The Code Library of xiaohai

Version 2.0

Volume 1 (Part I, II, III)

xiaohai

Lingxiao Ma   *Yi Li*   *Lanjun Duan*   *Anran Li*

College of Imformation Science and Technology

Beijing Normal University

2015 年 10 月 16 日

Contents

Preface

§ 0.1 Acknowledgement

本模板参考黄锟、詹钰、郭思瑶、kuangbin、cxlove、上交、浙大、吉大等模板，感谢!

§ 0.2 Build History

- 2013 The 38th ACM-ICPC Asia Regional Contest Changchun Site — Version 1.0
  - Author: Lingxiao Ma, Yi Li, Anran Li
  - Time: 2013.12.7-8
- 2014 The 39th ACM-ICPC Asia Regional Contest Anshan Site — Version 2.0 β
  - Author: Lingxiao Ma, Yi Li, Lanjun Duan
  - Time: 2014.10.18-19
- 2014 The 39th ACM-ICPC Asia Regional Contest Guangzhou Site — Version 2.0
  - Author: Lingxiao Ma, Yi Li, Lanjun Duan
  - Time: 2014.11.22-23

Part I

Fundamental

Chapter 1

JAVA(详见附录琨神模板)

§ 1.1 代码框架

```
import java.io.*;
import java.math.*;
import java.util.*;
public class test{
    public static void main(String args[]){
        Scanner in=new Scanner(System.in);
        while(in.hasNext()){
            String tmp=in.next();
            System.out.println(tmp);
        }
    }
}
```

§ 1.2 String

详见附录琨神模板

§ 1.3 BigInteger

详见附录琨神模板

§ 1.4 BigDecimal

详见附录琨神模板

§ 1.5 分数Fraction

```
class Fraction {

    public BigInteger a, b;
```

```
    Fraction() {
        a = BigInteger.ZERO;
        b = BigInteger.ONE;
    }

    Fraction(BigInteger aa, BigInteger bb) {
        a = aa;
        b = bb;
        this.simplify();
    }

    void simplify() {
        BigInteger d = a.gcd(b);
        a = a.divide(d);
        b = b.divide(d);
        if (b.signum() == -1) {
            a = a.negate();
            b = b.negate();
        }
    }

    Fraction add(Fraction f) {
        Fraction res = new Fraction();
        res.b = b.multiply(f.b);
        res.a = a.multiply(f.b).add(b.multiply(f.a));
        res.simplify();
        return res;
    }

    Fraction minus(Fraction f) {
        Fraction res = new Fraction();
        res.b = b.multiply(f.b);
        res.a = a.multiply(f.b).subtract(b.multiply(f.a));
        res.simplify();
        return res;
    }

    Fraction multiply(Fraction f) {
        Fraction res = new Fraction();
        res.b = b.multiply(f.b);
        res.a = a.multiply(f.a);
        res.simplify();
        return res;
    }
```

```
    Fraction divide(Fraction f) {
        Fraction res = new Fraction();
        res.b = b.multiply(f.a);
        res.a = a.multiply(f.b);
        res.simplify();
        return res;
    }
}
```

Chapter 2

Technology

§ 2.1 代码框架

可以在GCC编译选项中添加-Dxysmlx使得程序从in.cpp读入数据，提交OJ时因为没有-Dxysmlx，所以是标准输入

```
// #pragma comment(linker, "/STACK:102400000,102400000")
#include <cstdio>
#include <iostream>
#include <cstring>
#include <string>
#include <cmath>
#include <set>
#include <list>
#include <map>
#include <iterator>
#include <cstdlib>
#include <vector>
#include <queue>
#include <ctime>
#include <stack>
#include <algorithm>
#include <functional>
using namespace std;
typedef long long ll;
#define pb push_back
#define ROUND(x) round(x)
#define FLOOR(x) floor(x)
#define CEIL(x) ceil(x)
const int maxn = 0;
const int maxm = 0;
const int inf = 0x3f3f3f3f;
const ll inf64 = 0x3f3f3f3f3f3f3f3fLL;
const double INF = 1e30;
```

```
const double eps = 1e-6;
const int P[4] = {0, 0, -1, 1};
const int Q[4] = {1, -1, 0, 0};
const int PP[8] = { -1, -1, -1, 0, 0, 1, 1, 1};
const int QQ[8] = { -1, 0, 1, -1, 1, -1, 0, 1};
int kase;
void init()
{
    kase++;
}
void input()
{
    //
}
void debug()
{
    //
}
void solve()
{
    //
}
void output()
{
    //
}
int main()
{
    // 32-bit
    // int size = 256 << 20; // 256MB
    // char *p = (char *)malloc(size) + size;
    // __asm__("movl %0, %%esp\n" :: "r"(p));

    // 64-bit
    // int size = 256 << 20; // 256MB
    // char *p = (char *)malloc(size) + size;
    // __asm__("movq %0, %%rsp\n" :: "r"(p));

    // std::ios_base::sync_with_stdio(false);
#ifdef xysmlx
    freopen("in.txt", "r", stdin);
#endif

    kase = 0;
    while (1)
```

```
    {
        init();
        input();
        solve();
        output();
    }
    return 0;
}
```

§ 2.2 手动扩栈

```
/**
*32位G++
*需要放在main函数内
*/
int size = 256 << 20; // 256MB
char *p = (char *)malloc(size) + size;
__asm__("movl %0, %%esp\n" :: "r"(p));

/**
*64位G++
*需要放在main函数内
*/
int size = 256 << 20; // 256MB
char *p = (char *)malloc(size) + size;
__asm__("movq %0, %%rsp\n" :: "r"(p));

/**
*C++
*/
#pragma comment(linker, "/STACK:102400000,102400000")
```

§ 2.3 输出格式控制

2.3.1 C

```
输出long double: %Ld
科学计数法输出long double: %Le
科学计数法输出double: %e
```

2.3.2 C++

```
需要#include <iomanip>

setprecision(n) 设显示小数精度为n位
setw(n) 设域宽为n个字符
```

setioflags(ios::fixed) 固定的浮点显示
setioflags(ios::scientific) 指数表示
setiosflags(ios::left) 左对齐
setiosflags(ios::right) 右对齐
setiosflags(ios::skipws 忽略前导空白
setiosflags(ios::uppercase) 16进制数大写输出
setiosflags(ios::lowercase) 16进制小写输出
setiosflags(ios::showpoint) 强制显示小数点
setiosflags(ios::showpos) 强制显示符号

§ 2.4 输入输出优化

2.4.1 输入整数(正/负)

```
inline int ReadInt()
{
    bool flag = 0;
    char ch = getchar();
    int data = 0;
    while (ch < '0' || ch > '9')
    {
        if (ch == '-') flag = 1;
        ch = getchar();
    }
    do
    {
        data = data * 10 + ch - '0';
        ch = getchar();
    }
    while (ch >= '0' && ch <= '9');
    return flag ? -data : data;
}
```

2.4.2 输出整数

```
void print( int k )
{
    num = 0;
    while ( k > 0 ) ch[++num] = k % 10, k /= 10;
    while ( num )
        putchar( ch[num--] + 48 );
    putchar( 32 );
}
```

Chapter 3

Standard Templete Library(详见附录琨神模板)

§ 3.1 vector—数组/向量

front()	头元素(vector[0])
back()	尾元素
size()	目前大小
push_back()	从尾加入元素
pop_back()	从尾删除元素
empty()	测试是否空
sort(vec.begih(),vec.end(),cmp())	排序

§ 3.2 list—表

List($\#include < list >$) 又叫链表，是一种双线性列表，只能顺序访问（从前向后或者从后向前）

list仍然包涵了erase(),begin(),end(),insert(),push_back(),push_front() 这些基本函数

list1.merge(list2)	合并两个**排序**列表
list.sort()	列表的排序(<)

§ 3.3 string—字符串

将一个字符串截断

```
char *ch=strtok(str," ");//以空格截断，可任意改
while(ch!=NULL)
{
    //处理截断下来的字符串
    ch=strtok(NULL," ");
}
```

字符串转int, double

```
atoi(str);////返回int
strtod(str,NULL);//返回double，第二个参数表示截取后的位置
strtol(str,NULL,base);//返回long，base表示是几进制，比如base=10表示10进制，第二个参数同上
```

§ 3.4 map/multimap— 映射/多重映射

在map中是不允许一个键对应多个值的，在multimap 中，不支持$operator$[], 也就是说不支持map 中允许的下标操作。

$map < key, value >$ maptemp	第一个为唯一的关键字(如果为结构体则重载小于号) 第二个为值
$insert(pair < int, string > (key, value))$	插入(key,value)
$maptemp[key] = value$	插入(key,value)
$ite- > first$	ite指向的key
$ite- > second$	ite指向的value
$size()$	返回实际容量
$clear()$	清空
$empty()$	判空
$erase(ite)/erase(key)/erase(beg, end)$	删除
$find(key)$	查找，返回iterator(pair)
$count(key)$	由于key 的唯一性，所以返回0或1

§ 3.5 queue—队列

§ 3.6 deque—双向队列

assign(beg,end)	将[beg;end) 区间中的数据赋值
assign(n,elem)	将n各elem赋值
at(idx)	返回idx指向的数据
front()	返回第一个值
back()	返回最后一个值
begin()	返回头指针
end()	返回尾指针
rbegin	返回逆向队列头指针
rend()	返回逆向队列尾指针
push_back(elem)	尾插入
push_front(elem)	头插入
insert(pos,elem)	pos 插入
insert(pos,n,elem)	pos插入n 个
insert(pos,beg,end)	pos插入[beg;end)
pop_back()	尾删除
pop_front()	头删除
erase(pos)	删除pos
erase(beg,end)	删除[beg;end)
empty()	判断是否空
max_size()	最大容量
resize(num)	重赋容量
size()	实际容量
c1.swap(c2)/swap(c1,c2)	交换c1,c2

§ 3.7 stack—栈

§ 3.8 priority_queue— 优先队列/最大堆

§ 3.9 set/multiset— 集合/多重集合

集和多集的区别是：set支持唯一键值，set中的值都是特定的，而且只出现一次；而multiset中可以出现副本键，同一值可以出现多次。

$template < key, cmp >$	key是所存储的键的类型 cmp是排序比较函数的类型
insert(key)	插入key

§ 3.10 algorithm—算法

reverse(起始,终止) reverse(arr,arr+6)	翻转数组 将arr[0] 到arr[5]位置翻转
copy(起始,终止,目标) 例：copy(arr,(arr+6),arr1)	复制元素 复制arr[0]到arr[5]到arr1[0]开始覆盖
find(起始,终止,value)	如果找到，返回一个指向value 在序列中第一次出现的迭代； 如果没有找到，就返回一个指向last的迭代
search(first1,last1,first2,last2)	在源序列[first1,last1-1]查找目标序列[first2，last2-1]。 如果查找成功，返回一个指向源序列中目标序列出现的首位置的迭代。 查找失败则返回一个指向last 的迭代。
swap(a,b)	交换a,b位置
sort(起始,终止,cmp)	排序
count(起始,终止,value)	返回value 的数量
make_heap(起始,终止,cmp)	建堆，默认是大跟堆
pop_heap(起始,终止,cmp)	默认以建堆，把first和last交换
push_heap(起始,终止,cmp)	假设由[first,last-1) 是一个有效的堆 然后，再把堆中的新元素加进来，做成一个堆。
sort_heap(起始,终止,cmp)	对[first,last)中的序列进行排序。 它假设这个序列是有效堆。
fill(first,last,val)	first为容器的首迭代器 last 为容器的末迭代器 val 为将要替换的值
max_element(beg,end)	返回最大值，可用*max_element对数组操作
min_element(beg,end)	返回最大值，可用*min_element对数组操作

Chapter 4

Attention

§ 4.1 Tips

1. printf中%.1f 是四舍五入的%.1lf是取floor。(未知是否正确)
2. for循环中注意不要把函数写在for的条件判断中，否则容易TLE。比如：for(int i=0;i¡strlen(str);i++) 会TLE；先计算n=strlen(str); 然后for(int i=0;i¡n;i++)可以AC。。。
3. mod过后有减法的要注意数变成负数。
4. 调用多个STL的max或者min时，尽量写在一起（例如：a=max(b,max(c,d))），可节约时间。
5. 栈调用一般12万层左右。
6. 手动扩栈：#pragma comment(linker, ”/STACK:102400000,102400000”) , 注意用VC++交才行
7. 用G++ T的可以用C++交试一下，C++比G++快。
8. 乘法可用取log转化加法，同理用exp将加法转化为乘法
9. double如果暴精度，可以考虑取log、exp或者边乘边除等方法解决
10. double的eps向大开、向小开需注意

§ 4.2 Error

4.2.1 Compile Error

4.2.2 Runtime Error

1. 除以0
2. 指针
3. 数组越界
4. 栈溢出(可以尝试手动扩栈，用C++交)

4.2.3 Time Limit Exceeded

1. 死循环，如while(1)未跳出
2. 算法时间复杂度太高
3. for循环条件设置问题

§ 4.3 Debug Technology

检测死循环 puts("1") 看结果或者用system("pause")

检测错误结果 输出中间变量

检测爆内存 用VC测是指针问题还是爆栈，然后用puts("1")看结果或者用system("pause")

Part II

Mathematics

Chapter 5

Basic

§ 5.1 高精度算法

5.1.1 大整数类From kuangbin

```
/* ***********************************************
Author        :kuangbin
Created Time  :2013/9/28 星期六 12:54:45
File Name     :2013长春网络赛\1004.cpp
************************************************ */

#pragma comment(linker, "/STACK:1024000000,1024000000")
#include <stdio.h>
#include <string.h>
#include <iostream>
#include <algorithm>
#include <vector>
#include <queue>
#include <set>
#include <map>
#include <string>
#include <math.h>
#include <stdlib.h>
#include <time.h>
using namespace std;
/*
 * 完全大数模板
 * 输入cin>>a
 * 输出a.print();
 * 注意这个输入不能自动去掉前导0的，可以先读入到char数组，去掉前导0，再用构造函数。
 */
#define MAXN 9999
#define MAXSIZE 1010
```

```
#define DLEN 4

class BigNum
{
private:
    int a[500];  //可以控制大数的位数
    int len;
public:
    BigNum()
    {
        len = 1;    //构造函数
        memset(a, 0, sizeof(a));
    }
    BigNum(const int);     //将一个int类型的变量转化成大数
    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
    friend istream &operator>>(istream &, BigNum &); //重载输入运算符
    friend ostream &operator<<(ostream &, BigNum &); //重载输出运算符

    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
};
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;
}
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;
}
ostream &operator<<(ostream &out, BigNum &b) //重载输出运算符
{
    int i;
    cout << b.a[b.len - 1];
    for (i = b.len - 2; i >= 0; i--)
    {
        printf("%04d", b.a[i]);
    }
    return out;
}
BigNum BigNum::operator+(const BigNum &T)const   //两个大数之间的相加运算
{
    BigNum t(*this);
    int i, big;
    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[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
{
    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;
    }
    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
{
    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--;
        if (ln >= 0 && a[ln] > T.a[ln])
            return true;
        else
            return false;
    }
    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;
    printf("%d", a[len - 1]);
    for (i = len - 2; i >= 0; i--)
        printf("%04d", a[i]);
    printf("\n");
}

int main()
{
    //freopen("in.txt","r",stdin);
    //freopen("out.txt","w",stdout);
    int m, n;
    int T;
    scanf("%d", &T);
    while (T--)
    {
        scanf("%d%d", &m, &n);
        BigNum tt = 1;
        for (int i = 1; i < n; i++)
            tt = tt * m;
        int tmp = n;
        for (int i = 2; i <= n; i++)
        {
            while ( tmp % i == 0 && (tt % i == 0) )
            {
                tmp /= i;
                tt = tt / i;
            }
        }
        printf("%d/", tmp);
        tt.print();

    }
    return 0;
}
```

5.1.2 大整数类

```
#include<stdio.h>
#include<string.h>

/* windy7926778, 非负大数,10^4进制,所有的函数中的大数都为引用*/
const int N = 80; // 10 进制下的最大位数
struct big
{
```

```
    int d[N / 4];
    int len;
};
void  pri(big &x) // 输出大数 x ，不输出换行符
{
    int i;
    printf("%d", x.d[x.len - 1]);
    for (i = x.len - 2; i >= 0; i--)
        printf("%04d", x.d[i]);
}
void set(big &x, char str[]) //把不带符号的非空字符串 str 转化为大数 x ，并去除前
导零
{
    memset(&x, 0, sizeof(x));
    int i, w = 1, now = 0;
    for (i = strlen(str) - 1; i >= 0; i--)
    {
        now += w * (str[i] - 48);
        w *= 10;
        if (w == 10000)
        {
            x.d[x.len++] = now;
            now = 0;
            w = 1;
        }
    }
    if (now)x.d[x.len++] = now;
    while (x.len > 1 && !x.d[x.len - 1])x.len--;
}
void set(big &x, int n) //把非负整数 n 转化为大数 x
{
    memset(&x, 0, sizeof(x));
    if (!n)
    {
        x.len = 1;
        x.d[0] = 0;
    }
    else
    {
        while (n)
        {
            x.d[x.len++] = n % 10000;
            n /= 10000;
        }
    }
```

```
}
void add(big &x, big &y) // 大数 x = 大数 x + 大数 y
{
    int i, c = 0;
    for (i = 0; i < x.len || i < y.len; i++)
    {
        x.d[i] += y.d[i] + c;
        if (x.d[i] >= 10000)
        {
            x.d[i] -= 10000;
            c = 1;
        }
        else  c = 0;
    }
    if (c)x.d[i++] = 1;
    x.len = i;
}
void sub(big &x, big &y) // 大数 x = 大数 x - 大数 y , 要求 x >= y
{
    int i, c = 0;
    for (i = 0; i < y.len; i++)
    {
        x.d[i] -= y.d[i] + c;
        if (x.d[i] < 0)
        {
            x.d[i] += 10000;
            c = 1;
        }
        else c = 0;
    }
    for (; c; i++)
    {
        x.d[i] -= c;
        c = 0;
        if (x.d[i] < 0)
        {
            x.d[i] += 10000;
            c = 1;
        }
    }
    while (x.len > 1 && !x.d[x.len - 1])
        x.len--;
}

void mul(big &z, big &x, big &y) // 大数 z = 大数 x * 大数 y
```

```
{
    memset(&z, 0, sizeof(z));
    z.len = x.len + y.len;
    int i, j;
    for (i = 0; i < x.len; i++)
        for (j = 0; j < y.len; j++)
        {
            z.d[i + j] += x.d[i] * y.d[j];
            z.d[i + j + 1] += z.d[i + j] / 10000;
            z.d[i + j] %= 10000;
        }
    while (z.len > 1 && !z.d[z.len - 1])
        z.len--;
}

void mul(big &z, big &x, int e) // 大数 z = 大数 x * 非负整数 e
{
    memset(&z, 0, sizeof(z));
    int i, c = 0;
    for (i = 0; i < x.len; i++)
    {
        z.d[i] += x.d[i] * e;
        z.d[i + 1] = z.d[i] / 10000;
        z.d[i] %= 10000;
    }
    if (z.d[i] / 10000)
    {
        z.d[i + 1] = z.d[i] / 10000;
        z.d[i] %= 10000;
        i++;
    }
    z.len = i + 1;
    while (z.len > 1 && !z.d[z.len - 1])
        z.len--;
}

bool cmp(big &x, big &y) //return 大数 x <= 大数 y;
{
    if (x.len < y.len)
        return true;
    if (x.len > y.len)
        return false;
    int i;
    for (i = x.len - 1; i >= 0; i--)
        if (x.d[i] != y.d[i])
```

```
            return x.d[i] < y.d[i];
    return true;
}

void div(big &z, big &x, big &y, big &t) // 大数 z = 大数 x / 大数 y  余数为 t
{
    memset(&z, 0, sizeof(z));
    int i, j, p, q, now;
    big e;
    set(t, 0);
    for (i = x.len - 1; i >= 0; i--)
    {
        for (j = t.len; j > 0; j--)
            t.d[j] = t.d[j - 1];
        t.len++;
        t.d[0] = x.d[i];
        while (t.len > 1 && !t.d[t.len - 1])
            t.len--;
        p = 0;
        q = 9999;
        while (p <= q)
        {
            now = (p + q) >> 1;
            mul(e, y, now);
            if (cmp(e, t))
                p = now + 1;
            else
                q = now - 1;
        }
        mul(e, y, q);
        sub(t, e);
        z.d[i] = q;
    }
    z.len = x.len;
    while (z.len > 1 && !z.d[z.len - 1])
        z.len--;
}

char b[100000];
char a[100000];

int main()
{
    while (scanf("%s%s", a, b) != EOF)
    {
```

```
        big A, B;
        set(A, a);
        set(B, b);
        add(A, B);
        pri(A);
        printf("\n");
    }
    return 0;
}
```

5.1.3 高精度FFT乘法($O(N \lg N)$)

```
/**
*HDU 1402 A * B Problem Plus
*计算A*B
*/
#include<iostream>
#include<cmath>
using namespace std;
typedef struct vir
{
    double re, im;
    vir() {}
    vir(double a, double b)
    {
        re = a;
        im = b;
    }
    vir operator +(const vir &b)
    {
        return vir(re + b.re, im + b.im);
    }
    vir operator -(const vir &b)
    {
        return vir(re - b.re, im - b.im);
    }
    vir operator *(const vir &b)
    {
        return vir(re * b.re - im * b.im, re * b.im + b.re * im);
    }
} vir;
vir x1[200005], x2[200005];
const double Pi = acos(-1.0);
void change(vir *x, int len, int loglen)
{
```

```
    int i, j, k, t;
    for (i = 0; i < len; i++)
    {
        t = i;
        for (j = k = 0; j < loglen; j++, t >>= 1)
            k = (k << 1) | (t & 1);
        if (k < i)
        {
            vir wt =  x[k];
            x[k] = x[i];
            x[i] = wt;
        }
    }
}
void fft(vir *x, int len, int loglen)
{
    int i, j, t, s, e;
    change(x, len, loglen);
    t = 1;
    for (i = 0; i < loglen; i++, t <<= 1)
    {
        s = 0;
        e = s + t;
        while (s < len)
        {
            vir a, b, wo(cos(Pi / t), sin(Pi / t)), wn(1, 0);
            for (j = s; j < s + t; j++)
            {
                a = x[j];
                b = x[j + t] * wn;
                x[j] = a + b;
                x[j + t] = a - b;
                wn = wn * wo;
            }
            s = e + t;
            e = s + t;
        }
    }
}
void dit_fft(vir *x, int len, int loglen)
{
    int i, j, s, e, t = 1 << loglen;
    for (i = 0; i < loglen; i++)
    {
        t >>= 1;
```

```
        s = 0;
        e = s + t;
        while (s < len)
        {
            vir a, b, wn(1, 0), wo(cos(Pi / t), -sin(Pi / t));
            for (j = s; j < s + t; j++)
            {
                a = x[j] + x[j + t];
                b = (x[j] - x[j + t]) * wn;
                x[j] = a;
                x[j + t] = b;
                wn = wn * wo;
            }
            s = e + t;
            e = s + t;
        }
    }
    change(x, len, loglen);
    for (i = 0; i < len; i++)x[i].re /= len;
}
int main()
{
    char a[100005], b[100005];
    int i, len1, len2, t, over, len, loglen;

    while (scanf("%s%s", a, b) != EOF)
    {
        len1 = strlen(a) << 1;
        len2 = strlen(b) << 1;
        len = 1;
        loglen = 0;
        while (len < len1)
        {
            len <<= 1;
            loglen++;
        }
        while (len < len2)
        {
            len <<= 1;
            loglen++;
        }
        for (i = 0; a[i] != '\0'; i++)
        {
            x1[i].re = a[i] - '0';
            x1[i].im = 0;
```

```
        }
        for (; i < len; i++)
            x1[i].re = x1[i].im = 0;
        for (i = 0; b[i] != '\0'; i++)
        {
            x2[i].re = b[i] - '0';
            x2[i].im = 0;
        }
        for (; i < len; i++)
            x2[i].re = x2[i].im = 0;
        fft(x1, len, loglen);
        fft(x2, len, loglen);
        for (i = 0; i < len; i++)
            x1[i] = x1[i] * x2[i];
        dit_fft(x1, len, loglen);
        for (i = (len1 + len2) / 2 - 2, over = loglen = 0; i >= 0; i--)
        {
            t = x1[i].re + over + 0.5;
            a[loglen++] = t % 10;
            over =  t / 10;
        }
        while (over)
        {
            a[loglen++] = over % 10;
            over /= 10;
        }
        for (loglen--; loglen >= 0 && !a[loglen]; loglen--);
        if (loglen < 0)
            putchar('0');
        else
            for (; loglen >= 0; loglen--)
                putchar(a[loglen] + '0');
        putchar('\n');
    }
    return 0;
}
```

§ 5.2 统计x(二进制)中1的个数(未知复杂度)

```
/**
*统计x(二进制)中1的个数
*/
int cb(int x)
{
    int cnt;
```

```
    for(cnt=0; x>0; cnt++) x&=x-1;
    return cnt;
}
```

§ 5.3 二进制生成子集($O(2^N)$)

```
/**
*二进制生成子集：生成0--n-1的子集S
*/
void print_subset(int n,int s)
{
    for(int i=0; i<n; i++)
        if(s&(1<<i))printf("%d ",i);//此时也可以用比较弱化的版本写出统计1的个数
    printf("\n");
}
/*枚举0<=x<2^n中数*/
void subset(int n)
{
    int t=1<<n;
    for(int s=0; s<t; s++) print_subset(n,s);
}
```

Chapter 6

Computation Mathematics

§ 6.1 计算中序表达式($O(n)$)

```
/**
*计算中序表达式
*输入: str[](中序表达式, 示例: 1.1 + 2.2 #)(注: 必须包含符号, 否则无法计算)
*输出: 计算结果Item(可自定义类型)
*/
typedef long long Item;
const int MAXN=0;
const int OpNum=7;//操作符的数目
const char op[OpNum+1]="#+-*/()";// 操作符
const int out[OpNum]= {0,2,2,4,4,6,1};//操作符外优先级
const int in[OpNum]= {0,3,3,5,5,1,6};//操作符内优先级
int GetPri(char ope)//得到对应op 数组的下标
{
    int i;
    for(i=0; i<OpNum; i++) if(ope==op[i]) break;
    return i;
}
Item calculator(char *str)
{
    stack<Item> number;
    stack<char> opera;
    char temps[5*MAXN],ope;
    Item tempi,a,b;
    int i,j,k,len,ch1,ch2;
    bool OpFlag,flag,EndFlag=false;
    len=strlen(str);
    i=0;
    opera.push('#');
    while(!EndFlag)
    {
```

```
if(opera.top()=='#'&&i>=len) EndFlag=true;
j=0;
flag=false;
OpFlag=false;
for(; str[i]!=' '&&i<len; i++,j++) temps[j]=str[i];
if(i<len)
{
    temps[j]=0;
    i++;
}
if(i>=len)
{
    temps[0]='#';
    temps[1]=0;
}
for(k=0; k<OpNum; k++)
{
    if(temps[0]==op[k])
    {
        if(temps[1]!=0) flag=true;
        else OpFlag=true;
    }
}
if(OpFlag)
{
    ope=temps[0];
    ch1=GetPri(opera.top());
    ch2=GetPri(ope);
    if(in[ch1]>out[ch2])
    {
        b=number.top();
        number.pop();
        a=number.top();
        number.pop();
        switch(opera.top())
        {
        case '+':
            number.push(a+b);
            break;
        case '-':
            number.push(a-b);
            break;
        case '*':
            number.push(a*b);
            break;
```

```
                case '/':
                    number.push(a/b);
                    break;
                }
                opera.pop();
                if(i<len) i=i-2;
            }
            else if(in[ch1]<out[ch2]) opera.push(ope);
            else opera.pop();
        }
        else
        {
            tempi=strtod(temps,NULL);
            number.push(tempi);
        }
    }
    return number.top();
}
```

§ 6.2 快速幂取模($O(\lg n)$)

```
/**
*快速幂取模
*输入: a,b,n
*输出: (a^b)%n
*/
typedef long long LL;
LL modexp(LL a,LL b,LL n)
{
    LL ret=1;
    while(b)
    {
       //基数存在
       if(b&1) ret=ret*a%n;
       a=a*a%n;
       b>>=1;
    }
    return ret;
}
```

§ 6.3 二分法：单调函数($O(\lg(nf(n)))$)

可以使用STL 的lower_bound() 函数; 例子：lower_bound(A,A+j,A[j]-S);
或者:

```
//(L=...,R=...;)
```

```
while(L<R)
{
    int M=L+(R-L)/2;
    if(test(M)) R=M;
    else L=M+1;
}
```

§ 6.4 迭代法(略)

Too Simple

§ 6.5 三分法：凸性函数($O(\lg(nf(n)))$)

- 类似二分的定义$Left$和$Right$，$mid = (Left + Right)/2$，$midmid = (mid + Right)/2$;
-   - 如果mid 靠近极值点，则$Right = midmid$;
    - 否则(即$midmid$靠近极值点)，则$Left = mid$;

```
/**
*三分法求极大值
*输入: Calc()函数,left,right
*输出: left,right
*/
double Calc(Type a)
{
    /* 根据题目的意思计算 */
}

void Solve(void)
{
    double Left, Right;
    double mid, midmid;
    double mid_value, midmid_value;
    Left = MIN;
    Right = MAX;
    while (Left + EPS < Right)
    {
        mid = (Left + Right) / 2;
        midmid = (mid + Right) / 2;
        mid_area = Calc(mid);
        midmid_area = Calc(midmid);
        // 假设求解最大极值
        if (mid_area >= midmid_area) Right = midmid;
        else Left = mid;
        // 求解最小极值改动如下
        /*if (mid_area < midmid_area) Right = midmid;
```

```
        else Left = mid;*/
    }
}
```

§ 6.6 解线性方程组

6.6.1 LU 分解(方阵)($O(N^3)$)(优于高斯消元)

LU分解是矩阵分解的一种，可以将一个方阵分解为一个下三角矩阵和一个上三角矩阵的乘积（有时是它们和一个置换矩阵的乘积)。LU分解主要应用在数值分析中，用来解线性方程、求反矩阵或计算行列式。

一个可逆矩阵可以进行LU 分解当且仅当它的所有子式都非零。如果要求其中的L 矩阵（或U 矩阵）为单位三角矩阵，那么分解是唯一的。同理可知，矩阵的LDU 可分解条件也相同，并且总是唯一的。

求解$Ax = LUx = b$ 时，先解$Ly = b$ 得到y，再解$Ux = y$ 得到x。

```
/**
*高斯消元法: n*n=n的矩阵的($O(N^3)$)
*输入: a[][] = b[]
*输出: b[](解),gauss()(是否有解)
*/
//const int maxn=0;
//const double eps=1e-6;
double a[maxn][maxn],b[maxn];
bool gauss(int n)
{
    int i, j, k, row;
    double maxp, t;
    for (k = 0; k < n; k++)
    {
        for (maxp = 0, i = k; i < n; i++)
            if (fabs(a[i][k]) > fabs(maxp))
                maxp = a[row = i][k];
        if (fabs(maxp) < eps) return false;
        if (row != k)
        {
            for (j = k; j < n; j++)
            {
                t = a[k][j];
                a[k][j] = a[row][j];
                a[row][j] = t;
            }
            t = b[k];
            b[k] = b[row];
            b[row] = t;
        }
```

```
        for (j = k + 1; j < n; j++)
        {
            a[k][j] /= maxp;
            for (i = k + 1; i < n; i++)
                a[i][j] -= a[i][k] * a[k][j];
        }
        b[k] /= maxp;
        for (i = k + 1; i < n; i++) b[i] -= b[k] * a[i][k];
    }
    for (i = n - 1; i >= 0; i--)
        for (j = i + 1; j < n; j++)
            b[i] -= a[i][j] * b[j];
    return true;
}
```

6.6.2 高斯消元($O(N^3)$)

多功能复杂版

```
/**
*高斯消元($O(N^3)$)
*输入: a[][]=a[n](n*(n+1))
*输出: x[](解集)
*/
#include<stdio.h>
#include<algorithm>
#include<iostream>
#include<string.h>
#include<math.h>
using namespace std;

const int maxn=50;

int a[maxn][maxn];//增广矩阵
int x[maxn];//解集
bool free_x[maxn];//标记是否是不确定的变元

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

```
    return a;
}
inline int lcm(int a,int b)
{
    return a/gcd(a,b)*b;//先除后乘，防溢出
}

// 高斯消元法解方程组(Gauss-Jordan elimination).(-2表示有浮点数解，但无整数解，
//-1表示无解，0表示唯一解，大于0表示无穷解，并返回自由变元的个数)
//有equ个方程，var个变元。增广矩阵行数为equ,分别为0到equ-1,列数为var+1,分别为0到var.
int Gauss(int equ,int var)
{
    int i,j,k;
    int max_r;// 当前这列绝对值最大的行.
    int col;//当前处理的列
    int ta,tb;
    int LCM;
    int temp;
    int free_x_num;
    int free_index;

    for(int i=0; i<=var; i++)
    {
        x[i]=0;
        free_x[i]=true;
    }

    //转换为阶梯阵.
    col=0; // 当前处理的列
    for(k = 0; k < equ && col < var; k++,col++)
    {
        // 枚举当前处理的行.
        // 找到该col列元素绝对值最大的那行与第k行交换.(为了在除法时减小误差)
        max_r=k;
        for(i=k+1; i<equ; i++)
        {
            if(abs(a[i][col])>abs(a[max_r][col])) max_r=i;
        }
        if(max_r!=k)
        {
            // 与第k行交换.
            for(j=k; j<var+1; j++) swap(a[k][j],a[max_r][j]);
        }
        if(a[k][col]==0)
        {
```

```
            // 说明该col列第k行以下全是0了，则处理当前行的下一列.
            k--;
            continue;
        }
        for(i=k+1; i<equ; i++)
        {
            // 枚举要删去的行.
            if(a[i][col]!=0)
            {
                LCM = lcm(abs(a[i][col]),abs(a[k][col]));
                ta = LCM/abs(a[i][col]);
                tb = LCM/abs(a[k][col]);
                if(a[i][col]*a[k][col]<0)tb=-tb;// 异号的情况是相加
                for(j=col; j<var+1; j++)
                {
                    a[i][j] = a[i][j]*ta-a[k][j]*tb;
                }
            }
        }
    }

    // 1. 无解的情况: 化简的增广阵中存在(0, 0, ..., a)这样的行(a != 0).
    for (i = k; i < equ; i++)
    {
        // 对于无穷解来说，如果要判断哪些是自由变元，那么初等行变换中的交换就会影
响，则要记录交换.
        if (a[i][col] != 0) return -1;
    }
    // 2. 无穷解的情况: 在var * (var + 1)的增广阵中出现(0, 0, ..., 0) 这样的行，
即说明没有形成严格的上三角阵.
    // 且出现的行数即为自由变元的个数.
    if (k < var)
    {
        // 首先，自由变元有var - k 个，即不确定的变元至少有var - k 个.
        for (i = k - 1; i >= 0; i--)
        {
            // 第i行一定不会是(0, 0, ..., 0)的情况，因为这样的行是在第k行到第equ 行.
            // 同样，第i行一定不会是(0, 0, ..., a), a != 0的情况，这样的无解的.
            free_x_num = 0; // 用于判断该行中的不确定的变元的个数，如果超过1个，
则无法求解，它们仍然为不确定的变元.
            for (j = 0; j < var; j++)
            {
                if (a[i][j] != 0 && free_x[j]) free_x_num++, free_index = j;
            }
            if (free_x_num > 1) continue; // 无法求解出确定的变元.
```

```
            // 说明就只有一个不确定的变元free_index，那么可以求解出该变元，且该变
元是确定的.
            temp = a[i][var];
            for (j = 0; j < var; j++)
            {
                if (a[i][j] != 0 && j != free_index) temp -= a[i][j] * x[j];
            }
            x[free_index] = temp / a[i][free_index]; // 求出该变元.
            free_x[free_index] = 0; // 该变元是确定的.
        }
        return var - k; // 自由变元有var - k个.
    }
    // 3. 唯一解的情况: 在var * (var + 1)的增广阵中形成严格的上三角阵.
    // 计算出Xn-1, Xn-2 ... X0.
    for (i = var - 1; i >= 0; i--)
    {
        temp = a[i][var];
        for (j = i + 1; j < var; j++)
        {
            if (a[i][j] != 0) temp -= a[i][j] * x[j];
        }
        if (temp % a[i][i] != 0) return -2; // 说明有浮点数解，但无整数解.
        x[i] = temp / a[i][i];
    }
    return 0;
}
int main(void)
{
    freopen("in.txt", "r", stdin);
    freopen("out.txt","w",stdout);
    int i, j;
    int equ,var;
    while (scanf("%d %d", &equ, &var) != EOF)
    {
        memset(a, 0, sizeof(a));
        for (i = 0; i < equ; i++)
        {
            for (j = 0; j < var + 1; j++)
            {
                scanf("%d", &a[i][j]);
            }
        }
        int free_num = Gauss(equ,var);
        if (free_num == -1) printf(" 无解!\n");
        else if (free_num == -2) printf("有浮点数解，无整数解!\n");
```

```
        else if (free_num > 0)
        {
            printf("无穷多解! 自由变元个数为%d\n", free_num);
            for (i = 0; i < var; i++)
            {
                if (free_x[i]) printf("x%d 是不确定的\n", i + 1);
                else printf("x%d: %d\n", i + 1, x[i]);
            }
        }
        else
        {
            for (i = 0; i < var; i++)
            {
                printf("x%d: %d\n", i + 1, x[i]);
            }
        }
        printf("\n");
    }
    return 0;
}
```

kuangbin版

```
double a[maxn][maxn], x[maxn];
int equ, var;
int Gauss()
{
    int i, j, k, col, max_r;
    for (k = 0, col = 0; k < equ && col < var; k++, col++)
    {
        max_r = k;
        for (i = k + 1; i < equ; i++)
            if (fabs(a[i][col]) > fabs(a[max_r][col]))
                max_r = i;
        if (fabs(a[max_r][col]) < eps)return 0;
        if (k != max_r)
        {
            for (j = col; j < var; j++)
                swap(a[k][j], a[max_r][j]);
            swap(x[k], x[max_r]);
        }
        x[k] /= a[k][col];
        for (j = col + 1; j < var; j++)a[k][j] /= a[k][col];
        a[k][col] = 1;
        for (int i = 0; i < equ; i++)
```

```
            if (i != k)
            {
                x[i] -=  x[k] * a[i][k];
                for (j = col + 1; j < var; j++)
                    a[i][j] -= a[k][j] * a[i][col];
                a[i][col] = 0;
            }
    }
    return 1;
}
```

简化版(判是否有解，变上三角矩阵)

```
int gauss(int n)
{
    int i, j, k, x, y;
    for (i = j = 0; i <= n && j < N; i ++, j ++)
    {
        if (mat[i][j] == 0)
        {
            for (x = i + 1; x <= n; x ++)
                if (mat[x][j])
                {
                    for (y = j; y <= N; y ++)
                        swap(mat[i][y], mat[x][y]);
                    break;
                }
            if (x > n)
            {
                -- i;
                continue ;
            }
        }
        for (x = i + 1; x <= n; x ++)
            if (mat[x][j])
            {
                for (y = j; y <= N; y ++)
                    mat[x][y] ^= mat[i][y];
            }
    }
    for (k = i; k <= n; k ++)
        if (mat[k][N])
            return 0;
    return 1;
}
```

JAVA版

```
import java.util.*;
import java.math.*;

class fraction {
    BigInteger a, b;
    public fraction() {
        a = new BigInteger("0");
        b = new BigInteger("1");
    }

    fraction( BigInteger a0, BigInteger b0) {
        this.a = a0; this.b = b0;
    }
    void reduction() {
        BigInteger tmp = a.gcd( b );
        a = a.divide( tmp );
        b = b.divide( tmp );
        if ( b.compareTo( BigInteger.ZERO ) == - 1 ) {
            b = b.multiply( BigInteger.valueOf( -1 ));
            a = a.multiply( BigInteger.valueOf( -1 ));
        }
    }
    fraction add( fraction t ) {
        fraction tmp = new fraction( a.multiply( t.b ).add( b.multiply( t.a )) , b.multiply(t.b)
        tmp.reduction();
        return tmp;
    }
    fraction sub( fraction t ) {
        fraction tmp = new fraction( a.multiply( t.b ).subtract( b.multiply( t.a )) , b.multiply(
        tmp.reduction();
        return tmp;
    }
    fraction mult( fraction t) {
        fraction tmp = new fraction( a.multiply( t.a ), b.multiply( t.b ));
        tmp.reduction();
        return tmp;
    }
    fraction div( fraction t) {
        fraction tmp = new fraction( a.multiply( t.b ), b.multiply( t.a ));
        tmp.reduction();
        return tmp;
    }
    public void abs() {
```

```
        if ( this.a.compareTo( BigInteger.ZERO ) == - 1 ) {
            this.a = this.a.multiply( BigInteger.valueOf( -1 ));
        }
    }
    void out() {
        this.reduction();
        if ( b.compareTo( BigInteger.ONE ) == 0 )
            System.out.println(a);
        else
            System.out.println(a + "/" + b);
    }

    boolean biger( fraction p ) {
        fraction tmp = new fraction ( a, b );
        fraction t = new fraction(p.a, p.b);
        //t = p;
        tmp.reduction();
        if ( tmp.a.compareTo( BigInteger.ZERO ) == - 1 ) {
            tmp.a = tmp.a.multiply( BigInteger.valueOf( -1 ));
        }
        if ( t.a.compareTo( BigInteger.ZERO ) == - 1 ) {
            t.a = t.a.multiply( BigInteger.valueOf( -1 ));
        }
        tmp = tmp.sub( t );
        return tmp.a.compareTo( BigInteger.ZERO ) > -1;
    }

}

public class Main {

    public static void lup_solve( fraction x[], fraction y[], fraction L[][], fraction U[][], int
        int i, j;
        fraction z = new fraction( BigInteger.ZERO , BigInteger.ONE);
        fraction sum = z;//double sum;
        for ( i = 0 ; i < n ; i ++ ) {
            sum = z; //sum = 0;
            for ( j = 0 ; j < i ; j ++ ) {
                sum = sum.add( L[i][j].mult( y[j] ));//sum += L[i][j] * y[ j ];
            }
            y[i] = b[ pi[i] ].sub( sum );//y[i] = b[ pi[i] ] - sum;
        }
        for ( i = n - 1 ; i >= 0 ; i -- ) {
            sum = z ; //sum = 0;
            for ( j = i + 1 ; j < n ; j ++ ) {
```

```
                sum = sum.add( U[i][j].mult( x[j] ));//sum += U[i][j] * x[ j ];
            }
            x[i] = (y[i].sub( sum )).div( U[i][i] );//x[i] = (y[i] - sum)/U[i][i];
        }
    }

    public static int lup_decomposition( fraction a[][] , int n , int pi[] ) {
        int i, j, k, k1 = 0 ;
        fraction p = new fraction(BigInteger.valueOf(0), BigInteger.ONE ), z = new fraction( BigI
        for ( i = 0 ; i < n ; i ++ )
            pi[i] = i;// 置换

        for ( k = 0 ; k < n ; k ++ ) {
            p = z;
            for ( i = k ; i < n ; i ++ ) {
                if (  a[i][k].biger( p ) ) {
                    p = new fraction( a[i][k].a, a[i][k].b) ;
                    k1 = i;
                }
            }
            if ( p.a.compareTo( BigInteger.ZERO ) == 0 ) {
                return 0 ;// error
            }
            fraction tmp;

            int t = pi[ k ]; pi[ k ] = pi[ k1 ]; pi[k1] = t;
            for ( i = 0 ; i < n ; i ++ ) {
                tmp = a[ k ][i];  a[ k ][i] = a[ k1 ][i];  a[k1][i] = tmp;
            }//swap( a[k][i], a[k1][i] );
            for ( i = k + 1 ; i < n ; i ++ ) {
                a[i][k] = a[i][k].div( a[k][k] );
                for ( j = k + 1 ; j < n ; j ++ )
                    a[i][j] = a[i][j].sub( a[i][k].mult(a[k][j]));// - a[i][k] * a[k][j] ;
            }
        }
        return 1;
    }

    public static void check(fraction a[][], fraction x[], int n) {
        int i, j;
        fraction sum, z = new fraction( BigInteger.ZERO , BigInteger.ONE);
        for ( i = 0 ;  i < n ; i++ ) {
            sum = z;
            for ( j = 0 ; j < n ; j ++ ) {
                sum = sum.add( a[i][j].mult( x[j] ));
```

```
            }
            sum.out();
        }
    }

    public static void main(String[] agrs) {
        Scanner cin = new Scanner( System.in );
        int i, j;
        int n;
        while ( cin.hasNextInt() ) {
            //任何函数都要和一个class相连
            n = cin.nextInt();
            int pi[] = new int[n];
            fraction a[][] = new fraction[n][n];
            fraction aa[][] = new fraction[n][n];
            fraction B[] = new fraction[n];
            fraction x[] = new fraction[n];
            fraction y[] = new fraction[n];

            for ( i = 0 ; i < n ; i ++ ) {
                for ( j = 0 ; j < n ; j ++ ) {
                    a[i][j] = new fraction( BigInteger.valueOf(0), BigInteger.valueOf(1));
                    a[i][j].a = cin.nextBigInteger();
                    aa[i][j] = new fraction( BigInteger.valueOf(0), BigInteger.valueOf(1));
                    aa[i][j] = a[i][j];
                }
                B[i] = new fraction( BigInteger.valueOf(0), BigInteger.valueOf(1));
                B[i].a = cin.nextBigInteger();
                x[i] = new fraction( BigInteger.valueOf(0), BigInteger.valueOf(1));
                y[i] = new fraction( BigInteger.valueOf(0), BigInteger.valueOf(1));
            }
            if ( 1 == lup_decomposition( a, n, pi) ) {
                lup_solve( x, y, a, a, pi, B, n);
                for ( i = 0 ; i < n; i ++)
                    x[i].out();
                //check( aa, x, n);
            } else {
                System.out.println("No solution.");
            }
            System.out.println();
        }
    }
}
```

§ 6.7 解模线性方程或方程组(未知复杂度)

```
/**
*模线性方程ax=b(mod n): int modular_linear(int a,int b,int n,int* sol)
*输入: a,b,n
*输出: sol[]
*
*模线性方程组x=b[k](mod w[k]): int modular_linear_system(int b[],int w[],int k)
*for 一遍这个函数即可
*输入: b[],w[],k(方程组中的第k 个方程)
*输出: return int;
*/
typedef long long LL;
//扩展Euclid 求解gcd(a,b)=ax+by
int ext_gcd(int a,int b,int& x,int& y)
{
    int t,ret;
    if(!b)
    {
        x=1,y=0;
        return a;
    }
    ret=ext_gcd(b,a%b,x,y);
    t=x,x=y,y=t-a/b*y;
    return ret;
}

//计算m^a, O(loga), 本身没什么用, 注意这个按位处理的方法 :-P
int exponent(int m,int a)
{
    int ret=1;
    for(; a; a>>=1,m*=m) if(a&1) ret*=m;
    return ret;
}

//计算幂取模a^b mod n, O(logb)
int modular_exponent(int a,int b,int n)  //a^b mod n
{
    int ret=1;
    for(; b; b>>=1,a=(int)((LL)a)*a%n) if(b&1) ret=(int)((LL)ret)*a%n;
    return ret;
}

//求解模线性方程ax=b (mod n)
//返回解的个数,解保存在sol[] 中
```

```
//要求n>0, 解的范围0..n-1
int modular_linear(int a,int b,int n,int* sol)
{
    int d,e,x,y,i;
    d=ext_gcd(a,n,x,y);
    if(b%d) return 0;
    e=(x*(b/d)%n+n)%n;
    for(i=0; i<d; i++) sol[i]=(e+i*(n/d))%n;
    return d;
}

//求解模线性方程组(中国余数定理)
//  x = b[0] (mod w[0])
//  x = b[1] (mod w[1])
//  ...
//  x = b[k-1] (mod w[k-1])
//要求w[i]>0,w[i] 与w[j] 互质,解的范围1..n,n=w[0]*w[1]*...*w[k-1]
int modular_linear_system(int b[],int w[],int k)
{
    int d,x,y,a=0,m,n=1,i;
    for (i=0; i<k; i++) n*=w[i];
    for (i=0; i<k; i++)
    {
        m=n/w[i];
        d=ext_gcd(w[i],m,x,y);
        a=(a+y*m*b[i])%n;
    }
    return (a+n)%n;
}
```

§ 6.8 定积分计算

6.8.1 自适应辛普森

```
/*
*定积分运算: 自适应辛普森公式
*输入: a,b,eps,F(x)
*输出: 积分值asr(a, b, eps)
*/
double F(double x)
{
    return log10(x);
}

//三点辛普森公式
```

```
double simpson(double width, double fa, double fb, double fc)
{
    return (fb + fa + 4 * fc) * width / 6;
}

//自适应simpson公式递归过程
double asr(double a, double b, double eps, double A)
{
    double c = (a + b) / 2;
    double fa, fb, fc, L, R;
    fa = F(a); fb = F(b); fc = F(c);
    L = simpson(c - a, fa, fc, F((c + a) / 2));
    R = simpson(b - c, fc, fb, F((b + c) / 2));
    if (fabs(L + R - A) <= 15 * eps) return L + R + (L + R - A) / 15;
    return asr(a, c, eps / 2, L) + asr(c, b, eps / 2, R);
}

//自适应simpson公式主过程
double asr(double a, double b, double eps)
{
    return asr(a, b, eps, simpson(b - a, F(a), F(b), F((b + a) / 2)));
}
```

6.8.2 Romberg

```
/**
Romberg求定积分
输入：积分区间[a,b]，被积函数f(x,y,z)
输出：积分结果

f(x,y,z)示例：
double f0( double x, double l, double t )
{
    return sqrt(1.0+l*l*t*t*cos(t*x)*cos(t*x));
}
*/
const double PI = acos(-1.0);
const int maxn = 1010;

double f0( double x, double l, double t )
{
    return sqrt(1.0+l*l*t*t*cos(t*x)*cos(t*x));
}

double Romberg (double a, double b, double (*f)(double x, double y, double z), double eps,
```

```
                double l, double t)
{
    int i, j, temp2, min;
    double h, R[2][maxn], temp4;

    for (i = 0; i < maxn; i++)
    {
        R[0][i] = 0.0;
        R[1][i] = 0.0;
    }
    h = b - a;
    min = (int)(log(h * 10.0) / log(2.0)); //h should be at most 0.1
    R[0][0] = ((*f)(a, l, t) + (*f)(b, l, t)) * h * 0.50;
    i = 1;
    temp2 = 1;
    while (i < maxn)
    {
        i++;
        R[1][0] = 0.0;
        for (j = 1; j <= temp2; j++)
            R[1][0] += (*f)(a + h * ((double)j - 0.50), l, t);
        R[1][0] = (R[0][0] + h * R[1][0]) * 0.50;
        temp4 = 4.0;
        for (j = 1; j < i; j++)
        {
            R[1][j] = R[1][j - 1] + (R[1][j - 1] - R[0][j - 1]) / (temp4 - 1.0);
            temp4 *= 4.0;
        }
        if ((fabs(R[1][i - 1] - R[0][i - 2]) < eps) && (i > min))
            return R[1][i - 1];
        h *= 0.50;
        temp2 *= 2;
        for (j = 0; j < i; j++)
            R[0][j] = R[1][j];
    }
    return R[1][maxn - 1];
}

double Integral(double a, double b, double (*f)(double x, double y, double z), double eps,
                double l, double t)
{
    int n;
    double R, p, res;

    n = (int)(floor)(b * t * 0.50 / PI);
```

```
    p = 2.0 * PI / t;
    res = b - (double)n * p;
    if (n) R = Romberg (a, p, f0, eps / (double)n, l, t);
    R = R * (double)n + Romberg( 0.0, res, f0, eps, l, t );

    return R / 100.0;
}
```

§ 6.9 多项式求根: 牛顿法

```
/**
*多项式求根: 牛顿法
*a_n*x^n+a_(n-1)*x^(n-1)+...+a_1*x+a_0=0
*输入: 多项式系数c[], 多项式度数n, 求在[a,b]间的根
*输出: Polynomial_Root()(根)
*要求保证[a,b]间有根
*/

double fabs(double x)
{
    return (x < 0) ? -x : x;
}

double f(int m, double c[], double x)
{
    int i;
    double p = c[m];

    for (i = m; i > 0; i--)
        p = p * x + c[i - 1];
    return p;
}

int newton(double x0, double *r,
           double c[], double cp[], int n,
           double a, double b, double eps)
{
    int MAX_ITERATION = 1000;
    int i = 1;
    double x1, x2, fp, eps2 = eps / 10.0;

    x1 = x0;
    while (i < MAX_ITERATION)
    {
        x2 = f(n, c, x1);
```

```
        fp = f(n - 1, cp, x1);
        if ((fabs(fp) < 0.000000001) && (fabs(x2) > 1.0))
            return 0;
        x2 = x1 - x2 / fp;
        if (fabs(x1 - x2) < eps2)
        {
            if (x2 < a || x2 > b)
                return 0;
            *r = x2;
            return 1;
        }
        x1 = x2;
        i++;
    }
    return 0;
}

double Polynomial_Root(double c[], int n, double a, double b, double eps)
{
    double *cp;
    int i;
    double root;

    cp = (double *)calloc(n, sizeof(double));
    for (i = n - 1; i >= 0; i--) cp[i] = (i + 1) * c[i + 1];
    if (a > b) root = a; a = b; b = root;
    if ((!newton(a, &root, c, cp, n, a, b, eps)) &&
            (!newton(b, &root, c, cp, n, a, b, eps)))
        newton((a + b) * 0.5, &root, c, cp, n, a, b, eps);
    free(cp);
    if (fabs(root) < eps) return fabs(root);
    else return root;
}
```

§ 6.10 周期性方程: 追赶法

```
/*  追赶法解周期性方程
    周期性方程定义: | a1 b1 c1 ...                |          =  x1
                  |    a2 b2 c2 ...             |          =  x2
                  |       ...                   | *   X    =  ...
                  | cn-1 ...         an-1 bn-1  |          =  xn-1
                  | bn   cn               an    |          =  xn
    输入: a[],b[],c[],x[]
    输出: 求解结果X在x[]中
*/
```

```
void run()
{
    c[0] /= b[0]; a[0] /= b[0]; x[0] /= b[0];
    for (int i = 1; i < N - 1; i ++)
    {
        double temp = b[i] - a[i] * c[i - 1];
        c[i] /= temp;
        x[i] = (x[i] - a[i] * x[i - 1]) / temp;
        a[i] = -a[i] * a[i - 1] / temp;
    }
    a[N - 2] = -a[N - 2] - c[N - 2];
    for (int i = N - 3; i >= 0; i --)
    {
        a[i] = -a[i] - c[i] * a[i + 1];
        x[i] -= c[i] * x[i + 1];
    }
    x[N - 1] -= (c[N - 1] * x[0] + a[N - 1] * x[N - 2]);
    x[N - 1] /= (c[N - 1] * a[0] + a[N - 1] * a[N - 2] + b[N - 1]);
    for (int i = N - 2; i >= 0; i --)
        x[i] += a[i] * x[N - 1];
}
```

§ 6.11 线性规划(单纯形法)

```
/**
单纯形法解线性规划:
Simplex C(n+m)(n)
maximize:
    c[1]*x[1]+c[2]*x[2]+...+c[n]*x[n]=ans
subject to
    a[1,1]*x[1]+a[1,2]*x[2]+...a[1,n]*x[n] <= rhs[1]
    a[2,1]*x[1]+a[2,2]*x[2]+...a[2,n]*x[n] <= rhs[2]
    ......
    a[m,1]*x[1]+a[m,2]*x[2]+...a[m,n]*x[n] <= rhs[m]
限制:
    传入的矩阵必须是标准形式的, 即目标函数要最大化; 约束不等式均为<= ; xi为非负
数(>=0).
simplex返回参数:
    OPTIMAL         有唯一最优解
    UNBOUNDED       最优值无边界
    FEASIBLE        有可行解
    INFEASIBLE      无解
n为元素个数,m为约束个数
线性规划:
```

```
    max c[]*x;
    a[][]<=rhs[];
ans即为结果,x[]为一组解(最优解or可行解)
*/
const double eps = 1e-8;
const double inf = 0x3f3f3f3f;

#define OPTIMAL -1          //表示有唯一的最优基本可行解
#define UNBOUNDED -2        //表示目标函数的最大值无边界
#define FEASIBLE -3         //表示有可行解
#define INFEASIBLE -4       //表示无解
#define PIVOT_OK 1          //还可以松弛
#define maxn 1000

struct LinearProgramming
{
    int basic[maxn], row[maxn], col[maxn];
    double c0[maxn];

    double dcmp(double x)
    {
        if (x > eps)    return 1;
        else if (x < -eps)   return -1;
        return 0;
    }
    void init(int n, int m, double c[], double a[maxn][maxn], double rhs[],
             double &ans)   //初始化
    {
        for (int i = 0; i <= n + m; i++)
        {
            for (int j = 0; j <= n + m; j++)   a[i][j] = 0;
            basic[i] = 0; row[i] = 0; col[i] = 0;
            c[i] = 0; rhs[i] = 0;
        }
        ans = 0;
    }
    //转轴操作
    int Pivot(int n, int m, double c[], double a[maxn][maxn], double rhs[],
             int &i, int &j)
    {
        double min = inf;
        int k = -1;
        for (j = 0; j <= n; j ++)
            if (!basic[j] && dcmp(c[j]) > 0)
                if (k < 0 || dcmp(c[j] - c[k]) > 0)     k = j;
```

```
    j = k;
    if (k < 0)  return OPTIMAL;
    for (k = -1, i = 1; i <= m; i ++)   if (dcmp(a[i][j]) > 0)
            if (dcmp(rhs[i] / a[i][j] - min) < 0)
            {
                min = rhs[i] / a[i][j];
                k = i;
            }
    i = k;
    if (k < 0)  return UNBOUNDED;
    else    return PIVOT_OK;
}
int PhaseII(int n, int m, double c[], double a[maxn][maxn], double rhs[],
            double &ans, int PivotIndex)
{
    int i, j, k, l; double tmp;
    while (k = Pivot(n, m, c, a, rhs, i, j), k == PIVOT_OK || PivotIndex)
    {
        if (PivotIndex)
        {
            i = PivotIndex;
            j = PivotIndex = 0;
        }
        basic[row[i]] = 0;  col[row[i]] = 0;
        basic[j] = 1;   col[j] = i;     row[i] = j;
        tmp = a[i][j];
        for (k = 0; k <= n; k ++)   a[i][k] /= tmp;
        rhs[i] /= tmp;
        for (k = 1; k <= m; k ++)
            if (k != i && dcmp(a[k][j]))
            {
                tmp = -a[k][j];
                for (l = 0; l <= n; l ++)   a[k][l] +=  tmp * a[i][l];
                rhs[k] += tmp * rhs[i];
            }
        tmp = -c[j];
        for (l = 0; l <= n; l ++)   c[l] += a[i][l] * tmp;
        ans -= tmp * rhs[i];
    }
    return k;
}
int PhaseI(int n, int m, double c[], double a[maxn][maxn], double rhs[],
           double &ans)
{
    int i, j, k = -1;
```

```
    double tmp, min = 0, ans0 = 0;
    for (i = 1; i <= m; i ++)
        if (dcmp(rhs[i] - min) < 0)
        {
            min = rhs[i];
            k = i;
        }
    if (k < 0)  return FEASIBLE;
    for (i = 1; i <= m; i ++)   a[i][0] = -1;
    for (j = 1; j <= n; j ++)   c0[j] = 0;
    c0[0] = -1;
    PhaseII(n, m, c0, a, rhs, ans0, k);
    if (dcmp(ans0) < 0) return INFEASIBLE;
    for (i = 1; i <= m; i ++)   a[i][0] = 0;
    for (j = 1; j <= n; j ++)
        if (dcmp(c[j]) && basic[j])
        {
            tmp = c[j];
            ans += rhs[col[j]] * tmp;
            for (i = 0; i <= n; i ++)   c[i] -= tmp * a[col[j]][i];
        }
    return FEASIBLE;
}
//standard form
//n:原变量个数   m:原约束条件个数
//c:目标函数系数向量-[1~n],c[0] = 0;
//a:约束条件系数矩阵-[1~m][1~n]      rhs:约束条件不等式右边常数列向量-[1~m]
//ans:最优值   x:最优解||可行解向量-[1~n]
int simplex(int n, int m, double c[], double a[maxn][maxn], double rhs[],
            double &ans, double x[])
{
    int i, j, k;
    //标准形式变松弛形式
    for (i = 1; i <= m; i ++)
    {
        for (j = n + 1; j <= n + m; j ++)   a[i][j] = 0;
        a[i][n + i] = 1;  a[i][0] = 0;
        row[i] = n + i;   col[n + i] = i;
    }
    k = PhaseI(n + m, m, c, a, rhs, ans);
    if (k == INFEASIBLE)    return k;
    k = PhaseII(n + m, m, c, a, rhs, ans, 0);
    for (j = 0; j <= n + m; j ++) x[j] = 0;
    for (i = 1; i <= m; i ++)   x[row[i]] = rhs[i];
    return k;
```

```
    }
};   //Primal Simplex
```

§ 6.12 快速傅立叶变换

§ 6.13 随机算法

§ 6.14 0/1 分数规划

6.14.1 Theorm

0/1 分数规划就是给定两个数组，$a[i]$ 表示选取i 的收益，$b[i]$ 表示选取i的代价。如果选取i，定义$x[i] = 1$ 否则$x[i] = 0$。每一个物品只有选或者不选两种方案，求一个选择方案使得$R = \frac{\sum(a[i]*x[i])}{\sum(b[i]*x[i])}$ 取得最值，即所有选择物品的(总收益/总代价)的值最大或是最小。

变形：$F(R) = \sum(a[i] * x[i]) - R \times \sum(b[i] * x[i])$，即求$R$最大值，

函数$F(R)$有这样的一个性质：在前一段L 中可以找到一组对应的X使得$F(R) > 0$，这就提供了一种证据，即有一个比现在的R 更优的解，而在某个R 值使，存在一组解使得$F(R) = 0$,且其他的$F(R) < 0$，这时的R 无法继续增大，即这个R就是我们期望的最优解，之后的R会使得无论哪种方案都会造成$F(R) < 0$. 而我们已经知道，$F(R) < 0$是没有任何意义的，因为此时的R 值根本取不到。

方法：二分法求R最大值或者Dinkelbach算法（比二分快1倍）

6.14.2 Dinkelbach算法($O(\lg(nM))$)

```
/**
*0/1分数规划: Dinkelbach 算法($O(\lg(nM))$)
*输入: a[](收益),b[](费用)
*输出: dinkelbach()(最优比例)
*/
const int maxn=0;
const double INF=1e30;
const double eps=1e-6;
double a[maxn],b[maxn],d[maxn];
double dinkelbach()
{
    double L=任意值;
    double ans;
    do
    {
        ans=L;
        for(int i=0; i<n; i++) d[i]=a[i]-L*b[i];
        double p=0,q=0;
```

```
        for(i=0; i<n; i++)
        {
            if(元素I在解中)
            {
                p+=a[i];
                q+=b[i];
            }
        }
        L=p/q;
    }
    while(fabs(ans-L)>=eps);
    return ans;
}
```

6.14.3 二分法

同普通二分，略。

§ 6.15 迭代逼近

§ 6.16 矩阵快速幂($O(\lg n)$)

```
/**
*矩阵快速幂($O(\lg n)$)
*注意：矩阵不易过大
*输入：t(矩阵),n(矩阵规模),power(幂),mod(模)
*输出：mtx_pow(t,power)
*/
struct mtx
{
    int x[100][100];
};
int n;
const int mod=0;
mtx mtx_mul(mtx &aa, mtx &bb)
{
    mtx s;
    int i, j, k;
    for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++)
        {
            s.x[i][j] = 0;
            for (k = 1; k <= n; k++)
                s.x[i][j] = (s.x[i][j]+ aa.x[i][k] * bb.x[k][j]) % mod;
        }
```

```
    return s;
}
mtx mtx_pow(mtx t, int power)
{
    mtx res;
    int i, j;
    for (i = 1; i <= n; i++)
    {
        for (j = 1; j <= n; j++) res.x[i][j] = 0;
        res.x[i][i] = 1;
    }
    while (power != 0)
    {
        if ((power & 1) == 1) res = mtx_mul(t, res);
        power >>= 1;
        t = mtx_mul(t, t);
    }
    return res;
}
```

Chapter 7

Combinatorics

§ 7.1 排列数

§ 7.2 组合数

7.2.1 打印组合数表

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

```
const int maxn = 0;
int c[maxn][maxn];
void gen_comb(int n)
{
    for (int i = 0; i <= n; i++)
        for (int j = 0; j <= n; j++)
            c[i][j] = 0;
    c[0][0] = 1;
    for (int i = 0; i <= n; i++) c[i][0] = 1;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= i; j++)
            c[i][j] = c[i - 1][j - 1] + c[i - 1][j];
}
```

7.2.2 Theorm

一些关于组合数的定理：

1. $C(n,k)$ 的奇偶性判断：如果$(n\&k) == k$ 则$C(n,k)$ 为奇数，否则为偶数。
2. 杨辉三角从首行(首行行号为0) 到第n 次全为奇数时，行号为$2^n - 1$
3. 杨辉三角中从首行到第n 次出现全行为奇数时，所有的偶数的个数为$2^{2n-1} + 2^{n-1} - 3^n$，所有的奇数的个数为$3^n - 1$

§ 7.3 鸽笼原理和Ramsey 定理

7.3.1 鸽笼原理

基本

n+1 个物体放入n 个盒子，则至少有一个盒子有2 个物体。

加强

令$q_1, q_2, \cdots, q_n$ 为正整数。如果将$q_1 + q_2 + \cdots + q_n - n + 1$ 放入n 个盒子，那么或者第1 个盒子至少有q_1 个物体，或者第2 个盒子至少有q_2 个物体$\cdots\cdots$，或者第n 个盒子至少有q_n 个物体。

平均定理

如果n 个非负整数$m_1, m_2, \cdots, m_n$ 平均数至少等于r，则这n 个整数$m_1, m_2, \cdots, m_n$ 至少有一个满足$m_i \geqslant r$。

7.3.2 Ramsey 定理

定理

如果$m \geqslant 2, n \geqslant 2$ 是两个整数，则存在一个正整数p 使得如果给完全图K_p的顶点着红色或者蓝色，则一定存在红色的K_m 子图或者蓝色的K_m 子图。

Ramsey数

最小的满足条件的p 记为r(m,n)，即Ramsey数。

Ramsey 数表

§ 7.4 常用的公式和数

7.4.1 Sum of Reciprocal(1/n) Approximation

$$\sum_{i=1}^{n} \frac{1}{i} = \ln(n+1) + r(n \to \infty)$$
$$r = 0.57721566490153286060651209\cdots$$

7.4.2 生成Gray码

```
//生成reflected gray code
//每次调用gray取得下一个码
//000...000是第一个码,100...000是最后一个码
void gray(int n, int *code)
{
    int t = 0, i;
    for (i = 0; i < n; t += code[i++]);
```

```
    if (t & 1) for (n--; !code[n]; n--);
    code[n - 1] = 1 - code[n - 1];
}
```

7.4.3 Catalan Number

Theorm

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!n!}$$

$$C_n = \binom{2n}{n} - \binom{2n}{n+1}$$

$$C_0 = 1\ ,\ C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}$$

$$C_0 = 1\ ,\ C_{n+1} = \frac{2(2n+1)}{n+2} C_n$$

$$C_n \sim \frac{4^n}{n^{3/2}\sqrt{\pi}}$$

★所有的奇Catalan数C_n 都满足$n = 2^k - 1$。所有其他的Catalan数都是偶数。

应用

- C_n表示长度2n的Dyck word 的个数。Dyck word 是一个有n个X 和n 个Y组成的字串，且所有的前缀字串皆满足X 的个数大于等于Y 的个数。
- 将上例的X换成左括号，Y 换成右括号，C_n表示所有包含n组括号的合法运算式的个数。
- C_n表示有n个节点组成不同构二叉树的方案数。
- C_n表示有2n+1个节点组成不同构满二叉树（full binary tree）的方案数。
- C_n表示所有在$n \times n$ 格点中不越过对角线的单调路径的个数。
- C_n表示通过连结顶点而将n + 2边的凸多边形分成三角形的方法个数。
- C_n表示对$1, \cdots, n$ 依序进出栈的置换个数。
- C_n表示集合$1, \cdots, n$ 的不交叉划分的个数. 那么, C_n 永远不大于第n项Bell 数. C_n也表示集合$1, \cdots, 2n$ 的不交叉划分的个数。
- C_n表示用n个长方形填充一个高度为n的阶梯状图形的方法个数。

7.4.4 Lucas Number

$$1, 3, 4, 7, 11, 18, 29, 47, 76, 123...$$

$$L_1 = 1,\ L_2 = 3,\ L_n = L_{n-1} + L_{n-2}$$

$$L_n = (\frac{1+\sqrt{5}}{2})^n + (\frac{1-\sqrt{5}}{2})^n$$

7.4.5 Stirling Number

第一类

第一类Stirling数是有正负的，其绝对值是n个元素的项目分作k个环排列的方法数目，用$s(n,k)$(小写)表示。

$$s(n,0) = 0$$

$$s(1,1) = 1$$

$$s(n+1,k) = s(n,k-1) + ns(n,k)$$

递推关系的说明：考虑第n+1个物品，n+1 可以单独构成一个非空循环排列，这样前n 种物品构成k-1个非空循环排列，方法数为s(n,k-1)；也可以前n种物品构成k个非空循环排列，而第n+1个物品插入第i个物品的左边，这有n*s(n,k)种方法。

常用形态：

- $|s(n,1)| = (n-1)!$
- $s(n,k) = (-1)^{n+k}|s(n,k)|$
- $s(n,n-1) = -C(n,2)$
- $s(n,2) = (-1)^n(n-1)!H_{n-1}$
- $x^{\underline{n}} = x(x-1)\cdots(x-n+1) = \sum_{k=1}^{n} s(n,k)x^k$

第二类

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0
2	0	1	1	0	0	0	0	0	0	0
3	0	2	3	1	0	0	0	0	0	0
4	0	6	11	6	1	0	0	0	0	0
5	0	24	50	35	10	1	0	0	0	0
6	0	120	274	225	85	15	1	0	0	0
7	0	720	1764	1624	735	175	21	1	0	0
8	0	5040	13068	13132	6769	1960	322	28	1	0
9	0	40320	109584	118124	67284	22449	4536	546	36	1

第二类Stirling数是个元素的集定义k 个等价类的方法数目。常用的表示方法有$S(n,k)$(大写),$S_n^{(k)}$。

$$S(n,n) = S(n,1) = 1$$

$$S(n,k) = S(n-1,k-1) + kS(n-1,k)$$

递推关系的说明：考虑第n 个物品，n 可以单独构成一个非空集合，此时前n-1个物品构成k-1个非空的不可辨别的集合，方法数为S(n-1,k-1)；也可以前n-1 种物品构成k个非空的不可辨别的集合，第n个物品放入任意一个中，这样有k*S(n-1,k)种方法。

常用：

- $S(n, n-1) = C(n, 2) = n(n-1)/2$
- $S(n, 2) = 2^{n-1} - 1$
- $S(n, k) = \frac{1}{k!}\sum_{j=1}^{k}(-1)^{k-j}C(k, j)j^n$
- 贝尔数: $B_n = \sum_{k=1}^{n} S(n, k)$

7.4.6 Bell Number

B_n是基数为n的集合的划分方法的数目。

$$B_{n+1} = \sum_{k=0}^{n}\binom{n}{k}B_k$$

$$B_n = \frac{1}{e}\sum_{k=0}^{\inf}\frac{k^n}{k!}$$

$$B_{p+n} \equiv B_n + B_{n+1}(modp)$$

$$B_n = \sum_{k=1}^{n} S(n, k)$$

Triangle.png

贝尔三角形 [编辑]

用以下方法建构一个三角矩阵（形式类似杨辉三角形）：

- 第一行第一项是1（a_{1,1} = 1）
- 对于n>1，第n行第一项等同第n-1行最后一项。（$a_{n,1} = a_{n-1,n-1}$）
- 对于m, n>1，第n行第m项等于它左边和左上方的两个数之和。（$a_{n,m} = a_{n,m-1} + a_{n-1,m-1}$）

结果如下：（OEIS:A011971）

1							
1	2						
2	3	5					
5	7	10	15				
15	20	27	37	52			
52	67	87	114	151	203		
203	255	322	409	523	674	877	
877	1080	1335	1657	2066	2589	3263	4140
		⋮		⋮		⋮	

每行首项是贝尔数。每行之和是第二类Stirling数。

这个三角形称为贝尔三角形、Aitken阵列或Peirce三角形（Bell triangle, Aitken's array, Peirce triangle）。

7.4.7 Farey Sequence(法雷序列)

性质

- $F_n = \{a/b|gcd(a, b) = 1\&\&0 \leq a, b \leq n\}$;
- 除了1级法雷数列外，所有的法雷数列都有奇数个元素，其中居于正中间的那个元素一定是1/2.
- 当n趋于正无穷时，n级法雷数列包含的元素的个数趋于$3/(\Pi * \Pi) * n2 \approx 0.30396355 * n2$.

- n级法雷数列中，若相邻两个元素是a/b 和$c/d(a/b < c/d)$,则这两个数的差为$1/bd$, 这个差的最小值为$1/(n*(n-1))$, 最大值为$1/n$, 在法雷数列的第一个元素(0/1)与其后继以及最后一个元素(1/1)与前驱之间的差取到最大值，而正中间的那个元素1/2 与其前驱和后继元素之间的差取次大值$1/(n*2)$.
- 在法雷数列中，对于任意两个相邻分数，先算出前者的分母乘以后者的分子，再算出前者的分子乘以后者的分母，则这两个乘积一定刚好相差1.

递归

```
#include <iostream>
using namespace std;

void Produce(int a, int b, int c, int d, int n)
{
    if ( b + d > n ) return;
    Produce(a, b, a + c, b + d, n);
    cout << a + c << "/" << b + d << endl;
    Produce(a + c, b + d, c, d, n);
}
int main()
{
    int n;
    cin >> n;
    cout << 0 << "/" << 1 << endl;
    Produce(0, 1, 1, 1, n);
    cout << 1 << "/" << 1 << endl;
}
```

非递归

7.4.8 五角形数定理(Function Partion P)

http://mathworld.wolfram.com/PartitionFunctionP.html

求n被划分成多个整数的方案数

$$P(n) = \sum_{k=1}^{n} (-1)^{k+1} [P(n - \frac{1}{2}k(3k-1)) + P(n - \frac{1}{2}k(3k+1))]$$

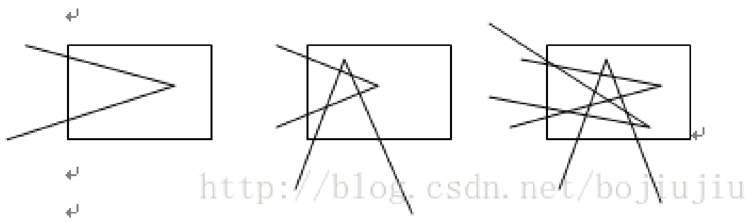
```
const int maxn=100010;
int dp[maxn];
void init()
{
    memset(dp,0,sizeof(dp));
    dp[0]=1;
    for(int i=1; i<=100000; i++)
```

```
    {
        for(int j=1; (i-(3*j*j-j)/2)>=0; j++)
        {
            if(j%2==1) dp[i]+=dp[i-(3*j*j-j)/2];
            else dp[i]-=dp[i-(3*j*j-j)/2];
            dp[i]%=mod;
            if(dp[i]<0) dp[i]+=mod;
            if((i-(3*j*j+j)/2>=0))
            {
                if(j%2==1) dp[i]+=dp[i-(3*j*j+j)/2];
                else dp[i]-=dp[i-(3*j*j+j)/2];
                dp[i]%=mod;
                if(dp[i]<0) dp[i]+=mod;
            }
        }
    }
}
```

7.4.9 折线划分平面

设用n个m折的线划分一个平面，将平面划分为$F(n)$块，则

$$F(0) = 1; \qquad F(n) = F(n-1) + m^2 n - m^2 - 1$$



§ 7.5 Pólya Counting

7.5.1 Burnside引理($O(nsp)$)

适用于单色

共轭类

设其中k阶循环出现的次数为c_k，$k = 1, 2, \cdots, n$,k阶循环出现c_k次，用$(k)^{c_k}$表示。

S_n属于$(1)^{c_1}(2)^{c_2}\cdots(n)^{c_n}$ 的个数为

$$\frac{n!}{c_1!c_2!\cdots c_n!1^{c_1}2^{c_2}\cdots n^{c_n}}$$

k不动置换类

设G是$[1, n]$上的一个置换群。$G \le S_n$. $k \in [1, n]$,G中使k保持不变的置换全体,称为k 不动置换类，记做Z_k.

群G中关于k的不动置换类Z_k 是G 的一个子群。

等价类

一般$[1,n]$上G将$[1,n]$分成若干等价类,满足等价类的3个条件.(a) 自反性;(b) 对称性;(c)传递性。

Burnside 引理

设G是$[1,n]$上的一个置换群，E_k是$[1,n]$在G的作用下包含k的等价类，Z_k是k不动置换类，有$|E_k||Z_k|=|G|$，$k=1,2,\cdots,n$。

设$G=a_1,a_2,\cdots,a_g$ 是目标集$[1,n]$上的置换群。每个置换都写成不相交循环的乘积。G将$[1,n]$ 换分成L 个等价类。$c_1(a_k)$是在置换a_k 的作用下不动点的个数，也就是长度为1的循环的个数。则有：$L=[c_1(a_1)+c_1(a_2)+\cdots+c_1(a_g)]/|G|$

7.5.2 Pólya Counting($O(sp)$)

设$\bar{G}$是n个对象的一个置换群,用m种颜色涂染这n个对象,则不同染色的方案数为

$$L=\frac{1}{\bar{G}}[m^{c(\bar{a_1})}+m^{c(\bar{a_2})}+\cdots+m^{c(\bar{a_g})}]$$

其中$\bar{G}=\bar{a_1},\bar{a_2},\cdots,\bar{a_g}$，$c(\bar{a_k})$为置换$\bar{a_k}$的循环节数。

7.5.3 求置换的循环节

```
/**
*求置换的循环节,polya原理
*输入: perm[0..n-1]为0..n-1的一个置换(排列)
*输出: 返回置换最小周期,num返回循环节个数
*/
const int maxn=0;
int gcd(int a,int b)
{
    if(!a) return b;
    if(!b) return a;
    while(a>b?a%=b:b%=a);
    return a+b;
}
int polya(int* perm,int n,int& num)
{
    int i,j,p,v[maxn],ret=1;
    memset(v,0,sizeof(v));
    for (num=i=0; i<n; i++)
        if (!v[i])
        {
            for (num++,j=0,p=i; !v[p=perm[p]]; j++)
                v[p]=1;
            ret*=j/gcd(ret,j);
        }
    return ret;
```

```
}
```

§ 7.6 生成函数

§ 7.7 离散变换与反演

Chapter 8

Game Theory

§ 8.1 Basic Game

8.1.1 Bash Game

只有一堆n个物品，两个人轮流从这堆物品中取物，规定每次至少取一个，最多取m个。最后取光者得胜。

保持给对手留下$(m+1)$的倍数，就能最后获胜

8.1.2 Wythoff Game

有两堆各若干个物品，两个人轮流从某一堆或同时从两堆中取同样多的物品，规定每次至少取一个，多者不限，最后取光者得胜。

用(a_k, b_k) $(a_k \le b_k, k = 0, 1, 2, \cdots, n)$表示两堆物品的数量并称其为奇异局势，其中$a_k = [k(1+\sqrt{5})/2]$ $b_k = a_k + k$ $(a_k \le b_k, k = 0, 1, 2, \cdots, n)$(方括号表示取整函数)

两个人如果都采用正确操作，那么面对非奇异局势，先拿者必胜；反之，则后拿者取胜。

模板

```
//POJ 1067 取石子游戏
const double CON=(1+sqrt(5))/2;
int main()
{
    int a,b;
    while(~scanf("%d%d",&a,&b))
    {
        if(a>b) swap(a,b);
        if(FLOOR((b-a)*CON)==a) puts("0");
        else puts("1");
    }
    return 0;
}
```

8.1.3 翻硬币游戏

8.1.4 Green Hackenbush

链删边

根节点的$sg =$链的长度。

树删边

在树中，叶子结点的$sg = 0$，其他节点的sg等于儿子结点的$sg + 1$的异或和。

局部连通图删边

- 对于一个单独的偶环，一定是先手必败，其sg=0。
- 对于一个单独的奇环，一定是先手必胜，其sg=1。

环缩点，奇环变成一个节点，偶环直接删掉。

无向图删边

环缩点，

§ 8.2 简单取子游戏

8.2.1 游戏模型

游戏定义

1. 有两个玩家
2. 游戏的操作状态是一个有限的集合（比如：限定大小的棋盘）
3. 游戏双方轮流操作
4. 双方的每次操作必须符合游戏规定
5. 当一方不能将游戏继续进行的时候，游戏结束，同时，对方为获胜方
6. 无论如何操作，游戏总能在有限次操作后结束

必败点和必胜点(P 点& N 点)

- 必败点(P 点)：前一个选手(Previous player) 将取胜的位置称为必败点。
- 必胜点(N 点)：下一个选手(Next player) 将取胜的位置称为必胜点。

必败(必胜)点属性

1. 所有终结点是必败点（P点）
2. 从任何必胜点（N点）操作，至少有一种方法可以进入必败点（P点）
3. 无论如何操作，从必败点（P点）都只能进入必胜点（N点）

算法

1. 将所有终结位置标记为必败点（P点）
2. 将所有一步操作能进入必败点（P点）的位置标记为必胜点（N点）
3. 如果从某个点开始的所有一步操作都只能进入必胜点（N点），则将该点标记为必败点（P点）
4. 如果在步骤3 未能找到新的必败（P点），则算法终止；否则，返回到步骤2

8.2.2 NIM 取石子游戏

NIM-SUM 定义

假设$(x_m, \cdots, x_0)_2$ 和$(y_m, \cdots, y_0)_2$ 的nim-sum 是$(z_m, \cdots, z_0)_2$，则我们表示成$(x_m, \cdots, x_0)_2 \bigoplus (y_m, \cdots, y_0)_2 = (z_m, \cdots, z_0)_2$，这里，$z_k = (x_k + y_k)\%2, (k = 0 \cdots m)$

NIM-SUM 定理

定理一 对于nim 游戏的某个位置$(x_m, \cdots, x_0)$，当且仅当它各部分的nim-sum 等于0 时（即$x_m \bigoplus \cdots \bigoplus x_0 = 0$ 时），则当前位于必败点；否则就是必胜点。

定理二 对于nim 游戏的某个必胜点$(x_m, \cdots, x_0)$，它各部分的nim-sum 的值为可行的操作方案数(待证)

§ 8.3 Sprague-Grundy 理论

8.3.1 游戏定义

基本版

给定一个有向无环图和一个起始顶点上的一枚棋子，两名选手交替的将这枚棋子沿有向边进行移动，无法移动者判负。也就是说：任何一个ICG 都可以通过把每个局面看成一个顶点，对每个局面和它的子局面连一条有向边来抽象成这个“有向图游戏”。

加强版

有向图上并不是只有一枚棋子，而是有n 枚棋子，每次可以任选一颗进行移动

8.3.2 Sprague-Grundy 函数

MEX(minimal excludant) 运算

施加于一个集合的运算，表示最小的不属于这个集合的非负整数。例如mex0,1,2,4=3、mex2,3,5=0、mex=0。

Sprague-Grundy 函数定义

对于一个给定的有向无环图，定义关于图的每个顶点的Sprague-Garundy 函数g 如下：$g(x) = mex\{g(y) \mid y\ is\ the\ successor\ of\ x\}$

Sprague-Grundy 函数特点

1. 所有的terminal position 所对应的顶点，也就是没有出边的顶点，其SG 值为0，因为它的后继集合是空集
2. 对于一个g(x)=0 的顶点x，它的所有后继y 都满足g(y)!=0
3. 对于一个g(x)!=0 的顶点，必定存在一个后继y 满足g(y)=0

8.3.3 游戏模型

基本版

顶点x 所代表的postion 是P-position 当且仅当g(x)=0 （跟P-positioin/N-position 的定义的那三句话是完全对应的）。

加强版

有向图游戏的和(Sum of Graph Games)：设$G_1, G_2, \cdots, G_n$ 是n 个有向图游戏，定义游戏G 是$G_1, G_2, \cdots, G_n$ 的和(Sum)，游戏G 的移动规则是：任选一个子游戏G_i 并移动上面的棋子。Sprague-Grundy Theorem 就是：$g(G) = g(G_1) \bigoplus g(G_2) \bigoplus \cdots \bigoplus g(G_n)$。也就是说，游戏的和的SG 函数值是它的所有子游戏的SG 函数值的异或。

8.3.4 算法和模板

一个求MEX 的模板

```
/**
*ural 1465 Pawn Game
*mex有规律
*1 1 2 0 3 1 1 0 3 3 2 2 4 0 5 2 2 3 3 0 1 1 3 0 2 1 1 0 4 5 2 7 4 0
*1 1 2 0 3 1 1 0 3 3 2 2 4 4 5 5 2 3 3 0 1 1 3 0 2 1 1 0 4 5 3 7 4 8
*
*1 1 2 0 3 1 1 0 3 3 2 2 4 4 5 5 9 3 3 0 1 1 3 0 2 1 1 0 4 5 3 7 4 8
*
*1 1 2 0 3 1 1 0 3 3 2 2 4 4 5 5 9 3 3 0 1 1 3 0 2 1 1 0 4 5 3 7 4 8
*/
#include<cstdio>
#include<iostream>
#include<cstring>
#include<string>
#include<cmath>
#include<set>
using namespace std;
const int maxn=1010;
int dp[maxn];
int mex(int n)//递归地找mex
{
    if(dp[n]!=-1) return dp[n];
```

```
    if(n==0)
    {
        dp[n]=0;
        return dp[n];
    }
    if(n==1||n==2)
    {
        dp[n]=1;
        return dp[n];
    }
    set<int> s;
    s.insert(mex(n-2));
    s.insert(mex(n-3));
    for(int i=0;i<=n-3;i++) s.insert(mex(n-i-3)^mex(i));

    for(int i=0;i<=n;i++)//按mex 定义返回mex()的值
    {
        if(s.find(i)==s.end())
        {
            dp[n]=i;
            return dp[n];
        }
    }
}
int main()
{
    memset(dp,-1,sizeof(dp));
    int n;
    scanf("%d",&n);
    int ans;
    if(n<=68) ans=mex(n);//规律
    else ans=mex((n-68)%34+68);// 规律
    if(ans) puts("White");
    else puts("Black");
    return 0;
}
```

Chapter 9

Number Theory(更多详见附录郭思瑶模板)

更多详见附录郭思瑶模板

§ 9.1 GCD

9.1.1 Theorm

一些GCD 相关的定理

1. 如果$a \equiv r(mod\ b)$，则GCD(a,b)=GCD(b,r)。
2. 欧几里德定理：$GCD(a,b) = GCD(b, a \mod b)$ (a¿b 且a mod b 不为0)
3. 拓展欧几里德定理：对于不完全为0 的非负整数a，b，必然存在整数对x，y，使得ax+by=gcd(a,b)。

9.1.2 欧几里得算法

思想

1. 整数a,b, 假设a¿b
2. 若b=0, 则GCD(a,b)=—a—; 否则GCD(a,b)=GCD(b,a%b)

递归($O(\lg n)$)

```
typedef long long LL;
LL gcd(LL a,LL b)
{
    return b==0? a:gcd(b,a%b);
}
```

```
/**
*二进制算法.避免了模运算.大整数时效率较高.($O(\lg n)$)
*/
```

```
LL gcd(LL a,LL b)
{
    if(!a) return b;
    if(!b) return a;
    if(!(a&1)&&!(b&1)) return gcd(a>>1,b>>1)<<1;
    else if(!(b&1)) return gcd(a,b>>1);
    else if(!(a&1)) return gcd(a>>1,b);
    else if(a<b) return gcd(b-a,a);
    else return gcd(a-b,b);
}
```

非递归($O(\lg n)$)

```
LL gcd(LL a,LL b)
{
    if(!a) return b;
    if(!b) return a;
    while(a>b?a%=b:b%=a);
    return a+b;
}
```

9.1.3 拓展欧几里得算法

思想

拓展欧几里德定理：对于不完全为0 的非负整数a，b，必然存在整数对x，y，使得ax+by=gcd(a,b)。

设$a > b$。

1. 显然当b=0,GCD(a,b)=a。此时x=1,y=0;
2. ab!=0 时,

递归

```
typedef long long LL;
void exgcd(LL a,LL b,LL &d,LL &x,LL &y)
{
    if(!b)
    {
        d=a;
        x=1;
        y=0;
    }
    else
    {
        exgcd(b,a%b,d,y,x);
        y-=x*(a/b);
```

```
    }
}
```

非递归

§ 9.2 LCM

§ 9.3 约数

9.3.1 求约数的个数

Theorm

约数个数定理：对于一个大于1 正整数n 可以分解质因数：$n=\prod_{i=1}^{k}P_i^{a_i}=P_1^{a_1}P_2^{a_2}\cdots P_k^{a_k}$，则n 的正约数的个数就是$n=\prod_{i=1}^{k}(a_i+1)=(a_1+1)(a_2+1)\cdots(a_k+1)$。其中$P_1,P_2,P_3,\cdots,P_k$都是n 的质因数；$a_1,a_2,a_3,\cdots,a_k$ 是$P_1,P_2,P_3,\cdots,P_k$ 的指数。

Algorithm

分解质因数法求约数个数

9.3.2 求n的所有约数之和($O(\sqrt{N})$)

思想

数形结合，求在双曲线x*y = n 在第一象限分支中下方的整点的个数。

作直线x=y，于是可以先计算上半部分（含x=y 这条直线）的点数。x=1 的时候有m 个，x=2 的时候有m/2-1 个。。。于是乘以2。然后x=y 这条直线上的i-1 个点多计算了一次，于是要减去(i-1) 个。

时间复杂度

$O(\sqrt{n})$

程序

```
long long int a[10]= {0,1,3,5,8,10};
long long int f(long long m)
{
    if(m <=5) return a[m];
    long long sum = 0;
    long long i;
    for(i = 1; i*i <= m; ++i) sum += m/i - (i - 1);
    return sum*2-i+1;
}
```

§ 9.4 欧拉函数PHI(x)

9.4.1 欧拉函数PHI(x) 定义

欧拉函数PHI(x) 等于不超过x 且和x 互素的整数的个数。

$$\phi(n) = n(1-\frac{1}{p_1})(1-\frac{1}{p_2})\cdots(1-\frac{1}{p_k})$$

9.4.2 计算PHI(n)($O(N)$)

```
/**
*计算PHI(n)
*输入: n
*输出: phi(n)
*/
int euler_phi(int n)
{
    int m=(int)sqrt(n+0.5);
    int ans=n;
    for(int i=2;i<=m;i++)
    {
        if(n%i==0)
        {
            ans=ans/i*(i-1);
            while(n%i==0) n/=i;
        }
    }
    if(n>1) ans=ans/n*(n-1);
    return ans;
}
```

9.4.3 打印PHI(n) 表($O(N\sqrt{N})$)

```
/**
*打印PHI(n) 表
*输入: n
*输出: phi[]
*/
int phi[maxn];
void phi_table(int n)
{
    for(int i=2;i<=n;i++) phi[i]=0;
    phi[1]=1;
    for(int i=2;i<=n;i++)
    {
        if(!phi[i])
```

```
        {
            for(int j=i;j<=n;j+=i)
            {
                if(!phi[j]) phi[j]=j;
                phi[j]=phi[j]/i*(i-1);
            }
        }
    }
}
```

9.4.4 PHI(x) 应用

§ 9.5 打印素数表($O(\sqrt{N})$)

```
/**
*打印素数表
*输入: n
*输出: prime[](素数),num(素数个数)
*/
int vis[maxn],prime[maxn],num;
void sieve(int n)
{
    int m=(int)sqrt(n+0.5);
    memset(vis,0,sizeof(vis));
    for(int i=2; i<=m; i++)
        if(!vis[i])
            for(int j=i*i; j<=n; j+=i) vis[j]=1;
}
void gen_prime(int n)
{
    sieve(n);
    num=0;
    for(int i=2; i<=n; i++) if(!vis[i]) prime[num++]=i;
}
```

§ 9.6 分解质因数($O(\sqrt{N})$)

$$k = \prod_{i=1}^{n} {p_i}^{e_i}$$

9.6.1 分解质因数(无素数表版)

```
/**
*分解质因数(无素数表版)
*输入: k
```

```
*输出: p[](k被分解的质数),ex[](k 被分解的这个质数的次数)
*/
typedef long long LL;
const int maxn=0;
int cnt;
LL p[maxn],ex[maxn];//p[]为k被分解的质数,ex[]为k被分解的质数的次数
void div_prime(LL k)
{
    cnt=0;
    for(LL i=2; i*i<=k; i++)
    {
        if(k%i==0)
        {
            ex[cnt]=0;
            p[cnt]=i;
            while(k%i==0)
            {
                k/=i;
                ex[cnt]++;
            }
            cnt++;
        }
    }
    if(k>1)
    {
        p[cnt]=k;
        ex[cnt++]=1;
    }
}
```

9.6.2 分解质因数(素数表版,适合批量)

```
/**
*分解质因数(素数表版)(需先调用init生成素数表)
*输入: k
*输出: p[](k被分解的质数),ex[](k 被分解的这个质数的次数)
*/
typedef long long LL;
const int maxn=0;
int vis[maxn],prime[maxn],num;
void sieve(int n)
{
    int m=(int)sqrt(n+0.5);
    memset(vis,0,sizeof(vis));
    for(int i=2; i<=m; i++)
```

```
        if(!vis[i])
            for(int j=i*i; j<=n; j+=i) vis[j]=1;
}
void gen_prime(int n)
{
    sieve(n);
    num=0;
    for(int i=2; i<=n; i++) if(!vis[i]) prime[num++]=i;
}
void init()
{
    gen_prime(maxn);
}

int cnt;
LL p[maxn],ex[maxn];//p[]为k被分解的质数,ex[]为k被分解的质数的次数
void div_prime(LL k)
{
    cnt=0;
    for(int i=0; i<num; i++)
    {
        if(prime[i]*prime[i]>k) break;
        if(k%prime[i]==0)
        {
            ex[cnt]=0;
            p[cnt]=prime[i];
            while(k%prime[i]==0)
            {
                k/=prime[i];
                ex[cnt]++;
            }
            cnt++;
        }
    }
    if(k>1)
    {
        p[cnt]=k;
        ex[cnt++]=1;
    }
}
```

§ 9.7 求n!被p(素数)整除的p 的个数

$$ANS = \lfloor\frac{n}{p}\rfloor + \lfloor\frac{n}{p^2}\rfloor + \lfloor\frac{n}{p^3}\rfloor + \cdots$$

§ 9.8 Primitive Root || 费马小定理

9.8.1 Theorm

费马小定理 假如p是质数，且$(a,p)=1$，那么$a^{(p-1)}\equiv 1(mod\ p)$。即：假如p是质数，且$a,p$互质，那么$a^{(p-1)}$除以p的余数恒等于1

Primitive Root(原根) 设m是正整数，a是整数，若$a\ mod\ m$的阶等于$\varphi(m)$，则称a为$mod\ m$的一个原根（其中$\varphi(m)$表示m的欧拉函数）

9.8.2 求Primitive Root(未知复杂度)

求原根目前的做法只能是从2开始枚举，然后暴力判断$g^{(P-1)}=1(mod\ P)$是否当且当指数为P-1的时候成立，而由于原根一般都不大，所以可以暴力得到.

```
/**
*求原根(未知复杂度)
*输入: p
*输出: g(原根)
*/
LL modexp(LL a,LL b,LL n)
{
    LL ret=1;
    LL tmp=a;
    while(b)
    {
       //基数存在
       if(b&0x1) ret=ret*tmp%n;
       tmp=tmp*tmp%n;
       b>>=1;
    }
    return ret;
}
LL primitive_root(LL p)
{
    vector<LL> temp;
    LL x=p-1;
    for(LL i=2;i<=x/i;i++)
    {
        if(x%i==0)
        {
            temp.push_back(i);
            while(x%i==0) x/=i;
        }
    }
    if(x!=1) temp.push_back(x);
```

```
    for(LL g=1;;g++)
    {
        int flag=1;
        for(LL i=0;i<temp.size();i++)
            if(modexp(g,(p-1)/temp[i],p)==1)
                flag=0;
        if(flag) return g;
    }
}
```

§ 9.9 同余乘：避免乘法溢出(未知复杂度)

```
/**
*同余乘：避免乘法溢出(未知复杂度)
*二进制思想
*输入：a(乘数),b(乘数),n(模)
*输出:
*/
typedef long long LL;
typedef const long long CLL;
LL mul_mod(LL a,LL b,CLL &n)
{
    LL ans(0),tmp((a%n+n)%n);
    b=(b%n+n)%n; //b%=n;
    while(b)
    {
        if(b&1) if((ans+=tmp)>=n) ans-=n;
        if((tmp<<=1)>=n) tmp-=n;
        b>>=1;
    }
    return ans;
}
```

Chapter 10

Computational Geometry(详见附录詹钰模板)

详见附录詹钰模板

§ 10.1 二维几何定义

10.1.1 常量

```
typedef double db;
const double eps = 1e-7;
const double PI = acos(-1.0);
const double INF = 1e50;

const int POLYGON_MAX_POINT = 1024;

db dmin(db a, db b){ return a>b?b:a; }
db dmax(db a, db b){ return a>b?a:b; }
int sgn(db a){ return a<-eps? -1 : a>eps; } //返回double型的符号
```

10.1.2 常用

```
//叉乘
double det2(double x1, double y1, double x2, double y2)
{
    return x1*y2 - x2*y1;
}
//ab X bc
db cross(Point2D a, Point2D b, Point2D c)
{
    return (b-a)*(c-b);
}
//ab.*bc
```

```
db dot(Point2D a, Point2D b, Point2D c)
{
    return (b-a)^(c-b);
}
```

10.1.3 点(Point2D)

```
struct Point2D
{
    db x, y;
    int id;
    Point2D(db _x = 0, db _y = 0): x(_x), y(_y) {}
    void input()
    {
        scanf("%lf%lf" , &x, &y);
    }
    void output()
    {
        printf("%.2f %.2f\n" , x, y);
    }
    //
    db len2()
    {
        return x * x + y * y;
    }
    //到原点距离
    double len()
    {
        return sqrt(len2());
    }
    //逆时针转90度
    Point2D rotLeft90()
    {
        return Point2D(-y, x);
    }
    //顺时针转90度
    Point2D rotRight90()
    {
        return Point2D(y, -x);
    }
    //绕原点逆时针旋转arc_u
    Point2D rot(double arc_u)
    {
        return Point2D( x * cos(arc_u) - y * sin(arc_u),
                        x * sin(arc_u) + y * cos(arc_u) );
```

```
}
//绕某点逆时针旋转arc_u
Point2D rotByPoint(Point2D &center, db arc_u)
{
    Point2D tmp( x - center.x, y - center.y );
    Point2D ans = tmp.rot(arc_u);
    ans = ans + center;
    return ans;
}

bool operator == (const Point2D &t) const
{
    return sgn(x - t.x) == 0 && sgn(y - t.y) == 0;
}
bool operator < (const Point2D &t) const
{
    if ( sgn(x - t.x) == 0 ) return y < t.y;
    else return x < t.x;
}
Point2D operator + (const Point2D &t) const
{
    return Point2D(x + t.x, y + t.y);
}

Point2D operator - (const Point2D &t) const
{
    return Point2D(x - t.x, y - t.y);
}

Point2D operator * (const db &t)const
{
    return Point2D( t * x, t * y );
}

Point2D operator / (const db &t) const
{
    return Point2D( x / t, y / t );
}
//点乘
db operator ^ (const Point2D &t) const
{
    return x * t.x + y * t.y;
}
//叉乘
db operator * (const Point2D &t) const
```

```
    {
        return x * t.y - y * t.x;
    }
    //两点之间的角度(-PI , PI]
    double rotArc(Point2D &t)
    {
        double perp_product = rotLeft90()^t;
        double dot_product = (*this)^t;
        if (sgn(perp_product) == 0 && sgn(dot_product) == -1) return PI;
        return sgn(perp_product) * acos(dot_product / len() / t.len() );
    }
    //标准化
    Point2D normalize()
    {
        return Point2D(x / len(), y / len());
    }
};
```

10.1.4 线段(Segment2D)

```
struct Segment2D
{
    Point2D s , e;
    Segment2D() {}
    Segment2D( Point2D _s, Point2D _e ):s(_s), e(_e) {}
};
```

10.1.5 直线(Line2D)

```
//ax + by + c = 0
struct Line2D
{
    db a, b, c;
    Line2D() {}
    Line2D(db _k, db _b):a(_k),b(-1),c(_b) {}
    Line2D(db _a, db _b, db _c):a(_a),b(_b),c(_c) {}
    Line2D(Point2D p1, Point2D p2)
    {
        SetLine(p1, p2);
    }
    //用两点定直线
    void SetLine(Point2D p1, Point2D p2)
    {
        a = p2.y - p1.y;
        b = p1.x - p2.x;
        c = p2.x * p1.y - p1.x * p2.y;
```

```
    }
    //dirx , diry为直线的方向向量
    void SetLine(Point2D p1, db dirx, db diry)
    {
        a = diry;
        b = -dirx;
        c = -a*p1.x-b*p1.y;
    }
    //用线段定直线
    void SetLine(Segment2D seg)
    {
        SetLine(seg.e, seg.s);
    }
};
```

10.1.6 圆(Circle)

```
struct Circle
{
    Point2D o;
    db r;
    void input()
    {
        o.input();
        scanf("%lf" , &r);
    }
    Circle(Point2D _o=Point2D(0,0) , db _r=1.0):o(_o),r(_r) {}
    //比另外一个圆小
    bool operator<(const Circle& t) const
    {
        return r<t.r;
    }
    //比另外一个圆大
    bool operator>(const Circle& t) const
    {
        return r>t.r;
    }
    //在另外一个圆中
    bool in(Circle t)
    {
        return sgn( (o-t.o).len() + (r - t.r) )<=0;
    }
    //和另外一个圆重合
    bool operator==(const Circle &t) const
    {
```

```
        return o==t.o && sgn(r-t.r)==0;
    }
};
```

10.1.7 多边形(POLYGON)

```
struct POLYGON
{
    Point2D v[POLYGON_MAX_POINT];
    int n;
};
```

10.1.8 凸包(CONVEX2D)

```
struct CONVEX2D
{
    Point2D v[POLYGON_MAX_POINT];
    int n;
};
```

§ 10.2 二维几何基本操作

10.2.1 得到直线上一点(GetPointOnLine)

```
Point2D GetPointOnLine(Line2D l)
{
    if( sgn(l.b)==0 ) return Point2D(-l.c/l.a , 0);
    else return Point2D( 0, -l.c/l.b);
}
```

10.2.2 两点构造直线(MakeLine2D)

```
Line2D MakeLine2D(Point2D p1, Point2D p2)
{
    Line2D l;
    l.a = p2.y - p1.y;
    l.b = p1.x - p2.x;
    l.c = p2.x * p1.y - p1.x * p2.y;
    return l;
}
```

10.2.3 构造两点的中垂线(MidPerpLine2D)

```
Line2D MidPerpLine2D(Point2D p, Point2D q)
{
Point2D mid;  Line2D l;
mid.x = (p.x + q.x) / 2;
```

```
mid.y = (p.y + q.y) / 2;
l.a = q.x - p.x;
l.b = q.y - p.y;
l.c = -(l.a * mid.x + l.b * mid.y);
    return l;
}
```

10.2.4 点是否在线段上(PointOnSeg2D)

```
//在线段上则为1；否则为0
bool PointOnSeg2D(Point2D p, Segment2D ls)
{
    return ( sgn((ls.s-p)*(p-ls.e))==0 ) && ( sgn((ls.s-p)^(p-ls.e))!= -1 );
}
```

10.2.5 判断直线相交(LineLineIntersect2D)

```
//平行和重合不算相交，交点为p
bool LineLineIntersect2D(Line2D l1, Line2D l2, Point2D &p)
{
    db crs = l1.a*l2.b - l2.a*l1.b;
    if(sgn(crs)==0) return false;
    p.x = (l1.b*l2.c - l1.c*l2.b)/crs;
    p.y = (l2.a*l1.c - l1.a*l2.c)/crs;
    return true;
}
```

10.2.6 判断线段相交(SegIntersect2D)

```
/**** 判断线段相交 P是否在线段矩形范围内***/
bool PinSegRange2D(Point2D p, Segment2D seg)
{//db 类型的需要
    if( min(seg.s.x, seg.e.x) <=p.x+eps  &&
        max(seg.s.x, seg.e.x) >=p.x-eps  &&
        min(seg.s.y, seg.e.y) <=p.y+eps  &&
        max(seg.s.y, seg.e.y) >=p.y-eps ) return true;
    else return false;
}

/**** 判断线段相交 ***/
//有公共点即算相交
bool SegIntersect2D(Segment2D a, Segment2D b)
{
    int d1 = sgn((a.e - a.s)*(b.s - a.s));
    int d2 = sgn((a.e - a.s)*(b.e - a.s));
    int d3 = sgn((b.e - b.s)*(a.s - b.s));
```

```
    int d4 = sgn((b.e - b.s)*(a.e - b.s));
    if(d1*d2<0 && d3*d4<0) return true;
    else if( (d1==0)&&PinSegRange2D(b.s, a) ) return true;
    else if( (d2==0)&&PinSegRange2D(b.e, a) ) return true;
    else if( (d3==0)&&PinSegRange2D(a.s, b) ) return true;
    else if( (d4==0)&&PinSegRange2D(a.e, b) ) return true;
    return false;
}

/**** 判断线段相交 并返回交点***/
bool SegIntersect2D(Segment2D a, Segment2D b, Point2D& p)
{
    int d1 = sgn((a.e - a.s)*(b.s - a.s));
    int d2 = sgn((a.e - a.s)*(b.e - a.s));
    int d3 = sgn((b.e - b.s)*(a.s - b.s));
    int d4 = sgn((b.e - b.s)*(a.e - b.s));
    if(d1*d2<0 && d3*d4<0) {
        Line2D l1,l2;
        l1.SetLine(a), l2.SetLine(b);
        LineLineIntersect2D(l1, l2, p);
        return true;
    }
    else if( (d1==0)&&PinSegRange2D(b.s, a) ) {p=b.s; return true;}
    else if( (d2==0)&&PinSegRange2D(b.e, a) ) {p=b.e; return true;}
    else if( (d3==0)&&PinSegRange2D(a.s, b) ) {p=a.s; return true;}
    else if( (d4==0)&&PinSegRange2D(a.e, b) ) {p=a.e; return true;}
    return false;
}
```

10.2.7 三点夹角(GetAnglePoint2D)

```
double GetAnglePoint2D(Point2D a, Point2D b, Point2D c)
{
    double dot = (b-a)^(c-a);
    double len = ((b-a).len())*((c-a).len());
    return acos(dot/len);
}
```

10.2.8 点关于直线的对称点(PointSymmetryOnLine2D)

```
//center为对称的中心点
Point2D PointSymmetryOnLine2D(Point2D p, Line2D l, Point2D &center)
{
    Line2D perpLine;
    perpLine.SetLine(p, l.a, l.b);
    LineLineIntersect2D(l, perpLine, center);
```

```
    return (center*2) - p;
}
```

10.2.9 反射方向(ReflectDir)

```
/***已知光线起点，入射点和法向量，求出射方向***/
//inSp光线起点，touch入射点，normal法向量
Point2D ReflectDir(Point2D inSp, Point2D touch, Line2D normal)
{
    Point2D tmp;
    Point2D ref = PointSymmetryOnLine2D(inSp, normal, tmp);
    return ref - touch;
}
```

10.2.10 点到直线距离(PointToLine2D)

```
double PointToLine2D(Point2D p, Line2D l)
{
    return fabs(l.a*p.x + l.b*p.y + l.c)/sqrt(l.a*l.a+l.b*l.b);
}
```

10.2.11 点到线段距离(PointToSegment2D)

```
double PointToSegment2D(Point2D p, Segment2D seg)
{
    if( ((p-seg.s)^(seg.e-seg.s)) < eps ) return (p-seg.s).len();
    if( ((p-seg.e)^(seg.s-seg.e)) < eps ) return (p-seg.e).len();
    return fabs((p-seg.s)*(seg.s-seg.e)) / (seg.s-seg.e).len();
}
```

10.2.12 点在直线上的投影(GetLineProjection)

```
/***点在直线上的投影***/
Point GetLineProjection(const Point &P, const Point &A, const Point B)
{
    Vector v = B - A;
    return A + v * (Dot(v, P - A) / Dot(v, v));
}
```

10.2.13 线段与线段距离(SegmentToSegment2D)

```
/****线段与线段最短距离*****/
double SegmentToSegment2D(Segment2D s1, Segment2D s2)
{
   if(SegIntersect2D(s1, s2)) return 0;
   double d1 = min( PointToSegment2D(s1.s, s2), PointToSegment2D(s1.e, s2) );
   double d2 = min( PointToSegment2D(s2.s, s1), PointToSegment2D(s2.e, s1) );
```

```
   return min(d1, d2);
}

/****线段与线段最短距离 并返回最近点对 Untested*****/
//p1 与 p2是一对最短的点，不唯一
double SegmentToSegment2D(Segment2D s1, Segment2D s2, Point2D &p1, Point2D &p2)
{
    if(SegIntersect2D(s1, s2, p1))
    {
        p2 = p1;
        return 0;
    }
    double minDis = 1e20; Point2D p3, p4;
    double ds1 = PointToSegment2D(s1.s, s2, p3);
    double de1 = PointToSegment2D(s1.e ,s2, p4);
    if(ds1 < de1 ){ p2 = p3;  p1 = s1.s; minDis = ds1;}
    else { p2 = p4; p1 = s1.e; minDis = de1; }

    double ds2 = PointToSegment2D(s2.s, s1, p3);
    double de2 = PointToSegment2D(s2.e, s1, p4);
    if(minDis > ds2){ p2 = p3; p1 = s2.s; minDis = ds2; }
    if(minDis > de2){ p2 = p4; p1 = s2.e; minDis = de2; }
    return minDis;
}
```

10.2.14 点到线段最短距离(PointToSegment2D)

```
/***点到线段最短距离 返回最近点 Need LineLineIntersect2D untested***/
//平行和重合不算相交，交点为p
bool LineLineIntersect2D(Line2D l1, Line2D l2, Point2D &p)
{
    db crs = l1.a*l2.b - l2.a*l1.b;
    if(sgn(crs)==0) return false;
    p.x = (l1.b*l2.c - l1.c*l2.b)/crs;
    p.y = (l2.a*l1.c - l1.a*l2.c)/crs;
    return true;
}
//在线段上则为1；否则为0
bool PointOnSeg2D(Point2D p, Segment2D ls)
{
    return ( sgn((ls.s-p)*(p-ls.e))==0 ) && ( sgn((ls.s-p)^(p-ls.e))!= -1 );
}
//minP为线段上离p最近的点
double PointToSegment2D(Point2D p, Segment2D seg, Point2D &minP)
```

```
{
    Line2D segLine(seg.s, seg.e);
    Line2D perpLine;
    perpLine.SetLine(p, segLine.a, segLine.b);
    LineLineIntersect2D(segLine, perpLine, minP);
    if(PointOnSeg2D(minP, seg))
    {
        return (minP - p).len();
    }else
    {
        double diss = (p-seg.s).len();
        double dise = (p-seg.e).len();
        if(diss<dise) {minP = seg.s; return diss;}
        else {minP = seg.e; return dise;}
    }
}
```

10.2.15 直线与直线距离(LineToLine2D)

```
/****直线与直线距离*****/
double LineToLine2D(Line2D l1, Line2D l2)
{
    Point2D p;
    if(LineLineIntersect2D(l1, l2, p)) return 0;
    p = GetPointOnLine(l1);
    return PointToLine2D(p , l2);
}
```

10.2.16 直线关于点对称(LineSymmetryOnPoint2D)

```
/***直线关于点对称 Untested***/
Line2D LineSymmetryOnPoint2D(Line2D l, Point2D center)
{
    Point2D A = GetPointOnLine(l);
    A = A*2 - center;
    Line2D la;
    la.SetLine(A, -l.b, l.a);
    return la;
}
```

10.2.17 直线与直线的夹角(LineLineArc)

```
/****直线与直线的夹角 Untested****/
//直线l1 l2的夹角[0,PI/2]
double LineLineArc(Line2D l1, Line2D l2)
{
```

```
    Point2D a = Point2D(l1.b, -l1.a);
    Point2D b = Point2D(l2.b, -l2.a);
    return acos( fabs(a^b)/a.len()/b.len() );
}
```

10.2.18 直线l1逆时针转到l2的角度(LineLineRotArc)

```
//求直线l1逆时针转到l2的角度
double LineLineRotArc(Line2D l1, Line2D l2)
{//[0, PI)
    Point2D a = Point2D(l1.b, -l1.a);
    Point2D b = Point2D(l2.b, -l2.a);
    double arc = a.rotArc(b);
    if(arc<-eps) arc+=PI;
    return arc;
}
```

10.2.19 直线关于直线对称(LineSymmetryOnLine2D)

```
/***直线关于直线对称 Untested****/
//直线l关于centerLine的对称直线
Line2D LineSymmetryOnLine2D(Line2D l, Line2D centerLine)
{
    Point2D centerP;
    if(!LineLineIntersect2D(l, centerLine, centerP))
    {
        centerP = GetPointOnLine(centerLine);
        return LineSymmetryOnPoint2D(l, centerP);
    }
    else
    {
        Point2D dir(-centerLine.b, centerLine.a);
        double arc = LineLineRotArc(l, centerLine);
        dir = dir.rot(arc);
        Line2D _line;
        _line.SetLine(centerP, dir.x, dir.y);
        return _line;
    }
}
```

10.2.20 直线与圆的交点(CircleLineIntersect)

```
//输出p1,p2
//return 0 表示无交点
//return 1 表示一个交点
//return 2 表示两个交点
```

```
int CircleLineIntersect(Circle cir, Line2D l, Point2D&p1, Point2D&p2)
{
    Point2D pcrs;
    PointSymmetryOnLine2D(cir.o , l, pcrs);
    double d = (cir.o-pcrs).len2();
    if( d > cir.r*cir.r + eps) return 0;
    d = sqrt(cir.r*cir.r - d);
    Point2D dir; dir.x = l.b, dir.y = -l.a; dir = dir.normalize();
    p1 = pcrs + (dir*d);
    p2 = pcrs - (dir*d);
    if(p1==p2) return 1;
    else return 2;
}
```

10.2.21 三点非共线直线确定圆(PointMakeCircle2D)

```
//输出cir
bool PointMakeCircle2D(Point2D a, Point2D b, Point2D c, Circle &cir)
{
    //找外心
    Line2D l1 = MidPerpLine2D(a, b);
    Line2D l2 = MidPerpLine2D(a, c);
    if( LineLineIntersect2D(l1, l2, cir.o) )
    {
        cir.r = (cir.o - a).len();
        return true;//有
    }
    else return false;//无
}
```

10.2.22 圆上的点p与弧ab的位置关系(PointOnArc2D)

```
//0: 优弧上
//1: 劣弧上
bool PointOnArc2D(Point2D p, Circle c, Point2D a, Point2D b)
{
    double th1, th2, th;
    th = GetAnglePoint2D(c.o, a, b);
    th1 = GetAnglePoint2D(c.o, a, p);
    th2 = GetAnglePoint2D(c.o, p, b);
    if(th1 + th2 > th + eps) return 0;
    return 1;
}
```

10.2.23 两圆交点(CirCirIntersect)

```
//输出p1,p2
//return 0 表示无交点
//return 1 表示一个交点
//return 2 表示两个交点
int CirCirIntersect(Circle c1, Circle c2, Point2D&p1, Point2D &p2)
{
    Point2D pcrs;
    double d = (c1.o - c2.o).len();
    if(d>c1.r+c2.r+eps || d<fabs(c1.r-c2.r)-eps) return 0;
    if(fabs(d)<eps) return 3 ;//同心圆
    double dt = (sq(c1.r)-sq(c2.r)) / d;
    double d1 = (d+dt)/2; //c1.o距圆心连线与中垂线交点的距离

    Point2D dir = c2.o - c1.o;
    dir = dir.normalize();
    pcrs = c1.o + dir*d1;
    dt = sqrt( sq(c1.r) - sq(d1) );
    dir = dir.rotLeft90();
    p1 = pcrs + dir*dt;
    p2 = pcrs - dir*dt;
    if( sgn((p1-c1.o)*(c2.o - c1.o))<0 ) swap(p1, p2);
    if(p1==p2) return 1;
    else return 2;
}
```

10.2.24 弓形的面积

```
double func(double r, double x)
{// 对2*sqrt(r^2-x^2)的积分公式( -r<=x<=r )
    return x*sqrt(r*r-x*x) + r*r*asin(x/r);
}

double ArchArea(double r, double h)
{//半径为r，高为h的弓形的面积
    return func(r, -r+h) - func(r, -r);
}
```

10.2.25 两圆/圆环的面积交和并(CircleUnionArea, CircleCommonArea)

```
double func(double r, double x)
{// 对2*sqrt(r^2-x^2)的积分公式( -r<=x<=r )
    return x*sqrt(r*r-x*x) + r*r*asin(x/r);
}
```

```
double ArchArea(double r, double h)
{//半径为r，高为h的弓形的面积
    return func(r, -r+h) - func(r, -r);
}

double CircleUnionArea(Circle& c1, Circle& c2)
{//两圆面积并
    double ds = (c1.o-c2.o).len();
    if(c1.r + c2.r < ds-eps) return c1.area() + c2.area();
    if(ds < fabs(c1.r - c2.r)+eps ) return max(c1.area(), c2.area());
    double dk = (sq(c1.r) - sq(c2.r)) / ds;
    double d1,d2; // d1+d2 = ds, d1-d2 = dk;
    d1 = (ds + dk)/2;  d2 = ds - d1;
    return ArchArea(c1.r, c1.r + d1) + ArchArea(c2.r , c2.r + d2);
}

double CircleCommonArea(Circle& c1, Circle& c2)
{//两圆面积交
    return c1.area()+c2.area()-CircleUnionArea(c1, c2);
}

两圆环面积并：（S1,S2为外圆，S11,S22 为内圆）
S1 + S2 - S11 - S22 - S1^S2 + S1^S22 + S2^S11 - S11^S22
```

10.2.26 多圆的面积交和并(CirclesUnionArea,CirclesCommonArea)

```
/**
*多圆的面积交和并
*输入：circle[](圆),n(圆的个数)
*输出：CirclesUnionArea(),CirclesCommonArea()
*/
const int IN=1;
const int OUT=0;
const int maxn=1010;
//需要圆的完整定义，圆与圆的交点 CirCirIntersect
double ArchArea(double r, double ang)
{
    //半径为r，弧度为ang的弧所在的弓形面积
    return r*r/2*(ang - sin(ang));
}

struct Ang
{
    double a;//记录极角 (0 -- 2PI)
    int type;//逆时针的入点或出点
```

```
    bool operator<(const Ang& t)const
    {
        if(sgn(a-t.a) == 0 ) return type< t.type ;
        else return a<t.a;
    }
} ang[maxn*4];

bool kicked[maxn];

/******圆面积求并*********/
void CirclesUnionKick(Circle c[], int &n)
{
    memset(kicked, false , sizeof(kicked));
    for(int i=0; i<n; ++i)
        for(int j=i+1; j<n; ++j)
        {
            if(!kicked[i] && !kicked[j])
            {
                if( c[i].in(c[j]) ) kicked[i] = true;
                else if( c[j].in(c[i])) kicked[j] = true;
            }
        }
    int idx = 0;
    for(int i=0; i<n; ++i)
        if(!kicked[i]) c[idx++] = c[i];
    n = idx;
}

double CirclesUnionArea( Circle c[], int n )
{
    CirclesUnionKick(c, n);
    if(n==1) return c[0].area();
    Point2D tmp, p1, p2;
    double a1,a2, b1, b2;
    double ansArea = 0;

    for(int i=0; i<n; ++i)
    {
        int m = 0;
        for(int j=0; j<n; ++j)
        {
            if(i==j) continue;
            int pn = CirCirIntersect(c[i], c[j], p1, p2);
//保证p1逆时针到p2，覆盖了c[i]的部分
            if(pn == 0) continue;
```

```
            tmp = p1-c[i].o;
            a1 = atan2(tmp.y, tmp.x);
            tmp = p2-c[i].o;
            a2 = atan2(tmp.y, tmp.x);
            if(a1<0) a1 += PI*2;
            if(a2<0) a2 += PI*2;
            if(a2<a1)
            {
                b1 = PI*2;
                b2 = a2, a2 = 0;
                ang[m].a = a1, ang[m].type = IN;
                ++m;
                ang[m].a = b1, ang[m].type = OUT;
                ++m;
                ang[m].a = a2, ang[m].type = IN;
                ++m;
                ang[m].a = b2, ang[m].type = OUT;
                ++m;
            }
            else
            {
                ang[m].a = a1, ang[m].type = IN;
                ++m;
                ang[m].a = a2, ang[m].type = OUT;
                ++m;
            }
        }//for(int j

        sort(ang, ang+m);

        int cov = 0;
        a1 = 0;
        for(int j=0; j<m; ++j)
        {
            if(ang[j].type == IN)
            {
                if(cov == 0)
                {
                    a2 = ang[j].a;
                    if( sgn(a2-a1)> 0)
                    {
                        ansArea += ArchArea(c[i].r, a2 - a1);
                        p1 = c[i].o +  (Point2D(cos(a1), sin(a1)) * c[i].r);
                        p2 = c[i].o +  (Point2D(cos(a2), sin(a2)) * c[i].r);
                        ansArea += (p1*p2)/2;
```

```
                    }
                }
                ++cov;
            }
            else
            {
                --cov;
                if(cov == 0) a1 = ang[j].a;
            }
        }//for(int j
        a2 = PI*2;
        if( sgn(a2-a1)> 0)
        {
            ansArea += ArchArea(c[i].r, a2 - a1);
            p1 = c[i].o +  (Point2D(cos(a1), sin(a1)) * c[i].r);
            p2 = c[i].o +  (Point2D(cos(a2), sin(a2)) * c[i].r);
            ansArea += (p1*p2)/2;
        }
    }//for(int i
    return ansArea;
}


/******圆面积求交*********/
bool CirclesCommonKick(Circle c[], int &n)
{
    memset(kicked, false , sizeof(kicked));
    for(int i=0; i<n; ++i)
        for(int j=i+1; j<n; ++j)
        {
            if( (c[i].o - c[j].o).len() > c[i].r + c[j].r - eps ) return false;
            if(!kicked[i] && !kicked[j])
            {
                if( c[i].in(c[j]) ) kicked[j] = true;
                else if( c[j].in(c[i])) kicked[i] = true;
            }
        }
    int idx = 0;
    for(int i=0; i<n; ++i)
        if(!kicked[i]) c[idx++] = c[i];
    n = idx;
    return true;
}

double CirclesCommonArea( Circle c[], int n )
```

```
{
    if( CirclesCommonKick(c, n) )
    {
        if(n==1) return c[0].area();
        Point2D tmp, p1, p2;
        double a1,a2, b1, b2;
        double ansArea = 0;

        for(int i=0; i<n; ++i)
        {
            int m = 0;
            for(int j=0; j<n; ++j)
            {
                if(i==j) continue;
                int pn = CirCirIntersect(c[i], c[j], p1, p2);
//保证p1逆时针到p2，覆盖了c[i]的部分
                if(pn==0) return 0;
                tmp = p1-c[i].o;
                a1 = atan2(tmp.y, tmp.x);
                tmp = p2-c[i].o;
                a2 = atan2(tmp.y, tmp.x);
                if(a1<0) a1 += PI*2;
                if(a2<0) a2 += PI*2;
                if(a2<a1)
                {
                    b1 = PI*2;
                    b2 = a2, a2 = 0;
                    ang[m].a = a1, ang[m].type = IN;
                    ++m;
                    ang[m].a = b1, ang[m].type = OUT;
                    ++m;
                    ang[m].a = a2, ang[m].type = IN;
                    ++m;
                    ang[m].a = b2, ang[m].type = OUT;
                    ++m;
                }
                else
                {
                    ang[m].a = a1, ang[m].type = IN;
                    ++m;
                    ang[m].a = a2, ang[m].type = OUT;
                    ++m;
                }
            }//for(int j
```

```
            sort(ang, ang+m);

            int cov = 0;
            for(int j=0; j<m; ++j)
            {
                if(ang[j].type == IN)
                {
                    ++cov;
                    if(cov == n-1) a1 = ang[j].a;
                }
                else
                {
                    if(cov == n-1)
                    {
                        a2 = ang[j].a;
                        ansArea += ArchArea(c[i].r, a2 - a1);
                        p1 = c[i].o +  (Point2D(cos(a1), sin(a1)) * c[i].r);
                        p2 = c[i].o +  (Point2D(cos(a2), sin(a2)) * c[i].r);
                        ansArea += (p1*p2)/2;
                    }
                    --cov;
                }//else
            }//for(int j
        }//for(int i
        return ansArea;
    }
    else return 0;
}
```

10.2.27 求圆外一点到圆的两个切点(OutTangentPoint)

```
void OutTangentPoint(Point2D out, Circle cir, Point2D &p1, Point2D &p2)
{//转换成圆圆相交来做
    double d2 = sqrt( (out - cir.o).len2() - sq(cir.r) );
    Circle c;  c.o = out; c.r = d2;
    CirCirIntersect(c, cir, p1, p2);
}
```

10.2.28 多边形有向面积(PolyArea)

```
double PolyArea(Point2D p[], int n )
{// 有向面积，逆时针为正，顺时针为负
    p[n] = p[0];
    double area = 0;
    for(int i=0; i<n; ++i)
    area += p[i]*p[i+1];
```

```
    return area/2;
}
```

10.2.29 多边形重心(PolyCenter)

```
Point2D TriCenter(Point2D& p1, Point2D& p2, Point2D& p3)
{ return (p1+(p2+p3))/3; } //三角形重心

Point2D PolyCenter(Point2D p[], int n)
{
    double area = 0 , wt;
    Point2D center;
    for(int i=1; i<n-1; ++i)
    {
        wt = (p[i]-p[0])*(p[i+1]-p[0]);
        area += wt;
        center = center + TriCenter(p[0], p[i], p[i+1])*wt;
    }
    if( sgn(area) ) return center/area;
    else return (p[0]+p[n-1])/2;//无意义
}
```

10.2.30 点与多边形位置关系(PointInPolygon)

```
//0: 点在多边形外
//1: 点在多边形内
bool PonSegment2D(Point2D p, Segment2D seg)
{
    return sgn((seg.s-p)*(p-seg.e))==0 && sgn((seg.s-p)^(p-seg.e))>=0;
}

/**** 判断线段在内部相交***/
bool SegIntersect2D(Segment2D a, Segment2D b)
{
    //必须在线段内相交
    int d1 = sgn((a.e - a.s)*(b.s - a.s));
    int d2 = sgn((a.e - a.s)*(b.e - a.s));
    int d3 = sgn((b.e - b.s)*(a.s - b.s));
    int d4 = sgn((b.e - b.s)*(a.e - b.s));
    if(d1*d2<0 && d3*d4<0) return true;
    return false;
}
const double INF = 1e10;
int PointInPolygon(Point2D p, Point2D poly[], int n)
{
    Segment2D l, seg;
```

```
    l.s = p;
    l.e = p;
    l.e.x = INF;//作射线
    int i, cnt = 0;
    Point2D p1, p2, p3, p4;
    for(i = 0; i < n; i++)
    {
        seg.s = poly[i], seg.e = poly[(i+1)%n];
        if(PonSegment2D(p, seg)) return 2;//点在多边形上
        p1 = seg.s, p2 = seg.e , p3 = poly[(i+2)%n], p4 = poly[(i+3)%n];
        if(SegIntersect2D(l, seg) ||
                PonSegment2D(p2, l) && sgn((p2-p1)*(p-p1))*sgn((p3-p2)*(p-p2)) > 0 ||
                PonSegment2D(p2, l) && PonSegment2D(p3, l) &&
                sgn((p2-p1)*(p-p1))*sgn((p4-p3)*(p-p3)) > 0 )
            cnt++;
    }
    if(cnt % 2 == 1) return 1;//点在多边形内
    return 0;//点在多边形外
}
```

10.2.31 半平面求交

10.2.32

10.2.33

10.2.34

10.2.35

10.2.36

10.2.37

10.2.38

§ 10.3 二维几何算法

10.3.1 平面最近点对($O(N \lg N)$)

```
/**
*平面最近点对($O(N\lg N)$)
*输入: 接入n个点坐标到A数组
*调用: 先调用ClosestPairInit(A,B,n)
*输出: ans = ClosestPointPair2D(0,n,A,B,C)(C数组是中间变量, 不用管)
*/
const int maxn=0;
int sgn(db a){ return a<-eps? -1 : a>eps; } //返回double型的符号
Point2D A[maxn] , B[maxn] , C[maxn];
bool cmpX(const Point2D& a, const Point2D& b)
{
    return a.x<b.x-eps || (sgn(a.x-b.x)==0 && a.y<b.y-eps);
}
bool cmpY(const Point2D&a, const Point2D& b)
{
    return a.y<b.y-eps || (sgn(a.y-b.y)==0 && a.x<b.x-eps);
}
void ClosestPairInit(Point2D a[], Point2D b[], int n)
{
    sort(a, a+n, cmpX);
    for(int i=0; i<n; ++i)
    {
        a[i].id = i;
        b[i] = a[i];
    }
    sort(b, b+n, cmpY);
}
double ClosestPointPair2D(int l, int r, Point2D a[], Point2D b[], Point2D c[])
{
    int i,j,k;
    double re = INF;
    if(r-l<=3)
    {
        for(i=l; i<r; ++i)
            for(j=i+1; j<r; ++j)
                re = min(re, (a[i]-a[j]).len());
        return re;
```

```
    }
    int mid = (l+r)/2 , p1 = l, p2 = mid;
    for(int i=l; i<r; ++i)
    {
        if(b[i].id < mid)  c[p1++] = b[i];
        else c[p2++] = b[i];
    }
    re = min( re, ClosestPointPair2D(l, mid, a, c, b));
    re = min( re, ClosestPointPair2D(mid, r, a, c, b));
    for(i=l,j=mid,k=l; i<mid && j<r; )//重组b数组
    {
        if(c[i].y>c[j].y+eps) b[k++] = c[j++];
        else b[k++] = c[i++];
    }
    while(i<mid) b[k++] = c[i++];
    while(j<r) b[k++] = c[j++];

    for(i=l, k=l; i<r; ++i)
    {
        if(fabs(b[i].x - a[mid].x) < re - eps) c[k++] = b[i];
    }
    for(i=l; i<k; ++i)
        for(j=i+1; j<k && c[i].y+re>c[j].y; ++j)
        {
            re = min(re, (c[i]-c[j]).len() );
        }
    return re;
}
```

10.3.2 最小圆覆盖点($O(N)$)

```
/**
*最小圆覆盖点($O(N)$)
*输入: p[](点),n(点数)
*输出: MinCircleCover()
*/
Circle MinCircleCover(Point2D p[], int n)
{
    //随机增量法 平均复杂度O(n)
    random_shuffle(p, p+n);
    Circle cir;
    cir.o = p[0];
    cir.r = 0;
    for(int i=1; i<n; ++i)
    {
```

```
        if(! cir.in(p[i]) )
        {
            cir.o = p[i];
            cir.r = 0;
        }
        else continue;

        for(int j=0; j<i; ++j)
        {
            if(cir.in(p[j])) continue;
            else
            {
                cir.o = (p[i] + p[j])/2.0;
                cir.r = (p[i]-p[j]).len() / 2;
            }
            for(int k=0; k<j; ++k)
            {
                //注意三点不能共线
                if(cir.in(p[k]) ) continue;
                else
                {
                    if (sgn((p[i]-p[j])*(p[j]-p[k])))
                        PointMakeCircle2D(p[i], p[j], p[k], cir);
                    else
                    {
                        Point2D tmp;
                        if(  sgn((p[i]-p[j])^(p[j]-p[k])) > 0) tmp = p[i];
                        else tmp = p[j];
                        cir.o = (p[k]+tmp)/2.0;
                        cir.r = (p[k]-tmp).len()/2;
                    }
                }
            }//for(int k
        }//for(int j
    }//for(int i
    return cir;
}
```

§ 10.4 三维几何定义

10.4.1

§ 10.5 三维几何算法

10.5.1

Part III

Data Structure

Chapter 11

基本数据结构

§ 11.1　队列

```
const int maxn = 0;
class Queue
{
public:
    Queue(int n = maxn)
    {
        first = 0;
        last = 0;
        maxSize = n;
    }
    ~Queue() {}
    void push(int item)
    {
        assert(!isfull());
        last = (last + 1) % maxSize;
        element[last] = item;
    }
    int pop()
    {
        assert(!isempty());
        first = (first + 1) % maxSize;
        return element[first];
    }
    int front()
    {
        assert(!isempty());
        return element[(first + 1) % maxSize];
    }
    void clear()
    {
```

```
        first = last = 0;
    }
    bool isempty()
    {
        return first == last;
    }
    bool isfull()
    {
        return (last + 1) % maxSize == first;
    }
    int size()
    {
        return (last - first + maxSize) % maxSize;
    }
private:
    int first, last;
    int element[maxn];
    int maxSize;
};
```

§ 11.2 堆

```
struct node
{
    LL i;
    int num;
    bool operator<(const node &a)const
    {
        return (i < a.i);
    }
}qq[500005<<1];
struct heap
{
    int n;
    void ini()
    {
        n=1;
    }
    void push(node &t)
    {
        n++;
        int now=n-1;
        while(now>1)
        {
            if(t<qq[now>>1])
```

```
            {
                qq[now]=qq[now>>1];
                now>>=1;
            }
            else
            {
                qq[now] = t;
                return ;
            }
        }
        qq[now] = t;
    }
    node top()
    {
        return qq[1];
    }
    void pop()
    {
        node t = qq[--n];
        int tmp = 1,k;
        while((tmp<<1)<n)
        {
            k = tmp<<1;
            if(k+1<n&&qq[k+1]<qq[k]) k = k+1;
            if(qq[k]<t)
            {
                qq[tmp] = qq[k];
                tmp = k;
            }
            else break;
        }
        qq[tmp] = t;
    }
    bool empty()
    {
        return n==1? 1:0;
    }
};
```

§ 11.3 并查集

11.3.1 扩展+异或(la 4487)

```
/*
题目大意:
```

有n(n<=20000)个未知的整数X0,X1,X2...Xn-1，有以下Q个(Q<=40000)操作：
I p v :告诉你Xp=v
I p q v :告诉你Xp Xor Xq=v
Q k p1 p2 … pk : 询问 Xp1 Xor Xp2 .. Xor Xpk， k不大于15。
题目解法：
由于异或运算满足很多特殊的性质，尤其是交换律，传递性，因此可以使用并查集维护这个集合。并查集为树状结构。设fa[x]为x结点的父亲，设ve[x]为x Xor fa[x]的值。
最核心的部分就是实现并查集的扩展。
在并查集进行压缩路径的时候，明显有：fa[x] = fa[pref]，之后就有ve[x] = ve[pref] ^ ve[x]，pref为压缩之前x结点的父亲，通过递归计算可以得到。
合并的时候，假设已知a Xor b = v，且a，b已经压缩路径，得到其父亲结点分别为pa，pb，那么fa[pa]=pb，且ve[a] = ve[a] Xor ve[b] Xor v。
有一个比较麻烦的问题需要解决。对于I p v这种对单个数字的操作需要另外处理。虽然I p v是对单个数字操作，但是可以认为存在一个“虚点”Xn=0，由于任何数与0异或等于自己本身，因此I p v相当于I p n v。注意，合并时必须要使Xn作为父亲，Xp作为Xn的儿子。
最后的难题就是查询了。对于Xp1 Xor Xp2 .. Xor Xpk，分别找出在并查集中位于同一个集合的数字，如果该集合的数字有奇数个，且这个集合里面没有Xn，那么意味着这个集合不能算出来，返回I don’t know。否则将各个集合的异或值再异或起来，可得查询值。
为什么集合的数字有奇数个就不能算出来呢？根据我们并查集ve数组的定义，ve[x]保存的是x与其父亲的异或值。由于经过压缩路径，因此树的深度只有一层。会有两种情况：
需要计算的集合没有树根。假设a,b,c,d,e在同一个集合中，要求a ^ b ^ c ^ d ^ e，且并查集中树根为r，那么通过ve数组，我们知道ve[a] = r ^ a, ve[b] = r ^ b, ve[c] = r ^ c..。把这些数字异或起来，是r xor a ^ r ^ b ^ r ^ c ^ r ^ d ^ r ^ e ^ r。r有奇数个，不能消去。但是如果有偶数个r，也就是待计算的集合有偶数个数字，由于r ^ r = 0，刚好消去，是可以计算出a^b^c^d^e来的。
需要计算的集合含有树根。假设a,b,c,d,r在同一个集合中，要求a ^ b ^ c ^ d ^ r，且并查集中树根为r，那么通过ve数组，我们知道ve[a] = r ^ a, ve[b] = r ^ b, ve[c] = r ^ c..。把这些数字异或起来，是r xor a ^ r ^ b ^ r ^ c ^ r ^ d ^ r。r有偶数个，刚好把需要多的一个r消去了。但是如果有奇数个r，也就是待计算的集合有偶数个数字，r不会被消去，是可以计算出a^b^c^d^r来 的。
但是，如果集合中的数字包含Xn，由于已知Xn=0，因此无论多了一个0还是少了一个0，异或的结果不变，因此可以计算。

```
*/
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
#define N 20005
int n, Q;
int fa[N], ve[N];
void init()
{
    for (int i = 0; i <= n + 1; i++)
        fa[i] = i;
```

```
    memset(ve, 0, sizeof(ve));
}
int find(int a)
{
    int pref = fa[a];
    if (fa[a] == a) return a;
    fa[a] = find(fa[a]);
    ve[a] = ve[pref] ^ ve[a];
    return fa[a];
}
bool uniset(int a, int b, int v)
{
    int pa = find(a), pb = find(b);
    if (pa == pb)
    {
        if ((ve[a] ^ ve[b]) != v) return 0;
    }
    else
    {
        if (pa == n) swap(pa, pb);
        fa[pa] = pb;
        ve[pa] = v ^ ve[a] ^ ve[b];
    }
    return 1;
}
int K, num[20];
int query()
{
    bool g[20] = { 0 };
    int i, j, ans = 0, ff, ret;
    for (i = 0; i < K; i++)
    {
        int sz = 1;
        if (g[i]) continue;
        ff = find(num[i]), ans ^= ve[num[i]];
        g[i] = 1;
        for (j = i + 1; j < K; j++)
        {
            if (g[j]) continue;
            ret = find(num[j]);
            if (ret == ff) g[j] = 1, sz++, ans ^= ve[num[j]];
        }
        if (ff != n && sz % 2 == 1) return -1;
    }
    return ans;
```

```
}
void solve()
{
    int T = 1, i, a, b, v;
    bool ig = 0;
    char s[500];
    init();
    while (Q--)
    {
        scanf("%s ", s);
        if (s[0] == 'Q')
        {
            scanf("%d", &K);
            for (i = 0; i < K; i++)
                scanf("%d", &num[i]);
            if (!ig)
            {
                int ret = query();
                if (ret == -1) printf("I don't know.\n");
                else printf("%d\n", ret);
            }
        }
        else
        {
            int blank = 0;
            gets(s);
            if (ig) continue;
            for (i = 0; i < (int) strlen(s); i++)
                if (s[i] == ' ') blank++;
            if (blank == 1) sscanf(s, "%d%d", &a, &v), b = n;
            else sscanf(s, "%d%d%d", &a, &b, &v);
            if (!uniset(a, b, v))
            {
                printf("The first %d facts are conflicting.\n", T);
                ig = 1;
            }
            T++;
        }
    }
}
int main()
{

    int ca = 1;
    while (scanf("%d%d\n", &n, &Q) && n)
```

```
    {
        printf("Case %d:\n", ca++);
        solve();
        printf("\n");
    }
    return 0;
}
```

11.3.2 元素的删除与计数

```
/**
元素的删除，根节点的计数等（cnt），实现集合的合并，查找等（O（1））
元素的删除：找替身的方法，将要删除的元素映射到新的空间元素（另开一个元素，并且
初始化新开的空间信息），然后通过每一个映射来操作每一个元素，这样的修改并没有改变原来
的
树状信息（比如说节点与父节点的关系等），但是我们每一次通过映射操作间接改变；
*/
#include<stdio.h>
#include<string.h>
#define MAXD 200010
int N, M, p[MAXD], id[MAXD], num[MAXD], cnt;
long long sum[MAXD];
void init()
{
    int i;
    for (i = 1; i <= N; i ++) id[i] = p[i] = sum[i] = i, num[i] = 1; ///id数组是
将i节点映射到id[i]编号节点上
    cnt = N;
}
int find(int x)
{
    return p[x] == x ? x : (p[x] = find(p[x]));
}
void Union(int x, int y)
{
    int tx = find(id[x]), ty = find(id[y]);
    p[tx] = ty, num[ty] += num[tx], sum[ty] += sum[tx];
}
void Delete(int x)
{
    int tx = find(id[x]);
    -- num[tx], sum[tx] -= x;
    id[x] = ++ cnt, p[id[x]] = id[x], num[id[x]] = 1, sum[id[x]] = x;
}
void solve()
```

```
{
    int i, x, y, op;
    for (i = 0; i < M; i ++)
    {
        scanf("%d", &op);
        if (op == 1)
        {
            scanf("%d%d", &x, &y);
            if (find(id[x]) != find(id[y])) Union(x, y);
        }
        else if (op == 2)
        {
            scanf("%d%d", &x, &y);
            if (find(id[x]) != find(id[y])) Delete(x), Union(x, y);
        }
        else
        {
            scanf("%d", &x);
            printf("%d %lld\n", num[find(id[x])], sum[find(id[x])]);
        }
    }
}
int main()
{
    while (scanf("%d%d", &N, &M) == 2)
    {
        init();
        solve();
    }
    return 0;
}
```

11.3.3 权值排序+维护根(长春区域赛E)

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <iostream>
typedef long long LL;
using namespace std;
const int  M = 2e5 + 10;

int fa[M], cnt[M];
LL val[M];
```

```
struct E
{
    int u, v, val;
} edge[M];
int n;
void build()
{
    for (int i = 0; i <= n; i++)
    {
        cnt[i] = 1;
        fa[i] = i;
        val[i] = 0;
    }
}
int find(int x)
{
    if (x == fa[x] )return fa[x];
    return fa[x] = find(fa[x]);
}
bool cmp(const struct E &a, const struct E &b)
{
    return a.val > b.val;
}
int main(void)
{
    while (scanf("%d", &n) != EOF)
    {
        for (int i = 1; i <= n - 1; i++)
            scanf("%d%d%d", &edge[i].u, &edge[i].v, &edge[i].val);
        build();
        sort(edge + 1, edge + n, cmp);
        LL  ans = 0;
        for (int i = 1; i <= n - 1 ; i++)
        {
            int u = edge[i].u, v = edge[i].v, w = edge[i].val;
            int ru = find(u), rv = find(v);
            //            printf("%d %d\n",ru,rv);
            LL  sumu = val[ru] + (LL)w * cnt[rv] , sumv = val[rv] + (LL)w * cnt[ru];
            if (sumu > sumv)
            {
                fa[rv] = u;
                cnt[u] += cnt[v];
                val[u] = sumu;
            }
            else
```

```
            {
                fa[u] = v;
                cnt[v] += cnt[u];
                val[v] = sumv;
            }
            ans = max(ans, max(sumu, sumv));
        }
        printf("%lld\n", ans);
    }
    return 0;
}
```

Chapter 12

高级数据结构

§ 12.1 线段树

12.1.1 单值更新

```
#include <cstdio>

#define lson l , m , rt << 1
#define rson m + 1 , r , rt << 1 | 1

/*单点替换，区间求和*/
const int maxn = 55555;
int sum[maxn<<2];
void PushUP(int rt)
{
    sum[rt] = sum[rt<<1] + sum[rt<<1|1];
}
void build(int l,int r,int rt)
{
    if (l == r)
    {
        scanf("%d",&sum[rt]);
        return ;
    }
    int m = (l + r) >> 1;
    build(lson);
    build(rson);
    PushUP(rt);
}
void update(int p,int add,int l,int r,int rt)
{
    if (l == r)
    {
```

```
        sum[rt] += add;
        return ;
    }
    int m = (l + r) >> 1;
    if (p <= m) update(p , add , lson);
    else update(p , add , rson);
    PushUP(rt);
}
int query(int L,int R,int l,int r,int rt)
{
    if (L <= l && r <= R)
    {
        return sum[rt];
    }
    int m = (l + r) >> 1;
    int ret = 0;
    if (L <= m) ret += query(L , R , lson);
    if (R > m) ret += query(L , R , rson);
    return ret;
}
int main()
{
    int T , n;
    scanf("%d",&T);
    for (int cas = 1 ; cas <= T ; cas ++)
    {
        printf("Case %d:\n",cas);
        scanf("%d",&n);
        build(1 , n , 1);
        char op[10];
        while (scanf("%s",op))
        {
            if (op[0] == 'E') break;
            int a , b;
            scanf("%d%d",&a,&b);
            if (op[0] == 'Q') printf("%d\n",query(a , b , 1 , n , 1));
            else if (op[0] == 'S') update(a , -b , 1 , n , 1);
            else update(a , b , 1 , n , 1);
        }
    }
    return 0;
}
```

12.1.2 段操作（整体区间操作）

成段增减

```
#include <cstdio>
#include <algorithm>
using namespace std;

#define lson l , m , rt << 1
#define rson m + 1 , r , rt << 1 | 1
#define LL long long
const int maxn = 111111;
LL add[maxn<<2];
LL sum[maxn<<2];
void PushUp(int rt)
{
    sum[rt] = sum[rt<<1] + sum[rt<<1|1];
}
void PushDown(int rt,int m)
{
    if (add[rt])
    {
        add[rt<<1] += add[rt];
        add[rt<<1|1] += add[rt];
        sum[rt<<1] += add[rt] * (m - (m >> 1));
        sum[rt<<1|1] += add[rt] * (m >> 1);
        add[rt] = 0;
    }
}
void build(int l,int r,int rt)
{
    add[rt] = 0;
    if (l == r)
    {
        scanf("%lld",&sum[rt]);
        return ;
    }
    int m = (l + r) >> 1;
    build(lson);
    build(rson);
    PushUp(rt);
}
void update(int L,int R,int c,int l,int r,int rt)
{
    if (L <= l && r <= R)
    {
```

```
        add[rt] += c;
        sum[rt] += (LL)c * (r - l + 1);
        return ;
    }
    PushDown(rt , r - l + 1);
    int m = (l + r) >> 1;
    if (L <= m) update(L , R , c , lson);
    if (m < R) update(L , R , c , rson);
    PushUp(rt);
}
LL query(int L,int R,int l,int r,int rt)
{
    if (L <= l && r <= R)
    {
        return sum[rt];
    }
    PushDown(rt , r - l + 1);
    int m = (l + r) >> 1;
    LL ret = 0;
    if (L <= m) ret += query(L , R , lson);
    if (m < R) ret += query(L , R , rson);
    return ret;
}
int main()
{
    int N , Q;
    scanf("%d%d",&N,&Q);
    build(1 , N , 1);
    while (Q --)
    {
        char op[2];
        int a , b , c;
        scanf("%s",op);
        if (op[0] == 'Q')
        {
            scanf("%d%d",&a,&b);
            printf("%lld\n",query(a , b , 1 , N , 1));
        }
        else
        {
            scanf("%d%d%d",&a,&b,&c);
            update(a , b , c , 1 , N , 1);
        }
    }
    return 0;
```

```
}
```

成段替换

```
#include <cstdio>
#include <algorithm>
using namespace std;

/*成段替换*/
#define lson l , m , rt << 1
#define rson m + 1 , r , rt << 1 | 1
const int maxn = 111111;
int h , w , n;
int col[maxn<<2];
int sum[maxn<<2];
void PushUp(int rt)
{
    sum[rt] = sum[rt<<1] + sum[rt<<1|1];
}
void PushDown(int rt,int m)
{
    if (col[rt])
    {
        col[rt<<1] = col[rt<<1|1] = col[rt];
        sum[rt<<1] = (m - (m >> 1)) * col[rt];
        sum[rt<<1|1] = (m >> 1) * col[rt];
        col[rt] = 0;
    }
}
void build(int l,int r,int rt)
{
    col[rt] = 0;
    sum[rt] = 1;//每一点的初始值为1 （区间原始的长度为r-l+1）
    if (l == r) return ;
    int m = (l + r) >> 1;
    build(lson);
    build(rson);
    PushUp(rt);
}
/*将L-R区间替换成c（数值修改）*/
void update(int L,int R,int c,int l,int r,int rt)
{
    if (L <= l && r <= R)
    {
        col[rt] = c;
```

```
        sum[rt] = c * (r - l + 1);
        return ;
    }
    PushDown(rt , r - l + 1);
    int m = (l + r) >> 1;
    if (L <= m) update(L , R , c , lson);
    if (R > m) update(L , R , c , rson);
    PushUp(rt);
}
int main()
{
    int T , n , m;
    scanf("%d",&T);
    for (int cas = 1 ; cas <= T ; cas ++)
    {
        scanf("%d%d",&n,&m);
        build(1 , n , 1);
        while (m --)
        {
            int a , b , c;
            scanf("%d%d%d",&a,&b,&c);
            update(a , b , c , 1 , n , 1);
        }
        printf("Case %d: The total value of the hook is %d.\n",cas , sum[1]);
    }
    return 0;
}
```

12.1.3 hash+成段更新

```
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
#define lson l , m , rt << 1
#define rson m + 1 , r , rt << 1 | 1

/*poj2528 Mayor’s posters
离散化简单的来说就是只取我们需要的值来用
比如说区间[1000,2000],[1990,2012]
我们用不到[-∞,999][1001,1989][1991,1999][2001,2011][2013,+∞] 这些值,
所以我只需要1000,1990,2000,2012 就够了,
将其分别映射到0,1,2,3,在于复杂度就大大的降下来了

所以离散化要保存所有需要用到的值,排序后,分别映射到1~n,这样复杂度就会小很多很多
```

```
而这题的难点在于每个数字其实表示的是一个单位长度(并非一个点),
这样普通的离散化会造成许多错误(包括我以前的代码,poj这题数据奇弱)
给出下面两个简单的例子应该能体现普通离散化的缺陷:
例子一:1-10 1-4 5-10
例子二:1-10 1-4 6-10
普通离散化后都变成了[1,4][1,2][3,4]
线段2覆盖了[1,2],线段3覆盖了[3,4],那么线段1是否被完全覆盖掉了呢?
例子一是完全被覆盖掉了,而例子二没有被覆盖
为了解决这种缺陷,我们可以在排序后的数组上加些处理,比如说[1,2,6,10]
如果相邻数字间距大于1的话,在其中加上任意一个数字,
比如加成[1,2,3,6,7,10], 然后再做线段树就好了.
线段树功能:update:成段替换 query:简单hash
*/

const int maxn = 11111;
bool hash[maxn];
int li[maxn] , ri[maxn];
int X[maxn*3];
int col[maxn<<4];
int cnt;

void PushDown(int rt)
{
    if (col[rt] != -1)
    {
        col[rt<<1] = col[rt<<1|1] = col[rt];
        col[rt] = -1;
    }
}
void update(int L,int R,int c,int l,int r,int rt)
{
    if (L <= l && r <= R)
    {
        col[rt] = c;
        return ;
    }
    PushDown(rt);
    int m = (l + r) >> 1;
    if (L <= m) update(L , R , c , lson);
    if (m < R) update(L , R , c , rson);
}
void query(int l,int r,int rt)
{
    if (col[rt] != -1)
    {
```

```
        if (!hash[col[rt]]) cnt ++;
        hash[ col[rt] ] = true;
        return ;
    }
    if (l == r) return ;
    int m = (l + r) >> 1;
    query(lson);
    query(rson);
}
int Bin(int key,int n,int X[])
{
    int l = 0 , r = n - 1;
    while (l <= r)
    {
        int m = (l + r) >> 1;
        if (X[m] == key) return m;
        if (X[m] < key) l = m + 1;
        else r = m - 1;
    }
    return -1;
}
int main()
{
    int T , n;
    scanf("%d",&T);
    while (T --)
    {
        scanf("%d",&n);
        int nn = 0;
        for (int i = 0 ; i < n ; i ++)
        {
            scanf("%d%d",&li[i] , &ri[i]);
            X[nn++] = li[i];
            X[nn++] = ri[i];
        }
        sort(X , X + nn);
        int m = 1;
        /*在区间端点添加一些点*/
        for (int i = 1 ; i < nn; i ++)
        {
            if (X[i] != X[i-1]) X[m ++] = X[i];
        }
        for (int i = m - 1 ; i > 0 ; i --)
        {
            if (X[i] != X[i-1] + 1) X[m ++] = X[i-1] + 1;
```

```
        }
        sort(X , X + m);
        memset(col , -1 , sizeof(col));
        for (int i = 0 ; i < n ; i ++)
        {
            int l = Bin(li[i] , m , X);
            int r = Bin(ri[i] , m , X);
            update(l , r , i , 0 , m , 1);
        }
        cnt = 0;
        memset(hash , false , sizeof(hash));
        query(0 , m , 1);
        printf("%d\n",cnt);
    }
    return 0;
}
```

12.1.4 区间交并补等

```
#include <cstdio>
#include <cstring>
#include <cctype>
#include <algorithm>
using namespace std;
#define lson l , m , rt << 1
#define rson m + 1 , r , rt << 1 | 1
/*这类题目会询问区间中满足条件的连续最长区间,所以PushUp的时候需要对左右儿子的区间
进行合并*/
/*poj3667 Hotel
题意:1 a:询问是不是有连续长度为a 的空房间,有的话住进最左边
2 a b:将[a,a+b-1]的房间清空
思路:记录区间中最长的空房间
线段树操作:update:区间替换 query:询问满足条件的最左断点
*/
const int maxn = 55555;
int lsum[maxn<<2] , rsum[maxn<<2] , msum[maxn<<2];
int cover[maxn<<2];
//cover[rt]=-1表示rt还没被更新
//cover[rt]=0表示全部区间都是0 （没有住人，未被占用）
//cover[rt]=1表示都是1 （全部是满的，被占用);
void PushDown(int rt,int m)
{
    if (cover[rt] != -1)
    {
        cover[rt<<1] = cover[rt<<1|1] = cover[rt];
```

```
        msum[rt<<1] = lsum[rt<<1] = rsum[rt<<1] = cover[rt] ? 0 : m - (m >> 1);
        msum[rt<<1|1] = lsum[rt<<1|1] = rsum[rt<<1|1] = cover[rt] ? 0 : (m >> 1);
        cover[rt] = -1;
    }
}
void PushUp(int rt,int m)
{
    lsum[rt] = lsum[rt<<1];
    rsum[rt] = rsum[rt<<1|1];
    if (lsum[rt] == m - (m >> 1)) lsum[rt] += lsum[rt<<1|1];
    if (rsum[rt] == (m >> 1)) rsum[rt] += rsum[rt<<1];
    msum[rt] = max(lsum[rt<<1|1] + rsum[rt<<1] , max(msum[rt<<1] , msum[rt<<1|1]));
}
void build(int l,int r,int rt)
{
    msum[rt] = lsum[rt] = rsum[rt] = r - l + 1;
    cover[rt] = -1;
    if (l == r) return ;
    int m = (l + r) >> 1;
    build(lson);
    build(rson);
}
void update(int L,int R,int c,int l,int r,int rt)
{
    if (L <= l && r <= R)
    {
        msum[rt] = lsum[rt] = rsum[rt] = c ? 0 : r - l + 1;
        cover[rt] = c;
        return ;
    }
    PushDown(rt , r - l + 1);
    int m = (l + r) >> 1;
    if (L <= m) update(L , R , c , lson);
    if (m < R) update(L , R , c , rson);
    PushUp(rt , r - l + 1);
}
int query(int w,int l,int r,int rt)
{
    if (l == r) return l;
    PushDown(rt , r - l + 1);
    int m = (l + r) >> 1;
    if (msum[rt<<1] >= w) return query(w , lson);
    else if (rsum[rt<<1] + lsum[rt<<1|1] >= w) return m - rsum[rt<<1] + 1;
    return query(w , rson);
}
```

```
int main()
{
    int n , m;
    scanf("%d%d",&n,&m);
    build(1 , n , 1);
    while (m --)
    {
        int op , a , b;
        scanf("%d",&op);
        if (op == 1)
        {
            scanf("%d",&a);
            if (msum[1] < a) puts("0");
            else
            {
                int p = query(a , 1 , n , 1);
                printf("%d\n",p);
                update(p , p + a - 1 , 1 , 1 , n , 1);
            }
        }
        else
        {
            scanf("%d%d",&a,&b);
            update(a , a + b - 1 , 0 , 1 , n , 1);
        }
    }
    return 0;
}
```

12.1.5 连续区间合并问题

```
#include <cstdio>
#include <cstring>
#include <cctype>
#include <algorithm>
using namespace std;
#define lson l , m , rt << 1
#define rson m + 1 , r , rt << 1 | 1
/*这类题目会询问区间中满足条件的连续最长区间,所以PushUp的时候需要对左右儿子的区间
进行合并*/
/*poj3667 Hotel
题意:1 a:询问是不是有连续长度为a 的空房间,有的话住进最左边
2 a b:将[a,a+b-1]的房间清空
思路:记录区间中最长的空房间
线段树操作:update:区间替换 query:询问满足条件的最左断点
```

```
*/
const int maxn = 55555;
int lsum[maxn<<2] , rsum[maxn<<2] , msum[maxn<<2];
int cover[maxn<<2];
//cover[rt]=-1表示rt还没被更新
//cover[rt]=0表示全部区间都是0 （没有住人，未被占用）
//cover[rt]=1表示都是1 （全部是满的，被占用);
void PushDown(int rt,int m)
{
    if (cover[rt] != -1)
    {
        cover[rt<<1] = cover[rt<<1|1] = cover[rt];
        msum[rt<<1] = lsum[rt<<1] = rsum[rt<<1] = cover[rt] ? 0 : m - (m >> 1);
        msum[rt<<1|1] = lsum[rt<<1|1] = rsum[rt<<1|1] = cover[rt] ? 0 : (m >> 1);
        cover[rt] = -1;
    }
}
void PushUp(int rt,int m)
{
    lsum[rt] = lsum[rt<<1];
    rsum[rt] = rsum[rt<<1|1];
    if (lsum[rt] == m - (m >> 1)) lsum[rt] += lsum[rt<<1|1];
    if (rsum[rt] == (m >> 1)) rsum[rt] += rsum[rt<<1];
    msum[rt] = max(lsum[rt<<1|1] + rsum[rt<<1] , max(msum[rt<<1] , msum[rt<<1|1]));
}
void build(int l,int r,int rt)
{
    msum[rt] = lsum[rt] = rsum[rt] = r - l + 1;
    cover[rt] = -1;
    if (l == r) return ;
    int m = (l + r) >> 1;
    build(lson);
    build(rson);
}
void update(int L,int R,int c,int l,int r,int rt)
{
    if (L <= l && r <= R)
    {
        msum[rt] = lsum[rt] = rsum[rt] = c ? 0 : r - l + 1;
        cover[rt] = c;
        return ;
    }
    PushDown(rt , r - l + 1);
    int m = (l + r) >> 1;
    if (L <= m) update(L , R , c , lson);
```

```
    if (m < R) update(L , R , c , rson);
    PushUp(rt , r - l + 1);
}
int query(int w,int l,int r,int rt)
{
    if (l == r) return l;
    PushDown(rt , r - l + 1);
    int m = (l + r) >> 1;
    if (msum[rt<<1] >= w) return query(w , lson);
    else if (rsum[rt<<1] + lsum[rt<<1|1] >= w) return m - rsum[rt<<1] + 1;
    return query(w , rson);
}
int main()
{
    int n , m;
    scanf("%d%d",&n,&m);
    build(1 , n , 1);
    while (m --)
    {
        int op , a , b;
        scanf("%d",&op);
        if (op == 1)
        {
            scanf("%d",&a);
            if (msum[1] < a) puts("0");
            else
            {
                int p = query(a , 1 , n , 1);
                printf("%d\n",p);
                update(p , p + a - 1 , 1 , 1 , n , 1);
            }
        }
        else
        {
            scanf("%d%d",&a,&b);
            update(a , a + b - 1 , 0 , 1 , n , 1);
        }
    }
    return 0;
}
```

12.1.6 扫描线

```
1.
/*这类题目需要将一些操作排序,然后从左到右用一根扫描线(当然是在我们脑子里)扫过去
```

```
最典型的就是矩形面积并,周长并等题
*/
/*hdu1542 Atlantis
题意:矩形面积并
思路:浮点数先要离散化;然后把矩形分成两条边,上边和下边,对横轴建树,
然后从下到上扫描上去,用cnt表示该区间下边比上边多几个,sum代表该区间内被覆盖的线段的长度总和
这里线段树的一个结点并非是线段的一个端点,而是该端点和下一个端点间的线段,
所以题目中r+1,r-1的地方可以自己好好的琢磨一下
线段树操作:update:区间增减 query:直接取根节点的值
*/
#include <cstdio>
#include <cstring>
#include <cctype>
#include <algorithm>
using namespace std;
#define lson l , m , rt << 1
#define rson m + 1 , r , rt << 1 | 1

const int maxn = 2222;
int cnt[maxn << 2];
double sum[maxn << 2];
double X[maxn];
struct Seg
{
    double h , l , r;
    int s;
    Seg() {}
    Seg(double a,double b,double c,int d) : l(a) , r(b) , h(c) , s(d) {}
    bool operator < (const Seg &cmp) const
    {
        return h < cmp.h;
    }
} ss[maxn];
void PushUp(int rt,int l,int r)
{
    if (cnt[rt]) sum[rt] = X[r+1] - X[l];
    else if (l == r) sum[rt] = 0;
    else sum[rt] = sum[rt<<1] + sum[rt<<1|1];
}
void update(int L,int R,int c,int l,int r,int rt)
{
    if (L <= l && r <= R)
    {
        cnt[rt] += c;
```

```
        PushUp(rt , l , r);
        return ;
    }
    int m = (l + r) >> 1;
    if (L <= m) update(L , R , c , lson);
    if (m < R) update(L , R , c , rson);
    PushUp(rt , l , r);
}
int Bin(double key,int n,double X[])
{
    int l = 0 , r = n - 1;
    while (l <= r)
    {
        int m = (l + r) >> 1;
        if (X[m] == key) return m;
        if (X[m] < key) l = m + 1;
        else r = m - 1;
    }
    return -1;
}
int main()
{
    int n , cas = 1;
    while (~scanf("%d",&n) && n)
    {
        int m = 0;
        while (n --)
        {
            double a , b , c , d;
            scanf("%lf%lf%lf%lf",&a,&b,&c,&d);
            X[m] = a;
            ss[m++] = Seg(a , c , b , 1);
            X[m] = c;
            ss[m++] = Seg(a , c , d , -1);
        }
        sort(X , X + m);
        sort(ss , ss + m);
        int k = 1;
        for (int i = 1 ; i < m ; i ++)
        {
            if (X[i] != X[i-1]) X[k++] = X[i];
        }
        memset(cnt , 0 , sizeof(cnt));
        memset(sum , 0 , sizeof(sum));
        double ret = 0;
```

```
        for (int i = 0 ; i < m - 1 ; i ++)
        {
            int l = Bin(ss[i].l , k , X);
            int r = Bin(ss[i].r , k , X) - 1;
            if (l <= r) update(l , r , ss[i].s , 0 , k - 1, 1);
            ret += sum[1] * (ss[i+1].h - ss[i].h);
        }
        printf("Test case #%d\nTotal explored area: %.2lf\n\n",cas++ , ret);
    }
    return 0;
}
2.
/*
hdu1828 Picture
题意:矩形周长并
思路:与面积不同的地方是还要记录竖的边有几个(numseg记录),
并且当边界重合的时候需要合并(用lbd和rbd表示边界来辅助)
线段树操作:update:区间增减 query:直接取根节点的值
*/

#include <cstdio>
#include <cstring>
#include <cctype>
#include <algorithm>
using namespace std;
#define lson l , m , rt << 1
#define rson m + 1 , r , rt << 1 | 1

const int maxn = 22222;
struct Seg
{
    int l , r , h , s;
    Seg() {}
    Seg(int a,int b,int c,int d):l(a) , r(b) , h(c) , s(d) {}
    bool operator < (const Seg &cmp) const
    {
        if (h == cmp.h) return s > cmp.s;
        return h < cmp.h;
    }
} ss[maxn];
bool lbd[maxn<<2] , rbd[maxn<<2];
int numseg[maxn<<2];
int cnt[maxn<<2];
int len[maxn<<2];
void PushUP(int rt,int l,int r)
```

```
{
    if (cnt[rt])
    {
        lbd[rt] = rbd[rt] = 1;
        len[rt] = r - l + 1;
        numseg[rt] = 2;
    }
    else if (l == r)
    {
        len[rt] = numseg[rt] = lbd[rt] = rbd[rt] = 0;
    }
    else
    {
        lbd[rt] = lbd[rt<<1];
        rbd[rt] = rbd[rt<<1|1];
        len[rt] = len[rt<<1] + len[rt<<1|1];
        numseg[rt] = numseg[rt<<1] + numseg[rt<<1|1];
        if (lbd[rt<<1|1] && rbd[rt<<1]) numseg[rt] -= 2;//两条线重合
    }
}
void update(int L,int R,int c,int l,int r,int rt)
{
    if (L <= l && r <= R)
    {
        cnt[rt] += c;
        PushUP(rt , l , r);
        return ;
    }
    int m = (l + r) >> 1;
    if (L <= m) update(L , R , c , lson);
    if (m < R) update(L , R , c , rson);
    PushUP(rt , l , r);
}
int main()
{
    int n;
    while (~scanf("%d",&n))
    {
        int m = 0;
        int lbd = 10000, rbd = -10000;
        for (int i = 0 ; i < n ; i ++)
        {
            int a , b , c , d;
            scanf("%d%d%d%d",&a,&b,&c,&d);
            lbd = min(lbd , a);
```

```
            rbd = max(rbd , c);
            ss[m++] = Seg(a , c , b , 1);
            ss[m++] = Seg(a , c , d , -1);
        }
        sort(ss , ss + m);
        int ret = 0 , last = 0;
        for (int i = 0 ; i < m ; i ++)
        {
            if (ss[i].l < ss[i].r)
                update(ss[i].l , ss[i].r - 1 , ss[i].s , lbd , rbd - 1 , 1);
            ret += numseg[1] * (ss[i+1].h - ss[i].h);
            ret += abs(len[1] - last);
            last = len[1];
        }
        printf("%d\n",ret);
    }
    return 0;
}
```

12.1.7 重要题目

LA3938 Ray, Pass me the dishes!

```
// LA3938 Ray, Pass me the dishes!
// Rujia Liu
//给定一个序列，多次询问区间的[ai,bi],求ai-->bi 中最大的连续区间和的最大值（num[q]+num[q+1]+num[q+2]
//并且给出所求区间的两端坐标（字典序最小）
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;

const int maxn = 500000 + 10;
const int maxnode = 1000000 + 10;
typedef long long LL;
typedef pair<int,int> Interval;

LL prefix_sum[maxn];//整个序列前缀和

LL sum(int L, int R)
{
    return prefix_sum[R] - prefix_sum[L-1];
}

LL sum(Interval p)
```

```
{
    return sum(p.first, p.second);
}

Interval better(Interval a, Interval b)
{
    if(sum(a) != sum(b)) return sum(a) > sum(b) ? a : b;
    return a < b ? a : b; // 利用pair 自带的字典序
}

int qL, qR;

struct IntervalTree
{
    int max_prefix[maxnode];
    int max_suffix[maxnode];
    Interval max_sub[maxnode];

    void build(int o, int L, int R)
    {
        if(L == R)
        {
            max_prefix[o] = max_suffix[o] = L;
            max_sub[o] = make_pair(L, L);
        }
        else
        {
            int M = L + (R-L)/2;
            // 递归创建子树
            int lc = o*2, rc = o*2+1;
            build(lc, L, M);
            build(rc, M+1, R);

            // 递推max_prefix
            LL v1 = sum(L, max_prefix[lc]);
            LL v2 = sum(L, max_prefix[rc]);
            if(v1 == v2) max_prefix[o] = min(max_prefix[lc], max_prefix[rc]);
            else max_prefix[o] = v1 > v2 ? max_prefix[lc] : max_prefix[rc];

            // 递推max_suffix
            v1 = sum(max_suffix[lc], R);
            v2 = sum(max_suffix[rc], R);
            if(v1 == v2) max_suffix[o] = min(max_suffix[lc], max_suffix[rc]);
            else max_suffix[o] = v1 > v2 ? max_suffix[lc] : max_suffix[rc];
```

```
            // 递推max_sub
            max_sub[o] = better(max_sub[lc], max_sub[rc]); // 完全在左子树或者右
子树
            max_sub[o] = better(max_sub[o], make_pair(max_suffix[lc], max_prefix[rc])); // 跨
越中线
        }
    }

    Interval query_prefix(int o, int L, int R)
    {
        if(max_prefix[o] <= qR) return make_pair(L, max_prefix[o]);
        int M = L + (R-L)/2;
        int lc = o*2, rc = o*2+1;
        if(qR <= M) return query_prefix(lc, L, M);
        Interval i = query_prefix(rc, M+1, R);
        i.first = L;
        return better(i, make_pair(L, max_prefix[lc]));
    }

    Interval query_suffix(int o, int L, int R)
    {
        if(max_suffix[o] >= qL) return make_pair(max_suffix[o], R);
        int M = L + (R-L)/2;
        int lc = o*2, rc = o*2+1;
        if(qL > M) return query_suffix(rc, M+1, R);
        Interval i = query_suffix(lc, L, M);
        i.second = R;
        return better(i, make_pair(max_suffix[rc], R));
    }

    Interval query(int o, int L, int R)
    {
        if(qL <= L && R <= qR) return max_sub[o];
        int M = L + (R-L)/2;
        int lc = o*2, rc = o*2+1;
        if(qR <= M) return query(lc, L, M);
        if(qL > M) return query(rc, M+1, R);
        Interval i1 = query_prefix(rc, M+1, R); // 右半的前缀
        Interval i2 = query_suffix(lc, L, M); // 左半的后缀
        Interval i3 = better(query(lc, L, M), query(rc, M+1, R));
        return better(make_pair(i2.first, i1.second), i3);
    }
};

IntervalTree tree;
```

```
int main()
{
    int kase = 0, n, a, Q;
    while(scanf("%d%d", &n, &Q) == 2)
    {
        prefix_sum[0] = 0;
        for(int i = 0; i < n; i++)
        {
            scanf("%d", &a);
            prefix_sum[i+1] = prefix_sum[i] + a;
        }
        tree.build(1, 1, n);
        printf("Case %d:\n", ++kase);
        while(Q--)
        {
            int L, R;
            scanf("%d%d", &L, &R);
            qL = L;
            qR = R;
            Interval ans = tree.query(1, 1, n);
            printf("%d %d\n", ans.first, ans.second);
        }
    }
    return 0;
}
```

§ 12.2 伸展树(SPlay)

12.2.1 SPlay模板一

```
/*
每次先调用一下splay::init()
不同的题目注意修改newNode，pushdown，update三个函数就足够了
splay命名空间里的函数都是对树的操作，对结点的操作都归类到Node里。
空间吃紧时可修改erase函数，增加内存池管理
*/
#include <algorithm>
#include <cstdio>
#include <iostream>
#include  <string>
#include  <cstring>
//#include <windows.h>
using namespace std;
const int N = 1e5 + 10;
```

```
struct Node
{
    int key, s, cnt; ///key是键值，s是以this为根的所有子树的节点数，cnt是本节点的
重复数
    bool rvs;
    Node *f, *ch[2];
    void set(int c, Node *x);
    void fix();
    void pushdown();
    void update();
    void rotate();
    void splay(Node *);
    Node()
    {
        key = 0; s = 0 ; cnt = 0 ;
        rvs = 0;
        ch[0] = ch[1] = f = NULL;
    }
} statePool[N];
Node *null = new Node();

void Node::set(int c, Node *x)///另x为this节点的第c儿子，c==1为右儿子
{
    ch[c] = x;
    x->f = this;
}

void Node::fix()///将儿子指向父亲
{
    ch[0]->f = this;
    ch[1]->f = this;
}

void Node::pushdown()///根据题目而定
{
    if (this == null) return;
    if (rvs)
    {
        ch[0]->rvs ^= 1;
        ch[1]->rvs ^= 1;
        rvs = 0;
        swap(ch[0], ch[1]);
    }
}
```

```
void Node::update()///用儿子更新根
{
    if (this == null) return;
    s = ch[0]->s + ch[1]->s + cnt;
}

void Node::rotate()
{
    Node *x = f;
    bool o = f->ch[0] == this;
    x->set(!o, ch[o]);
    x->f->set(x->f->ch[1] == x, this);
    set(o, x);
    x->update();
    update();
}

void Node::splay(Node *goal = null)
{
    pushdown();
    for (; f != goal;)
    {
        f->f->pushdown();
        f->pushdown();
        pushdown();
        if (f->f == goal)
        {
            rotate();
        }
        else if ((f->f->ch[0] == f) ^ (f->ch[0] == this))
        {
            rotate();
            rotate();
        }
        else
        {
            f->rotate();
            rotate();
        }
    }
}

struct Splay
{
```

```
    int poolCnt;///key值不同的个数
    Node *root ;
    int id[100000];///id[i]:将序列中的标号（元素值，一定注意不是序号）为i的节点在statePool中的存储位置
    Node *newNode(int key = 0)///
    {
        Node *p = &statePool[poolCnt++];
        p->f = p->ch[0] = p->ch[1] = null;
        p->s = p->cnt = 1;
        p->rvs = 0;
        p->key = key;
        id[p->key] = poolCnt - 1; ///做内存映射，根据题目而定（通常是将1-n的序列映射成另一个排列
        ///值为p->key的内存对应位置位 statePool[ id[p->key] ];
        return p;
    }
    void init()
    {
        poolCnt = 1;
        root = null;
    }
    /// use an array a to build a balanced splay tree. return its root.
    template <class T> Node *build(int l, int r, T a[])
    {
        if (l > r) return null;
        int mid = (l + r) / 2;
        Node *p = newNode(a[mid]);
        //        p->key = a[mid];
        //        cout<<"l,r: "<<l<<" " <<r<<endl;
        if (l < r)
        {
            p->ch[0] = build(l, mid - 1, a);
            p->ch[1] = build(mid + 1, r, a);
            p->update();
            p->fix();
        }
        return root = p;
    }

    /// select the i-th element in the splay tree. index start from 1.
    /// take care you must splay after select
    Node *select(Node *rt, int kth) ///查找第k小的值
    {
        if (rt == null || kth > rt->s)
            return null;
```

```
    Node *x = rt;
    while (1)
    {
        x->pushdown();
        if (x->ch[0]->s + 1 <= kth && kth <= x->ch[0]->s + x->cnt)
            break;
        else if (kth <= x->ch[0]->s)
            x = x->ch[0];
        else
        {
            kth -= x->ch[0]->s + x->cnt;
            x = x->ch[1];
        }
    }
    return x;
}

// find the node which just >= a in the tree.
// take care you must splay after lower_bound
template <class T> Node *lower_bound(T a)
{
    Node *ret = null;
    for (Node *p = root; p != null; )
    {
        p->pushdown();
        if (a < p->key)
        {
            p = p->ch[1];
        }
        else
        {
            ret = p;
            p = p->ch[0];
        }
    }
    return ret;
}

/// merge two splay tree. p, q should be root. return the root
///p在序列左边，q在右边
Node *merge(Node *p, Node *q)
{
    p->f = q->f = null;
    if (p == null) return q;
    if (q == null) return p;
```

```
    q = select(q, 1);
    q->splay();
    q->set(0, p);
    q->update();
    return q;
}

//when p is root, q is p's right child(take care after pushdown),
//reverse from p to q, return the root of this tree.
Node *reverse(Node *p, Node *q)
{
    swap(p->ch[0], q->ch[0]);
    p->ch[1] = q->ch[1];
    q->ch[1] = p;
    q->f = null;
    p->fix();
    q->fix();
    p->update();
    q->update();
    p->ch[0]->rvs ^= 1;
    return q;
}

// reverse the l-th element to the r-th element in the splay tree(inclusive),
// return the root of this tree, index start from 1.
void reverse(int l, int r)
{
    if (l >= r) return ;
    Node *p = select(root, l);
    p->splay();
    Node *q = select(p, r);
    q->splay(p);
    root = reverse(p, q);
}

///insert q before p. return the root of this tree.
Node *insert(Node *p, Node *q)
{
    p->splay();
    if (p->ch[0] == null)
    {
        p->set(0, q);
    }
    else
    {
```

```
        Node *t = select(p, p->ch[0]->s);
        t->splay(p);
        t->set(1, q);
        t->update();
    }
    p->update();
    return p;
}

///insert q before the i-th node.
///return the root of this tree. index start from 1.
///相当于在第i个位置插入节点q, 后面位置依次往后移动
void insert(Node *q, int i)
{
    if (i > root->s)
    {
        Node *p = select(root, root->s);
        p->splay();
        p->set(1, q);
        p->update();
        root = p;
    }
    else
    {
        Node *p = select(root, i);
        root = insert(p, q);
    }
}

///insert element into the splaytree,return the root of tree.
template<class T>Node *insert(T key)
{
    if (root == null)
    {
        poolCnt = 0;
        return root = newNode(key);
    }
    Node *x = root, *y;
    while (1)
    {
        x->s++;
        if (key == x->key)
        {
            x->cnt++;
            x->update();
```

```
            y = x;
            break;
        }
        else if (key < x->key)
        {
            if (x->ch[0] != null)
                x = x->ch[0];
            else
            {
                y = newNode(key);
                x->ch[0] = y;
                y->f = x;
                break;
            }
        }
        else
        {
            if (x->ch[1] != null)
                x = x->ch[1];
            else
            {
                y = newNode(key);
                x->ch[1] = y;
                y->f = x;
                break;
            }
        }
    }
    y->splay();
    return root = y;
}
///erase the subtree whose root is p. return the root.
Node *erase(Node *p)
{
    if (p->f != null)
    {
        Node *q = p->f;
        q->pushdown();
        q->set(q->ch[1] == p, null);
        q->update();
        q->splay();
        return root = q;
    }
    else
    {
```

```
        return root = null;
    }
}
/// erase the element whose index in [l, r].
/// index start from 1. return the root.
Node *erase(Node *&dump, int l, int r) ///
{
    if (l > r) return root;
    if (l == r)
    {
        Node *p = select(root, l);
        p->splay();
        Node *lp = p->ch[0], *rp = p->ch[1];
        p->ch[0] = p->ch[1] = null;
        dump = p;
        dump -> update();
        return  root = merge(lp, rp);
    }
    else
    {
        Node *p = select(root, l);
        p->splay();
        Node *q = select(p, r);
        q->splay(p);
        Node *lp = p->ch[0], *rp = p->ch[1];
        p->ch[0] = p->ch[1] = null;
        Node *lq = q->ch[0], *rq = q->ch[1];
        q->ch[0] = q->ch[1] = null;
        q->update();
        p->update();
        dump = merge(p, lq);
        dump = merge(dump, q);
        dump->update();
        return  root = merge(lp, rq);
    }
}
/// 把o的前k小结点放在left里，其他的放在right里。1<=k<=o->s。当k=o->s时，right=null
void split(int k, Node *&left, Node *&right)
{
    Node *p = select(root, k);
    p->splay();
    left = p;
    right = p->ch[1];
    p->ch[1] = null;
    left->update();
```

```
}

///以下是基于键值的维护，不能用来维护序列
Node *search(Node *root, int key) ///查找一个值，返回指针
{
    if (root == null)
        return null;
    Node *x = root, *y = null;
    while (1)
    {
        if (key == x->key)
        {
            y = x;
            break;
        }
        else if (key > x->key)
        {
            if (x->ch[1] != null)
                x = x->ch[1];
            else
                break;
        }
        else
        {
            if (x->ch[0] != null)
                x = x->ch[0];
            else
                break;
        }
    }
    x->splay();
    return root  = x;
}
Node *searchmin(Node *x) ///查找最小值，返回指针
{
    Node *y = x->f;
    if (y == null)return x;
    while (x->ch[0] != null) ///遍历到最左的儿子就是最小值
    {
        x = x->ch[0];
    }
    x->splay(y);
    return x;
}
void del(int key) ///删除一个值
```

```
    {
        if (root == null)
            return;
        Node *x = search(root, key), *y;
        if (x == null)
            return;
        if (x->cnt > 1)
        {
            x->cnt--;
            x->update();
            return;
        }
        else if (x->ch[0] == null && x->ch[1] == null)
        {
            init();
            return;
        }
        else if (x->ch[0] == null)
        {
            root = x->ch[1];
            x->ch[1]->f = null;
            return;
        }
        else if (x->ch[1] == null)
        {
            root = x->ch[0];
            x->ch[0]->f = null;
            return;
        }
        y = searchmin(x->ch[1]);
        y->f = null;
        y->ch[0] = x->ch[0];
        x->ch[0]->f = y;
        y->update();
        root = y;
    }
} spt;
int seq[N];

void splaySeq(Node *x)
{
    if (x != null)
    {
        splaySeq(x->ch[0]);
        printf("节点: %2d  左儿子:%2d 右儿子:%2d  cnt : %2d  s: %2d\n",
```

```
                x->key, x->ch[0]->key, x->ch[1]->key, x->cnt , x->s);
        splaySeq(x->ch[1]);
    }
}
void debug(Node *fir)
{
    printf("内存映射: \n");
    for (int i = 1; i <= n ; i++)
        printf("%d ", statePool[ spt.id[ seq[i] ] ].key);
    printf("\n");
    printf("root: %d\n", fir->key);
    splaySeq(fir);
    printf("\n");
}
void prin_ans(Node *rt)
{
    if (rt != null)
    {
        rt->pushdown();
        prin_ans(rt->ch[0]);
        printf("%d\n", rt->key);
        prin_ans(rt->ch[1]);
    }
}
```

12.2.2 SPlay模板二

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#define key_tree root->ch[1]->ch[0]
using namespace std;

const int M = 500000 * 2;
const int INF = 1000000000;

inline int max(int a, int b)
{
    return a > b ? a : b;
}
inline int min(int a, int b)
{
    return a < b ? a : b;
}
```

```
struct Node
{
    int sum;
    int value;
    int s;
    Node *ch[2];
    Node *pre;
    bool rev;
    bool same;
    int lx, rx, mx;
};


struct Splay
{

    Node  data[M];
    Node *store[M];///人工堆栈
    Node *null;
    Node *root;
    int number;
    int top;

    Node *newNode(int val)
    {
        Node *p;
        if (top > 0)
        {
            p = store[top--];
        }
        else
        {
            p = &data[number++];
        }
        p->s = 1;
        p->value = val;
        p->sum = val;
        p->rev = p->same = false;
        p->lx = p->rx = p->mx = val;
        p->ch[0] = p->ch[1] = null;
        p->pre = null;
        return p;
    }
```

```
    void init()
    {
        number = 0;
        top = 0;
        null = newNode(-INF);
        null->s = null->sum = 0;
        root = newNode(-INF);
        root->ch[1] = newNode(-INF);
        root->ch[1]->pre = root;
        push_up(root);//这里主要是为了更新s，sum的更新并不正确，但是在rotate和splay操
作中会正确

    }

    void push_up(Node *p)
    {
        if (p == null)
        {
            return;
        }
        push_down(p);
        push_down(p->ch[0]);//以求和为例，p节点值的更新需要依赖两个字节点，那么这
两个字节点的sum要先得到更新，所以要下放
        push_down(p->ch[1]);

        p->s = p->ch[0]->s + p->ch[1]->s + 1;
        p->sum = p->ch[0]->sum + p->ch[1]->sum + p->value;
        p->lx = max(p->ch[0]->lx, p->ch[0]->sum + p->value + max(0, p->ch[1]->lx));
        p->rx = max(p->ch[1]->rx, p->ch[1]->sum + p->value + max(0, p->ch[0]->rx));
        p->mx = max(p->ch[0]->mx, p->ch[1]->mx);
        p->mx = max(p->mx, max(p->ch[0]->rx + p->ch[1]->lx, 0) + p->value);
        p->mx = max(p->mx, max(p->ch[0]->rx, p->ch[1]->lx) + p->value);
    }

    void push_down(Node *p)
    {
        if (p == null)
        {
            return;
        }
        if (p->rev)
        {
            p->rev = false;
            p->ch[0]->rev ^= 1;
            p->ch[1]->rev ^= 1;
```

```
        swap(p->ch[0], p->ch[1]);
        swap(p->lx, p->rx);
    }
    if (p->same)
    {
        p->same = false;
        p->ch[0]->same = p->ch[1]->same = true;
        p->ch[0]->value = p->ch[1]->value = p->value;
        p->sum = p->value * p->s;
        p->lx = p->rx = p->mx = p->s * p->value;
        if (p->value < 0)
        {
            p->lx = p->rx = p->mx = p->value;
        }
    }

}


//中序遍历
void dfs(Node *p)
{
    if (p == null)
    {
        return;
    }
    dfs(p->ch[0]);
    printf("%d   ", p->value);
    dfs(p->ch[1]);
}
//c=1 右旋
void rotate(Node *x, int c)
{
    Node *y = x->pre;
    push_down(y);
    push_down(x);

    y->ch[!c] = x->ch[c];
    if (x->ch[c] != null)
    {
        x->ch[c]->pre = y;
    }
    x->pre = y->pre;
    if (y->pre != null)
    {
```

```
        if (y->pre->ch[0] == y)
        {
            y->pre->ch[0] = x;
        }
        else
        {
            y->pre->ch[1] = x;
        }
    }
    x->ch[c] = y;
    y->pre = x;
    push_up(y);
    if (y == root)
    {
        root = x;
    }

}
void splay(Node *x, Node *f)
{
    if (x == null)
    {
        return;
    }

    for (push_down(x); x->pre != f;)
    {
        if (x->pre->pre == f) //单旋
        {
            if (x == x->pre->ch[0])
            {
                rotate(x, 1);
            }
            else
            {
                rotate(x, 0);
            }
        }
        else
        {
            Node *y = x->pre;
            Node *z = y->pre;
            if (z->ch[0] == y)
            {
                if (y->ch[0] == x) //一字形旋转
```

```
                {
                    rotate(y, 1);
                    rotate(x, 1);

                }
                else//之字形
                {
                    rotate(x, 0);
                    rotate(x, 1);
                }
            }
            else
            {
                if (y->ch[1] == x) //一字形
                {
                    rotate(y, 0);
                    rotate(x, 0);
                }
                else//之字形
                {
                    rotate(x, 1);
                    rotate(x, 0);
                }
            }

        }
    }
    push_up(x);
}
void select(int k, Node *f)//k是指第几个节点,从1开始
{
    Node *t;
    //k = k-1+1;//其实正常情况下应该是 k = k-1;但是前后加两个节点的话，正好向
右移动一位
    for (t = root;;)
    {
        push_down(t);
        int s = t->ch[0]->s;
        if (k == s)
        {
            break;
        }
        if (k < s)
        {
            t = t->ch[0];
```

```
        }
        else
        {
            k -= s + 1;
            t = t->ch[1];
        }

    }
    splay(t, f);
}

Node *build(int left, int right, int *ary)
{
    if (left > right)
    {
        return null;
    }
    int mid = (left + right) >> 1;
    Node *p  =  newNode(ary[mid]);
    p->ch[0] = build(left, mid - 1, ary);
    if (p->ch[0] != null)
    {
        p->ch[0]->pre = p;
    }
    p->ch[1] = build(mid + 1, right, ary);
    if (p->ch[1] != null)
    {
        p->ch[1]->pre = p;
    }
    push_up(p);//Update操作
    return p;
}


void insert(int pos, int *ary, int n)
{
    select(pos, null);
    select(pos + 1, root);
    Node *p = build(1, n, ary);
    key_tree = p;
    p->pre = root->ch[1];
    splay(p, null); //将p移至根节点
}
void del(int pos, int end)
{
```

```
        select(pos - 1, null);
        select(end, root);
        Node *p = key_tree; //如果不回收的话，将会造成内存泄漏
        key_tree = null;
        splay(root->ch[1], null); //别忘了维护
        recyle(p);
    }

    void recyle(Node *p)//人工回收删除的空间，防止栈溢出
    {
        if (p == null)
        {
            return;
        }
        recyle(p->ch[0]);
        recyle(p->ch[1]);
        store[++top] = p;

    }
    int getSum(int start, int end)
    {
        select(start - 1, null);
        select(end, root); //这里完全不用担心，肯定会移到root的右子树，因为end比start-1大，
根据查询二叉树的性质
        return key_tree->sum;
    }
    int maxSum(int pos, int end)
    {
        select(pos - 1, null);
        select(end, root);
        Node *p = key_tree;
        return p->mx;
    }
    void makeSame(int pos, int end, int s)
    {
        select(pos - 1, null);
        select(end, root);
        Node *p = key_tree;
        p->same = true;
        p->value = s;
        splay(p, null);
    }
    void reverse(int pos, int end)
    {
        select(pos - 1, null);
```

```
        select(end, root);
        Node *p = key_tree;
        p->rev ^= 1;
        splay(p, null);
    }
} sp;
int ary[M];
int main()
{
    int T;
    for (scanf("%d", &T); T--;)
    {
        int n, m;
        int x, y;
        int s;
        Node *p;
        char cmd[20];

        scanf("%d%d", &n, &m);
        for (int i = 1; i <= n; i++)
        {
            scanf("%d", &ary[i]);
        }
        sp.init();
        sp.insert(0, ary, n);

        for (; m--;)
        {
            scanf("%s", cmd);
            if (strcmp(cmd, "GET-SUM") == 0)
            {
                scanf("%d%d", &x, &y);
                printf("%d\n", sp.getSum(x, x + y));
            }
            if (strcmp(cmd, "MAX-SUM") == 0)
            {
                printf("%d\n", sp.maxSum(1, sp.root->s - 1));
            }

            if (strcmp(cmd, "INSERT") == 0)
            {
                scanf("%d%d", &x, &y);
                for (int i = 1; i <= y; i++)
                {
                    scanf("%d", &ary[i]);
```

```
                }
                sp.insert(x, ary, y);
            }
            if (strcmp(cmd, "DELETE") == 0)
            {
                scanf("%d%d", &x, &y);
                sp.del(x, x + y);
            }
            if (strcmp(cmd, "REVERSE") == 0)
            {
                scanf("%d%d", &x, &y);
                sp.reverse(x, x + y);
            }
            if (strcmp(cmd, "MAKE-SAME") == 0)
            {
                scanf("%d%d%d", &x, &y, &s);
                sp.makeSame(x, x + y, s);
            }
        }
        //dfs(root);
        //printf("\n");
    }
    return 0;
}
```

§ 12.3 RMQ

```
/**
*RMQ实现的是在一个区间里面求最大（小）值，并且可以返回这个最大小值的索引id。
*预处理的时间复杂度度是O（n*lgn），查询是O（1）。
*一维RMQ ST算法
*构造RMQ数组 makermq(int n,int b[]) O(nlog(n))的算法复杂度
*dp[i][j] 表示从i到i+2^j -1中最小的一个值(从i开始持续2^j个数)
*dp[i][j]=min{dp[i][j-1],dp[i+2^(j-1)][j-1]}
*查询RMQ rmq(int s,int v)
*将s-v 分成两个2^k的区间
*即 k=(int)log2(s-v+1)
*查询结果应该为 min(dp[s][k],dp[v-2^k+1][k])
*/

//1.返回值
void makermq(int n, int b[])
{
    int i, j;
    for (i = 0; i < n; i++)
```

```
        dp[i][0] = b[i];
    for (j = 1; (1 << j) <= n; j++)
        for (i = 0; i + (1 << j) - 1 < n; i++)
            dp[i][j] = min(dp[i][j - 1], dp[i + (1 << (j - 1))][j - 1]);
}
int rmq(int s, int v)
{
    int k = (int)(log((v - s + 1) * 1.0) / log(2.0));
    /* 或者int d = v - s+1 , k;
    for(k = 0; (1<<k) <= d; k++) ;
    k- -;*/
    return min(dp[s][k], dp[v - (1 << k) + 1][k]);
}

//2.返回最小值对应的下标
void makeRmqIndex(int n, int b[])
{
    int i, j;
    for (i = 0; i < n; i++)
        dp[i][0] = i;
    for (j = 1; (1 << j) <= n; j++)
        for (i = 0; i + (1 << j) - 1 < n; i++)
            dp[i][j] = b[dp[i][j - 1]] < b[dp[i + (1 << (j - 1))][j - 1]] ? dp[i][j -
                        1] : dp[i + (1 << (j - 1))][j - 1];
}
int rmqIndex(int s, int v, int b[])
{
    int k = (int)(log((v - s + 1) * 1.0) / log(2.0));
    return b[dp[s][k]] < b[dp[v - (1 << k) + 1][k]] ? dp[s][k] : dp[v - (1 <
            < k) + 1][k];
}
```

Chapter 13

字符串

§ 13.1 KMP

13.1.1 普通KMP

```
//kmp模板:
/*
pku3461(Oulipo), hdu1711(Number Sequence)
*/
#include <iostream>
#include <cstring>
using namespace std;

const int N = 1000002;
int next[N];
char S[N], T[N];
//s是文本串，T是模式串
int slen, tlen;
//求next数组: next[i]=j表示T[0...j-1]=T[i-j...i-1];next[i]=max{k|T[0...k-1]=T[i-k...i-1]}(i-k>0)
void getNext(char *T)
{
    int j, k;
    j = 0; k = -1; next[0] = -1;
    while (j < tlen)
        if (k == -1 || T[j] == T[k])
            next[++j] = ++k;
        else
            k = next[k];
}
/*
返回模式串T在主串S中首次出现的位置
*/
int kmpindex(char *S, char *T)
```

```
{
    int i = 0, j = 0;
    getNext(T);

    while (i < slen && j < tlen)
    {
        if (j == -1 || S[i] == T[j])
        {
            i++; j++;
        }
        else
            j = next[j];
    }
    if (j == tlen)
        return i - tlen;
    else
        return -1;
}
/*
返回模式串在主串S中出现的次数
*/
int kmpcount(char *S, char *T)
{
    int ans = 0;
    int i, j = 0;

    if (slen == 1 && tlen == 1)
    {
        if (S[0] == T[0])
            return 1;
        else
            return 0;
    }
    getNext(T);
    for (i = 0; i < slen; i++)
    {
        while (j > 0 && S[i] != T[j])
            j = next[j];
        if (S[i] == T[j])
            j++;
        if (j == tlen)
        {
            ans++;
            j = next[j];
        }
```

```
    }
    return ans;
}
void check_next()
{
    for (int i = 0; i <= tlen; i++)
    {
        cout << next[i] << " " ;
    }
    cout << endl;
}
int main()
{

    int TT;
    int i, cc;
    cin >> TT;
    while (TT--)
    {
        cin >> S >> T;
        slen = strlen(S);
        tlen = strlen(T);
        cout << "模式串T在主串S中首次出现的位置是: " << kmpindex(S, T) << endl;
        cout << "模式串T在主串S中出现的次数为: " << kmpcount(S, T) << endl;
    }
    return 0;
}
```

13.1.2 拓展KMP

```
/*
拓展kmp模板:
*/
#include <iostream>
#include <string>
#include <cstring>
#include <cstdio>
using namespace std;
const int MM = 100005;

int next[MM], extand[MM];
char S[MM], T[MM];

//next[i] = T[i...tlen-1]与T[0...len-1]的最长公共前缀;
void GetNext(const char *T)
```

```
{
    int len = strlen(T), a = 0;
    next[0] = len;
    while (a < len - 1 && T[a] == T[a + 1]) a++;
    next[1] = a;
    a = 1;
    for (int k = 2; k < len; k++)
    {
        int p = a + next[a] - 1, L = next[k - a];
        if ( (k - 1) + L >= p)
        {
            int j = (p - k + 1) > 0 ? (p - k + 1) : 0;
            while (k + j < len && T[k + j] == T[j]) j++;
            next[k] = j;
            a = k;
        }
        else
            next[k] = L;
    }
}
//extend[i]表示S[i...slen-1]与T[0...len-1]的最长公共前缀
void GetExtand(const char *S, const char *T)
{
    GetNext(T);
    int slen = strlen(S), tlen = strlen(T), a = 0;
    int MinLen = slen < tlen ? slen : tlen;
    while (a < MinLen && S[a] == T[a]) a++;
    extand[0] = a;
    a = 0;
    for (int k = 1; k < slen; k++)
    {
        int p = a + extand[a] - 1, L = next[k - a];
        if ( (k - 1) + L >= p)
        {
            int j = (p - k + 1) > 0 ? (p - k + 1) : 0;
            while (k + j < slen && j < tlen && S[k + j] == T[j]) j++;
            extand[k] = j;
            a = k;
        }
        else
            extand[k] = L;
    }
}
int main()
{
```

```
    while (scanf("%s%s", S, T) == 2)
    {
        GetExtand(S, T);
        printf("next: \n");
        for (int i = 0; i < strlen(T); i++)
            printf("%d ", next[i]);
        puts("\n");
        printf("extend: \n");
        for (int i = 0; i < strlen(S); i++)
            printf("%d ", extand[i]);
        puts("");
    }
    return 0;
}
```

§ 13.2 后缀数组

13.2.1 后缀数组模板

```
#include <string.h>
#include <math.h>
#include <stdio.h>
#include <iostream>
using namespace std;
const int M = 200000;
int wa[M], wb[M], wv[M], wsum[M];
int sa[M];
int cmp(int *r, int a, int b, int l)
{
    return r[a] == r[b] && r[a + l] == r[b + l];
}
/**
sa[i](i: 0 - n-1,接进来如果是0-n-2的话，最后一个元素是补0的)范围是0-n;
sa[0] == n-1 (如果最后元素补的是0)
m是字符串的取值范围
n是开区间，n = len + 1;
*/
void da(int *r, int *sa, int n, int m)
{
    int i, j, p, *x = wa, *y = wb, *t;
    for (i = 0; i < m; i++)
        wsum[i] = 0;
    for (i = 0; i < n; i++)
        wsum[x[i] = r[i]]++;
    for (i = 1; i < m; i++)
```

```
        wsum[i] += wsum[i - 1];
    for (i = n - 1; i >= 0; i--)
        sa[--wsum[x[i]]] = i;
    for (j = 1, p = 1; p < n; j *= 2, m = p)
    {
        for (p = 0, i = n - j; i < n; i++)
            y[p++] = i;
        for (i = 0; i < n; i++)
            if (sa[i] >= j)
                y[p++] = sa[i] - j;
        for (i = 0; i < n; i++)
            wv[i] = x[y[i]];
        for (i = 0; i < m; i++)
            wsum[i] = 0;
        for (i = 0; i < n; i++)
            wsum[wv[i]]++;
        for (i = 1; i < m; i++)
            wsum[i] += wsum[i - 1];
        for (i = n - 1; i >= 0; i--)
            sa[--wsum[wv[i]]] = y[i];
        for (t = x, x = y, y = t, p = 1, x[sa[0]] = 0, i = 1; i < n; i++)
            x[sa[i]] = cmp(y, sa[i - 1], sa[i], j) ? p - 1 : p++;
    }
}
/**
height[i](i: 1 - n): height[1] == 0 (当最后一个元素补0的时候，这个等式成立)
height[i]表示sa[i-1]与sa[i]的lcp，于是当总共最后一个元素下标可以是n（总共n+1个元素）
*/
int height[M], rank[M];
void calheight(int *r, int *sa, int n)
{
    int i, j, k = 0;
    for (i = 0; i <= n; i++)
        rank[sa[i]] = i;
    for (i = 0; i < n; height[rank[i++]] = k)
        for (k ? k-- : 0, j = sa[rank[i] - 1]; r[i + k] == r[j + k]; k++);
    //    for(int i=1;i<=n;i++)
    //     cout<<height[i]<<" ";
    //    cout<<endl;
}
int f[M][20];

void makeRMQ(int n)
{
```

```
    int i, j, k, m;
    m = (int)(log(1.0 * n) / log(2.0));
    /**
    height是下标是1-n的，于是在1-n上做rmq
    */
    for (i = 1; i <= n; i++)
        f[i][0] = height[i];
    for (i = 1; i <= m; i++)
        for (j = n; j >= 1; j--)
        {
            f[j][i] = f[j][i - 1];
            k = 1 << (i - 1);
            if (j + k <= n)
                f[j][i] = min(f[j][i], f[j + k][i - 1]);
        }
}
int LCP(int x, int y)
{
    int m, t;
    x = rank[x];
    y = rank[y];
    if (x > y)
        t = x, x = y, y = t;
    x++;
    m = (int)(log(1.0 * (y - x + 1)) / log(2.0));
    return min(f[x][m], f[y - (1 << m) + 1][m]);
}

char str[M];
int num[M];
int main(void)
{
    while (~scanf("%s", str))
    {
        int len = strlen(str);
        for (int i = 0; i < len; i++)
            num[i] = str[i] - 'a' + 1;
        num[len] = 0;
        da(num, sa, len + 1, 128);
        calheight(num, sa, len);
        /**
        makeRMQ(len);
        int a,b;scanf("%d %d",&a,&b);
        cout<<LCP(a,b)<<endl;
        */
```

```
    }

    return 0;
}
```

13.2.2 倍增算法构造后缀数组

```
#include<iostream>
#include<cstring>
#include<cstdio>
#include<cstdlib>
#include<algorithm>
#include<cmath>
using std::sort;
int const M = 1001000;
int const N = 20100;
int rank[N], wb[M], wv[M], ws[M], height[N], num[N], sa[N];
int n, k;
int cmp(int *r, int a, int b, int l)
{
    return r[a] == r[b] && r[a + l] == r[b + l];
}
void da(int *r, int *sa, int n, int m)
{
    int i, j, p, *x = rank, *y = wb, *t;
    for (i = 0; i < m; i++) ws[i] = 0;
    for (i = 0; i < n; i++) ws[x[i] = r[i]]++;
    for (i = 1; i < m; i++) ws[i] += ws[i - 1];
    for (i = n - 1; i >= 0; i--) sa[--ws[x[i]]] = i;
    for (j = 1, p = 1; p < n; j *= 2, m = p)
    {
        for (p = 0, i = n - j; i < n; i++) y[p++] = i;
        for (i = 0; i < n; i++) if (sa[i] >= j) y[p++] = sa[i] - j;
        for (i = 0; i < n; i++) wv[i] = x[y[i]];
        for (i = 0; i < m; i++) ws[i] = 0;
        for (i = 0; i < n; i++) ws[wv[i]]++;
        for (i = 1; i < m; i++) ws[i] += ws[i - 1];
        for (i = n - 1; i >= 0; i--) sa[--ws[wv[i]]] = y[i];
        for (t = x, x = y, y = t, p = 1, x[sa[0]] = 0, i = 1; i < n; i++)
            x[sa[i]] = cmp(y, sa[i - 1], sa[i], j) ? p - 1 : p++;
    }
    return;
}

void calheight(int *r, int *sa, int n)
```

```
{
    int i, j, k = 0;
    for (i = 1; i <= n; i++) rank[sa[i]] = i;
    for (i = 0; i < n; height[rank[i++]] = k)
        for (k ? k-- : 0, j = sa[rank[i] - 1]; r[i + k] == r[j + k]; k++);
    return;
}
bool check(int len)
{
    int tot = 0;
    for (int i = 2; i <= n; i++)
    {
        if (height[i] < len)tot = 0;
        else
        {
            tot++;
            if (tot == k - 1)return true;
        }
    }
    return false;
}
int main()
{
    while (~scanf("%d %d", &n, &k))
    {
        for (int i = 0; i < n; i++)
        {
            scanf("%d", &num[i]);
        }
        num[n] = 0;
        da(num, sa, n + 1, M);
        calheight(num, sa, n);
        int ans = 0;
        int l = 0, r = n;
        while (l <= r)
        {
            int m = (l + r) >> 1;
            if (check(m))
                l = m + 1, ans = m;
            else
                r = m - 1;
        }
        printf("%d\n", ans);
    }
    return 0;
```

```
}
```

13.2.3 应用：子串多次出现在多个串中

```
#include <stdio.h>
#include <string.h>

#define maxn 101001
int wa[maxn], wb[maxn], wv[maxn], ws[maxn];
int cmp(int *r, int a, int b, int l)
{
    return r[a] == r[b] && r[a + l] == r[b + l];
}
void da(int *r, int *sa, int n, int m)
{
    int i, j, p, *x = wa, *y = wb, *t;
    for (i = 0; i < m; i++) ws[i] = 0;
    for (i = 0; i < n; i++) ws[x[i] = r[i]]++;
    for (i = 1; i < m; i++) ws[i] += ws[i - 1];
    for (i = n - 1; i >= 0; i--) sa[--ws[x[i]]] = i;
    for (j = 1, p = 1; p < n; j *= 2, m = p)
    {
        for (p = 0, i = n - j; i < n; i++) y[p++] = i;
        for (i = 0; i < n; i++) if (sa[i] >= j) y[p++] = sa[i] - j;
        for (i = 0; i < n; i++) wv[i] = x[y[i]];
        for (i = 0; i < m; i++) ws[i] = 0;
        for (i = 0; i < n; i++) ws[wv[i]]++;
        for (i = 1; i < m; i++) ws[i] += ws[i - 1];
        for (i = n - 1; i >= 0; i--) sa[--ws[wv[i]]] = y[i];
        for (t = x, x = y, y = t, p = 1, x[sa[0]] = 0, i = 1; i < n; i++)
            x[sa[i]] = cmp(y, sa[i - 1], sa[i], j) ? p - 1 : p++;
    }
    return;
}
int rank[maxn], height[maxn];
void calheight(int *r, int *sa, int n)
{
    int i, j, k = 0;
    for (i = 1; i <= n; i++) rank[sa[i]] = i;
    for (i = 0; i < n; height[rank[i++]] = k)
        for (k ? k-- : 0, j = sa[rank[i] - 1]; r[i + k] == r[j + k]; k++);
    return;
}

int len1, len, n, up, mx;
```

```
char s[maxn], s1[maxn];
int sa[maxn], a[maxn], cate[maxn];
int flag[110];

int check(int tlen)
{
    int i, j, k, cnt;
    i = j = 1;
    while (i <= len && j <= len)
    {
        for (k = 0; k < n; k++)
            flag[k] = 0;
        while (height[i] < tlen && i <= len)
            i++;
        j = i;
        while (height[j] >= tlen && j <= len)
            j++;
        if (j - i + 2 <= n / 2)
        {
            i = j;
            continue;
        }
        for (k = i - 1; k < j; k++)
        {
            if (cate[sa[k]] != -1) //差了这一句，wa了好久啊啊啊 嗄。。。
                flag[cate[sa[k]]] = 1;
        }
        for (cnt = 0, k = 0; k < n; k++)
            cnt += flag[k];
        if (cnt > n / 2)
            return 1;
        i = j;
    }
    return 0;
}
void print(int tlen)
{
    if (tlen == 0)
    {
        printf("?\n");
        return;
    }
    int i, j, k, cnt;
    i = j = 1;
    while (i <= len && j <= len)
```

```
    {
        for (k = 0; k < n; k++)
            flag[k] = 0;
        while (height[i] < tlen && i <= len)
            i++;
        j = i;
        while (height[j] >= tlen && j <= len)
            j++;
        if (j - i + 2 <= n / 2)
        {
            i = j;
            continue;
        }
        for (k = i - 1; k < j; k++)
        {
            if (cate[sa[k]] != -1)
                flag[cate[sa[k]]] = 1;
        }
        for (cnt = 0, k = 0; k < n; k++)
            cnt += flag[k];
        if (cnt > n / 2)
        {
            for (k = 0; k < tlen; k++)
                printf("%c", a[sa[i] + k] - 1);
            printf("\n");
        }
        i = j;
    }
}

void solve()
{
    int l, r, ans, mid;
    l = 0, r = mx;
    while (l <= r)
    {
        mid = (l + r) >> 1;
        if (check(mid))
            l = mid + 1;
        else
            r = mid - 1;
    }
    ans = r;
    //printf("%d.....\n", ans);//...
    print(ans);
```

```
}
int main()
{
    int i, j, k;
    scanf("%d", &n);
    while (1)
    {
        if (n == 0)
            break;
        up = 140;
        mx = 1;
        for (i = 0, j = 0; i < n; i++)
        {
            scanf("%s", s1);
            len1 = strlen(s1);
            if (len1 > mx)
                mx = len1;
            for (k = 0; k < len1; k++)
            {
                cate[j] = i;
                a[j++] = s1[k] + 1; //加1了的输出的时候注意一下。。
            }
            cate[j] = -1;
            a[j++] = up + i;
        }
        a[--j] = 0;
        len = j;

        if (n == 1)
        {
            printf("%s\n", s1);
            continue;
        }
        da(a, sa, len + 1, 250);
        calheight(a, sa, len);
        //for(i=0; i<=len; i++)//..................
        //  printf("i=%2d..%2d %2d..\n", i, sa[i], height[i]);
        solve();

        scanf("%d", &n);
        if (n == 0)
            break;
        else
            printf("\n");
    }
```

```
    return 0;
}
```

13.2.4 应用：不同子串个数

```
/**
每一个子串一定是某个后缀的前缀，那么问题便等价于求所有
后缀之间的不相同的前缀个数。我们按sa的顺序来考虑，当加入sa[k]
的时候，sa[k]这个后缀的长度为n-sa[k]，那么便有n-sa[k]个前缀，但是由heigh数组可
知sa[k]与sa[k-1]有height[k]个前缀是相同的，所以要除去，最终的答案便是sigma(n-sa[k]+height[k])
*/
#include<iostream>
#include<cstdio>
#include<cstring>
#include<queue>
#include<cmath>
#include<string>
#include<vector>
#include<algorithm>
#define maxn 50005
using namespace std;
//以下为倍增算法求后缀数组
int wa[maxn], wb[maxn], wv[maxn], Ws[maxn];
int cmp(int *r, int a, int b, int l)
{
    return r[a] == r[b] && r[a + l] == r[b + l];
}
void da(const char *r, int *sa, int n, int m)
{
    int i, j, p, *x = wa, *y = wb, *t;
    for (i = 0; i < m; i++) Ws[i] = 0;
    for (i = 0; i < n; i++) Ws[x[i] = r[i]]++;
    for (i = 1; i < m; i++) Ws[i] += Ws[i - 1];
    for (i = n - 1; i >= 0; i--) sa[--Ws[x[i]]] = i;
    for (j = 1, p = 1; p < n; j *= 2, m = p)
    {
        for (p = 0, i = n - j; i < n; i++) y[p++] = i;
        for (i = 0; i < n; i++) if (sa[i] >= j) y[p++] = sa[i] - j;
        for (i = 0; i < n; i++) wv[i] = x[y[i]];
        for (i = 0; i < m; i++) Ws[i] = 0;
        for (i = 0; i < n; i++) Ws[wv[i]]++;
        for (i = 1; i < m; i++) Ws[i] += Ws[i - 1];
        for (i = n - 1; i >= 0; i--) sa[--Ws[wv[i]]] = y[i];
        for (t = x, x = y, y = t, p = 1, x[sa[0]] = 0, i = 1; i < n; i++)
            x[sa[i]] = cmp(y, sa[i - 1], sa[i], j) ? p - 1 : p++;
```

```
    }
    return;
}
int sa[maxn], Rank[maxn], height[maxn];
//求height数组
void calheight(const char *r, int *sa, int n)
{
    int i, j, k = 0;
    for (i = 1; i <= n; i++) Rank[sa[i]] = i;
    for (i = 0; i < n; height[Rank[i++]] = k)
        for (k ? k-- : 0, j = sa[Rank[i] - 1]; r[i + k] == r[j + k]; k++);
    return;
}
char str[maxn];
int slove(int n)
{
    int sum = 0;
    for (int i = 1; i <= n; i++)
        sum += n - sa[i] - height[i];
    return sum;
}
int main()
{
    int t;
    scanf("%d", &t);
    while (t--)
    {
        scanf("%s", str);
        da(str, sa, strlen(str) + 1, 130);
        calheight(str, sa, strlen(str));
        for (int i = 1; i <= strlen(str); i++)
            printf("%d %d\n", sa[i], height[i]);
        printf("%d\n", slove(strlen(str)));
    }
    return 0;
}
```

13.2.5 应用：重复次数最多的连续子串

```
/**
poj 3693
题意:
给一个串, 求重复次数最多的连续子串（并且字典序最小,输出整个重复的字符串）
比如:
ccabababc
```

```
daabbccaa
Case 1: ababab
Case 2: aa
**/
#include <string.h>
#include <math.h>
#include <stdio.h>
#include <iostream>
using namespace std;
const int M = 200000;
int wa[M], wb[M], wv[M], wsum[M];
int height[M], sa[M], rank[M];
int n, ans, len, pos;
char str[M];
int R[M];
int f[M][20];
int a[M], num;
int cmp(int *r, int a, int b, int l)
{
    return r[a] == r[b] && r[a + l] == r[b + l];
}
/**
sa[i](i: 0 - n,接进来如果是0-n-1的话，最后一个元素是补0的)范围是0-n;
sa[0] == n (如果最后元素补的是0)
*/
void da(int *r, int *sa, int n, int m)
{
    int i, j, p, *x = wa, *y = wb, *t;
    for (i = 0; i < m; i++)
        wsum[i] = 0;
    for (i = 0; i < n; i++)
        wsum[x[i] = r[i]]++;
    for (i = 1; i < m; i++)
        wsum[i] += wsum[i - 1];
    for (i = n - 1; i >= 0; i--)
        sa[--wsum[x[i]]] = i;
    for (j = 1, p = 1; p < n; j *= 2, m = p)
    {
        for (p = 0, i = n - j; i < n; i++)
            y[p++] = i;
        for (i = 0; i < n; i++)
            if (sa[i] >= j)
                y[p++] = sa[i] - j;
        for (i = 0; i < n; i++)
            wv[i] = x[y[i]];
```

```
        for (i = 0; i < m; i++)
            wsum[i] = 0;
        for (i = 0; i < n; i++)
            wsum[wv[i]]++;
        for (i = 1; i < m; i++)
            wsum[i] += wsum[i - 1];
        for (i = n - 1; i >= 0; i--)
            sa[--wsum[wv[i]]] = y[i];
        for (t = x, x = y, y = t, p = 1, x[sa[0]] = 0, i = 1; i < n; i++)
            x[sa[i]] = cmp(y, sa[i - 1], sa[i], j) ? p - 1 : p++;
    }
}
/**
height[i](i: 1 - n): height[1] == 0 (当最后一个元素补0的时候，这个等式成立)
height[i]表示sa[i-1]与sa[i]的lcp，于是当总共最后一个元素下标可以是n（总共n+1个元素）
*/
void calheight(int *r, int *sa, int n)
{
    int i, j, k = 0;
    for (i = 0; i <= n; i++)
        rank[sa[i]] = i;
    for (i = 0; i < n; height[rank[i++]] = k)
        for (k ? k-- : 0, j = sa[rank[i] - 1]; r[i + k] == r[j + k]; k++);
    //    for(int i=1;i<=n;i++)
    //     cout<<height[i]<<" ";
    //    cout<<endl;
}
int mmin(int x, int y)
{
    return x < y ? x : y;
}
void rmqinit(int n)
{
    int i, j, k, m;
    m = (int)(log(1.0 * n) / log(2.0));
    /**
    height是下标是1-n的，于是在1-n上做rmq
    */
    for (i = 1; i <= n; i++)
        f[i][0] = height[i];
    for (i = 1; i <= m; i++)
        for (j = n; j >= 1; j--)
        {
            f[j][i] = f[j][i - 1];
```

```
            k = 1 << (i - 1);
            if (j + k <= n)
                f[j][i] = mmin(f[j][i], f[j + k][i - 1]);
        }
}
int get_rmq(int x, int y)
{
    int m, t;
    x = rank[x];
    y = rank[y];
    if (x > y)
        t = x, x = y, y = t;
    x++;
    m = (int)(log(1.0 * (y - x + 1)) / log(2.0));
    return mmin(f[x][m], f[y - (1 << m) + 1][m]);
}
int main()
{
    int i, j, k, ca = 0, l, s, t, p, cnt;
    while (~scanf("%s", str))
    {
        if (str[0] == '#')
            break;
        n = strlen(str);
        for (i = 0; i < n; i++)
            R[i] = str[i] - 'a' + 1;
        R[n] = 0;
        da(R, sa, n + 1, 28);
        calheight(R, sa, n);
        rmqinit(n);
        ans = 1;
        num = 0;
        pos = 0;
        for (l = 1; l <= n / 2; l++)
        {
            for (i = 0; i < n - l; i += l)
            {
                if (str[i] != str[i + l])
                    continue;
                k = get_rmq(i, i + l);
                s = k / l + 1;
                p = i;
                t = l - k % l;
                cnt = 0;
                for (j = i - 1; j >= 0 && j > i - l && str[j] == str[j + l]; j--)
```

```
                {
                    cnt++;
                    if (cnt == t)
                    {
                        s++;
                        p = j;
                    }
                    else if (rank[j] < rank[p])
                    {
                        p = j;
                    }
                }
                if (ans < s)
                {
                    pos = p;
                    len = s * l;
                    ans = s;

                }
                else if (ans == s && rank[pos] > rank[p])
                {
                    pos = p;
                    len = s * l;
                }

            }
        }
        printf("Case %d: ", ++ca);
        if (ans < 2)
        {
            char c = 'z';
            for (i = 0; i < n; i++)
                if (str[i] < c)
                    c = str[i];
            printf("%c\n", c);
            continue;
        }
        for (i = 0; i < len; i++)
            printf("%c", str[i + pos]);
        puts("");
    }
    return 0;
}
```

13.2.6 应用：最长不重复子串

```
#include<stdio.h>
#include<string.h>
#include<iostream>
#include<cstdio>
#include<cmath>
#include<vector>
#include<cstring>
using namespace std;

const int nMax = 1000012;

int  num[nMax];
int sa[nMax], rank[nMax], height[nMax];
int wa[nMax], wb[nMax], wv[nMax], wd[nMax];
int mmin(int a, int b)
{
    if (a > b) return b;
    return a;
}
int cmp(int *r, int a, int b, int l)
{
    return r[a] == r[b] && r[a + l] == r[b + l];
}

void da(int *r, int n, int m)          //  倍增算法 r为待匹配数组  n为总长度 m为
字符范围
{
    int i, j, p, *x = wa, *y = wb, *t;
    for (i = 0; i < m; i ++) wd[i] = 0;
    for (i = 0; i < n; i ++) wd[x[i] = r[i]] ++;
    for (i = 1; i < m; i ++) wd[i] += wd[i - 1];
    for (i = n - 1; i >= 0; i --) sa[-- wd[x[i]]] = i;
    for (j = 1, p = 1; p < n; j *= 2, m = p)
    {
        for (p = 0, i = n - j; i < n; i ++) y[p ++] = i;
        for (i = 0; i < n; i ++) if (sa[i] >= j) y[p ++] = sa[i] - j;
        for (i = 0; i < n; i ++) wv[i] = x[y[i]];
        for (i = 0; i < m; i ++) wd[i] = 0;
        for (i = 0; i < n; i ++) wd[wv[i]] ++;
        for (i = 1; i < m; i ++) wd[i] += wd[i - 1];
        for (i = n - 1; i >= 0; i --) sa[-- wd[wv[i]]] = y[i];
        for (t = x, x = y, y = t, p = 1, x[sa[0]] = 0, i = 1; i < n; i ++)
        {
```

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

void calHeight(int *r, int n)            //  求height数组。
{
    int i, j, k = 0;
    for (i = 1; i <= n; i ++) rank[sa[i]] = i; // 1->n
    for (i = 0; i < n; i++)
    {
        for (k ? k -- : 0, j = sa[rank[i] - 1]; r[i + k] == r[j + k]; k ++);
        height[rank[i]] = k;
    }
}

int Log[nMax];
int best[20][nMax];//best[i][j] 表示从j开始的长度为2的i次方的一段元素的最小值
void initRMQ(int n)
{
    //初始化RMQ
    int  i, j;
    for (i = 1; i <= n ; i ++) best[0][i] = height[i];
    for (i = 1; i <= Log[n] ; i ++)
    {
        int limit = n - (1 << i) + 1;
        for (j = 1; j <= limit ; j ++)
        {
            best[i][j] = mmin(best[i - 1][j] , best[i - 1][j + (1 << i>>1)]);
        }
    }
}
int lcp(int a, int b) //询问a,b后缀的最长公共前缀
{
    a = rank[a];    b = rank[b];
    if (a > b) swap(a, b);
    a ++;
    int t = Log[b - a + 1];
    return mmin(best[t][a] , best[t][b - (1 << t) + 1]);
}

void get_log()
{
    int i;
    Log[0] = -1;
```

```
    for (i = 1; i <= nMax; i++)
    {
        // 求log2,这么强大的位运算。。
        Log[i] = (i & (i - 1)) ? Log[i - 1] : Log[i - 1] + 1 ;
    }
}
char str[nMax];
int ans[nMax];
int n;
int a[nMax];
int solve(int x)
{
    ///注意通过判断sa[i]-sa[i-1]>=k决定不重复长度是否大于k是不行的 因为有可能有好
几个重复的排在一起
    ///对于每个都不大于k 但是最后一个和第一个的距离是大于k的
    ///注意sa[i],sa[i-1]不一定哪个大那个小
    int i, mx, mn;
    mx = 0, mn = nMax;
    for (i = 1; i <= n; i++)
    {
        if (height[i] >= x)
        {
            mx = max(mx, sa[i]);
            mn = min(mn, sa[i]);
            if (mx - mn >= x) return 1;
        }
        else
        {
            mx = mn = sa[i];
        }
    }

    return 0;
}

int main()
{
    int i, j;
    get_log();
    while (scanf("%d", &n) != EOF)
    {
        if (!n) break;
        for (i = 0; i < n; i++) scanf("%d", &a[i]);
        // n--;
        for (i = 1; i < n; i++)
```

```
        {
            num[i] = a[i] - a[i - 1] + 100; //加100防止出现负数
        }
        num[n] = 0;
        da(num, n + 1, 300); //这里要开大一点  300
        calHeight(num, n);
        initRMQ(n);
        /*
          for(i=0; i<n+1; i++)  // rank[i] : suffix(i)排第几
             printf("rank[%d] =  %d\n",i,rank[i]);
          printf("\n");
          for(i=0; i<n+1; i++)   // sa[i] : 排在第i个的是谁
             printf("sa[%d] =  %d\n",i,sa[i]);
         */
        for (i = 0; i < n + 1; i++)
            printf("%d ", height[i]);
        printf("\n");
        int left, right, mx = 0, mid;
        left = 4; right = n / 2 + 1;
        while (left <= right)
        {
            mid = (left + right) / 2;
            if (solve(mid) && mid > mx)
            {
                mx = mid;
                left = mid + 1;
            }
            else
            {
                right = mid - 1;
            }
        }
        if (mx == 0)
        {
            printf("0\n");
            continue;
        }
        printf("%d\n", mx + 1);
    }
    return 0;
}
```

§ 13.3 AC自动机

```
#include<cstring>
```

```
#include<queue>
#include<cstdio>
#include<map>
#include<string>
using namespace std;
const int  MN = 90000 ;    ///儿子的最大个数，意思为：每一个元素节点的取值范围
const int R = 26;
//const int MAXS = 150 + 10;

struct AhoCorasickAutomata
{
    int ch[MN][R];
    int f[MN];    /// fail函数
    int val[MN];  /// 每个字符串的结尾结点都有一个非0的val
    int last[MN]; /// 输出链表的下一个结点
    //    int cnt[MAXS];
    int sz;

    void init()
    {
        sz = 1;///只有跟，根从0开始
        memset(ch[0], 0, sizeof(ch[0]));
    }

    /// 字符c的编号
    int idx(char c)
    {
        return c - 'a';
    }

    /// 插入字符串。v必须非0
    void insert(char *s, int v)
    {
        int u = 0, n = strlen(s);
        for (int i = 0; i < n; i++)
        {
            int c = idx(s[i]);
            if (!ch[u][c])
            {
                memset(ch[sz], 0, sizeof(ch[sz]));
                val[sz] = 0;
                ch[u][c] = sz++;
            }
            u = ch[u][c];
        }
```

```
        ///
        val[u] = v;
        ///根据题目填写后面的附件情况
        //        ms[string(s)] = v;
    }

    /// 递归打印以结点j结尾的所有字符串,根据题目而定
    void print(int j)
    {
        if (j)
        {
            //            cnt[val[j]]++;
            print(last[j]);
        }
    }

    /// 在T中找模板
    int find(char *T)
    {
        int n = strlen(T);
        int j = 0; // 当前结点编号, 初始为根结点
        for (int i = 0; i < n; i++)  // 文本串当前指针
        {
            int c = idx(T[i]);
            //            while(j && !ch[j][c]) j = f[j]; // 顺着细边走, 直到可
以匹配
            j = ch[j][c];
            if (val[j]) print(j);
            else if (last[j]) print(last[j]); // 找到了!
        }
    }

    /// 计算fail函数
    void getFail()
    {
        queue<int> q;
        f[0] = 0;
        /// 初始化队列
        for (int c = 0; c < R; c++)
        {
            int u = ch[0][c];
            if (u)
            {
                f[u] = 0;
                q.push(u);
```

```
                last[u] = 0;
            }
        }
        /// 按BFS顺序计算fail
        while (!q.empty())
        {
            int r = q.front();
            q.pop();
            for (int c = 0; c < R; c++)
            {
                int u = ch[r][c];
                if (!u)
                {
                    ch[r][c] = ch[f[r]][c];
                    continue;
                }
                q.push(u);
                int v = f[r];
                while (v && !ch[v][c]) v = f[v];
                f[u] = ch[v][c];
                last[u] = val[f[u]] ? f[u] : last[f[u]];
            }
        }
    }
};
```

§ 13.4 回文串

13.4.1 判回文串

字符串哈希，正着来一遍与反着来一遍的哈希值相同，则为回文串

更改字符，判任意区间是否为回文串(线段树+字符串哈希)(Ural 1989)

```
/*
操作:
1.更改字符串的任意字符
2.询问从l到r是否为回文串
*/
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;
const int M = 1e5 + 10;
```

```
unsigned int ltor[M<<2],rtol[M<<2];
int L,R,top;
char s[M];
unsigned int seed = 131;
const unsigned int Ha = 0x7FFFFFFF;
unsigned int ha[M];
void init()
{
    ha[0] = 1;
    for(int i = 1; i<M; i++)
    {
        ha[i] = seed * ha[i-1];
        ha[i] &= Ha;
    }
}
void pushup(int rt,int len)
{
    ltor[rt] = ltor[rt<<1] + ((ha[len-(len>>1)]*ltor[rt<<1|1])&Ha);
    ltor[rt] &= Ha;
    rtol[rt] = ( (rtol[rt<<1]*ha[len>>1])&Ha) + rtol[rt<<1|1];
    rtol[rt] &= Ha;
    return;
}
void build(int l,int r,int rt)
{
    if(l==r)
    {
        ltor[rt] = rtol[rt] = s[top++] - 'a';
        return;
    }
    ltor[rt] = rtol[rt] = 0;
    int m = (l+r)>>1;
    build(l,m,rt<<1);
    build(m+1,r,rt<<1|1);
    pushup(rt,r-l+1);
}
void debug(int l,int r,int rt)
{
    cout<<l<<" "<<r<<" "<<ltor[rt]<<" "<<rtol[rt]<<endl;
    if(l==r)
    {
        return;
    }
    int m = (l+r)>>1;
    debug(l,m,rt<<1);
```

```
    debug(m+1,r,rt<<1|1);
}
void update(int l,int r,int rt,int pos,int val)
{
    //cout<<l<<" "<<r<<" "<<ltor[rt]<<" "<<rtol[rt]<<endl;
    if(l==r)
    {
        ltor[rt] = rtol[rt] = val;
        return;
    }
    int m=(l+r)>>1;
    if(pos<=m)update(l,m,rt<<1,pos,val);
    else update(m+1,r,rt<<1|1,pos,val);
    pushup(rt,r-l+1);
}
unsigned int query(int l,int r,int rt,int kind)
{
    //cout<<l<<" "<<r<<" "<<ltor[rt]<<" "<<rtol[rt]<<endl;
    if(L<=l && R>=r)
    {
        if(kind)return ((ltor[rt]*ha[l-L])&Ha);
        else return ((rtol[rt]*ha[R-r])&Ha);
    }
    int m = (l+r)>>1;
    unsigned int ret = 0;
    if(L <= m)
    {
        ret += query(l,m,rt<<1,kind);
        //cout<<"ret: "<<ret<<endl;
        ret &= Ha;
    }
    if(m<R) ret += query(m+1,r,rt<<1|1,kind);
    return ret &= Ha;
}
int main(void)
{
//    freopen("in.cpp","r",stdin);
    init();
    scanf("%s",s);
    top = 0;
    int len = strlen(s);
    build(1,len,1);
    //debug(1,len,1);
    int q;
    scanf("%d",&q);
```

```
    for(int i = 1; i<=q; i++)
    {
        char t[20];
        scanf("%s",t);
        if(t[0] == 'p')
        {
            scanf("%d%d",&L,&R);
            unsigned int ltr = query(1,len,1,0);
            unsigned int rtl = query(1,len,1,1);
            //cout<<ltr<<" "<<rtl<<endl;
            if(ltr == rtl)printf("Yes\n");
            else printf("No\n");
        }
        else
        {
            int pos;
            scanf("%d",&pos);
            char st[20];
            scanf("%s",st);
            update(1,len,1,pos,st[0]-'a');
            // debug(1,len,1);
        }
    }
    return 0;
}
```

13.4.2 最长回文串

```
#include<vector>
#include<iostream>
using namespace std;

const int N = 300010;
int n, p[N];
char s[N], str[N];

#define _min(x, y) ((x)<(y)?(x):(y))

void kp()
{
    int i;
    int mx = 0;
    int mx = 0;
    int id;
    for (i = n; str[i] != 0; i++)
```

```
        str[i] = 0;
    for (i = 1; i < n; i++)
    {
        if ( mx > i )
            p[i] = _min( p[2 * id - i], p[id] + id - i );
        else
            p[i] = 1;
        for (; str[i + p[i]] == str[i - p[i]]; p[i]++)
            ;
        if ( p[i] + i > mx )
        {
            mx = p[i] + i;
            id = i;
        }
    }
}

void init()
{
    int i, j, k;
    str[0] = '$';
    str[1] = '#';
    for (i = 0; i < n; i++)
    {
        str[i * 2 + 2] = s[i];
        str[i * 2 + 3] = '#';
    }
    n = n * 2 + 2;
    s[n] = 0;
}

int main()
{
    int i, ans;
    while (scanf("%s", s) != EOF)
    {
        n = strlen(s);
        init();
        kp();
        for (int i = 0; i < )
            ans = 0;
        for (i = 0; i < n; i++)
            if (p[i] > ans)
                ans = p[i];
        printf("%d\n", ans - 1);
```

```
    }
    return 0;
}



//最长回文子串模板



//hdu3068，最长回文子串模板，Manacher算法，时间复杂度O(n)，相当快

#include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;
#define M 20000050

char str1[M], str[2 * M]; //start from index 1
int rad[M], nn, n;

void Manacher(int *rad, char *str, int n)/*str是这样一个字符串（下标从1开始）:
举例：若原字符串为"abcd"，则str为"$#a#b#c#d#"，最后还有一个终止符。
n为str的长度，若原字符串长度为nn，则n=2*nn+2。
rad[i]表示回文的半径，即最大的j满足str[i-j+1...i] = str[i+1...i+j]，
而rad[i]-1即为以str[i]为中心的回文子串在原串中的长度*/
{
    int i;
    int mx = 0;
    int id;
    for (i = 1; i < n; i++)
    {
        if ( mx > i )
            rad[i] = rad[2 * id - i] < mx - i ? rad[2 * id - i] : mx - i;
        else
            rad[i] = 1;
        for (; str[i + rad[i]] == str[i - rad[i]]; rad[i]++)
            ;
        if ( rad[i] + i > mx )
        {
            mx = rad[i] + i;
            id = i;
        }
    }
}
```

```
int main()
{
    int i, ans, Case = 1;
    while (scanf("%s", str1) != EOF)
    {
        nn = strlen(str1);
        n = 2 * nn + 2;
        str[0] = '$';
        for (i = 0; i <= nn; i++)
        {
            str[2 * i + 1] = '#';
            str[2 * i + 2] = str1[i];
        }
        Manacher(rad, str, n);
        ans = 1;
        for (i = 0; i < n; i++)
            ans = rad[i] > ans ? rad[i] : ans;
        printf("%d\n", ans - 1);
    }
    return 0;
}


//扩展kmp版，时间复杂度O(nlogn)，稍慢

//hdu3068
/*给一个w长的字符串，求最长回文子串的长度
解题思路：不是传说中的高深的后缀数组，而是扩展的kmp算法。
扩展kmp中，模式串与主串都有一个next向量，相应的next[i],记录该串的后缀与模式串的最大匹配数，关于扩展kmp的具体实现这里就不说了，现在只说一下解此题的思路：
令所给字符串为a，则找到中点mid位置，一后半段做为模式串，把a倒过来生成b，求出b串在该模式串的每一位的next1[i];在以字符串a的前半段的逆串作为模式串，求出a串在该模式串下的next2。然后遍历a串的每一位，根据next1和next2就可以判断此处是否回文，但这个回文只能判断跨越中点mid的回文，因此这里要进行划分递归，分别在a的前半段和后半段重复此算法即可找到最大回文串。
*/

#include<iostream>
#include<cstring>
#include<cstdio>
using namespace std;
#define N 110005

char tmp1[N], tmp2[N], s[N], tt[N];
```

```
int tmp[N], ans, ne1[N], ne2[N];

char *rev(char *str, int ll)
{
    for (int i = 0; i < ll; i++)
        tt[i] = str[ll - i - 1];
    tt[ll] = 0;
    return tt;
}

void getA(char *pat, int *aa)
{
    int j = 0;
    while (pat[1 + j] && pat[j] == pat[1 + j])
        j++;
    aa[1] = j;
    int k = 1;
    int len, l;
    for (int i = 2; pat[i]; i++)
    {
        len = k + aa[k];
        l = aa[i - k];
        if (l < len - i)
            aa[i] = l;
        else
        {
            j = (0 > len - i) ? 0 : len - i;
            while (pat[i + j] && pat[j] == pat[i + j])
                j++;
            aa[i] = j;
            k = i;
        }
    }
}

void getB(char *pat, char *str, int *bb, int length)
{
    getA(pat, tmp);
    int j = 0;
    while (pat[j] && str[j] && pat[j] == str[j])
        j++;
    bb[0] = j;
    int k = 0;
    int len, l;
    for (int i = 1; i < length; i++)
```

```
    {
        len = k + bb[k];
        l = tmp[i - k];
        if (l < len - i)
            bb[i] = l;
        else
        {
            j = (0 > len - i) ? 0 : len - i;
            while (i + j < length && pat[j] && pat[j] == str[i + j])
                j++;
            bb[i] = j;
            k = i;
        }
    }
}

void find(int l, int r)
{
    if (r - l + 1 <= ans)
        return;
    int x;
    int mid = (l + r) / 2;
    strncpy(tmp1, s + l, mid - l + 1);
    tmp1[mid - l + 1] = 0;
    strncpy(tmp2, s + mid + 1, r - mid);
    tmp2[r - mid] = 0;
    getB(tmp2, rev(s + l, r - l + 1), ne1, r - l + 1);
    getB(rev(tmp1, mid - l + 1), s + l, ne2, r - l + 1);
    ne1[r - l + 1] = ne2[r - l + 1] = 0;
    for (int i = l; i <= mid; i++)
    {
        if (ne2[i - l] * 2 >= mid - i + 1)
        {
            x = mid - i + 1 + ne1[r - i + 1] * 2;
            if (ans < x)
                ans = x;
        }
    }
    if (ans < 2 * ne2[mid + 1 - l])
        ans = 2 * ne2[mid + 1 - l];
    for (int i = mid + 1; i <= r; i++)
    {
        if (ne1[r - i] * 2 >= i - mid)
        {
            x = i - mid + ne2[i - l + 1] * 2;
```

```
            if (ans < x)
                ans = x;
        }
    }
    find(l, mid);
    find(mid + 1, r);
}

int main()
{
    while (scanf("%s", s) != EOF)
    {
        ans = 1;
        find(0, strlen(s) - 1);
        printf("%d\n", ans);
    }
    return 0;
}

//下面注释掉的也对，只要把上面的rev函数换成下面的rev函数，并把find函数换成下面的solve函数就可以了


int nextb[N], nexta1[N], nexta2[N];
char a[N], b[N];

void rev(char *a, int len)
{
    char t;
    for (int i = 0; i < len / 2; i++)
        t = a[i], a[i] = a[len - 1 - i], a[len - 1 - i] = t;
}
void solve(char *a, int len)
{
    if (len <= ans || len < 2)return ;
    int mid = len >> 1;
    int i, j, k;
    for (i = mid; i < len; i++) b[i - mid] = a[i];
    b[i - mid] = 0;
    rev(a, len);
    getB(b, a, nexta1, len);
    rev(a, len);
    for (i = 0; i < mid; i++) b[i] = a[mid - i - 1];
    b[i] = 0;
    getB(b, a, nexta2, len);
```

```
    nexta1[len] = nexta2[len] = 0;
    for (i = 0; i < mid; i++)
    {
        if (nexta2[i] >= (mid - i) / 2)
        {
            int x = mid - i + 2 * nexta1[len - i];
            if (x > ans)ans = x;
        }
    }
    for (i = mid; i < len; i++)
    {
        if (nexta1[len - i] >= (i - mid) / 2)
        {
            int x = i - mid + 2 * nexta2[i];
            if (x > ans)ans = x;
        }
    }
    solve(a, mid - 1);
    solve(a + mid, len - mid);
}
int main()
{
    int i, j, k;
    while (scanf("%s", a) != EOF)
    {
        ans = 1;
        solve(a, strlen(a));
        printf("%d\n", ans);
    }
}
```

§ 13.5 Other

13.5.1 字符串哈希(推荐BKDR)

字符串哈希算法

普通字符串哈希算法

```
// ELF Hash
unsigned int ELFHash(char *str)
{
    unsigned int hash = 0, x = 0;
    while (*str)
    {
        hash = (hash << 4) + (*str++);
```

```
        if ((x = hash & 0xF0000000L) != 0)
        {
            hash ^= (x >> 24);
            hash &= ~x;
        }
    }
    return (hash & 0x7FFFFFFF);
}
// BKDR Hash
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);
}
// DJB Hash
unsigned int DJBHash(char *str)
{
    unsigned int hash = 5381;
    while (*str)hash += (hash << 5) + (*str++);
    return (hash & 0x7FFFFFFF);
}
// AP Hash
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);
}
```

$O(N)$预处理$O(1)$查询的字符串哈希

刘汝佳大白书P225

13.5.2 最小表示法 & 最大表示法

```
/**
*最小表示法
*/
```

```
int getmin()
{
    int len = strlen(pat);
    int i=0,j=1,k=0;
    while(i<len && j<len && k<len)
    {
        int t = pat[(i+k)%len] - pat[(j+k)%len];
        if(!t) k++;
        else
        {
            if(t>0) i = i+k+1;
            else j = j+k+1;
            if(i == j) j++;
            k = 0 ;
        }
    }
    return i<j?i:j;
}
/**
*最大表示法
*/
int getmax()
{
    int len = strlen(pat);
    int i=0,j=1,k=0;
    while(i<len && j<len && k<len)
    {
        int t = pat[(i+k)%len] - pat[(j+k)%len];
        if(!t) k++;
        else
        {
            if(t>0) j = j+k+1;
            else i = i+k+1;
            if(i == j) j++;
            k = 0 ;
        }
    }
    return i<j?i:j;
}
```

13.5.3 找某个字符串的不同子串的数目($O(N^2)$)

```
/**
*找某个字符串的不同子串的数目($O(N^2)$)
*输入: str
```

```
*输出：不同子串的数目
*/
int f(string str)
{
    set<string> st;
    st.clear();
    for(int i=0;i<st.size();i++)
        for(int j=1;i+j<st.size();j++)
            st.insert(str.substr(i,j));
    return st.size();
}
```

Chapter 14

树

§ 14.1 树的分治

14.1.1 树的重心：POJ 1655 Balancing Art

解题思路，树形dp balance[i]表示i结点的平衡度balance[i] = max(node(k), node(k))表示以i的孩子结点k为根的子树的总结点个数，其中i为整棵树的根结点可以规定节点1为这棵树的根结点，进行深度遍历，那么就形成了一个有向树，在深度遍历的过程中，求出每个结点的出度，根据出度进行拓扑排序，将出度为0的点入队列深度遍历的过程中求出以每个结点为根的子树的总个数node[k]，blanance[k]初始化为num - node[k]（num为整棵树的总结点个数）遍历完毕后，按拓扑排序求balance[i] = max(node[k],balance[i])，k为i的孩子结点

```
/**
*树的重心：POJ 1655 Balancing Art
*题目大意：给一个树，删除其中一个点就会形成一个森林，
*点的平衡度为删除了这个节点后，所形成多个树，
*其中组成树的节点最多，节点个数就是那个平衡度。
*要你求出最小平衡度，输出这个节点和平衡度，
*要是有多个节点的平衡度一样，输出节点序号最小。
*输出树的重心和最大的子树的节点数
*/
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <queue>
using namespace std;

struct nodes
{
    int u, next;
};

const int maxn = 20010;
```

```
int t, e, num, node[maxn], n, parent[maxn], balance[maxn], degree[maxn], head[maxn];
nodes edge[maxn * 10];
queue<int> q;

void addEdge(int u, int v);
void dfs(int u, int pre);

int main()
{
    scanf("%d", &t);
    while (t-- != 0)
    {
        memset(node, 0, sizeof(node));
        memset(balance, 0, sizeof(balance));
        memset(degree, 0, sizeof(degree));
        memset(head, -1, sizeof(head));
        e = 0;
        while (!q.empty())
            q.pop();
        scanf("%d", &num);
        for (int i = 0; i < num - 1; i++)
        {
            int u, v;
            scanf("%d %d", &u, &v);
            addEdge(u, v);
        }
        int ans = 0x7fffffff, tnode;
        dfs(1, -1);
        while (!q.empty())
        {
            int u = q.front();
            q.pop();
            if (ans > balance[u])
            {
                ans = balance[u];
                tnode = u;
            }
            else if (ans == balance[u])
                tnode = min(tnode, u);
            if (parent[u] != -1)
            {
                balance[parent[u]] = max(balance[parent[u]], node[u]);
                degree[parent[u]]--;
                if (degree[parent[u]] == 0)
```

```
                q.push(parent[u]);
            }
        }

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

    return 0;
}

void addEdge(int u, int v)
{
    edge[e].u = v;
    edge[e].next = head[u];
    head[u] = e++;

    edge[e].u = u;
    edge[e].next = head[v];
    head[v] = e++;
}

void dfs(int u, int pre)
{
    parent[u] = pre;
    node[u] = 1;
    if (pre != -1)
        degree[pre]++;
    for (int i = head[u]; i != -1; i = edge[i].next)
    {
        int v = edge[i].u;
        if (v != pre)
        {
            dfs(v, u);
            node[u] += node[v];
        }
    }
    balance[u] = num - node[u];
    if (degree[u] == 0)
        q.push(u);
}
```

一开始想到的是模仿求树的直径那样子去Dp，两次DFS。son[i] - 结点i的儿子结点数目第一遍求出son；h[i] - 结点i向上的结点数目h[i] = h[k] + son[k] - son[i] - 1；blance = max(son[j] , h[i]) 第二遍求出h，和blance；

后来去看题解，才发现有更简单的方法。应用一个性质，h[i] = n - son[i] -1；blance = max(son[j] , n - son[i] -1)；这样只需要一次DFS。

```
#include <cstdio>
#include <iostream>
#include <fstream>
#include <cstring>
#include <string>
#include <vector>
#define OP(s) cout<<#s<<"="<<s<<" ";
#define PP(s) cout<<#s<<"="<<s<<endl;
using namespace std;
int n;
vector <int> adj[20010];

int son[20010];
bool vd[20010];
int ans,asize = 1<<29;
void DFS(int s)
{
    vd[s] = 1;
    son[s] = 0;
    int blance = 0;
    int size = adj[s].size();
    for (int j = 0;j < size;j++)
    {
        int u = adj[s][j];
        if (vd[u]) continue;
        DFS(u);
        son[s] += son[u]+1;
        blance = max(blance,son[u]+1);
    }
    blance = max(blance,n - son[s] - 1);
    if (blance < asize || blance == asize && s < ans)
        ans = s,asize = blance;
}

int main()
{
//    freopen("test.txt","r",stdin);
    int T;
    cin>>T;
    while(T--)
    {
        cin>>n;
        for (int i = 1;i <= n;i++) adj[i].clear();
        for (int i = 1;i <= n-1;i++)
        {
```

```
            int u,v;
            scanf("%d%d",&u,&v);
            adj[u].push_back(v);
            adj[v].push_back(u);
        }

        memset(vd,0,sizeof(vd));
        asize = 1<<29;
        DFS(1);
        cout<<ans<<" "<<asize<<endl;
    }

    return 0;
}
```

14.1.2 树链剖分

HDU 5029 Relief grain

```
/*
题目分析：这题的方法太美妙了！
首先看到这是一颗树，那么很容易想到树链剖分。然后想到可以将查询排个序，然后每一种查询执行完以后
更新每个点的最优值。但是这样进行的复杂度太高！尤其是当z给的没有一样的时候尤其如此。
那么我们是否可以找到更加高效的算法？
答案是肯定的！
先简化一下问题，如果这些操作是在线段上进行的，我们怎么求解？
我们很容易可以想到标记法：区间【L，R】染上颜色X，
则位置L标记为X，表示从L开始染色X，位置R+1标记为-X，表示从R+1开始结束染色。
用邻接表保存当前位置上的标记，然后从左往右，每到一个点上就把所有的标记执行了，为此我们建立
一棵权值线段树，+X，我们就在位置X上+1，-X，我们就在位置X上-1，然后用maxv标记记录区间最大值。
执行完这些操作以后就是查询了，顺着maxv == 最大值走，尽量往左走就行了。
然后我们回到本题，完全就是一个套路啊！只不过把连续的区间分成logN个不是
连续的区间而已。还是按照上面的操作就可以了，不过要注意扫描时的点是线段树上的点，而不是原来的点，还要映射回去。
而且由于本题的特殊性，我们直接将线段树改成非递归效率更佳
*/
#pragma comment(linker, "/STACK:102400000,102400000")
#include <cstdio>
#include <cstring>
#include <set>
#include <algorithm>
#include <string>
#include <queue>
```

```
#include <vector>
using namespace std;
const int M = 100003;

struct Edge
{
    int u, v, next;
    Edge(int _u, int _v, int _next): u(_u), v(_v), next(_next) {}
    Edge() {}
} edge[M << 1];
int head[M], edgenum, n, m;

void addedge(int u, int v)
{
    edge[edgenum] = Edge(u, v, head[u]);
    head[u] = edgenum++;
}
int hson[M], size[M], fa[M], dep[M];
void dfs(int u, int p)
{
    if (p == 0) dep[u] = 0;
    fa[u] = p; dep[u] = dep[p] + 1;
    size[u] = 1; hson[u] = -1;
    for (int i = head[u]; i != -1; i = edge[i].next)
    {
        int v = edge[i].v;
        if (v == p)continue;
        dfs(v, u);
        size[u] += size[v];
        if (hson[u] == -1 || size[v] > size[ hson[u] ]) hson[u] = v;
    }
}
int seq[M], reseq[M], top[M], num;
void link(int u, int tp)
{
    seq[u] = ++num;
    reseq[ seq[u] ] = u;
    top[u] = tp;
    if (hson[u] == -1)return;
    link(hson[u], tp);
    for (int i = head[u]; i != -1; i = edge[i].next)
    {
        int v = edge[i].v;
        if (v == fa[u] || v == hson[u]) continue;
        link(v, v);
```

```
    }
}

//segTree
int ma[M << 2], val[M << 2];

void pushup(int rt)
{
    if (ma[rt << 1] < ma[rt << 1 | 1])
    {
        ma[rt] = ma[rt << 1 | 1];
        val[rt] = val[rt << 1 | 1];
    }
    else
    {
        ma[rt] = ma[rt << 1];
        val[rt] = val[rt << 1];
    }
}
void build(int rt, int l, int r)
{
    if (l == r)
    {
        ma[rt] = 0; val[rt] = l;
        return;
    }
    int m = (l + r) >> 1;
    build(rt << 1, l, m);
    build(rt << 1 | 1, m + 1, r);
    pushup(rt);
}
void update(int rt, int l, int r, int k, int v)
{
    if (l == r)
    {
        ma[rt] += v;
        return;
    }
    int m = (l + r) >> 1;
    if (k <= m) update(rt << 1, l, m, k, v);
    else update(rt << 1 | 1, m + 1, r, k, v);
    pushup(rt);
}
vector<int>in[M], de[M];
void change(int u, int v, int w)
```

```
{
    int fu = top[u], fv = top[v];
    while (fu != fv)
    {
        if (dep[fu] < dep[fv])
        {
            swap(u, v);
            swap(fu, fv);
        }
        in[ seq[fu] ].push_back(w);
        de[ seq[u] + 1].push_back(w);
        u = fa[fu];
        fu = top[u];
    }
    if (dep[u] > dep[v])swap(u, v);
    in[ seq[u] ].push_back(w);
    de[ seq[v] + 1].push_back(w);
}

void init()
{
    memset(head, -1, sizeof(head));
    edgenum = num = 0;
    for (int i = 0; i < M; i++)
    {
        in[i].clear();
        de[i].clear();
    }
}
int ans[M];
int main(void)
{
    while (scanf("%d%d", &n, &m) != EOF)
    {
        if (n == 0 && m == 0)break;
        init();
        for (int i = 1; i < n; i++)
        {
            int u, v; scanf("%d%d", &u, &v);
            addedge(u, v);
            addedge(v, u);
        }
        int rt = (n + 1) / 2;
        dfs(rt, 0); //fa[rt] = 0;
        link(rt, rt);
```

```
        build(1, 1, M - 1);
        for (int i = 1; i <= m; i++)
        {
            int u, v, w;
            scanf("%d%d%d", &u, &v, &w);
            change(u, v, w);
        }
        for (int  i = 1; i <= n; i++)
        {
            for (int j = 0; j < in[i].size(); j++)
                update(1, 1, M - 1, in[i][j], 1);
            for (int j = 0; j < de[i].size(); j++)
                update(1, 1, M - 1, de[i][j], -1);
            int u = reseq[i];
            ans[u] = val[1];
            if (ma[1] == 0)ans[u] = 0;
        }
        for (int i = 1; i <= n; i++)
            printf("%d\n", ans[i]);
    }
    return 0;
}
```

SPOJ 375 (更改边值，询问极值)

```
/*
一棵树，每条边有个权值

两种操作

一个修改每条边权值

一个询问两点之间这一条链的最大边权

点数<=10000

多组测试数据，case<=20
*/
#include <cstdio>
#include <algorithm>
#include <iostream>
#include <string.h>
using namespace std;
const int maxn = 10010;
struct Tedge
```

```
{
    int b, next;
} e[maxn * 2];
int tree[maxn];
int zzz, n, z, edge, root, a, b, c;
int d[maxn][3];
int first[maxn], dep[maxn], w[maxn], fa[maxn];
int top[maxn], son[maxn], siz[maxn];
char ch[10];

void insert(int a, int b, int c)
{
    e[++edge].b = b;
    e[edge].next = first[a];
    first[a] = edge;
}

void dfs(int v)
{
    siz[v] = 1; son[v] = 0;
    for (int i = first[v]; i > 0; i = e[i].next)
        if (e[i].b != fa[v])
        {
            fa[e[i].b] = v;
            dep[e[i].b] = dep[v] + 1;
            dfs(e[i].b);
            if (siz[e[i].b] > siz[son[v]]) son[v] = e[i].b;
            siz[v] += siz[e[i].b];
        }
}

void build_tree(int v, int tp)
{
    w[v] = ++ z; top[v] = tp;
    if (son[v] != 0) build_tree(son[v], top[v]);
    for (int i = first[v]; i > 0; i = e[i].next)
        if (e[i].b != son[v] && e[i].b != fa[v])
            build_tree(e[i].b, e[i].b);
}

void update(int root, int lo, int hi, int loc, int x)
{
    if (loc > hi || lo > loc) return;
    if (lo == hi)
    {
```

```
        tree[root] = x;
        return;
    }
    int mid = (lo + hi) / 2, ls = root * 2, rs = ls + 1;
    update(ls, lo, mid, loc, x);
    update(rs, mid + 1, hi, loc, x);
    tree[root] = max(tree[ls], tree[rs]);
}

int maxi(int root, int lo, int hi, int l, int r)
{
    if (l > hi || r < lo) return 0;
    if (l <= lo && hi <= r) return tree[root];
    int mid = (lo + hi) / 2, ls = root * 2, rs = ls + 1;
    return max(maxi(ls, lo, mid, l, r), maxi(rs, mid + 1, hi, l, r));
}

inline int find(int va, int vb)
{
    int f1 = top[va], f2 = top[vb], tmp = 0;
    while (f1 != f2)
    {
        if (dep[f1] < dep[f2])
        {
            swap(f1, f2);
            swap(va, vb);
        }
        tmp = max(tmp, maxi(1, 1, z, w[f1], w[va]));
        va = fa[f1]; f1 = top[va];
    }
    if (va == vb) return tmp;//va此时为lca
    if (dep[va] > dep[vb]) swap(va, vb);//深度小的为lca
    return max(tmp, maxi(1, 1, z, w[son[va]], w[vb]));  //
}

void init()
{
    scanf("%d", &n);
    root = (n + 1) / 2;
    fa[root] = z = dep[root] = edge = 0;
    z = 1;
    memset(siz, 0, sizeof(siz));
    memset(first, 0, sizeof(first));
    memset(tree, 0, sizeof(tree));
    for (int i = 1; i < n; i++)
```

```
    {
        scanf("%d%d%d", &a, &b, &c);
        d[i][0] = a; d[i][1] = b; d[i][2] = c;
        insert(a, b, c);
        insert(b, a, c);
    }
    dfs(root);
    build_tree(root, root);
    //     for(int i = 1;i <= n;i++)   //
    //        printf("%d ",w[i]);
    //     printf("\n");
    for (int i = 1; i < n; i++)
    {
        if (dep[d[i][0]] > dep[d[i][1]]) swap(d[i][0], d[i][1]);//保证大小
        update(1, 1, z, w[d[i][1]], d[i][2]);//w是树上的顶点对应线段树中的标号
        //每一个顶点的全脂记录的是它和父亲节点连边的权值
    }
}

inline void read()
{
    ch[0] = ' ';
    while (ch[0] < 'C' || ch[0] > 'Q') scanf("%s", &ch);
}

void work()
{
    for (read(); ch[0] != 'D'; read())
    {
        scanf("%d%d", &a, &b);
        if (ch[0] == 'Q') printf("%d\n", find(a, b));
        else update(1, 1, z, w[d[a][1]], b);
    }
}

int main()
{
#ifdef xiaohai
    freopen("in.cpp", "r", stdin);
#endif
    for (scanf("%d", &zzz); zzz > 0; zzz--)
    {
        init();
        work();
    }
```

```
    return 0;
}
```

POJ 3237 Tree

```
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
const int maxn = 10010;
struct edge
{
    int v, next;
} e[maxn * 2];
int first[maxn], cnt;

int top[maxn], dep[maxn], sz[maxn], f[maxn];
int son[maxn], rank[maxn], tid[maxn];
int tp, tim;
int d[maxn][3];
int n;
void AddEdge(int u, int v)
{
    e[cnt].v = v;
    e[cnt].next = first[u];
    first[u] = cnt++;

    e[cnt].v = u;
    e[cnt].next = first[v];
    first[v] = cnt++;

}
void init()
{
    memset(first, -1, sizeof(first));
    cnt = 1;
    memset(son, -1, sizeof(son));
    tim = 0;
}

void dfs1(int u, int fa, int d)
{
    sz[u] = 1;
    dep[u] = d;
    f[u] = fa;
```

```
    for (int i = first[u]; i != -1; i = e[i].next)
    {
        int v = e[i].v;
        if (v == fa)
            continue;
        dfs1(v, u, d + 1);
        sz[u] += sz[v];
        if (son[u] == -1 || sz[son[u]] < sz[v])
            son[u] = v;
    }
}

void dfs2(int u, int tp)
{
    top[u] = tp;
    tid[u] = ++tim;
    rank[tid[u]] = u;
    if (son[u] == -1)
        return;
    dfs2(son[u], tp);
    for (int i = first[u]; i != -1; i = e[i].next)
    {
        int v = e[i].v;
        if (v != f[u] && son[u] != v)
            dfs2(v, v);
    }
}

int ma[maxn << 2];
int mi[maxn << 2];
int lz[maxn << 2];
void pushup(int l, int r, int rt)
{
    ma[rt] = max(ma[rt << 1], ma[rt << 1 | 1]);
    mi[rt] = min(mi[rt << 1], mi[rt << 1 | 1]);
}
void build(int l, int r, int rt)
{
    ma[rt] = 0;
    mi[rt] = 0;
    lz[rt] = 0;
    if (l == r)
    {
        return;
    }
```

```
    int m = (l + r) >> 1;
    build(l, m, rt << 1);
    build(m + 1, r, rt << 1 | 1);
}
void pushdown(int l, int r, int rt)
{
    if (l == r)
        return;
    if (lz[rt])
    {
        mi[rt << 1] = -mi[rt << 1];
        ma[rt << 1] = -ma[rt << 1];
        swap(mi[rt << 1], ma[rt << 1]);
        mi[rt << 1 | 1] = -mi[rt << 1 | 1];
        ma[rt << 1 | 1] = -ma[rt << 1 | 1];
        swap(mi[rt << 1 | 1], ma[rt << 1 | 1]);
        lz[rt << 1] ^= 1;
        lz[rt << 1 | 1] ^= 1;
        lz[rt] = 0;
    }
}
void update(int x, int y, int l, int r, int rt)
{
    if (x == l && y == r)
    {
        mi[rt] = -mi[rt];
        ma[rt] = -ma[rt];
        swap(mi[rt], ma[rt]);
        lz[rt] ^= 1;
        return;
    }
    pushdown(l, r, rt);
    int m = (l + r) >> 1;
    if (y <= m)
        update(x, y, l, m, rt << 1);
    else if (x > m)
        update(x, y, m + 1, r, rt << 1 | 1);
    else
    {
        update(x, m, l, m, rt << 1);
        update(m + 1, y, m + 1, r, rt << 1 | 1);
    }
    pushup(l, r, rt);
}
void update2(int x, int l, int r, int rt, int w)
```

```
{
    if (l == r)
    {
        ma[rt] = mi[rt] = w;
        lz[rt] = 0;
        return;
    }
    pushdown(l, r, rt);
    int m = (l + r) >> 1;
    if (x <= m)
        update2(x, l, m, rt << 1, w);
    else
        update2(x, m + 1, r, rt << 1 | 1, w);
    pushup(l, r, rt);
}
int query(int x, int y, int l, int r, int rt)
{
    if (l == x && r == y)
        return ma[rt];
    pushdown(l, r, rt);

    int m = (l + r) >> 1;
    if (y <= m)
        return query(x, y, l, m, rt << 1);
    else if (x > m)
        return query(x, y, m + 1, r, rt << 1 | 1);
    else
    {
        return max(query(x, m, l, m, rt << 1),
                   query(m + 1, y, m + 1, r, rt << 1 | 1));
    }
}

void change(int u, int v)
{
    while (top[u] != top[v])
    {
        if (dep[top[u]] < dep[top[v]])
            swap(u, v);
        update(tid[top[u]], tid[u], 1, n, 1);
        u = f[top[u]];
    }
    if (u == v)
        return;
    if (dep[u] > dep[v])
```

```
        swap(u, v);
    update(tid[son[u]], tid[v], 1, n, 1);
}
int find(int u, int v)
{
    int ans = -999999999;
    while (top[u] != top[v])
    {
        if (dep[top[u]] < dep[top[v]])
            swap(u, v);
        ans = max(ans, query(tid[top[u]], tid[u], 1, n, 1));
        u = f[top[u]];
    }
    if (u == v)
        return ans;
    if (dep[u] > dep[v])
        swap(u, v);
    ans = max(ans, query(tid[son[u]], tid[v], 1, n, 1));
    return ans;
}
int main()
{
    int T;
    scanf("%d", &T);
    while (T--)
    {
        init();

        scanf("%d", &n);
        for (int i = 1; i < n; i++)
        {
            int u, v, w;
            scanf("%d %d %d", &u, &v, &w);
            AddEdge(u, v);
            d[i][0] = u;
            d[i][1] = v;
            d[i][2] = w;
        }
        dfs1(1, 0, 0);
        dfs2(1, 1);
        build(1, n, 1);

        for (int i = 1; i < n; i++)
        {
            if (dep[d[i][0]] > dep[d[i][1]])
```

```
                swap(d[i][0], d[i][1]);
            update2(tid[d[i][1]], 1, n, 1, d[i][2]);
        }
        char s[10];
        while (scanf("%s", s) && strcmp(s, "DONE"))
        {
            if (s[0] == 'Q')
            {
                int x, y;
                scanf("%d %d", &x, &y);
                printf("%d\n", find(x, y));
            }
            else if (s[0] == 'C')
            {
                int x, y;
                scanf("%d %d", &x, &y);
                update2(tid[d[x][1]], 1, n, 1, y);
            }
            else
            {
                int x, y;
                scanf("%d %d", &x, &y);
                change(x, y);
            }
        }
    }
    return 0;
}
```

§ 14.2 动态树(LCT)

14.2.1 HDU 4010

```
#include <stdio.h>
#include <string.h>
#include <iostream>
#include <algorithm>
#include <vector>
#include <queue>
#include <set>
#include <map>
#include <string>
#include <math.h>
#include <stdlib.h>
#include <time.h>
```

```
using namespace std;
//动态维护一组森林，要求支持一下操作：
//link(a,b) : 如果a,b不在同一颗子树中，则通过在a,b之间连边的方式，连接这两颗子树
//cut(a,b)  : 如果a,b在同一颗子树中，且a!=b,则将a视为这颗子树的根以后，切断b与其父亲结点的连接
//ADD(a,b,w): 如果a,b在同一颗子树中，则将a,b之间路径上所有点的点权增加w
//query(a,b): 如果a,b在同一颗子树中，返回a,b之间路径上点权的最大值
const int MAXN = 300010;
int ch[MAXN][2], pre[MAXN], key[MAXN];
int add[MAXN], rev[MAXN], Max[MAXN];
bool rt[MAXN];

void Update_Add(int r, int d)
{
    if (!r)return;
    key[r] += d;
    add[r] += d;
    Max[r] += d;
}
void Update_Rev(int r)
{
    if (!r)return;
    swap(ch[r][0], ch[r][1]);
    rev[r] ^= 1;
}
void push_down(int r)
{
    if (add[r])
    {
        Update_Add(ch[r][0], add[r]);
        Update_Add(ch[r][1], add[r]);
        add[r] = 0;
    }
    if (rev[r])
    {
        Update_Rev(ch[r][0]);
        Update_Rev(ch[r][1]);
        rev[r] = 0;
    }
}
void push_up(int r)
{
    Max[r] = max(max(Max[ch[r][0]], Max[ch[r][1]]), key[r]);
}
void Rotate(int x)
```

```
{
    int y = pre[x], kind = ch[y][1] == x;
    ch[y][kind] = ch[x][!kind];
    pre[ch[y][kind]] = y;
    pre[x] = pre[y];
    pre[y] = x;
    ch[x][!kind] = y;
    if (rt[y]) //if y isn't his father's preferred child
        rt[y] = false, rt[x] = true;//don't let x be y's orgin father's child
    else
        ch[pre[x]][ch[pre[x]][1] == y] = x; //in splay , rt[x] must be false
    push_up(y);
}
//P函数先将根结点到r的路径上所有的结点的标记逐级下放
void P(int r)
{
    if (!rt[r])P(pre[r]);
    push_down(r);
}
void Splay(int r)
{
    P(r);
    while ( !rt[r] )
    {
        int f = pre[r], ff = pre[f];
        if (rt[f])
            Rotate(r);
        else if ( (ch[ff][1] == f) == (ch[f][1] == r) )
            Rotate(f), Rotate(r);
        else
            Rotate(r), Rotate(r);
    }
    push_up(r);
}
int Access(int x)
{
    int y = 0;
    //go on splaying x to its root of splay tree until x be root of whole forest
    for ( ; x ; x = pre[y = x])
    {
        Splay(x);//the right subtree of x is deeper than x
        rt[ch[x][1]] = true, rt[ch[x][1] = y] = false;
        push_up(x);
    }
    return y;
```

```
}
//判断是否是同根(真实的树，非splay)
bool judge(int u, int v)
{
    while (pre[u]) u = pre[u];
    while (pre[v]) v = pre[v];
    return u == v;
}
//使r成为它所在的树的根
void mroot(int r)
{
    Access(r);
    Splay(r);
    Update_Rev(r);
}
//调用后u是原来u和v的lca,v和ch[u][1]分别存着lca的2个儿子
//(原来u和v所在的2颗子树)
void lca(int &u, int &v)
{
    Access(v), v = 0;
    while (u)
    {
        Splay(u);
        if (!pre[u])return;
        rt[ ch[u][1] ] = true;
        rt[ ch[u][1] = v ] = false;
        push_up(u);
        u = pre[v = u];
    }
}
void link(int u, int v)
{
    if (judge(u, v))
    {
        puts("-1");
        return;
    }
    mroot(u);
    pre[u] = v;
}
//使u成为u所在树的根，并且v和它父亲的边断开
void cut(int u, int v)
{
    if (u == v || !judge(u, v))
    {
```

```
        puts("-1");
        return;
    }
    mroot(u);
    Splay(v);
    pre[ch[v][0]] = pre[v];
    pre[v] = 0;
    rt[ch[v][0]] = true;
    ch[v][0] = 0;
    push_up(v);
}
void ADD(int u, int v, int w)
{
    if (!judge(u, v))
    {
        puts("-1");
        return;
    }
    lca(u, v);
    Update_Add(ch[u][1], w);
    Update_Add(v, w);
    key[u] += w;
    push_up(u);
}
void query(int u, int v)
{
    if (!judge(u, v))
    {
        puts("-1");
        return;
    }
    lca(u, v);
    printf("%d\n", max(max(Max[v], Max[ch[u][1]]), key[u]));
}

struct Edge
{
    int to, next;
} edge[MAXN * 2];
int head[MAXN], tot;
void addedge(int u, int v)
{
    edge[tot].to = v;
    edge[tot].next = head[u];
    head[u] = tot++;
```

```
}
void dfs(int u)
{
    for (int i = head[u]; i != -1; i = edge[i].next)
    {
        int v = edge[i].to;
        if (pre[v] != 0)continue;
        pre[v] = u;
        dfs(v);
    }
}

int main()
{
    //freopen("in.txt","r",stdin);
    //freopen("out.txt","w",stdout);
    int n, q, u, v;
    while (scanf("%d", &n) == 1)
    {
        tot = 0;
        for (int i = 0; i <= n; i++)
        {
            head[i] = -1;
            pre[i] = 0;
            ch[i][0] = ch[i][1] = 0;
            rev[i] = 0;
            add[i] = 0;
            rt[i] = true;
        }
        Max[0] = -2000000000;
        for (int i = 1; i < n; i++)
        {
            scanf("%d%d", &u, &v);
            addedge(u, v);
            addedge(v, u);
        }
        for (int i = 1; i <= n; i++)
        {
            scanf("%d", &key[i]);
            Max[i] = key[i];
        }
        scanf("%d", &q);
        pre[1] = -1;
        dfs(1);
        pre[1] = 0;//must make sure the root of forest's father is 0
```

```
        int op;
        while (q--)
        {
            scanf("%d", &op);
            if (op == 1)
            {
                int x, y;
                scanf("%d%d", &x, &y);
                link(x, y);
            }
            else if (op == 2)
            {
                int x, y;
                scanf("%d%d", &x, &y);
                cut(x, y);
            }
            else if (op == 3)
            {
                int w, x, y;
                scanf("%d%d%d", &w, &x, &y);
                ADD(x, y, w);
            }
            else
            {
                int x, y;
                scanf("%d%d", &x, &y);
                query(x, y);
            }
        }
        printf("\n");
    }
    return 0;
}
```

14.2.2 HDU 5002

```
/*
Problem Description
You are given a tree with N nodes which are numbered by integers 1..N. Each node is associated wi

Your task is to deal with M operations of 4 types:

1.Delete an edge (x, y) from the tree, and then add a new edge (a, b). We ensure that it still co

2.Given two nodes a and b in the tree, change the weights of all the nodes on the path connecting
```

```
3.Given two nodes a and b in the tree, increase the weights of all the nodes on the path connecti

4.Given two nodes a and b in the tree, compute the second largest weight on the path connecting r


Input
The first line contains an integer T (T<=3), which means there are T test cases in the input.

For each test case, the first line contains two integers N and M (N, M<=10^5). The second line co

In next N-1 lines, there are two integers a and b (1<=a, b<=N), which means there exists an edge

The next M lines describe the operations you have to deal with. In each line the first integer is

If c = 1, there are four integers x, y, a, b (1<= x, y, a, b <=N) after c.
If c = 2, there are three integers a, b, x (1<= a, b<=N, |x|<=10^4) after c.
If c = 3, there are three integers a, b, d (1<= a, b<=N, |d|<=10^4) after c.
If c = 4 (it is a query operation), there are two integers a, b (1<= a, b<=N) after c.

All these parameters have the same meaning as described in problem description.


Output
For each test case, first output "Case #x:"" (x means case ID) in a separate line.

For each query operation, output two values: the second largest weight and the number of times it
*/

#pragma comment(linker, "/STACK:1024000000,1024000000")
#include<stdio.h>
#include<algorithm>
#include<string.h>
#define ls son[0][rt]
#define rs son[1][rt]
using namespace std ;

const int maxn = 100001 ;
const int INF = -1110011111 ;
int son[2][maxn] , mx[2][maxn] , cnt[2][maxn] , size[maxn] ;
int  fa[maxn] , is[maxn] , val[maxn] ;
int col[maxn] , add[maxn] , rev[maxn] ;
int st[55] ;
void push_up ( int rt )
{
```

```
    size[rt] = size[ls] + size[rs] + 1 ;
    st[1] = mx[0][ls] ;
    st[2] = mx[1][ls] ;
    st[3] = mx[0][rs] ;
    st[4] = mx[1][rs] ;
    st[5] = val[rt] ;
    sort ( st + 1 , st + 6 ) ;
    int T = unique ( st + 1 , st + 6 ) - st - 1 ;
    mx[0][rt] = st[T] , mx[1][rt] = st[T - 1] ;
    cnt[0][rt] = cnt[1][rt] = 0 ;
    for ( int i = 0 ; i < 2 ; i ++ )
        for ( int j = 0 ; j < 2 ; j ++ )
        {
            int v = mx[i][son[j][rt]] ;
            if ( v == mx[0][rt] ) cnt[0][rt] += cnt[i][son[j][rt]] ;
            if ( v == mx[1][rt] ) cnt[1][rt] += cnt[i][son[j][rt]] ;
        }
    if ( val[rt] == mx[0][rt] ) cnt[0][rt] ++ ;
    if ( val[rt] == mx[1][rt] ) cnt[1][rt] ++ ;
}

void reverse ( int rt )
{
    if ( !rt ) return ;
    swap ( son[0][rt] , son[1][rt] ) ;
    rev[rt] ^= 1 ;
}

void color ( int rt , int c )
{
    if ( !rt ) return ;
    val[rt] = c ;
    mx[0][rt] = col[rt] = c ;
    mx[1][rt] = add[rt] = INF ;
    cnt[0][rt] = size[rt] ;
    cnt[1][rt] = 0 ;
}

void ADD ( int rt , int c )
{
    if ( !rt ) return ;
    if ( add[rt] == INF ) add[rt] = c ;
    else add[rt] += c ;
    mx[0][rt] += c ;
    val[rt] += c ;
```

```
    if ( mx[1][rt] != INF ) mx[1][rt] += c ;
}

void push_down ( int rt )
{
    if ( rev[rt] )
    {
        reverse ( ls ) ;
        reverse ( rs ) ;
        rev[rt] = 0 ;
    }
    if ( col[rt] != INF )
    {
        color ( ls , col[rt] ) ;
        color ( rs , col[rt] ) ;
        col[rt] = INF ;
    }
    if ( add[rt] != INF )
    {
        ADD ( ls , add[rt] ) ;
        ADD ( rs , add[rt] ) ;
        add[rt] = INF ;
    }
}

void down ( int rt )
{
    if ( !is[rt] ) down ( fa[rt] ) ;
    push_down ( rt ) ;
}

void rot ( int rt )
{
    int y = fa[rt] , z = fa[y] , c = rt == son[0][y] ;
    son[!c][y] = son[c][rt] ; fa[son[c][rt]] = y ;
    son[c][rt] = y ; fa[y] = rt ;
    fa[rt] = z ;
    if ( is[y] ) is[y] = 0 , is[rt] = 1 ;
    else son[y == son[1][z]][z] = rt ;
    push_up ( y ) ;
}

void splay ( int rt )
{
    down ( rt ) ;
```

```
    while ( !is[rt] )
    {
        int y = fa[rt] , z = fa[y] ;
        if ( !is[y] )
            rot ( (rt == son[0][y]) == (y == son[0][z]) ? y : rt ) ;
        rot ( rt ) ;
    }
    push_up ( rt ) ;
}

void access ( int rt )
{
    for ( int v = 0 ; rt ; rt = fa[rt] )
    {
        splay ( rt ) ;
        is[rs] = 1 ; is[v] = 0 ;
        rs = v ;
        v = rt ;
        push_up ( rt ) ;
    }
}

void change_root ( int rt )
{
    access ( rt ) ;
    splay ( rt ) ;
    reverse ( rt ) ;
}

void cut ( int a , int b )
{
    change_root ( a ) ;
    //    printf("fa[a]: %d\n",fa[a]);
    access ( a ) ;//clear the flag about a and his son
    //    printf("fa[a]: %d\n",fa[a]);
    splay ( b ) ;
    //    printf("fa[b]: %d\n",fa[b]);
    fa[b] = 0 ;
}

void link ( int a , int b )
{
    change_root ( b ) ;
    fa[b] = a ;
}
```

```
void gao1 ( int a , int b , int c , int d )
{
    cut ( a , b ) ;
    link ( c , d ) ;
}

void gao2 ( int a , int b , int c )
{
    change_root ( a ) ;
    access ( b ) ;
    splay ( b ) ;
    color ( b , c ) ;
}

void gao3 ( int a , int b , int c )
{
    change_root ( a ) ;
    access ( b ) ;
    splay ( b ) ;
    ADD ( b , c ) ;
}

void print ( int rt )
{
    if ( !rt ) return ;
    printf ( "rt = %d , mx0 = %d , mx1 = %d\n" ,
             rt , mx[0][rt] , mx[1][rt] ) ;
    print ( ls ) ;
    print ( rs ) ;
}
void gao4 ( int a , int b )
{
    change_root ( a ) ;
    access ( b ) ;
    splay ( b ) ;
    //   print ( b ) ;
    if ( mx[1][b] == INF ) puts ( "ALL SAME" ) ;
    else printf ( "%d %d\n" , mx[1][b] , cnt[1][b] ) ;
}

void init ( int rt )
{
    int a ;
    scanf ( "%d" , &a ) ;
```

```
    val[rt] = mx[0][rt] = a ;
    mx[1][rt] = INF ;
    cnt[0][rt] = 1 ;
    cnt[1][rt] = 0 ;
    size[rt] = 1 ;
    add[rt] = col[rt] = INF ;
    son[0][rt] = son[1][rt] = fa[rt] = rev[rt] = 0 ;
    is[rt] = 1 ;
}

int main ()
{
    int T , ca = 0 ;
    int n , m ;
    //    freopen ( "main.in" , "r" , stdin ) ;
    mx[0][0] = mx[1][0] = INF ;
    is[0] = 1 , fa[0] = 0 ;
    scanf ( "%d" , &T ) ;
    while ( T -- )
    {
        scanf ( "%d%d" , &n , &m ) ;
        for ( int i = 1 ; i <= n ; i ++ )
            init ( i ) ;
        for ( int i = 1 ; i < n ; i ++ )
        {
            int a , b ;
            scanf ( "%d%d" , &a , &b ) ;
            link ( a , b ) ;
        }
        printf ( "Case #%d:\n" , ++ ca ) ;
        for ( int i = 1 ; i <= m ; i ++ )
        {
            int a , b , c , d ;
            scanf ( "%d" , &c ) ;
            if ( c == 1 )
            {
                int f ;
                scanf ( "%d%d%d%d" , &a , &b , &d , &f ) ;
                if ( a == b || d == f ) continue ;
                gao1 ( a , b , d , f ) ;
            }
            else if ( c == 2 )
            {
                scanf ( "%d%d%d" , &a , &b , &d ) ;
                gao2 ( a , b , d ) ;
```

```
            }
            else if ( c == 3 )
            {
                scanf ( "%d%d%d" , &a , &b , &d ) ;
                gao3 ( a , b , d ) ;
            }
            else
            {
                scanf ( "%d%d" , &a , &b ) ;
                gao4 ( a , b ) ;
            }
        }
    }
}
/*
2
3 2
1 1 2
1 2
1 3
4 2 3
*/
```