b);
    printf("%lld\n", prime_num);
  }
  return 0;
}
```

分解质因数

```
int cnt[maxn];//存储质因子是什么
int num[maxn];//该质因子的个数
int tot = 0;//质因子的数量
void factorization(int x)//输入 x，返回 cnt 数组和 num 数组
{
  for(int i=2;i*i<=x;i++)
  {
    if(x%i==0)
    {
      cnt[tot]=i;
      num[tot]=0;
      while(x%i==0)
      {
        x/=i;
        num[tot]++;
      }
      tot++;
    }
  }
  if(x!=1)
  {
    cnt[tot]=x;
    num[tot]=1;
    tot++;
  }
}
```

PollardRho 质因数分解

```
long long factor[100];//质因数分解结果（刚返回时是无序的）
int tol;//质因数的个数。数组小标从 0 开始

long long gcd(long long a,long long b)
{
  if(a==0)return 1;//???????
  if(a<0) return gcd(-a,b);
  while(b)
  {
    long long t=a%b;
    a=b;
    b=t;
  }
  return a;
}

long long Pollard_rho(long long x,long long c)
{
  long long i=1,k=2;
  long long x0=rand()%x;
  long long y=x0;
  while(1)
  {
    i++;
    x0=(mult_mod(x0,x0,x)+c)%x;
    long long d=gcd(y-x0,x);
    if(d!=1&&d!=x) return d;
    if(y==x0) return x;
    if(i==k){y=x0;k+=k;}
  }
}
//对 n 进行素因子分解
void findfac(long long n)
{
  if(Miller_Rabin(n))//素数
  {
```

```
    factor[tol++]=n;
    return;
  }
  long long p=n;
  while(p>=n)p=Pollard_rho(p,rand()%(n-1)+1);
  findfac(p);
  findfac(n/p);
}
int main()
{
   // srand(time(NULL));//需要 time.h 头文件  //POJ 上 G++ 要去掉这句话
  int T;
  long long n;
  scanf("%d",&T);
  while(T--)
  {
    scanf("%I64d",&n);
    if(Miller_Rabin(n))
    {
      printf("Prime\n");
      continue;
    }
    tol=0;
    findfac(n);
    long long ans=factor[0];
    for(int i=1;i<tol;i++)
      if(factor[i]<ans)
       ans=factor[i];
    printf("%I64d\n",ans);
  }
  return 0;
}
```

反素数

```
/*
Author: Simon
其实顾名思义，素数就是因子只有两个的数，那么反素数，就是因子最多的数（并且
  ↪ 因子个数相同的时候值最小），所以反素数是相对于一个集合来说的。
反素数的特点。
1. 反素数肯定是从 2 开始的连续素数的幂次形式的乘积。
2. 数值小的素数的幂次大于等于数值大的素数的幂次，即
  e1>=e2>=e3>=……>=ek
*/
int a[16]={2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53}
//给定因子数 n，求满足因子数个数等于 n 的最小数。
void dfs(int dep/* 第 dep 个素数 */,int sum/* 当前的数 */,int num/* 因子个数
  ↪ */,int up/* 上个素数的幂 */){
  if(dep>=16||num>n) return ;
  if(num==n){
    ans=min(ans,sum);
    return ;
  }
  for(int i=1;i<=up;i++){ //由性质 2 可知，数值大的素数的幂次小于等于数值小
    ↪ 的素数的幂次
    if(sum*fpow(a[dep],i)>ans) break;
    dfs(dep+1,sum*fpow(a[dep],i),num*(i+1),i);
  }
}
//求小于等于 n 的因子数最多的数。与上面做法相同，只是 dfs 结束条件改一下。
```

最大公约数

```
ll gcd(ll a, ll b) {
  ll t;
  while(b != 0) {
    t=a%b;
    a=b;
    b=t;
  }
  return a;
}
```

最小公倍数

```
ll lcm(ll a, ll b) {
  return a * b / gcd(a, b);
}
```

扩展欧几里得

```
/*
 * Author: Simon
 * 复杂度: O(log(max(a,b)))
 * 功能: 求解 a*x+b*y=c
 */

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

```cpp
}
/*
 * 解 a*x+b*y=c
 * 假设有一对特解 x=n,y=m
 * 则其通解为: x=n-b*t or y=m+a*t
 */
void solve(int n, int A, int B, int C) {
  int x, y;
  int g = exgcd(A, B, x, y);
  if (C % g != 0) {
    cout << "-1" << endl;
    return;
  }
  A /= g, B /= g, C /= g;
  x *= C % B;
  x = (x % B + B) % B; y = (C - A * x) / B; /* 求最小非负整数 x, 则 y=(c-a*x)/b */
  // y *= C % A;
  // y = (y % A + A) % A; x = (C - B * y) / A; /* 求最小非负整数 y, 则 x=(c-b*y)/a */
  /*
    具体题目
  */
}
```

中国剩余定理

```cpp
/*
 * Author: Simon
 * 复杂度: O(n)
 * 功能: 求解 x=ai(mod mi) 同余方程组
 * 某些计数问题或数论问题出于加长代码、增加难度、或者是一些其他不可告人的原因 给出的模数: 不是质数!
 * 但是对其质因数分解会发现它没有平方因子, 也就是该模数是由一些不重复的质数相乘得到。
 * 那么我们可以分别对这些模数进行计算, 最后用 CRT 合并答案。
 */
int crt(int ai[], int mi[], int len) {
  int ans = 0, lcm = 1;
  for (int i = 0; i < len; i++) lcm *= mi[i];
  for (int i = 0; i < len; i++) {
    int Mi = lcm / mi[i];
    int inv = fpow(Mi, mi[i] - 2, mi[i]);
    int x = fmul(fmul(inv, Mi, lcm), ai[i], lcm); //若 lcm 大于 1e9 需要用快速乘 fmul
    ans = (ans + x) % lcm;
  }
  return ans;
}
```

扩展 CRT

```cpp
/*
 * Author:Simon
 * 功能: 模数可以不互质情况下, 求解同余方程组。
 * 若返回-1 则无解, 否则返回最小非负整数解 x, 通解为 x+i*M
 * 复杂度: O(nlog)
 */
int excrt(int mi[],int ai[],int n){ //扩展中国剩余定理
  int M=mi[1],ans=ai[1]; //x=ans+i*M, 得到一个通解
  for(int i=2;i<=n;i++){
    int a=M,b=mi[i],c=((ai[i]-ans)%b+b)%b/* 将 c 化为正数 */,x,y; //与第二个方程组成不定方程
    int gcd=exgcd(a,b,x,y); //通过扩展欧几里得解的一组特解 (p,q)
    if(c%gcd!=0) return -1;
    a/=gcd,b/=gcd;
    (x*=(c/gcd)%b)%=b; x=(x+b)%b;
    ans+=x*M; //则 新同余方程的解 x=ans+p*M
    M*=b; //所有模数的最小公倍数 (M*b)/gcd=M*(b/gcd)
    ans%=M; //最小整数解
  }
  return (ans+M)%M;
}
```

欧拉函数

```cpp
LL EulerPhi(LL n){
  LL m = sqrt(n + 0.5);
  LL ans = n;
  for(LL i = 2; i <= m; ++i)
  if(n % i == 0) {
    ans = ans - ans / i;
  while(n % i == 0)
    n/=i;
  }
  if(n > 1)
    ans = ans - ans / n;
  return ans;
}
```

原根

```cpp
/*
Author: Simon
平均复杂度 O(loglog(p))
若不存在原根则返回-1
```

```
对于所有素数 p>2, 正整数 e, 当前仅当 n=1, 2, 4, p^e,2p^e 有原根
若 g 是 p 的原根, 对于 1<=i<p, g^i mod p, 互不相同, 即唯一。
*/
int proot(int p){ //fac 为 (p-1) 的所有质因子。
  for(int a=2;a<p;a++){
    bool flag=0;
    for(int i=0;i<fac.size();i++){
      int v=fac[i];
      if(fpow(a,(p-1)/v,p)==1){ //如果存在 d, a^{p-1/d} %p=1 则 a 不是 p 的
        ↪ 原根。
        flag=1;break;
      }
    }
    if(!flag) return a;
  }
  return -1;
}
```

求逆元

```
ll Inv(ll a, ll n){
  return PowMod(a, EulerPhi(n) - 1, n);
  //return PowMod(a,n-2,n); //n 为素数
}

int Inv(int a, int n) {
  int d, x, y;
  d = extended_euclid(a, n, x, y);
  if(d == 1)  return (x%n + n) % n;
  else      return -1; // no solution
}
```

快速乘法取模

```
//by sevenkplus
#define ll long long
#define ld long double
ll mul(ll x,ll y,ll z){return (x*y-(ll)(x/(ld)z*y+1e-3)*z+z)%z;}

//by Lazer2001
inline long long mmul (long long a, long long b, const long long& Mod) {
  long long lf = a * (b >> 25LL) % Mod * (1LL << 25) % Mod;
    long long rg = a * ( b & ( ( 1LL << 25 ) - 1 ) ) % Mod ;
  return (lf + rg) % Mod ;
}
```

快速幂取模

```
using ll = long long;

ll PowMod(ll a, ll b, const ll &Mod) {
  a %= Mod;
  ll ans = 1;
  while(b) {
    if (b & 1){
      ans = (ans * a) % Mod;
    }
    a = (a * a) % Mod;
    b >>= 1;
  }
  return ans;
}

ll Inv(ll a, ll n){
  return PowMod(a,n-2,n);
}


ll C(const ll &n, const ll &m, const int &pr) {
  ll ans = 1;
  for (int i = 1; i <= m; i++) {
    ll a = (n - m + i) % pr;
    ll b = i % pr;
    ans = (ans * (a * Inv(b, pr)) % pr) % pr;
  }
  return ans;
}
```

互质对数计数

```
//Written by Simon
//求 r 以内与 n 不互质的数的个数
int solve(int r) {
  int sum=0;
  for(int i=1;i<(1<<fac.size());i++) {//枚举质因数的每一种组合
    int ans=1,num=0;
    for(int j=0;j<fac.size();j++) {//求当前组和的积
      if(i&(1<<j)) {
        ans *= fac[j];
        num++;
      }
    }
    if(num&1) sum+=r/ans;//如果当前组合个数为奇数个, 加上 r 以内能被 ans 整
      ↪ 除的数的个数
    else sum-=r/ans;//否则减去 r 以内能被 ans 整除的数的个数
```

```
  }
  return sum;
}
```

BSGS

```
//Author: Simon
#include <algorithm>
#include <cmath>
#include <cstring>
using ll = long long;
const int maxn = 1000005;
const ll mod = 611977;

struct HashMap {
  ll head[mod+5], key[maxn], value[maxn], nxt[maxn], tol;
  inline void clear() {
    tol=0;
    memset(head,-1,sizeof(head));
  }
  HashMap() {
    clear();
  }
  inline void insert(ll k,ll v) {
    ll idx = k % mod;
    for(ll i = head[idx]; ~i; i = nxt[i]) {
      if(key[i] == k) {
        value[i] = std::min(value[i], v);
        return ;
      }
    }
    key[tol] = k;
    value[tol] = v;
    nxt[tol] = head[idx];
    head[idx] = tol++;
  }
  inline ll operator [](const ll &k) const {
    ll idx = k % mod;
    for(ll i=head[idx]; ~i; i=nxt[i]) {
      if(key[i]==k) return value[i];
    }
    return -1;
  }
}mp;

inline ll fpow(ll a, ll b, ll mod) {
  a %= mod;
  ll ans = 1;
  while (b) {
    if(b&1) ans = ans * a % mod;
    a = a * a % mod;
    b >>= 1;
  }
  return ans;
}
inline ll exgcd(ll a,ll b,ll &x,ll &y) {
  if (b==0) {
    x=1, y=0;
    return a;
  }
  ll ans = exgcd(b, a%b, y, x);
  y -= a/b*x;
  return ans;
}

inline ll Bsgs(ll a,ll b,ll mod) {
  a %= mod, b %= mod;
  if (b==1) return 0;
  ll m = ceil(sqrt(mod)), inv, y;
  exgcd(fpow(a, m, mod), mod, inv, y);
  inv = (inv % mod + mod) % mod;
  mp.insert(1, 1);
  for(ll i=1, e=1; i<m; i++) {
    e = e * a % mod;
    if(mp[e] == -1) mp.insert(e, i+1);
  }
  for(ll i = 0; i <= m; i++) {
    if(mp[b] != -1) {
      ll ans = mp[b]-1;
      return ans + i * m;
    }
    b = b * inv % mod;
  }
  return -1;
}

inline ll gcd(ll a, ll b) {
  return b==0 ? a : gcd(b, a%b);
}

inline int exBsgs(int a,int b,int mod) {//扩展 BSGS，处理 a，mod 不互质的情况
  if(b==1) return 0;
  for(int g=gcd(a,mod),i=0;g!=1;g=gcd(a,mod),i++)  {
    if(b%g) return -1;//保证 g 为 a,b,mod 的最大公约数
    mod/=g;
  }
  return Bsgs(a,b,mod);
}
```

Pohlig Hellman

```
/*
 * Author: Simon
 * 功能: 求解 a^x=b(mod p), 其中 p 可达 1e18, 但 p 的质因数个数很少。
 * 复杂度: 复杂度 O(k · e_i · log(p_i^{e_i})+k · log(p_i^{e_i}))
 * 其中 k 为 p-1 的质因子的个数, e_i 为 p-1 的质因子的最高幂次。p_i 最高幂
 ↪ 次对应的质因子。
 */
#include<bits/stdc++.h>
using namespace std;
typedef int Int;
#define int __int128_t
#define INF 0x3f3f3f3f
#define maxn 200005
struct Istream {
  template <class T>
  Istream &operator >>(T &x) {
    static char ch;static bool neg;
    for(ch=neg=0;ch<'0' || '9'<ch;neg|=ch=='-',ch=getchar());
    for(x=0;'0'<=ch && ch<='9';(x*=10)+=ch-'0',ch=getchar());
    x=neg?-x:x;
    return *this;
  }
}fin;
struct Ostream {
  template <class T>
  Ostream &operator <<(T x) {
    x<0 && (putchar('-'),x=-x);
    static char stack[233];static int top;
    for(top=0;x;stack[++top]=x%10+'0',x/=10);
    for(top==0 && (stack[top=1]='0');top;putchar(stack[top--]));
    return *this;
  }

  Ostream &operator <<(char ch) {
    putchar(ch);
    return *this;
  }
}fout;
vector<pair<int,int> >fac;
void solve(int n){ /* 求解质因数 */
  fac.clear();
  for(int i=2;i*i<=n;i++){
    int tmp=0;
    if(n%i==0){
      while(n%i==0) n/=i,tmp++;
      fac.push_back({i,tmp});
    }
  }
  if(n>1) fac.push_back({n,1});
}
int fpow(int a,int b,int mod){
  int ans=1;a%=mod;
  while(b){
    if(b&1) (ans*=a)%=mod;
    (a*=a)%=mod;
    b>>=1;
  }
  return ans;
}
int proot(int p){ //求原根
  for(int a=2;a<p;a++){
    bool flag=0;
    for(int i=0;i<fac.size();i++){
      int v=fac[i].first;
      if(fpow(a,(p-1)/v,p)==1){
        flag=1;break;
      }
    }
    if(!flag) return a;
  }
  return -1;
}
int cal(int a,int b,int p,pair<int,int>fac){
  int ans=0,t=fac.first;
  map<int,int>mp;
  for(int i=0;i<fac.first;i++) mp[fpow(a,i*(p-1)/t,p)]=i;
  for(int i=1;i<=fac.second;i++){
    int c=mp[fpow(b,(p-1)/t,p)];
    (ans+=c*t/fac.first)%=p;
    (b*=fpow(fpow(a,c*t/fac.first,p),p-2,p))%=p;
    (t*=fac.first)%=p;
  }
  return ans;
}
int exgcd(int a,int b,int &x,int &y){
  if(b==0){
    x=1,y=0;
    return a;
  }
  int g=exgcd(b,a%b,y,x);
  y-=a/b*x;
  return g;
}
int excrt(int mi[],int ai[],int n){ //扩展中国剩余定理
  int M=mi[1],ans=ai[1];
  for(int i=2;i<=n;i++){
    int a=M,b=mi[i],c=((ai[i]-ans)%b+b)%b;
    int x,y;
```

```cpp
    int gcd=exgcd(a,b,x,y);
    if(c%gcd!=0) return -1;
    a/=gcd,b/=gcd;
    (x*=(c/gcd)%b)%=b; x=(x+b)%b;
    ans+=x*M;
    M*=b;
    ans%=M;
  }
  return (ans+M)%M;
}
int pohlig_hellman(int a,int b,int p,vector<pair<int,int> >fac){ /* 求解
  ↪ a^x=b(mod p),a 为 p 的原根 */
  int mi[fac.size()+5]={0},ai[fac.size()+5]={0};
  for(int i=0;i<fac.size();i++){
    mi[i+1]=fpow(fac[i].first,fac[i].second,p); /*pi^{ei} */
    ai[i+1]=cal(a,b,p,fac[i]); /* 求解 xi=x (mod pi^{ei}) */
  }
  return excrt(mi,ai,fac.size()); /* 扩展中国剩余定理合并同余方程 */
}
Int main(){
  Int T;cin>>T;
  while(T--){
    int p,a,b;fin>>p>>a>>b; /* 求 a^x=b(mod p) */
    solve(p-1); /* 求得 p-1 的质因数，及其幂次 */
    int root=proot(p); /* 求得 p 的原根 */
    int pa=pohlig_hellman(root,a,p,fac)/*g^{pa}=a(mod p)
      ↪ */,pb=pohlig_hellman(root,b,p,fac);/*g^{pb}=b(mod p) */
    if(pa==0){ /* 转换为求 g^{pa · x}=g^{pb}(mod p)，由欧拉定理得 pa ·
      ↪ x=pb(mod (p-1))*/
      if(pb==0) fout<<1;
      else fout<<-1;
    }
    else{ /* 求解 pa · x=pb(mod (p-1)) */
      int x,y;
      int gcd=exgcd(pa,p-1,x,y);
      if(pb%gcd!=0){
        fout<<-1;
        cout<<endl;
        continue;
      }
      int B=(p-1)/gcd;pa/=gcd;
      x*=pb/gcd;
      fout<<((x%B+B)%B);
    }
    cout<<endl;
  }
  cin.get(),cin.get();
  return 0;
}
```

二分分数树（Stern-Brocot Tree）

```cpp
//Author:CookiC
//未做模板调整，请自行调整
#include <cmath>
#define LL long long
#define LD long double

void SternBrocot(LD X, LL &A, LL &B) {
  A=X+0.5;
  B=1;
  if(A==X)
    return;
  LL la=X, lb=1, ra=X+1, rb=1;
  long double C=A, a, b, c;
  do {
    a = la+ra;
    b = lb+rb;
    c = a/b;
    if(std::abs(C-X) > std::abs(c-X)) {
      A=a;
      B=b;
      C=c;
      if(std::abs(X-C) < 1e-10) {
        break;
      }
    }
    if(X<c) {
      ra=a;
      rb=b;
    } else {
      la=a;
      lb=b;
    }
  } while(lb+rb<=1e5);
}
```

二次剩余

```cpp
//求解 X^2==n(mod p)
ll P;

inline ll Pow(ll a,ll b){
  ll ret=1;
  for (;b;b>>=1,a=a*a%P)
    if (b&1)
      ret=ret*a%P;
  return ret;
```

```cpp
}

inline ll legendre(ll a){
  return Pow(a,(P-1)>>1);
}

struct abcd{
  ll a,b,w; //a+b*sqrt(w)
  abcd(ll a=0,ll b=0,ll w=0):a(a),b(b),w(w) { }
  friend abcd operator *(abcd A,abcd B){
    return abcd((A.a*B.a%P+A.b*B.b%P*A.w%P)%P,(A.a*B.b%P+A.b*B.a%P)%P,A.w);
  }
};

inline abcd Pow(abcd a,int b){
  abcd ret=abcd(1,0,a.w);
  for (;b;b>>=1,a=a*a)
    if (b&1)
      ret=ret*a;
  return ret;
}

inline ll Solve(ll n,ll p){
  P=p;
  if (P==2) return 1;
  if (legendre(n)==P-1) return -1;
  ll a,w;
  while (1){
    a=rand()%P;
    w=((a*a-n)%P+P)%P;
    if (legendre(w)==P-1) break;
  }
  return Pow(abcd(a,1,w),(P+1)>>1).a;
}
```

二次剩余

```cpp
/*
 * Author: Simon
 * 功能: 求解 x^2=n(mod p), 即 x=sqrt(n)(mod p)
 * 复杂度: O(sqrt(p))
 */

/* 类似复数 单位元为 w(复数的单位元为-1)*/
struct Complex {
  int x, y, w;
  Complex() {}
  Complex(int x, int y, int w) : x(x), y(y), w(w) {}
};
/* 类复数乘法 */
Complex mul(Complex a, Complex b, int p) {
  Complex ans;
  ans.x = (a.x * b.x % p + a.y * b.y % p * a.w % p) % p;
  ans.y = (a.x * b.y % p + a.y * b.x % p) % p;
  ans.w = a.w;
  return ans;
}
/* 类复数快速幂 */
Complex Complexfpow(Complex a, int b, int mod) {
  Complex ans = Complex(1, 0, a.w);
  while (b) {
    if (b & 1) ans = mul(ans, a, mod);
    a = mul(a, a, mod);
    b >>= 1;
  }
  return ans;
}
int fpow(int a, int b, int mod) {
  int ans = 1;
  a %= mod;
  while (b) {
    if (b & 1) (ans *= a) %= mod;
    (a *= a) %= mod;
    b >>= 1;
  }
  return ans;
}
/* 求解 x^2=n(mod p) */
int solve(int n, int p) {
  n %= p;
  if (n == 0) return 0;
  if (p == 2) return n;
  if (fpow(n, (p - 1) / 2, p) == p - 1) return -1; /* 勒让德定理判断 n 不是
    ↪ p 的二次剩余 */
  mt19937 rnd(time(0));
  int a, t, w;
  do {
    a = rnd() % p;
    t = a * a - n;
    w = (t % p + p) % p;                    /* 构造 w=a^2-n */
  } while (fpow(w, (p - 1) / 2, p) != p - 1); /* 找到一个 w 不是 p 的二次剩
    ↪ 余 */
  Complex ans = Complex(a, 1, w);
  ans = Complexfpow(ans, (p + 1) / 2, p); /* 答案为 (a+w)^{(p+1)/2} */
  return ans.x;
}
int exgcd(int a,int b,int &x,int &y){
  if(b==0){
    x=1,y=0;
```

```
    return a;
  }
  int g=exgcd(b,a%b,y,x);
  y-=a/b*x;
  return g;
}
/* 求解 x^2=n (mod p^k) */
int exsolve(int n,int p,int k,int pk){
  int r=solve(n,p);/* 先求出 x^2=n (mod p) */
  if(r==-1) return -1;
  Complex ans=Complex(r,1,n);
  ans=Complexfpow(ans,k,pk);/* 求出 (r+sqrt(n))^k=t+u · sqrt(n) */
  int a=ans.y,b=pk,c=ans.x,x,y;
  int g=exgcd(a,b,x,y);/* 求解 u · x=t (mod p^k) */
  if(c%g!=0) return -1;
  a/=g;b/=g;
  x*=c/g;
  return (x%b+b)%b;
}
```

超级乘方

```
/*
 * Author: Simon
 * 功能: 求 a^a^a^a^...^a mod m, 总共 b 个
 * 复杂度: O(log(m))
 */
#include<bits/stdc++.h>
using namespace std;
typedef int Int;
#define int long long
#define INF 0x3f3f3f3f
#define maxn 1000005
int Phi(int x) {
  int ans = x;
  for (int i = 2; i * i <= x; i++) {
    if (x % i == 0) {
      ans = ans / i * (i - 1);
      while (x % i == 0) x /= i;
    }
  }
  if (x != 1) ans = ans / x * (x - 1);
  return ans;
}
int fpow(int a,int b,int mod){
  a%=mod;int ans=1;
  while(b){
    if(b&1) (ans*=a)%=mod;
    (a*=a)%=mod;
    b>>=1;
  }
  return ans%mod;
}
int gcd(int a,int b){
  return b==0?a:gcd(b,a%b);
}
/* 判断 a^b mod p 中 b 是否小于 p */
bool check(int a,int b,int p){
  int ans=1;
  if(a>=p) return 0;
  for(int i=1;i<=b;i++){
    if(ans>=20) return 0;/*p 最大 1e6, 所以 ans>=20 则肯定大于 p */
    ans=fpow(a,ans,1e18);
    if(ans>=p) return 0;
  }
  return 1;
}
/* 递归欧拉降幂 */
int f(int a,int b,int m){
  if(m==1) return 0;
  if(b<=1) return fpow(a,b,m);
  int p=Phi(m);
  int t=f(a,b-1,p);/*f(a,b,m)=a^{f(a,b-1,p) mod 待定} mod m */
  int g=gcd(a,m);
  if(g==1/*gcd(a,m)=1 */||check(a,b-1,p)/*f(a,b-1,INF)<p */) return
    ↪ fpow(a,t,m); //扩展欧拉定理
  else/*f(a,b-1,INF)>=p */ return fpow(a,t+p,m);
}
Int main(){
  ios::sync_with_stdio(false);
  cin.tie(0);
  int T;cin>>T;
  while(T--){
    int a,b,m;cin>>a>>b>>m;
    /*f(a,b,m)=a^a^a^...^a mod m, 总共 b 个 a */
    cout<<f(a,b,m)%m<<endl;
  }
  return 0;
}
```

计算莫比乌斯函数

```
bool vis[maxn];
int prim[maxn];
int mu[maxn];
int cnt;

void get_mu(int n){
```

```cpp
    mu[1]=1;
    for(int i=2;i<=n;i++){
        if(!vis[i]){
            prim[++cnt]=i;
            mu[i]=-1;
        }
        for(int j=1;j<=cnt && prim[j]*i<=n;j++){
            vis[prim[j]*i]=1;
            if(i%prim[j]==0) break;
            else mu[i*prim[j]]=-mu[i];
        }
    }
}
```

区间中与 p 互质的数之和

```cpp
/*
 * Author: Simon
 * 复杂度: O(2^k+sqrt(n)) 其中 k 为 n 中不同的质因数的个数, n<=10^5 时, k 最大为 5
 * 功能: 容斥求 [1,n] 中与 p 互质的数的和
 */
/* 求 n 的质因数 */
void solve(int n){ /* 需预处理欧拉筛 */
    fac.clear();
    for(int i=1;i<=cnt&&prime[i]*prime[i]<=n;i++){
        if(n%prime[i]==0){
            fac.push_back(prime[i]);
            while(n%prime[i]==0) n/=prime[i];
        }
    }
    if(n>1) fac.push_back(n);
}
/* 求和公式 */
int sum_1(int n){
    return n*(n+1)/2;
}
/* 容斥求 [1,n] 中与 p 互质的数的和 */
int cal(int n,int p){
    solve(p); /* 求出 p 的质因数 */
    int ans=0;
    for(int i=1;i<(1<<fac.size());i++){ /* 枚举子集 */
        int num=0,lcm=1;
        for(int j=0;j<fac.size();j++){
            if(i>>j&1){
                num++;lcm*=fac[j];
            }
        }
        if(num&1) ans+=sum_1(n)-sum_1(n/lcm)*lcm; /* 总和减去不互质的数的和 */
        else ans-=sum_1(n)-sum_1(n/lcm)*lcm;
    }
    return ans;
}
```

杜教筛

```cpp
int DuJiao(int n)// 杜教筛--欧拉函数之和
{
    if(n<maxn) return Phi[n]; //欧拉函数前缀和
    if(mp[n]!=-1) return mp[n];
    int sum=0,z=n%mod;
    // for(int l=2,r;l<=n;l=r+1) // #version 1
    // {
    //     r=n/(n/l);
    //     sum+=DuJiao(n/l)*(r-l+1);
    //     sum%=mod;
    // }
    for(int i=1;i*i<=n;i++) // #vsesion 2-----对每一个 i=[2...n] 求 sum[phi(1)+...+phi(n/i)]
    {
        sum+=DuJiao(i)*(n/i-n/(i+1));
        sum%=mod;
        int x=n/i; //x 为值, 枚举 i 求 x;
        if(x==i||i==1) continue;
        sum+=DuJiao(x)*(n/x-n/(x+1));
        sum%=mod;
    }
    sum=((z*(z+1)%mod*inv2%mod)%mod-sum%mod+mod)%mod; //等差数列前 n 项和-sum
    mp.insert(n,sum);//加入 HashMap
    return sum%mod;
}
```

min25 筛

```cpp
/*
 * Author: Simon
 * 复杂度: O(n^{3/4}/logn)
 * 功能: 解决一类积形函数前缀和问题
 * 适用条件: 在质数处表达式为多项式, 在质数的高次幂处可以快速求值
 */
#include<bits/stdc++.h>
using namespace std;
typedef int Int;
#define int long long
#define INF 0x3f3f3f3f
#define maxn 1000005
const int mod=1e9+7;
```

```cpp
int prime[maxn],cnt=0,w[maxn],g[maxn],h[maxn],m=0;
int id1[maxn],id2[maxn],Sqr,sp[maxn];
bool vis[maxn]={1,1};
void Euler(int n){
  for(int i=2;i<=n;i++){
    if(!vis[i]) prime[++cnt]=i,sp[cnt]=sp[cnt-1]+i;
    for(int j=1;j<=cnt&&i*prime[j]<=n;j++){
      vis[i*prime[j]]=1;
      if(i%prime[j]==0) break;
    }
  }
}
int getG(int p){
  return p;
}
int getH(int p){
  return 1;
}
int getSigmaG(int p){ /* 素数和 */
  return sp[p];
}
int getSigmaH(int p){
  return p;
}
int getF(int p,int e){
  return (p^e);
}
int fpow(int a,int b,int mod){
  int ans=1;a%=mod;
  while(b){
    if(b&1) (ans*=a)%=mod;
    (a*=a)%=mod;
    b>>=1;
  }
  return ans;
}
int S(int x,int y,int n){
  if (x<=1||prime[y]>x) return 0;
  int k=(x<=Sqr)?id1[x]:id2[n/x];
  int res=((1LL*g[k]-h[k])-(getSigmaG(y-1)-getSigmaH(y-1)))%mod;/*g(n,|P|)-sigma(f(P_i)) */
  res=(res+mod)%mod;
  if (y==1) res+=2; //特判。
  for (int i=y;i<=cnt&&1LL*prime[i]*prime[i]<=x;++i){
    int p1=prime[i],p2=1LL*prime[i]*prime[i];
    for (int e=1;p2<=x;++e,p1=p2,p2*=prime[i])
      (res+=(1LL*S(x/p1,i+1,n)*getF(prime[i],e)%mod+getF(prime[i],(e+1)))%mod)%=mod;
  }
  return res;
}
Int main()
{
  ios::sync_with_stdio(false);
  cin.tie(0);
  int n;cin>>n;
  Sqr=sqrt(n);Euler(Sqr);
  for(int i=1,j;i<=n;i=j+1){ /*f(i)=g(i)-h(i) */
    j=n/(n/i);
    w[++m]=n/i; /* 预处理离散化 xk=n/i */
    h[m]=(w[m]-1)%mod; /*h(m,0) 即 h 函数的前缀和减去 h(1) */
    g[m]=((w[m]+1)%mod*w[m]%mod*fpow(2,mod-2,mod)%mod-1+mod)%mod; /*g(m,0) 即 g 函数的前缀和减去 g(1)*/
    w[m]<=Sqr?id1[w[m]]=m:id2[n/w[m]]=m;
  }
  for(int j=1;j<=cnt;j++){
    for(int i=1;i<=m&&prime[j]*prime[j]<=w[i];i++){
      int k=w[i]/prime[j]<=Sqr?id1[w[i]/prime[j]]:id2[n/(w[i]/prime[j])];
      (g[i]-=getG(prime[j])%mod*(g[k]-getSigmaG(j-1))%mod+mod)%=mod;/* 根据转移方程即 g(i,j)-=f(P_j) · g(n/P_j,j-1)-sigma(f(P_i)) */
      (h[i]-=getH(prime[j])%mod*(h[k]-getSigmaH(j-1))%mod+mod)%=mod; /*h 函数转移同上 */
    }
  }
  cout<<((S(n,1,n)+1)%mod+mod)%mod<<endl;
  cin.get(),cin.get();
  return 0;
}
```

常用公式

1. 约数定理：若 $n=\prod_{i=1}^{k} p_i^{a_i}$，则
   (a) 约数个数 $f(n)=\prod_{i=1}^{k}(a_i+1)$
   (b) 约数和 $g(n)=\prod_{i=1}^{k}(\sum_{j=0}^{a_i} p_i^j)$
2. 小于 n 且互素的数之和为 $n\varphi(n)/2$
3. 若 $gcd(n,i)=1$，则 $gcd(n,n-i)=1(1\le i\le n)$
4. 错排公式：$D(n)=(n-1)(D(n-2)+D(n-1))=\sum_{i=2}^{n}\frac{(-1)^k n!}{k!}=[\frac{n!}{e}+0.5]$
5. 威尔逊定理：$p\ is\ prime\ \Rightarrow (p-1)!\equiv -1(mod\ p)$
6. 欧拉定理：$gcd(a,n)=1\Rightarrow a^{\varphi(n)}\equiv 1(mod\ n)$
7. 欧拉定理推广：$gcd(n,p)=1\Rightarrow a^n\equiv a^{n\%\varphi(p)}(mod\ p)$
8. 素数定理：对于不大于 n 的素数个数 $\pi(n)$，$\lim_{n\to\infty}\pi(n)=\frac{n}{\ln n}$

9. 位数公式：正整数 x 的位数 $N = log10(n) + 1$

10. 斯特灵公式 $n! \approx \sqrt{2\pi n}(\frac{n}{e})^n$

11. 设 $a > 1, m, n > 0$, 则 $gcd(a^m - 1, a^n - 1) = a^{gcd(m,n)} - 1$

12. 设 $a > b, gcd(a, b) = 1$, 则 $gcd(a^m - b^m, a^n - b^n) = a^{gcd(m,n)} - b^{gcd(m,n)}$

$$G = gcd(C_n^1, C_n^2, ..., C_n^{n-1}) = \begin{cases} n, & n \text{ is prime} \\ 1, & n \text{ has multy prime factors} \\ p, & n \text{ has single prime factor } p \end{cases}$$

$gcd(Fib(m), Fib(n)) = Fib(gcd(m, n))$

13. 若 $gcd(m, n) = 1$, 则:

    (a) 最大不能组合的数为 $m * n - m - n$

    (b) 不能组合数个数 $N = \frac{(m-1)(n-1)}{2}$

14. $(n + 1)lcm(C_n^0, C_n^1, ..., C_n^{n-1}, C_n^n) = lcm(1, 2, ..., n + 1)$

15. 若 p 为素数，则 $(x + y + ... + w)^p \equiv x^p + y^p + ... + w^p (mod\ p)$

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

    $h(0) = h(1) = 1, h(n) = \frac{(4n-2)h(n-1)}{n+1} = \frac{C_{2n}^n}{n+1} = C_{2n}^n - C_{2n}^{n-1}$

17. 伯努利数：$B_n = -\frac{1}{n+1}\sum_{i=0}^{n-1} C_{n+1}^i B_i$

$$\sum_{i=1}^{n} i^k = \frac{1}{k+1}\sum_{i=1}^{k+1} C_{k+1}^i B_{k+1-i}(n+1)^i$$

18. FFT 常用素数

$r\ 2^k + 1$	r	k	g
3	1	1	2
5	1	2	2
17	1	4	3
97	3	5	5
193	3	6	5
257	1	8	3
7681	15	9	17
12289	3	12	11
40961	5	13	3
65537	1	16	3
786433	3	18	10
5767169	11	19	3
7340033	7	20	3
23068673	11	21	3
104857601	25	22	3
167772161	5	25	3
469762049	7	26	3
998244353	119	23	3
1004535809	479	21	3
2013265921	15	27	31
2281701377	17	27	3
3221225473	3	30	5
75161927681	35	31	3
77309411329	9	33	7
206158430209	3	36	22
2061584302081	15	37	7
2748779069441	5	39	3
6597069766657	3	41	5
39582418599937	9	42	5
79164837199873	9	43	5
263882790666241	15	44	7
1231453023109121	35	45	3
1337006139375617	19	46	3
3799912185593857	27	47	5
4222124650659841	15	48	19
7881299347898369	7	50	6
31525197391593473	7	52	3
180143985094819841	5	55	6
1945555039024054273	27	56	5
4179340454199820289	29	57	3

数论公式

1. 小于 n 的 i，j，gcd(i,j)=1 的 i,j 的对数与欧拉函数的关系 $\sum_{i=1}^{n}\sum_{j=1}^{n}[gcd(i,j) = 1] = 2\sum_{i=1}^{n}\sum_{j=1}^{i}[gcd(i,j) = 1] - \sum_{i=1}^{n}[gcd(i,i) = 1] = \left(2\sum_{i=1}^{n}\varphi(i)\right) - 1$

2. 小于 n 的 i,j，且 gcd(i,j)=1 时，所有 i*j 的和与欧拉函数的关系 $\sum_{i=1}^{n} i \sum_{j=1}^{n}[gcd(i,j) = 1] \cdot j = 2\sum_{i=1}^{n} i \sum_{j=1}^{i}[gcd(i,j) = 1] \cdot j - \sum_{i=1}^{n}[gcd(i,i) = 1] \cdot i$
$= \left(2\sum_{i=1}^{n} i \frac{i \cdot \varphi(i) + [i=1]}{2}\right) - 1 = \left(\sum_{i=1}^{n} i^2 \cdot \varphi(i) + [i = 1]\right) - 1$

3. 约数，倍数之间重要的变换 $\sum_{k=1}^{n}\sum_{d|k} d \cdot k = \sum_{k=1}^{n}\sum_{d=1}^{\frac{n}{k}} d \cdot k \cdot d = \sum_{d=1}^{n}\sum_{k|d} d \cdot$

$\frac{d}{k} = \sum_{d=1}^{n} \sum_{k=1}^{\frac{n}{d}} d \cdot k \cdot d = \sum_{d=1}^{n} \sum_{d|k} d \cdot k = \sum_{k=1}^{n} \sum_{k|d} d \cdot \frac{d}{k}$

4. $C_m^n = C_{m-1}^{n-1} + C_{m-1}^{n}$
5. $C_n^m = \frac{n!}{m!(n-m)!}$
6. $C_n^m = C_n^{n-m} = C_{n-1}^{m} + C_{n-1}^{m-1}$（杨辉三角）
7. $C_n^0 + C_n^1 + C_n^2 + ... + C_n^n = \sum_{i=0}^{n} C_n^i = 2^n$
8. $C_n^0 + C_n^2 + C_n^4 + ... = C_n^1 + C_n^3 + C_n^5 + ... = 2^{n-1}$
9. $C_n^m + C_{n+1}^m + C_{n+2}^m + ... + C_{n+m}^m = \sum_{i=0}^{m} C_{n+i}^m = C_{n+m+1}^{m+1}$
10. $kC_n^k = nC_{n-1}^{k-1}$，$\frac{C_n^k}{k+1} = \frac{C_{n+1}^{k+1}}{n+1}$
11. $\sum\limits_{k=0}^{n} k * C_m^k = \sum\limits_{k=0}^{n-1} m * C_{m-1}^k$
12. $C_m^n \% p = C_{m/p}^{n/p} * C_{m\%p}^{n\%p} \% p$（Lucas 定理）
13. $C_{m+n}^k = \sum\limits_{i=0}^{k} C_m^i C_n^{k-i}$（Vandermonde 恒等式）
14. $\frac{n!}{\prod\limits_{i=1}^{k} a_i}$（有重复元素的全排列公式，$a_i$ 为第 i 种元素的个数，k 为元素种类数）

数学

杨辉三角求组合数

```
using ll = long long;
const int maxn=60;

ll c[maxn][maxn];

void gaoC() {
  int i,j;
  for (i = 0; i < maxn; i++) {
    c[i][0] = c[i][i] = 1;
  }
  for (i = 1; i < maxn; i++) {
    for(j = 1; j <= i; j++) {
      c[i][j]=c[i-1][j]+c[i-1][j-1];
//      c[i][j] %= mod;
    }
  }
}
```

C(n,m) mod p (n 很大 p 可以很大)

```
ll C(const ll &n, const ll &m, const int &pr) {
  ll ans = 1;
  for (int i = 1; i <= m; i++) {
    ll a = (n - m + i) % pr;
    ll b = i % pr;
    ans = (ans * (a * Inv(b, pr)) % pr) % pr;
  }
  return ans;
}
```

Lucas 定理

```
//C(n, m) mod p(n 很大 p 较小 (不知道能不能为非素数)
LL Lucas(LL n, LL m, const int &pr) {
  if (m == 0) return 1;
  return C(n % pr, m % pr, pr) * Lucas(n / pr, m / pr, pr) % pr;
}
```

计算从 C(n, 0) 到 C(n, p) 的值

```
//by Yuhao Du
int p;
std::vector<int> gao(int n) {
  std::vector<int> ret(p+1,0);
  if (n==0) {
    ret[0]=1;
  } else if (n%2==0) {
    std::vector<int> c = gao(n/2);
    for(int i = 0; i <= p+1; i++) {
      for(int j = 0; j <= p+1; j++) {
        if (i+j<=p) ret[i+j]+=c[i]*c[j];
      }
    }
  } else {
    std::vector<int> c = gao(n-1);
    for(int i = 0; i <= p+1; i++) {
      for(int j = 0; j <= 2; j++) {
        if (i+j<=p) ret[i+j]+=c[i];
      }
    }
  }
  return ret;
}
```

计算第一类斯特林数

```
int seq[60][maxn << 1] , ptr = 0;
long long B[maxn << 1] , C[maxn << 1];

int DFS( int l , int r ){
  if( l == r ){
    int id = ptr ++ ;
    seq[id][1] = l ;
    seq[id][0] = 1 ;
    return id;
  } else {
    int mid = l + r >> 1;
    int lid = DFS( l , mid );
    int rid = DFS( mid + 1 , r );
    ptr -= 2;
    int newid = ptr ++ ;
    int len = 1;
    while( len <= r - l + 1 ) len <<= 1;
    for(int i = 0 ; i < len ; ++ i) B[i] = seq[lid][i] , C[i] = seq[rid][i] , seq[lid][i] = seq[rid][i] = 0;
    ntt( B , len , 1 );
    ntt( C , len , 1 );
    for(int i = 0 ; i < len ; ++ i) B[i] = B[i] * C[i] % Mod;
    ntt( B , len , -1 );
    for(int i = 0 ; i < len ; ++ i) seq[newid][i] = B[i];
    return newid;
  }
}

//int id = DFS( 0 , N - 1 );
```

```
//for(int i = N ; i >= 0 ; -- i) {
//  printf( "f[%d] is %d \n" , N - i , seq[id][i] );
//}
```

Bell 数

```
/*
 * Author: Simon
 * 功能: 模 p 意义下用记忆化搜索快速求第 n 个贝尔数
 * 描述: 贝尔数为集合的划分数
 */
#include<iostream>
#include<cstring>
#include<algorithm>
#include<cstdio>
using namespace std;
#define int long long
#define maxn 1000005
#define INF 0x3f3f3f3f
int n,p;
int a[maxn],fac[maxn],inv[maxn];
int dfs(int n){
  if(a[n]) return a[n];
  if(n-p>=0&&n+1-p>=0){ /*a[n]=(a[n-p]+a[n+1-p])%p */
    return a[n]=(dfs(n-p)+dfs(n+1-p))%p;
  }
  int tmp=1,sum=0;
  for(int k=0;k<n;k++){ /* 如果 n 小于 p, 则暴力求 a[n]*/
    tmp=fac[n-1]%p*inv[k]%p*inv[n-1-k]%p;
    sum=(sum+tmp*dfs(k)%p)%p;
  }
  return a[n]=sum;
}
int fpow(int a,int b,int mod){
  int ans=1;a%=mod;
  while(b){
    if(b&1) ans=1LL*ans*a%mod;
    a=1LL*a*a%mod;
    b>>=1;
  }
  return ans;
}
signed main(){
  ios::sync_with_stdio(false);
  cin.tie(0);
  int T;cin>>T;
  while(T--){
    cin>>n>>p;
    memset(fac,0,sizeof(fac));
    memset(inv,0,sizeof(inv));
    memset(a,0,sizeof(a));
    a[0]=1;a[1]=1; /* 初始化前 2 个贝尔数 */
    int t=min(n,p-1); /* 模 p 意义下, 预处理 t 个阶乘即可 */
    fac[0]=1;for(int i=1;i<=t;i++) fac[i]=fac[i-1]*i%p;
    inv[t]=fpow(fac[t],p-2,p);
    for(int i=t;i>=1;i--){
      inv[i-1]=inv[i]*i%p;
    }
    cout<<dfs(n)<<endl;
  }
  return 0;
}
```

自适应辛普森

```
double F(double x) {
  //Simpson 公式用到的函数
}
double simpson(double a, double b) {//三点 Simpson 法, 这里要求 F 是一个全局
  ↪ 函数
  double c = a + (b - a) / 2;
  return (F(a) + 4 * F(c) + F(b))*(b - a) / 6;
}
double asr(double a, double b, double eps, double A) {//自适应 Simpson 公式
  ↪(递归过程)。已知整个区间 [a,b] 上的三点 Simpson 值 A
  double c = a + (b - a) / 2;
  double L = simpson(a, c), R = simpson(c, b);
  if (fabs(L + R - A) <= 15 * eps)return L + R + (L + R - A) / 15.0;
  return asr(a, c, eps / 2, L) + asr(c, b, eps / 2, R);
}
double asr(double a, double b, double eps) {//自适应 Simpson 公式 (主过程)
  return asr(a, b, eps, simpson(a, b));
}
```

博弈论

```
Nim Game
  最经典最基础的博弈.
  n 堆石子, 双方轮流从任意一堆石子中取出至少一个, 不能取的人输.
  对于一堆 x 个石子的情况, 容易用归纳法得到 SG(x)=x.
  所以所有石子个数的异或和为 0 是必败态, 否则为必胜态.

Bash Game
  每人最多一次只能取 m 个石子, 其他规则同 Nim Game.
  依旧数学归纳…SG(x)=xmod(m+1).

```

```
NimK Game
  每人一次可以从最多 K 堆石子中取出任意多个，其他规则同 Nim Game.
  结论：在二进制下各位上各堆石子的数字之和均为 (K+1) 的倍数的话则为必败态，否则为必胜态.
  这个证明要回到原始的方法上去.
  补：这个游戏还可以推广，即一个由 n 个子游戏组成的游戏，每次可以在最多 K 个子游戏中进行操作.
  然后只要把结论中各堆石子的个数改为各个子游戏的 SG 值即可，证明也还是一样的.

Anti-Nim Game
  似乎又叫做 Misère Nim.
  不能取的一方获胜，其他规则同 Nim Game.
  关于所谓的”Anti-SG 游戏”及”SJ 定理”贾志鹏的论文上有详细说明，不过似乎遇到并不多.
  结论是一个状态是必胜态当且仅当满足以下条件之一:
  SG 值不为 0 且至少有一堆石子数大于 1;
  SG 值为 0 且不存在石子数大于 1 的石子堆.

Fibonacci Nim
  有一堆个数为 n(n>=2) 的石子，游戏双方轮流取石子，规则如下:
    1) 先手不能在第一次把所有的石子取完，至少取 1 颗;
    2) 之后每次可以取的石子数至少为 1，至多为对手刚取的石子数的 2 倍。
  约定取走最后一个石子的人为赢家。
  结论：当 n 为 Fibonacci 数的时候，必败。

Staircase Nim
  每人一次可以从第一堆石子中取走若干个，或者从其他石子堆的一堆中取出若干个放到左边一堆里 (没有石子的石子堆不会消失)，其他规则同 Nim Game.
  这个游戏的结论比较神奇:
  当且仅当奇数编号堆的石子数异或和为 0 时为必败态.
  简单的理解是从偶数编号堆中取石子对手又可以放回到奇数编号堆中，而且不会让对手不能移动. 比较意识流，然而可以归纳证明.

Wythoff Game
  有两堆石子，双方轮流从某一堆取走若干石子或者从两堆中取走相同数目的石子，不能取的人输.
  容易推理得出对任意自然数 k，都存在唯一的一个必败态使得两堆石子数差为 k，设其为 Pk=(ak,bk)，表示石子数分别为 ak,bk(ak<=bk).
  那么 ak 为在 Pk0(k0<k) 中未出现过的最小自然数,bk=ak+k.
  数学班的说，用 Betty 定理以及显然的单调性就可以推出神奇的结论:
  ak=floor(k*5√+12),bk=floor(k*5√+32).

Take & Break
  有 n 堆石子，双方轮流取出一堆石子，然后新增两堆规模更小的石子堆 (可以没有石子)，无法操作者输.
  这个游戏似乎只能暴力 SG，知道一下就好.

树上删边游戏
  给出一个有 n 个结点的树，有一个点作为树的根节点，双方轮流从树中删去一条边边，之后不与根节点相连的部分将被移走，无法操作者输.
  结论是叶子结点的 SG 值为 0，其他结点 SG 值为其每个儿子结点 SG 值加 1 后的异或和，证明也并不复杂.

翻硬币游戏
  n 枚硬币排成一排，有的正面朝上，有的反面朝上。
  游戏者根据某些约束翻硬币 (如：每次只能翻一或两枚，或者每次只能翻连续的几枚)，但他所翻动的硬币中，最右边的必须是从正面翻到反面。
  谁不能翻谁输。

  需要先开动脑筋把游戏转化为其他的取石子游戏之类的，然后用如下定理解决:
  局面的 SG 值等于局面中每个正面朝上的棋子单一存在时的 SG 值的异或和。

无向图删边游戏
  一个无向连通图，有一个点作为图的根。
  游戏者轮流从图中删去边， 删去一条边后，不与根节点相连的部分将被移走。
  谁无路可走谁输。

  对于这个模型，有一个著名的定理——Fusion Principle:
  我们可以对无向图做如下改动：将图中的任意一个偶环缩成一个新点，任意一个奇环缩成一个新点加一个新边；所有连到原先环上的边全部改为与新点相连。这样的改动不会影响图的 SG 值。
```

SG 函数

```
/*
 * author: Simon
 * f[m]: 可改变当前状态的方式，m 为方式的种类，f[m] 要在 getSG 之前先预处理
 * sg[]:0~n 的 SG 函数值
 * mex[]: 为 x 后继状态的集合
 * 若 sg 值为正数，则先手必赢，否则若为 0，则先手必输。
 */
int f[maxn],sg[maxn],mex[maxn];
void getSG(int n/* 需要求多少个 sg 值 */,int m/* 有多少种操作方式 */){
  memset(sg,0,sizeof(sg));
  for(int i = 1; i <= n; i++){ /* 因为 SG[0] 始终等于 0，所以 i 从 1 开始 */
    memset(mex,0,sizeof(mex)); /* 每一次都要将上一状态 的 后继集合 重置 */
    for(int j = 0; f[j] <= i && j < m; j++)
      mex[sg[i-f[j]]] = 1;  /* 将后继状态的 SG 函数值进行标记 */
    for(int j = 0;; j++) if(!mex[j]){   /* 查询当前后继状态 SG 值中最小的非零值 */
```

```
      sg[i] = j;
      break;
    }
  }
}
```

异或线性基

```
//Author: Menci
struct LinearBasis {
  long long a[MAXL + 1];

  LinearBasis() {
    std::fill(a, a + MAXL + 1, 0);
  }

  LinearBasis(long long *x, int n) {
    build(x, n);
  }

  void insert(long long t) {
    for (int j = MAXL; j >= 0; j--) {
      if (!t) return;
      if (!(t & (1ll << j))) continue;

      if (a[j]) {t ^= a[j];
      } else {
        for (int k = 0; k < j; k++) {
          if (t & (1ll << k)) {
            t ^= a[k];
          }
        }
        for (int k = j + 1; k <= MAXL; k++) {
          if (a[k] & (1ll << j)) {
            a[k] ^= t;
          }
        }
        a[j] = t;
        break;
      }
    }
  }

  // 数组 x 表示集合 S，下标范围 [1...n]
  void build(long long *x, int n) {
    std::fill(a, a + MAXL + 1, 0);
    for (int i = 1; i <= n; i++) {
      insert(x[i]);
    }
  }

  long long queryMax() {
    long long res = 0;
    for (int i = 0; i <= MAXL; i++) {
      res ^= a[i];
    }
    return res;
  }

  void mergeFrom(const LinearBasis &other) {
    for (int i = 0; i <= MAXL; i++) {
      insert(other.a[i]);
    }
  }

  static LinearBasis merge(const LinearBasis &a, const LinearBasis &b) {
    LinearBasis res = a;
    for (int i = 0; i <= MAXL; i++) res.insert(b.a[i]);
    return res;
  }
};
```

java 大数开方

```
import java.math.BigInteger;

public class Main {
  static BigInteger n,mod;
  public static BigInteger Sqrt(BigInteger c) {
    if(c.compareTo(BigInteger.ONE)<=0)
      return c;
    BigInteger temp=null,x;
    x=c.shiftRight((c.bitLength()+1)/2);
    while(true) {
      temp=x;
      x=x.add(c.divide(x)).shiftRight(1);
      if(temp.equals(x)||x.add(BigInteger.ONE).equals(temp)) break;
    }
    return x;
  }
  public static boolean judge(BigInteger c) {
    BigInteger x=Sqrt(c);
    if(x.multiply(x).equals(c)) {
      return true;
    } else {
      return false;
```

```
    }
  }
}
```

单纯形法

```
// 单纯形解线性规划 by zimpha
// 给出 m 个这样的约束条件：sum(A[i]*X[i])<=B
// 求出 X 的解，在满足 X[i]>=0 的情况下，sum(C[i]*X[i]) 达到最大
#include <cstdio>
#include <cstring>
#include <algorithm>
#define fo(i,a,b) for(int i=a;i<=b;i++)
using namespace std;
typedef long double db;
const int N=25;
db a[N][N], eps=1e-9;
int id[N*2],n,m,t,x;
double ans[N*2];
bool pd;
db abs(db x) {return x<0?-x:x;}
void pivot(int l,int e) {
  swap(id[n+l],id[e]);
  db t=a[l][e];a[l][e]=1;
  fo(i,0,n) a[l][i]/=t;
  fo(i,0,m)
    if (i!=l&&abs(a[i][e])>eps) {
      db t=a[i][e];a[i][e]=0;
      fo(j,0,n) a[i][j]-=t*a[l][j];
    }
}
void prepare() {
  while (1) {
    int l=0,e=0;
    fo(i,1,m) if (a[i][0]<-eps&&(!l||(rand()&1))) l=i;
    if (!l) break;
    fo(i,1,n) if (a[l][i]<-eps&&(!e||(rand()&1))) e=i;
    if (!e) {pd=1;return;}
    pivot(l,e);
  }
}
void solve() {
  while (1) {
    int l=0,e=0;db mn=1e18;
    fo(i,1,n) if (a[0][i]>eps) {e=i;break;}
    if (!e) break;
    fo(i,1,m)
      if (a[i][e]>eps&&a[i][0]/a[i][e]<mn) {
        mn=a[i][0]/a[i][e];
        l=i;
      }
    if (!l) {pd=1;return;}
    pivot(l,e);
  }
}
int main() {
  srand(233);
  scanf("%d%d%d",&n,&m,&t);
  fo(i,1,n) scanf("%d",&x),a[0][i]=x;
  fo(i,1,m) {
    fo(j,1,n) scanf("%d",&x),a[i][j]=x;
    scanf("%d",&x);
    a[i][0]=x;
  }
  fo(i,1,n+m) id[i]=i;
  prepare();
  if (pd) { //不存在满足所有约束的解
    printf("Infeasible\n");
    return 0;
  }
  pd=0;
  solve();
  if (pd) { //对于任意的 M, 都存在一组解使得目标函数的值大于 M
    printf("Unbounded\n");
    return 0;
  }
  printf("%.15lf\n",-(double)a[0][0]);
  if (t) {
    fo(i,1,m) ans[id[i+n]]=a[i][0];
    fo(i,1,n) printf("%.15lf ",ans[i]);
  }
}
```

容斥

```
for(int i=0;i<fac.size();i++){ //容斥求 [0,m) 内，a1,a2...ak 的倍数的和，每个数只记一次
  if(vis[i]==num[i]) continue;// vis 数组为 fac[i] 这个数要用几次，这里vis[i]=1
  int n=(m-1)/fac[i];
  ans+=(1+n)*n/2*fac[i]*(vis[i]-num[i]);// num[i] 数组为 fac[i] 这个数已经用了几次，多了就要减去多用的次数
  n=vis[i]-num[i];          //用于更新已经用的次数
  for(int j=i;j<fac.size();j++){
    if(fac[j]%fac[i]==0){
      num[j]+=n; //在此题中 将所有 fac[i] 的倍数 更新已使用次数
    }
```

```
  }
}
```

矩阵快速幂

```
//Author: Simon
#define maxn 16
const int mod=1e9+7;
struct Matrix{ //矩阵类
  int m[maxn][maxn];
  Matrix(){
    for(int i=0;i<maxn;i++) for(int j=0;j<maxn;j++) m[i][j]=0;
  }
  void init(){
    for(int i=0;i<maxn;i++) m[i][i]=1;
  }
  void set(int len){ //构造矩阵，根据题目变化
    for(int i=0;i<len;i++){
      for(int j=i-1;j<=i+1;j++){
        if(j<0||j>=len) continue;
        m[i][j]=1;
      }
    }
  }
  int *operator [](int x){
    return m[x];
  }
};
Matrix operator *(Matrix a,Matrix b){ //矩阵乘法，多组数据可以加个全局变量
  ↪ len 控制矩阵大小 O(len^3)
  Matrix c;
  for(int i=0;i<maxn;i++){
    for(int j=0;j<maxn;j++){
      for(int k=0;k<maxn;k++){
        c[i][j]=(c[i][j]+a[i][k]*b[k][j])%mod;
      }
    }
  }
  return c;
}
Matrix fpow(Matrix a,int b){ //矩阵快速幂
  Matrix c;c.init();
  while(b){
    if(b&1) c=c*a;
    a=a*a;
    b>>=1;
  }
  return c;
}
Matrix ans; //答案矩阵，仅第一列有用，ans[0][0]=f(n)
void init(int x){ //若题目类型为分段求和，则可能使用
  for(int i=x;i<maxn;i++){
    ans[i][0]=0;
  }
}
```

线性基

```
/*
 * Author: Simon
 * 功能：线性基的插入
 */
int a[maxn];
void insert(int val){
  for(int i=60;i>=0;i--){
    if(val&(1LL<<i)){
      if(!a[i]){a[i]=val;break;}
      else val^=a[i];
    }
  }
}
/*
 * Author: Simon
 * 功能：将上三角矩阵线性基化为对角矩阵形式
 */
int p[maxn],cnt=0;
void rebuild(int n){
  for(int i=60;i>=0;i--){
    for(int j=i-1;j>=0;j--){
      if(a[i]&(1LL<<j)) a[i]^=a[j];
    }
  }cnt=0;
  for(int i=60;~i;i--){
    if(a[i]) p[cnt++]=a[i];
  }
}
/*
 * Author: Simon
 * 功能：线性基的合并
 */
void merge(int *g){
  for(int i=0;i<=60;i++){
    if(g[i]) insert(g[i]);
  }
}
/*
 * Author: Simon
 * 功能：线性基查询最大值
```

```
 */
int query(){
  int ans=0;
  for(int i=60;~i;i--){
    if(ans^a[i])>ans) ans^=a[i];
  }
  return ans;
}
/*
 * Author: Simon
 * 功能: 线性基查询最小值
 */
int query(){
  for(int i=0;i<=60;i++){
    if(a[i]) return a[i];
  }
  return 0;
}
/*
 * Author: Simon
 * 功能: 线性基化查询第 k 小值 (无重复)
 */
int query(int k,int cnt) /* 需先化为对角矩阵 */
  int ans=0;
  for(int i=0;i<cnt;i++){}
    if(k>>i&1LL) ans^=p[i];
  }
}
/*
 * Author: Simon
 * 功能: 在线查询区间最大异或和
 * 按右端点分类, 构造 n 个线性基, 并记录每个值插入的位置
 * 同时保证插入时靠右的值具有优先插入权
 */
int a[maxn],base[maxn][25]/* 最大位置到 i 的线性基 */,pos[maxn][25];
void insert(int val,int p){
  int k=p;
  for(int i=0;i<=20;i++) base[p][i]=base[p-1][i],pos[p][i]=pos[p-1][i]; /* 复制最大位置为 p-1 的线性基, 在此基础上插入 */
  for(int i=20;i>=0;i--) if(val>>i&1){
    if(!base[p][i]){
      base[p][i]=val;
      pos[p][i]=k;
      break;
    }
    if (k > pos[p][i]) {
      swap(pos[p][i], k); /* 位置大的优先, 注意交换的位置是 k, 不是原数 p */
      swap(base[p][i], val);
    }
    val ^= base[p][i];
  }
}
int query(int l,int r){
  int ans=0;
  for(int i=20;i>=0;i--){
    if((ans^base[r][i])>ans&&pos[r][i]>=l) ans^=base[r][i];
  }
  return ans;
}
/*
 * Author: Simon
 * 功能: 线性基查询 q 在 n 个数的任意组合的异或值的排名 (有重复)
 * n 个数, 它们随意组合的异或值有 2^n 个数, 但是去重后只有 2^r 个数, r 为线性基的个数。
 * 所以对于每个数它们出现的次数一定相同, 为 2^(n-r) 次。
 */
void rebuild(int n){ /* 线性基的重建, 将上三角矩阵转化为对角矩阵 */
  for(int i=60;i>=0;i--){
    for(int j=i-1;j>=0;j--){
      if(a[i]&(1LL<<j)) a[i]^=a[j];
    }
  }cnt=0;
  for(int i=0;i<=60;i++){
    if(a[i]) p[cnt++]=i;
  }
}
int query(int q){
  int ans=0;
  for(int i=0;i<cnt;i++) if(q&(1LL<<p[i])) ans=(ans+(1<<i));
  return ans;
}
```

多项式乘法/平方/取模

```
namespace fft {
  typedef int type;
  typedef double db;
  struct cp {
    db x, y;

    cp() { x = y = 0; }

    cp(db x, db y) : x(x), y(y) {}
  };
  inline cp operator+(cp a, cp b) { return cp(a.x + b.x, a.y + b.y); }
  inline cp operator-(cp a, cp b) { return cp(a.x - b.x, a.y - b.y); }
  inline cp operator*(cp a, cp b) { return cp(a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x); }
```

```
inline cp conj(cp a) { return cp(a.x, -a.y); }

type base = 1;
vector<cp> roots = {{0, 0},
          {1, 0}};
vector<type> rev = {0, 1};
const db PI = acosl(-1.0);
void ensure_base(type nbase) {
  if (nbase <= base) {
    return;
  }
  rev.resize(static_cast<unsigned long>(1 << nbase));
  for (type i = 0; i < (1 << nbase); i++) {
    rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (nbase - 1));
  }
  roots.resize(static_cast<unsigned long>(1 << nbase));
  while (base < nbase) {
    db angle = 2 * PI / (1 << (base + 1));
    for (type i = 1 << (base - 1); i < (1 << base); i++) {
      roots[i << 1] = roots[i];
      db angle_i = angle * (2 * i + 1 - (1 << base));
      roots[(i << 1) + 1] = cp(cos(angle_i), sin(angle_i));
    }
    base++;
  }
}
void fft(vector<cp> &a, type n = -1) {
  if (n == -1) {
    n = a.size();
  }
  assert((n & (n - 1)) == 0);
  type zeros = __builtin_ctz(n);
  ensure_base(zeros);
  type shift = base - zeros;
  for (type i = 0; i < n; i++) {
    if (i < (rev[i] >> shift)) {
      swap(a[i], a[rev[i] >> shift]);
    }
  }
  for (type k = 1; k < n; k <<= 1) {
    for (type i = 0; i < n; i += 2 * k) {
      for (type j = 0; j < k; j++) {
        cp z = a[i + j + k] * roots[j + k];
        a[i + j + k] = a[i + j] - z;
        a[i + j] = a[i + j] + z;
      }
    }
  }
}
vector<cp> fa, fb;
vector<type> multiply(vector<type> &a, vector<type> &b) {
  type need = a.size() + b.size() - 1;
  type nbase = 0;
  while ((1 << nbase) < need) nbase++;
  ensure_base(nbase);
  type sz = 1 << nbase;
  if (sz > (type) fa.size())
    fa.resize(static_cast<unsigned long>(sz));
  for (type i = 0; i < sz; i++) {
    type x = (i < (type) a.size() ? a[i] : 0);
    type y = (i < (type) b.size() ? b[i] : 0);
    fa[i] = cp(x, y);
  }
  fft(fa, sz);
  cp r(0, -0.25 / sz);
  for (type i = 0; i <= (sz >> 1); i++) {
    type j = (sz - i) & (sz - 1);
    cp z = (fa[j] * fa[j] - conj(fa[i] * fa[i])) * r;
    if (i != j) {
      fa[j] = (fa[i] * fa[i] - conj(fa[j] * fa[j])) * r;
    }
    fa[i] = z;
  }
  fft(fa, sz);
  vector<type> res(static_cast<unsigned long>(need));
  for (type i = 0; i < need; i++) {
    res[i] = fa[i].x + 0.5;
  }
  return res;
}
vector<type> multiply_mod(vector<type> &a, vector<type> &b, type m, type eq
  ↪ = 0) {
  type need = a.size() + b.size() - 1;
  type nbase = 0;
  while ((1 << nbase) < need) nbase++;
  ensure_base(nbase);
  type sz = 1 << nbase;
  if (sz > (type) fa.size()) {
    fa.resize(static_cast<unsigned long>(sz));
  }
  for (type i = 0; i < (type) a.size(); i++) {
    type x = (a[i] % m + m) % m;
    fa[i] = cp(x & ((1 << 15) - 1), x >> 15);
  }
  fill(fa.begin() + a.size(), fa.begin() + sz, cp{0, 0});
  fft(fa, sz);
  if (sz > (type) fb.size()) {
    fb.resize(static_cast<unsigned long>(sz));
  }
  if (eq) {
```

```cpp
      copy(fa.begin(), fa.begin() + sz, fb.begin());
    } else {
      for (type i = 0; i < (type) b.size(); i++) {
        type x = (b[i] % m + m) % m;
        fb[i] = cp(x & ((1 << 15) - 1), x >> 15);
      }
      fill(fb.begin() + b.size(), fb.begin() + sz, cp{0, 0});
      fft(fb, sz);
    }
    db ratio = 0.25 / sz;
    cp r2(0, -1);
    cp r3(ratio, 0);
    cp r4(0, -ratio);
    cp r5(0, 1);
    for (type i = 0; i <= (sz >> 1); i++) {
      type j = (sz - i) & (sz - 1);
      cp a1 = (fa[i] + conj(fa[j]));
      cp a2 = (fa[i] - conj(fa[j])) * r2;
      cp b1 = (fb[i] + conj(fb[j])) * r3;
      cp b2 = (fb[i] - conj(fb[j])) * r4;
      if (i != j) {
        cp c1 = (fa[j] + conj(fa[i]));
        cp c2 = (fa[j] - conj(fa[i])) * r2;
        cp d1 = (fb[j] + conj(fb[i])) * r3;
        cp d2 = (fb[j] - conj(fb[i])) * r4;
        fa[i] = c1 * d1 + c2 * d2 * r5;
        fb[i] = c1 * d2 + c2 * d1;
      }
      fa[j] = a1 * b1 + a2 * b2 * r5;
      fb[j] = a1 * b2 + a2 * b1;
    }
    fft(fa, sz);
    fft(fb, sz);
    vector<type> res(static_cast<unsigned long>(need));
    for (type i = 0; i < need; i++) {
      long long aa = fa[i].x + 0.5;
      long long bb = fb[i].x + 0.5;
      long long cc = fa[i].y + 0.5;
      res[i] = (aa + ((bb % m) << 15) + ((cc % m) << 30)) % m;
    }
    return res;
  }
  vector<type> square(vector<type> &a) {
    return multiply(a, a);
  }
  vector<type> square_mod(vector<type> &a, type m) {
    return multiply_mod(a, a, m, 1);
  }
  vector<type> kiss_me(vector<type>&b, long long k, type mod) {
    vector<type> a = b;
    vector<type> res(1, 1);
    for (; k; k >>= 1, a = square_mod(a, mod)) {
      if (k & 1) {
        res = multiply_mod(res, a, mod);
      }
    }
    return res;
  }
  pair<vector<type>, vector<type> > mul2(vector<type>&b, long long k) {
    return make_pair(kiss_me(b, k, (type)1e9 + 7), kiss_me(b, k, (type)1e9 +
      ↪ 9));
  }
  vector<vector<type> > muln(vector<type>&b, long long k, vector<int>
    ↪ mod_list) {
    vector< vector<type> > res(mod_list.size());
    for (int i = 0; i < mod_list.size(); ++i) {
      res[i] = kiss_me(b, k, mod_list[i]);
    }
    return res;
  }
};
```

拉格朗日插值

```cpp
/*
 * Author: Simon
 * 复杂度: O(n)
 * 功能: 已知多项式前 n+1 项, 求第 k 项。
 * n 次多项式的前缀和是 n+1 次多项式。
 */
int inv[maxn]/* 阶乘逆元 */,bit[maxn]/* 阶乘 */;
int ubit[maxn],subit[maxn];
void init(){
  bit[0] = 1;
  for (int i = 1; i < maxn; i++) bit[i] = 1LL * bit[i - 1] * i % mod;
  inv[maxn - 1] = fpow(bit[maxn - 1], mod - 2, mod);
  for (int i = maxn - 1; i >= 1; i--) inv[i - 1] = 1LL * inv[i] * i % mod;
}
int Lagrangian(int y[]/* 值域 */,int n/* 变量 */,int k/* 待求 y(k)*/ ){
  if(k<=n) return y[k];
  ubit[0]=subit[n]=1;
  for(int i=1;i<=n;i++){
    ubit[i]=1LL*ubit[i-1]*((k-i+1))%mod;/* ubit[i]=prod_{j\in [0,i-1]}(n-j)
      ↪ */
    subit[n-i]=1LL*subit[n-i+1]*((k-n+i-1))%mod;
      ↪ /*subit[i]=prod_{j\in[i+1,n]}(n-j) */
  }
  int ans=0;
  for(int i=0;i<=n;i++){
```

```cpp
    int s1=1LL*y[i]%mod*ubit[i]%mod*subit[i]%mod;/* 分子 */
    int s2=1LL*inv[i]%mod*inv[n-i]%mod; /* 分母 */
    ans=(1LL*ans+1LL*((n-i)&1?-1:1)*s1%mod*s2%mod)%mod;
  }
  return (ans+mod)%mod;
}
/*
 * Author: Simon
 * 复杂度: O(n^2)
 * 功能: 已知多项式任意 n 项, 求第 k 项
 */
int Lagrangian(int y[]/* 值域 */,int x[]/* 变量 */,int k/* 待求 y_k*/,int mod){
  int ans=0;
  for(int i=1;i<=n;i++){
    int s1=y[i]%mod,s2=1;
    for(int j=1;j<=n;j++){
      if(i==j) continue;
      (s1*=((k-x[j])%mod))%=mod;
      (s2*=((x[i]-x[j])%mod))%=mod;
    }
    (ans+=s1*fpow(s2,mod-2,mod)%mod)%=mod;
  }
  return (ans+mod)%mod;
}
```

快速傅里叶变换

```cpp
const double PI = acos(-1.0);
//复数结构体
struct Complex {
  double x, y; //实部和虚部 x+yi
  Complex(double _x = 0.0, double _y = 0.0) { x = _x, y = _y; }
  Complex operator-(const Complex& b) const { return Complex(x - b.x, y - b.y); }
  Complex operator+(const Complex& b) const { return Complex(x + b.x, y + b.y); }
  Complex operator*(const Complex& b) const { return Complex(x * b.x - y * b.y, x * b.y + y * b.x); }
};
/*
* 进行 FFT 和 IFFT 前的反转变换。
* 位置 i 和 (i 二进制反转后位置) 互换
* len 必须取 2 的幂
*/
void change(Complex y[], int len) {
  for (int i = 1, j = len / 2; i < len - 1; i++) {
    if (i < j) std::swap(y[i], y[j]);
    //交换互为小标反转的元素, i<j 保证交换一次
    //i 做正常的 +1, j 左反转类型的 +1, 始终保持 i 和 j 是反转的
    int k = len / 2;
    while (j >= k) j -= k, k /= 2;
    if (j < k) j += k;
  }
}

/*
* 做 FFT
* len 必须为 2^k 形式,
* on==1 时是 DFT, on==-1 时是 IDFT
*/
void fft(Complex y[], int len, int on) {
  change(y, len);
  for (int h = 2; h <= len; h <<= 1) {
    Complex wn(cos(-on * 2 * PI / h), sin(-on * 2 * PI / h));
    for (int j = 0; j < len; j += h) {
      Complex w(1, 0);
      for (int k = j; k < j + h / 2; k++) {
        Complex u = y[k];
        Complex t = w * y[k + h / 2];
        y[k] = u + t, y[k + h / 2] = u - t;
        w = w * wn;
      }
    }
  }
  if (on == -1) for (int i = 0; i < len; i++) y[i].x /= len;
}
```

快速数论变换

```cpp
// ---
// 模数 P 为费马素数, G 为 P 的原根。
// G^((P-1)/n) 具有和 w_n = e^(2iπ/n) 相似的性质。
// 具体的 P 和 G 可参考 1.11
// ---

const int mod = 119 << 23 | 1;
const int G = 3;
int wn[20];

void getwn() { //  千万不要忘记
  for (int i = 0; i < 20; i++) wn[i] = Pow(G, (mod - 1) / (1 << i), mod);
}

void change(int y[], int len) {
  for (int i = 1, j = len / 2; i < len - 1; i++) {
    if (i < j) swap(y[i], y[j]);
    int k = len / 2;
```

```
    while (j >= k) j -= k, k /= 2;
    if (j < k) j += k;
  }
}

void ntt(int y[], int len, int on) {
  change(y, len);
  for (int h = 2, id = 1; h <= len; h <<= 1, id++) {
    for (int j = 0; j < len; j += h) {
      int w = 1;
      for (int k = j; k < j + h / 2; k++) {
        int u = y[k] % mod;
        int t = 1LL * w * (y[k + h / 2] % mod) % mod;
        y[k] = (u + t) % mod, y[k + h / 2] = ((u - t) % mod + mod) % mod;
        w = 1LL * w * wn[id] % mod;
      }
    }
  } if (on == -1) {
    //  原本的除法要用逆元
    int inv = Pow(len, mod - 2, mod);
    for (int i = 1; i < len / 2; i++) swap(y[i], y[len - i]);
    for (int i = 0; i < len; i++) y[i] = 1LL * y[i] * inv % mod;
  }
}
```

快速沃尔什变换

```
void fwt(int f[], int m) {
  int n = __builtin_ctz(m);
  for (int i = 0; i < n; ++i)
    for (int j = 0; j < m; ++j)
      if (j & (1 << i)) {
        int l = f[j ^ (1 << i)], r = f[j];
        f[j ^ (1 << i)] = l + r, f[j] = l - r;
        // or: f[j] += f[j ^ (1 << i)];
        // and: f[j ^ (1 << i)] += f[j];
      }
}

void ifwt(int f[], int m) {
  int n = __builtin_ctz(m);
  for (int i = 0; i < n; ++i)
    for (int j = 0; j < m; ++j)
      if (j & (1 << i)) {
        int l = f[j ^ (1 << i)], r = f[j];
        f[j ^ (1 << i)] = (l + r) / 2, f[j] = (l - r) / 2;
        // 如果有取模需要使用逆元
        // or: f[j] -= f[j ^ (1 << i)];
        // and: f[j ^ (1 << i)] -= f[j];
      }
```

```
}
```

分治 fft

```
//dp[i] = sigma(a[j] * dp[i-j]) (j < i);
const int maxn = "Edit";
int dp[maxn], a[maxn];
Complex x[maxn<<2], y[maxn<<2];
void solve(int L, int R){
  if(L == R) return ;
  int mid = (L + R) >> 1;
  solve(L, mid);
  int len = 1, len1 = R - L + 1;
  while(len <= len1) len <<= 1;
  for(int i = 0; i < len1; ++i) x[i] = Complex(a[i], 0);
  for(int i = len1; i <= len; ++i) x[i] = Complex(0, 0);
  for(int i = L; i <= mid; ++i)
    y[i-L] = Complex(dp[i], 0);
  for(int i = mid - L + 1; i <= len; ++i) y[i] = Complex(0, 0);
  fft(x, len, 1);
  fft(y, len, 1);
  for(int i = 0; i < len; ++i) x[i] = x[i] * y[i];
  fft(x, len, -1);
  for(int i = mid + 1; i <= R; ++i){
    dp[i] += x[i-L].x + 0.5;
  }
  solve(mid + 1, R);
}
```

polya 项链染色

```
/*
 *  c 种颜色的珠子，组成长为 s 的项链，项链没有方向和起始位置
 */
int gcd(int a, int b) {
  return b ? gcd(b, a % b) : a;
}

int main(int argc, const char * argv[]) {
  int c, s;
  while (std::cin >> c >> s) {
    int k;
    long long p[64];
    p[0] = 1;                    // power of c
    for (k = 0; k < s; k++) {
      p[k + 1] = p[k] * c;
    }
    // reflection part
```

```
    long long count = s & 1 ? s * p[s / 2 + 1] : (s / 2) * (p[s / 2] + p[s /
      ↪ 2 + 1]);
    // rotation part
    for (k = 1 ; k <= s ; k++) {
      count += p[gcd(k, s)];
      count /= 2 * s;
    }
    std::cout << count << '\n';
  }
  return 0;
}
```

染色多项式

```
完全图: t(t-1)(t-2)...(t-(n-1))
有 n 个顶点的树: t(t-1)^(n-1)
环: (t-1)^n + (-1)^n(t-1)
加边减边: P(G)=P(G+e)+P(G•e)
```

错位排列递推公式

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

BBP 公式求 pi 十六进制的第 k 位

```
// BBP 算法 询问十六进制下圆周率的第 n 位
// 时间复杂度 O(nlogn)

using ll = long long;

ll remain( ll m, ll n, ll k, ll extra) {
  ll temp1=1,temp2=1;
  if (n==0) return extra%k;
  if (n==1) return (m*extra)%k;
  while(n>1) {
    temp1=m;
    temp1*=temp1;
    if(temp1>=k)temp1%=k;
    if(n%2==1)temp2=m*temp2;
    temp2%=k;
    m=temp1;
    n=n/2;
  }
  temp1=(temp1*temp2)%k;
  return (temp1*extra)%k;
}

ll remain_nex( ll m, ll n, ll k) {
  ll temp1 = 1, temp2 = 1;
  if (n == 0) return 1;
  if (n == 1) return m%k;
  while (n>1) {
    temp1 = m;
    temp1 *= temp1;
    if (temp1 >= k) temp1%=k;
    if (n % 2 == 1) temp2=m*temp2;
    temp2 %= k;
    m = temp1;
    n = n / 2;
  }
  return (temp1*temp2)%k;
}

char compute_n(int j) {
   ll m;
  long double sum=0,temp=1.0,temp1;
  int i;
  j--;
  temp1=1.0;
  for (i=0;i<=j;i++) sum=sum+remain(16,j-i,8*i+1,4)/(long double)(8.0*i+1);
  for (i=0;i<=j;i++) sum=sum-remain(16,j-i,8*i+4,2)/(long double)(8.0*i+4);
  for (i=0;i<=j;i++) sum=sum-remain_nex(16,j-i,8*i+5)/(long double)(8.0*i+5);
  for (i=0;i<=j;i++) sum=sum-remain_nex(16,j-i,8*i+6)/(long double)(8.0*i+6);
  temp=1.0;
  for (;temp>0.000001;i++) {
    temp=temp/16.0;sum=sum+(4.0/(8*i+1)-2.0/(8*i+4)-1.0/(8*i+5)-1.0/(8*i+6))*temp;
  }
  for (;sum<0;) sum=sum+16;
  m=sum;
  sum=sum-m;
  sum=sum*16;
  m=sum;
  return (char)(m<10 ? m+48: m+55);
}
```

图论

前向星

```
const int maxn = 10005;   //点的最大个数

int head[maxn], cnt=0;//head 用来表示以 i 为起点的第一条边存储的位置, cnt
  ↪ 读入边的计数器

struct Edge {
  int next; //同一起点的上一条边的储存位置
  int to; //第 i 条边的终点
  int w; //第 i 条边权重
};

Edge edge[maxn];

void addedge(int u,int v,int w) {
  edge[cnt].w = w;
  edge[cnt].to = v;
  edge[cnt].next = head[u];
  head[u] = cnt++;
}

void traverse() {
  for(int i=0; i<=n; i++) {
    for(int j=head[i]; j! =-1; j=edge[j].next) {
      std::cout << i << " " << head[i].to << " " << head[i].w << '\n';
    }
  }
}
```

并查集

```
const int maxn = 10005;   //点的最大个数

int head[maxn], cnt=0;//head 用来表示以 i 为起点的第一条边存储的位置, cnt
  ↪ 读入边的计数器

struct Edge {
  int next; //同一起点的上一条边的储存位置
  int to; //第 i 条边的终点
  int w; //第 i 条边权重
};

Edge edge[maxn];

void addedge(int u,int v,int w) {
  edge[cnt].w = w;
  edge[cnt].to = v;
  edge[cnt].next = head[u];
  head[u] = cnt++;
}

void traverse() {
  for(int i=0; i<=n; i++) {
    for(int j=head[i]; j! =-1; j=edge[j].next) {
      std::cout << i << " " << head[i].to << " " << head[i].w << '\n';
    }
  }
}
```

可撤销并查集（按秩合并）

```
#include <iostream>
#include <stack>
#include <utility>

class UFS {
  private:
    int *fa, *rank;
    std::stack <std::pair <int*, int> > stk ;
  public:
    UFS() {}
    UFS(int n) {
      fa = new int[(const int)n + 1];
      rank = new int[(const int)n + 1];
      memset (rank, 0, n+1);
      for (int i = 1; i <= n; ++i) {
        fa [i] = i;
      }
    }
    inline int find(int x) {
      while (x ^ fa[x]) {
        x = fa[x];
      }
      return x ;
    }
    inline int Join (int x, int y) {
      x = find(x), y = find(y);
      if (x == y) {
        return 0;
      }
      if (rank[x] <= rank[y]) {
        stk.push(std::make_pair (fa + x, fa[x]));
        fa[x] = y;
        if (rank[x] == rank[y]) {
```

```
          stk.push(std::make_pair (rank + y, rank[y]));
          ++rank[y];
          return 2;
        }
        return 1 ;
      }
      stk.push(std::make_pair(fa + y, fa [y]));
      return fa[y] = x, 1;
    }
    inline void Undo ( )  {
      *stk.top( ).first = stk.top( ).second ;
      stk.pop( ) ;
    }
}T;
```

Kruskal 最小生成树

```
#include <vector>
#include <algorithm>

const int maxn = "Edit";
const int maxm = "Edit";

class Kruskal {
  struct UdEdge {
    int u, v, w;
    UdEdge(){}
    UdEdge(int u,int v,int w):u(u), v(v), w(w){}
  };
  int N, M;
  UdEdge pool[maxm];
  UdEdge *E[maxm];
  int P[maxn];
  int Find(int x){
    if(P[x] == x)
      return x;
    return P[x] = Find(P[x]);
  }
  public:
  static bool cmp(const UdEdge *a, const UdEdge *b) {
    return a->w < b->w;
  }
  void Clear(int n) {
    N = n;
    M = 0;
  }
  void AddEdge(int u, int v, int w) {
    pool[M] = UdEdge(u, v, w);
    E[M] = &pool[M];
    ++M;
  }
  int Run() {
    int i, ans=0;
    for(i = 1; i <= N; ++i)
      P[i] = i;
    std::sort(E, E+M, cmp);
    for(i = 0; i < M; ++i) {
      UdEdge *e = E[i];
      int x = Find(e->u);
      int y = Find(e->v);
      if(x != y) {
        P[y] = x;
        ans += e->w;
      }
    }
    return ans;
  }
};
```

Prim 最小生成树

```
int d[maxn][maxn];
int lowc[maxn];
int vis[maxn];

int prim(int n) {
  int ans = 0;
  memset(vis, 0, sizeof(vis));
  for (int i = 2; i <= n; i++)
    lowc[i] = d[1][i];
  vis[1] = 1;
  for (int i = 1; i < n; i++) {
    int minc = INF;
    int p = -1;
    for (int j = 1; j <= n; j++) {
      if (!vis[j] && minc > lowc[j]) {
        minc = lowc[j];
        p = j;
      }
    }
    vis[p] = 1;
    ans += minc;
    for (int j = 1; j <= n; j++) {
      if (!vis[j] && lowc[j] > d[p][j])
        lowc[j] = d[p][j];
    }
  }
  return ans;
```

```
}
```

SPFA 最短路

```
#include <queue>
#include <cstring>
#include <vector>
#define maxn 10007
#define INF 0x7FFFFFFF
using namespace std;
struct Edge{
  int v,w;
  Edge(int v,int w):v(v),w(w){}
};
int d[maxn];
bool inq[maxn];
vector<Edge> G[maxn];
void SPFA(int s){
  queue<int> q;
  memset(inq,0,sizeof(inq));
  for(int i=0;i<maxn;++i)
    d[i]=INF;
  d[s]=0;
  inq[s]=1;
  q.push(s);
  int u;
  while(!q.empty()){
    u=q.front();
    q.pop();
    inq[u]=0;
    for(vector<Edge>::iterator e=G[u].begin();e!=G[u].end();++e) {
      if(d[e->v]>d[u]+e->w){
        d[e->v]=d[u]+e->w;
        if(!inq[e->v]){
          q.push(e->v);
          inq[e->v]=1;
        }
      }
      }
  }
}
```

dijkstra 最短路

```
#include <vector>
#include <queue>
#define INF 0x7FFFFFFF
#define maxn 1000
using namespace std;
class Dijkstra{
private:
  struct HeapNode{
    int u;
    int d;
    HeapNode(int u, int d) :u(u), d(d){}
    bool operator < (const HeapNode &b) const{
      return d > b.d;
    }
  };
  struct Edge{
    int v;
    int w;
    Edge(int v, int w) :v(v), w(w){}
  };
  vector<Edge>G[maxn];
  bool vis[maxn];
public:
  int d[maxn];
  void clear(int n){
    int i;
    for(i=0;i<n;++i)
      G[i].clear();
    for(i=0;i<n;++i)
      d[i] = INF;
    memset(vis, 0, sizeof(vis));
  }
  void AddEdge(int u, int v, int w){
    G[u].push_back(Edge(v, w));
  }
  void Run(int s){
    int u;
    priority_queue<HeapNode> q;
    d[s] = 0;
    q.push(HeapNode(s, 0));
    while (!q.empty()){
      u = q.top().u;
      q.pop();
      if (!vis[u]){
        vis[u] = 1;
        for (vector<Edge>::iterator e = G[u].begin(); e != G[u].end(); ++e)
          if (d[e->v] > d[u] + e->w){
            d[e->v] = d[u] + e->w;
            q.push(HeapNode(e->v, d[e->v]));
          }
      }
    }
  }
};
```

Floyd 任意两点间最短路

```
//#define inf maxn*maxw+10
for(int i = 0; i < n; i++) {
  for(int j = 0; j < n; j++) {
    d[i][j] = inf;
  }
}
d[0][0] = 0;
for(int k = 0; k < n; k++) {
  for(int i = 0; i < n; i++) {
    for(int j = 0; j < n; j++) {
      d[i][j] = std::min(d[i][j], d[i][k] + d[k][j]);
    }
  }
}
```

拓扑排序

```
#include <iostream>
#include <algorithm>
#include <queue>
#include <vector>

std::vector<int> g[maxn];
int du[maxn], n, m, l[maxn];

bool topsort() {
  std::fill(du, du+maxn, 0);
  for(int i = 0; i < n; i++) {
    for(int j = 0; j < g[i].size(); j++) {
      du[g[i][j]]++;
    }
  }
  int tot = 0;
  std::queue<int> q;
  for(int i = 0; i < n; i++) {
    if(!du[i]) {
      q.push(i);
    }
  }
  while(!q.empty()) {
    int x = q.front();
    q.pop();
    l[tot++] = x;
    for(int j = 0; j < g[x].size(); j++) {
      int t = g[x][j];
      du[t]--;
      if(!du[t]) {
        q.push(t);
      }
    }
  }
  if(tot == n) {
    return 1;
  }
  return 0;
}
```

2-SAT 问题

```
class TwoSAT{
  private:
    const static int maxm=maxn*2;

    int S[maxm],c;
    vector<int> G[maxm];

    bool DFS(int u){
      if(vis[u^1])
        return false;
      if(vis[u])
        return true;
      vis[u]=1;
      S[c++]=u;
      for(auto &v:G[u])
        if(!DFS(v))
          return false;
      return true;
    }

  public:
    int N;
    bool vis[maxm];

    void Clear(){
      for(int i=2;i<(N+1)*2;++i)
        G[i].clear();
      memset(vis,0,sizeof(bool)*(N+1)*2);
    }

    void AddClause(int x,int xv,int y,int yv){
      x=x*2+xv;
      y=y*2+yv;
      G[x].push_back(y);
      G[y].push_back(x);
    }

    bool Solve(){
```

```
      for(int i=2;i<(N+1)*2;i+=2)
        if(!vis[i]&&!vis[i+1]){
          c=0;
          if(!DFS(i)){
            while(c>0)
              vis[S[--c]]=0;
            if(!DFS(i+1))
              return false;
          }
        }
      return true;
    }
  };
```

tarjan 强连通分量

```
//written by kuangbin
const int maxn = "Edit";
const int maxm = "Edit";

struct node {
  int to, next;
} edge[maxm];

int head[maxn], tot;
int low[maxn], dfn[maxn], stack[maxn], belong[maxn];
int cur, top, scc;
bool instack[maxn];
int num[maxn];

int in[maxn], out[maxn];

void init() {
  tot = 0;
  std::fill(head, head+maxn, -1);
  std::fill(in, in+maxn, 0);
  std::fill(out, out+maxn, 0);
}

void addedge(int u, int v) {
  edge[tot].to = v;
  edge[tot].next = head[u];
  head[u] = tot++;
}

void tarjan(int u) {
  int v;
  low[u] = dfn[u] = ++cur;
  stack[top++] = u;
  instack[u] = 1;
  for (int i = head[u]; i != -1; i = edge[i].next) {
    v = edge[i].to;
    if (!dfn[v]) {
      tarjan(v);
      if (low[u] > low[v]) low[u] = low[v];
    } else if (instack[v] && low[u] > dfn[v]) {
      low[u] = dfn[v];
    }
  }
  if (low[u] == dfn[u]) {
    scc++;
    do {
      v = stack[--top];
      instack[v] = 0;
      belong[v] = scc;
      num[scc]++;
    } while (v != u);
  }
}

void solve(int n) {
  std::fill(dfn, dfn+maxn, 0);
  std::fill(instack, instack+maxn, 0);
  std::fill(num, num+maxn, 0);
  cur = scc = top = 0;
  for (int i = 1; i <= n; i++) {
    if (!dfn[i]) {
      tarjan(i);
    }
  }
}

void in_out(int n) {
  for (int u = 1; u <= n; u++) {
    for (int i = head[u]; i != -1; i = edge[i].next) {
      if (belong[u] != belong[edge[i].to]) {
        in[belong[edge[i].to]]++;
        out[belong[u]]++;
      }
    }
  }
}
```

Kosaraju 强连通分量

```
#include <vector>
#include <algorithm>

const int maxn = "Edit";
```

```cpp

class Kosaraju {
private:
  std::vector<int> s[maxn],g[maxn];
  bool vis[maxn]={0};
public:
  int st[maxn], top=0, contract[maxn]={0};
  int n, m;
  void dfs(int x){
    vis[x]=1;
    for(int i=0;i<(int)s[x].size();++i){
      if(!vis[s[x][i]])dfs(s[x][i]);
    }
    st[top++]=x;
  }
  void dfs2(int x,int k){
    if(contract[x])return;
    contract[x]=k;/*x 屬於第 k 個 contract*/
    for(int i=0;i<(int)g[x].size();++i){
      dfs2(g[x][i],k);
    }
  }
  void addedge(int a, int b) {
    s[a].push_back(b);
    g[b].push_back(a);
  }
  void kosaraju() {
    for(int i=0;i<n;++i){
      if(!vis[i]) {
        dfs(i);
      }
    }
    for(int i=top-1,t=0;i>=0;--i){
      if(!contract[st[i]]) {
        dfs2(st[i],++t);
      }
    }
  }
};
```

点双联通分量

```cpp
//Author: CookiC
#include <stack>
#include <vector>
#define maxn 1000
using namespace std;

class BCC{
private:
  int clk, cnt;
  int pre[maxn];
  stack<int> s;

  int DFS(int u,int f){
    int lowu = pre[u] = ++clk;
    int child = 0;
    s.push(u);
    for (auto it = G[u].begin(); it != G[u].end(); ++it){
      int v = *it;
      if (!pre[v]){
        s.push(u);
        ++child;
        int lowv = DFS(v, u);
        if (lowu > lowv)
          lowu = lowv;
        if (lowv >= pre[u]){
          iscut[u] = 1;
          ++cnt;
          int x;
          do{
            x = s.top();
            s.pop();
            if (num[x] != cnt)
              num[x] = cnt;
          }while (x != u);
        }
      }
      else if (pre[v] < pre[u] && v != f){
        if (lowu > pre[v])
          lowu = pre[v];
      }
    }
    if (f < 0 && child == 1)
      iscut[u] = 0;
    return lowu;
  }
public:
  vector<int> G[maxn];
  bool iscut[maxn];
  int num[maxn];

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

  void AddEdge(int u,int v){
    G[u].push_back(v);
    G[v].push_back(u);
```

```
  }

  void Find(){
    int i;
    memset(pre, 0, sizeof(pre));
    memset(iscut, 0, sizeof(iscut));
    memset(num,0,sizeof(num));
    clk = cnt = 0;
    for (i = 0; i < n; ++i)
      if (!pre[i]){
        while(!s.empty())
          s.pop();
        DFS(i,-1);
      }
  }
};
```

边双联通分量

```
//Author: XieNaoban
//在求桥的基础上修改
#include <algorithm>
#include <cstring>
#include <vector>
#include <cmath>
#include <set>

class CutEdge {
private:
  int N;
  int clk, pre[Maxn];

  int DFS(int u, int f) {
    int lowu = pre[u] = ++clk;
    for (auto e = G[u].begin(); e != G[u].end(); ++e) {
      int v = *e;
      if (!pre[v]) {
        int lowv = DFS(v, u);
        lowu = min(lowu, lowv);
        if (lowv > pre[u]) {
          Cut[u].insert(v);
          Cut[v].insert(u);
        }
      }
      else if (pre[u] > pre[v]) {
        int cnt = 0; //重复边的处理
        for (const auto &e : G[u]) if (e == v) ++cnt;
        if (cnt > 1 || v != f) {
          lowu = min(lowu, pre[v]);
        }
      }
    }
    return lowu;
  }

  void DFS2(int u, int id) {
    ID[u] = id;
    for (const auto &v : G[u]) if (!ID[v]) {
      if (Cut[u].count(v)) {
        ++Num;
        G2[id].push_back(Num);
        G2[Num].push_back(id);
        DFS2(v, Num);
      }
      else DFS2(v, id);
    }
  }

public:
  vector<int> G[Maxn];
  set<int> Cut[Maxn];

  vector<int> G2[Maxn]; //缩点后的图 (以 ID 为结点)
  int ID[Maxn]; //每个点所在的子图
  int Num; //ID 个数

  void Clear(int n) {
    N = n;
    memset(ID, 0, sizeof(ID));
    memset(pre, 0, sizeof(pre));
    for (int i = 1; i <= N; ++i) {
      G[i].clear();
      G2[i].clear();
      Cut[i].clear();
    }
    clk = 0;
    Num = 1;
  }

  void AddEdge(int u, int v) {
    G[u].push_back(v);
    G[v].push_back(u);
  }

  void Find() {
    for (int i = 1; i <= N; ++i)
      if (!pre[i])
        DFS(i, -1);
  }
```

```

  //求边双联通部分
  int BCC() { //要求先运行 Find
    DFS2(1, Num);
    return Num;
  }
};
```

求桥

```
class bcc_bridges {
  public:
    struct edge {
      int y;
      edge * next;
    };
    edge e[M], *li[N];
    int etop;
    void init() {
      memset(li, 0, sizeof(li));
      etop = 0;
    }
    inline void add_edge(int u, int v) {
      e[etop].y = v;
      e[etop].next = li[u];
      li[u] = &e[etop++];
    }
    std::vector<std::pair<int, int>> briges;
    int dfn[N],low[N];
    int clk;
    void dfs(int u, int fa) {
      dfn[u]=low[u]=++clk;
      int v;
      for (edge * t = li[u]; t; t = t->next) {
        v = t->y;
        if (!dfn[v]) {
          dfs(v,u);
          low[u]=std::min(low[u],low[v]);
          if(low[v] > dfn[u])
            briges.emplace_back(u, v); // u <-> v is a bridge
        }
        else if(dfn[v] < dfn[u] && v != fa)
          low[u]=std::min(low[u],dfn[v]);
      }
    }
    void find_bridge(int n) {
      clk = 0;
      std::fill(dfn + 1, dfn + n + 1, 0);
      std::fill(low, low + n + 1, 0);
      for (int i = 1; i <= n; ++i) {
        if (!dfn[i])
          dfs(i, 0);
      }
    }
};
```

欧拉回路

```
const int maxn = 100;

int n;
int step;
int path[maxn];

void find_path_u(int now, int mat[][maxn]) {
  for (int i=n-1; i>=0; i--) {
    while (mat[now][i]) {
      mat[now][i]--, mat[i][now]--;
      find_path_u(i, mat);
    }
  }
  path[step++] = now;
}

void find_path_d(int now, int mat[][maxn]) {
  for (int i=n-1; i>=0; i--) {
    while (mat[now][i]) {
      mat[now][i]--;
      find_path_d(i, mat);
    }
  }
  path[step++] = now;
}


int euler_circuit(int start, int mat[][maxn]) {
  step = 0;
  find_path_u(start, mat);
  // find_path_d(start, mat);
  return step;
}

int main() {

}
```

k 短路

```cpp
#include <cstdio>
#include <cstring>
#include <queue>
#include <vector>
#include <algorithm>
using namespace std;

const int maxn = 10000 + 5;
const int INF = 0x3f3f3f3f;
int s, t, k;

bool vis[maxn];
int dist[maxn];

struct Node {
  int v, c;
  Node (int _v = 0, int _c = 0) : v(_v), c(_c) {}
  bool operator < (const Node &rhs) const {
    return c + dist[v] > rhs.c + dist[rhs.v];
  }
};

struct Edge {
  int v, cost;
  Edge (int _v = 0, int _cost = 0) : v(_v), cost(_cost) {}
};

vector<Edge>E[maxn], revE[maxn];

void Dijkstra(int n, int s) {
  memset(vis, false, sizeof(vis));
  for (int i = 1; i <= n; i++) dist[i] = INF;
  priority_queue<Node>que;
  dist[s] = 0;
  que.push(Node(s, 0));
  while (!que.empty()) {
    Node tep = que.top(); que.pop();
    int u = tep.v;
    if (vis[u]) continue;
    vis[u] = true;
    for (int i = 0; i < (int)E[u].size(); i++) {
      int v = E[u][i].v;
      int cost = E[u][i].cost;
      if (!vis[v] && dist[v] > dist[u] + cost) {
        dist[v] = dist[u] + cost;
        que.push(Node(v, dist[v]));
      }
    }
  }
}

int astar(int s) {
  priority_queue<Node> que;
  que.push(Node(s, 0)); k--;
  while (!que.empty()) {
    Node pre = que.top(); que.pop();
    int u = pre.v;
    if (u == t) {
      if (k) k--;
      else return pre.c;
    }
    for (int i = 0; i < (int)revE[u].size(); i++) {
      int v = revE[u][i].v;
      int c = revE[u][i].cost;
      que.push(Node(v, pre.c + c));
    }
  }
  return -1;
}

void addedge(int u, int v, int w) {
  revE[u].push_back(Edge(v, w));
  E[v].push_back(Edge(u, w));
}

int main() {
  int n, m, u, v, w;
  while (scanf("%d%d", &n, &m) != EOF) {
    for (int i = 0; i <= n; i++) {
      E[i].clear();
      revE[i].clear();
    }
    int aaa;
    scanf("%d%d%d%d", &s, &t, &k, &aaa);
    for (int i = 0; i < m; i++) {
      scanf("%d%d%d", &u, &v, &w);
      addedge(u, v, w);
    }
    Dijkstra(n, t);
    if (dist[s] == INF) {
      printf("No Solution\n");
      continue;
    }
    if (s == t) k++;
    ans = astar(s);
  }
  return 0;
}
```

最小环

```
int min=INT_MAX;

for(k=1;k<=n;k++) {
  for(i=1;i<=n;i++) {
    for(j=1;j<=n;++j) {
      if(dist[i][j]!=INF&&map[j][k]!=INF&&map[k][i]!=INF&&dist[i][j]+dist[j][k]+map[k][i]<mindis)
        {
        mindis=min(mindis,dist[i][j]+map[j][k]+map[k][i]);
      }
    }
  }
  for(i=1;i<=n;i++) {
    for(j=1;j<=n;j++) {
      if(dist[i][k]!=INF&&dist[k][j]!=INF&&dist[i][k]+dist[k][j]<dist[i][j])
        {
        dist[i][j]=dist[i][k]+dist[k][j];
        pre[i][j]=pre[k][j];
      }
    }
  }
}
```

最小树形图

```
#include <cstdio>
#include <cmath>
#define type int

type c[mm], in[mn], w[mn], ans;
int s[mm], t[mm], id[mn], pre[mn], q[mn], vis[mn];

type Directed_MST(int root,int NV,int NE) {
  type ret=0, sum=0, tmp;
  int i, j, u, v, r;
  bool huan=1;
  for (i=0;i<=NV;++i) in[i]=0, id[i]=i, pre[i]=-1;
  while (huan) {
    for(i=0;i<=NV;++i)
      if(pre[j=id[i]]>=0) {
        if(pre[i]<0)in[i]+=w[j],id[i]=id[j];
        else in[i]+=w[i],ret+=w[i];
      }
    for(i=0;i<=NV;++i)pre[i]=-1,vis[i]=0;
    for(i=0;i<NE;++i)
      if((u=id[s[i]])!=(v=id[t[i]])&&(w[v]>(tmp=c[i]-in[t[i]])||pre[v]<0))
        pre[v]=u,w[v]=tmp;
    for(i=1;i<=NV;++i)
      if(i!=root&&id[i]==i&&pre[i]<0)return -1;
    for(pre[root]=-1,sum=i=0;i<=NV;++i)
      if(pre[i]>=0)sum+=w[i];
    for(i=huan=0;i<=NV;++i)
      if(!vis[i]) {
        r=0,j=i;
        while(j>=0&&vis[j]>=0) {
          if(vis[j]>0) {
            while(q[--r]!=j)id[q[r]]=j,vis[q[r]]=-1;
            huan=1,vis[j]=-1;
          }
          else vis[q[r++]=j]=1,j=pre[j];
        }
        while(r--)vis[q[r]]=pre[q[r]]=-1;
      }
  }
  return ret+sum;
}

int main() {
  int n,m,e,T,cas=0;
  scanf("%d",&T);
  while(T--) {
    scanf("%d%d",&n,&m),--n;
    e=0;
    while(m--)scanf("%d%d%d",&s[e],&t[e],&c[e]),e+=(s[e]!=t[e]);
    ans=Directed_MST(0,n,e);
    if(ans<0)printf("Case #%d: Possums!\n",++cas);
    else printf("Case #%d: %d\n",++cas,ans);
  }
  return 0;
}
```

次小生成树 (Prim)

```
// 0-indexed
bool vis[maxn];
int d[maxn][maxn];
int lowc[maxn];
int pre[maxn];
int Max[maxn][maxn];  // Max[i][j] 表示 i 到 j 的路径上的最大边权
bool used[maxn][maxn];
int Prim(int n) {
  int ans = 0;
  memset(vis, false, sizeof(vis));
  memset(Max, 0, sizeof(Max));
  memset(used, false, sizeof(used));
  vis[0] = true;
  pre[0] = -1;
  for (int i = 1; i < n; i++) {
```

```
    lowc[i] = d[0][i];
    pre[i] = 0;
  }
  lowc[0] = 0;
  for (int i = 1; i < n; i++) {
    int minc = INF;
    int p = -1;
    for (int j = 0; j < n; j++)
      if (!vis[j] && minc > lowc[j]) {
        minc = lowc[j];
        p = j;
      }
    if (minc == INF)return -1;
    ans += minc;
    vis[p] = true;
    used[p][pre[p]] = used[pre[p]][p] = true;
    for (int j = 0; j < n; j++) {
      if (vis[j]) Max[j][p] = Max[p][j] = max(Max[j][pre[p]], lowc[p]);
      if (!vis[j] && lowc[j] > d[p][j]) {
        lowc[j] = d[p][j];
        pre[j] = p;
      }
    }
  }
  return ans;
}
int SMST(int n, int ans) {
  int Min = INF;
  for (int i = 0; i < n; i++)
    for (int j = i + 1; j < n; j++)
      if (d[i][j] != INF && !used[i][j])
        Min = min(Min, ans + d[i][j] - Max[i][j]);
  if (Min == INF) return -1;
  return Min;
}
```

次小生成树 (Kruskal)

```
//1-indexed
struct edge {
  int s, t, w;  //从 s 到 t 权值 w
  bool vis;
  edge() {}
  edge(int s, int t, int w) :s(s), t(t), w(w) {}
  bool operator < (const edge e) const {
    return w < e.w;
  }
}e[maxm];

int pre[maxn];
int Max[maxn][maxn];  // Max[i][j] 表示从 i 到 j 路径上的最大边权

int find(int x) {
  int r = x, i = x, j;
  while (pre[r] != r)
    r = pre[r];
  while (i != r) {  // 状态压缩
    j = pre[i];
    pre[i] = r;
    i = j;
  }
  return r;
}

int kruskal(int n, int m) {  // n 为边数 m 为点数
  int lef = m - 1, ans = 0;
  memset(Max, 0, sizeof(Max));
  vector<int>v[maxn];
  for (int i = 1; i <= m; i++) {
    pre[i] = i;
    v[i].push_back(i);
  }
  sort(e + 1, e + n + 1);
  for (int i = 1; i <= n; i++) {
    int fs = find(e[i].s), ft = find(e[i].t), len1, len2;
    if (fs != ft) {
      pre[fs] = ft;
      ans += e[i].w;
      lef--; e[i].vis = true;
      len1 = v[fs].size(), len2 = v[ft].size();
      for (int j = 0; j < len1; j++)
        for (int k = 0; k < len2; k++)
          Max[v[fs][j]][v[ft][k]] = Max[v[ft][k]][v[fs][j]] = e[i].w;
      int tmp[maxn];
      for (int j = 0; j < len1; j++)
        tmp[j] = v[fs][j];
      for (int j = 0; j < len2; j++)
        v[fs].push_back(v[ft][j]);
      for (int j = 0; j < len1; j++)
        v[ft].push_back(tmp[j]);
    }
    if (!lef)break;
  }
  if (lef) ans = -1;  // 图不连通
  return ans;
}

int SMST(int n, int ans) {  // n 为边数,ans 为最小生成树权值
  int ret = INF;
```

```
  for (int i = 1; i <= n; i++)
    if (!e[i].vis)
      ret = min(ret, ans + e[i].w - Max[e[i].s][e[i].t]);
  if (ret == INF) return -1;
  return ret;
}
```

最小生成树计数

```
// 无向图，求生成树个数 Determinant 算法
ll A[maxn][maxn], B[maxn][maxn];
ll determinant(int n) {
  ll res = 1;
  for (int i = 1; i <= n; i++) {
    if (!B[i][i]) {
      bool flag = false;
      for (int j = i + 1; j <= n; j++) {
        if (B[j][i]) {
          flag = true;
          for (int k = i; k<n; k++)
            swap(B[i][k], B[j][k]);
          res = -res;
          break;
        }
      }
      if (!flag) return 0;
    }
    for (int j = i + 1; j <= n; j++) {
      while (B[j][i]) {
        ll t = B[i][i] / B[j][i];
        for (int k = i; k <= n; k++) {
          B[i][k] = B[i][k] - t * B[j][k];
          swap(B[i][k], B[j][k]);
        }
        res = -res;
      }
    }
    res *= B[i][i];
  }
  return res;
}
int main()
{
  int n, m, k;
  while (~scanf("%d%d%d", &n, &m, &k)) {
    memset(A, 0, sizeof(A));
    memset(B, 0, sizeof(B));
    for (int i = 1; i <= m; i++) {
      int a, b;
      scanf("%d%d", &a, &b);
      A[a][b] = A[b][a] = 1;
    }
    for (int i = 1; i <= n; i++) {
      for (int j = 1; j <= n; j++) {
        if (i != j && !A[i][j]) {
          B[i][i]++;
          B[i][j] = -1;
        }
      }
    }
    n--;
    ll ans = determinant(n);
    printf("%lld\n", ans);
  }
}
```

最小树形图计数

```
// 有向图最小生成树计数
struct node {
  int a, b, cost;
}edge[maxm];

int n, m, o;
ll ans, mod;
int pre[maxn], ka[maxn];
ll G[maxn][maxn], B[maxn][maxn];
bitset<maxn> vis;
vector<int> v[maxn];

bool cmp(node a, node b) { return a.cost < b.cost; }
int find(int x) { return pre[x] == x ? pre[x] : pre[x] = find(pre[x]); }

ll det(ll a[][maxn], int n) { //Matrix-Tree 定理求 Kirchhoff 矩阵
  for (int i = 0; i<n; i++)
    for (int j = 0; j<n; j++) a[i][j] %= mod;
  ll ret = 1;
  for (int i = 1; i<n; i++) {
    for (int j = i + 1; j<n; j++)
      while (a[j][i]) {
        ll t = a[i][i] / a[j][i];
        for (int k = i; k<n; k++) a[i][k] = (a[i][k] - a[j][k] * t) % mod;
        for (int k = i; k<n; k++) swap(a[i][k], a[j][k]);
        ret = -ret;
      }
    if (a[i][i] == 0) return 0;
    ret = ret * a[i][i] % mod;
  }
  return (ret + mod) % mod;
```

```
}

void Matrix_Tree() {
  for (int i = 1; i <= n; i++) { //根据访问标记找出连通分量
    if (vis[i]) {
      v[find(i)].push_back(i);
      vis[i] = 0;
    }
  }
  for (int i = 1; i <= n; i++) {
    if (v[i].size() > 1) { //枚举连通分量
      memset(B, 0, sizeof(B));
      int len = v[i].size();
      for (int a = 0; a < len; a++) {
        for (int b = a + 1; b < len; b++) {
          int la = v[i][a], lb = v[i][b];
          B[b][a] -= G[la][lb];
          B[a][b] = B[b][a];
          B[a][a] += G[la][lb];
          B[b][b] += G[la][lb];
        } //构造矩阵
      }
      ll ret = det(B, len) % mod;
      ans = ans * ret % mod;
      for (int j = 0; j < len; j++)
        pre[v[i][j]] = i;
    }
  }
  for (int i = 1; i <= n; i++) { //连通图缩点 + 初始化
    pre[i] = find(i);
    ka[i] = pre[i];
    v[i].clear();
  }
}

int main()
{
  while (scanf("%d%d%lld", &n, &m, &mod), n || m || mod) {
    for (int i = 1; i <= m; i++)
      scanf("%d%d%d", &edge[i].a, &edge[i].b, &edge[i].cost);
    sort(edge + 1, edge + m + 1, cmp);
    for (int i = 1; i <= n; i++)
      v[i].clear();
    for (int i = 1; i <= n; i++)
      pre[i] = ka[i] = i;
    vis.reset();
    memset(G, 0, sizeof(G));
    ans = 1;
    o = edge[1].cost;
    for (int i = 1; i <= m; i++) {
      int pa = find(edge[i].a), pb = find(edge[i].b);
      if (pa != pb) {
        vis[pa] = 1;
        vis[pb] = 1;
        ka[find(pa)] = find(pb);
        G[pa][pb]++;
        G[pb][pa]++;
      }
      if (i == m || edge[i + 1].cost != o) { //所有相同的边并成一组
        Matrix_Tree();
        o = edge[i + 1].cost;
      }
    }
    bool done = true;
    for (int i = 2; i <= n; i++) {
      if (ka[i] != ka[i - 1]) {
        done = false;
        break;
      }
    }
    if (!done) printf("0\n");
    else {
      ans %= mod;
      printf("%lld\n", ans);
    }
  }
  return 0;
}
```

Dinic 最大流

```
#include <queue>
#include <vector>
#include <cstring>
#include <algorithm>

const int maxn = "Edit";
const int inf = 0x7FFFFFFF;

struct Edge {
  int c, f;
  unsigned v, flip;
  Edge(unsigned v, int c, int f, unsigned flip) : v(v), c(c), f(f),
    ↪ flip(flip) {}
};

/*
*b:BFS 使用 ,
*a: 可改进量 , 不会出现负数可改进量。
```

```
*p[v]:u 到 v 的反向边，即 v 到 u 的边。*cur[u]:i 开始搜索的位置 ，此位置前
  ↪ 所有路已满载。*s: 源点。
*t: 汇点 。
*/

class Dinic {
private:
  bool b[maxn];
  int a[maxn];
  unsigned p[maxn], cur[maxn], d[maxn];
  std::vector<Edge> G[maxn];
public:
  unsigned s, t;
  void Init(unsigned n) {
    for(int i=0; i<=n; ++i)
      G[i].clear();
  }
  void AddEdge(unsigned u, unsigned v, int c) {
    G[u].push_back(Edge(v, c, 0, G[v].size()));
    G[v].push_back(Edge(u, 0, 0, G[u].size()-1)); //使用无向图时将 0 改为 c
      ↪ 即可
  }
  bool BFS() {
    unsigned u, v;
    std::queue<unsigned> q;
    memset(b, 0, sizeof(b));
    q.push(s);
    d[s] = 0;
    b[s] = 1;
    while (!q.empty()) {
      u = q.front();
      q.pop();
      for (auto it = G[u].begin(); it != G[u].end(); ++it) {
        Edge &e = *it;
        if(!b[e.v] && e.c > e.f){
          b[e.v] = 1;
          d[e.v] = d[u] + 1;
          q.push(e.v);
        }
      }
    }
    return b[t];
  }
  int DFS(unsigned u, int a){
    if(u==t || a==0)
      return a;
    int flow = 0, f;
    for (unsigned &i = cur[u]; i<G[u].size(); ++i){
      Edge &e = G[u][i];
      if (d[u]+1 == d[e.v] && (f = DFS(e.v, std::min(a, e.c - e.f))) > 0) {
        a -= f;
        e.f += f;
        G[e.v][e.flip].f -= f;
        flow += f;
        if (!a) break;
      }
    }
    return flow;
  }
  int MaxFlow(unsigned s, unsigned t){
    int flow = 0;
    this->s = s;
    this->t = t;
    while (BFS()) {
      memset(cur, 0, sizeof(cur));
      flow += DFS(s, inf);
    }
    return flow;
  }
};
```

ISAP 最大流

```
const int maxn = "Edit";
struct ISAP {
  int n, m, s, t;      //结点数，边数（包括反向弧），源点编号和汇点编号
  vector<Edge> edges;  //边表。edges[e] 和 edges[e^1] 互为反向弧
  vector<int> G[maxn]; //邻接表，G[i][j] 表示结点 i 的第 j 条边在 e 数组中
    ↪ 的序号
  bool vis[maxn];      //BFS 使用
  int d[maxn];         //起点到 i 的距离
  int cur[maxn];       //当前弧下标
  int p[maxn];         //可增广路上的一条弧
  int num[maxn];       //距离标号计数
  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.pb(Edge(from, to, cap, 0));
    edges.pb(Edge(to, from, 0, 0));
    int m = edges.size();
    G[from].pb(m - 2);
    G[to].pb(m - 1);
  }
  int Augumemt() {
    int x = t, a = INF;
    while (x != s) {
```

```
      Edge& e = edges[p[x]];
      a = min(a, e.cap - e.flow);
      x = edges[p[x]].from;
    }
    x = t;
    while (x != s) {
      edges[p[x]].flow += a;
      edges[p[x] ^ 1].flow -= a;
      x = edges[p[x]].from;
    }
    return a;
  }
  void BFS() {
    clr(vis, 0);
    clr(d, 0);
    queue<int> q;
    q.push(t);
    d[t] = 0;
    vis[t] = 1;
    while (!q.empty()) {
      int x = q.front();
      q.pop();
      int len = G[x].size();
      for (int i = 0; i < len; i++) {
        Edge& e = edges[G[x][i]];
        if (!vis[e.from] && e.cap > e.flow) {
          vis[e.from] = 1;
          d[e.from] = d[x] + 1;
          q.push(e.from);
        }
      }
    }
  }
  int Maxflow(int s, int t) {
    this->s = s;
    this->t = t;
    int flow = 0;
    BFS();
    clr(num, 0);
    for (int i = 0; i < n; i++)
      if (d[i] < INF) num[d[i]]++;
    int x = s;
    clr(cur, 0);
    while (d[s] < n) {
      if (x == t) {
        flow += Augumemt();
        x = s;
      }
      int ok = 0;
      for (int i = cur[x]; i < G[x].size(); i++) {
        Edge& e = edges[G[x][i]];
        if (e.cap > e.flow && d[x] == d[e.to] + 1) {
          ok = 1;
          p[e.to] = G[x][i];
          cur[x] = i;
          x = e.to;
          break;
        }
      }
      if (!ok) {//Retreat
        int m = n - 1;
        for (int i = 0; i < G[x].size(); i++) {
          Edge& e = edges[G[x][i]];
          if (e.cap > e.flow) m = min(m, d[e.to]);
        }
        if (--num[d[x]] == 0) break; //gap 优化
        num[d[x] = m + 1]++;
        cur[x] = 0;
        if (x != s) x = edges[p[x]].from;
      }
    }
    return flow;
  }
};
```

最小费用最大流

```
#include <iostream>
#include <vector>
#include <queue>

const int MAXE = 1000;
const int MAXN = 1000;
const int INF = 1000000;

using ii = std::pair<int, int>;

struct edge {
  int u, v, cost, cap, flow;
} E[MAXE], * pred[MAXN];

std::vector<edge *> g[MAXN];
int N, M, EE, dist[MAXN], phi[MAXN];

inline edge * opp(edge * e) {
  return E + ((e - E) ^ 1);
}

void inti() {
  for (int i = 0; i <= N; i++) {
```

```
    g[i].clear();
  }
  EE = 0;
}

void add_edge(int u, int v, int cost, int cap) {
  E[EE] = { u, v, cost, cap, 0 };
  g[u].emplace_back(E + (EE++));
  E[EE] = { v, u, -cost, 0, 0 };
  g[v].emplace_back(E + (EE++));
}

bool dijkstra(int S, int T) {
  std::fill(dist, dist + N, INF);
  std::fill(pred, pred + N, nullptr);
  std::priority_queue<ii, std::vector<ii>, std::greater<ii>> pq;
  dist[S] = 0;
  for (pq.emplace(dist[S], S); !pq.empty(); ) {
    int u = pq.top().second;
    pq.pop();
    for (auto e : g[u]) {
      if (e->cap - e->flow > 0 && dist[e->v] > dist[e->u] + e->cost +
       ↪ phi[e->u] - phi[e->v]) {
        dist[e->v] = dist[e->u] + e->cost + phi[e->u] - phi[e->v];
        pred[e->v] = e;
        pq.emplace(dist[e->v], e->v);
      }
    }
  }
  for (int i = 0; i < N; i++) {
    phi[i] = std::min(INF, phi[i] + dist[i]);
  }
  return dist[T] != INF;
}

std::pair<int, int> mincost_maxflow(int S, int T) {
  int mincost = 0, maxflow = 0;
  std::fill(phi, phi + N, 0);
  while (dijkstra(S, T)) {
    int flow = INF;
    for (edge * e = pred[T]; e; e = pred[e->u])
      flow = std::min(flow, e->cap - e->flow);
    for (edge * e = pred[T]; e; e = pred[e->u]) {
      mincost += e->cost * flow;
      e->flow += flow;
      opp(e)->flow -= flow;
    }
    maxflow += flow;
  }
  return std::make_pair(mincost, maxflow);
}
```

ZKW 费用流

```
const int inf = ~0U>>1;
const int N = "Edit";

typedef struct seg{
  int to,op,cost,nxt,f;
}seg;

seg v[N*40];

int ans =0,tot,dis[N],base[N],vis[N],ttf = 0;

int S,T; int cur[N];

void inti() {
  memset(base,0,sizeof(base));
  memset(dis,0,sizeof(dis));
  tot = 0; ans = 0; ttf = 0;
  memset(vis,0,sizeof(vis));
}

int aug(int u,int flow){
  if (u == T){
    ans += flow * dis[S];
    ttf += flow;
    return flow;
  }
  vis[u] = 1;
  int now = 0;
  for (int i = base[u];i;i = v[i].nxt){
    int x = v[i].to;
    if (vis[x] || v[i].f <= 0 || dis[u] != dis[x] + v[i].cost)
      continue;
    int tmp = aug(x,std::min(flow - now,v[i].f));
    v[i].f -= tmp; v[v[i].op].f += tmp;
    now += tmp;
    if (now == flow) return flow;
  }
  return now;
}


int modlabel() {
  int del = inf;
  for (int i = S; i <= T; i++) {
    if (vis[i]) for (int j = base[i];j;j = v[j].nxt) {
      if (v[j].f){
```

```cpp
        int x = v[j].to;
        if (!vis[x]) del = std::min(del,dis[x] + v[j].cost - dis[i]);
      }
    }
  }
  if (del == inf) {
    return 0;
  }
  for (int i = S;i <= T;i++) {
    if (vis[i]) {
      vis[i] = 0,dis[i] += del,cur[i] = base[i];
    }
  }
  return 1;
}


int zkw() {
  for (int i = S;i <= T;i++) cur[i] = base[i];
  int fl, t = 0;
  do {
    t = 0;
    while((t = aug(S,inf))) memset(vis,0,sizeof(vis));
  } while(modlabel());
  return ans;
}

void add(int x, int y, int c, int f){
  v[++tot].to = y; v[tot].op = tot + 1;
  v[tot].f = f; v[tot].cost = c;
  v[tot].nxt = base[x]; base[x] = tot;
  v[++tot].to = x; v[tot].op = tot - 1;
  v[tot].f = 0; v[tot].cost = -c;
  v[tot].nxt = base[y]; base[y] = tot;
}
```

上下界网络流

```cpp
/*
    首先建立一个源 S 和一个汇 T，一般称为附加源和附加汇。
    对于图中的每条弧 <u,v>，假设它容量上界为 c，下界 b，那么把这条边拆为三条只有上界的弧。
    一条为 <S,v>，容量为 b；
    一条为 <u,T>，容量为 b；
    一条为 <u,v>，容量为 c-b。
    其中前两条弧一般称为附加弧。
    然后对这张图跑最大流，以 S 为源，以 T 为汇，如果所有的附加弧都满流，则原图有可行流；否则就是无解。
    这时，每条非附加弧的流量加上它的容量下界，就是原图中这条弧应该有的流量。

    对于原图中的每条弧，我们把 c-b 称为它的自由流量，意思就是只要它流满了下界，这些流多少都没问题。
    既然如此，对于每条弧 <u,v>，我们强制给 v 提供 b 单位的流量，并且强制从 u 那里拿走 b 单位的流量，这一步对应着两条附加弧。
    如果这一系列强制操作能完成的话，也就是有一组可行流了。
    注意：这张图的最大流只是对应着原图的一组可行流，而不是原图的最大或最小流。
*/
using namespace std;
const int oo = (1LL << 31) - 1;
const int LEN = 1e5 + 5;
struct node {
    int x, y, l, r;
} a[LEN];
namespace ISAP {
    int flow, tot, n, m, src, tar, qh, qt, cnt, ans;
    struct edge {
        int vet, next, len;
    } E[LEN * 2];
    int dis[LEN], gap[LEN], head[LEN], cur[LEN], q[LEN], vis[LEN], IN[LEN];
    void add(int u, int v, int c) {
        E[++tot] = (edge){v, head[u], c};
        head[u] = tot;
    }
    void join(int u, int v, int c) {
        add(u, v, c);
        add(v, u, 0);
    }
    void bfs(int s) {
        qh = qt = 0;
        q[++qt] = s;
        dis[s] = 0;
        vis[s] = 1;
        while (qh < qt) {
            int u = q[++qh];
            gap[dis[u]]++;
            for (int e = head[u]; e != -1; e = E[e].next) {
                int v = E[e].vet;
                if (E[e ^ 1].len && !vis[v]) {
                    dis[v] = dis[u] + 1;
                    vis[v] = 1;
                    q[++qt] = v;
                }
            }
        }
    }
    int isap(int u, int aug) {
```

```
        if (u == tar) return aug;
        int flow = 0;
        for (int e = head[u]; e != -1; e = E[e].next) {
            int v = E[e].vet;
            if (E[e].len && dis[v] == dis[u] - 1) {
                int tmp = isap(v, min(aug - flow, E[e].len));
                E[e].len -= tmp;
                E[e ^ 1].len += tmp;
                flow += tmp;
                head[u] = e;
                if (flow == aug || dis[src] == cnt) return flow;
            }
        }
        if (!--gap[dis[u]++]) dis[src] = cnt;
        ++gap[dis[u]];
        head[u] = cur[u];
        return flow;
    }
    void init() {
        tot = -1, gap[0] = 0;
        for (int i = 1; i <= cnt; i++) {
            dis[i] = gap[i] = vis[i] = IN[i] = 0;
            head[i] = -1;
        }
    }
    int maxflow(int s, int t) {
        src = s, tar = t;
        int res = 0;
        for (int i = 1; i <= cnt; i++) cur[i] = head[i];
        bfs(tar);
        while (dis[src] < cnt) res += isap(src, oo);
        return res;
    }
}
using namespace ISAP;
int main() {
    scanf("%d %d", &n, &m);
    cnt = n;
    src = ++cnt, tar = ++cnt;
    init();
    for (int i = 1; i <= m; i++) {
        int x, y, l, r;
        scanf("%d %d %d %d", &x, &y, &l, &r);
        a[i] = (node){x, y, l, r};
        join(x, y, r - l);
        IN[y] += l, IN[x] -= l;
    }
    for (int i = 1; i <= n; i++) {
        if (IN[i] < 0) join(i, tar, -IN[i]);
        else {
            join(src, i, IN[i]);
            flow += IN[i];
        }
    }
    int ans = maxflow(src, tar);
    if (flow == ans) {
        puts("YES");
        for (int i = 1; i <= m; i++) printf("%d\n", a[i].l + E[i * 2 - 1].len);
    } else puts("NO");
    return 0;
}

/*
    先来看有源汇可行流
    建模方法：
    建立弧 <t,s>，容量下界为 0，上界为 oo。
    然后对这个新图（实际上只是比原图多了一条边）按照无源汇可行流的方法建模，
    如果所有附加弧满流，则存在可行流。
    求原图中每条边对应的实际流量的方法，同无源汇可行流，只是忽略掉弧 <t,s> 就好。
    而且这时候弧 <t,s> 的流量就是原图的总流量。
    理解方法：
    有源汇相比无源汇的不同就在于，源和汇是不满足流量平衡的，那么连接 <t,s> 之后，
    源和汇也满足了流量平衡，就可以直接按照无源汇的方式建模。
    注意：这张图的最大流只是对应着原图的一组可行流，而不是原图的最大或最小流。

    有源汇最大流
    建模方法：
    首先按照有源汇可行流的方法建模，如果不存在可行流，更别提什么最大流了。
    如果存在可行流，那么在运行过有源汇可行流的图上（就是已经存在流量的那张图，流量不要清零），
    跑一遍从 s 到 t 的最大流（这里的 s 和 t 是原图的源和汇，不是附加源和附加汇），就是原图的最大流。
    理解方法：
    为什么要在那个已经有了流量的图上跑最大流？因为那张图保证了每条弧的容量下界，在这张图上跑最大流，
    实际上就是在容量下界全部满足的前提下尽量多得获得“自由流量”。
    注意，在这张已经存在流量的图上，弧 <t,s> 也是存在流量的，千万不要忽略这条弧。
    因为它的相反弧 <s,t> 的流量为 <t,s> 的流量的相反数，且 <s,t> 的容量为 0，所以这部分的流量也是会被算上的。
*/
using namespace std;
typedef long long ll;
```

```
const int LEN = 1e5 + 5;
const int oo = (1LL << 31) - 1;
namespace DINIC {
    int tot, n, m, src, tar, qh, qt, cnt, s, t, S, T;
    int ans, flow;
    struct edge {
        int vet, next, len;
    } E[LEN * 2];
    int dis[LEN], gap[LEN], head[LEN], cur[LEN], q[LEN], vis[LEN], IN[LEN];
    void add(int u, int v, int c) {
        E[++tot] = (edge){v, head[u], c};
        head[u] = tot;
    }
    void join(int u, int v, int c) {
        add(u, v, c);
        add(v, u, 0);
    }
    void init() {
        tot = -1;
        for (int i = 1; i <= cnt; i++) head[i] = -1;
    }
    bool bfs() {
        for (int i = 1; i <= cnt; i++) dis[i] = 0;
        qh = qt = 0;
        q[++qt] = src;
        dis[src] = 1;
        while (qh < qt) {
            int u = q[++qh];
            for (int e = head[u]; e != -1; e = E[e].next) {
                int v = E[e].vet;
                if (E[e].len && !dis[v]) {
                    dis[v] = dis[u] + 1;
                    if (v == tar) return 1;
                    q[++qt] = v;
                }
            }
        }
        return dis[tar];
    }
    int dfs(int u, int aug) {
        if (u == tar || !aug) return aug;
        int tmp = 0;
        for (int &e = cur[u]; e != -1; e = E[e].next) {
            int v = E[e].vet;
            if (dis[v] == dis[u] + 1) {
                if (tmp = dfs(v, min(aug, E[e].len))) {
                    E[e].len -= tmp;
                    E[e ^ 1].len += tmp;
                    return tmp;
                }
            }
        }
        return 0;
    }
    int maxflow(int s, int t) {
        src = s, tar = t;
        int res = 0, flow = 0;
        while (bfs()) {
            for (int i = 1; i <= cnt; i++) cur[i] = head[i];
            while (flow = dfs(src, oo)) res += flow;
        }
        return res;
    }
}
using namespace DINIC;
int main() {
    scanf("%d %d %d %d", &n, &m, &s, &t);
    cnt = n;
    S = ++cnt, T = ++cnt;
    init();
    for (int i = 1; i <= m; i++) {
        int x, y, l, r;
        scanf("%d %d %d %d", &x, &y, &l, &r);
        join(x, y, r - l);
        IN[y] += l, IN[x] -= l;
    }
    for (int i = 1; i <= n; i++) {
        if (IN[i] < 0) join(i, T, -IN[i]);
        else if (IN[i] > 0) {
            flow += IN[i];
            join(S, i, IN[i]);
        }
    }
    join(t, s, oo);
    ans = maxflow(S, T);
    if (ans != flow) puts("please go home to sleep");
    else {
        ans = maxflow(s, t);
        printf("%lld\n", ans);
    }
    return 0;
}

/*
    先来看有源汇可行流
    建模方法:
    建立弧 <t,s>, 容量下界为 0, 上界为 oo。
    然后对这个新图(实际上只是比原图多了一条边)按照无源汇可行流的方法建模,
    如果所有附加弧满流,则存在可行流。
```

```
    求原图中每条边对应的实际流量的方法，同无源汇可行流，只是忽略掉弧 <t,s> 就好。
    而且这时候弧 <t,s> 的流量就是原图的总流量。
    理解方法：
    有源汇相比无源汇的不同就在于，源和汇是不满足流量平衡的，那么连接 <t,s> 之后，
    源和汇也满足了流量平衡，就可以直接按照无源汇的方式建模。
    注意：这张图的最大流只是对应着原图的一组可行流，而不是原图的最大或最小流。

    有源汇最小流
    有源汇最小流的常见建模方法比较多，我就只说我常用的一种。
    建模方法：
    首先按照有源汇可行流的方法建模，但是不要建立 <t,s> 这条弧。
    然后在这个图上，跑从附加源 ss 到附加汇 tt 的最大流。
    这时候再添加弧 <t,s>，下界为 0，上界 oo。
    在现在的这张图上，从 ss 到 tt 的最大流，就是原图的最小流。
    理解方法：
    我们前面提到过，有源汇可行流的流量只是对应一组可行流，并不是最大或者最小流。
    并且在跑完有源汇可行流之后，弧 <t,s> 的流量就是原图的流量。
    从这个角度入手，我们想让弧 <t,s> 的流量尽量小，就要尽量多的消耗掉那些“本来不需要经过 <t,s>” 的流量。
    于是我们在添加 <t,s> 之前，跑一遍从 ss 到 tt 的最大流，就能尽量多的消耗那些流量啦 QwQ。
*/
using namespace std;
typedef long long ll;
const int LEN = 2e5 + 5;
const int oo = (1LL << 31) - 1;
namespace DINIC {
    int tot, n, m, src, tar, qh, qt, cnt, s, t, S, T, ans, flow;
    struct edge {
        int vet, next, len;
    } E[LEN * 2];
    int dis[LEN], gap[LEN], head[LEN], cur[LEN], q[LEN], vis[LEN], IN[LEN];
    void add(int u, int v, int c) {
        E[++tot] = (edge){v, head[u], c};
        head[u] = tot;
    }
    void join(int u, int v, int c) {
        add(u, v, c);
        add(v, u, 0);
    }
    void init() {
        tot = -1;
        for (int i = 1; i <= cnt; i++) head[i] = -1;
    }
    bool bfs() {
        for (int i = 1; i <= cnt; i++) dis[i] = 0;
        qh = qt = 0;
        q[++qt] = src;
        dis[src] = 1;
        while (qh < qt) {
            int u = q[++qh];
            for (int e = head[u]; e != -1; e = E[e].next) {
                int v = E[e].vet;
                if (E[e].len && !dis[v]) {
                    dis[v] = dis[u] + 1;
                    if (v == tar) return 1;
                    q[++qt] = v;
                }
            }
        }
        return dis[tar];
    }
    int dfs(int u, int aug) {
        if (u == tar || !aug) return aug;
        int tmp = 0;
        for (int &e = cur[u]; e != -1; e = E[e].next) {
            int v = E[e].vet;
            if (dis[v] == dis[u] + 1) {
                if (tmp = dfs(v, min(aug, E[e].len))) {
                    E[e].len -= tmp;
                    E[e ^ 1].len += tmp;
                    return tmp;
                }
            }
        }
        return 0;
    }
    int maxflow(int s, int t) {
        src = s, tar = t;
        int res = 0, flow = 0;
        while (bfs()) {
            for (int i = 1; i <= cnt; i++) cur[i] = head[i];
            while (flow = dfs(src, oo)) res += flow;
        }
        return res;
    }
}
using namespace DINIC;
int main() {
    scanf("%d %d %d %d", &n, &m, &s, &t);
    cnt = n;
    S = ++cnt, T = ++cnt;
    init();
    for (int i = 1; i <= m; i++) {
        int x, y, l, r;
```

```
        scanf("%d %d %d %d", &x, &y, &l, &r);
        join(x, y, r - l);
        IN[y] += l, IN[x] -= l;
    }
    for (int i = 1; i <= n; i++) {
        if (IN[i] < 0) join(i, T, -IN[i]);
        else if (IN[i] > 0) {
            flow += IN[i];
            join(S, i, IN[i]);
        }
    }
    ans = maxflow(S, T);
    flow -= ans;
    join(t, s, oo);
    ans = maxflow(S, T);
    if (ans != flow) puts("please go home to sleep");
    else printf("%d\n", ans);
    return 0;
}
```

图匹配理论

```
二分图匹配:
点覆盖、最小点覆盖
点覆盖集即一个点集，使得所有边至少有一个端点在集合里。或者说是“点” 覆盖了所有“边”。极小点覆盖 (minimal vertex covering): 本身为点覆盖，其真子集都不是。最小点覆盖 (minimum vertex covering): 点最少的点覆盖。点覆盖数 (vertex covering number): 最小点覆盖的点数。

边覆盖、极小边覆盖
边覆盖集即一个边集，使得所有点都与集合里的边邻接。或者说是“边” 覆盖了所有“点”。极小边覆盖 (minimal edge covering): 本身是边覆盖，其真子集都不是。最小边覆盖 (minimum edge covering): 边最少的边覆盖。边覆盖数 (edge covering number): 最小边覆盖的边数。

独立集、极大独立集
独立集即一个点集，集合中任两个结点不相邻，则称 V 为独立集。或者说是导出的子图是零图 (没有边) 的点集。极大独立集 (maximal independent set): 本身为独立集，再加入任何点都不是。最大独立集 (maximum independent set): 点最多的独立集。独立数 (independent number): 最大独立集的点。

团
团即一个点集，集合中任两个结点相邻。或者说是导出的子图是完全图的点集。极大团 (maximal clique): 本身为团，再加入任何点都不是。最大团 (maximum clique): 点最多的团。团数 (clique number): 最大团的点数。

边独立集、极大边独立集
边独立集即一个边集，满足边集中的任两边不邻接。极大边独立集 (maximal edge independent set): 本身为边独立集，再加入任何边都不是。最大边独立集 (maximum edge independent set): 边最多的边独立集。边独立数 (edge independent number): 最大边独立集的边数。

边独立集又称匹配 (matching)，相应的有极大匹配 (maximal matching)，最大匹配 (maximum matching)，匹配数 (matching number)。

支配集、极小支配集
支配集即一个点集，使得所有其他点至少有一个相邻点在集合里。或者说是一部分的“点” 支配了所有 “点”。极小支配集 (minimal dominating set): 本身为支配集，其真子集都不是。最小支配集 (minimum dominating set): 点最少的支配集。支配数 (dominating number): 最小支配集的点数。

边支配集、极小边支配集
边支配集即一个边集，使得所有边至少有一条邻接边在集合里。或者说是一部分的“边” 支配了所有 “边”。极小边支配集 (minimal edge dominating set): 本身是边支配集，其真子集都不是。最小边支配集 (minimum edge dominating set): 边最少的边支配集。边支配数 (edge dominating number): 最小边支配集的边数。

最小路径覆盖
最小路径覆盖 (path covering): 是 “路径” 覆盖 “点”，即用尽量少的不相交简单路径覆盖有向无环图 G 的所有顶点，即每个顶点严格属于一条路径。路径的长度可能为 0(单个点)。
最小路径覆盖数＝ G 的点数 - 最小路径覆盖中的边数。应该使得最小路径覆盖中的边数尽量多，但是又不能让两条边在同一个顶点相交。拆点: 将每一个顶点 i 拆成两个顶点 Xi 和 Yi。然后根据原图中边的信息，从 X 部往 Y 部引边。所有边的方向都是由 X 部到 Y 部。因此，所转化出的二分图的最大匹配数则是原图 G 中最小路径覆盖上的边数。因此由最小路径覆盖数＝原图 G 的顶点数 - 二分图的最大匹配数便可以得解。

匹配
匹配 (matching) 是一个边集，满足边集中的边两两不邻接。匹配又称边独立集 (edge independent set)。
在匹配中的点称为匹配点 (matched vertex) 或饱和点；反之，称为未匹配点 (unmatched vertex) 或未饱和点。
交错轨 (alternating path) 是图的一条简单路径，满足任意相邻的两条边，一条在匹配内，一条不在匹配内。
增广轨 (augmenting path): 是一个始点与终点都为未匹配点的交错轨。
最大匹配 (maximum matching) 是具有最多边的匹配。
匹配数 (matching number) 是最大匹配的大小。
完美匹配 (perfect matching) 是匹配了所有点的匹配。
完备匹配 (complete matching) 是匹配了二分图较小集合 (二分图 X, Y 中小的那个) 的所有点的匹配。
```

```
增广轨定理：一个匹配是最大匹配当且仅当没有增广轨。
所有匹配算法都是基于增广轨定理：一个匹配是最大匹配当且仅当没有增广轨。这个定理适用于任意图。

二分图的性质
二分图中，点覆盖数是匹配数。
(1) 二分图的最大匹配数等于最小覆盖数，即求最少的点使得每条边都至少和其中的一个点相关联，很显然直接取最大匹配的一段节点即可。
(2) 二分图的独立数等于顶点数减去最大匹配数，很显然的把最大匹配两端的点都从顶点集中去掉这个时候剩余的点是独立集，这是 |V|-2*|M|，同时必然可以从每条匹配边的两端取一个点加入独立集并且保持其独立集性质。
(3) DAG 的最小路径覆盖，将每个点拆点后作最大匹配，结果为 n-m，求具体路径的时候顺着匹配边走就可以，匹配边 i→j',j→k',k→l'.... 构成一条有向路径。
(4) 最大匹配数 = 左边匹配点 + 右边未匹配点。因为在最大匹配集中的任意一条边，如果他的左边没标记，右边被标记了，那么我们就可找到一条新的增广路，所以每一条边都至少被一个点覆盖。
(5) 最小边覆盖 = 图中点的个数-最大匹配数 = 最大独立集。

定理 1：最小覆盖数 = 最大匹配数
定理 2：最大独立集 S 与 最小覆盖集 T 互补

有向无环图最小不相交路径覆盖
定义：用最少的不相交路径覆盖所有顶点。
定理：把原图中的每个点 V 拆成 Vx 和 Vy，如果有一条有向边 A->B，那么就加边 Ax-By。这样就得到了一个二分图，最小路径覆盖 = 原图的节点数-新图最大匹配。

有向无环图最小可相交路径覆盖
定义：用最小的可相交路径覆盖所有顶点。
算法：先用 floyd 求出原图的传递闭包，即如果 a 到 b 有路，那么就加边 a->b。然后就转化成了最小不相交路径覆盖问题。

Kuhn-Munkers 算法的几种变形应用
1.Kuhn-Munkers 算法是求最大权完备匹配，如果要求最小权完备匹配怎么办？方法很简单，只需将所有的边权值取其相反数，求最大权完备匹配，匹配的值再取相反数即可。
2.Kuhn-Munkers 算法的运行要求是必须存在一个完备匹配，如果求一个最大权匹配(不一定完备) 该如何办？依然很简单，把不存在的边权值赋为 0。
3.Kuhn-Munkers 算法求得的最大权匹配是边权值和最大，如果我想要边权之积最大，又怎样转化？还是不难办到，每条边权取自然对数，然后求最大和权匹配，求得的结果 a 再算出 e^a 就是最大积匹配。
```

二分图最大匹配匈牙利算法

```
const int maxn = 10005;   //点的最大个数

int head[maxn], cnt=0;//head 用来表示以 i 为起点的第一条边存储的位置，cnt 读入边的计数器

struct Edge {
  int next; //同一起点的上一条边的储存位置
  int to; //第 i 条边的终点
  int w; //第 i 条边权重
};

Edge edge[maxn];

void addedge(int u,int v,int w) {
  edge[cnt].w = w;
  edge[cnt].to = v;
  edge[cnt].next = head[u];
  head[u] = cnt++;
}

void traverse() {
  for(int i=0; i<=n; i++) {
    for(int j=head[i]; j! =-1; j=edge[j].next) {
      std::cout << i << " " << head[i].to << " " << head[i].w << '\n';
    }
  }
}
```

二分图最大权匹配 KM 算法

```
const int maxn = "Edit";
const int inf = 2e9;

int n, cost[maxn][maxn];
int lx[maxn], ly[maxn], match[maxn], slack[maxn], prev[maxn];
bool vy[maxn];

void augment(int root) {
  std::fill(vy + 1, vy + n + 1, false);
  std::fill(slack + 1, slack + n + 1, inf);
  int py;
  match[py = 0] = root;
  do {
    vy[py] = true;
    int x = match[py], delta = inf, yy;
```

```
    for (int y = 1; y <= n; y++) {
      if (!vy[y]) {
        if (lx[x] + ly[y] - cost[x][y] < slack[y]) {
          slack[y] = lx[x] + ly[y] - cost[x][y];
          prev[y] = py;
        }
        if (slack[y] < delta) {
          delta = slack[y];
          yy = y;
        }
      }
    }
    for (int y = 0; y <= n; y++) {
      if (vy[y]) {
        lx[match[y]] -= delta;
        ly[y] += delta;
      } else {
        slack[y] -= delta;
      }
    }
    py = yy;
  } while (match[py] != -1);
  do {
    int pre = prev[py];
    match[py] = match[pre];
    py = pre;
  } while (py);
}

int KM() {
  for (int i = 1; i <= n; i++) {
    lx[i] = ly[i] = 0;
    match[i] = -1;
    for (int j = 1; j <= n; j++) {
      lx[i] = std::max(lx[i], cost[i][j]);
    }
  }
  int answer = 0;
  for (int root = 1; root <= n; root++) {
    augment(root);
  }
  for (int i = 1; i <= n; i++) {
    answer += lx[i];
    answer += ly[i];
    //printf("%d %d\n", match[i], i);
  }
  return answer;
}
```

数据结构

树状数组

```
void add(int i, int x) {
  for(;i <= n; i += i & -i)
    tree[i] += x;
}

int sum(int i) {
  int ret = 0;
  for(; i; i -= i & -i) ret += tree[i];
  return ret;
}
```

差分数组

```
//Author:CookiC
/*
*a 为原数组
*C 为差分数组
*/
int a[]={0, 1, 1, 1, 1, 1, 1};
int N, C[maxn];

int Sum(unsigned n) {
  int sum = 0;
  while(n>0){
    sum += C[n];
    n -= lowbit(n);
  }
  return sum;
}

void Add(unsigned n, int d) {
  while(n<=N){
    C[n]+=d;
    n+=lowbit(n);
  }
}

void Add(int L,int R, int d) {
  Add(L,d);
  Add(R+1,-d);
}

void Init() {
  memset(C, 0, sizeof(C));
  Add(1, a[1]);
  for(int i=2; i<=N; ++i)
    Add(i, a[i]-a[i-1]);
}

void Update() {
  for(int i=1; i<=N; ++i)
    a[i] = Sum(i);
}
```

序列自动机

```
#include <bits/stdc++.h>
using namespace std;

const int maxn = 1e6 + 10;
int nx[maxn][30];
string s;

void init() {
  int len = s.length();
  for(int i = 0; i < 26; i ++)
    nx[len][i] = nx[len + 1][i] = len + 1;
  for(int i = len - 1; i >= 1; i --) {
    for(int j = 0; j < 26; j ++)
      nx[i - 1][j] = nx[i][j];
    nx[i - 1][s[i] - 'a'] = i;
  }
}

int main() {
  cin >> s;
  init();
  int Q;
  scanf("%d", &Q);
  while(Q --) {
    string t;
    cin >> t;
    bool flag = true;
    int lt = t.length();
    int st = 0;
    for(int i = 0; i < lt; i ++) {
      st = nx[st][t[i] - 'a'];
      if(st == 0) {
        flag = false;
        break;
      }
    }

    if(flag) printf("YES\n");
    else printf("NO\n");

```

```
  }
  return 0;
}
```

单调栈单调队列

```
/*
 * Author: Simon
 * 功能: 单调栈求某子序列中的最小值乘以子序列所有元素和最大
 * 最基础的应用就是给定一组数，针对每个数，寻找它和它右边第一个比它大的数
   ↪ 之间有多少个数。
 * 给定一序列，寻找某一子序列，使得子序列中的最小值乘以子序列的长度最大。
 * 给定一序列，寻找某一子序列，使得子序列中的最小值乘以子序列所有元素和最
   ↪ 大。
 * 给定一序列，在限定每个字母出现次数的情况下，求其字典序最小的 k 长子序
   ↪ 列。可求后缀和，
 *      当一个字母出栈时，判断此后位置当前字母的个数是否满足限制条件，若满
   ↪ 足出栈，否则不出栈。
 * 复杂度: O(n)
 */
int Stack[maxn],lft[maxn],top=0,ans=0,a[maxn];
a[n+1]=INF;
for(int i=1;i<=n+1;i++){
  int t=i;lft[i]=i;
  while(top&&a[i]<a[Stack[top]]){
    t=Stack[top--];
    ans=max(ans,(i-lft[t])*a[t]);
  }
  Stack[++top]=i;
  lft[i]=lft[t];
}
/*
 * Author: Simon
 * 功能: 求区间长度小于 k 的区间最小值
 * 复杂度: O(n)
 */
int q[maxn],l=1,r=0,a[maxn];
for(int i=1;i<=n;i++){
  while(l<=r&&a[i]<=a[q[r]]) r--; //维护单调递增区间
  q[++r]=i;
  while(l<=r&&i-q[l]>=k) l++; //维护不大于 k 的区间长度
  if(i-k>=0) return a[q[l]];
}
```

二维树状数组

```
int N;
int c[maxn][maxn];

inline int lowbit(int t) {
  return t&(-t);
}

void update(int x, int y, int v) {
  for (int i=x; i<=N; i+=lowbit(i)) {
    for (int j=y; j<=N; j+=lowbit(j)) {
      c[i][j]+=v;
    }
  }
}

int query(int x, int y) {
  int s = 0;
  for (int i=x; i>0; i-=lowbit(i)) {
    for (int j=y; j>0; j-=lowbit(j)) {
      s += c[i][j];
    }
  }
  return s;
}

int sum(int x, int y, int xx, int yy) {
  x--, y--;
  return query(xx, yy) - query(xx, y) - query(x, yy) + query(x, y);
}
```

树状数组求逆序对

```
//树状数组求逆序对
const int maxn = "Edit";

int lowbit(int x) {
  return (x&-x);
}

bool cmp(std::pair<int, int> no1, std::pair<int, int> no2) {
  return no1.first < no2.first;
}

int d[maxn], p[maxn], n;
std::pair<int, int> start[maxn];

void add(int x) {
  while (x <= n) {
    d[x]++;
    x += lowbit(x);
  }
```

```cpp
}

long long sum(int x) {
  long long sum = 0;
  while (x) {
    sum += d[x];
    x -= lowbit(x);
  }
  return sum;
}

int main(int argc, char *argv[]) {
  long long ans;
  std::cin>>n;
  memset(d,0,sizeof(d));
  ans=0;
  for (int i=1;i<=n;i++) {
    std::cin >> start[i].first;
    start[i].second = i;
  }
  std::sort(start+1, start+n+1, cmp);
  int id = 1;
  p[start[1].second]=1;
  for (int i = 2; i <= n; ++i) {
    if (start[i].first == start[i-1].first) {
      p[start[i].second] = id;
    } else {
      p[start[i].second] = ++id;
    }
  }
  for (int i=1;i<=n;i++) {
     add(p[i]);
     ans += i - sum(p[i]);
  }
  std::cout << ans << std::endl;
  return 0;
}
```

堆

```cpp
const int N = 1000;

template <class T>
class Heap {
  private:
    T h[N];
    int len;
  public:
    Heap() {
      len = 0;
    }
    inline void push(const T& x) {
      h[++len] = x;
      std::push_heap(h+1, h+1+len, std::greater<T>());
    }
    inline T pop() {
      std::pop_heap(h+1, h+1+len, std::greater<T>());
      return h[len--];
    }
    inline T& top() {
      return h[1];
    }
    inline bool empty() {
      return len == 0;
    }
};
```

RMQ

```cpp
//A 为原始数组，d[i][j] 表示从 i 开始，长度为 (1<<j) 的区间最小值

int A[maxn];
int d[maxn][30];

void init(int A[], int len) {
  for (int i = 0; i < len; i++)d[i][0] = A[i];
  for (int j = 1; (1 << j) <= len; j++) {
    for (int i = 0; i + (1 << j) - 1 < len; i++) {
      d[i][j] = min(d[i][j - 1], d[i + (1 << (j - 1))][j - 1]);
    }
  }
}

int query(int l, int r) {
  int p = 0;
  while ((1 << (p + 1)) <= r - l + 1)p++;
  return min(d[l][p], d[r - (1 << p) + 1][p]);
}
```

RMQ

```cpp
//author: wavator
#include <algorithm>
#include <vector>

template <class T>
struct RMQ {
  std::vector<std::vector<T> > rmq;
```

```
  // vector<T> rmq[20]; or T[100002][20] if need speed
  //T kInf = numeric_limits<T>::max(); // if need return a value when the interval fake
  void init(const std::vector<T>& a) { // 0 base
    int n = (int)a.size(), base = 1, depth = 1;
    while (base < n)
      base <<= 1, ++depth;
    rmq.assign((unsigned)depth, a);
    for (int i = 0; i < depth - 1; ++i)
      for (int j = 0; j < n; ++j) {
        rmq[i + 1][j] = std::min(rmq[i][j], rmq[i][std::min(n - 1, j + (1 << i))]);
      }
  }
  T q(int l, int r) { // [l, r)
    if(l>r)return 0x3f3f3f3f;
    int dep = 31 - __builtin_clz(r - l); // log(b - a)
    return min(rmq[dep][l], rmq[dep][r - (1 << dep)]);
  }
};
```

线段树

```
//A 为原始数组，sum 记录区间和，Add 为懒惰标记

int A[maxn], sum[maxn << 2], Add[maxn << 2];

void pushup(int rt) {
  sum[rt] = sum[rt << 1] + sum[rt << 1 | 1];
}

void pushdown(int rt, int l, int r) {
  if (Add[rt]) {
    int mid = (l + r) >> 1;
    Add[rt << 1] += Add[rt];
    Add[rt << 1 | 1] += Add[rt];
    sum[rt << 1] += (mid - l + 1)*Add[rt];
    sum[rt << 1 | 1] += (r - mid)*Add[rt];
    Add[rt] = 0;
  }
}

void build(int l, int r, int rt) {
  if (l == r) {
    sum[rt] = A[l];
    return;
  }
  int mid = (l + r) >> 1;
  build(l, mid, rt << 1);
  build(mid + 1, r, rt << 1 | 1);
  pushup(rt);
}

//区间加值
void update(int L, int R, int val, int l, int r, int rt) {
  if (L <= l && R >= r) {
    Add[rt] += val;
    sum[rt] += (r - l + 1)*val;
    return;
  }
  pushdown(rt, l, r);
  int mid = (l + r) >> 1;
  if (L <= mid)update(L, R, val, l, mid, rt << 1);
  if (R > mid)update(L, R, val, mid + 1, r, rt << 1 | 1);
  pushup(rt);
}

//点修改
void update(int index, int val, int l, int r, int rt) {
  if (l == r) {
    sum[rt] = val;
    return;
  }
  int mid = (l + r) >> 1;
  if (index <= mid)update(index, val, l, mid, rt << 1);
  else update(index, val, mid + 1, r, rt << 1 | 1);
  pushup(rt);
}

//区间查询
int query(int L, int R, int l, int r, int rt) {
  if (L <= l && R >= r) {
    return sum[rt];
  }
  pushdown(rt, l, r);
  int mid = (l + r) >> 1;
  int ret = 0;
  if (L <= mid)ret += query(L, R, l, mid, rt << 1);
  if (R > mid)ret += query(L, R, mid + 1, r, rt << 1 | 1);
  return ret;
}
```

ZKW 线段树

```
const int maxn = 50009;
using ll = long long;

ll T[maxn*4];
int M,n;
```

```cpp
void build() {
  for(M=1;M<=n+1;M<<=1);
  for(int i=1;i<=n;i++)
    std::cin >> T[i+M];
  for(int i=M-1;i;i--)
    T[i]=T[i<<1]+T[i<<1|1];
}

void update(int x,int val) {
  T[x+=M]=val;    //修改
//  T[x+=M]+=val;   //加值
  for(x>>=1;x>=1;x>>=1) {
    T[x]=T[x<<1]+T[x<<1|1];
  }
}

ll query(int l,int r) {
  l=l+M-1,r=r+M+1;
  ll ans=0;
  for(;l^r^1;l>>=1,r>>=1) {
    if(~l&1) ans+=T[l^1];
    if(r&1)  ans+=T[r^1];
  }
  return ans;
}
```

吉司机线段树

```cpp
//使用方法
//Build(1, 1, n) 建树
//读入 ql,  qr,  qt 调用函数 XXX(1, 1, n)
using ll = long long;

const int N = "Edit";
const int M = N<<2;

int mx[M], sx[M], cx[M], mn[M], sn[M], cn[M];
ll sum[M];
int ta[M];

inline void update(int x){
  int l = x<<1, r = x<<1|1;
  sum[x] = sum[l] + sum[r];
  if (mx[l] == mx[r]) {
    mx[x] = mx[l], cx[x] = cx[l] + cx[r], sx[x] = std::max(sx[l], sx[r]);
  } else { // r>l
    if (mx[l] > mx[r]) std::swap(l,r);
    mx[x] = mx[r];
    cx[x] = cx[r];
    sx[x] = std::max(sx[r], mx[l]);
  }
  if (mn[l] == mn[r]) {
    mn[x] = mn[l], cn[x] = cn[l] + cn[r], sn[x] = std::min(sn[l], sn[r]);
  } else { // r<l
    if (mn[l] < mn[r]) std::swap(l,r);
    mn[x] = mn[r];
    cn[x] = cn[r];
    sn[x] = std::min(sn[r], mn[l]);
  }
}

//建树
inline void Build(int x, int l, int r){
  if (l == r) {
    int a;
    std::cin >> a;
    sum[x] = mx[x] = mn[x] = a; cx[x] = cn[x] = 1;
    sx[x] = -(1<<30); sn[x]=1<<30;  ta[x]=0;
    return;
  }
  int mid=(l+r)>>1;
  Build(x<<1,l,mid);
  Build(x<<1|1,mid+1,r);
  update(x);
}

inline void _add(int x, int l, int r, int t) {
  sum[x] += (ll)(r-l+1)*t;
  mn[x]+=t; sn[x]+=t; mx[x]+=t; sx[x]+=t;
  ta[x]+=t;
}

inline void _min(int x,int l,int r,int t){
  sum[x] -= (ll)cx[x]*(mx[x]-t);
  mx[x]=t; mn[x]=std::min(mn[x],t);
  if (mn[x] == mx[x]) {
    sum[x] = (ll)(r-l+1)*t; cx[x] = cn[x] = r-l+1; sx[x] = -(1<<30); sn[x] =
      ↪ 1<<30;
  } else {
    sn[x]=std::min(sn[x],t);
  }
}

inline void _max(int x,int l,int r,int t){
  sum[x] += (ll)cn[x]*(t-mn[x]);
  mn[x] = t; mx[x] = std::max(mx[x], t);
  if (mn[x] == mx[x]) {
    sum[x]=(ll)(r-l+1)*t; cx[x] = cn[x] = r-l+1; sx[x] = -(1<<30); sn[x] =
      ↪ 1<<30;
```

```
  } else {
    sx[x] = std::max(sx[x], t);
  }
}

inline void push(int x, int l, int r){
  int mid = (l+r)>>1;
  if (ta[x]) {
    _add(x<<1, l, mid, ta[x]);
    _add(x<<1|1, mid+1, r, ta[x]);
    ta[x] = 0;
  }
  if (mx[x<<1]>mx[x] && sx[x<<1]<mx[x]) _min(x<<1, l, mid, mx[x]);
  if (mx[x<<1|1]>mx[x] && sx[x<<1|1]<mx[x]) _min(x<<1|1, mid+1, r, mx[x]);
  if (mn[x<<1]<mn[x] && sn[x<<1]>mn[x]) _max(x<<1, l, mid, mn[x]);
  if (mn[x<<1|1]<mn[x] && sn[x<<1|1]>mn[x]) _max(x<<1|1, mid+1, r, mn[x]);
}

int ql, qr, qt;
int n;

//把一个区间 [L,R] 里小于 x 的数变成 x
inline void Mmax(int x, int l, int r){
  if (mn[x] >= qt) return;
  if (ql<=l && r<=qr && qt<sn[x]){
    _max(x, l, r, qt); return;
  }
  push(x, l, r); int mid = (l+r)>>1;
  if (ql<=mid) Mmax(x<<1, l, mid);
  if (qr>mid) Mmax(x<<1|1, mid+1, r);
  update(x);
}

//把一个区间 [L,R] 里大于 x 的数变成 x
inline void Mmin(int x,int l,int r) {
  if (mx[x]<=qt) return;
  if (ql<=l && r<=qr && qt>sx[x]) {
    _min(x,l,r,qt); return;
  }
  push(x,l,r); int mid=(l+r)>>1;
  if (ql<=mid) Mmin(x<<1, l, mid);
  if (qr>mid) Mmin(x<<1|1, mid+1, r);
  update(x);
}

//区间加值
inline void Add(int x, int l, int r) {
  if (ql<=l && r<=qr) {
    _add(x, l, r, qt); return;
  }
  push(x, l, r); int mid=(l+r)>>1;
  if (ql<=mid) Add(x<<1, l, mid);
  if (qr>mid) Add(x<<1|1, mid+1, r);
  update(x);
}

//区间最大值
inline int Max(int x, int l, int r) {
  if (ql<=l && r<=qr) return mx[x];
  push(x, l, r);
  int ret=-(1<<30); int mid=(l+r)>>1;
  if (ql<=mid) ret=std::max(ret, Max(x<<1, l, mid));
  if (qr>mid) ret=std::max(ret, Max(x<<1|1, mid+1, r));
  return ret;
}

//区间最小值
inline int Min(int x, int l, int r) {
  if (ql<=l && r<=qr)  return mn[x];
  push(x, l, r) ;
  int ret=1<<30; int mid=(l+r) >>1;
  if (ql<=mid) ret=std::min(ret, Min(x<<1, l, mid) );
  if (qr>mid)  ret=std::min(ret, Min(x<<1|1, mid+1, r) );
  return ret;
}

//区间求和
inline ll Sum(int x, int l, int r) {
  if (ql<=l && r<=qr)  return sum[x];
  push(x, l, r) ;
  ll ret=0; int mid=(l+r) >>1;
  if (ql<=mid) ret+=Sum(x<<1, l, mid) ;
  if (qr>mid)  ret+=Sum(x<<1|1, mid+1, r) ;
  return ret;
}
```

扫描线

```
// 矩形面积并（交） 求并 FLAG=0, 求交 FLAG=1
struct Line {
  double l, r, h;
  int d;
  Line() {}
  Line(double l, double r, double h, int d) : l(l), r(r), h(h), d(d) {}
  bool operator < (const Line L) const {
    return h < L.h;
  }
}line[maxn << 1];

```

```
int FLAG;  // 求矩形面积并 FLAG = 0, 求矩形面积交 FLAG = 1
int Cover[maxn << 3];
double A[maxn << 1];
double Sum[maxn << 3];
double X1[maxn << 1], X2[maxn << 1], Y1[maxn << 1], Y2[maxn << 1];

void pushdown(int rt, int l, int r) {
  int mid = (l + r) >> 1;
  if (Cover[rt] != -1) {
    Cover[rt << 1] = Cover[rt << 1 | 1] = Cover[rt];
    Sum[rt << 1] = (Cover[rt] > FLAG ? (A[mid + 1] - A[l]) : 0);
    Sum[rt << 1 | 1] = (Cover[rt] > FLAG ? (A[r + 1] - A[mid + 1]) : 0);
  }
}

void pushup(int rt, int l, int r) {
  if (Cover[rt << 1] == -1 || Cover[rt << 1 | 1] == -1) Cover[rt] = -1;
  else if (Cover[rt << 1] != Cover[rt << 1 | 1]) Cover[rt] = -1;
  else Cover[rt] = Cover[rt << 1];
  Sum[rt] = Sum[rt << 1] + Sum[rt << 1 | 1];
}

void build(int l, int r, int rt) {
  if (l == r) {
    Cover[rt] = 0;
    Sum[rt] = 0;
    return;
  }
  int mid = (l + r) >> 1;
  build(l, mid, rt << 1);
  build(mid + 1, r, rt << 1 | 1);
  pushup(rt, l, r);
}

void update(int L, int R, int v, int l, int r, int rt) {
  if (L <= l && r <= R) {
    if (Cover[rt] != -1) {
      Cover[rt] += v;
      Sum[rt] = (Cover[rt] > FLAG ? (A[r + 1] - A[l]) : 0);
      return;
    }
  }
  pushdown(rt, l, r);
  int mid = (l + r) >> 1;
  if (L <= mid) update(L, R, v, l, mid, rt << 1);
  if (mid < R) update(L, R, v, mid + 1, r, rt << 1 | 1);
  pushup(rt, l, r);
}

int find(double key, int n, double d[]) {
  int l = 1, r = n;
  while (r >= l) {
    int mid = (r + l) >> 1;
    if (d[mid] == key) return mid;
    else if (d[mid] > key) r = mid - 1;
    else l = mid + 1;
  }
  return -1;
}

int init(int n) {
  int N = 0;
  for (int i = 1; i <= n; i++) {
    A[++N] = X1[i];
    line[N] = Line(X1[i], X2[i], Y1[i], 1);
    A[++N] = X2[i];
    line[N] = Line(X1[i], X2[i], Y2[i], -1);
  }
  sort(A + 1, A + N + 1);
  sort(line + 1, line + N + 1);
  int k = 1;
  for (int i = 2; i <= N; i++)
    if (A[i] != A[i - 1])
      A[++k] = A[i];
  build(1, k - 1, 1);
  return k;
}

double query(int n, int k) {
  double ret = 0;
  for (int i = 1; i < n; i++) {
    int l = find(line[i].l, k, A);
    int r = find(line[i].r, k, A) - 1;
    if (l <= r) update(l, r, line[i].d, 1, k - 1, 1);
    ret += Sum[1] * (line[i + 1].h - line[i].h);
  }
  return ret;
}
/*
int main()
{
  int n, T;
  scanf("%d", &T);
  while (T--) {
    scanf("%d", &n);
    for (int i = 1; i <= n; i++)
      scanf("%lf%lf%lf%lf", &X1[i], &Y1[i], &X2[i], &Y2[i]);
    int k = init(n);
    double ans = query(n << 1, k);
    printf("%.2lf\n", ans);
```

```cpp
  }
}
*/

//========================================================================================================

// 矩形周长并
int Sum[maxn << 3], cnt[maxn << 3], vNum[maxn << 3];
bool lbd[maxn << 3], rbd[maxn << 3];
double X1[maxn << 1], X2[maxn << 1], Y1[maxn << 1], Y2[maxn << 1];
double A[maxn << 1];

struct Line {
  double l, r, h;
  int label;
  Line() {}
  Line(double l, double r, double h, int label) :l(l), r(r), h(h), label(label) {}
  bool operator < (const Line L) const {
    return h < L.h;
  }
}line[maxn << 1];

void pushup(int l, int r, int rt) {
  if (cnt[rt]) {
    lbd[rt] = rbd[rt] = true;
    Sum[rt] = A[r + 1] - A[l];
    vNum[rt] = 2;
  }
  else if (l == r) Sum[rt] = vNum[rt] = lbd[rt] = rbd[rt] = 0;
  else {
    lbd[rt] = lbd[rt << 1];
    rbd[rt] = rbd[rt << 1 | 1];
    Sum[rt] = Sum[rt << 1] + Sum[rt << 1 | 1];
    vNum[rt] = vNum[rt << 1] + vNum[rt << 1 | 1];
    if (rbd[rt << 1] && lbd[rt << 1 | 1]) vNum[rt] -= 2;
  }
}

void update(int L, int R, int v, int l, int r, int rt) {
  if (L <= l && r <= R) {
    cnt[rt] += v;
    pushup(l, r, rt);
    return;
  }
  int mid = (l + r) >> 1;
  if (L <= mid) update(L, R, v, l, mid, rt << 1);
  if (R > mid) update(L, R, v, mid + 1, r, rt << 1 | 1);
  pushup(l, r, rt);
}

int find(double key, int n, double d[]) {
  int l = 1, r = n;
  while (r >= l) {
    int mid = (r + l) >> 1;
    if (d[mid] == key) return mid;
    else if (d[mid] > key) r = mid - 1;
    else l = mid + 1;
  }
  return -1;
}

int init(int n) {
  for (int i = 1; i <= n; i++) {
    A[i] = X1[i]; A[i + n] = X2[i];
    line[i].l = X1[i]; line[i].r = X2[i];
    line[i].h = Y1[i]; line[i].label = 1;
    line[i + n].l = X1[i]; line[i + n].r = X2[i];
    line[i + n].h = Y2[i]; line[i + n].label = -1;
  }
  n <<= 1;
  int k = 1;
  sort(A + 1, A + n + 1);
  sort(line + 1, line + n + 1);
  for (int i = 2; i <= n; i++)
    if (A[i] != A[i - 1])
      A[++k] = A[i];
  return k;
}

double query(int n, int k) {
  double ret = 0, lst = 0;
  for (int i = 1; i <= n; i++) {
    if (line[i].l < line[i].r) {
      int l = find(line[i].l, k, A);
      int r = find(line[i].r, k, A);
      update(l, r - 1, line[i].label, 1, k - 1, 1);
    }
    ret += vNum[1] * (line[i + 1].h - line[i].h);
    ret += abs(Sum[1] - lst);
    lst = Sum[1];
  }
  return ret;
}
/*
int main()
{
  int n;
  while (~scanf("%d", &n)) {
    for (int i = 1; i <= n; i++)
```

```
      scanf("%lf%lf%lf%lf", &X1[i], &Y1[i], &X2[i], &Y2[i]);
    int k = init(n);
    double ans = query(n << 1, k);
    printf("%lf\n", ans);
  }
  return 0;
}
*/
```

固定大小矩形最大点覆盖

```
// 扫描线 求 矩形最大点覆盖
struct Line {
  ll x, y1, y2, k;  // k 为矩形权值
  bool operator < (const Line nod) const {
    return x < nod.x || (x == nod.x && k < nod.k);
  }
}line[maxn];
struct segTree {
  ll ma, l, r, lazy;
}tree[maxn << 2];
ll yy[maxn];
int cnt, ycnt;
void pushup(int rt) {
  tree[rt].ma = max(tree[rt << 1].ma, tree[rt << 1 | 1].ma) + tree[rt].lazy;
}
void build(int l, int r, int rt) {
  tree[rt].ma = tree[rt].lazy = 0;
  tree[rt].l = yy[l], tree[rt].r = yy[r];
  if (r - l == 1) return;
  int mid = (l + r) >> 1;
  build(l, mid, rt << 1);
  build(mid, r, rt << 1 | 1);
  pushup(rt);
}
void update(ll L, ll R, ll w, int rt) {
  if (tree[rt].l >= L && tree[rt].r <= R) {
    tree[rt].lazy += w;
    tree[rt].ma += w;
    return;
  }
  if (L < tree[rt << 1].r)
    update(L, min(R, tree[rt << 1].r), w, rt << 1);
  if (R > tree[rt << 1 | 1].l)
    update(max(tree[rt << 1 | 1].l, L), R, w, rt << 1 | 1);
  pushup(rt);
}
int main()
{
  ll n, W, H, x, y, w, ma;
  while (~scanf("%lld%lld%lld", &n, &W, &H)) {
    cnt = 0; ycnt = 1; ma = 0;
    for (int i = 1; i <= n; i++) {
      scanf("%lld%lld%lld", &x, &y, &w);
      line[cnt].x = x; line[cnt].y1 = y; line[cnt].y2 = y + H;
      line[cnt++].k = w;
      line[cnt].x = x + W; line[cnt].y1 = y; line[cnt].y2 = y + H;
      line[cnt++].k = -w;
      yy[ycnt++] = y;
      yy[ycnt++] = y + H;
    }
    sort(yy + 1, yy + ycnt);
    ycnt = unique(yy + 1, yy + ycnt) - (yy + 1);
    sort(line, line + cnt);
    build(1, ycnt, 1);
    for (int i = 0; i < cnt; i++) {
      update(line[i].y1, line[i].y2, line[i].k, 1);
      if (line[i].k > 0) ma = max(ma, tree[1].ma);
    }
    printf("%lld\n", ma);
  }
  return 0;
}
```

二维线段树（单点更新区间最值）

```
// 二维线段树单点更新 + 区间最值 树套树实现
int n;
int y2rt[maxn], x2rt[maxn];

struct Nodey {
  int l, r;
  int Max, Min;
};

struct Nodex {
  int l, r;
  Nodey nodey[maxn << 2];

  void build(int l, int r, int rt) {
    nodey[rt].l = l;
    nodey[rt].r = r;
    nodey[rt].Max = -inf;
    nodey[rt].Min = inf;
    if (l == r) {
      y2rt[l] = rt;
      return;
    }
    int mid = (l + r) >> 1;
```

```
    build(l, mid, rt << 1);
    build(mid + 1, r, rt << 1 | 1);
  }

  int queryMin(int rt, int L, int R) {
    if (nodey[rt].l == L && nodey[rt].r == R)
      return nodey[rt].Min;
    int mid = (nodey[rt].l + nodey[rt].r) >> 1;
    if (R <= mid) return queryMin(rt << 1, L, R);
    else if (L > mid) return queryMin(rt << 1 | 1, L, R);
    else return min(queryMin(rt << 1, L, mid), queryMin(rt << 1 | 1, mid + 1, R));
  }

  int queryMax(int rt, int L, int R) {
    if (nodey[rt].l == L && nodey[rt].r == R)
      return nodey[rt].Max;
    int mid = (nodey[rt].l + nodey[rt].r) >> 1;
    if (R <= mid) return queryMax(rt << 1, L, R);
    else if (L > mid) return queryMax((rt << 1) | 1, L, R);
    else return max(queryMax(rt << 1, L, mid), queryMax((rt << 1) | 1, mid + 1, R));
  }
}nodex[maxn << 2];

void build(int l, int r, int rt) {
  nodex[rt].l = l;
  nodex[rt].r = r;
  nodex[rt].build(1, n, 1);
  if (l == r) {
    x2rt[l] = rt;
    return;
  }
  int mid = (l + r) >> 1;
  build(l, mid, rt << 1);
  build(mid + 1, r, rt << 1 | 1);
}

// 点修改
void update(int x, int y, int val) {
  int rtx = x2rt[x];
  int rty = y2rt[y];
  nodex[rtx].nodey[rty].Min = nodex[rtx].nodey[rty].Max = val;
  for (int i = rtx; i; i >>= 1) {
    for (int j = rty; j; j >>= 1) {
      if (i == rtx && j == rty)continue;
      if (j == rty) {
        nodex[i].nodey[j].Min = min(nodex[i << 1].nodey[j].Min, nodex[(i << 1) | 1].nodey[j].Min);
        nodex[i].nodey[j].Max = max(nodex[i << 1].nodey[j].Max, nodex[(i << 1) | 1].nodey[j].Max);
      }
      else {
        nodex[i].nodey[j].Min = min(nodex[i].nodey[j << 1].Min, nodex[i].nodey[(j << 1) | 1].Min);
        nodex[i].nodey[j].Max = max(nodex[i].nodey[j << 1].Max, nodex[i].nodey[(j << 1) | 1].Max);
      }
    }
  }
}

int queryMin(int rt, int x1, int x2, int y1, int y2) {
  if (nodex[rt].l == x1 && nodex[rt].r == x2)
    return nodex[rt].queryMin(1, y1, y2);
  int mid = (nodex[rt].l + nodex[rt].r) >> 1;
  if (x2 <= mid)return queryMin(rt << 1, x1, x2, y1, y2);
  else if (x1 > mid)return queryMin(rt << 1 | 1, x1, x2, y1, y2);
  else return min(queryMin(rt << 1, x1, mid, y1, y2), queryMin(rt << 1 | 1, mid + 1, x2, y1, y2));
}

int queryMax(int rt, int x1, int x2, int y1, int y2) {
  if (nodex[rt].l == x1 && nodex[rt].r == x2)
    return nodex[rt].queryMax(1, y1, y2);
  int mid = (nodex[rt].l + nodex[rt].r) >> 1;
  if (x2 <= mid)return queryMax(rt << 1, x1, x2, y1, y2);
  else if (x1 > mid)return queryMax(rt << 1 | 1, x1, x2, y1, y2);
  else return max(queryMax(rt << 1, x1, mid, y1, y2), queryMax(rt << 1 | 1, mid + 1, x2, y1, y2));
}
```

二维线段树（区间加值单点查询）

```
// 二维线段树区间加值 + 单点查询 树套树实现
int n;
int x2rt[maxn], y2rt[maxn];

struct Nodey {
  int l, r;
  int val;
};

struct Nodex {
  int l, r;
  Nodey nodey[maxn << 2];

  void build(int l, int r, int rt) {
    nodey[rt].l = l;
    nodey[rt].r = r;
```

```
    nodey[rt].val = 0;
    if (l == r) {
      y2rt[l] = rt;
      return;
    }
    int mid = (l + r) >> 1;
    build(l, mid, rt << 1);
    build(mid + 1, r, rt << 1 | 1);
  }

  void addVal(int rt, int L, int R, int val) {
    if (nodey[rt].l == L && nodey[rt].r == R) {
      nodey[rt].val += val;
      return;
    }
    int mid = (nodey[rt].l + nodey[rt].r) >> 1;
    if (R <= mid) addVal(rt << 1, L, R, val);
    else if (L > mid) addVal(rt << 1 | 1, L, R, val);
    else {
      addVal(rt << 1, L, mid, val);
      addVal(rt << 1 | 1, mid + 1, R, val);
    }
  }
}nodex[maxn << 2];

void build(int l, int r, int rt) {
  nodex[rt].l = l;
  nodex[rt].r = r;
  nodex[rt].build(1, n, 1);
  if (l == r) {
    x2rt[l] = rt;
    return;
  }
  int mid = (l + r) >> 1;
  build(l, mid, rt << 1);
  build(mid + 1, r, rt << 1 | 1);
}

void addVal(int rt, int x1, int x2, int y1, int y2, int val) {
  if (nodex[rt].l == x1 && nodex[rt].r == x2) {
    nodex[rt].addVal(1, y1, y2, val);
    return;
  }
  int mid = (nodex[rt].l + nodex[rt].r) >> 1;
  if (x2 <= mid) addVal(rt << 1, x1, x2, y1, y2, val);
  else if (x1 > mid) addVal(rt << 1 | 1, x1, x2, y1, y2, val);
  else {
    addVal(rt << 1, x1, mid, y1, y2, val);
    addVal(rt << 1 | 1, mid + 1, x2, y1, y2, val);
  }
}

int getVal(int x, int y) {
  int ret = 0;
  for (int i = x2rt[x]; i; i >>= 1)
    for (int j = y2rt[y]; j; j >>= 1)
      ret += nodex[i].nodey[j].val;
  return ret;
}
```

主席树

```
// 主席树 支持查询 [l,r] 区间第 k 大，以及区间内不重复数字个数
// M = maxn * 30;
int n, q, m, tot;  // n 为数组大小，m 为离散化后数组大小
int A[maxn], T[maxn];  // A 为原数组，T 为离散化数组
int tree[M], lson[M], rson[M], Cnt[M];  // Cnt[i] 表示节点 i 的子树包含数字
   ↪ 的总数

void Init_hash() {
  for (int i = 1; i <= n; i++) T[i] = A[i];
  sort(T + 1, T + n + 1);
  m = unique(T + 1, T + n + 1) - T - 1;
}

inline int Hash(int x) { return lower_bound(T + 1, T + m + 1, x) - T; }

int build(int l, int r) {
  int root = tot++;
  Cnt[root] = 0;
  if (l != r) {
    int mid = (l + r) >> 1;
    lson[root] = build(l, mid);
    rson[root] = build(mid + 1, r);
  }
  return root;
}

int update(int root, int pos, int val) {
  int newroot = tot++, tmp = newroot;
  Cnt[newroot] = Cnt[root] + val;
  int l = 1, r = m;
  while (l < r) {
    int mid = (l + r) >> 1;
    if (pos <= mid) {
      lson[newroot] = tot++; rson[newroot] = rson[root];
      newroot = lson[newroot]; root = lson[root];
      r = mid;
    }
```

```
    else {
      rson[newroot] = tot++; lson[newroot] = lson[root];
      newroot = rson[newroot]; root = rson[root];
      l = mid + 1;
    }
    Cnt[newroot] = Cnt[root] + val;
  }
  return tmp;
}

void init() {  // 查询 l~r 第 k 大
  Init_hash();
  tree[0] = build(1, m);
  for (int i = 1; i <= n; i++) {
    int pos = Hash(A[i]);
    tree[i] = update(tree[i - 1], pos, 1);
  }
}

int query(int lrt, int rrt, int k) {  // 查询 l~r 第 k 大: T[query(tree[l - 1], tree[r], k)]
  int l = 1, r = m;
  while (l < r) {
    int mid = (l + r) >> 1;
    if (Cnt[lson[rrt]] - Cnt[lson[lrt]] >= k) {
      r = mid;
      lrt = lson[lrt];
      rrt = lson[rrt];
    }
    else {
      l = mid + 1;
      k -= Cnt[lson[rrt]] - Cnt[lson[lrt]];
      lrt = rson[lrt];
      rrt = rson[rrt];
    }
  }
  return l;
}

void init() {  // 查询 l~r 内不重复数字个数
  tree[0] = build(1, n);
  map<int, int>mp;
  for (int i = 1; i <= n; i++) {
    if (mp.find(A[i]) == mp.end())
      tree[i] = update(tree[i - 1], i, 1);
    else {
      int tmp = update(tree[i - 1], mp[A[i]], -1);
      tree[i] = update(tmp, i, 1);
    }
    mp[A[i]] = i;
  }
}

int query(int root, int pos) {  // 查询 l~r 内不重复数字个数: query(tree[r], l)
  int ret = 0;
  int l = 1, r = n;
  while (pos > l) {
    int mid = (l + r) >> 1;
    if (pos <= mid) {
      ret += Cnt[rson[root]];
      root = lson[root];
      r = mid;
    }
    else {
      root = rson[root];
      l = mid + 1;
    }
  }
  return ret + Cnt[root];
}
```

主席树动态 k 大

```
// 主席树求 [l,r] 第 k 大, 可单点修改 使用树状数组套主席树在线操作, 树状数组维护改变量
// M = maxn * 40;
int n, q, m, tot;
int A[maxn], T[maxn];
int tree[maxn], lson[M], rson[M], Cnt[M];
int Ntree[maxn], use[maxn];  // Ntree[i] 表示动态第 i 棵树的树根, use[i] 表示第 i 个树根是谁在使用

struct Query {
  int kind;
  int l, r, k;
}query[10005];

void Init_hash(int k) {
  sort(T, T + k);
  m = unique(T, T + k) - T;
}

int Hash(int x) { return lower_bound(T, T + m, x) - T; }

int build(int l, int r) {
  int root = tot++;
  Cnt[root] = 0;
  if (l != r) {
```

```
    int mid = (l + r) >> 1;
    lson[root] = build(l, mid);
    rson[root] = build(mid + 1, r);
  }
  return root;
}

int update(int root, int pos, int val) {
  int newroot = tot++, tmp = newroot;
  int l = 0, r = m - 1;
  Cnt[newroot] = Cnt[root] + val;
  while (l < r) {
    int mid = (l + r) >> 1;
    if (pos <= mid) {
      lson[newroot] = tot++; rson[newroot] = rson[root];
      newroot = lson[newroot]; root = lson[root];
      r = mid;
    }
    else {
      rson[newroot] = tot++; lson[newroot] = lson[root];
      newroot = rson[newroot]; root = rson[root];
      l = mid + 1;
    }
    Cnt[newroot] = Cnt[root] + val;
  }
  return tmp;
}

inline int lowbit(int x) { return x & (-x); }

int sum(int x) {
  int ret = 0;
  while (x > 0) {
    ret += Cnt[lson[use[x]]];
    x -= lowbit(x);
  }
  return ret;
}

void Modify(int x, int pos, int val) {
  while (x <= n) {
    Ntree[x] = update(Ntree[x], pos, val);
    x += lowbit(x);
  }
}

int Query(int left, int right, int k) {
  int lrt = tree[left - 1];
  int rrt = tree[right];
  int l = 0, r = m - 1;
  for (int i = left - 1; i; i -= lowbit(i)) use[i] = Ntree[i];
  for (int i = right; i; i -= lowbit(i)) use[i] = Ntree[i];
  while (l < r) {
    int mid = (l + r) >> 1;
    // sum(right) - sum(left - 1) 为改变量, Cnt[lson[rrt]] - Cnt[lson[lrt]]
      ↪ 为基础差值
    int tmp = sum(right) - sum(left - 1) + Cnt[lson[rrt]] - Cnt[lson[lrt]];
    if (tmp >= k) {
      r = mid;
      for (int i = left - 1; i; i -= lowbit(i))
        use[i] = lson[use[i]];
      for (int i = right; i; i -= lowbit(i))
        use[i] = lson[use[i]];
      lrt = lson[lrt];
      rrt = lson[rrt];
    }
    else {
      l = mid + 1;
      k -= tmp;
      for (int i = left - 1; i; i -= lowbit(i))
        use[i] = rson[use[i]];
      for (int i = right; i; i -= lowbit(i))
        use[i] = rson[use[i]];
      lrt = rson[lrt];
      rrt = rson[rrt];
    }
  }
  return l;
}

int main()
{
  int Tcase;
  char op[10];
  scanf("%d", &Tcase);
  while (Tcase--) {
    scanf("%d%d", &n, &q);
    tot = 0; m = 0;
    for (int i = 1; i <= n; i++) {
      scanf("%d", &A[i]);
      T[m++] = A[i];
    }
    for (int i = 0; i < q; i++) {
      scanf("%s", op);
      if (op[0] == 'Q') {
        query[i].kind = 0;
        scanf("%d%d%d", &query[i].l, &query[i].r, &query[i].k);
      }
      else {
        query[i].kind = 1;
```

```
        scanf("%d%d", &query[i].l, &query[i].r);
        T[m++] = query[i].r;
      }
    }
    Init_hash(m);
    tree[0] = build(0, m - 1);
    for (int i = 1; i <= n; i++)
      tree[i] = update(tree[i - 1], Hash(A[i]), 1);
    for (int i = 1; i <= n; i++) Ntree[i] = tree[0];
    for (int i = 0; i < q; i++) {
      if (query[i].kind == 0)
        printf("%d\n", T[Query(query[i].l, query[i].r, query[i].k)]);
      else {
        Modify(query[i].l, Hash(A[query[i].l]), -1);
        Modify(query[i].l, Hash(query[i].r), 1);
        A[query[i].l] = query[i].r;
      }
    }
  }
  return 0;
}
```

Treap 树

```
typedef int value;

enum { LEFT, RIGHT };
struct node {
  int size, priority;
  value x, subtree;
  node *child[2];
  node(const value &x): size(1), x(x), subtree(x) {
    priority = rand();
    child[0] = child[1] = nullptr;
  }
};

inline int size(const node *a) { return a == nullptr ? 0 : a->size; }

inline void update(node *a) {
  if (a == nullptr) return;
  a->size = size(a->child[0]) + size(a->child[1]) + 1;
  a->subtree = a->x;
  if (a->child[LEFT] != nullptr) a->subtree = a->child[LEFT]->subtree +
    ↪ a->subtree;
  if (a->child[RIGHT] != nullptr) a->subtree = a->subtree +
    ↪ a->child[RIGHT]->subtree;
}

node *rotate(node *a, bool d) {
  node *b = a->child[d];
  a->child[d] = b->child[!d];
  b->child[!d] = a;
  update(a); update(b);
  return b;
}

node *insert(node *a, int index, const value &x) {
  if (a == nullptr && index == 0) return new node(x);
  int middle = size(a->child[LEFT]);
  bool dir = index > middle;
  if (!dir) a->child[LEFT]  = insert(a->child[LEFT], index, x);
  else      a->child[RIGHT] = insert(a->child[RIGHT], index - middle - 1, x);
  update(a);
  if (a->priority > a->child[dir]->priority) a = rotate(a, dir);
  return a;
}

node *erase(node *a, int index) {
  assert(a != nullptr);
  int middle = size(a->child[LEFT]);
  if (index == middle) {
    if (a->child[LEFT] == nullptr && a->child[RIGHT] == nullptr) {
      delete a;
      return nullptr;
    } else if (a->child[LEFT] == nullptr) a = rotate(a, RIGHT);
    else if (a->child[RIGHT] == nullptr) a = rotate(a, LEFT);
    else a = rotate(a, a->child[LEFT]->priority < a->child[RIGHT]->priority);
    a = erase(a, index);
  } else {
    bool dir = index > middle;
    if (!dir) a->child[LEFT] = erase(a->child[LEFT], index);
    else      a->child[RIGHT] = erase(a->child[RIGHT], index - middle - 1);
  }
  update(a);
  return a;
}

void modify(node *a, int index, const value &x) {
  assert(a != nullptr);
  int middle = size(a->child[LEFT]);
  if (index == middle) a->x = x;
  else {
    bool dir = index > middle;
    if (!dir) modify(a->child[LEFT], index, x);
    else      modify(a->child[RIGHT], index - middle - 1, x);
  }
  update(a);
}

```

```
value query(node *a, int l, int r) {
  assert(a != nullptr);
  if (l <= 0 && size(a) - 1 <= r) return a->subtree;
  int middle = size(a->child[LEFT]);
  if (r < middle) return query(a->child[LEFT], l, r);
  if (middle < l) return query(a->child[RIGHT], l - middle - 1, r - middle -
    ↪ 1);
  value res = a->x;
  if (l < middle && a->child[LEFT] != nullptr)
    res = query(a->child[LEFT], l, r) + res;
  if (middle < r && a->child[RIGHT] != nullptr)
    res = res + query(a->child[RIGHT], l - middle - 1, r - middle - 1);
  return res;
}
```

函数式 Treap

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cmath>
#include <algorithm>
#include <cstdlib>
#include <ctime>
using namespace std;
const int MAXN=100001;
static void read(int &n) {
  char c='+';int x=0;bool flag=0;
  while(c<'0'||c>'9'){c=getchar();if(c=='-')flag=1;}
  while(c>='0'&&c<='9'){x=(x<<1)+(x<<3)+(c-48);c=getchar();}
  flag==1?n=-x:n=x;
}
int ch[MAXN][3];// 0 左孩子 1 右孩子
int val[MAXN];// 每一个点的权值
int pri[MAXN];// 随机生成的附件权值
int siz[MAXN];// 以 i 为节点的树的节点数量
int sz;// 总结点的数量
void update(int x) {
  siz[x]=1+siz[ch[x][0]]+siz[ch[x][1]];
}
int new_node(int v) {
  siz[++sz]=1;// 新开辟一个节点
  val[sz]=v;
  pri[sz]=rand();
  return sz;
}
int merge(int x,int y) {// 合并
  if(!x||!y)    return x+y;// x 和 y 中必定有一个是 0
  if(pri[x]<pri[y])// 把 x 加到左边的树上
  {
    ch[x][1]=merge(ch[x][1],y);// 不懂的看 GIF 图
    update(x);
    return x;
  }
  else
  {
    ch[y][0]=merge(x,ch[y][0]);
    update(y);
    return y;
  }
}
void split(int now,int k,int &x,int &y) {
  if(!now) x=y=0;// 到达叶子节点
  else {
    if(val[now]<=k)// 分离右子树
      x=now,split(ch[now][1],k,ch[now][1],y);
    else
      y=now,split(ch[now][0],k,x,ch[now][0]);
    update(now);
  }
}
int kth(int now,int k) {// 查询排名
  while(1) {
    if(k<=siz[ch[now][0]])
      now=ch[now][0];// 在左子树中，且数量小于左子树的大小，迭代寻找
    else if(k==siz[ch[now][0]]+1)
      return now;// 找到了
    else
      k-=siz[ch[now][0]]+1,now=ch[now][1];// 去右子树找
  }
}
int main() {
  srand((unsigned)time(NULL));
  int n;
  read(n);
  int root=0,x,y,z;
  for(int i=1;i<=n;i++) {
    int how,a;
    read(how);read(a);
    if(how==1) {// 插入
      split(root,a,x,y);
      root=merge(merge(x,new_node(a)),y);
    }
    else if(how==2) {//删除 x
    {
      split(root,a,x,z);
      split(x,a-1,x,y);
      y=merge(ch[y][0],ch[y][1]);
      root=merge(merge(x,y),z);
```

```
    }
    else if(how==3) {//查询 x 的排名
      split(root,a-1,x,y);
      printf("%d\n",siz[x]+1);
      root=merge(x,y);
    }
    else if(how==4) {// 查询排名为 x 的数
      printf("%d\n",val[kth(root,a)]);
    }
    else if(how==5) {// 求 x 的前驱
      split(root,a-1,x,y);
      printf("%d\n",val[kth(x,siz[x])]);
      root=merge(x,y);
    }
    else if(how==6) {// 求 x 的后继
      split(root,a,x,y);
      printf("%d\n",val[kth(y,1)]);
      root=merge(x,y);
    }
  }
  return 0;
}
```

Splay 树

```
// splay tree. HDU 3726: 插入、删除、合并

const int MAXN = 20010;
struct Node;
Node* null;
struct Node {
  Node *ch[2], *fa;//指向儿子和父亲结点
  int size, key;
  Node() {
    ch[0] = ch[1] = fa = null;
  }
  inline void setc(Node* p, int d) {
    ch[d] = p;
    p->fa = this;
  }
  inline bool d() {
    return fa->ch[1] == this;
  }
  void push_up() {
    size = ch[0]->size + ch[1]->size + 1;
  }
  void clear() {
    size = 1;
    ch[0] = ch[1] = fa = null;
  }
  inline bool isroot() {
    return fa == null || this != fa->ch[0] && this != fa->ch[1];
  }
};

inline void rotate(Node* x) {
  Node *f = x->fa, *ff = x->fa->fa;
  int c = x->d(), cc = f->d();
  f->setc(x->ch[!c], c);
  x->setc(f, !c);
  if (ff->ch[cc] == f)ff->setc(x, cc);
  else x->fa = ff;
  f->push_up();
}

inline void splay(Node* &root, Node* x, Node* goal) {
  while (x->fa != goal) {
    if (x->fa->fa == goal)rotate(x);
    else {
      bool f = x->fa->d();
      x->d() == f ? rotate(x->fa) : rotate(x);
      rotate(x);
    }
  }
  x->push_up();
  if (goal == null)root = x;
}

//找到 r 子树里面的第 k 个
Node* get_kth(Node* r, int k) {
  Node* x = r;
  while (x->ch[0]->size + 1 != k) {
    if (k < x->ch[0]->size + 1)x = x->ch[0];
    else {
      k -= x->ch[0]->size + 1;
      x = x->ch[1];
    }
  }
  return x;
}


void erase(Node* &root, Node* x) {
  splay(root, x, null);
  Node* t = root;
  if (t->ch[1] != null) {
    root = t->ch[1];
    splay(root, get_kth(root, 1), null);
    root->setc(t->ch[0], 0);
```

```
  }
  else {
    root = root->ch[0];
  }
  root->fa = null;
  if (root != null)root->push_up();
}

void insert(Node* &root, Node* x) {
  if (root == null) {
    root = x;
    return;
  }
  Node* now = root;
  Node* pre = root->fa;
  while (now != null) {
    pre = now;
    now = now->ch[x->key >= now->key];
  }
  x->clear();
  pre->setc(x, x->key >= pre->key);
  splay(root, x, null);
}

void merge(Node* &A, Node* B) {
  if (A->size <= B->size)swap(A, B);
  queue<Node*>Q;
  Q.push(B);
  while (!Q.empty()) {
    Node* fr = Q.front();
    Q.pop();
    if (fr->ch[0] != null)Q.push(fr->ch[0]);
    if (fr->ch[1] != null)Q.push(fr->ch[1]);
    fr->clear();
    insert(A, fr);
  }
}

Node pool[MAXN], *tail;

struct Edge {
  int u, v;
}edge[60010];
int a[MAXN];
bool del[60010];
struct QUERY {
  char op[10];
  int u, v;
}query[500010];
int y[500010];

Node* node[MAXN];
Node* root[MAXN];
int F[MAXN];
int find(int x) {
  if (F[x] == -1)return x;
  return F[x] = find(F[x]);
}

void debug(Node *root) {
  if (root == null)return;
  debug(root->ch[0]);
  printf("size: %d, key = %d\n", root->size, root->key);
  debug(root->ch[1]);
}

int main()
{
  int n, m;
  int iCase = 0;
  while (scanf("%d%d", &n, &m) == 2) {
    if (n == 0 && m == 0)break;
    iCase++;
    memset(F, -1, sizeof(F));
    tail = pool;
    null = tail++;
    null->size = 0; null->ch[0] = null->ch[1] = null->fa = null;
    null->key = 0;
    for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
    for (int i = 0; i < m; i++) {
      scanf("%d%d", &edge[i].u, &edge[i].v);
      del[i] = false;
    }
    int Q = 0;
    while (1) {
      scanf("%s", &query[Q].op);
      if (query[Q].op[0] == 'E')break;
      if (query[Q].op[0] == 'D') {
        scanf("%d", &query[Q].u);
        query[Q].u--;
        del[query[Q].u] = true;
      }
      else if (query[Q].op[0] == 'Q') {
        scanf("%d%d", &query[Q].u, &query[Q].v);
      }
      else {
        scanf("%d%d", &query[Q].u, &query[Q].v);
        y[Q] = a[query[Q].u];
        a[query[Q].u] = query[Q].v;
      }
```

```cpp
      Q++;
    }
    for (int i = 1; i <= n; i++) {
      node[i] = tail++;
      node[i]->clear();
      node[i]->key = a[i];
      root[i] = node[i];
    }
    for (int i = 0; i < m; i++)
      if (!del[i]) {
        int u = edge[i].u;
        int v = edge[i].v;
        int t1 = find(u);
        int t2 = find(v);
        if (t1 == t2)continue;
        F[t2] = t1;
        merge(root[t1], root[t2]);
      }
    vector<int>ans;
    for (int i = Q - 1; i >= 0; i--) {
      if (query[i].op[0] == 'D') {
        int u = edge[query[i].u].u;
        int v = edge[query[i].u].v;
        int t1 = find(u);
        int t2 = find(v);
        if (t1 == t2)continue;
        F[t2] = t1;
        merge(root[t1], root[t2]);
      }
      else if (query[i].op[0] == 'Q') {
        int u = query[i].u;
        int k = query[i].v;
        u = find(u);
        if (k <= 0 || k > root[u]->size) {
          ans.push_back(0);
        }
        else {
          k = root[u]->size - k + 1;
          Node* p = get_kth(root[u], k);
          ans.push_back(p->key);
        }
      }
      else {
        int u = query[i].u;
        int t1 = find(u);
        Node* p = node[u];
        erase(root[t1], p);
        p->clear();
        p->key = y[i];
        a[u] = y[i];
        insert(root[t1], p);
      }
    }
    double ret = 0;
    int sz = ans.size();
    for (int i = 0; i < sz; i++)ret += ans[i];
    if (sz)ret /= sz;
    printf("Case␣%d:␣%.6lf\n", iCase, ret);
  }
  return 0;
}
```

Splay 树

```cpp
// splay tree: 仅伸展操作
#include<cstdio>
#include<iostream>
#include<algorithm>
#include<cstring>
#include<queue>
using namespace std;

const int maxn = 100005;
struct Node;
Node* null;
struct Node {
  Node *ch[2], *fa;
  int size, rev, key;
  Node() { ch[0] = ch[1] = fa = null; rev = 0; }
  inline void push_up() {
    if (this == null)return;
    size = ch[0]->size + ch[1]->size + 1;
  }
  inline void setc(Node* p, int d) {
    ch[d] = p;
    p->fa = this;
  }
  inline bool d() {
    return fa->ch[1] == this;
  }
  void clear() {
    size = 1;
    ch[0] = ch[1] = fa = null;
    rev = 0;
  }
  void Update_Rev() {
    if (this == null)return;
    swap(ch[0], ch[1]);
    rev ^= 1;
  }
```

```
  inline void push_down() {
    if (this == null)return;
    if (rev) {
      ch[0]->Update_Rev();
      ch[1]->Update_Rev();
      rev = 0;
    }
  }
  inline bool isroot() {
    return fa == null || this != fa->ch[0] && this != fa->ch[1];
  }
};
Node pool[maxn], *tail;
Node *node[maxn], *root;

inline void rotate(Node* x) {
  Node *f = x->fa, *ff = x->fa->fa;
  f->push_down();
  x->push_down();
  int c = x->d(), cc = f->d();
  f->setc(x->ch[!c], c);
  x->setc(f, !c);
  if (ff->ch[cc] == f)ff->setc(x, cc);
  else x->fa = ff;
  f->push_up();
}

inline void splay(Node* &root, Node* x, Node* goal) {
  while (x->fa != goal) {
    if (x->fa->fa == goal) rotate(x);
    else {
      x->fa->fa->push_down();
      x->fa->push_down();
      x->push_down();
      bool f = x->fa->d();
      x->d() == f ? rotate(x->fa) : rotate(x);
      rotate(x);
    }
  }
  x->push_up();
  if (goal == null)root = x;
}

Node* get_kth(Node* r, int k) {
  Node* x = r;
  x->push_down();
  while (x->ch[0]->size + 1 != k) {
    if (k < x->ch[0]->size + 1)x = x->ch[0];
    else {
      k -= x->ch[0]->size + 1;
      x = x->ch[1];
    }
    x->push_down();
  }
  return x;
}

Node* get_next(Node* p) {
  p->push_down();
  p = p->ch[1];
  p->push_down();
  while (p->ch[0] != null) {
    p = p->ch[0];
    p->push_down();
  }
  return p;
}

void build(Node* &x, int l, int r, Node* fa) {
  if (l > r)return;
  int mid = (l + r) >> 1;
  x = tail++;
  x->clear();
  x->fa = fa;
  node[mid] = x;
  build(x->ch[0], l, mid - 1, x);
  build(x->ch[1], mid + 1, r, x);
  x->push_up();
}

void init(int n) {
  tail = pool;
  null = tail++;
  null->fa = null->ch[0] = null->ch[1] = null;
  null->size = 0; null->rev = 0;
  Node *p = tail++;
  p->clear();
  root = p;
  p = tail++;
  p->clear();
  root->setc(p, 1);
  build(root->ch[1]->ch[0], 1, n, root->ch[1]);
  root->ch[1]->push_up();
  root->push_up();
}

int a[maxn], b[maxn];
bool cmp(int i, int j) { return a[i] < a[j] || (a[i] == a[j] && i < j); }

int main() {
```

```
  int n;
  while (scanf("%d", &n), n) {
    for (int i = 1; i <= n; i++) {
      scanf("%d", &a[i]);
      b[i] = i;
    }
    init(n);
    sort(b + 1, b + n + 1, cmp);
    for (int i = 1; i <= n; i++) {
      splay(root, node[b[i]], null);
      int sz = root->ch[0]->size;
      printf("%d", root->ch[0]->size);
      if (i == n) printf("\n");
      else printf(" ");
      splay(root, get_kth(root, i), null);
      splay(root, get_kth(root, sz + 2), root);
      root->ch[1]->ch[0]->Update_Rev();
    }
  }
  return 0;
}
```

Splay 树

```
typedef int value;

enum { LEFT, RIGHT };
struct node {
  node * child[2], * parent;
  value v, subtree;
  int size;
} pool[MAXN], * pool_next = pool;

node * allocate(const value & v) {
  node * x = pool_next++;
  x->parent = x->child[LEFT] = x->child[RIGHT] = nullptr;
  x->subtree = x->v = v;
  x->size = 1;
  return x;
}

struct tree {
  node * root;
  tree(): root(allocate(0)) {}

  bool child_dir(const node * x, const node * y) { return (x->child[LEFT] ==
    ↪ y) ? LEFT : RIGHT; }
  bool is_child(const node * x, const node * y) { return x->child[LEFT] == y
    ↪ || x->child[RIGHT] == y; }

  void update(node * x) {
    x->size = 1;
    x->subtree = x->v;
    FOR (d, 2) if (x->child[d] != nullptr) {
      x->size += x->child[d]->size;
      if (d == LEFT) x->subtree = x->child[LEFT]->subtree + x->subtree;
      else x->subtree = x->subtree + x->child[RIGHT]->subtree;
    }
  }

  void set_child(node * x, bool dir, node * y) {
    if ((x->child[dir] = y) != nullptr) y->parent = x;
    update(x);
  }

  node * rotate(node * x, bool dir) {
    node * parent = x->parent, * y = x->child[dir];
    set_child(x, dir, y->child[!dir]);
    set_child(y, !dir, x);
    set_child(parent, child_dir(parent, x), y);
    return y;
  }

  node * splay(node * x) {
    node * old_p = nullptr;
    while (x->parent != nullptr) {
      node * p = x->parent;
      x = rotate(p, child_dir(p, x));
      if (old_p != nullptr && is_child(p, old_p)) rotate(p, child_dir(p,
        ↪ old_p));
      old_p = p;
    }
    return x;
  }

  node * insert(int order, const value & v) { // order is 0-indexed
    bool dir = LEFT;
    node * parent = root, * x = parent->child[LEFT];
    while (x != nullptr) {
      int left_size = (x->child[LEFT] == nullptr) ? 0 : x->child[LEFT]->size;
      parent = x;
      if (order <= left_size) x = x->child[dir = LEFT];
      else {
        order -= left_size + 1;
        x = x->child[dir = RIGHT];
      }
    }
    set_child(parent, dir, x = allocate(v));
    return splay(x);
  }
```

```cpp

  node * find(int order) {
    node * x = root->child[LEFT];
    while (true) {
      int left_size = (x->child[LEFT] == nullptr) ? 0 : x->child[LEFT]->size;
      if (order < left_size) x = x->child[LEFT];
      else if (order == left_size) break;
      else {
        order -= left_size + 1;
        x = x->child[RIGHT];
      }
    }
    return splay(x);
  }

  void erase(const int& order) {
    node * x = find(order);
    if (x->child[LEFT] == nullptr) set_child(root, LEFT, x->child[RIGHT]);
    else if (x->child[RIGHT] == nullptr) set_child(root, LEFT,
     ↪ x->child[LEFT]);
    else {
      node * y = x->child[RIGHT];
      while (y->child[LEFT] != nullptr) y = y->child[LEFT];
      y = splay(y);
      set_child(y, LEFT, x->child[LEFT]);
      set_child(root, LEFT, y);
    }
  }

  value query(int e) { // e is the prefix length desired.
    node * x = root->child[LEFT];
    if (e <= 0) return 0;
    if (e >= x->size) return x->subtree;
    x = find(e - 1);
    if (x->child[LEFT] != nullptr) return x->child[LEFT]->subtree * x->v;
    else return x->v;
  }
};
```

点分治

```cpp
const int maxn = "Edit";

struct Edge {
  int to, nxt, dis;
} g[maxn];
int head[maxn], cnt, f[maxn], dd[maxn], size[maxn], d[maxn];
int n, k, rt, ans, con, len;
bool vis[maxn];

void add(int u, int v, int dis) {
  g[++ cnt] = (Edge){v, head[u], dis};
  head[u] = cnt;
}

void add_edge(int u, int v, int dis) {
  add(u, v, dis);
  add(v, u, dis);
}

void clr(){
  for(int i = 1; i <= n; i ++) {
    vis[i] = f[i] = size[i] = head[i] = dd[i] = 0;
  }
  cnt = rt = 0, f[0] = 1e9, con = n, len = ans = 0;
}

void getrt(int u, int fafa){
  size[u] = 1;
  f[u] = 0;
  for(int i = head[u]; i; i = g[i].nxt){
    int v = g[i].to; if(v == fafa || vis[v]) continue;
    getrt(v, u);
    size[u] += size[v];
    f[u] = std::max(f[u], size[v]);
  }
  f[u] = std::max(f[u], con - size[u]);
  if(f[u] < f[rt]) {
    rt = u;
  }
}

void getdis(int u, int fafa){
  size[u] = 1;
  dd[++ len] = d[u];
  for(int i = head[u]; i; i = g[i].nxt){
    int v = g[i].to; if(v == fafa || vis[v]) continue;
    d[v] = d[u] + g[i].dis; getdis(v, u);
    size[u] += size[v];
  }
}

int cal(int u, int w){
  len = 0; d[u] = w; getdis(u, 0);
  std::sort(dd + 1, dd + len + 1);
  int l = 1, r = len, sum = 0;
  while(l < r){
    if(dd[l] + dd[r] <= k) sum += r - l, l ++;
    else r --;
  }
```

```
  return sum;
}

void solve(int u){
  vis[u] = 1; ans += cal(u, 0);
  for(int i = head[u]; i; i = g[i].nxt){
    int v = g[i].to; if(vis[v]) continue;
    ans -= cal(v, g[i].dis);
    rt = 0; con = size[v];
    getrt(v, 0);
    solve(rt);
  }
}
```

树上启发式合并

```
// 树上启发式合并: dsu on tree
int n, x, y, Son, Max;
int sz[maxn], son[maxn];
ll sum, ans[maxn];
vector<int> v[maxn];

void getson(int u, int fa) {
  sz[u] = 1;
  for (int i = 0; i < v[u].size(); i++) {
    int to = v[u][i];
    if (to != fa) {
      getson(to, u);
      sz[u] += sz[to];
      if (sz[to] > sz[son[u]])
        son[u] = to;
    }
  }
}

void add(int u, int fa, int val) {
  // 更新节点数据
  // cnt[attr[u]] += val;
  for (int i = 0; i < v[u].size(); i++) {
    int to = v[u][i];
    if (to != fa && to != Son)
      add(to, u, val);
  }
}

void dfs(int u, int fa, int k) {
  for (int i = 0; i < v[u].size(); i++) {
    int to = v[u][i];
    if (to != fa && to != son[u])
      dfs(to, u, 0);
  }
  if (son[u]) dfs(son[u], u, 1), Son = son[u];
  add(u, fa, 1); Son = 0;
  // 此处统计 u 节点处答案
  // ans[u] = sum;
  if (!k) add(u, fa, -1), Max = sum = 0;
}

// getson(1, 0);
// dfs(1, 0, 0);
```

0-1trie 区间异或最大值

```
// written by calabash_boy
// 01Trie 求区间异或和的最大值

#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
const int MAX = 1e6+100;
int bas[35];
const int INF = 2147483645;

struct Trie {
  int nxt[MAX<<2][2];  int l[MAX<<2];
  int cnt;  int ansl,ansr,ansv;
  void init() {
    cnt =0;
    memset(nxt[0],0,sizeof (nxt[0]));
    memset(l,0x3f3f3f3f,sizeof(l));
    ansv = 0;
  }
  int create() {
    cnt++;
    memset(nxt[cnt],0,sizeof (nxt[cnt]));
    return cnt;
  }
  void insert(int id,int x) {
    int y = 0;
    for (int i=30;i>=0;i--) {
      int t = x&bas[i];
      t>>=i;
      if (!nxt[y][t]) {
        nxt[y][t] = create();
      }
      y = nxt[y][t];
    }
    l[y] = min(l[y],id);
```

```
  }
  void query(int id,int x) {
    int y=0;  int res =0;
    for (int i=30;i>=0;i--) {
      int t = x&bas[i];
      t>>=i;
      if (nxt[y][!t]) {
        y =nxt[y][!t];
        res+=bas[i];
      } else{
        y = nxt[y][t];
      }
    }
    if (res==ansv) {
      if (l[y]<ansl) {
        ansl = l[y];  ansr = id;
      }
    } else if (res>ansv) {
      ansv = res;
      ansl = l[y];
      ansr = id;
    }
  }
  void print(int id) {
    printf("Case #%d:\n%d %d\n",id,ansl+1,ansr);
  }
}trie;

void init() {
  bas[0] = 1;
  for (int i=1;i<=30;i++) {
    bas[i] = bas[i-1]<<1;
  }
}
int main() {
  init();
  int n, Cas;
  scanf("%d",&Cas);
  for (int i=1;i<=Cas;i++) {
    trie.init();  trie.insert(0,0);
    scanf("%d",&n);
    int sum=0;
    for (int j=1;j<=n;j++) {
      int ai;
      scanf("%d",&ai);  sum^=ai;
      trie.query(j,sum);  trie.insert(j,sum);
    }
    trie.print(i);
  }
  return 0;
}
```

0-1trie 子树异或最大值

```
// 可持久化 01Trie+DFS 序 子树上的点抑或最大值:
// written by calabash_boy

#include <iostream>
#include <cstdio>
using namespace std;
const int MAX = 1e5+100;
int bas[35];  int nxt[MAX<<5][2];
int root[MAX];  int sum[MAX<<5];
int n,q;  vector<int>E[MAX];
int st[MAX],en[MAX],rk[MAX];
int a[MAX];  int cnt;  int tot;
void sheet(){
  bas[0]=1;
  for (int i=1;i<=30;i++){
    bas[i] = bas[i-1]<<1;
  }
}
void init(){
  for (int i=0;i<=n;i++){  E[i].clear(); }
  cnt =tot=0;
  memset(nxt[0],0,sizeof nxt[0]);
}
void input(){
  for (int i=1;i<=n;i++){  scanf("%d",a+i);  }
  for (int u=2;u<=n;u++){
    int v;  scanf("%d",&v);
    E[u].push_back(v);  E[v].push_back(u);
  }
}
void dfs(int node ,int  father ){
  st[node] = ++tot;  rk[tot] = node;
  for (int des:E[node]){
    if(des==father){  continue; }
    dfs(des,node);
  }
  en[node] = tot;
}
int create(){
  cnt++;
  memset(nxt[cnt],0,sizeof nxt[cnt]);
  return cnt;
}
int insert(int rt,int val){
  int y = ++cnt;  int x = rt;  int res = y;
  for (int i=30;i>=0;i--){
```

```
    sum[y] = sum[x]+1;
    nxt[y][0] = nxt[x][0];  nxt[y][1] = nxt[x][1];
    int t = val&bas[i];
    t>>=i;
    nxt[y][t] = create();
    y = nxt[y][t];  x = nxt[x][t];
  }
  sum[y] = sum[x]+1;
  return res;
}
int query(int l,int r,int val){
  int res =0; int x = l; int y = r;
  for (int i=30;i>=0;i--){
    int t = val&bas[i];
    t>>=i;
    if (sum[nxt[y][!t]]-sum[nxt[x][!t]]){
      y = nxt[y][!t];  x = nxt[x][!t];
      res+=bas[i];
    }else{
      y = nxt[y][t];  x = nxt[x][t];
    }
  }
  return res;
}
void solve(){
  dfs(1,0);
  for (int i=1;i<=n;i++){
    root[i] = insert(root[i-1],a[rk[i]]);
  }
  while (q--){
    int nod,x;
    scanf("%d%d",&nod,&x);
    printf("%d\n",query(root[st[nod]-1],root[en[nod]],x));
  }
}
int main(){
  sheet();
  while (scanf("%d%d",&n,&q)!=EOF){
    init();
    input();
    solve();
  }
  return 0;
}
```

莫队算法

```
//Author:marszed
/*
* 离线区间处理问题。
* 从区间 [l,r] 得到区间 [l+1,r+1] [l-1,r-1] 信息的转移复杂度为 O(1)。
*siz 为块大小。
*cnt 为位于第几个块。
*modify() 函数为转移函数。
*/

#include <iostream>
#include <algorithm>
#include <cmath>

const int maxn = 2e5 + 10;

int n, siz, q;
int a[maxn];

struct Node {
  int id, l, r, val, cnt;

  int operator< (const Node& b) {
    return cnt == b.cnt ? r < b.r : cnt < b.cnt;
  }
} nod[maxn];

void modify(int i, int flag) {

}

void mo() {
  std::cin >> n >> q;
  siz = sqrt(n);
  for (int i = 1; i <= n; i++) {
    std::cin >> a[i];
  }
  for (int i = 1; i <= q; i++) {
    std::cin >> nod[i].l >> nod[i].r;
    nod[i].id = i;
    nod[i].cnt = nod[i].l / siz;
  }
  std::sort(nod + 1, nod + q + 1);
  int l = 0, r = 0;
  for (int i = 1; i <= q; i++) {
    while (l < nod[i].l - 1)  modify(++l, 1);
    while (l >= nod[i].l)     modify(l--, 1);
    while (r < nod[i].r)     modify(++r, 1);
    while (r > nod[i].r)     modify(r--, 1);
    ans[nod[i].id] = Ans;
  }
}

```

```
int main() {}
```

最近公共祖先（在线）

```
// 时间复杂度 O(nlogn+q)
// By CSL

const int maxn = "Edit";
std::vector<int> G[maxn], sp;
int dep[maxn], dfn[maxn];

std::pair<int, int> dp[21][maxn << 1];

void init(int n) {
  for (int i = 0; i < n; i++) G[i].clear();
  sp.clear();
}

void dfs(int u, int fa) {
  dep[u] = dep[fa] + 1;
  dfn[u] = sp.size();
  sp.push_back(u);
  for (auto& v : G[u]) {
    if (v == fa) continue;
    dfs(v, u);
    sp.push_back(u);
  }
}

void initrmq() {
  int n = sp.size();
  for (int i = 0; i < n; i++) dp[0][i] = {dfn[sp[i]], sp[i]};
  for (int i = 1; (1 << i) <= n; i++)
    for (int j = 0; j + (1 << i) - 1 < n; j++)
      dp[i][j] = std::min(dp[i - 1][j], dp[i - 1][j + (1 << (i - 1))]);
}

int lca(int u, int v) {
  int l = dfn[u], r = dfn[v];
  if (l > r) std::swap(l, r);
  int k = 31 - __builtin_clz(r - l + 1);
  return std::min(dp[k][l], dp[k][r - (1 << k) + 1]).second;
}
```

最近公共祖先（离线）

```
// 时间复杂度 O(n+q)
// By CSL

#include <iostream>
#include <algorithm>
#include <vector>

const int maxn = "Edit";
int par[maxn];                       //并查集
int ans[maxn];                       //存储答案
std::vector<int> G[maxn];              //邻接表
std::vector<std::pair<int, int>> query[maxn];  //存储查询信息
bool vis[maxn];                      //是否被遍历

inline void init(int n) {
  for (int i = 1; i <= n; i++) {
    G[i].clear(), query[i].clear();
    par[i] = i, vis[i] = 0;
  }
}

int find(int u) {
  return par[u] == u ? par[u] : par[u] = find(par[u]);
}

void unite(int u, int v) {
  par[find(v)] = find(u);
}

inline void add_edge(int u, int v) {
  G[u].push_back(v);
}

inline void add_query(int id, int u, int v) {
  query[u].push_back(std::make_pair(v, id));
  query[v].push_back(std::make_pair(u, id));
}

void tarjan(int u) {
  vis[u] = 1;
  for (auto& v : G[u]) {
    if (vis[v]) continue;
    tarjan(v);
    unite(u, v);
  }
  for (auto& q : query[u]) {
    int &v = q.first, &id = q.second;
    if (!vis[v]) continue;
    ans[id] = find(v);
  }
}
```

最近公共祖先

```cpp
// LCA ST 算法
int n, top, root;
int a[maxn << 1], d[maxn], st[maxn];
int f[maxn << 1][18], loc[maxn << 1][18];
vector<int> v[maxn];

int log2(int x) {
  int k = 0;
  while (x > 1) {
    x /= 2;
    k++;
  }
  return k;
}

void dfs(int u, int dep) {
  d[u] = dep;
  a[++top] = u;
  for (int i = 0; i<=v[u].size(); i++) {
    int to = v[u][i];
    dfs(to, dep + 1);
    a[++top] = u;
  }
}

void init() {
  int s = log2(top);
  for (int i = 1; i <= top; i++) {
    f[i][0] = d[a[i]];
    loc[i][0] = a[i];
  }
  for (int j = 1; j <= s; j++) {
    int k = top - (1 << j) + 1;
    for (int i = 1; i <= k; i++) {
      int x = i + (1 << (j - 1));
      if (f[i][j - 1] <= f[x][j - 1]) {
        f[i][j] = f[i][j - 1];
        loc[i][j] = loc[i][j - 1];
      }
      else {
        f[i][j] = f[x][j - 1];
        loc[i][j] = loc[x][j - 1];
      }
    }
  }
}

int query(int x, int y) {
  x = st[x], y = st[y];
  if (x > y) swap(x, y);
  int i = log2(y - x);
  int k = y - (1 << i) + 1;
  return f[x][i] < f[k][i] ? loc[x][i] : loc[k][i];
}

//==============================================================================

// LCA Tarjan 算法
int n, root, cnt;
int pre[maxn], ans[maxn];
vector<int> v[maxn], s[maxn], num[maxn];

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

void dfs(int u) {
  pre[u] = u;
  for (int i = 0; i < v[u].size(); i++) {
    int to = v[u][i];
    dfs(to);
    pre[find(pre[to])] = find(pre[u]);
  }
  for (int i = 0; i < s[u].size(); i++) {
    int to = s[u][i];
    if (pre[to] != to)
      ans[num[u][i]] = find(pre[to]);
  }
}

/*
for (int i = 1; i <= q; i++) {
  scanf("%d%d", &x, &y);
  if (x == y) ans[i] = x;
  s[x].push_back(y);
  s[y].push_back(x);
  num[x].push_back(i);
  num[y].push_back(i);
}
dfs(root);
*/

//==============================================================================

// LCA 倍增算法
int n, ma, root;
int d[maxn], f[maxn][20];
vector<int> v[maxn];
inline void dfs(int u, int dep, int fa) {
  d[u] = dep;
```

```cpp
  f[u][0] = fa;
  ma = max(ma, dep);
  for (int i = 0; i < v[u].size(); i++)
    if (v[u][i] != fa) dfs(v[u][i], dep + 1, u);
}
inline int log2(int x) {
  int k = 0;
  while (x > 1) {
    x >>= 1;
    k++;
  }
  return k;
}
inline void init() {
  dfs(root, 0, 0);
  int s = log2(ma);
  for (int j = 1; j <= s; j++)
    for (int i = 1; i <= n; i++)
      f[i][j] = f[f[i][j - 1]][j - 1];
}
// 求 x 与 y 的 LCA
inline int query(int x, int y) {
  if (d[x] < d[y]) swap(x, y);
  int s = log2(d[x] - d[y]);
  while (d[x] > d[y]) {
    if (d[x] - (1 << s) >= d[y])
      x = f[x][s];
    s--;
  }
  s = log2(d[x]);
  while (s > -1) {
    if (f[x][s] != f[y][s]) {
      x = f[x][s];
      y = f[y][s];
    }
    s--;
  }
  return x == y ? x : f[x][0];
}
// 判断 a 与 p 是否在同一树边上 (p 在 a 上方)
inline bool check(int a, int p) {
  if (d[a] < d[p]) return false;
  if (d[a] == d[p]) return a == p;
  int s = log2(d[a] - d[p]);
  while (d[a] > d[p]) {
    if (d[a] - (1 << s) >= d[p])
      a = f[a][s];
    s--;
  }
  return a == p;
}
// 求一条树边上 x 到 y 的距离
inline int getlen(int x, int y) {
  int ret = 0;
  if (d[x] < d[y]) swap(x, y);
  int s = log2(d[x] - d[y]);
  while (d[x] > d[y]) {
    if (d[x] - (1 << s) >= d[y]) {
      ret += (1 << s);
      x = f[x][s];
    }
    s--;
  }
  return ret;
}
```

树链剖分

```cpp
// 树链剖分 点权
/**
* top[v] 表示 v 所在的重链的顶端节点
* fa[v] 表示 v 的父节点
* deep[v] 表示 v 的深度 (根的深度为 1)
* snum[v] 表示以 v 为根的子树的节点数
* p[v] 表示 v 所在 (线段树中) 的位置
* fp[v] 与 p[v] 相反, 表示对应位置的节点
* son[v] 表示 v 的重儿子
* Edge 存树边
**/

struct Edge {
  int to, next;
}edge[maxn << 1];

int pos, n, m, tot;  // n 为节点数
int head[maxn], top[maxn], fa[maxn], deep[maxn], num[maxn], p[maxn], fp[maxn], son[maxn];

void init() {
  tot = 0;
  pos = 1;
  memset(head, -1, sizeof(head));
  memset(son, -1, sizeof(son));
  for (int i = 0; i <= n; i++)
    v[i].clear();
}

void addedge(int u, int v) {
  edge[tot].to = v;
```

```
  edge[tot].next = head[u];
  head[u] = tot++;
}

void dfs1(int u, int pre, int d) {
  deep[u] = d;
  fa[u] = pre;
  num[u] = 1;
  for (int i = head[u]; i != -1; i = edge[i].next) {
    int to = edge[i].to;
    if (to != pre) {
      dfs1(to, u, d + 1);
      num[u] += num[to];
      if (son[u] == -1 || num[to] > num[son[u]])
        son[u] = to;
    }
  }
}

void dfs2(int u, int sp) {
  top[u] = sp;
  p[u] = pos++;
  fp[p[u]] = u;
  if (son[u] == -1) return;
  dfs2(son[u], sp);
  for (int i = head[u]; i != -1; i = edge[i].next) {
    int to = edge[i].to;
    if (to != son[u] && to != fa[u])
      dfs2(to, to);
  }
}
/*
// 使用范例
int getsum(int a, int b) {
  int f1 = top[a], f2 = top[b];
  int ret = 0;
  while (f1 != f2) {
    if (deep[f1] < deep[f2]) {
      swap(f1, f2);
      swap(a, b);
    }
    ret += query(p[f1], p[a], 1, n - 1, 1);
    a = fa[f1]; f1 = top[a];
  }
  if (a == b) return ret;
  if (deep[a] > deep[b]) swap(a, b);
  return ret + query(p[son[a]], p[b], 1, n - 1, 1);
}
*/
```

字符串

KMP

```
//Author:CookiC
//返回下标最大的匹配串
#include <cstring>

void getFail(char *P, int *f) {
  int i, j;
  f[0] = 0;
  f[1] = 0;
  for(i=1; P[i]; ++i) {
    j = f[i];
    while(j && P[i]!=P[j]) {
      j = f[j];
    }
    f[i+1] = P[i]==P[j]? j+1: 0;
  }
}

int kmp(char *T, char *P) {
  int ans = -1;
  int n = strlen(T), m = strlen(P);
  int f[maxn];
  getFail(P, f);
  int j = 0;
  for(int i=0; i<n; ++i){
    while(j && P[j]!=T[i])
    j = f[j];
    if(P[j]==T[i]) {
      ++j;
    }
    if(j==m) {
      j = f[j];
      ans = i-m+1;
    }
  }
  return ans;
}


/*
* Author: Simon * 复 杂 度: O(n)
*/
int Next[maxn]; /*i 之前相同前缀后缀的长度，  例 ababc,Next[5]=2; */
void getNext(int m, char p[]) {
  memset(Next, 0, sizeof(int)*(m+5));
  int k=-1,j=0;
  Next[0]=-1;
  while (j<m) {
    if (k==-1||p[k]==p[j]) {
    k++,j++;
    if (p[k]!=p[j]) Next[j]=k;
    else Next[j]=Next[k];
    }
    else k=Next[k];
  }
}

int KMP(int n,int m,char s[],char p[]){
  int i=0,j=0,ans=0;
  while(i<n){
    if(j==-1||s[i]==p[j]) i++,j++;
    else j=Next[j];
    if(j==m) ans++; /* 计 数 (可 重 叠)*/
    //if(j==m) ans++,j=0;/* 计 数 ( 不 可 重 叠 )*/
    //if(j==m) return i-m+1; /* 返回第一个匹配的位置 */
  }
  //return j;/* 返回 s 后缀与 p 前缀匹配的最长长度 */
  return ans;
}
```

扩展 KMP

```
const int maxn = "Edit";
int ans, nexr[maxn], ex[maxn];
void getnexr(char s[]) {
  int i = 0, j, po, len = strlen(s);
  nexr[0] = len;
  while (s[i] == s[i+1] && i + 1 < len) i++;
  nexr[1] = i;
  po = 1;
  for (i = 2; i < len; i++) {
    if (nexr[i-po] + i < nexr[po] + po) {
      nexr[i] = nexr[i - po];
    } else {
       j = nexr[po] + po - i;
       if (j < 0) j = 0;
       while (i + j < len && s[j] == s[i+j]) j++;
       nexr[i] = j;
       po = i;
     }
  }
}

void exkmp(char s1[], char s2[]) {
  int i = 0, j, po = 0, len = strlen(s1), l2 = strlen(s2);
  while (s1[i] == s2[i] && i < l2 && i < len) i++;
  ex[0] = i;
  for (i = 1; i < len; i++) {
```

```
    if (nexr[i - po] + i < ex[po] + po) {
      ex[i] = nexr[i-po];
    } else {
       j = ex[po] + po - i;
       if (j < 0) j = 0;
       while (i + 1 < len && s1[j+i] == s2[j]) j++;
       ex[i] = j;
       po = i;
     }
  }
}
```

扩展 KMP

```
//解决如下问题：定义母串 s 和子串 T，设 s 的长度为 n,T 的长度为 m，求 T 与
  ↪ S 的每一个后缀的最长公共前缀
//extend[i] 表示 T 与 S[i,n-1] 的最长公共前缀。(0<=i<n)
//一个辅助工具为 nxt[i] 表示 T[i,m-1] 和 T 的最长公共前缀长度
//下标都从 0 开始

O(n+m)

const int maxn=100010;   //字符串长度最大值
int nxt[maxn],ex[maxn]; //ex 数组即为 extend 数组
//预处理计算 next 数组
void GETNEXT(char *str)
{
  int i=0,j,po,len=strlen(str);
  nxt[0]=len;//初始化 next[0]
  while(str[i]==str[i+1]&&i+1<len)//计算 next[1]
  i++;
  nxt[1]=i;
  po=1;//初始化 po 的位置
  for(i=2;i<len;i++)
  {
    if(nxt[i-po]+i<nxt[po]+po)//第一种情况，可以直接得到 next[i] 的值
    nxt[i]=nxt[i-po];
    else//第二种情况，要继续匹配才能得到 next[i] 的值
    {
      j=nxt[po]+po-i;
      if(j<0)j=0;//如果 i>po+nxt[po]，则要从头开始匹配
      while(i+j<len&&str[j]==str[j+i])//计算 next[i]
      j++;
      nxt[i]=j;
      po=i;//更新 po 的位置
    }
  }
}
//计算 extend 数组
void EXKMP(char *s1,char *s2)
{
  int i=0,j,po,len=strlen(s1),l2=strlen(s2);
  GETNEXT(s2);//计算子串的 next 数组
  while(s1[i]==s2[i]&&i<l2&&i<len)//计算 ex[0]
  i++;
  ex[0]=i;
  po=0;//初始化 po 的位置
  for(i=1;i<len;i++)
  {
    if(nxt[i-po]+i<ex[po]+po)//第一种情况，直接可以得到 ex[i] 的值
    ex[i]=nxt[i-po];
    else//第二种情况，要继续匹配才能得到 ex[i] 的值
    {
      j=ex[po]+po-i;
      if(j<0)j=0;//如果 i>ex[po]+po 则要从头开始匹配
      while(i+j<len&&j<l2&&s1[j+i]==s2[j])//计算 ex[i]
      j++;
      ex[i]=j;
      po=i;//更新 po 的位置
    }
  }
}
```

TRIE

```
int tree[maxn][26];
int sum[maxn];
int tot;
void Insert(char *str) {
  int len = strlen(str);
  int root = 0;
  for (int i = 0; i < len; i++) {
    int id = str[i] - 'a';
    if (!tree[root][id]) tree[root][id] = ++tot;
    sum[tree[root][id]]++;
    root = tree[root][id];
  }
}

int Find(char *str) {
  int len = strlen(str);
  int root = 0;
  for (int i = 0; i < len; i++) {
    int id = str[i] - 'a';
    if (!tree[root][id]) return 0;
    root = tree[root][id];
  }
```

```
  return sum[root];
}
```

AC 自动机

```
#include <cstdio>
#include <iostream>
#include <algorithm>
#include <cstring>
#include <queue>
using namespace std;

struct Trie {
  int next[500010][26], fail[500010], end[500010];
  int root, L;
  int newnode() {
    for (int i = 0; i < 26; i++)
      next[L][i] = -1;
    end[L++] = 0;
    return L - 1;
  }
  void init() {
    L = 0;
    root = newnode();
  }
  void insert(char buf[]) {
    int len = strlen(buf);
    int now = root;
    for (int i = 0; i < len; i++) {
      if (next[now][buf[i] - 'a'] == -1)
        next[now][buf[i] - 'a'] = newnode();
      now = next[now][buf[i] - 'a'];
    }
    end[now]++;
  }
  void build() {
    queue<int>Q;
    fail[root] = root;
    for (int i = 0; i < 26; i++)
      if (next[root][i] == -1)
        next[root][i] = root;
      else {
        fail[next[root][i]] = root;
        Q.push(next[root][i]);
      }
      while (!Q.empty()) {
        int now = Q.front();
        Q.pop();
        for (int i = 0; i < 26; i++)
          if (next[now][i] == -1)
            next[now][i] = next[fail[now]][i];
          else
          {
            next[now][i] = next[fail[now]][i];
            fail[next[now][i]] = next[fail[now]][i];
            Q.push(next[now][i]);
          }
      }
  }
  // 查询 buf 字符串包含的模板串
  int query(char buf[]) {
    int len = strlen(buf);
    int now = root;
    int res = 0;
    for (int i = 0; i < len; i++) {
      now = next[now][buf[i] - 'a'];
      int temp = now;
      while (temp != root) {
        res += end[temp];
        end[temp] = 0;
        temp = fail[temp];
      }
    }
    return res;
  }
};
char buf[1000010];
```

后缀数组（倍增）

```
//author: Menci
#include <algorithm>
#include <string>
#include <iostream>

const int maxn = 1000;

char s[maxn];
int n, ht[maxn], rk[maxn], sa[maxn];

inline void suffixArray() {
  static int set[maxn + 1], a[maxn + 1];
  std::copy(s, s + n, set + 1);
  std::sort(set + 1, set + n + 1);
  int *end = std::unique(set + 1, set + n + 1);
  for (int i = 1; i <= n; i++) a[i] = std::lower_bound(set + 1, end, s[i]) -
    set;

  static int fir[maxn + 1], sec[maxn + 1], tmp[maxn + 1], buc[maxn + 1];
  for (int i = 1; i <= n; i++) buc[a[i]]++;
```

```cpp
  for (int i = 1; i <= n; i++) buc[i] += buc[i - 1];
  for (int i = 1; i <= n; i++) rk[i] = buc[a[i] - 1] + 1;

  for (int t = 1; t <= n; t *= 2) {
    for (int i = 1; i <= n; i++) fir[i] = rk[i];
    for (int i = 1; i <= n; i++) sec[i] = i + t > n ? 0 : rk[i + t];

    std::fill(buc, buc + n + 1, 0);
    for (int i = 1; i <= n; i++) buc[sec[i]]++;
    for (int i = 1; i <= n; i++) buc[i] += buc[i - 1];
    for (int i = 1; i <= n; i++) tmp[n - --buc[sec[i]]] = i;

    std::fill(buc, buc + n + 1, 0);
    for (int i = 1; i <= n; i++) buc[fir[i]]++;
    for (int i = 1; i <= n; i++) buc[i] += buc[i - 1];
    for (int j = 1, i; j <= n; j++) i = tmp[j], sa[buc[fir[i]]--] = i;

    bool unique = true;
    for (int j = 1, i, last = 0; j <= n; j++) {
      i = sa[j];
      if (!last) rk[i] = 1;
      else if (fir[i] == fir[last] && sec[i] == sec[last]) rk[i] = rk[last], unique = false;
      else rk[i] = rk[last] + 1;

      last = i;
    }

    if (unique) break;
  }

  for (int i = 1, k = 0; i <= n; i++) {
    if (rk[i] == 1) k = 0;
    else {
      if (k > 0) k--;
      int j = sa[rk[i] - 1];
      while (i + k <= n && j + k <= n && a[i + k] == a[j + k]) k++;
    }
    ht[rk[i]] = k;
  }
}

int main() {
  std::cin >> n >> s;
  suffixArray();
  for (int i = 1; i <= n; i++) {
    std::cout << sa[i] << " ";
  }
}
```

后缀数组 (sais)

```cpp
namespace SA {
  int sa[N], rk[N], ht[N], s[N<<1], t[N<<1], p[N], cnt[N], cur[N];
  #define pushS(x) sa[cur[s[x]]--] = x
  #define pushL(x) sa[cur[s[x]]++] = x
  #define inducedSort(v) std::fill_n(sa, n, -1); std::fill_n(cnt, m, 0); \
    for (int i = 0; i < n; i++) cnt[s[i]]++; \
    for (int i = 1; i < m; i++) cnt[i] += cnt[i-1]; \
    for (int i = 0; i < m; i++) cur[i] = cnt[i]-1; \
    for (int i = n1-1; ~i; i--) pushS(v[i]); \
    for (int i = 1; i < m; i++) cur[i] = cnt[i-1]; \
    for (int i = 0; i < n; i++) if (sa[i] > 0 &&  t[sa[i]-1]) pushL(sa[i]-1); \
    for (int i = 0; i < m; i++) cur[i] = cnt[i]-1; \
    for (int i = n-1;  ~i; i--) if (sa[i] > 0 && !t[sa[i]-1]) pushS(sa[i]-1)
  void sais(int n, int m, int *s, int *t, int *p) {
    int n1 = t[n-1] = 0, ch = rk[0] = -1, *s1 = s+n;
    for (int i = n-2; ~i; i--) t[i] = s[i] == s[i+1] ? t[i+1] : s[i] > s[i+1];
    for (int i = 1; i < n; i++) rk[i] = t[i-1] && !t[i] ? (p[n1] = i, n1++) : -1;
    inducedSort(p);
    for (int i = 0, x, y; i < n; i++) if (~(x = rk[sa[i]])) {
      if (ch < 1 || p[x+1] - p[x] != p[y+1] - p[y]) ch++;
      else for (int j = p[x], k = p[y]; j <= p[x+1]; j++, k++)
        if ((s[j]<<1|t[j]) != (s[k]<<1|t[k])) {ch++; break;}
      s1[y = x] = ch;
    }
    if (ch+1 < n1) sais(n1, ch+1, s1, t+n, p+n1);
    else for (int i = 0; i < n1; i++) sa[s1[i]] = i;
    for (int i = 0; i < n1; i++) s1[i] = p[sa[i]];
    inducedSort(s1);
  }
  template<typename T>
  int mapCharToInt(int n, const T *str) {
    int m = *max_element(str, str+n);
    std::fill_n(rk, m+1, 0);
    for (int i = 0; i < n; i++) rk[str[i]] = 1;
    for (int i = 0; i < m; i++) rk[i+1] += rk[i];
    for (int i = 0; i < n; i++) s[i] = rk[str[i]] - 1;
    return rk[m];
  }
```

```
  // Ensure that str[n] is the unique lexicographically smallest character in
    ↪ str.
  template<typename T>
  void suffixArray(int n, const T *str) {
    int m = mapCharToInt(++n, str);
    sais(n, m, s, t, p);
    for (int i = 0; i < n; i++) rk[sa[i]] = i;
    for (int i = 0, h = ht[0] = 0; i < n-1; i++) {
      int j = sa[rk[i]-1];
      while (i+h < n && j+h < n && s[i+h] == s[j+h]) h++;
      if (ht[rk[i]] = h) h--;
    }
  }
};
```

后缀自动机

```
//Author:CookiC
#include<cstring>
#define MAXN 10000

struct State{
  State *f,*c[26];
  int len;
};

State *root,*last,*cur;
State StatePool[MAXN];

State* NewState(int len){
  cur->len=len;
  cur->f=0;
  memset(cur->c,0,sizeof(cur->c));
  return cur++;
}

void Init(){
  cur=StatePool;
  last=StatePool;
  root=NewState(0);
}

void Extend(int w){
  State *p = last;
  State *np = NewState(p->len+1);
  while(p&&!p->c[w]) {
    p->c[w] = np;
    p = p->f;
  }
  if(!p) {
    np->f=root;
  } else {
    State *q=p->c[w];
    if(p->len+1==q->len) {
      np->f=q;
    } else {
      State *nq = NewState(p->len+1);
      memcpy(nq->c, q->c, sizeof(q->c));
      nq->f = q->f;
      q->f = nq;
      np->f = nq;
      while(p&&p->c[w]==q) {
        p->c[w]=nq;
        p=p->f;
      }
    }
  }
  last=np;
}

bool Find(char *s,int len) {
  int i;
  State *p=root;
  for(i=0;i<len;++i) {
    if(p->c[s[i]-'a']) {
      p=p->c[s[i]-'a'];
    } else {
      return false;
    }
  }
  return true;
}
```

最长回文子串

```
const int maxn=2000005;
int f[maxn];
std::string a, s;
int manacher() {
  int n=0, res=0, maxr=0, pos=0;
  for (int i=0; a[i]; i++) {
    s[++n] = '#', s[++n] = a[i];
    s[++n] = '#';
  }
  for (int i=1; i<=n; i++) {
    f[i] = (i<maxr? std::min(f[pos*2-i], maxr-i+1): 1);
    while (i-f[i]>0 && i+f[i]<=n && s[i-f[i]]==s[i+f[i]]) {
      f[i]++;
    }
```

```
    if (i+f[i]-1 > maxr) {
      maxr=i+f[i]-1;
      pos=i;
    }
    res = std::max(res,f[i]-1);
  }
  return res;
}
```

manacher

```
char tmp[maxn<<1];//转换后的字符串
int Len[maxn<<1];
//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
ll 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
  ll num=0;
  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;
    }
    int l=(i-1)/2-(Len[i]-1)/2;
    int r=(i-1)/2+(Len[i]-1)/2;
    if(Len[i]&1)
      r--;
    num+=((r-l+2)/2);
    ans=max(ans,Len[i]);
  }
  return num;  //返回回文子串的个数
  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
}
```

回文树

```
#include <iostream>
#include <vector>
#define rep(i, a, b) for (int i=a; i<=b; i++)
#define drep(i, a, b) for (int i=a; i>=b; i--)

const int inf = 1e9;
typedef long long ll;
const int maxn=300010;

char s[maxn];
int n;
struct Ptree {
  int last;
  struct Node {
    int cnt, lenn, fail, son[27];
    Node(int lenn, int fail):lenn(lenn), fail(fail), cnt(0){
      memset(son, 0, sizeof(son));
    };
  };
  std::vector<Node> st;
  inline int newnode(int lenn, int fail=0) {
    st.emplace_back(lenn, fail);
    return st.size()-1;
  }
  inline int getfail(int x, int n) {
    while (s[n-st[x].lenn-1] != s[n]) x=st[x].fail;
    return x;
  }
  inline void extend(int c, int i) {
    int cur=getfail(last, i);
    if (!st[cur].son[c]) {
      int nw=newnode(st[cur].lenn+2, st[getfail(st[cur].fail, i)].son[c]);
      st[cur].son[c]=nw;
    }
    st[ last=st[cur].son[c] ].cnt++;
  }
  void init() {
    scanf("%s", s+1);
    n=strlen(s+1);
    s[0]=0;
    newnode(0, 1), newnode(-1);
    last=0;
    rep(i, 1, n) extend(s[i]-'a', i);
```

```
  }
  ll count() {
    drep(i, st.size()-1, 0) st[st[i].fail].cnt += st[i].cnt;
    ll ans = 0;
    rep(i, 2, st.size()-1) ans = std::max(ans, 1LL*st[i].lenn*st[i].cnt);
    return ans;
  }
};
```

回文树

```
/*
 * Author: Simon
 * 复杂度: O(n · log(n))
 * 1. 求串 S 前缀 0~i 内本质不同回文串的个数 (两个串长度不同或者长度相同且至少有一个字符不同便是本质不同)
 * 2. 求串 S 内每一个本质不同回文串出现的次数
 * 3. 求串 S 内回文串的个数 (其实就是 1 和 2 结合起来)
 * 4. 求以下标 i 结尾的回文串的个数
 */
struct PAM{
/*Int */int tree[maxn][30]  /* 和字典树类似，指向的串为当前串两端加上同一个字符构成 */,
    fail[maxn]      /* 失配后跳转到 fail 指针指向的节点，其为最长回文后缀 */,
    cnt[maxn]       /* 表示节点 i 表示的回文串的个数 (建树时求出的不是完全的，最后 count() 函数跑一遍以后才是正确的) */,
    num[maxn]       /* 表示以节点 i 表示的最长回文串的最右端点为回文串结尾的回文串个数 */,
    len[maxn]       /*len[i] 表示节点 i 表示的回文串的长度 (一个节点表示一个回文串) */,
    s[maxn]         /* 存放添加的字符 */, lst/* 指向新添加一个字母后所形成的最长回文串表示的节点。 */,
    n               /* 表示添加的字符个数。 */,p/* 表示添加的节点个数。 */;
  int newNode(int l){  /* 新增一个节点，其最长回文串长度为 l */
    for(int i=0;i<26;i++) tree[p][i]=0;
    cnt[p]=num[p]=0,len[p]=l;
    return p++;
  }
  void init(){
    n=p=lst=0;
    newNode(0)/* 偶节点 */,newNode(-1)/* 奇节点 */;
    s[0]=-1,fail[0]=1/* 偶节点失配指针指向奇节点 */;
  }
  int getFail(int x){
    while(s[n-len[x]-1]!=s[n]) x=fail[x];
    return x;
  }
  void add(int c){
    c-='a'; s[++n]=c;
    int now=getFail(lst);/* 找到最长的回文子串，并且前后添加 c 字符后还是回文子串 */
    if(!tree[now][c]){/* 如果树中未存在此回文串 */
      int next=newNode(len[now]+2);/* 为此串建立新节点 */
      fail[next]=tree[getFail(fail[now])][c];/* 为新节点添加失配指针的指向 */
      tree[now][c]=next;/* 记录新串指向的节点 */
      num[next]=num[fail[next]]+1;/* 更新 num 数组 (num 数组含义在上面) */
    }
    lst=tree[now][c];/*c 字母所形成的最长回文子串所在的节点为 lst */
    cnt[lst]++;/* 统计此回文串出现的次数 */
  }
  void count(){
    for(int i=p-1;i>=0;i--) cnt[fail[i]]+=cnt[i];/* 节点 i 表示的回文串的个数要加上包含此回文串的串的个数, cnt[aa]+=cnt[baab] */
  }
}pam;
```

字符串哈希算法

```
// RS Hash Function
unsigned int RSHash(char *str) {
  unsigned int b = 378551;
  unsigned int a = 63689;
  unsigned int hash = 0;
  while (*str) {
    hash = hash * a + (*str++);
    a *= b;
  }
  return (hash & 0x7FFFFFFF);
}

// JS Hash Function
unsigned int JSHash(char *str) {
  unsigned int hash = 1315423911;
  while (*str) {
    hash ^= ((hash << 5) + (*str++) + (hash >> 2));
  }
  return (hash & 0x7FFFFFFF);
}

// P. J. Weinberger Hash Function
unsigned int PJWHash(char *str) {
  unsigned int BitsInUnignedInt = (unsigned int)(sizeof(unsigned int) * 8);
  unsigned int ThreeQuarters     = (unsigned int)((BitsInUnignedInt  * 3) / 4);
  unsigned int OneEighth         = (unsigned int)(BitsInUnignedInt / 8);
```

```
  unsigned int HighBits          = (unsigned int)(0xFFFFFFFF) <<
    (BitsInUnignedInt - OneEighth);
  unsigned int hash              = 0;
  unsigned int test              = 0;
  while (*str) {
    hash = (hash << OneEighth) + (*str++);
    if ((test = hash & HighBits) != 0) {
      hash = ((hash ^ (test >> ThreeQuarters)) & (~HighBits));
    }
  }
  return (hash & 0x7FFFFFFF);
}

// ELF Hash Function
unsigned int ELFHash(char *str) {
  unsigned int hash = 0;
  unsigned int x    = 0;
  while (*str) {
    hash = (hash << 4) + (*str++);
    if ((x = hash & 0xF0000000L) != 0) {
      hash ^= (x >> 24);
      hash &= ~x;
    }
  }
  return (hash & 0x7FFFFFFF);
}

// BKDR Hash Function
unsigned int BKDRHash(char *str) {
  unsigned int seed = 131; // 31 131 1313 13131 131313 etc..
  unsigned int hash = 0;
  while (*str) {
    hash = hash * seed + (*str++);
  }
  return (hash & 0x7FFFFFFF);
}

// SDBM Hash Function
unsigned int SDBMHash(char *str) {
  unsigned int hash = 0;
  while (*str) {
    hash = (*str++) + (hash << 6) + (hash << 16) - hash;
  }
  return (hash & 0x7FFFFFFF);
}

// DJB Hash Function
unsigned int DJBHash(char *str) {
  unsigned int hash = 5381;
  while (*str) {
    hash += (hash << 5) + (*str++);
  }
  return (hash & 0x7FFFFFFF);
}

// AP Hash Function
unsigned int APHash(char *str) {
  unsigned int hash = 0;
  int i;
  for (i=0; *str; i++) {
    if ((i & 1) == 0) {
      hash ^= ((hash << 7) ^ (*str++) ^ (hash >> 3));
    } else {
      hash ^= (~((hash << 11) ^ (*str++) ^ (hash >> 5)));
    }
  }
  return (hash & 0x7FFFFFFF);
}
```

字符串哈希表

```
typedef unsigned long long ull;
const ull base = 163;
char s[maxn];
ull hash[maxn];

void init() {
    p[0] = 1;
    hash[0] = 0;
    int n = strlen(s + 1);
   for(int i = 1; i <=100000; i ++)p[i] =p[i-1] * base;
   for(int i = 1; i <= n; i ++)hash[i] = hash[i - 1] * base + (s[i] - 'a');
}

ull get(int l, int r, ull g[]) {
    return g[r] - g[l - 1] * p[r - l + 1];
}

struct HASHMAP {
  int size;
  int head[maxh], next[maxn], f[maxn];  // maxh 为 hash 链表最大长度
  ull state[maxn];
  void init() {
    size = 0;
    memset(head, -1, sizeof(head));
  }
  int insert(ull val, int id) {
    int h = val % maxh;
    for (int i = head[h]; i != -1; i = next[i])
      if (val == state[i]) return f[i];
```

```
    f[size] = id;
    state[size] = val;
    next[size] = head[h];
    head[h] = size;
    return f[size++];
  }
};
```

几何

平面几何公式

```
三角形:
  1. 半周长 P=(a+b+c)/2
  2. 面积 S=aHa/2=absin(C)/2=sqrt(P(P-a)(P-b)(P-c))
  3. 中线 Ma=sqrt(2(b^2+c^2)-a^2)/2=sqrt(b^2+c^2+2bccos(A))/2
  4. 角平分线 Ta=sqrt(bc((b+c)^2-a^2))/(b+c)=2bccos(A/2)/(b+c)
  5. 高线 Ha=bsin(C)=csin(B)=sqrt(b^2-((a^2+b^2-c^2)/(2a))^2)
  6. 内切圆半径 r=S/P=asin(B/2)sin(C/2)/sin((B+C)/2)
                 =4Rsin(A/2)sin(B/2)sin(C/2)=sqrt((P-a)(P-b)(P-c)/P)
                 =Ptan(A/2)tan(B/2)tan(C/2)
  7. 外接圆半径 R=abc/(4S)=a/(2sin(A))=b/(2sin(B))=c/(2sin(C))


  四边形:
  D1,D2 为对角线,M 对角线中点连线,A 为对角线夹角
  1. a^2+b^2+c^2+d^2=D1^2+D2^2+4M^2
  2. S=D1D2sin(A)/2
  (以下对圆的内接四边形)
  3. ac+bd=D1D2
  4. S=sqrt((P-a)(P-b)(P-c)(P-d)),P 为半周长


  正 n 边形:
  R 为外接圆半径,r 为内切圆半径
  1. 中心角 A=2PI/n
  2. 内角 C=(n-2)PI/n
  3. 边长 a=2sqrt(R^2-r^2)=2Rsin(A/2)=2rtan(A/2)
  4. 面积 S=nar/2=nr^2tan(A/2)=nR^2sin(A)/2=na^2/(4tan(A/2))


  圆:
  1. 弧长 l=rA
  2. 弦长 a=2sqrt(2hr-h^2)=2rsin(A/2)
  3. 弓形高 h=r-sqrt(r^2-a^2/4)=r(1-cos(A/2))=atan(A/4)/2
  4. 扇形面积 S1=rl/2=r^2A/2
  5. 弓形面积 S2=(rl-a(r-h))/2=r^2(A-sin(A))/2


  棱柱:
  1. 体积 V=Ah,A 为底面积,h 为高
  2. 侧面积 S=lp,l 为棱长,p 为直截面周长
  3. 全面积 T=S+2A


  棱锥:
  1. 体积 V=Ah/3,A 为底面积,h 为高
  (以下对正棱锥)
  2. 侧面积 S=lp/2,l 为斜高,p 为底面周长
  3. 全面积 T=S+A


  棱台:
  1. 体积 V=(A1+A2+sqrt(A1A2))h/3,A1.A2 为上下底面积,h 为高
  (以下为正棱台)
  2. 侧面积 S=(p1+p2)l/2,p1.p2 为上下底面周长,l 为斜高
  3. 全面积 T=S+A1+A2


  圆柱:
  1. 侧面积 S=2PIrh
  2. 全面积 T=2PIr(h+r)
  3. 体积 V=PIr^2h


  圆锥:
  1. 母线 l=sqrt(h^2+r^2)
  2. 侧面积 S=PIrl
  3. 全面积 T=PIr(l+r)
  4. 体积 V=PIr^2h/3


  圆台:
  1. 母线 l=sqrt(h^2+(r1-r2)^2)
  2. 侧面积 S=PI(r1+r2)l
  3. 全面积 T=PIr1(l+r1)+PIr2(l+r2)
  4. 体积 V=PI(r1^2+r2^2+r1r2)h/3


  球:
  1. 全面积 T=4PIr^2
  2. 体积 V=4PIr^3/3


  球台:
  1. 侧面积 S=2PIrh
  2. 全面积 T=PI(2rh+r1^2+r2^2)
  3. 体积 V=PIh(3(r1^2+r2^2)+h^2)/6
```

```
球扇形:
1. 全面积 T=PIr(2h+r0),h 为球冠高,r0 为球冠底面半径
2. 体积 V=2PIr^2h/3
```

```
    res[++top] = pnt[i];
  }
  return top; //  返回凸包中点的个数
}
```

求凸包

```
/*
 *  Graham 求凸包 O(N * logN)
 *  CALL: nr = graham(pnt, int n, res); res[] 为凸包点集;
 */
struct point {
  double x, y;
};

bool mult(point sp, point ep, point op) {
  return (sp.x - op.x) * (ep.y - op.y) >= (ep.x - op.x) * (sp.y - op.y);
}

inline bool operator < (const point &l, const point &r) {
    return l.y < r.y || (l.y == r.y && l.x < r.x);
}

int graham(point pnt[], int n, point res[]) {
  int i, len, top = 1;
  sort(pnt, pnt + n);
  if (n == 0) {
    return 0;
  }
  res[0] = pnt[0];
  if (n == 1) {
    return 1;
  }
  res[1] = pnt[1];
  if (n == 2) {
    return 2;
  }
  res[2] = pnt[2];
  for (i = 2; i < n; i++) {
    while (top && mult(pnt[i], res[top], res[top - 1])) {
      top--;
    }
    res[++top] = pnt[i];
  }
  len = top;
  res[++top] = pnt[n - 2];
  for (i = n - 3; i >= 0; i--) {
    while (top != len && mult(pnt[i], res[top], res[top - 1])) {
      top--;
    }
```

四点共面

```
struct  point {
  double x, y, z;
  point  operator - (point &o) {
    point  ans;
    ans.x = this->x - o.x;
    ans.y = this->y - o.y;
    ans.z = this->z - o.z;
    return ans;
  }
};

double  dot_product(const point &a, const point &b) {
  return a.x * b.x + a.y * b.y + a.z * b.z;
}

point cross_product(const point &a, const point &b) {
  point  ans;
  ans.x = a.y * b.z - a.z * b.y;
  ans.y = a.z * b.x - a.x * b.z;
  ans.z = a.x * b.y - a.y * b.x;
  return ans;
}

int main() {
  point p[4];
  int T;
  for (scanf("%d", &T); T--;) {
    for (int i = 0; i < 4; ++i) {
      scanf("%lf%lf%lf", &p[i].x, &p[i].y, &p[i].z);
    }
    puts(dot_product(p[3] - p[0], cross_product(p[2] - p[0], p[1] - p[0])) ==
      0.0 ? "Yes\n" : "No\n");
  }
  return 0;
}
```

多边形重心

```
/*
 *  求多边形重心
 *  INIT: pnt[] 已按顺时针 (或逆时针) 排好序; | CALL: res = bcenter(pnt, n);
 */
```

```cpp
struct point {
  double x, y;
};

point bcenter(point pnt[], int n) {
  point p, s;
  double tp, area = 0, tpx = 0, tpy = 0;
  p.x = pnt[0].x;
  p.y = pnt[0].y;
  for (int i = 1; i <= n; ++i) {   //  point:0 ~ n - 1
    s.x = pnt[(i == n) ? 0 : i].x;
    s.y = pnt[(i == n) ? 0 : i].y;
    tp = (p.x * s.y - s.x * p.y);
    area += tp / 2;
    tpx += (p.x + s.x) * tp;
    tpy += (p.y + s.y) * tp;
    p.x = s.x;
    p.y = s.y;
  }
  s.x = tpx / (6 * area);
  s.y = tpy / (6 * area);
  return s;
}
```

旋转卡壳

```cpp
struct Point {
  int x, y;
  Point(int _x = 0, int _y = 0) {
    x = _x;
    y = _y;
  }
  Point operator - (const Point &b)const {
    return Point(x - b.x, y - b.y);
  }
  int operator ^(const Point &b)const {
    return x * b.y - y * b.x;
  }
  int operator *(const Point &b)const {
    return x * b.x + y * b.y;
  }
  void input() {
    scanf("%d%d", &x, &y);
    return ;
  }
};

//  距离的平方
int dist2(Point a, Point b) {
  return (a - b) * (a - b);
}

//  二维凸包,int
const int MAXN = 50010;
Point list[MAXN];
int Stack[MAXN], top;
bool _cmp(Point p1, Point p2) {
  int tmp = (p1 - list[0]) ^ (p2 - list[0]);
  if (tmp > 0) {
    return true;
  }
  else if (tmp == 0 && dist2(p1, list[0]) <= dist2(p2, list[0])) {
    return true;
  } else {
    return false;
  }
}

void Graham(int n) {
  Point p0;
  int k = 0;
  p0 = list[0];
  for (int i = 1; i < n; i++) {
    if (p0.y > list[i].y || (p0.y == list[i].y && p0.x > list[i].x)) {
      p0 = list[i];
      k = i;
    }
  }
  swap(list[k], list[0]);
  sort(list + 1, list + n, _cmp);
  if (n == 1) {
    top = 1;
    Stack[0] = 0;
    return ;
  }
  if (n == 2) {
    top = 2;
    Stack[0] = 0;
    Stack[1] = 1;
    return ;
  }
  Stack[0] = 0;
  Stack[1] = 1;
  top = 2;
  for (int i = 2; i < n; i++) {
    while (top > 1 && ((list[Stack[top - 1]] - list[Stack[top - 2]]) ^
      (list[i] - list[Stack[top - 2]])) <= 0) {
      top--;
    }
    Stack[top++] = i;
```

```
  }
  return ;
}

//  旋转卡壳，求两点间距离平方的最大值
int rotating_calipers(Point p[],int n) {
  int ans = 0;
  Point v;
  int cur = 1;
  for (int i = 0; i < n; i++) {
    v = p[i] - p[(i + 1) % n];
    while ((v ^ (p[(cur + 1) % n] - p[cur])) < 0) {
      cur = (cur + 1) % n;
    }
    ans = max(ans, max(dist2(p[i], p[cur]), dist2(p[(i + 1) % n], p[(cur + 1)
     ↪ % n])));
  }
  return ans;
}

Point p[MAXN];

int main() {
  int n;
  while (scanf("%d", &n) == 1) {
    for (int i = 0; i < n; i++) {
      list[i].input();
    }
    Graham(n);
    for (int i = 0; i < top; i++) {
      p[i] = list[Stack[i]];
    }
    printf("%d\n", rotating_calipers(p, top));
  }
  return 0;
}
```

模拟退火

```
//模拟退火
//平面上找一个点 使得 sigma(1..N)dist(a, i)*Wi 最小

#include <cstdlib>
#include <cstdio>
#include <cstring>
#include <cmath>

#define INF (1e17)
#define EPS (1e-3)
#define PI  (acos(-1.0))
#define FIRE(x) (x *= 0.99)
using namespace std;
const int MAXN = 10000 + 10;
int N;
double total = INF;
struct Point {
  double x, y, w;
  Point (double _x, double _y) : x(_x), y(_y) {}
  Point (void) {}
  void Read(void) {
    scanf("%lf%lf%lf", &x, &y, &w);
  }
  void operator += (Point t) {
    x += t.x; y += t.y;
  }
  void operator /= (int N) {
    x /= N, y /= N;
  }
};
Point now, ans, point[MAXN];
inline double Dist(Point a, Point b) {
  return sqrt((a.x - b.x) * (a.x - b.x) +
              (a.y - b.y) * (a.y - b.y));
}
inline double Statistic(Point p) {
  double res = 0.0;
  for (int i = 0; i < N; i++) res += Dist(p, point[i]) * point[i].w;
  if (res < total) total = res, ans = p;
  return res;
}
inline double Rand(void) {
  return (rand() % 1000 + 1) / 1000.0;
}
int main(void) {
  srand(10086);
  scanf("%d", &N);
  register int i;
  for (i = 0; i < N; i++) point[i].Read(), now += point[i];
  now /= N;
  double T = 100000.0, alpha, sub;
  while (T > EPS) {
    alpha = 2.0 * PI * Rand();
    Point tmp(now.x + T * cos(alpha), now.y + T * sin(alpha));
    sub = Statistic(now) - Statistic(tmp);
    if(sub >= 0 || exp(sub / T) >= Rand()) now = tmp;
    FIRE(T);
  }
  T = 0.001;
  for (i = 1; i <= 1000; ++i) {
```

```
    alpha = 2.0 * PI * Rand();
    Point tmp(ans.x + T * cos(alpha) * Rand(), ans.y + T * sin(alpha) *
      ↪ Rand());
    Statistic(tmp);
  }
  printf("%.3lf %.3lf\n", ans.x, ans.y);
  return 0;
}
```

半平面交

```
#include <cstdio>
#include <algorithm>
#include <queue>
#include <cmath>
using namespace std;
const double eps = 1e-8;
struct Point{
  double x,y;
  Point(double xx=0.0,double yy=0.0) :x(xx) ,y(yy) {}
  Point operator - (const Point &b) const {
    return Point(x-b.x,y-b.y) ;
  }
  Point operator +(const Point &b) const {
    return Point(x+b.x,y+b.y) ;
  }
  Point operator /(const double &b) const {
    return Point(x/b,y/b) ;
  }
  Point operator *(const double &b) const {
    return Point(x*b,y*b) ;
  }
  double operator ^(const Point &b) const {
    return x*b.y-y*b.x;
  }
};
typedef Point myvec;
double cross(myvec a,myvec b) {
  return a^b;
}
struct Line{
  Point p;
  myvec v;
  double ang;
  Line() {}
  Line( Point pp,myvec vv) :p(pp) ,v(vv) {}
  bool operator < (const  Line &l) const {
    return ang < l.ang;
  }

};
//点 p 在有向直线 L 的左边 (线上不算)
bool on_left( Line l,Point p) {
  return cross(l.v,p-l.p) >0;
}
//直线交点 假设交点唯一存在
Point get_inter_section(Line a,Line b) {
  myvec u = a.p - b. p;
  double t = cross(b.v,u) /cross(a.v,b.v) ;
  return a.p+a.v*t;

}
int half_plane_inter_section(Line *L,int n,Point *poly) {
  sort(L,L+n) ;//级角排序
  int fir,lst;//双向队列的第一个元素和最后一个元素的下标
  Point *p = new Point[n];//p[i] 为 q[i] 和 q[i+1] 的交点
  Line *q = new Line[n];//双端队列
  q[ fir = lst = 0 ] = L[0];//双端队列初始化为只有一个半平面的 L[0]
  for( int i =1; i <n ; ++i)
  {
    while( fir < lst && !on_left(L[i],p[lst-1])  )
      lst--;
    while( fir<lst && !on_left(L[i],p[fir])  )
      fir++;
    q[++lst] = L[i];
    if( fabs( cross(q[lst].v,q[lst-1].v)  )  < eps ) { //两向量平行且同向 取
      ↪ 内侧一个
      lst--;
      if( on_left(q[lst],L[i].p)  )
        q[lst] = L[i];
    }
    if( fir < lst )
      p[lst-1] = get_inter_section(q[lst-1],q[lst]) ;
  }
  while( fir< lst && !on_left(q[fir],p[lst-1]) )
    lst--;//删除无用的平面
  if(lst - fir <=1 )
    return 0;//空集
  p[lst] = get_inter_section(q[lst],q[fir]) ;//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 = fir;i<=lst;++i)
    poly[m++] = p[i];
   return m;
}
```

计算几何

```cpp
//上交计算几何算法
/******************************************
* COMPUTATIONAL GEOMETRY ROUTINES
* WRITTEN BY : LIU Yu (C) 2003
******************************************/
//    叉乘
//    两个点的距离
//    点到直线距离
//    返回直线 Ax + By + C =0  的系数
//    线段
//    圆
//    两个圆的公共面积
//    矩形
//    根据下标返回多边形的边
//    两个矩形的公共面积
//    多边形 ，逆时针或顺时针给出 x,y
//    多边形顶点
//    多边形的边
//    多边形的周长
//    判断点是否在线段上
//    判断两条线断是否相交，端点重合算相交
//    判断两条线断是否平行
//    判断两条直线断是否相交
//    直线相交的交点
//    判断是否简单多边形
//    求多边形面积
//    判断是否在多边形上
//    判断是否在多边形内部
//    点阵的凸包，返回一个多边形
//   最近点对的距离
#include <cmath>
#include <cstdio>
#include <memory>
#include <algorithm>
#include <iostream>
using namespace std;
typedef double TYPE;
#define Abs(x) (((x)>0)?(x):(-(x)))
#define Sgn(x) (((x)<0)?(-1):(1))
#define Max(a,b) (((a)>(b))?(a):(b))
#define Min(a,b) (((a)<(b))?(a):(b))
#define Epsilon 1e-10
#define Infinity 1e+10
#define Pi 3.14159265358979323846
TYPE Deg2Rad(TYPE deg) {
  return (deg * Pi / 180.0);
}
TYPE Rad2Deg(TYPE rad) {
  return (rad * 180.0 / Pi);
}
TYPE Sin(TYPE deg) {
  return sin(Deg2Rad(deg));
}
TYPE Cos(TYPE deg) {
  return cos(Deg2Rad(deg));
}
TYPE ArcSin(TYPE val) {
  return Rad2Deg(asin(val));
}
TYPE ArcCos(TYPE val) {
  return Rad2Deg(acos(val));
}
TYPE Sqrt(TYPE val) {
  return sqrt(val);
}
struct POINT {
  TYPE x;
  TYPE y;
  TYPE z;
  POINT() : x(0), y(0), z(0) {};
  POINT(TYPE _x_, TYPE _y_, TYPE _z_ = 0) : x(_x_), y(_y_), z(_z_) {};
};
// cross product of (o->a) and (o->b)// 叉乘
TYPE Cross(const POINT & a, const POINT & b, const POINT & o) {
  return (a.x - o.x) * (b.y - o.y) - (b.x - o.x) * (a.y - o.y);
}
// planar points' distance//  两个点的距离
TYPE Distance(const POINT & a, const POINT & b) {
  return Sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y) + (a.z -
    ↪ b.z) * (a.z
        - b.z));
}
struct LINE {
  POINT a;
  POINT b;
  LINE() {};
  LINE(POINT _a_, POINT _b_) : a(_a_), b(_b_) {}};
//点到直线距离
double PointToLine(POINT p0 ,POINT p1 ,POINT p2 ,POINT &cp) {
  double d = Distance(p1 ,p2);
  double s = Cross(p1 ,p2 ,p0) / d;
  cp.x = p0.x + s*( p2.y-p1.y) / d;
  cp.y = p0.y - s*( p2.x-p1.x) / d;
  return Abs(s);
}
// 返回直线 Ax + By + C =0  的系数
void Coefficient(const LINE & L, TYPE & A, TYPE & B, TYPE & C) {
```

```
  A = L.b.y - L.a.y;
  B = L.a.x - L.b.x;
  C = L.b.x * L.a.y - L.a.x * L.b.y;
}
void Coefficient(const POINT & p,const TYPE a,TYPE & A,TYPE & B,TYPE & C) {
  A = Cos(a);    // 线段
  B = Sin(a);
  C = - (p.y * B + p.x * A);
}
struct SEG {
  POINT a;
  POINT b;
  SEG() {};
  SEG(POINT  _a_, POINT _b_):a(_a_),b(_b_) {};
};
// 圆
struct CIRCLE {
  TYPE x;
  TYPE y;
  TYPE r;
  CIRCLE() {}
  CIRCLE(TYPE _x_, TYPE _y_, TYPE _r_) : x(_x_), y(_y_), r(_r_) {}};
POINT Center(const CIRCLE & circle) {
  return POINT(circle.x, circle.y);
}
TYPE Area(const CIRCLE & circle) {
  return Pi * circle.r * circle.r;    //两个圆的公共面积
}
TYPE CommonArea(const  CIRCLE & A, const CIRCLE & B) {
  TYPE area = 0.0;
  const CIRCLE & M = (A.r > B.r) ? A : B;
  const CIRCLE & N = (A.r > B.r) ? B : A;
  TYPE D = Distance(Center(M), Center(N));
  if ((D < M.r + N.r) && (D > M.r - N.r)) {
    TYPE cosM = (M.r * M.r + D * D - N.r * N.r) / (2.0 * M.r * D);
    TYPE cosN = (N.r * N.r + D * D - M.r * M.r) / (2.0 * N.r * D);
    TYPE alpha = 2.0 * ArcCos(cosM);
    TYPE beta  = 2.0 * ArcCos(cosN);
    TYPE TM = 0.5*M.r*M.r*Sin(alpha);
    TYPE TN=0.5 * N.r * N.r * Sin(beta);
    TYPE FM = (alpha / 360.0) * Area(M);
    TYPE FN = (beta / 360.0) * Area(N);
    area = FM + FN - TM - TN;
  } else if (D <=  M.r - N.r) {
    area = Area(N);
  }
  return area;
}
//    矩形
//    矩形的线段
//        2
//       --------------- b
//       |               |
//    3 |               |  1
//    a ---------------
//         0
struct RECT {
  POINT a; // 左下点  POINT b;  // 右上点
  RECT() {};
  RECT(const POINT & _a_, const POINT & _b_) {
    a = _a_;
    b = _b_;
  }
};
//根据下标返回多边形的边
SEG Edge(const RECT & rect, int idx) {
  SEG edge;
  while (idx < 0) idx +=  4;
  switch (idx % 4) {
  case 0:
    edge.a = rect.a;
    edge.b = POINT(rect.b.x, rect.a.y);
    break;
  case 1:
    edge.a = POINT(rect.b.x, rect.a.y);
    edge.b = rect.b;
    break;
  case 2:
    edge.a = rect.b;
    edge.b = POINT(rect.a.x, rect.b.y);
    break;
  case 3:
    edge.a = POINT(rect.a.x, rect.b.y);
    edge.b = rect.a;
    break;
  default:
    break;
  }
  return edge;
}
TYPE Area(const RECT & rect) {
  return (rect.b.x - rect.a.x) * (rect.b.y - rect.a.y);
}
//  两个矩形的公共面积
TYPE CommonArea(const  RECT & A, const RECT & B) {
  TYPE area = 0.0;
  POINT LL(Max(A.a.x, B.a.x), Max(A.a.y, B.a.y));
  POINT UR(Min(A.b.x,  B.b.x), Min(A.b.y, B.b.y));
  if ((LL.x <=  UR.x) && (LL.y <= UR.y)) {
    area = Area(RECT(LL, UR));
```

```
  }
  return area;
}// 多边形  , 逆时针或顺时针给出 x,y
struct POLY {
  int n; //n 个点  TYPE * x;  //x,y 为点的指针，首尾必须重合  TYPE * y;
  POLY() : n(0), x(NULL), y(NULL) {};
  POLY(int _n_, const TYPE * _x_, const TYPE * _y_) {
    n = _n_;
    x = new TYPE[n + 1];
    memcpy(x,  _x_, n*sizeof(TYPE));
    x[n] = _x_[0];
    y = new TYPE[n + 1];
    memcpy(y, _y_, n*sizeof(TYPE));
    y[n] = _y_[0];
  }
};//多边形顶点
POINT Vertex(const POLY & poly, int idx) {
  idx %= poly.n;    //多边形的边
  return POINT(poly.x[idx],  poly.y[idx]);
}
SEG Edge(const POLY & poly, int idx) {
  idx %= poly.n;
  return SEG(POINT(poly.x[idx],  poly.y[idx]),
         POINT(poly.x[idx  + 1], poly.y[idx + 1]));
} //多边形的周长
TYPE Perimeter(const POLY & poly) {
  TYPE p = 0.0;
  for (int i = 0; i < poly.n; i++)
    p = p + Distance(Vertex(poly, i), Vertex(poly, i + 1));
  return p;
}
bool IsEqual(TYPE a, TYPE b) {
  return (Abs(a - b) < Epsilon);
}
bool IsEqual(const POINT & a, const POINT & b) {
  return (IsEqual(a.x, b.x) && IsEqual(a.y, b.y));
}
bool IsEqual(const LINE & A, const LINE & B) {
  TYPE A1, B1, C1;
  TYPE A2, B2, C2;
  Coefficient(A, A1, B1, C1);
  Coefficient(B, A2, B2, C2);
  return IsEqual(A1 * B2, A2 * B1) &&
       IsEqual(A1 * C2, A2 * C1) &&  IsEqual(B1 * C2, B2 * C1);
}
// 判断点是否在线段上
bool IsOnSeg(const SEG & seg, const POINT & p) {
  return (IsEqual(p, seg.a) || IsEqual(p, seg.b)) ||
       (((p.x - seg.a.x) * (p.x - seg.b.x) < 0 || (p.y - seg.a.y) * (p.y -
         ↪ seg.b.y) < 0) &&
       (IsEqual(Cross(seg.b, p, seg.a), 0)));
}
//判断两条线断是否相交，端点重合算相交
bool IsIntersect(const SEG & u, const SEG & v) {
  return (Cross(v.a, u.b, u.a) * Cross(u.b, v.b, u.a) >=  0) &&
       (Cross(u.a, v.b, v.a) * Cross(v.b, u.b, v.a) >= 0) &&(Max(u.a.x,
         ↪ u.b.x)>=Min(v.
           a.x, v.b.x))&&(Max(v.a.x, v.b.x)>= Min(u.a.x,u.b.x))&&(Max(u.a.y,
             ↪ u.b.y)>=Min(
               v.a.y, v.b.y))&&(Max(v.a.y, v.b.y)>=Min(u.a.y,  u.b.y));
}
//判断两条线断是否平行
bool IsParallel(const LINE & A, const LINE & B) {
  TYPE A1, B1, C1;
  TYPE A2, B2, C2;
  Coefficient(A, A1, B1, C1);
  Coefficient(B, A2, B2, C2);
  return (A1*B2==  A2*B1) &&((A1 * C2 != A2 * C1) || (B1 * C2 != B2 * C1));
}
//判断两条直线断是否相交
bool IsIntersect(const LINE & A, const LINE & B) {
  return !IsParallel(A, B);    //直线相交的交点
}

POINT Intersection(const LINE & A, const LINE & B) {
  TYPE A1, B1, C1;
  TYPE A2, B2, C2;
  Coefficient(A, A1, B1, C1);
  Coefficient(B, A2, B2, C2);
  POINT I(0, 0);
  I.x = - (B2 * C1 - B1 * C2) / (A1 * B2 - A2 * B1);
  I.y =   (A2 * C1 - A1 * C2) / (A1 * B2 - A2 * B1);
  return I;
}

bool IsInCircle(const CIRCLE & circle, const RECT & rect) {
  return (circle.x - circle.r >=  rect.a.x) &&
       (circle.x + circle.r <= rect.b.x) &&(circle.y - circle.r >=  rect.a.y)
         ↪ &&
       (circle.y + circle.r <= rect.b.y);
}
//判断是否简单多边形
bool IsSimple(const POLY & poly) {
  if (poly.n < 3) return false;
  SEG L1, L2;
  for (int i = 0; i < poly.n - 1; i++) {
    L1 = Edge(poly, i);
    for (int j = i + 1; j < poly.n; j++) {
      L2 = Edge(poly, j);
      if (j == i+1) {
```

```
        if  (IsOnSeg(L1, L2.b)||IsOnSeg(L2, L1.a))  return false;
      } else if (j == poly.n - i - 1) {
        if (IsOnSeg(L1, L2.a) || IsOnSeg(L2, L1.b))  return false;
      } else {
        if (IsIntersect(L1, L2))  return false;    // for i
      }
    } // for j
  }
  return true;
}
//求多边形面积
TYPE Area(const POLY & poly) {
  if (poly.n < 3) return TYPE(0);
  double s = poly.y[0] * (poly.x[poly.n - 1] - poly.x[1]);
  for (int i = 1; i < poly.n; i++) {
    s += poly.y[i] * (poly.x[i - 1] - poly.x[(i + 1) % poly.n]);
  }
  return s/2;
}
//判断是否在多边形上
bool IsOnPoly(const POLY & poly, const POINT & p) {
  for (int i = 0; i < poly.n; i++) {
    if (IsOnSeg(Edge(poly,  i), p))   return true;
  }
  return false;
}
//判断是否在多边形内部
bool IsInPoly(const POLY & poly, const POINT & p) {
  SEG L(p, POINT(Infinity, p.y));
  int count = 0;
  for (int i = 0; i < poly.n; i++) {
    SEG S = Edge(poly, i);
    if (IsOnSeg(S, p)) {
      return false; //如果想让 在 poly 上则返回 true, 则改为 true
    }
    if (!IsEqual(S.a.y, S.b.y)) {
      POINT & q = (S.a.y > S.b.y)?(S.a):(S.b);
      if (IsOnSeg(L, q)) {
        ++count;
      }
      else if(!IsOnSeg(L,S.a)&&!IsOnSeg(L,S.b)&&IsIntersect(S,L)) {
        ++count;
      }
    }
  }
  return (count % 2 != 0);
}
// 点阵的凸包，返回一个多边形
POLY ConvexHull(const POINT * set, int n) {
  POINT * points = new POINT[n];
  memcpy(points, set, n * sizeof(POINT));
  TYPE * X = new TYPE[n];
  TYPE * Y = new TYPE[n];
  int i, j, k = 0, top = 2;
  for(i = 1; i < n; i++) {
    if((points[i].y<points[k].y)||((points[i].y==points[k].y)&&
                     (points[i].x<points[k].x))) {
      k  = i;
    }
  }
  std::swap(points[0], points[k]);
  for (i = 1; i < n - 1; i++) {
    k = i;
    for (j = i + 1; j < n; j++) {
      if ((Cross(points[j], points[k], points[0]) >0)||((Cross(points[j],
        ↪ points[k],
          points[0]) == 0) && (Distance(points[0],
            ↪ points[j])<Distance(points[0], points[k]
                                             )))) {
        k = j;
      }
    }
    std::swap(points[i], points[k]);
  }
  X[0] = points[0].x;
  Y[0] = points[0].y;
  X[1] = points[1].x;
  Y[1] = points[1].y;
  X[2] = points[2].x;
  Y[2] = points[2].y;
  for (i = 3; i < n; i++) {
    while(Cross(points[i],POINT(X[top],Y[top]),POINT(X[top
          -1],Y[top-1]))>=0) {
      top--;
    }
    ++top;
    X[top] = points[i].x;
    Y[top] = points[i].y;
  }
  delete [] points;
  POLY poly(++top, X, Y);
  delete [] X;
  delete [] Y;
  return poly;
}
//最近点对的距离, Written By PrincessSnow
#define MAXN 100000
POINT pt[MAXN];
bool cmp(POINT n1, POINT n2) {
  return (n1.x<n2.x  || n1.x==n2.x  && n1.y<n2.y);
```

```
}
double Get(double dis, int mid, int start, int end) {
  int s=mid,  e=mid,  i, j;
  double t;
  while(s > start && pt[mid].x - pt[s].x <= dis)     s--;
  while(e < end && pt[e].x - pt[mid].x <=  dis)     e++;
  for(i=s; i <= e; i++)
    for(j=i+1;  j <= e && j <=  i+7; j++) {
      t = Distance(pt[i], pt[j]);
      if(t < dis)     dis=t;
    }
  return dis;
}
double ClosestPairDistance(int start, int end) {
  int m = end-start+1, mid, i;
  double t1, t2, dis=-1, t;
  if(m <= 3) {
    for(i=start; i < end; i++) {
      t = Distance(pt[i] , pt[i+1]);
      if(t < dis || dis ==  -1)     dis = t;
    }
    t = Distance(pt[start] , pt[end]);
    if(t < dis) dis=t;
    return dis;
  }
  if(m%2  == 0)     mid = start + m/2 - 1;
  else               mid = start + m/2;
  if(m%2  == 0) {
    t1 = ClosestPairDistance(start, mid);
    t2 = ClosestPairDistance(mid+1, end);
  } else {
    t1 =ClosestPairDistance(start,mid);
    t2=ClosestPairDistance(mid+1,end);
  }
  if(t1 < t2)     dis = t1;
  else            dis = t2;
  dis = Get(dis, mid, start, end);
  return dis;
}


//1.  球面上两点最短距离
// 计算圆心角 lat 表示纬度, -90 <= w <= 90, lng 表示经度
// 返回两点所在大圆劣弧对应圆心角,  0 <= angle <=  pi
double angle(double lng1, double lat1, double lng2, double lat2) {
  double dlng = fabs(lng1 - lng2) * pi / 180;
  while(dlng >= pi+pi)      dlng -= pi+pi;
  if(dlng > pi)     dlng = pi + pi - dlng;
  lat1 *= pi / 180,     lat2 *= pi / 180;
  return acos( cos(lat1)*cos(lat2)*cos(dlng) + sin(lat1)*sin(lat2) );
}
// 计算距离, r 为球半径
double line_dist(double  r, double lng1, double lat1, double lng2, double
  ↪ lat2) {
  double dlng = fabs(lng1 - lng2) * pi / 180;
  while(dlng  >= pi+pi)      dlng -= pi+pi;
  if(dlng > pi)     dlng = pi + pi - dlng;
  lat1 *= pi / 180,     lat2 *= pi / 180;
  return r*sqrt(2-2*( cos(lat1)*cos(lat2)*cos(dlng)+ sin(lat1)*sin(lat2)) );
}
// 计算球面距离, r 为球半径
double sphere_dist(double r, double lng1,double lat1,double lng2, double
  ↪ lat2) {
  return r * angle(lng1, lat1, lng2, lat2);
}


//2.  三点求圆心坐标
double GetRadiusBy3Points(double  x1, double y1,
              double x2, double y2,  double x3, double y3, double &x, double
                ↪ &y) {
  // 由 ( x - x1 )^2 + ( y - y1 )^2 = ( x - x2 )^2 + ( y - y2 )^2 得
  // 2*( x2 - x1 )*x + 2*( y2 - y1 )*y = x2^2 - x1^2 + y2^2 - y1^2
  // 同理得
  // 2*( x3 - x2 )*x + 2*( y3 - y2 )*y = x3^2 - x2^2 + y3^2 - y2^2
  // 由行列式解方程得 x , y
  double a11, a12, a21, a22, b1, b2;
  double d, d1, d2 ;
  a11 = 2 * ( x3 - x2 );
  a12 = 2 * ( y3 - y2 );
  a21 = 2 * ( x2 - x1 );
  a22 = 2 * ( y2 - y1 );
  b1 = x3*x3 - x2*x2 + y3*y3 - y2*y2;
  b2 = x2*x2 - x1*x1 + y2*y2 - y1*y1;
  d = a11*a22 - a12*a21;
  d1 = b1*a22 - a12*b2;
  d2 = a11*b2 - b1*a21;
  // x , y 是圆心坐标
  x = d1 / d;
  y = d2 / d;
  return  (x1 - x)*(x1 - x) + (y1 - y)*(y1 - y);
}
//
//
//3.  三角形几个重要的点
//设三角形的三条边为 a,  b, c, 且不妨假设 a <=  b <=  c
//三角形的面积可以根据海伦公式算得, 如下:
//s = sqrt(p * (p - a) * (p - b) * (p - c)), p = (a +  b + c) / 2
//1. 费马点 (该点到三角形三个顶点的距离之和最小)
//有个有趣的结论: 若三角形的三个内角均小于 120 度,
```

```
//那么该点连接三个顶点形成的三个角均为 120 度；若三角形存在一个内角
//大于 120 度，则该顶点就是费马点)
//计算公式如下:
//若有一个内角大于 120 度 (这里假设为角 C)，则距离为 a  +  b
//若三个内角均小于 120 度，则距离为
//sqrt((a * a +  b * b + c * c + 4 * sqrt(3.0) * s) / 2)，其中
//2. 内心----角平分线的交点
//令 x =  (a + b - c) / 2, y =  (a - b +  c) / 2, z =  (-a +  b + c) / 2, h
    ↪=  s / p
// 计算公式为 sqrt(x * x + h * h) + sqrt(y * y +  h * h) +  sqrt(z * z +  h
    ↪* h)
//3. 重心----中线的交点
//ACM 算法模板集
//  - 46 -
//计算公式如下:
//2.0 / 3 * (sqrt((2 * (a * a +  b * b) - c * c) / 4)
//             +  sqrt((2 * (a * a +  c * c) - b * b) / 4)
//             +  sqrt((2 * (b * b + c * c) - a * a) / 4))
//4. 垂心----垂线的交点
//计算公式如下:
//3 * (c / 2 / sqrt(1 - cosC * cosC))
```

类

点类

```
struct point {
  double x, y;
  point() { };
  point(double x, double y) :x(x), y(y) { }
  point operator - (const point &b) const {
    return point(x - b.x, y - b.y);
  }
  point operator + (const point &b) const {
    return point(x + b.x, y + b.y);
  }
  point operator * (const double k) const {
    return point(k * x, k * y);
  }
  point operator / (const double k) const {
    return point(x / k, y / k);
  }
  double slope() {
    return y / x;
  }
};
```

分数类

```
struct Fraction {
  long long num;
  long long den;
  Fraction(long long num=0,long long den=1) {
    if(den<0) {
      num=-num;
      den=-den;
    }
    assert(den!=0);
    long long g=gcd(abs(num),den);
    this->num=num/g;
    this->den=den/g;
  }
  Fraction operator +(const Fraction &o)const {
    return Fraction(num*o.den+o.num,den*o.den);
  }
  Fraction operator -(const Fraction &o)const {
    return Fraction(num*o.den-den*o.num,den*o.den);
  }
  Fraction operator *(const Fraction &o)const {
    return Fraction(num*o.num,den*o.den);
  }
  Fraction operator /(const Fraction &o)const {
    return Fraction(num*o.den,den*o.num);
  }
  bool operator <(const Fraction &o)const {
    return num*o.den< den*o.num;
  }
  bool operator ==(const Fraction &o)const {
    return num*o.den==den*o.num;
  }
};
```

矩阵

```
#define maxm 10
typedef long long LL;

const LL Mod=1e9+7;
struct Matrix {
  int n, m;
  LL mat[maxm][maxm];
  void clear() {
    memset(mat, 0, sizeof(mat));
  }

  Matrix(int n, int m) :n(n), m(m) {
    //不要设置默认构造函数，让编译器检查初始化遗漏
    clear();
  }

  Matrix operator +(const Matrix &M) const {
    Matrix res(n, m);
    for (LL i = 0; i < n; ++i) for (LL j = 0; j < m; ++j) {
       res.mat[i][j] = (mat[i][j] + M.mat[i][j]) % Mod;
    }
    return res;
  }

  Matrix operator *(const Matrix &M) const {
    if (m != M.n){
      std::cout << "Wrong!" << std::endl;
      return Matrix(-1, -1);
    }
    Matrix res(n, M.m);
    res.clear();
    int i,j,k;
    for (i = 0; i < n; ++i)
      for (j = 0; j < M.m; ++j)
        for (k = 0; k < m; ++k) {
          res.mat[i][j] += mat[i][k] * M.mat[k][j]%Mod;
          res.mat[i][j] %= Mod;
        }
    return res;
```

```
  }
  Matrix operator *(const LL &x) const {
    Matrix res(n,m);
    int i,j;
    std::cout << n << ' ' << m << std::endl;
    for (i = 0; i < n; ++i)
      for (j = 0; j < m; ++j)
        res[i][j] = mat[i][j] * x % Mod;
    return res;
  }

  Matrix operator ^(LL b) const { // 矩阵快速幂 ， 取余 Mod
    if (n != m)
      return Matrix(-1, -1);
    Matrix a(*this);
    Matrix res(n, n);
    res.clear();
    for (LL i = 0; i < n; ++i)
      res.mat[i][i] = 1;
    for (; b; b >>= 1) {
      if (b & 1) {
        res = a * res;
      }
      a = a * a;
    }
    return res;
  }

  LL* operator [](int i) {
    return mat[i];
  }

  void Print() const {
    for (int i = 0; i < n; ++i) {
      for (int j = 0; j < m; ++j)
        std::cout << mat[i][j] << ' ';
      std::cout << '\n';
    }
  }
};
```

01 矩阵

```
#include <bitset>
#define maxn 1000
struct Matrix01{
  int n,m;
  std::bitset<maxn> a[maxn];
  void Resize(int x,int y){
    n=x;
    m=y;
  }
  std::bitset<maxn>& operator [] (int n) {
    return a[n];
  }
  void print(){
    for(int i = 0; i < n; ++i)
      std::cout << a[i] << std::endl;
    }
};

Matrix01 operator & (Matrix01 &a,Matrix01 &b){ int i,j,k;
  Matrix01 c;
  c.Resize(a.n,b.m);
  for(i = 0; i < a.n; ++i) {
  c[i].reset();
  for(j = 0; j < b.m; ++j)
    if(a[i][j])
      c[i]|=b[j];
    }
  return c;
}
```

简单大数

```
const int maxn = 10005;   //点的最大个数

int head[maxn], cnt=0;//head 用来表示以 i 为起点的第一条边存储的位置, cnt
  ↪ 读入边的计数器

struct Edge {
  int next; //同一起点的上一条边的储存位置
  int to; //第 i 条边的终点
  int w; //第 i 条边权重
};

Edge edge[maxn];

void addedge(int u,int v,int w) {
  edge[cnt].w = w;
  edge[cnt].to = v;
  edge[cnt].next = head[u];
  head[u] = cnt++;
}

void traverse() {
```

```
  for(int i=0; i<=n; i++) {
    for(int j=head[i]; j! =-1; j=edge[j].next) {
      std::cout << i << " " << head[i].to << " " << head[i].w << '\n';
    }
  }
}
```

大数

```
#include <bits/stdc++.h>
using namespace std;
class BigNum {
public:
  static const int maxn = 9999;
  static const int maxsize = 10;
  static const int dlen = 4;
  int a[105];    //可以控制大数的位数
  int len;       //大数长度
  BigNum(){ len = 1;memset(a,0,sizeof(a)); }   //构造函数
  BigNum(const int);       //将一个 int 类型的变量转化为大数
  BigNum(const char*);     //将一个字符串类型的变量转化为大数
  BigNum(const BigNum &);  //拷贝构造函数
  BigNum &operator=(const BigNum &);   //重载赋值运算符，大数之间进行赋值运算

  BigNum operator+(const BigNum &) const;   //重载加法运算符，两个大数之间的相加运算
  BigNum operator-(const BigNum &) const;   //重载减法运算符，两个大数之间的相减运算
  BigNum operator*(const BigNum &) const;   //重载乘法运算符，两个大数之间的相乘运算
  BigNum operator/(const int   &) const;    //重载除法运算符，大数对一个整数进行相除运算

  BigNum operator^(const int  &) const;    //大数的 n 次方运算
  int    operator%(const int  &) const;    //大数对一个 int 类型的变量进行取模运算
  bool   operator>(const BigNum & T)const;   //大数和另一个大数的大小比较
  bool   operator>(const int & t)const;      //大数和一个 int 类型的变量的大小比较

  void print();       //输出大数

  friend istream&operator >>(istream&in, BigNum &b) {
    char ch[maxsize*4];
    int i = -1;
    in>>ch;
    int l=strlen(ch);
    int count=0,sum=0;
    for(i=l-1;i>=0;) {
      sum = 0;
      int t=1;
      for(int j=0;j<4&&i>=0;j++,i--,t*=10) {
        sum+=(ch[i]-'0')*t;
      }
      b.a[count]=sum;
      count++;
    }
    b.len =count++;
    return in;
  }
  friend ostream& operator<<(ostream& out,  BigNum& b) {  //重载输出运算符
    int i;
    out << b.a[b.len - 1];
    for(i = b.len - 2 ; i >= 0 ; i--) {
      out.width(dlen);
      out.fill('0');
      out << b.a[i];
    }
    return out;
  }

};

BigNum::BigNum(const int b) {    //将一个 int 类型的变量转化为大数
  int c,d = b;
  len = 0;
  memset(a,0,sizeof(a));
  while(d > maxn) {
    c = d - (d / (maxn + 1)) * (maxn + 1);
    d = d / (maxn + 1);
    a[len++] = c;
  }
  a[len++] = d;
}

BigNum::BigNum(const char*s) {    //将一个字符串类型的变量转化为大数
  int t,k,index,l,i;
  memset(a,0,sizeof(a));
  l=strlen(s);
  len=l/dlen;
  if(l%dlen)
    len++;
  index=0;
  for(i=l-1;i>=0;i-=dlen) {
    t=0;
    k=i-dlen+1;
```

```
    if(k<0)
      k=0;
    for(int j=k;j<=i;j++)
      t=t*10+s[j]-'0';
    a[index++]=t;
  }
}

BigNum::BigNum(const BigNum & T) : len(T.len) { //拷贝构造函数
  int i;
  memset(a,0,sizeof(a));
  for(i = 0 ; i < len ; i++)
    a[i] = T.a[i];
}

BigNum & BigNum::operator=(const BigNum & n) {  //重载赋值运算符，大数之间进行赋值运算
  int i;
  len = n.len;
  memset(a,0,sizeof(a));
  for(i = 0 ; i < len ; i++)
    a[i] = n.a[i];
  return *this;
}


BigNum BigNum::operator+(const BigNum & T) const {  //两个大数之间的相加运算
  BigNum t(*this);
  int i,big;      //位数
  big = T.len > len ? T.len : len;
  for(i = 0 ; i < big ; i++) {
    t.a[i] +=T.a[i];
    if(t.a[i] > maxn) {
      t.a[i + 1]++;
      t.a[i] -=maxn+1;
    }
  }
  if(t.a[big] != 0)
    t.len = big + 1;
  else
    t.len = big;
  return t;
}
BigNum BigNum::operator-(const BigNum & T) const {   //两个大数之间的相减运算
  int i,j,big;
  bool flag;
  BigNum t1,t2;
  if(*this>T) {
    t1=*this;
    t2=T;
    flag=0;
  } else {
    t1=T;
    t2=*this;
    flag=1;
  }
  big=t1.len;
  for(i = 0 ; i < big ; i++) {
    if(t1.a[i] < t2.a[i]) {
      j = i + 1;
      while(t1.a[j] == 0)
        j++;
      t1.a[j--]--;
      while(j > i)
        t1.a[j--] += maxn;
      t1.a[i] += maxn + 1 - t2.a[i];
    }
    else
      t1.a[i] -= t2.a[i];
  }
  t1.len = big;
  while(t1.a[t1.len - 1] == 0 && t1.len > 1) {
    t1.len--;
    big--;
  }
  if(flag)
    t1.a[big-1]=0-t1.a[big-1];
  return t1;
}

BigNum BigNum::operator*(const BigNum & T) const {  //两个大数之间的相乘运算
  BigNum ret;
  int i,j,up;
  int temp,temp1;
  for(i = 0 ; i < len ; i++) {
    up = 0;
    for(j = 0 ; j < T.len ; j++) {
      temp = a[i] * T.a[j] + ret.a[i + j] + up;
      if(temp > maxn) {
        temp1 = temp - temp / (maxn + 1) * (maxn + 1);
        up = temp / (maxn + 1);
        ret.a[i + j] = temp1;
      } else {
        up = 0;
        ret.a[i + j] = temp;
      }
    }
    if(up != 0)
      ret.a[i + j] = up;
```

```cpp
  }
  ret.len = i + j;
  while(ret.a[ret.len - 1] == 0 && ret.len > 1)
    ret.len--;
  return ret;
}
BigNum BigNum::operator/(const int & b) const {  //大数对一个整数进行相除运算
  BigNum ret;
  int i,down = 0;
  for(i = len - 1 ; i >= 0 ; i--) {
    ret.a[i] = (a[i] + down * (maxn + 1)) / b;
    down = a[i] + down * (maxn + 1) - ret.a[i] * b;
  }
  ret.len = len;
  while(ret.a[ret.len - 1] == 0 && ret.len > 1)
    ret.len--;
  return ret;
}
int BigNum::operator %(const int & b) const {   //大数对一个 int 类型的变量进行取模运算
  int i,d=0;
  for (i = len-1; i>=0; i--) {
    d = ((d * (maxn+1))% b + a[i])% b;
  }
  return d;
}
BigNum BigNum::operator^(const int & n) const {   //大数的 n 次方运算
  BigNum t,ret(1);
  int i;
  if(n<0)
    exit(-1);
  if(n==0)
    return 1;
  if(n==1)
    return *this;
  int m=n;
  while(m>1) {
    t=*this;
    for( i=1;i<<1<=m;i<<=1) {
      t=t*t;
    }
    m-=i;
    ret=ret*t;
    if(m==1)
      ret=ret*(*this);
  }
  return ret;
}

bool BigNum::operator>(const BigNum & T) const {  //大数和另一个大数的大小比较
  int ln;
  if(len > T.len)
    return true;
  else if(len == T.len) {
    ln = len - 1;
    while(a[ln] == T.a[ln] && ln >= 0)
      ln--;
    return ln >= 0 && a[ln] > T.a[ln];
  } else
    return false;
}

bool BigNum::operator >(const int & t) const {  //大数和一个 int 类型的变量的大小比较
  BigNum b(t);
  return *this>b;
}

void BigNum::print() {    //输出大数
  int i;
  cout << a[len - 1];
  for(i = len - 2 ; i >= 0 ; i--) {
    cout.width(dlen);
    cout.fill('0');
    cout << a[i];
  }
  cout << endl;
}
```

java 大数

1. valueOf(parament); 将参数转换为制定的类型。如：String s=" 12345" ;BigInteger c=BigInteger.valueOf(s); 则 c=12345；
2. add(); 大整数相加；
3. subtract(); 相减；
4. multiply(); 相乘；
5. divide(); 相除取整；
6. remainder(); 取余；
7. pow(); a.pow(b)=a^b；
8. gcd(); 最大公约数；
9. abs(); 绝对值；

10. negate(); 取反数；
11. mod(); a.mod(b)=a%b=a.remainder(b)；
12. max(); min()；
13. public int comareTo()；
14. boolean equals(); 是否相等；
15. BigInteger(String val); 将指定字符串转换为十进制表示形式；
16. BigInteger(String val,int radix); 将指定基数的 BigInteger 的字符串表示形式转换为 BigInteger。

A=BigInteger.ONE 1
B=BigInteger.TEN 10
C=BigInteger.ZERO 0

杂项

离散化

```
//数组离散化 含重复元素
std::sort(sub_a, sub_a+n);
int size = std::unique(sub_a, sub_a+n) - sub_a;//size 为离散化后元素个数
for (i = 0; i < n; i++) {
  a[i] = std::lower_bound(sub_a, sub_a+size, a[i]) - sub_a + 1;//k 为 b[i]
    ↪ 经离散化后对应的值
}

//坐标离散化
int compress(int *x1, int *x2, int w){
  std::vector<int> xs;
  for (int i = 0; i < N; i++) {
    for (int d = -1; d <= 1; d++) {
      int tx1 = x1[i] + d, tx2 = x2[i] + d;
      if (1 <= tx1 && tx1 <= w) xs.push_back(tx1);
      if (1 <= tx2 && tx2 <= w) xs.push_back(tx2);
    }
  }
  std::sort(xs.begin(), xs.end());
  xs.erase(unique(xs.begin(), xs.end()), xs.end());
  for (int i = 0; i < N; i++) {
    x1[i] = find(xs.begin(), xs.end(), x1[i]) - xs.begin();
    x2[i] = find(xs.begin(), xs.end(), x2[i]) - xs.begin();
  }
  return xs.size();
}
```

快速枚举子集

```
void print_subset(int n, int s) {
  for (int i = 0; i < n; i++) {
    if (s & (1 << i)) {
      std::cout << i << " ";
    }
    std::cout << '\n';
  }
}
int main(int argc, char *argv[]) {
  int n;
  std::cin >> n;
  for (int i = 0; i < (1 << n); i++) print_subset(n, i);
}

//当 x 代表集合 x 的子集: for (int i = x; i; i=(i-1)&x) {}
```

跳舞链

```
struct DLX{
  const static int maxn=20010;
  #define FF(i,A,s) for(int i = A[s];i != s;i = A[i])
  int L[maxn],R[maxn],U[maxn],D[maxn];
  int size,col[maxn],row[maxn],s[maxn],H[maxn];
  bool vis[70];
  int ans[maxn],cnt;
  void init(int m){
    for(int i=0;i<=m;i++){
      L[i]=i-1;R[i]=i+1;U[i]=D[i]=i;s[i]=0;
    }
    memset(H,-1,sizeof(H));
    L[0]=m;R[m]=0;size=m+1;
  }
  void link(int r,int c){
     U[size]=c;D[size]=D[c];U[D[c]]=size;D[c]=size;
     if(H[r]<0)H[r]=L[size]=R[size]=size;
     else {
       L[size]=H[r];R[size]=R[H[r]];
       L[R[H[r]]]=size;R[H[r]]=size;
     }
     s[c]++;col[size]=c;row[size]=r;size++;
   }
  void del(int c){//精确覆盖
    L[R[c]]=L[c];R[L[c]]=R[c];
    FF(i,D,c)FF(j,R,i)U[D[j]]=U[j],D[U[j]]=D[j],--s[col[j]];
  }
  void add(int c){  //精确覆盖
    R[L[c]]=L[R[c]]=c;
    FF(i,U,c)FF(j,L,i)++s[col[U[D[j]]=D[U[j]]=j]];
  }
  bool dfs(int k){//精确覆盖
    if(!R[0]){
      cnt=k;return 1;
    }
    int c=R[0];FF(i,R,0)if(s[c]>s[i])c=i;
    del(c);
    FF(i,D,c){
      FF(j,R,i)del(col[j]);
      ans[k]=row[i];if(dfs(k+1))return true;
      FF(j,L,i)add(col[j]);
    }
    add(c);
    return 0;
  }
  void remove(int c){//重复覆盖
    FF(i,D,c)L[R[i]]=L[i],R[L[i]]=R[i];
  }
```

```
  void resume(int c){//重复覆盖
    FF(i,U,c)L[R[i]]=R[L[i]]=i;
  }
 int A(){//估价函数
   int res=0;
   memset(vis,0,sizeof(vis));
   FF(i,R,0)if(!vis[i]){
       res++;vis[i]=1;
       FF(j,D,i)FF(k,R,j)vis[col[k]]=1;
     }
   return res;
 }
 void dfs(int now,int &ans){//重复覆盖
   if(R[0]==0)ans=min(ans,now);
   else if(now+A()<ans){
     int temp=INF,c;
     FF(i,R,0)if(temp>s[i])temp=s[i],c=i;
     FF(i,D,c){
       remove(i);FF(j,R,i)remove(j);
       dfs(now+1,ans);
       FF(j,L,i)resume(j);resume(i);
     }
   }
 }
}dlx;
```

A* 启发式搜索

```
/*
 * Author: Simon
 * 功能: A* 启发式搜索 (例: 八数码问题)
 */
int Hash[9]={1,1,2,6,24,120,720,5040,40320};
int dir[4][2]={-1,0,1,0,0,-1,0,1};
char d[5]="udlr";
int vis[maxn];
struct node{
  int f[3][3];
  int g,h,hashval,x,y;
  bool operator <(const node a) const{
    return a.g+a.h<g+h;
  }
};
struct path{
  int pre;
  char ch;
}p[maxn];
int get_h(int f[][3]){
  int ans=0;
  for(int i=0;i<3;i++){
    for(int j=0;j<3;j++){
      if(f[i][j]){
        ans+=abs(i-(f[i][j]-1)/3)+abs(j-(f[i][j]-1)%3);
      }
    }
  }
  return ans;
}
bool checkedge(node next){
  if(next.x>=0&&next.y>=0&&next.x<3&&next.y<3) return 1;
  return 0;
}
void As_bfs(node e){
  priority_queue<node>q;
  node now,next;
  for(int i=0;i<9;i++) now.f[i/3][i%3]=(i+1)%9;
  int end_ans=get_hash(now);
  e.h=get_h(e);e.g=0;
  e.hashval=get_hash(e);
  p[e.hashval].pre=-1;
  q.push(e);
  while(!q.empty()){
    now=q.top(); q.pop();
    if (now.hashval == end_ans) {
      print(now.hashval);
      cout << endl;
      return;
    }
    if(vis[now.hashval]) continue;vis[now.hashval]=1;
    for(int i=0;i<4;i++){
      next=now;
      next.x=now.x+dir[i][0];
      next.y=now.y+dir[i][1];
      if(checkedge(next)){
        swap(next.f[now.x][now.y], next.f[next.x][next.y]);
        next.hashval = get_hash(next);
        if(vis[next.hashval]) continue;
        next.g++; next.h = get_h(next);
        p[next.hashval].pre=now.hashval;
        p[next.hashval].ch=d[i];
        q.push(next);
      }
    }
  }
}
```

K-D 树

```
#include <queue>
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
const int N = 55555, K = 5;
const int inf = 0x3f3f3f3f;

#define sqr(x) (x)*(x)
int k,n,idx;   //k 为维数,n 为点数
struct point {
  int x[K];
  bool operator < (const point &u) const {
   return x[idx]<u.x[idx];
  }
}po[N];

typedef pair<double,point>tp;
priority_queue<tp>nq;

struct kdTree {
  point pt[N<<2];
  int son[N<<2];

  void build(int l,int r,int rt=1,int dep=0) {
    if(l>r) return;
    son[rt]=r-l;
    son[rt*2]=son[rt*2+1]=-1;
    idx=dep%k;
    int mid=(l+r)/2;
    nth_element(po+l,po+mid,po+r+1);
    pt[rt]=po[mid];
    build(l,mid-1,rt*2,dep+1);
    build(mid+1,r,rt*2+1,dep+1);
  }
  void query(point p,int m,int rt=1,int dep=0) {
    if(son[rt]==-1) return;
    tp nd(0,pt[rt]);
    for(int i=0;i<k;i++) nd.first+=sqr(nd.second.x[i]-p.x[i]);
    int dim=dep%k,x=rt*2,y=rt*2+1,fg=0;
    if(p.x[dim]>=pt[rt].x[dim]) swap(x,y);
    if(~son[x]) query(p,m,x,dep+1);
    if(nq.size()<m) nq.push(nd),fg=1;
    else {
      if(nd.first<nq.top().first) nq.pop(),nq.push(nd);
      if(sqr(p.x[dim]-pt[rt].x[dim])<nq.top().first) fg=1;
    }
    if(~son[y]&&fg) query(p,m,y,dep+1);
  }
}kd;

void print(point &p) {
  for(int j=0;j<k;j++) printf("%d%c",p.x[j],j==k-1?'\n':' ');
}

int main() {
  while(scanf("%d%d",&n,&k)!=EOF) {
    for(int i=0;i<n;i++) for(int j=0;j<k;j++) scanf("%d",&po[i].x[j]);
    kd.build(0,n-1);
    int t,m;
    for(scanf("%d",&t);t--;) {
       point ask;
       for(int j=0;j<k;j++) scanf("%d",&ask.x[j]);
       scanf("%d",&m); kd.query(ask,m);
       printf("the closest %d points are:\n", m);
       point pt[20];
       for(int j=0;!nq.empty();j++) pt[j]=nq.top().second,nq.pop();
       for(int j=m-1;j>=0;j--) print(pt[j]);
    }
  }
  return 0;
}
```

随机

```
//#include <iostream>
//#include <random>

std::vector<int> permutation(100);
for (int i = 0; i < 100; i++) {
  permutation[i] = i+1;
}
std::mt19937_64 mt1(1); //64 位
std::mt19937 mt2(2); //32 位
shuffle(permutation.begin(), permutation.end(), mt2); // 打乱序列
for (auto it: permutation) {
  std::cout << it << " ";
}
```

珂朵莉树（Old Driver Tree)

```
#include <set>
#include <algorithm>

using ll = long long;

struct node {
```

```cpp
  int l, r;
  mutable ll v;
  node(int L, int R = -1, ll V = 0) : l(L), r(R), v(V) {}
  bool operator < (const node& o) const {
    return l < o.l;
  }
};

std::set<node> s;

//分割 SET 返回一个 pos 位置的迭代器
std::set<node>::iterator split(int pos) {
  auto it = s.lower_bound(node(pos));
  if (it != s.end() && it->l == pos) return it;
  --it;
  if (pos > it->r) return s.end();
  int L = it->l, R = it->r;
  ll V = it->v;
  s.erase(it);
  s.insert(node(L, pos - 1, V));
  return s.insert(node(pos, R, V)).first;
}

//区间加值
void add(int l, int r, ll val=1) {
  split(l);
  auto itr = split(r+1), itl = split(l);
  for (; itl != itr; ++itl) itl->v += val;
}

//区间赋值
void assign(int l, int r, ll val = 0) {
  split(l);
  auto itr = split(r+1), itl = split(l);
  s.erase(itl, itr);
  s.insert(node(l, r, val));
}
```

CDQ 分治

```cpp
//Author:marsed
/*
* 将区间分成左右两部分 递归处理
一层递归计算当前左区间的修改操作对右区间的查询操作的影响
当 flag 为 1 代表修改操作 为 0 代表查询操作
*/
#include <algorithm>
#define mid (l + r)/2

const int maxn = "Edit";

struct Node {
  int id, x1,x2;
  int operator<(const Node &b) {   //按照参数的优先级排序
    return ;
  }
};

Node nod[maxn], tmp[maxn];

void cdq(int l, int r) {
  if (l == r) return;
  cdq(l, mid); cdq(mid + 1, r);
  int p = l, q = mid + 1, cnt = 0;
  while (p <= mid&&q <= r) {
    if (nod[p] < nod[q]) {
      if (nod[p].flag) ;    //左区间里的修改操作会对右区间的查询操作有影响
        ↪ 计算影响
      tmp[cnt++] = nod[p++];
    } else {
      if (!nod[q].flag) ;//计算右区间的查询操作的值
      tmp[cnt++] = nod[q++];
    }
  }
  while (p <= mid) tmp[cnt++] = nod[p++];
  while (q <= r) {
    if (!nod[q].flag) ;
    tmp[cnt++] = nod[q++];
  }
  for (int i = l; i <= r; i++)
    nod[i] = tmp[i - l];
}

int main()
{
  cdq(1, q);
  return 0;
}
```

0-1 分数规划

```cpp
template <size_t N, typename T, typename Z = double>
struct zero_one_plan {
  Z f[N];
  Z solve(T *c, T *s, int n, int k) { // max-> sigma(c[i])/sigma(s[i])
    Z l=0,r=*max_element(c,c+n);
    while(fabs(r-l)>eps){
      Z mid=(l+r)/2.;
```

```
      rep(i,0,n)f[i]=1.*c[i]-mid*s[i];
      nth_element(f,f+k,f+n,greater<Z>());
      Z sm=0;
      rep(i,0,k)sm+=f[i];
      if(sm>-eps)l=mid;
      else r=mid;
    }
    return l;
  }
};
```

BM 线性递推

```
//author: xudyh

namespace linear_seq {
  const int N = 10010;
  typedef long long ll;
  constexpr ll mod = (ll) 1e9 + 7;

  ll pow_mod(ll a, ll b) {
    ll r = 1;
    for (a %= mod; b; b >>= 1, a = a * a % mod) {
      if (b & 1)r = r * a % mod;
    }
    return r;
  }

  ll res[N], base[N], _c[N], _md[N];
  vector<int> Md;

  void mul(ll *a, ll *b, int k) {
    k <<= 1;
    for (int i = 0; i < k; ++i) _c[i] = 0;
    k >>= 1;
    for (int i = 0; i < k; ++i) {
      if (a[i]) {
        for (int j = 0; j < k; ++j) {
          _c[i + j] = (_c[i + j] + a[i] * b[j]) % mod;
        }
      }
    }
    for (int i = k + k - 1; i >= k; i--) {
      if (_c[i]) {
        for (const int md: Md) {
          _c[i - k + md] = (_c[i - k + md] - _c[i] * _md[md]) % mod;
        }
      }
    }
    for (int i = 0; i < k; ++i) {
      a[i] = _c[i];
    }
  }

  int solve(ll n, vector<int> a, vector<int> b) { // a 系数 b 初值
    ↪ b[n+1]=a[0]*b[n]+...
//        printf("SIZE %d\n",SZ(b));
    ll ans = 0, pnt = 0;
    int k = (int) a.size();
    assert(a.size() == b.size());
    for (int i = 0; i < k; ++i) {
      _md[k - 1 - i] = -a[i];
    }
    _md[k] = 1;
    Md.clear();
    for (int i = 0; i < k; ++i) {
      if (_md[i] != 0) {
        Md.push_back(i);
      }
    }
    for (int i = 0; i < k; ++i) {
      res[i] = base[i] = 0;
    }
    res[0] = 1;
    while ((1ll << pnt) <= n) {
      pnt++;
    }
    for (int p = pnt; p >= 0; p--) {
      mul(res, res, k);
      if ((n >> p) & 1) {
        for (int i = k - 1; i >= 0; i--) {
          res[i + 1] = res[i];
        }
        res[0] = 0;
        for (const int md: Md) {
          res[md] = (res[md] - res[k] * _md[md]) % mod;
        }
      }
    }
    for (int i = 0; i < k; ++i) {
      ans = (ans + res[i] * b[i]) % mod;
    }
    if (ans < 0) ans += mod;
    return ans;
  }

  vector<int> BM(vector<int> s) {
    vector<int> C(1, 1), B(1, 1);
    int L = 0, m = 1, b = 1;
    for (int n = 0; n < (int) s.size(); ++n) {
```

```
      ll d = 0;
      for (int i = 0; i <= L; ++i) {
        d = (d + (ll) C[i] * s[n - i]) % mod;
      }
      if (d == 0) {
        ++m;
      }
      else if (2 * L <= n) {
        vector<int> T = C;
        ll c = mod - d * pow_mod(b, mod - 2) % mod;
        while (C.size() < B.size() + m) {
          C.push_back(0);
        }
        for (int i = 0; i < (int) B.size(); ++i) {
          C[i + m] = (C[i + m] + c * B[i]) % mod;
        }
        L = n + 1 - L;
        B = T;
        b = d;
        m = 1;
      } else {
        ll c = mod - d * pow_mod(b, mod - 2) % mod;
        while (C.size() < B.size() + m) {
          C.push_back(0);
        }
        for (int i = 0; i < (int) B.size(); ++i) {
          C[i + m] = (C[i + m] + c * B[i]) % mod;
        }
        ++m;
      }
    }
    return C;
  }

  int gao(vector<int> a, ll n) {
    vector<int> c = BM(a);
    c.erase(c.begin());
    for (int &x:c) {
      x = (mod - x) % mod;
    }
    return solve(n, c, vector<int>(a.begin(), a.begin() + c.size()));
  }
}
```