福州大学 ACM/ICPC 集训队资料

Implemented in C/C++

2010年11月27日

目 录

1 语言基础 **1**
- 1.1 **C**语言 . . . 1
- 1.2 **C++** . . . 5
- 1.3 注意事项 . . . 8
- 1.4 **Java** . . . 9

2 数据结构 **14**
- 2.1 Hash . . . 14
- 2.2 堆 . . . 16
- 2.3 常用数据结构 . . . 20
- 2.4 后缀数组 . . . 32
- 2.5 数据结构的典型应用 . . . 39

3 基础算法 **41**
- 3.1 搜索算法 . . . 41
- 3.2 动态规划 . . . 41
- 3.3 排列与阶乘 . . . 46
- 3.4 格雷码 . . . 47
- 3.5 归并排序求逆序数 . . . 48
- 3.6 日期计算 . . . 48
- 3.7 KMP算法 . . . 49
- 3.8 字符串最小表示法 . . . 50
- 3.9 求第k小元素的O(n)算法 . . . 51
- 3.10 N皇后的一组构造解 . . . 52

4 计算几何 **53**
- 4.1 基础知识、点，向量 . . . 53
- 4.2 线段、直线、曲线 . . . 55
- 4.3 凸包与顶点对 . . . 61
- 4.4 三角形 . . . 65
- 4.5 多边形 . . . 68
- 4.6 圆、椭圆 . . . 77
- 4.7 三维几何 . . . 84
- 4.8 四面体 . . . 93
- 4.9 注意事项 . . . 93

5 图论 **95**
5.1 图的表示 . . . 95
5.2 图的基本算法 . . . 96
5.3 树的优化算法 . . . 97
5.4 生成树 . . . 98
5.5 最短路 . . . 100
5.6 连通性 . . . 104
5.7 网络流 . . . 108
5.8 二分图匹配 . . . 118
5.9 独立集与支配集 . . . 125
5.10 注意事项 . . . 126

6 数论 **127**
6.1 欧几里德算法 . . . 127
6.2 模线性方程 . . . 127
6.3 中国剩余定理 . . . 128
6.4 模方程合并 . . . 128
6.5 $a^b \bmod c$ . . . 129
6.6 Baby-step giant-step . . . 130
6.7 素数判定随机算法 . . . 131
6.8 分解质因数随机算法Pollard . . . 132
6.9 欧拉函数 . . . 133
6.10 欧拉定理解方程$A^x modB = 1$ 中的x . . . 134
6.11 筛法生成质数表 . . . 135
6.12 Pell方程 . . . 137
6.13 分数逼近 . . . 138
6.14 初等数论 . . . 139

7 组合数学 **142**
7.1 常用公式 . . . 142
7.2 Fibonacci数 . . . 145
7.3 母函数 . . . 149

8 数值计算 **153**
8.1 解线性方程组 . . . 153
8.2 追赶法解周期性方程 . . . 154
8.3 矩阵运算 . . . 155
8.4 Romberg积分 . . . 157
8.5 线性规划单纯型算法linear programming simplex algorithm . . . 157
8.6 判线性相关(正交化) . . . 159
8.7 牛顿法解多项式的根 . . . 159
8.8 高精度计算 . . . 160
8.9 分数运算 . . . 164
8.10 行列式$mod\ m$的值 . . . 165
8.11 FFT a*b . . . 166

9 博弈论 **170**
9.1 经典博弈 170
9.2 Sprague – Grundy 函数 170
9.3 Nim-Multiplication 171
9.4 常见限制条件下的SG值 172

10 经典问题 **173**

第一章 语言基础

1.1 C语言

1 输入

由于竞赛试题均采用测试数据组的形式输入测试数据，我们总结出一个一般的框架：

```
while (scanf("%d",&n),结束条件) {
  scanf("%d%d",&a,&b);
  if (满足特殊条件)
    {输出结果;continue;}
  初始化;
  一般情况的处理;
}
```

```
int in(void)
{
  读入数据
  if (满足结束条件) return 0;
  继续读数据
  初始化
  return 1;
}
int main() {while(in()) {处理} }
```

【结束条件的判断】

- n=0 indicates the termination of the input data set.
  `while(scanf("%d",&n),n) {`
- The first line of the input contains a single integer t, the number of test cases, followed by the input data for each test case.
  `scanf("%d",&t); while (t-->0) {`
- Input is terminated by end-of-file.
  `while(scanf("%d",&n)!=EOF) {`

【输入格式】

- scanf中加空格可以过滤头尾的空格如`scanf("␣%c␣",&c)` 不会把空格读入c
- 通过前导空格`for(;*str=='␣';str++);`
- 读入一个字符

`getch();`	无缓冲，不回显	`getche()`	无缓冲，回显
`getchar()`	行缓冲	`scanf("%c",&c)`	无缓冲

- 字符串转换(字符串转换为非标准函数，比赛中会ce,请勿用)
  - `atof(char *str)` 字符串→**double**，可含有非数字，如 $1.2e5BC$ ，转换后得 $1.2 * 10^5$
  - `atoi(char *str)` 字符串→**int**
  - `atol(char *str)` 字符串→**long**
  - `strtol(char *str,char **p,base)`字符串→**long**可指定进制base,`**p`为'0'表示成功转化,否则**p非法字符前的数字将被返回, **p使用前要 `p=(char **)malloc(sizeof(char *))`
  - `strtoll(char *str,char **p,base)`和`strtoq(char *str,char **p,base)`用法与strtol相同，返回为64位整型（VC6.0不支持）
  - `strtoul(char *str,char **p,base),strtoull(char *str,char **p,base)`与上面的类似
- 读入一行字符串（包括空格）`gets(str)`
- `%[a-z]` 只接受a-z
- `%[^:]` 字符串以:作为分割,:不读入并且停止读入
- `%*c` *表示忽略这一项

2 输出

输出数据时要注意回车的处理：

- Print a blank line **between** each game case.
- Output a blank line **after** each paragraph.

【输出格式】

%5d	最小宽度5
%05d或%.5d	不足宽度5时补0
%.4lf	小数点后4位（四舍五入）
%.4g	4位有效数字
%5.7s	字符串最少5个字符，最多7个字符
%.4d	必须显示的最小位数
%-5d	左对齐
%#lf	确保出现小数点

3 快速排序

C标准库stdlib.h中提供了快速排序函数

qsort(数组名,数组长度,每个元素长度,比较函数名);

其中比较函数决定了排序的方式和关键字，函数返回类型为**int**，a,b相等返回0，a排在b前面返回-1，a排在b后面返回1。

```
CSeg seg[n];
qsort(seg,n,sizeof(CSeg),cmp);
int cmp(const void *e1,const void *e2)
{
  CSeg *a,*b;
  a=(CSeg *)e1; b=(CSeg *)e2;
  if (a-b<epsilon&&b-a<epsilon) return 0;
  if (a<b+epsilon) return -1;
  return 1;
}
```

4 基本操作

【类型定义】

- 64**位整数**定义：**long long**(GCC)或__int64 (VC) **格式符**：%lld(GCC)或%I64d(VC)。定义64位整型常数时后面加上LL如100000000LL。
- **long double** 输入输出要用%Lf, L要大写, 数学函数用到**long double** 的时候要在原有的函数后加l,例如sqrtl。
- const确保变量在定义域内部不被修改，但可以被外部修改。但可以被外部修改。如**void** f(**const char** *s)不能改变s数组的值。
- volatile char *port编译器不会对其优化。此修饰符可去除一些C的优化导致的错误。
- extern声明在外部或后面定义的变量。
- static对于局部变量始终保持局部变量空间，全局变量则只对定义它的文件域有效。例：**int** f(**void**) {**static int** a=100; a=a+23; **return** f;} 每次调用f()返回$123, 146, \cdots$.
- register将变量置于尽可能快的存储空间，可用于定义频繁使用的变量。
- C中不能传递数组，可通过指针实现或把数组放入结构体用结构体赋值。例

  f(**int** *s)(x[10])(x[])(x[30])这些都是等价的。接收二维数组要声明第二维，例$10*10$数组，**void** f(**int** x[][10])

- 函数指针，可用函数指针的数组代替大型的switch

```
int (*p)(const char*,const char*);
p=strcmp;  (*p)(s1,s2);
```

【运算符】

- /作用于**int,char**时仅返回整数部分。要得到精确结果需要强制类型转换(**double**)i/j;
- true 非0值；false 0
- &按位与用于清除某位，例ch&127表示去除第8位。$127 = (01111111)_2$
- |按位或用于将某位置1，例ch|3表示末尾2位置1。$3 = (00000011)_2$
- ^按位异或用于标出两个操作数的不一致部分，例(0110)^(1111)=(1001)
- <<n相当于乘以2^n，>>n相当于除以2^n
- x的二进制表示中从低到高的第i位:x&(1<<(i-1))
- x末尾0的个数为k: 2^k =x&(x^(x-1))
- &m表示变量m的地址；*p表示p所指向的地址的内容；

5 字符串操作

- strcpy(s1,s2) s2复制到s1
- strcmp(s1,s2) s1=s2返回0，s1>s2返回> 0，s1<s2返回< 0
- strncmp(s1,s2,n) 对前n个字符操作
- strcat(s1,s2) s1=s1+s2
- strchr(s1,chr) 返回chr在s1第一次出现的指针
- strrchr(s1,chr) 返回chr在s1最后一次出现的指针
- strstr(s1,s2) 返回s2在s1第一次出现的指针
- strspn(s1,s2) 返回s1中不匹配s2中任何字符的第一个字符下标
- strcspn(s1,s2) 返回s1中全由非s2字符组成的第一个子串的长度
- strpbrk(s1,s2) 返回s1中匹配s2中任意字符的第一个字符的指针
- strlen(s1) s1的长度
- memchr(s1,int chr,n) 类似strchr，注意该函数以字节为单位。
- memcpy(Dest,Src,count) 类似strcpy
- memcmp(s1,s2) 类似strcmp
- memset(s,char chr,sizeof(s)) 每个字节置chr
- isalnum(c) 字母数字
- isalpha(c) 字母
- iscntrl(c) 控制字符
- isupper(c) 大写字母
- isdigit(c) 数字
- isgraph(c) 除空格外的所有可打印字符
- isprint(c) 含空格可打印
- islower(c) 小写字母
- ispunct(c) 非字母数字空格的可打印字符
- isspace(c) 空格,Tab,CR
- isxdigit(c) 16进制数

- 显示文本内容`while((ch=fgetc(fp))!=EOF) printf("%c",ch);`

- `char buffer[128]; setvbuf(fp,buffer,mode,128);` 将流fp的缓冲区置为mode状态，缓冲区大小128字节。其中mode包括：_IOLBF行缓冲 _IOFBF 缓冲满 _IONBF不缓冲

- `sprintf(str,"%d%c",2,'3');` 向字符串str写入23，不进行边界检查；类似的函数有sscanf。与C++的字符串流不同sscanf读入后字符串指针不变，不能用于循环读入中用空格分隔的连续的字符串

1.2 C++

1 输入输出

- 标准输入输出流的头文件是<iostream>，文件输入输出流的头文件是<fstream>，控制IO输出格式的头文件是<iomanip>；

- endl表示回车换行符'\n'，并flush输出缓冲区；ends相当于C语言中字符串结束符'\0'。

- 字符串输入输出流(<sstream>中的字符串流是用来支持STL的string类的，而<strstream>中的可以支持为**char**数组)，在使用上和cin, cout一样方便：

```
#include <strstream>
char buf[255];
istrstream strin(buf);
strin >> oct >> x;
ostrstream strout(buf, sizeof(buf));
strout << "Result␣is␣:␣" << x << endl;

#include <sstream>
string str;
istringstream strin(str);
ostringstream strout;
strin >> hex >> x;

//用于处理输入一行中字符输入个数未知的输入方式
#include <sstream>
while (gets(buf)) {
    stringstream ssin((string) buf);
    while (ssin >> num) {
        // 你要的操作
    }
}
```

- 输入输出格式控制

`cin >> noskipws;`	输入不跳过空字符(包括回车和TAB)
`boolalpha(cout)/noboolalpha(cout)`	bool以true/1输出
`cout<<dec<<oct<<hex`	十、八、十六进制
`cout<<scientific`	科学计数法
`cout.precision(2)`	两位有效位数字
`cout.precision(2);cout<<fixed<<i`	保留两位小数
`cout<<setw(10)<<i`	右对齐"␣␣␣␣-123"setw 设置场宽
`cout<<setw(10)<<internal<<i`	符号左对齐"-␣␣␣123"
`cout.width(20)`	设置宽度
`cout.fill('.')`	空格用“.”填充
`cout<<showpoint/noshowpoint`	小数位为0的浮点数输出/不输出其小数位
`cout<<showpos`	输出非负数前输出一个＋号

2 STL

<algorithm>

这里用数组int a[n]代替vector容器。

- `min(a,b)` 返回a,b中的最小值
- `max(a,b)` 返回a,b中的最大值
- `fill(a,a+n,val)` 用val填充a数组
- `sort(a,a+n,cmp)` 若`bool cmp(const int a,const int b);`省略使用小于号;cmp 为真时a在前
- `stable_sort(a,a+n,cmp)` 稳定排序
- `list.sort(cmp)` 基于链表的归并排序
- `make_heap(a,a+n,cmp)` 默认是最大堆化，即cmp为真时a做叶子
- `pop_heap(a,a+n,cmp)` 将堆顶元素移至`a[n-1]`且`a[0:n-2]`仍为堆
- `push_heap(a,a+n,cmp)` 将`a[n-1]`加入堆`a[0:n-2]`
- `sort_heap(a,a+n,cmp)` 将堆`a[n-1]`化为排序好的数组`a[n-1]`
- `lower_bound(a,a+n,val)` 二分返回第一次出现val的位置(迭代器)
- `upper_bound(a,a+n,val)` 返回一次出现位置的后一个位置(迭代器)
- `max_element(a,a+n,cmp)` 返回数组最大值第一次出现位置(迭代器)
- `min_element(a,a+n,cmp)` 返回数组最小值第一次出现位置(迭代器)
- `lexicographical_compare (a,a+n,b,b+m,cmp)` 按字典顺序比较大小

- `swap(a,b)` 交换a,b
- `copy(a,a+n,b)` 将a[0..n-1]拷贝到b[0..n-1]
- `swap_ranges(a,a+n,b)` 交换a[0..n-1]和b[0..n-1]
- `prev_permutation(a,a+n,cmp)` 产生a[0..n-1]的前一个排列，返回false表示a[0..n-1]已经是第一个排列
- `next_permutation(a,a+n,cmp)` 产生a[0..n-1]的下一个排列，返回false 表示a[0..n-1]已经是最后一个排列
- `unique_copy(a,a+n,b)` 去除a[0..n-1]中重复的多余元素，拷贝到b中
- `nth_element(a,a+k,a+n,cmp)` 根据第k个元素排序（找第k小元素）

平衡二叉树

- `<set>`
  - 定义：`set<double,greater<int>> t;multiset<int> t(a.begin(),a.end(),cmp);`
  - 插入：`tree.insert(val);` multiset返回bool; set返回pair其中.second表示是否插入成功, .first表示新元素或现存同值元素的位置。
  - 改变：该类型内容是只读的，不能改变
  - 查找：`tree.find(val);`返回值为val的第一个元素的迭代器;
    `tree.lower_bound(val);` 返回第一个大于等于val的元素位置
  - 删除：`tree.erase(tree.begin());`
- `<map>`
  - 定义：`map<key_type,val_type> tree;`
  - 查找：`tree.find(key);`
  - 访问：`tree.begin().first`表示keytype的值
  - 插入、删除：`tree.insert(make_pair(key,val));tree.erase(tree.begin());`失败返回tree.end();
  - 插入或更改：`tree[key]=val;`

线性数据结构

vector(数组) ,deque(双向数组) stack(栈) ,queue(队列)，priority_queue(优先队列)
vector可以动态增长在一些场合可以节省空间，经常用于图论邻接表的构建。

```
vector <int> graph[N]
...
cin>>x>>y;
graph[x].push_back(y);
...
```

graph[i].size()表示于i 相连的点的个数

<bitset> 位类型

- `string bitval("1010");bitset<32> b(bitval);`定义由‘0’,‘1’组成的字符串为bitset
- `bitset<100> a`定义长度100的bitset
- `a.test(pos);`pos位是否为1
- `a.any();`任意位是否为1
- `a.none();` 是否没有位为1
- `a.count();` 值是1的位的个数
- `a.size();` 位元素的个数
- `a[pos];` 访问pos位
- `a.flip();` 翻转所有位
- `a.flip(pos);` 翻转pos位
- `a.set();` 所有位置1
- `a.set(pos);` pos位置1
- `a.reset();` 所有位置0
- `a.reset(pos);` pos位置0
- `s=a.to_string();` 将类转成string
- bitset 类支持位操作符`&|^`等。

迭代器的使用

运用迭代器可以把STL中数据结构的元素取出，如下顺序取出map 中的元素

```
map<string,int>::iterator iter = text_map->begin();
while (iter != text_map->end() )
{
     string s=iter->first;
     int t=iter->second;
     ...
     iter++;
}
```

元素取出的顺序是按string从小到大的。其他类型也是类似的。

1.3 注意事项

1. **Windows下的有些GCC编译器用scanf读** 64 **位整数要用**$\%I64d$**, 尽量用cin，Linux下没问题**
2. 读单个字符时要特别注意格式，很容易把回车或空格误读进来。**读的时候在前面或后面加一个空格**，滤掉回车，如`"␣%s"`
3. 对于有字符串的题目要小心出现空串,特别是有些题目中在读入整数后在读入空串，必要时用gets过虑,有的题目结束条件是负数，不要给例子里的-1给欺骗了
4. 浮点格式的数据一律用`double`类型，必要的时候用long double，不要用`float`
5. 对于复杂的格式，可以先用`gets(str)`把数据读进来，再用`sscanf`函数处理
6. 宏定义 min(x,y),max(x,y) 时为避免优先级混乱 x,y 要加括号 (x),(y),整个表达式也要括号

7. 在做四舍五入的时候，**正数应该加**0.5，**负数应该减**0.5，**printf的四舍五入对于**0.5**的情况是随机处理的要小心。**

8. 浮点数输出不能遗漏小数点:`printf("%.0lf\n",0.);`

9. 定义π: `#define pi acos(-1.)`

10. 如果程序中用到数学函数，一定要`#include<math.h>`，否则虽然编译能通过，但无法得到正确结果

11. 时间复杂度与问题规模的关系

    (a) $n \leq 50$, 一般要用搜索。特别地，如果$n \leq 15$，一定要用搜索了,特别很多时候可以使用状态DP。

    (b) $n \leq 500$, 各种多项式算法均可，复杂度最大可至$O(n^4)$，再大就要优化了。

    (c) $n = 10000$, 可能是$O(n^2)$算法的时间极限，$O(n^2)$以上的算法肯定不适用。

    (d) $n \leq 50000$, 多为$O(n), O(n \log n)$算法，可能用到分治。

    (e) $n \geq 50000$，只能用$O(n)$算法了，可能用贪心或者构造

    (f) 如果n再大，不但算法不能超过$O(n)$，并且每个对象只能处理一次。

1.4 Java

1 基本框架

```
import java.io.*;    //io
import java.math.*;  //BigInteger,BigDecimal
import java.util.*;  //Date

public class Main {  //filename must be Main.java
 public static void main(String args[]) throws Exception
 {
   //输入输出重定向
   BufferedInputStream in=new BufferedInputStream(
                  new FileInputStream("a.in"));
   System.setIn(in);
   PrintStream out=new PrintStream(new FileOutputStream("a.out"));
   System.setOut(out);

   BufferedReader stdin=new BufferedReader(
                       new InputStreamReader(System.in));
   String line;
   while ((line=stdin.readLine())!=null) {
     StringTokenizer st=new StringTokenizer(line);    //通过st解析输入流
     BigDecimal R=new BigDecimal(st.nextToken().trim());
     int n=Integer.parseInt(st.nextToken().trim());
     System.out.println("Result:"+R+"^"+n);   //输出
   }
   //out.close();  如果有重定向需要加上这句
```

```
  }
}
```

2 字符流处理

- **while** ((s=br.readLine())!=**null**) 按行读
- **while** ((c=br.read())!=-1) 按字符读
- 系统自带的解析器：StringTokenizer st=**new** StringTokenizer(line);
- 通过st.nextToken();返回当前token，并进入下一个token
- 手工解析String[] tokens=line.trim().split("[␣\t\n\r]+"); 得到tokens[]数组。
- 字符串替换s=s.replaceAll("[␣\t\n\r]+","^_^");
- 字符串的正则表达式匹配str.matches("a*b") 返回**true**,**false**
- 异常处理

```
try {
  System.out.println("aaaaaaaab".matches("?"));
}catch(Exception e) {
  System.out.println("Not␣valid␣regular␣expression");
}
```

- 如果输出不好处理，可以将String转换为**char**[] res=res2.toCharArray();

3 数据类型与数学方法

【Java基本数据类型】

类型	分配空间(位)	取值范围
boolean	1	true,false
byte	8	$-128 \sim +127$
char	16	$\backslash u0000 \sim \backslash uFFFF$
double	64	$\pm 4.9 \times 10^{-324} \sim \pm 1.8 \times 10^{308}$
float	32	$\pm 1.4 \times 10^{-45} \sim \pm 3.4 \times 10^{38}$
int	32	$-2147283648 \sim 2147283647$
long	64	$-9.2 \times 10^{17} \sim +9.2 \times 10^{17}$
short	16	$-32768 \sim 32767$

【常见数学方法】

Math.abs(x)	x绝对值	Math.max(x,y)	x,y中较大者
Math.ceil(x)	天花板	Math.min(x,y)	x,y中较小者
Math.cos(x)	余弦	Math.pow(x,y)	x^y
Math.exp(x)	e^x	Math.sin(x)	正弦
Math.floor(x)	地板	Math.sqrt(x)	平方根
Math.log(x)	自然对数	Math.tan(x)	正切

4 高精度运算

- 一般类型初始化：`double a=Double.parseDouble(str.trim());`
- 高精度初始化：`BigDecimal num1=new BigDecimal(s1.trim()), num2=BigDecimal.valueOf(1);`
- 运算方法：

BigInteger `a.add(b),a.subtract(b),a.multiply(b),a.divide(b), a.gcd(b),a.probablePrime(b,int rand),a.pow(b)`

BigDecimal `a.add(b),a.subtract(b),a.multiply(b),a.divide(b,int)`

5 日期处理

```
Date ddd=new Date();        //定义
long time=ddd.getTime();    //获取当前时间(从1970.1.1 0:00:00GMT以来的毫秒数)
time+=5*24*3600*1000;       //计算时不要忘记乘以1000
ddd.setTime(time);          //设置时间
System.out.println(ddd);
```

6 正则表达式

Character classes

XY	X followed by Y
(X)	X as a capturing group
[abc]	a, b, or c (simple class)
[^abc]	Any character except a, b, or c (negation)
[a-zA-Z]	a through z or A through Z, inclusive (range)
[a-d[m-p]]	a through d, or m through p: [a-dm-p] (union)
[a-z&&[def]]	d, e, or f (intersection)
[a-z&&[^bc]]	a through z, except for b and c: [ad-z] (subtraction)
[a-z&&[^m-p]]	a through z, and not m through p: [a-lq-z](subtraction)

Predefined character classes

.	Any character (may or may not match line terminators)
\d	A digit: [0-9]
\D	A non-digit: [^0-9]
\s	A whitespace character: [\t\n\x0B\f\r]
\S	A non-whitespace character: [^\s]
\w	A word character: [a-zA-Z_0-9]
\W	A non-word character: [^\w]

Boundary matchers

^	The beginning of a line
$	The end of a line
\b	A word boundary
\B	A non-word boundary
\A	The beginning of the input
\G	The end of the previous match
\Z	The end of the input but for the final terminator, if any
\z	The end of the input

Greedy quantifiers

X?	X once or not at all
X*	X zero or more times
X+	X one or more times
X{n}	X exactly n times
X{n,}	X at least n times
X{n,m}	X at least n but not more than m times

7 Java实现next_permutation

```
//Members of vals must be distinct
//Based on C++ next_permutation function
int[] nextperm(int[] vals) {
    int i = vals.length - 1;
    while (true) {
        int ii = i;
        i--;
        if (vals[i] < vals[ii]) {
            int j = vals.length;
            while (vals[i] >= vals[--j]);
            int temp = vals[i]; //Swap
            vals[i] = vals[j];
            vals[j] = temp;
            int begin = ii, end = vals.length - 1;
            while (end > begin) {
                int stemp = vals[end]; //Swap
                vals[end] = vals[begin];
                vals[begin] = stemp;
                end--;
                begin++;
            }
            return vals;
        } else if (vals[i] == vals[0]) {
            int begin = 0, end = vals.length - 1;
            while (end > begin) {
                int stemp = vals[end]; //Swap
                vals[end] = vals[begin];
                vals[begin] = stemp;
```

```
                end--;
                begin++;
            }
            return vals;
        }
    }
}
```

第二章 数据结构

2.1 Hash

【用于整数的Hash函数】
[说明] 用于整数的Hash
$ins()$ 如果插入成功返回-1, 否则返回冲突key
$inhash()$ 如果找到，返回key,否则返回-1
使用以前必须调用$hashinit()$
如果数据组数比较多,可以考虑使用$O(1)$初始化.
#define 也可以使用$const$,效率更高!
本Hash表使用位运算求mod来加快速度,综合效率高.

```
typedef int hdata;
#define SIZE (1<<20)
#define BUF (1<<19)
struct Hash{hdata val;int next;}hash[SIZE+BUF];
bool flag[SIZE];
int htop;
void hashinit(){memset(flag,false,sizeof(flag));htop=SIZE;}
int ins(hdata val)
{
    int key=val&(SIZE-1);
    if(!flag[key])
    {
        flag[key]=true;
        hash[key].val=val;
        hash[key].next=-1;
        return -1;
    }
    while(true)
    {
        if(hash[key].val==val)return key;
        if(hash[key].next==-1)break;
        key=hash[key].next;
    }
    hash[key].next=++htop;
    hash[htop].next=-1;
    hash[htop].val=val;
    return -1;
```

```
}
int inhash(hdata val)
{
    int key=val&(SIZE-1);
    if(!flag[key])return -1;
    while(key!=-1)
    {
        if(hash[key].val==val)return key;
        key=hash[key].next;
    }
    return -1;
}
```

【字符串用的Hash函数】

```
#define Size 1001  //Size要用大质数,1811,5087,10657,50503,100109,500119

int ELFhash(char *key) {
    unsigned long h=0, g;
    while (*key)
    {
        h=(h<<4)+(*key++);
        g=h & 0xf0000000L;
        if (g) h^=g>>24;
        h&=~g;
    }
    return h;
}

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

    while (*str)
        hash = hash * seed + (*str++);

    return (hash & 0x7FFFFFFF);
}

// DJB Hash Function
unsigned int DJBHash(char *str)
{
    unsigned int hash = 5381;

    while (*str)
        hash += (hash << 5+hash) + (*str++);

    return (hash & 0x7FFFFFFF);
}


// AP Hash Function
```

```
unsigned int APHash(char* str)
{
   unsigned int hash = 0xAAAAAAAA;
   unsigned int i    = 0;

   for(i = 0; *str; str++, i++)
   {
      hash ^= ((i & 1) == 0) ? (  (hash <<  7) ^ (*str) * (hash >> 3)) :
                               (~((hash << 11) + ((*str) ^ (hash >> 5))));
   }

   return hash;
}

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

2.2 堆

【最小堆】可修改比较函数和元素类型,注意push和top操作要确保是非空堆

```
typedef int elem_t;
#define _cp(a, b) ((a) > (b))
#define MAXN 10000
struct heap {
    elem_t h[MAXN];
    int n;
    heap() { n =0; }
    heap(elem_t a[], int l):n(l) {
        for(int i = 0;i < l;++ i)  h[i+1] = a[i];
        for(int i = n >> 1;i;-- i) percolate_down(i);
    }
    bool empty() { return n == 0; }
    int size() { return n; }
    void clear() { n = 0; }
    void pop() {
        if(n == 0) return ;
        h[1] = h[n--];
        percolate_down(1);
    }

    elem_t top() { return h[1]; }
    void push(elem_t x) {
        int hole = ++ n;
        for(;hole > 1 && _cp(x, h[hole>>1]);hole >>= 1) h[hole] = h[hole>>1];
        h[hole] = x;
    }
    void percolate_down(int hole) {
```

```
        int child;
        elem_t tmp = h[hole];
        for(;(hole << 1) <= n;hole = child) {
            child = hole << 1;
            if(child != n && _cp(h[child+1], h[child])) ++ child;
            if(_cp(h[child], tmp)) h[hole] = h[child];
            else break;
        }
        h[hole] = tmp;
    }
};
```

【映射二分堆】可插入,获取并删除任意元素,复杂度均$O(logn)$
[说明] 插入时提供一个索引值,删除时按该索引删除,获取并删除最小元素时一起获得该索引.
[说明] 索引值ind范围0..MAXN-1,不能重复,不负责维护索引的唯一性,不在此返回请另外映射.
[说明] 主要用于图论算法,该索引值可以是节点的下标.可更改元素类型,修改比较符号或换成比较函数

```
const int MAXN = 10000 + 5;
#define _cp(a, b) ((a) < (b))
typedef int elem_t;

struct minheap {
    elem_t h[MAXN];
    int ind[MAXN], mp[MAXN], n, p, c;
    void init() { n = 0; }
    void ins(int i, elem_t e) {
      for(p = ++ n;p > 1 && _cp(e, h[p>>1]);p >>= 1)
        h[mp[ind[p] = ind[p>>1]] = p] = h[p>>1];
      h[mp[ind[p] = i] = p] = e;
    }
    int del(int i, elem_t &e) {
      i = mp[i]; if(i < 1 || i > n) return 0;
      for(e=h[p=i];p>1;h[mp[ind[p]=ind[p>>1]]=p]=h[p>>1],p>>=1);
      for(c = 2;c < n && _cp(h[c+=(c<n-1 && _cp(h[c+1], h[c]))], h[n]);p = c, c<<=1)
        h[mp[ind[p]=ind[c]]=p]=h[c];
      h[mp[ind[p]=ind[n]]=p]=h[n];n--;return 1;
    }
    int delmin(int &i, elem_t &e) {
      if(n < 1) return 0; i = ind[1]; e=h[p=1];
      for(c = 2;c < n && _cp(h[c+=(c<n-1 && _cp(h[c+1], h[c]))], h[n]);p = c, c<<=1)
        h[mp[ind[p]=ind[c]]=p]=h[c];
      h[mp[ind[p]=ind[n]]=p]=h[n];n--;return 1;
    }
    int top(int &i, elem_t &e) {
      i = ind[1]; e = h[1]; return n > 0;
    }
};
```

【左偏树】支持基本二叉堆的基本功能，但多一个合并堆的功能，效率为$O(logn)$

```
struct NODE {
    int key, dist;
    NODE * pl, *pr, *pf; //左右儿子节点指针和父亲节点指针
```

```
    NODE(){}
    NODE(int k) { key = k; pl = pr = pf = NULL; }
};

NODE mem[N];
int mem_pos;

inline NODE * new_node()
{
    NODE *pt = mem + (++ mem_pos);
    memset(pt, 0, sizeof(NODE));
    return pt;
}

//清除节点信息
inline void clear(NODE *pos) {
    if(pos == NULL) return ;
    pos->pl = pos->pr = pos->pf = NULL;
    pos->dist = 0;
}

//根据树中某个节点，获取跟节点
inline NODE * get_father(NODE *pt)
{
    while(pt->pf) pt = pt->pf;
    return pt;
}
//合并堆O(log N)
NODE * merge(NODE * pa, NODE *pb)
{
    if(pa == NULL) return pb;
    if(pb == NULL) return pa;
     //maximum vertex heap
    if(pa->key < pb->key) swap(pa, pb);
    pa->pr = merge(pa->pr, pb);
    if(pa->pr && (pa->pl == NULL || pa->pl->dist < pa->pr->dist))
        swap(pa->pl, pa->pr);
    if(pa->pr == NULL) pa->dist = 0;
    else pa->dist = pa->pr->dist + 1;
    if(pa->pl) pa->pl->pf = pa;
    if(pa->pr) pa->pr->pf = pa;
    return pa;
}

//插入节点O(log N)
inline NODE * insert(NODE * root, NODE *v)
{
    return merge(root, v);
}

//删除最大值O(log N)
inline NODE * delete_max(NODE *root)
{
```

```
    NODE *pt = root;
    root = merge(root->pl, root->pr);
    if(root) root->pf = NULL;
    clear(pt);
    return root;
}

//获得最大值O(1)
inline int get_max(NODE *root)
{
    return root->key;
}

//构建左偏树O(N)
inline NODE * make_left_tree(int val[], int n)
{
    queue<NODE *> Q;
    int i;
    NODE *ptemp = new_node();
    for(i = 0;i < n;++ i) {
        ptemp->key = val[i];
        Q.push(ptemp);
    }
    while(!Q.empty()) {
        int l = Q.size();
        if(l == 1) return Q.front();
        while(l --) {
            NODE *pa = Q.front();
            Q.pop();
            NODE *pb = Q.front();
            Q.pop();
            Q.push(merge(pa, pb));
        }
    }
}

//删除已知任意点O(log N)
inline void delete_any(NODE * pos)
{
    NODE * ppre = pos->pf;
    NODE * pnew = delete_max(pos);
    if(pnew) pnew->pf = ppre;
    if(ppre) {
        if(ppre->pl == pos) ppre->pl = pnew;
        else ppre->pr = pnew;
    }
    while(ppre) {
        int vl = -1, vr = -1;
        if(ppre->pl) vl = ppre->pl->dist;
        if(ppre->pr) vr = ppre->pr->dist;
        if(vl < vr) swap(ppre->pl, ppre->pr);
        if(vr + 1 == ppre->dist) return ;
        ppre->dist = vr + 1;
```

```
        pnew = ppre; ppre = ppre->pf;
    }
}

//另外一个左偏树模板
class leftlist {
private:
    node *ROOT;
    int tot;
    inline int Get_dist(node *a) { return a == NULL ? -1 : a->dist; };
    void SWAP(node *&a, node *&b) { node *tmp = a; a = b; b = tmp; };
    node *Merge(node *a, node *b) {
        if (a == NULL) return b;
        if (b == NULL) return a;
        if (a->key < b->key) SWAP(a, b);
        a->rch = Merge(a->rch, b);
        if (Get_dist(a->lch) < Get_dist(a->rch)) SWAP(a->lch, a->rch);
        a->dist = Get_dist(a->rch) + 1;
        return a;
    };
public:
    inline void clear() { ROOT = NULL; tot = 0; }
    inline  node *root() { return ROOT; };
    inline int size() { return tot; };
    inline bool empty() { return tot == 0; };
    inline int insert(int k) { ROOT = Merge(ROOT, new node(k)); tot++; };
    inline int top() { return ROOT != NULL ? ROOT->key : -0x7fffffff; };
    inline int pop() {
        if (ROOT == NULL) return 1;
        node *tmp = ROOT;
        ROOT = Merge(ROOT->lch, ROOT->rch);
        delete tmp;
        tot--;
    };
    inline int merge(leftlist &t) {
        tot += t.size();
        ROOT = Merge(ROOT, t.root());
        t.clear();
    };
};
```

2.3 常用数据结构

【树形并查集】如果空间不够rank 数组可以不用,此时uniont(a,b) 等价于parent[find(a)]=find(b);

```
#define MAXN 100000
struct ufind {
    int parent[MAXN], rank[MAXN];
    void init(int n) { for (int i=0;i<n;i++) {parent[i]=i;rank[i]=0;} }
    int find(int x)
    {
        int r = x, q;
```

```
        while(parent[r] != r) r = parent[r];
        while(x != r) q = parent[x], parent[x] = r, x = q;
        return r;
    }
    void uniont(int a, int b)
    {
        a = find(a), b = find(b);
        if(rank[a] > rank[b]) parent[b] = a;
        else {
            parent[a] = b;
            if(rank[a] == rank[b]) ++ rank[b];
        }
    }
};
```

【trie树】使用前请先调用init()函数，可根据具体情况修改get_idx()

```
const int CN = 26;          //字符集大小
const int TRIEN = 80000; //节点最大数
struct Trie {
  struct NODE {
   int idx, c[CN], ord;
   void init() { memset(this, -1, sizeof(*this)); }
  };

  NODE mem[TRIEN];
  int posn, cnt;
  void init() { posn = 1; mem[0].init(); cnt = 0;}
  int get_idx(char &e) { return e - 'a'; }
  int insert(char *s, int i = 0) {
    for(int r;*s;++ s, i = mem[i].c[r])
      if(mem[i].c[r = get_idx(*s)] == -1)
        mem[(mem[i].c[r] = posn++)].init();
    if(mem[i].ord == -1) mem[i].ord = cnt ++;
    return mem[i].ord;
   }

  //返回出现的序号，未出现返回-1
  int search(char *s, int i = 0) {
    for(int r;*s;++ s, i = mem[i].c[r])
      if(mem[i].c[r = get_idx(*s)] == -1) return -1;
    return mem[i].ord;
  }
};
```

【AC自动机】

[说明] 使用时先调用$PreProcess()$然后插入字符串,再调用BFS函数建立失败指针

```
const int TSIZE=100+5;  //状态个数上限=单词总数*单词长度
const int CN=4;         //可选择字符数
struct Trie             //Trie 结构
{
    int c[CN];
```

```
    int fail;            //fail指针
    bool flag;           //状态标志(有时需要int类型)
    void init()           //节点初始化
    {
        memset(c,0,sizeof(c));
        fail=-1;
        flag=0;
    }
};
Trie trie[TSIZE];
int len;                //当前长度
void PreProcess()
{
    trie[0].init();
    len=1;
}
int Get(char ch)  //根据具体题目修改
{
    switch (ch)
    {
        case 'A':return 0;break;
        case 'T':return 1;break;
        case 'C':return 2;break;
        case 'G':return 3;break;
    }
}
void Insert(char *str,bool id)  //插入单词,标志
{
    int r,no;
    int i;
    r=0;
    for (i=0;str[i];i++)
    {
        no=Get(str[i]);
        if (!trie[r].c[no])
        {
            trie[len].init();
            trie[r].c[no]=len++;
        }
        r=trie[r].c[no];
    }
    trie[r].flag=id;
}
queue<int>que;
void BFS()                //利用BFS建立失败指针,构造trie图
{
    while (!que.empty()) que.pop();
    que.push(0);         //根进队列
    int cur,fail,pos;
    int i;
    while (!que.empty())
    {
        cur=que.front();
```

```
        que.pop();
        for (i=0;i<CN;i++)
            if (trie[cur].c[i])
            {
                pos=trie[cur].c[i];
                que.push(pos);
                fail=trie[cur].fail;
                while (fail!=-1 && !trie[fail].c[i]) fail=trie[fail].fail;
                if (fail!=-1)
                    trie[pos].fail=trie[fail].c[i];    //连接失败指针
                else
                    trie[pos].fail=0;

             //必要时可以在此处添加一句处理标志,可以避免以后查找需要遍历失败指针
             //例如trie[pos].flag|=trie[trie[pos].fail].flag;
            }
            else
            {
                 if (trie[cur].fail!=-1)                    //完善trie图
                     trie[cur].c[i]=trie[trie[cur].fail].c[i];
            }
    }
}
```

【Dancing Links 完美覆盖】

[说明] $solve()$ 返回是否存在精确覆盖，使用时传入参数分别为行数n, 列数m（编号均从1开始），01矩阵$DL[][]$

[说明] 经典例题：$POJ3740$, $HUST1017$

```
const int MAXN = 1005;

int L[MAXN*MAXN], R[MAXN*MAXN], U[MAXN*MAXN], D[MAXN*MAXN];
int S[MAXN];
int Col[MAXN*MAXN], Row[MAXN*MAXN];

void Remove(int c) {
    L[R[c]] = L[c];
    R[L[c]] = R[c];
    for (int i = D[c]; i != c; i = D[i])
        for (int j = R[i]; j != i; j = R[j]) {
            U[D[j]] = U[j];
            D[U[j]] = D[j];
            -- S[Col[j]];
        }
}

void Resume(int c) {
    for (int i = U[c]; i != c; i = U[i])
        for (int j = L[i]; j != i; j = L[j]) {
            U[D[j]] = j;
```

```
            D[U[j]] = j;
            ++ S[Col[j]];
        }
    L[R[c]] = c;
    R[L[c]] = c;
}

bool dfs(int depth) {
    if (R[0] == 0) return true;

    int i, j, c, minnum = INT_MAX;
    for (i = R[0]; i != 0; i = R[i]) {
        if (S[i] < minnum) {
            minnum = S[i];
            c = i;
        }
    }
    Remove(c);
    for (i = U[c]; i != c; i = U[i]) {
        //如果需要的话，在这里记录一组解(Ans[depth] = Row[i])
        for (j = R[i]; j != i; j = R[j]) Remove(Col[j]);
        if (dfs(depth + 1)) return true;
        for (j = L[i]; j != i; j = L[j]) Resume(Col[j]);
    }
    Resume(c);
    return false;
}

int solve(int n, int m, int DL[][MAXN]) {
    for (int i = 0; i <= m; i ++) {
        L[i] = i - 1;
        R[i] = i + 1;
        U[i] = D[i] = i;
    }
    L[0] = m;
    R[m] = 0;

    int cnt = m + 1;
    memset(S, 0, sizeof (S));
    for (int i = 1; i <= n; i ++) {
        int head = cnt, tail = cnt;
        for (int c = 1; c <= m; c ++) if (DL[i][c]) {

            S[c] ++;
            Row[cnt] = i;
            Col[cnt] = c;

            U[D[c]] = cnt;
            D[cnt] = D[c];
            U[cnt] = c;
            D[c] = cnt;

            L[cnt] = tail;
```

```
            R[tail] = cnt;
            R[cnt] = head;
            L[head] = cnt;
            tail = cnt;

            cnt ++;
        }
    }
    if (dfs(0)) return true;
    return false;
}
```

【Dancing Links 重复覆盖】
[说明] solve()返回最优值，使用时传入参数分别为行数n, 列数m（编号均从1开始），01矩阵$DL[][]$，以及最优值可能的最大范围
[说明] 经典例题：$FOJ1686$

```
const int MAXN = 225;

int L[MAXN*MAXN], R[MAXN*MAXN], U[MAXN*MAXN], D[MAXN*MAXN];
int S[MAXN];
int Col[MAXN*MAXN];
int limit;

void Remove(int x) {
    for (int i = D[x]; i != x; i = D[i]) {
        L[R[i]] = L[i];
        R[L[i]] = R[i];
    }
}

void Resume(int x) {
    for (int i = U[x]; i != x; i = U[i]) {
        L[R[i]] = R[L[i]] = i;
    }
}

int Hash() {
    int ans = 0;
    bool hash[MAXN] = {0};

    for (int c = R[0]; c != 0; c = R[c])
        if (! hash[c]) {
            hash[c] = true;
            ans ++;
            for (int i = D[c]; i != c; i = D[i])
                for (int j = R[i]; j != i; j = R[j])
                    hash[Col[j]] = true;
        }
    return ans;
}
```

```
bool dfs(int depth) {
    if (depth + Hash() > limit) return false;

    if (R[0] == 0) return true;

    int i, j, c, minnum = INT_MAX;
    for (i = R[0]; i != 0; i = R[i]) {
        if (S[i] < minnum) {
            minnum = S[i];
            c = i;
        }
    }

    for (i = U[c]; i != c; i = U[i]) {
        Remove(i);
        for (j = R[i]; j != i; j = R[j]) Remove(j);
        if (dfs(depth + 1)) {
            for (j = L[i]; j != i; j = L[j]) Resume(j);
            Resume(i);
            return true;
        }
        for (j = L[i]; j != i; j = L[j]) Resume(j);
        Resume(i);
    }
    return false;
}

int solve(int n, int m, int DL[][MAXN], int maxdepth) {
    if (maxdepth > n) maxdepth = n;
    for (int i = 0; i <= m; i ++) {
        L[i] = i - 1;
        R[i] = i + 1;
        U[i] = D[i] = i;
    }
    L[0] = m;
    R[m] = 0;

    int cnt = m + 1;
    memset(S, 0, sizeof (S));
    for (int i = 1; i <= n; i ++) {
        int head = cnt, tail = cnt;
        for (int c = 1; c <= m; c ++) if (DL[i][c]) {

                S[c] ++;
                Col[cnt] = c;

                U[D[c]] = cnt;
                D[cnt] = D[c];
                U[cnt] = c;
                D[c] = cnt;

                L[cnt] = tail;
                R[tail] = cnt;
```

```
                R[cnt] = head;
                L[head] = cnt;
                tail = cnt;

                cnt ++;
            }
    }
    int best = 0, worst = maxdepth;
    while (best <= worst) {
        limit = (worst + best) >> 1;
        if (dfs(0)) worst = limit - 1;
        else best = limit + 1;
    }
    return best;
}
```

【Size Banlance Tree - SBT】平衡二叉搜索树,插入$O(logN)$,删除$O(logN)$
[说明] 定义结构体以后必须先调用$init()$
[说明] 所有的Insert, Del操作传入的(int &t) 都填写root

```
const int maxn = 1 << 18;
struct SBT{
    int root,L[maxn],R[maxn],s[maxn],idx,val[maxn];
    void init(){root = idx = 0;}
    void RR(int &t){
        int k = L[t];L[t]=R[k];R[k]=t;
        s[k]=s[t];s[t]=s[L[t]]+s[R[t]] + 1;
        t = k;
    }
    void LR(int &t){
        int k = R[t];R[t] = L[k];L[k] = t;
        s[k]=s[t];s[t]=s[L[t]]+s[R[t]] + 1;
        t = k;
    }
    void MT(int &t){
        if(s[L[L[t]]] > s[R[t]]) RR(t),MT(R[t]),MT(t); else
        if(s[R[L[t]]] > s[R[t]]) LR(L[t]),RR(t),MT(L[t]),MT(R[t]),MT(t);else
        if(s[R[R[t]]] > s[L[t]]) LR(t),MT(L[t]),MT(t); else
        if(s[L[R[t]]] > s[L[t]]) RR(R[t]),LR(t),MT(L[t]),MT(R[t]),MT(t);
    }
    void Insert(int &t,int x){
        if(! t){
            t = ++ idx;
            val[t] = x;
            s[t] = 1; L[t] = R[t] = 0;
            return ;
        }
        ++ s[t];
        if(x < val[t]) Insert(L[t],x); else Insert(R[t],x);
        MT(t);
    }
    int Del(int &t,int x){
        int ret;
        -- s[t];
```

```
        if(x == val[t] || (x < val[t] && L[t] == 0) || (x > val[t] && R[t] == 0)){
            ret = val[t];
            if(L[t] && R[t]) val[t] = Del(L[t],x + 1);else t = L[t] + R[t];
            return ret;
        }
        if(x < val[t]) return Del(L[t],x); else return Del(R[t],x);
    }
};
```

【败者树】建树$O(n)$,查询$O(\log n)$, 更新$O(\log n)$
[说明] query(s, t) 返回s,t中的最小值.可更改数据类型和比较函数
[注意]使用前请现调用$setN$函数

```
typedef int elem_t;
const int MAXN = 2000005;

#define _cp(a, b) ((a) < (b))
struct loserTree {
    elem_t  f[MAXN<<1];
    int N;
    void setN(int n) { for(N = 1;N < n;N <<= 1); }
    void build(elem_t a[], int n) { // 建树，a[0..n-1]存在f[N..N+n-1]
        memcpy(f + N, a, sizeof(a[0]) * n);
        for (int i = N, j = N + n - 1; i > 1; i >>= 1, j >>= 1) {
            for (int k = i; k < j; k += 2)
                    f[k >> 1] = _cp(f[k], f[k + 1]) ? f[k] : f[k + 1];
            if (!(j&1)) f[j >> 1] = f[j];
        }
    }
    int query(int s, int t) { //查询
        s += N, t += N;
        int r = (_cp(f[s], f[t]) ? s : t);
        for (int i = s, j = t; i < j; i >>= 1, j >>= 1) {
            if (!(i & 1) && i + 1 < j && _cp(f[i + 1], f[r])) r = i + 1;
            if ((j & 1) && j - 1 > i && _cp(f[j - 1], f[r])) r = j - 1;
        } //return f[r];    //f[r]为[s,t]中最值
        while (r < N) {
            if (f[r] == f[r << 1]) r <<= 1;
            else if (f[r] == f[(r<<1) + 1]) r = (r<<1) + 1;
        }
        return r - N;
    }
    void update(int k, int best) { //更新
        for(k += N, f[k] = best;k > 0 && _cp(f[k], f[k>>1]);k>>=1)
            f[k>>1] = f[k];
    }
};
```

【二维线段树】
[说明] 插入点,查询区间,区别为左开右闭,cnt存此区间的点数
[说明] 使用前请先调用$init()$,$minx$,$maxx$,$miny$,$maxy$为点的上下限
[说明] [经典例题]:ZOJ 3018 $Population$ / HDU 3373 $Point$

```
struct Seg
{
    int a,b,c,d,cnt;
    void clear()
    {
        a=b=c=d=-1;
        cnt=0;
    }
}seg[maxn*40];

int tot,root;
int minx,maxx,miny,maxy;

void init()
{
    root=-1;
    tot=0;
}

void add(int nowx,int nowy,int minx,int maxx,int miny,int maxy,int inc,int &v)
{
    if (nowx<minx || nowx>=maxx || nowy<miny || nowy>=maxy) return ;
    if (v==-1)
    {
        v=tot++;
        seg[v].clear();
    }
    seg[v].cnt+=inc;
    if (minx+1==maxx && miny+1==maxy)
        return;

    int midx=(minx+maxx)>>1;
    int midy=(miny+maxy)>>1;

    add(nowx,nowy,minx,midx,miny,midy,inc,seg[v].a);
    add(nowx,nowy,midx,maxx,miny,midy,inc,seg[v].b);
    add(nowx,nowy,minx,midx,midy,maxy,inc,seg[v].c);
    add(nowx,nowy,midx,maxx,midy,maxy,inc,seg[v].d);
}

int find(int sx,int tx,int sy,int ty,int minx,int maxx,int miny,int maxy,int v)
{
    if (v==-1  || seg[v].cnt==0 || sx>=maxx || tx<=minx || sy>=maxy || ty<=miny)
        return 0;

    if (sx<=minx && maxx<=tx && sy<=miny && maxy<=ty)
        return seg[v].cnt;

    int res=0;

    int midx=(minx+maxx)>>1;
    int midy=(miny+maxy)>>1;
```

```
    res+=find(sx,tx,sy,ty,minx,midx,miny,midy,seg[v].a);
    res+=find(sx,tx,sy,ty,midx,maxx,miny,midy,seg[v].b);
    res+=find(sx,tx,sy,ty,minx,midx,midy,maxy,seg[v].c);
    res+=find(sx,tx,sy,ty,midx,maxx,midy,maxy,seg[v].d);

    return res;
}
```

【RMQ】预处理$O(nlogn)$,查询$O(1)$
[说明] RMQ查询a[i]...a[j]间的最小值，返回最小元素的下标，从a[0]开始

```
//M =(int)((log(N) / log(2.00)) + 1);
//st[j][i] 表示从 i 起连续 2^j 次方个数的最大值
int st[M][N];

void make_st(int n, int a[]) {
    int i, j;
    for (i = 0; i < n; ++i) st[0][i] = i;
    for (j = 1; (1 << j) <= n; ++j)
        for (i = 0; i + (1 << j) <= n; ++i) {
            int p = st[j - 1][i], q = st[j - 1][i + (1 << (j - 1))];
            st[j][i]  = (a[p] < a[q] ? p : q);
        }
}

//返回RMQi, j 间最小元素的下标
int RMQ(int i, int j, int a[]) {
    if (i > j) swap(i, j);
    int k ;
    for(k = 0;(1 << k) <= (j - i + 1);++ k); -- k;
    i = st[k][i], j = st[k][j - (1 << k) + 1];
    return (a[i] < a[j] ? i : j);
}
```

【二维RMQ】
[说明] RMQ查询$matrix[x1][y1]...matrix[x2][y2]$间的最大(小)值，返回最大元素的值,原始矩阵Matrix下标从1开始。

```
const int M = 9;
const int N = 300+1;
int Matrix[N][N];
int st[M][M][N][N];
int POW[]={1,1<<1,1<<2,1<<3,1<<4,1<<5,1<<6,1<<7,1<<8,1<<9,1<<10};
inline int Max(int a,int b) {  return a>b?a:b; }
void make_rmq(int n,int m,int b[][N]) {
    int row,col,i,j;
    for(row=1;row<=n;++row)
    {
        for(col=1;col<=m;++col)
        {
            st[0][0][row][col]=b[row][col];
        }
    }
    for(i=0;POW[i]<=n;++i)
```

```
    {
    for(j=0;POW[j]<=m;++j)
    {
     if(i==0 && j==0) continue;
     for(row=1;row+POW[i]-1<=n;++row)
     {
       for(col=1;col+POW[j]-1<=m;++col)
       {
        if(i==0)
         st[i][j][row][col]=Max(st[i][j-1][row][col],st[i][j-1][row][col+POW[j-1]]);
        else
         st[i][j][row][col]=Max(st[i-1][j][row][col],st[i-1][j][row+POW[i-1]][col]);
       }
     }
    }
   }
}
inline int get_k(int a,int b) {
    return (int)(log((b-a+1)*1.0)/log(2.0));
}

//返回RMQ (x1,y1),(x2,y2) 间最大元素的值
inline int RMQ(int x1,int x2,int y1,int y2) {
    if(x1>x2)swap(x1,x2);
    if(y1>y2)swap(y1,y2);
    int kx=get_k(x1,x2),ky=get_k(y1,y2),Max1,Max2;
    Max1 = Max(st[kx][ky][x1][y1],st[kx][ky][x1][y2-POW[ky]+1]);
    Max2 = Max(st[kx][ky][x2-POW[kx]+1][y1],st[kx][ky][x2-POW[kx]+1][y2-POW[ky]+1]);
    return Max(Max1,Max2);
}
```

【树状数组】维护和查询复杂度均为O(logn)

[说明] 支持元素动态改变,query(i)返回$a[0..i-1]$的和.使用前请先调用init(i).可更改数据类型

```
#define lowbit(x) ((x)&((x)^((x)-1)))
#define MAXN 10000
typedef int elem_t;

struct indexTree{
    elem_t a[MAXN],c[MAXN],ret;
    int n;
    void init(int i){memset(a,0,sizeof(a));memset(c,0,sizeof(c));n=i;}
    void update(int i,elem_t v){for (v-=a[i],a[i++]+=v;i<=n;c[i-1]+=v,i+=lowbit(i));}
    elem_t query(int i){for (ret=0;i;ret+=c[i-1],i^=lowbit(i));return ret;}
};
```

【二维树状数组】维护和查询复杂度均为O(logm*logn)

[说明] query(i,j)返回$suma[0..i-1][0..j-1]$,用于动态求子阵和,数组内容保存在$a[][]$中

```
#define lowbit(x) ((x)&((x)^((x)-1)))
#define MAXN 100
typedef int elem_t;

struct indexTree{
```

```
    elem_t a[MAXN][MAXN],c[MAXN][MAXN],ret;
    int m,n,t;
    void init(int i,int j){memset(a,0,sizeof(a));memset(c,0,sizeof(c));m=i,n=j;}
    void update(int i,int j,elem_t v){
        for (v-=a[i][j],a[i++][j++]+=v,t=j;i<=m;i+=lowbit(i))
            for (j=t;j<=n;c[i-1][j-1]+=v,j+=lowbit(j));
    }
    elem_t query(int i,int j){
        for (ret=0,t=j;i;i^=lowbit(i))
            for (j=t;j;ret+=c[i-1][j-1],j^=lowbit(j));
        return ret;
    }
};
```

【点树】具有树状数组的所有功能，但又比树状数组强，可直接在log(N)的复杂度内查找第k小元素。但时间效率及空间效率都比树状数组差一些。使用前请先调用setQ函数。

```
class  PointTree
{
  int a[N], Q; //N至少为最大值的两倍
public:
  PointTree() { Q = 0; }
  void setQ(int n) { for (Q = 1; Q < n; Q<<=1) ; } //可用区间为[0, n)
  void clear() { memset(a, 0, (Q<<1)*sizeof(a[0])); }
  int getSize() { return a[1]; } //元素的个数
  void Update(int n, int c=1) //增加n的个数，c可为负数
  {
    for (int i=Q+n; i; i>>=1) a[i]+=c;
  }
  int kth(int k) //查找第k小元素
  {
    int i = 1;
    while (i < Q) if (k > a[i<<=1]) k-=a[i++];
    return i-Q;
  }
  int kth_large(int k) { return kth(a[1]-k+1); } //查找第k大元素
  int cntLs(int n) //统计小于n的个数
  {
    int c = 0;
    for (n+=Q; n > 1; n>>=1) if (n&1) c+=a[n-1];
    return c;
  }
  int cntGt(int n) { return a[1]-a[n+Q]-cntLs(n); } //统计大于n的个数
  int cntEq(int n) { return a[n+Q]; } //统计等于n的个数
};
```

2.4 后缀数组

1 构造后缀数组

统一说明：

s[]　原始数组，
sa[]　后缀数组存放，
H[]　lcp，其中H[i] = lcp(sa[i], sa[i-1]),故其起点是从1开始，不是从0开始
R[]　名次数组
n 原始数组长度，
K 元素取值范围 (原始数组最后一个存 0 ，其它存1...K)
调用方法 suffix_array(s, n, 300); 300是数值范围，根据需要自行修改

【三分法O(n)】

```
s[n] = s[n+1] = s[n+2] = 0;  //调用下面这个构造法前现加上这句

int s[N], sa[N], R[N], H[N];  // 注意s整数数组，输入字符串的话要赋值到整数数组中

inline bool leq(int a1, int a2, int b1, int b2)
{
    return a1 < b1 || a1 == b1 && a2 <= b2;
}

inline bool leq(int a1, int a2, int a3, int b1, int b2, int b3)
{
    return a1 < b1 || a1 == b1 && leq(a2, a3, b2, b3);
}

void radix_sort(int a[], int b[], int r[], int n, int K)
{
    int i, sum, t, *c = new int[K+1];
    memset(c, 0, sizeof(int) * (K + 1));
    for(i = 0;i < n;++ i) ++ c[r[a[i]]];
    for(i = sum = 0;i <= K;++ i) t = c[i], c[i] = sum, sum += t;
    for(i = 0;i < n;++ i) b[c[r[a[i]]]++] = a[i];
    delete [] c;
}

void suffix_array(int s[], int sa[], int n, int K)
{
    int i, j, k, p, t;
    int n0 = (n + 2) / 3, n1 = (n + 1) / 3, n2 = n / 3, n3 = n0 + n2;
    int *s2 = new int [n3 + 3]; s2[n3] = s2[n3+1] = s2[n3+2] = 0;
    int *s1 = new int [n3 + 3]; s1[n3] = s1[n3+1] = s1[n3+2] = 0;
    int *s0 = new int [n0], *sa0 = new int [n0];
    for(i = j = 0;i < n + (n0 - n1);++ i)
        if(i % 3 != 0) s2[j++] = i;
    radix_sort(s2, s1, s + 2, n3, K);
    radix_sort(s1, s2, s + 1, n3, K);
    radix_sort(s2, s1, s    , n3, K);
    int name = 0, c0 = -1, c1 = -1, c2 = -1;
    for(i = 0;i < n3;++ i) {
        if(s[s1[i]] != c0 || s[s1[i] + 1] != c1 || s[s1[i]+2] != c2)
            ++ name, c0 = s[s1[i]], c1 = s[s1[i]+1], c2 = s[s1[i] + 2];
        if(s1[i] % 3 == 1) s2[s1[i]/3] = name;
        else s2[s1[i]/3 + n0] = name;
    }
```

```
   if(name < n3) {
        suffix_array(s2, s1, n3, name);
        for(i = 0;i < n3;++ i) s2[s1[i]] = i + 1;
    }
    else for(i = 0;i < n3;++ i) s1[s2[i]-1] = i;
    for(i = j = 0;i < n3;++ i) if(s1[i] < n0) s0[j++] = 3 * s1[i];
    radix_sort(s0, sa0, s, n0, K);
    for(p = 0, t = n0 - n1, k = 0;k < n;++ k) {
        #define GetI() (s1[t] < n0 ? s1[t] * 3 + 1 : (s1[t] - n0) * 3 + 2)
        i = GetI();
        j = sa0[p];
        if(s1[t] < n0 ? leq(s[i], s2[s1[t] + n0], s[j], s2[j/3]) :
            leq(s[i], s[i+1], s2[s1[t] - n0 + 1], s[j], s[j+1], s2[j/3+n0])) {
            sa[k] = i, ++ t;
            if(t == n3) for(++k;p < n0;++p, ++k) sa[k] = sa0[p];
        }
        else {
            sa[k] = j; ++ p;
            if(p == n0)
                for(++ k;t < n3;++t, ++ k)
                    sa[k] = GetI();
        }
    }
    delete [] s2; delete [] s1; delete [] sa0; delete [] s0;
}
```

【倍增算法,O(n log n)】这个方法直接求出的R数组有时会错的，请调用get_height()求R[]

```
int smem[3][N], cnt[N], H[N];
int *R, *nR, *sa, *osa;

void suffix_array(int s[], int n, int F = 300)
{
    int i, j, k;
    R = smem[0], sa = smem[1], osa = smem[2];
    memset(cnt, 0, sizeof(int) * F);
    //计数排序
    for(i = 0;i < n;++ i) ++ cnt[s[i]];
    for(i = 1;i < F;++ i) cnt[i] += cnt[i-1];
    for(i = 0;i < n;++ i) sa[--cnt[s[i]]] = i;
    //求Rank
    for(R[sa[0]] = 0, i = 1;i < n;++ i)
        R[sa[i]] = R[sa[i-1]] + (s[sa[i]] != s[sa[i-1]]?1:0);
    //倍增算法
   for(k = 1;k < n && R[sa[n-1]] < n - 1; k *= 2) {
        for(i = 0;i < n;++ i) cnt[R[sa[i]]] = i + 1;
        for(i = n - 1;i >= 0;-- i)
            if(sa[i] >= k) osa[--cnt[R[sa[i]-k]]] = sa[i] - k;
        for(i = n - k;i < n;++ i) osa[--cnt[R[i]]] = i;
        nR = sa, sa = osa;
        for(i = 1, nR[sa[0]] = 0;i < n;++ i) {
            nR[sa[i]] = nR[sa[i-1]];
            if(R[sa[i]] != R[sa[i-1]] || R[sa[i] + k] != R[sa[i-1]+k])
                ++ nR[sa[i]];
```

```
        }
        osa = R, R = nR;
    }
}
```

2 求H[], R[]

求H 和R 数组，保存在H 中, 注意H[]的起点是从1开始,即H[i] = lcp(sa[i], sa[i-1])

```
void get_height(int s[], int n)
{
    int i, j, k;
    for(i = 0;i < n;++ i) R[sa[i]] = i;
    for(j = i = 0;i < n;++ i) {
        if(R[i] == 0) H[0] = j = 0;
        else {
            for(k = sa[R[i] - 1], j = max(j - 1, 0); s[i+j] && s[i+j] == s[k+j];++ j);
            H[R[i]] = j;
        }
    }
}
```

3 后缀数组的一些应用

【2个字符串公共子串的长度】

```
int s[N];
int comm_2_str(char x[], char y[])
{
    int m = strlen(x), n = strlen(y), i, maxt = 0;
    for (i = 0; i <= m; ++i) s[i] = x[i];
    for (n += m + 2; i < n; ++i) s[i] = y[i - m - 1];
    s[n] = s[n + 1] = s[n + 2] = 0;
    suffix_array(s, n);
    get_height(s, n);
    for (i = 1; i < n; ++i)
        if (sa[i] > m && sa[i - 1] < m || sa[i] < m && sa[i - 1] > m)
            if (maxt < H[i]) maxt = H[i];
    return maxt;
}
```

【多个字符串的公共子串的长度】结合RMQ实现，调用comm_n_str(n, m)

```
//RMQ使用稀疏数组,int适当可以改成short
int st[18][N];
void make_st(int n, int a[])
{
    int i, j;
    for (i = 0; i < n; ++i) st[0][i] = a[i];
    for (j = 1; (1 << j) <= n; ++j)
        for (i = 0; i + (1 << j) <= n; ++i)
            st[j][i] = min(st[j - 1][i], st[j - 1][i + (1 << (j - 1))]);
```

```
}

inline int RMQ(int i, int j)
{
    if (i > j) swap(i, j);
    int k;
    for (k = 0; (1 << k) <= (j - i + 1); ++k);
    --k;
    return min(st[k][i], st[k][j - (1 << k) + 1]);
}

//pos 保存所有公共子串的起始位置,top表示公共子串的个数,祥使用见output函数
int s[N],p[N], pos[N], top;
char str[N];  //源串保存处

//求n个串中至少在m个串中出现的公共子串,返回公共子串长度
//当m==n时求这n个串的公共子串长度
int comm_n_str(int n, int m)
{
    int i, j, len, l, r, k, best, up;
    for (i = j = 0; i < n; ++i) {
        scanf("␣%s", str + j);
        while (str[j]) {
            s[j] = str[j];
            p[j++] = i;
        }
        s[j] = 0;
        p[j++] = i;
    }
    len = j;
    s[len] = s[len + 1] = s[len + 2] = 0;
    suffix_array(s, len);
    get_height(s, len);
    make_st(len, H);
    l = r = k = best = top = 0;
    memset(cnt, 0, sizeof (int) * n);
    while (r < len) {
        while (r < len && k <= m - 1)
            if (!cnt[p[sa[r++]]]++)
                ++k;
        while (k > m - 1)
            if (!--cnt[p[sa[l++]]])
                --k;
        up = RMQ(l, r - 1);
        if (up == best) pos[top++] = sa[l - 1];
        else if (up > best) {
            best = up;
            top = 0;
            pos[top++] = sa[l - 1];
        }
        if (up > best) best = up;
    }
    return best;
```

```
}

//输出公共串,输出的串已经是字典序的了
void output(int best)
{
    if (best == 0) puts("Not␣Common␣SubString");
    else {
        int i = 0, j;
        for (j = pos[i]; j < pos[i] + best; ++j)
            printf("%c", str[j]);
        puts("");
    }
}
```

【一个串中，重复出现可重叠的的子串长度】即为$max\{lcp[i], 1 <= i < n\}$
【一个串中，重复出现但不重叠的子串长度】使用二分答案方法

```
bool chk(int len)
{
    int i, k, l = n, r = 0, a, b;
    for (i = 1; i < n; ++i) {
        if (H[i] < len) l = n, r = 0;
        else {
            a = sa[i], b = sa[i - 1];
            if (a > b) swap(a, b);
            l = min(a, l);
            r = max(b, r);
            if (r - l >= len) return true;
        }
    }
    return false;
}

int repeat_no_cover(int n, int s[])
{
    suffix_array(s, n);
    get_height(s, n);
    int low = 0, up = n, mid;
    while (low < up) {
        mid = (low + up + 1) / 2;
        if (chk(mid)) low = mid;
        else up = mid - 1;
    }
    return low + 1;
}
```

【一个串中至少重复出现k次可重叠的最长字串长度】使用二分答案方法

```
bool chk(int n, int len, int ks) {
    int i, j, k = 1;
    for (i = 1; i < n && k < ks; ++i)
        if (H[i] < len) k = 1;
        else ++k;
    return k >= ks;
```

```
}

int repeat_k(int n, int k, int s[]) {
    suffix_array(s, n);
    get_height(s, n);
    int down = 0, up = n, mid;
    while (up > down) {
        mid = (up + down + 1) >> 1;
        if (chk(n, mid, k)) down = mid;
        else up = mid - 1;
    }
    return down;
}
```

【一个串中至少重复出现k次不可重叠的最长字串长度】

[说明] 二分答案,*mnp*,*reach*为全局变量,*reach*为长度,*mnp*为字典序最小的满足条件的起始位置

```
int mnp,reach;
int sp[maxn];
bool is_ok(int size,int k,int n)
{
    int cnt = 0;
    int start;
    int count = 0;
    bool ok = false;
    for ( int i = 1; i <= n; ++i )
    {
        if( H[i] < size )
        {
            if( cnt != 0 )
            {
                sort(sp, sp + cnt + 1);
                start = sp[0];
                count = 1;
                for ( int i = 1; i <= cnt; ++i )
                {
                    if( sp[i] >= start + size )
                        ++ count, start = sp[i];
                    if( count >= k )
                    {
                        if( size == reach && R[sp[0]] < R[mnp] )
                            mnp = sp[0];
                        else if( size > reach )
                            mnp = sp[0], reach = size;
                        ok = true;
                        break;
                    }
                }
            }
            cnt = 0;
        }
        else
        {
```

```
            if( cnt == 0 )
                sp[0] = sa[i-1];
            sp[++cnt] = sa[i];
        }
    }
    return ok;
}
void chk(int k,int n)
{
    s[n] = s[n+1] = s[n+2] = 0;
    suffix_array(s,n);
    get_height(s,n);
    int low = 0, high = n;
    mnp = -1, reach = 0;
    while ( low <= high )
    {
        int mid = (low + high) / 2;
        if( is_ok(mid,k,n) )
            low = mid + 1;
        else
            high = mid - 1;
    }
}
```

2.5 数据结构的典型应用

【最近公共祖先lca】

ST算法:时间复杂度$O(Nlog(N)) - O(1)$,空间复杂度$O(Nlog(N))$.为在线算法
$Tarjan$算法:时间复杂度$O(Na(N) + Q)$,空间复杂度$O(N)$.是离线算法
[注意]$tarjan$无固定代码，需更根据目灵活运用

```
//st算法
const int N = 100000;
int st[M][N * 2 + 1];
int Index[N * 2 + 1];
int level[N * 2 + 1];
int pos[N];
int ordcnt = 0;
vector<int> gra[N]; //树是基于邻接表的，可以用其他的。

bool mark[N]; //给的结构如果确认是一棵树，可以不用mark数组

//DFS
void dfs(int k, int depth) {
    Index[ordcnt] = k; //顺序记录遍历时的结点
    pos[k] = ordcnt; //记录每个结点第一次出现的位置
    level[ordcnt++] = depth; //记录当前遍历的结点的深度
    mark[k] = true;
    int i, next, sz = (int) gra[k].size();
    for (i = 0; i < sz; ++i) {
        next = gra[k][i];
```

```
        if (mark[next]) continue;
        dfs(next, depth + 1);
        Index[ordcnt] = k;
        level[ordcnt++] = depth;
    }
}

//返回i, j的最近公共祖先
int lca(int i, int j) {
    return Index[RMQ(pos[i], pos[j], level)];
}

//tarjan算法
const int MAXN = 100000;
vector<int> G[MAXN]; //树结构邻接表存储
vector<int> Q[MAXN]; //需要的查询,邻接表存储
int parent[MAXN];
bool vis[MAXN] = {0};

int find(int x) {
    int r = x, q;
    while (r != parent[r]) r = parent[r];
    while (x != r) {
        q = parent[x];
        parent[x] = r;
        x = q;
    }
    return r;
}

void uniont(int a, int b) {
    parent[find(b)] = find(a);
}

void tarjan(int pos) {
    int next, i, j;
    parent[pos] = pos;
    vis[pos] = true;
    for (i = 0; i < Q[pos].size(); ++i) {
        next = Q[pos][i];
        if (vis[next]) {
            //你的操作
            //此时find(next)为标号为next的节点与pos节点的lca
        }
    }
    for (i = 0; i < G[pos].size(); ++i) {
        next = G[pos][i];
        if (vis[next]) continue;
        tarjan(next);
        uniont(pos, next;
    }
}
```

第三章　基础算法

3.1　搜索算法

【搜索优化策略】

1. 找出问题无解的条件，避免搜索整个状态空间
2. 考虑极端情况，优化算法在最坏情况下的复杂度，适当进行预处理。
3. 数据有序化。就是将杂乱的数据，通过简单的分类和排序，变成有序的数据，从而加快搜索的速度。减少搜索算法对输入数据的依赖性。
4. 在其它条件相当的前提下，我们让取值最少的元素或约束力大的元素优先搜索，可以较快得到答案。
5. 充分利用Hash表，记忆式搜索，消除状态空间冗余。
6. 适当表示所求问题，利用之前的搜索结果构造上界函数。

3.2　动态规划

1　四边形不等式

设$w(i,j) \in \mathbb{R}, 1 \le i < j \le n$。

$$m[i,j] = \begin{cases} 0 & i = j \\ w(i,j) + \min\limits_{i<k\le j}\{m[i,k-1] + m[k,j]\} & i < j \end{cases} \tag{3.2.1}$$

当函数$w(i,j)$满足 $w(i,j) + w(i',j') \le w(i',j) + w(i,j'),\ i \le i' < j \le j'$ 时，称w满足**四边形不等式**。

当函数$w(i,j)$满足 $w(i',j) \le w(i,j'),\ i \le i' < j \le j'$ 时，称w**关于关于区间包含关系单调**。

对于满足四边形不等式的单调函数w，可推知由递归式定义的函数$m(i,j)$也满足四边形不等式。

$$m(i,j) + m(i',j') \le m(i',j) + m(i,j'),\ i \le i' < j \le j'$$

定义$s(i,j)=\max\{k|m(i,j)=m(i,k-1)+m(k,j)+w(i,j)\}$，由函数$m(i,j)$的四边形不等式性质可推出函数$s(i,j)$的单调性，即

$$s(i,j)\le s(i,j+1)\le s(i+1,j+1),\ i\le j$$

因此，当w是满足四边形不等式的单调函数时，函数$s(i,j)$单调，从而

$$\min_{i<k\le j}\{m(i,k-1)+m(k,j)\}=\min_{s(i,j-1)\le k\le s(i+1,j)}\{m(i,k-1)+m(k,j)\}\tag{3.2.2}$$

```
memset(m,0,sizeof(m)); memset(s,0,sizeof(s));
for (r=1;r<n;r++)
  for (i=1;i<=n-r;i++) {
    j=i+r;
    i1=s[i][j-1]>i?s[i][j-1]:i;
    j1=s[i+1][j]>i?s[i+1][j]:j-1;
    m[i][j]=m[i][i1]+m[i1+1][j];
    s[i][j]=i1;
    for (k=i1+1;k<=j1;k++) {
      t=m[i][k]+m[k+1][j];
      if (t<=m[i][j]) { m[i][j]=t; s[i][j]=k; }
    }
    m[i][j]+=w[i][j];
  }
```

【复杂度分析】

$$\begin{aligned}
& O\left(\sum_{r=0}^{n-1}\sum_{i=1}^{n-r}(1+s(i+1,i+r)-s(i,i+r-1))\right)\\
= \ & O\left(\sum_{r=0}^{n-1}(n-r+s(n-r,n)-s(1,r))\right)\\
= \ & O\left(\sum_{r=0}^{n-1}n\right)\\
= \ & O(n^2)
\end{aligned}$$

2 一些典型的DP

【DP求最长公共子序列】复杂度O(mn)
[输入]2序列,及其长度
[输出]最长公共子序列长度,重载返回一个最大匹配
[注意]做字符串匹配是串末的'\0'没有置!
[说明] 可更改元素类型,更换匹配函数和匹配价值函数

```
#define max(a,b) ((a)>(b)?(a):(b))
#define _match(a,b) ((a)==(b))
#define _value(a,b) 1
typedef char elem_t;

int LCS(int m,elem_t* a,int n,elem_t* b){
```

```
    int match[MAXN+1][MAXN+1],i,j;
    memset(match,0,sizeof(match));
    for (i=0;i<m;i++)
        for (j=0;j<n;j++)
            match[i+1][j+1]=max(max(match[i][j+1],match[i+1][j]),
                (match[i][j]+_value(a[i],b[i]))*_match(a[i],b[j]));
    return match[m][n];
}
int LCS(int m,elem_t* a,int n,elem_t* b,elem_t* ret){
    int match[MAXN+1][MAXN+1],last[MAXN+1][MAXN+1],i,j,t;
    memset(match,0,sizeof(match));
    for (i=0;i<m;i++)
        for (j=0;j<n;j++){
            match[i+1][j+1]=(match[i][j+1]>match[i+1][j]?match[i][j+1]:match[i+1][j]);
            last[i+1][j+1]=(match[i][j+1]>match[i+1][j]?3:1);
            if ((t=(match[i][j]+_value(a[i],b[i]))*_match(a[i],b[j]))>match[i+1][j+1])
                match[i+1][j+1]=t,last[i+1][j+1]=2;
        }
    for (;match[i][j];i-=(last[t=i][j]>1),j-=(last[t][j]<3))
        ret[match[i][j]-1]=(last[i][j]<3?a[i-1]:b[j-1]);
    return match[m][n];
}
```

【最长公共递增子序列】时间复杂度$O(n^2)$，空间$O(n^2)$
[输入]2个序列及其长度
[输出]公共递增子序列长度，最长公共递增子序列在ans中
[说明] 可以在调用此函数前先去除调只在一边出现的数

```
typedef int elem_t;
int GCIS(int m, elem_t a[], int n, elem_t b[], elem_t ans[])
{
    int f[MAXN + 1][MAXN + 1];
    int DP[MAXN + 1];
    int i, j, k, max;
    memset(f, 0, sizeof (f));
    memset(DP, 0, sizeof (DP));
    for (i = 1; i <= m; i++) {
        memcpy(f[i], f[i - 1], sizeof (f[0]));
        for (k = 0, j = 1; j <= n; j++) {
            if (b[j - 1] < a[i - 1] && DP[j] > DP[k]) k = j;
            if (b[j - 1] == a[i - 1] && DP[k] + 1 > DP[j])
                DP[j] = DP[k] + 1, f[i][j] = i * (n + 1) + k;
        }
    }
    for (max = 0, i = 1; i <= n; i++)
        if (DP[i] > DP[max]) max = i;
    for (i = m * n + m + max, j = DP[max]; j; i = f[i / (n + 1)][i % (n + 1)], j--)
        ans[j - 1] = b[i % (n + 1) - 1];
    return DP[max];
}
```

【LIS 最长上升序列】二分查找$O(n \log n)$
函数LIS(int a[], int n)返回a的最长上升序列长度。若要求最长非降序列的最少覆盖，根

据Dilworth定理，只要求最长上升序列的长度即可。如果覆盖不满足偏序关系，则只能转化为最小路径覆盖。

```
int LIS(int a[], int n){
  int b[N];
  b[1] = a[0];
  int len = 1;
  for (int i = 1; i < n; ++i) {
    int k = lower_bound(b+1, b+len+1, a[i]) - b;  //改成upper_bound则为最长非降子序列
    if (k > len) len++;
    b[k]=a[i];
  }
  return len;
}
```

【最长单调子序列】复杂度$O(nlogn)$
[注意]最小序列覆盖和最长序列的对应关系,例如"define $\_cp(a,b)((a) > (b))$"求解最长严格递减序列, 则"define $\_cp(a,b)(!((a) > (b)))$"求解最小严格递减序列覆盖
[说明] 可更改元素类型和比较函数

```
#define MAXN 10000
#define _cp(a,b) (!((a)>(b)))
typedef int elem_t;

int subseq(int n,elem_t* a){
    int b[MAXN],i,l,r,m,ret=0;
    for (i=0;i<n;b[l]=i++,ret+=(l>ret))
        for (m=((l=1)+(r=ret))>>1;l<=r;m=(l+r)>>1)
            if (_cp(a[b[m]],a[i])) l=m+1;
            else r=m-1;
    return ret;
}

int subseq(int n,elem_t* a,elem_t* ans){
    int b[MAXN],p[MAXN],i,l,r,m,ret=0;
    for (i=0;i<n;p[b[l]=i++]=b[l-1],ret+=(l>ret))
        for (m=((l=1)+(r=ret))>>1;l<=r;m=(l+r)>>1)
            if (_cp(a[b[m]],a[i])) l=m+1;
            else r=m-1;
    for (m=b[i=ret];i;ans[--i]=a[m],m=p[m]);
    return ret;
}
```

【最大子段和】复杂度$O(n)$
[输入]串长n, 内容list[]
[输出]最大子段和
[说明] 重载返回子段位置(maxsum=list[start]+...+list[end])

```
typedef int elem_t;

elem_t maxsum(int n,elem_t* list){
    elem_t ret,sum=0;
    int i;
```

```
    for (ret=list[i=0];i<n;i++)
        sum=(sum>0?sum:0)+list[i],ret=(sum>ret?sum:ret);
    return ret;
}

elem_t maxsum(int n,elem_t* list,int& start,int& end){
    elem_t ret,sum=0;
    int s,i;
    for (ret=list[start=end=s=i=0];i<n;i++,s=(sum>0?s:i))
        if ((sum=(sum>0?sum:0)+list[i])>ret)
            ret=sum,start=s,end=i;
    return ret;
}
```

【最大子阵和】复杂度$O(n^3)$
[输入]阵的大小m,n和内容mat[][]
[输出]最大子阵和
[说明] 重载返回子阵位置(maxsum=list[s1][s2]+...+list[e1][e2])

```
#define MAXN 100
typedef int elem_t;

elem_t maxsum(int m,int n,elem_t mat[][MAXN]){
    elem_t matsum[MAXN][MAXN+1],ret,sum;
    int i,j,k;
    for (i=0;i<m;i++)
        for (matsum[i][j=0]=0;j<n;j++)
            matsum[i][j+1]=matsum[i][j]+mat[i][j];
    for (ret=mat[0][j=0];j<n;j++)
        for (k=j;k<n;k++)
            for (sum=0,i=0;i<m;i++)
                sum=(sum>0?sum:0)+matsum[i][k+1]-matsum[i][j],ret=(sum>ret?sum:ret);
    return ret;
}

elem_t maxsum(int m,int n,elem_t mat[][MAXN],int& s1,int& s2,int& e1,int& e2){
    elem_t matsum[MAXN][MAXN+1],ret,sum;
    int i,j,k,s;
    for (i=0;i<m;i++)
        for (matsum[i][j=0]=0;j<n;j++)
            matsum[i][j+1]=matsum[i][j]+mat[i][j];
    for (ret=mat[s1=e1=0][s2=e2=j=0];j<n;j++)
        for (k=j;k<n;k++)
            for (sum=0,s=i=0;i<m;i++,s=(sum>0?s:i))
                if ((sum=(sum>0?sum:0)+matsum[i][k+1]-matsum[i][j])>ret)
                    ret=sum,s1=s,s2=i,e1=j,e2=k;
    return ret;
}
```

3.3 排列与阶乘

【排列转化为阶乘进制】

```
int perm2num(int n,int *p){
    int i,j,ret=0,k=1;
    for(i=n-2;i>=0;k*=n-(i--))
        for(j=i+1;j<n;j++)
            if(p[j]<p[i])ret+=k;
    return ret;
}
```

【阶乘进制转化为排列】

```
void num2perm(int n,int *p,int t){
    int i,j;
    for(i=n-1;i>=0;i--)
        p[i]=t%(n-i),t/=n-i;
    for(i=n-1;i;i--)
        for(j=i-1;j>=0;j--)
            if(p[j]<=p[i])p[i]++;
}
```

【字典序组合与序号的转换】

```
//comb为组合数C(n,m),必要时换成大数,注意处理C(n,m)=0|n<m
int comb(int n,int m){
    int ret=1,i;
    m=m<(n-m)?m:(n-m);
    for (i=n-m+1;i<=n;ret*=(i++));
    for (i=1;i<=m;ret/=(i++));
    return m<0?0:ret;
}
int comb2num(int n,int m,int *c){
    int ret=comb(n,m),i;
    for (i=0;i<m;i++)
        ret-=comb(n-c[i],m-i);
    return ret;
}
void num2comb(int n,int m,int* c,int t){
    int i,j=1,k;
    for (i=0;i<m;c[i++]=j++)
        for (;t>(k=comb(n-j,m-i-1));t-=k,j++);
}
```

【生成排列和组合】生成全排列一般直接使用STL用的$next_permutation()$

```
typedef long long llong;
void create_permutation(int n)  \\全排列
{
    llong total = 1;
    int i, j, k, t, *a = new int [n], top;
    for(i = 1;i <= n;++ i) a[i] = i, total *= i;
    for(i = 1;i < n;++ i) printf("%d␣", a[i]);
    printf("%d\n", a[n]);
```

```
    for(i = 1;i < total;++ i) {
        for(j = n;a[j] < a[j-1];-- j);
        for(k = n;a[j-1] > a[k];-- k);
        t = a[j-1]; a[j-1] = a[k]; a[k] = t;
        top = (j + n - 1) >> 1;
        for(k = j;k <= top;++ k) {
            t = a[k]; a[k] = a[n - k + j]; a[n - k + j] = t;
        }
        for(j = 1;j < n;++ j) printf("%d␣", a[j]);
        printf("%d\n", a[n]);
    }
}

void create_fab(int m,int n) //全组合
{
     int i, j, *a = new int[n + 2];
     for(i = 1;i <= n;++ i)     a[i] = i;
     a[n + 1] = m + 1;
     for(j = 1;j < n;++ j)     printf("%d␣",a[j]);
     printf("%d\n",a[n]);
    while(a[1] < m - n + 1)
        for(i = n;i > 0;-- i)
            if(a[i] < a[i + 1] - 1) {
                ++ a[i];
                for(j = i;j < n;++ j) a[j + 1] = a[j] + 1;
                for(j = 1;j < n;++ j) printf("%d␣", a[j]);
                printf("%d\n", a[n]);
                break;
            }
}
```

【阶乘最后非零位】复杂度$O(nlogn)$
[说明] 返回该位,n以字符串方式传入

```
#define MAXN 10000

int lastdigit(char* buf){
    const int mod[20]={1,1,2,6,4,2,2,4,2,8,4,4,8,4,6,8,8,6,8,2};
    int len=strlen(buf),a[MAXN],i,c,ret=1;
    if (len==1) return mod[buf[0]-'0'];
    for (i=0;i<len;i++) a[i]=buf[len-1-i]-'0';
    for (;len;len-=!a[len-1]){
        ret=ret*mod[a[1]%2*10+a[0]]%5;
        for (c=0,i=len-1;i>=0;i--)
            c=c*10+a[i],a[i]=c/5,c%=5;
    }
    return ret+ret%2*5;
}
```

3.4 格雷码

```
//生成格雷码
void Gray(int n)
{
    unsigned long g = 0;
    unsigned long max = (unsigned long) (1L) << n;
    do {
        cout << g << "␣";
        g ^= 1;
        cout << g << "␣";
        g ^= (g ^(g - 1)) + 1;
    } while (g < max);
}
int DecimaltoGray(int x) { return x ^ (x >> 1); } //十进制数到格雷码的转换
int GraytoDecimal(int x) { int y; for(y = x;x >>= 1;y ^= x); return y; }//格雷码到十进制数转换
```

3.5 归并排序求逆序数

最后的结果中数组a是从小到大排序的

```
//注意：逆序数很容易超出 int ，必要时用 long long
int temp[N<<1];
long long msort(int a[], int l, int r)
{
    if(l == r) return 0;
    int i, j, k, m = (l + r) >> 1;
    long long t = msort(a, l, m) + msort(a, m + 1, r);
    for(i = l, j = m + 1, k = l;k <= r;++ k)
        if(i <= m && a[i] <= a[j] || j > r) temp[k] = a[i++];
        else temp[k] = a[j++], t += m - i + 1;
    memcpy(a + l, temp + l, (r - l + 1) * sizeof(a[0]));
    return t;
}
```

3.6 日期计算

连分数展开的近似算法，有效范围：公元前4713年1月1日 - 100000年1月1日。也可把转换后的数值加(减)一定天数后转换回来. a.y=0是公元前1年,a.y=-1是公元前2年。

```
int days[12]={31,28,31,30,31,30,31,31,30,31,30,31};
struct TDate { int y, m, d; };


//判闰年
inline int leap(int year) {
    return (year % 4 == 0 && year % 100 != 0) || year % 400 == 0;
}

//判合法性
```

```
inline int legal(TDate a) {
    if (a.m < 0 || a.m > 12)return 0;
    if (a.m == 2)return a.d > 0 && a.d <= 28 + leap(a.y);
    return a.d > 0 && a.d <= ds[a.m - 1];
}

 //将格林历转成儒略历
int G2J(TDate a)
{
    int jd;
    jd=(1461*(a.y+4800+(a.m-14)/12))/4+(367*(a.m-2-12*((a.m-14)/12)))/12-
        (3*((a.y+4900+(a.m-14)/12)/100 ))/4+a.d-32075;
    return jd;
}

//将儒略历转成格林历
TDate J2G(int jd)
{
    TDate a;
    int t,m,i,j,d,n,y;
    t=jd+68569;
    n=(4*t)/146097;
    t=t-(146097*n+3)/4;
    i=(4000*(t+1))/1461001;
    t=t-(1461*i)/4+31;
    j=(80*t)/2447;
    a.d=t-(2447*j)/80;
    t=j/11;
    a.m=j+2-(12*t);
    a.y=100*(n-49)+i+t;
    return a;
}

//返回指定日期是星期几
int weekday(TDate a) {
    int tm = a.m >= 3 ? (a.m - 2) : (a.m + 10);
    int ty = a.m >= 3 ? a.y : (a.y - 1);
    return (ty + ty / 4 - ty / 100 + ty / 400 + (int) (2.6 * tm - 0.2) + a.d) % 7;
}
```

3.7 KMP算法

```
// 前缀函数,与多数教科书上略有不同这里是从0开始不是1开始
void prefix(char s[], int * pre)
{
    pre[0] = 0;
    for(int i = 1, k = 0;s[i];++ i) {
        while(k > 0 && s[k] != s[i]) k = pre[k-1];
        if(s[k] == s[i]) ++ k;
        pre[i] = k;
    }
```

```
}

//x主串，y模式串
int KMP(char *x, char *y)
{
    int m = strlen(y), j = 0, i;
    int *pre = new  int [m];  //  如有必要静态建立
    prefix(y, pre);  // 这里避免重复求统一串前缀
    for(i = 0;x[i];++ i) {
        while(j > 0 && y[j]  != x[i]) j = pre[j - 1];
        if(y[j] == x[i]) ++ j;
        if(!y[j]) {
            delete [] pre;
            return i - m  + 1;
            //j = pre[j-1]; // 不马上返回可以继续寻找下一个匹配
        }
    }
    delete [] pre;
    return -1;
}
```

3.8 字符串最小表示法

返回字符串最小表示的首字母位置,在@(0...len-1)@之间,修改char * 和strlen 还可以用于整数，结构体等

```
int MinString(char *str)
{
    int i, j, k = 0, len = strlen(str);
    char * s = new char [len << 1]; //建议先静态开好
    for(i = 0;i < len;++ i) s[i] = s[i+len] = str[i];
    for(i = 0, j = 1;k < len && i < len && j < len;) {
        for(k = 0;k < len && s[i+k] == s[j+k];++ k);
        if(k < len) {
            if(s[i+k] > s[j+k]) i += k + 1;
            else j += k + 1;
            j += (i == j);
        }
    }
    return min(i, j);
}

//输入2 个字符串，判断它们之间是否是环串，如果是则返回1 ，否则返回0
int CircleString(char *str, char * s, int len) //为字符串长度len
{
    int i, j, k;
    char *st = (char *) malloc(len << 1); //建议先静态开好
    char *ss = (char *) malloc(len << 1);
    for (i = 0; i < len; ++i)
        st[i] = st[i + len] = str[i];
    for (i = 0; i < len; ++i)
```

```
        ss[i] = ss[i + len] = s[i];
    for (i = 0, j = 0, k = 0; k < len && i < len && j < len;) {
        for (k = 0; k < len && st[i + k] == ss[j + k]; ++k);
        if (k < len)
            if (st[i + k] > ss[j + k]) i += k + 1;
            else j += k + 1;
    }
    return k == len; //return max(i, j) < len; 这个一样的
}
```

3.9 求第k小元素的O(n)算法

find(x, 0, n, k);找数组x[0..n - 1]中的第k小元素

```
int partition(int x[], int l , int r, int t, int &m) {
  int  i = l - 1, j = r, p = l, q = r - 1, k;
  while (true) {
    while(x[++i] < t);
    while(x[--j] > t);
    if (i >= j) break;
    std::swap(x[i], x[j]);
    if (x[j] == t) std::swap(x[q--], x[j]);
    if (x[i] == t) std::swap(x[p++], x[i]);
  }
  for (k = l; k < p; --j, ++k)    std::swap(x[k], x[j]);
  for (k = r - 1; k > q; ++i, --k)    std::swap(x[k], x[i]);
  m = i;
  return j;
}

int findk(int x[], int l, int r, int k) {
  if (r - l <= 75) {
     std::sort(x + l, x + r);
     return x[l + k - 1];
  }
  int i, j, t, p, q;
  for (i = 0; i <= (r - l - 5) / 5; ++i) {
    std::sort(x + l + 5 * i, x + l + 5 * (i + 1));
    std::swap(x[l + i], x[l + 5 * i + 2]);
  }
  t = findk(x, l, l + (r - l - 5) / 5, (r - l - 5) / 10);
  i = partition(x, l, r, t, q);
  p = q - l;
  if (k <= i - l + 1) return findk(x, l , i, k);
  if (k > p)  return findk(x, q, r, k - p);
  return t;
}
```

3.10 N皇后的一组构造解

NQuees(n, a);将n皇后的一组构造解存放在数组a中

```
void NQueens(int n, int *a)
{
  int i = 0, j, k, m;
  if (n%6 < 2 || n%6 > 3) {
    for (j = 2; j <= n; j+=2) a[i++] = j;
    for (j = 1; j <= n; j+=2) a[i++] = j;
  } else {
    k = n>>1;
    m = n - ((k&1) && (n&1));
    for (j = k; j <= m; j+=2) a[i++] = j;
    for (j = 2 - (k&1); j <= k - 2; j+=2) a[i++] = j;
    m = n - (n & 1);
    for (j = k+3; j <= m; j+=2) a[i++] = j;
    for (j = 1 + (k&1); j <= k+1; j+=2) a[i++] = j;
    if (n & 1) a[n-1] = n;
  }
}
```

第四章 计算几何

4.1 基础知识、点，向量

【基本定义】

```
#define PI 3.14159265358979323846264338327795
const double PI = acos(-1.00);
const double EPS = 1e-6;
```

【符号函数】为避免误差判断浮点数的符号,eps可取一个很小的数如1e-6

```
int sign{double d)
{
    return d < -EPS ? -1 : (d > EPS ? 1 : 0);
}
int sign(double d)
{
    if(fabs(d) < EPS) return 0;
    return (d > 0) ? 1 : -1;
}

bool equal(double a, double b)
{ return sign(a - b) == 0; }
```

【类型定义】

```
struct TPoint {
    double x, y;
    TPoint(){}
    TPoint(const double &_x, const double &_y):x(_x), y(_y) {}
};
struct TPoint { int x, y; };
TPoint(double u, double v) { x = u, y = v; }
bool operator==(const TPoint & a, const TPoint & b)
    { return (fabs(a.x-b.x)<EPS &&fabs(a.y-b.y)<EPS); }
bool operator!=(const TPoint & a, const TPoint & b)
    { return (fabs(a.x-b.x)>EPS||fabs(a.y-b.y)>EPS);}
TPoint operator-(const TPoint & a, const TPoint b)
{ return TPoint(a.x - b.x, a.y - b.y); }
 TPoint operator+(const TPoint & a, const TPoint & b)
{ return TPoint(a.x + b.x, a.y + b.y); }
```

【向量叉乘】$AB \times AC = (p_1 - p_0) \times (p_2 - p_0)$.

```
double cross(const TPoint & a, const TPoint & b)
{ return (a.x * b.y - a.y * b.x); }
double cross(const TPoint & a, const TPoint & b, const TPoint & c)
{ return (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x); }
```

1、叉乘的绝对值又为向量AB,AC构成的平行四边形的面积.

2、$\boldsymbol{u} \times \boldsymbol{v} = \begin{vmatrix} u_x & u_y \\ v_x & v_y \end{vmatrix}$，其大小等于$|AB||AC|sin\alpha$，方向根据右手定则求出。

- $AB \times AC > 0$，则$< p_0, p_1 >$在$< p_0, p_2 >$顺时针方向
- $AB \times AC = 0$，则$< p_0, p_1 >, < p_0, p_2 >$共线
  - 若(p_1.x-p_0.x)*(p_2.x-p_0.x)>0 或(p_1.y-p_0.y)*(p_2.y-p_0.y)>0，则AB, AC同向
  - 若(p_1.x-p_0.x)*(p_2.x-p_0.x)<0 或(p_1.y-p_0.y)*(p_2.y-p_0.y)<0，则AB, AC反向
- $AB \times AC < 0$，则$< p_0, p_1 >$在$< p_0, p_2 >$逆时针方向

【向量点乘】

```
double dot(TPoint & a, TPoint & b)
{ return a.x * b.x + a.y * b.y; }
double dot(TPoint & a, TPoint & b, TPoint & c)
{ return (b.x - a.x) * (c.x - a.x) + (b.y - a.y) * (c.y - a.y); }
```

$AB \bullet AC = |AB||AC|cos\alpha = (x_B - x_A)(x_C - x_A) + (y_B - y_A)(y_C - y_A)$

— $AB \bullet AC < 0$，则AB, AC夹角大于90^o

— $AB \bullet AC = 0$，则$AB \perp AC$

— $AB \bullet AC > 0$，则AB, AC夹角小于90^o

【向量长度】注意:如果TPoint 的点为整数,小心溢出.如果有必要，可以考虑把sqrt放到外面

```
double len(const TPoint & a)
{ return sqrt((double)(a.x*a.x+a.y*a.y)); }
```

【距离】

```
double dis(const TPoint & a, const TPoint & b)
{ return len(b-a); }
```

【向量夹角计算】[输出]向量a按逆时针方向,旋转到向量b的角度.角度小于pi，返回正值;角度大于pi，返回负值

```
double angle(const TPoint &a, const TPoint &b)
{
    double ret = acos(dot(a, b) / (dis(a, b) * dis(a, b)));
    if(cross(a, b) < 0) return ret;
    return -ret;
}

//返回顶角在o点,起始边为os,终止边为oe的夹角(单位:弧度),规定逆时针为正方向
```

```
//可以用于求线段之间的夹角
double angle(const TPoint &o, const TPoint &s, const TPoint &e)
{   return angle(s - o, e - o); }
```

【向量的旋转】

坐标或向量向量(x,y)关于原点的逆时针旋转α后的坐标为(x', y'),

公式为 $\begin{cases} x' = x*\cos\alpha - y*\sin\alpha \\ y' = x*\sin\alpha + y*\cos\alpha \end{cases}$,变换矩阵为 $\begin{bmatrix} x' & y' \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix}$

```
//返回点p以点o为中心旋转alpha后的坐标
TPoint rotate(TPoint & o, double alpha, TPoint  p)
{
    TPoint tmp;
    p.x -= o.x; p.y -= o.y;
    tmp.x = p.x * cos(alpha) - p.y * sin(alpha) + o.x;
    tmp.y = p.y * cos(alpha) + p.x * sin(alpha) + o.y;
    return tmp;
}
//返回向量p旋转alpha后的坐标
TPoint rotate(double alpha, TPoint &p)
{
    TPoint tmp;
    tmp.x = p.x * cos(alpha) - p.y * sin(alpha);
    tmp.y = p.y * cos(alpha) + p.x * sin(alpha);
    return tmp;
}
```

4.2 线段、直线、曲线

【类型定义】

```
struct TLine { double a, b, c; };  //直线
struct TSegment { TPoint s, e; };  //线段，也可看作直线点式2
```

【过 $(x_1,y_1),(x_2,y_2)$ 直线方程】

$Ax+By+C=0$，其中 $A=y_2-y_1, B=x_1-x_2, C=x_2y_1-x_1y_2$

[输入]两点坐标

[输出]直线一般式方程

[说明] $ax+by+c=0a >= 0$,这个方程的前提是每个点都有意义,而非是表示直线，射线的点，而且要保证两点不是相同的点。亦可用于线段产生直线方程

```
TLine get_line(TPoint & p1,TPoint & p2)
{
    TLine tl;
    tl.a=p2.y-p1.y;
    tl.b=p1.x-p2.x;
    tl.c=p1.y*p2.x-p1.x*p2.y;
    if (tl.a<0) tl.a*=-1.0,tl.b*=-1.0,tl.c*=-1.0;
    return tl;
}
```

【根据直线上的两点，返回直线的倾斜角】

[输出]返回值范围为[0,2*PI),0表示与x轴方向相同,注意返回的方向是$p1- > p2$的,也就是向量p1p2的倾斜角

```
double slope_angle(TPoint & p1, TPoint & p2)
{
    double ang, tmp;
    TPoint p;
    p.x = p2.x - p1.x;
    p.y = p2.y - p1.y;
    if (fabs(p.x) < EPS) {
        if (p.y > 0) ang = PI * 0.5;
        else ang = 3.0 * PI * 0.5;
    } else {
        ang = atan(p.y / p.x);
        if (p.x < 0) ang += PI;
    }
    if (ang < 0) ang += 2.0 * PI;
    return ang;
}
```

【判断两点是否在一线段同侧】

```
//返回<0(-1)表示异侧，0表示至少有一点在ab上，>0(1)表示同侧
int same_side(TPoint & a, TPoint & b, TPoint & c, TPoint & d)
{ return (sign(cross(a, b, c)) * sign(cross(a, b, d))); }
//返回true 同侧, false异侧
bool same_side(TPoint & p1, TPoint & p2, TLine line)
{
    return (line.a * p1.x + line.b * p1.y + line.c) *
        (line.a * p2.x + line.b * p2.y + line.c) > 0;
}
```

【点与线段的关系】

$r = \frac{AC \bullet AB}{|AB|^2}$, r has the following meaning:

r = 0 P = A

r = 1 P = B

r < 0 P is on the backward extension of AB

r > 1 P is on the forward extension of AB

0 < r < 1 P is interior to AB

```
double relation(const TPoint &p, const TSegment &seg)
{   return dot(seg.s, p, seg.e) / (dis(seg.s, seg.e) * dis(seg.s, seg.e)); }
```

【点在直线上的垂足】

```
TPoint perp_seg_pnt(const TPoint &p, const TSegment &seg)
{
    double r = relation(p, seg);
    TPoint ret;
    ret.x = seg.s.x + r * (seg.e.x - seg.s.x);
    ret.y = seg.s.y + r * (seg.e.y - seg.s.y);
    return ret;
}
```

```
TPoint perp_line_pnt(TPoint & p, TLine l)
{
    TPoint q;
    double L = l.a * l.a + l.b * l.b;
    q.x = (l.b * (l.b * p.x - l.a * p.y) - l.a * l.c) / L;
    q.y = (l.a * (l.a * p.y - l.b * p.x) - l.b * l.c) / L;
    return q;
}
```

【点到直线上的距离】

点P到直线AB的距离计算向量公式: $\frac{|(P-A)\times(B-A)|}{|B-A|}$

```
double dis_p2_line(TPoint & p, TLine l)
{ return fabs(l.a * p.x + l.b * p.y +l. c) / sqrt(l.a * l.a + l.b * l.b); }
inline double dis_p2_line(TPoint & a,TPoint & b,TPoint & p)
{ return fabs(cross(a,p,b))/dis(a,b); }
```

【点到线段的距离】

```
//线段ab，点c
double dis_p2_seg(TPoint & c, TPoint & a, TPoint & b) {
    double x, y, z;
    x = (a.x - c.x) * (a.x - c.x) + (a.y - c.y) * (a.y - c.y);
    y = (b.x - c.x) * (b.x - c.x) + (b.y - c.y) * (b.y - c.y);
    z = (a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y);
    if (x + z - y < 0 || y + z - x < 0) return sqrt(x < y ? x : y);
    return fabs(cross(c, a, b) / sqrt(z));
}

double dis_p2_seg(TPoint & c,TPoint & a, TPoint & b)
{
    TPoint ac, ab, interp;
    double f, d;
    ac.x = c.x - a.x, ac.y = c.y - a.y;
    ab.x = b.x - a.x, ab.y = b.y - a.y;
    if ((f = dot(ac, ab)) < 0) return len(ac);
    d = ab.x * ab.x + ab.y * ab.y;
    if (f > d) return dis(c, b);
    interp.x = a.x + ab.x * f / d;
    interp.y = a.y + ab.y * f / d;
    return dis(interp, c);
}

// 求点p到线段seg的最短距离,并返回线段上距该点最近的点ret
// 注意:  ret是线段l上到点p最近的点,不一定是垂足
double dis_p2_seg(const TPoint &p, const TSegment &seg, TPoint &ret)
{
    double r = relation(p, seg);
    if (r < 0) ret = seg.s;
    else if (r > 1) ret = seg.e;
    else ret = perp_seg_pnt(p, seg);
    return dis(p, ret);
```

```
}
```

【点关于直线的对称点】

[输入]点p的坐标，及直线l(一般式表示)
[返回]点p关于直线l的对称点

```
TPoint symmetry(TPoint & pt, TLine l)
{
    TPoint q, p = perp_line_pnt(pt, l);
    q.x = p.x * 2.0 - pt.x; q.y = p.y *2.0 - pt.y;
    return q;
}
TPoint symmetry(TPoint p, TSegment seg) {
    TPoint ret;
    double a = seg.e.x - seg.s.x;
    double b = seg.e.y - seg.s.y;
    double t = ((p.x - seg.s.x) * a + (p.y - seg.s.y) * b) / (a * a + b * b);
    ret.x = 2 * seg.s.x + 2 * a * t - p.x;
    ret.y = 2 * seg.s.y + 2 * b * t - p.y;
    return ret;
}
```

【求线段垂直平分线】

```
TSegment prep_mid_seg(TPoint a, TPoint b)
{
    TSegment tmp;
    tmp.s.x = (a.x + b.x) * 0.5;
    tmp.s.y = (a.y + b.y) * 0.5;
    tmp.e.x = tmp.s.x - (a.y - b.y);
    tmp.e.y = tmp.s.y + (a.x - b.x);
    return tmp;
}
```

【判断点是否在直线上(包括端点)】

[说明] 也可以通过计算点到直线的距离是否为0来判定

```
bool on_line(TPoint & p, TPoint & a, TPoint & b)
{ return sign(cross(c, a, b)) == 0; }
```

【判断点是否在线段上(包括端点)】

程序一：
[输入]点c坐标，线段ab坐标
[输出]返回true or false
[说明]1.c是否在直线ab上,2.c是否在以ab为对角线的矩形中

```
bool on_segment(TPoint & c, TPoint & a, TPoint & b)
{
    return (fabs(cross(c, a, b)) < EPS) &&
           (c.x - a.x) * (c.x - b.x) < EPS &&
           (c.y - a.y) * (c.y - b.y) < EPS;
}
```

程序二：
[输入]点c坐标，线段ab坐标
[输出]-1表示c在线段ab中(不在端点), 0表示在线段ab端点(点a或点b), 1表示点c不在线段ab上(在ab延长线上), 2表示点c不在线段ab所在直线上。
[说明]betweenCmp还可以直接用点乘dot函数代替，会更加简洁，对整数点很适用，但对于浮点数会涉及到浮点运算，误差更大。其调用方式为dot(c, a, b);

```
int xycmp(double p, double mint, double maxt)  //坐标比较后相乘
{ return sign(p - mint) * sign(p - maxt); }

//判断点c 是否在a、b范围内
int betweenCmp(TPoint & c, TPoint & a, TPoint & b)
{
    if(fabs(a.x - b.x) > fabs(a.y - b.y))
        return xycmp(c.x, min(a.x, b.x), max(a.x, b.x));
    return xycmp(c.y, min(a.y, b.y), max(a.y, b.y));
}

//判断点c 是否在线段ab上
int on_segment(TPoint & c, TPoint & a, TPoint & b)
{
    if(sign(cross(c, a, b)) != 0) return 2;
    return betweenCmp(c, a, b);
}
```

【判断线段是否相交、求交点】
程序一：
[输入]线段ab 两端点的坐标线段cd 两端点坐标, 跨立试验
[输出]返回true or false

```
bool seg_inter(TPoint & a, TPoint & b, TPoint & c, TPoint & d)
{
    return
    //排斥试验
    max(a.x, b.x) >= min(c.x, d.x) && max(c.x, d.x) >= min(a.x, b.x) &&
    max(a.y, b.y) >= min(c.y, d.y) && max(c.y, d.y) >= min(a.y, b.y) &&
    //跨立试验
    cross(a, b, c) * cross(a, b, d) <= EPS &&
    cross(c, d, a) * cross(c, d, b) <= EPS;
}
```

程序二:
[说明]已知两线段端点，求交点(此两线段必交前提下使用)

```
TPoint seg_inter_pnt(TPoint & a, TPoint & b, TPoint & c, TPoint & d)
{
    double s1 = cross(a, b, c),  s2 = cross(a, b, d);
    TPoint tmp;
    tmp.x = (c.x * s2 - d.x * s1) / (s2 - s1);
    tmp.y = (c.y * s2 - d.y * s1) / (s2 - s1);
    return tmp;
}
```

程序三:
[输入]线段ab 两端点的坐标线段cd 两端点坐标，交点坐标存放变量e
[输出]返回0表示不相交，1表示不规范相交，2表示规范相交，交点保存在e中。

```
//判断线段 ab 和 cd 是否相交
int seg_inter(TPoint & a, TPoint & b, TPoint & c, TPoint & d, TPoint &e)
{
    double s1, s2, s3, s4;
    int d1, d2, d3, d4;
    d1 = sign(s1 = cross(a, b, c));
    d2 = sign(s2 = cross(a, b, d));
    d3 = sign(s3 = cross(c, d, a));
    d4 = sign(s4 = cross(c, d, b));

    //规范相交
    if((d1 ^ d2) == -2 && (d3 ^ d4) == -2) {
        e.x = (c.x * s2 - d.x * s1) / (s2 - s1);
        e.y = (c.y * s2 - d.y * s1) / (s2 - s1);
        return 2;
    }

    //非规范相交
    if((d1 == 0 && betweenCmp(c, a, b) <= 0)  ||
        (d2 == 0 && betweenCmp(d, a, b) <= 0) ||
        (d3 == 0 && betweenCmp(a, c, d) <= 0) ||
        (d4 == 0 && betweenCmp(b, c, d) <= 0))
        return 1;
    return 0;
}
```

【求直线交点】
[输入]ab 直线与cd 直线方程
[输出]直线交点坐标

```
TPoint line_intersect(TLine l1, TLine l2)//求两直线得交点坐标
{
    TPoint tmp;
    if(l1.b == 0){
        tmp.x = -l1.c / (double)l1.a;
        tmp.y = (-l2.c - l2.a * tmp.x) / (double)l2.b;
    }
    else{
        tmp.x = (double)(l1.c * l2.b - l1.b * l2.c) /
            (double)(l1.b * l2.a - l2.b * l1.a);
        tmp.y = (double)(-l1.c - l1.a * tmp.x) /(double)l1.b;
    }
    return tmp;
}
```

【平行线直接的距离】平行线ab与cd直接的距离

```
double dis_parallel_line(TPoint & a, TPoint & b, TPoint & c, TPoint & d)
{
    return min(min(dis_p2_seg(c, a, b), dis_p2_seg(d, a, b)),
```

```
            min(dis_p2_seg(a, c, d), dis_p2_seg(b, c, d)));
}
```

【求距离线段offset的平行线】求离线段a的距离是$offset$的平行线,有2条(b,c)

```
void get_dis_parall(double offset, TSegment &a, TSegment &b, TSegment &c)
{
    TSegment prep = prep_mid_seg(a.s, a.e);
    TPoint vec_p = prep.e - prep.s, tempp;
    double temp = sqrt(1.0 / (vec_p.x * vec_p.x + vec_p.y * vec_p.y)) * offset;
    tempp.x = temp * vec_p.x;
    tempp.y = temp * vec_p.y;
    b.s = a.s + tempp;   b.e = a.e + tempp;
    c.s = a.s - tempp;   c.e = a.e - tempp;
}
```

【镜面反射线】已知入射线、镜面，求反射线。$a1,b1,c1$为镜面直线方程$(a1x+b1y+c1=0$,下同)系数; $a2,b2,c2$为入射光直线方程系数; a,b,c为反射光直线方程系数. 光是有方向的，使用时注意：入射光向量:$<-b2,a2>$ 反射光向量:$<b,-a>$. 不要忘记结果中可能会有"$negativezeros$"

```
void reflect(double a1,double b1,double c1,double a2,
    double b2,double c2,double &a,double &b,double &c)
{
    double n,m;
    double tpb,tpa;
    tpb=b1*b2+a1*a2;
    tpa=a2*b1-a1*b2;
    m=(tpb*b1+tpa*a1)/(b1*b1+a1*a1);
    n=(tpa*b1-tpb*a1)/(b1*b1+a1*a1);
    if(fabs(a1*b2-a2*b1)<1e-20)
    {
        a=a2;b=b2;c=c2;
        return;
    }
    double xx,yy; //(xx,yy)是入射线与镜面的交点。
    xx=(b1*c2-b2*c1)/(a1*b2-a2*b1);
    yy=(a2*c1-a1*c2)/(a1*b2-a2*b1);
    a=n;
    b=-m;
    c=m*yy-xx*n;
}
```

4.3 凸包与顶点对

【Graham-Scan算法】
[输入]输入点集p[n], 点数n
[输出]返回凸包点数, hull中为凸包逆时针顺序
[注意]特殊情况：$n=1,2;top=1,2$,凸包上是否会三点共线请仔细考虑

```
TPoint p[N];
```

```
bool graham_cmp(const TPoint &b, const TPoint &c)
{
    double tmp = cross(b, c, p[0]);
    if(tmp > EPS) return true;
    if(fabs(tmp) < EPS && (dis(b, p[0]) < dis(c, p[0]))) return true;
    return false;
}

int graham_scan(TPoint hull[], int n)
{
    int top, i, k = 0;
    for(i = 1;i < n;++ i)
        if((p[k].y-p[i].y>EPS)||(fabs(p[i].y-p[k].y)<EPS&&p[k].x-p[i].x>EPS ))
            k = i;
    swap(p[0], p[k]);
    sort(p+1, p + n, graham_cmp);
    hull[0] = p[0], hull[1] = p[1], hull[2] = p[2];
    if(n < 3) return n; else top = 3;
    for(i = 3;i < n;++ i) {
        while(top >= 2 && cross(hull[top-2], hull[top-1], p[i]) < EPS) -- top;
        hull[top++] = p[i];
    }
    return top;
}
```

【melkman算法】

[输入]点集p[n], 点数n

[输出]返回凸包点数, hull中为凸包逆时针顺序

[注意]特殊情况：$n = 1, 2; top = 1, 2$，凸包上是否会三点共线请仔细考虑

[说明]hull[]的长度至少应为n的2倍

```
//凸包p[hull[i]] 0<= i < ret (ret为返回值)
bool cmp(TPoint &a,TPoint &b)
{
   return a.y<b.y||(a.y==b.y&&a.x<b.x);
}
int melkman(int hull[], int n)
{
    sort(p, p + n,cmp);
    int top = n, bottom = n, i;
    for(i = 0;i < n;++ i) {
        hull[top] = hull[bottom] = i;
        while(top - n >= 2 && cross(p[hull[top-2]],
            p[hull[top-1]], p[hull[top]]) < 0) {
            hull[top-1] = hull[top]; -- top;
        }
        while(n - bottom >= 2 && cross(p[hull[bottom+2]],
         p[hull[bottom+1]], p[hull[bottom]]) > 0) {
            hull[bottom+1] = hull[bottom]; ++ bottom;
        }
        ++ top; -- bottom;
    }
    for(i = 0, -- top, ++ bottom;bottom < top;)
```

```
        hull[i++] = hull[bottom++];
    return i;
}
```

【判断凸包是否可以唯一确定 $pku1228$】

```
bool CheckConvex(int n,TPoint pt[])
    {
        if(n<6) return false;
        int chos=0,i,point_cnt=2,last_point=n-1;
        for(i=1;i<n;i++)
           if(pt[i].y<pt[chos].y||(pt[i].y==pt[chos].y&&pt[i].x<pt[chos].x)) chos=i;
        TPoint temp=pt[0];pt[0]=pt[chos];pt[chos]=temp;sort(pt+1,pt+n,cmp);pt[n]=pt[0];
        while(last_point>=0&&
            fabs(cross(pt[last_point-1],pt[last_point],pt[last_point+1]))<EPS)
               --last_point;

        $//所有点共线和最后一条边上只有两个点 则不满足条件$
        if(last_point<0||last_point==n-1) return false;
        pt[last_point]=pt[n-1];
        for(i=2;i<=last_point;++i)
            if(fabs(cross(pt[i-2],pt[i-1],pt[i]))<EPS) ++point_cnt;
            else{ if(point_cnt<3) return false;point_cnt=2;}
    }
```

【最远点对】
[输入]顶点集$p[]$, 顶点数n
[输出]返回最远点长度
[说明] 最远点对必然在凸包上，通过寻找对趾点判断
[注意]点少的时候可能会出错，必要时硬做，一般点要多于5个再用

```
double longest_pair(TPoint p[], int n)
{
    TPoint c[MAXN];
    int i, j, k = 1, l = Graham_Scan(p, c, n);    //如果已是凸包这里可省略
    double d, maxt = 0;
    c[l] = c[0], c[l+1] = c[1];
    while(k < l && sign(cross(c[0], c[1], c[k]) -
        cross(c[0], c[1], c[k+1])) < 0)
        ++ k;
    for(j = k, i = 0, ++ k;i < j || k < l;) {
        if(sign(cross(c[i], c[i+1], c[k]) - cross(c[i], c[i+1], c[k+1])) < 0)
            ++ k;
        else ++ i;
        d = dis(c[i], c[k]);
        if(d > maxt) maxt = d;
    }
    return maxt;
}
```

【最近点对】
[输入]$closest_pair$函数，点集$p[]$,点数n

[输出]最近顶点对距离
[说明] 分治法求最近点对,注意有点重合的情况处理

```
bool cmpx(const TPoint &a, const TPoint &b)
{
    return a.x < b.x || a.x == b.x && a.y < b.y;
}

bool cmpy(const int &a, const int &b)
{
    return p[a].y < p[b].y;
}

double dis2(TPoint a, TPoint b)
{
    return ((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
}

int p1[MAXN / 2 + 1], p2[MAXN / 2 + 1];
double find_pair(int left, int right, TPoint p[])
{
    double best = 1e20, temp;
    //if(right - left == 1) return dis2(p[left], p[right]);
    if(right - left < 10) {
        for(;left < right;++ left)
            for(int j = left + 1;j <= right;++ j)
                if(best > (temp = dis2(p[left], p[j]))) best = temp;
        return best;
    }
    int mid = (left + right) >> 1;
    double min1, min2;
    min1 = find_pair(left, mid, p);
    min2 = find_pair(mid, right, p);
    best = (min1 < min2) ? min1 : min2;
    int cnt1 = 0, cnt2 = 0, L = mid - 1, r = mid + 1, i, j, k;
    while(L >= left && p[mid].x - p[L].x < best) p1[cnt1++] = L --;
    while(r <= right && p[r].x - p[mid].x < best) p2[cnt2++] = r ++;
    //sort(p1, p1 + cnt1, cmpy);
    sort(p2, p2 + cnt2, cmpy);
    for(L = k = 0;L < cnt1;++ L) {
        for(i = p1[L], r = k;r < cnt2;++ r) {
            j = p2[r];
            if(fabs(p[j].x - p[i].x) >= best) continue;
            if(p[j].y - p[i].y >= best) break; /*KEY*/
            if(p[i].y - p[j].y >= best) { k = r; continue; }
            temp = dis2(p[j], p[i]);
            if(temp < best) best = temp;
        }
    }
    return best;
}

double closest_pair(int n, TPoint p[])
{
```

```
    sort(p, p + n, cmpx); //cmpx改成重载<会快一点点
    return sqrt(find_pair(0, n - 1, p)) * 0.5;
}
```

4.4 三角形

【基本定义】

```
struct TTriangle{ TPoint p[3]; };
```

【三角形面积】
$S=\frac{ah}{2}=\frac{ab\sin c}{2}=\frac{abc}{(4R)}=\frac{(a+b+c)*r}{2}=\sqrt{(p*(p-a)*(p-b)*(p-c))}$,
其中$p=\frac{a+b+c}{2}$,r为三角形内切圆半径,R为三角形外接圆半径

```
double triangle_area(TPoint & a, TPoint & b,TPoint & c) //三顶点坐标
{   return fabs(cross(a,b,c)) / 2.00; }

double heron(double a, double b, double c) //三边长(海伦公式)
{
    double p = (double)(a+b+c)/2.0;  //半周长
    return sqrt(p * (p - a) * (p - b) * (p - c));
}
```

【三角形外接圆】外心：三边中垂线交点，到三角形三个顶点的距离相等

```
TCircle get_out_circle(TPoint p1, TPoint p2, TPoint p3)
{
    double a11, a12, a21, a22, b1, b2;
    double d, d1, d2;
    a11 = 2 * (p3.x - p2.x);
    a12 = 2 * (p3.y - p2.y);
    a21 = 2 * (p2.x - p1.x);
    a22 = 2 * (p2.y - p1.y);
    b1 = p3.x * p3.x - p2.x * p2.x + p3.y * p3.y - p2.y*p2.y;
    b2 = p2.x * p2.x - p1.x * p1.x + p2.y * p2.y - p1.y*p1.y;
    d = a11 * a22 - a12*a21;
    d1 = b1 * a22 - a12*b2;
    d2 = a11 * b2 - b1*a21;
    TCircle ret;
    ret.o.x = d1 / d;  ret.o.y = d2 / d;
    ret.r = dis(ret.o, p1);
    return ret;
}
```

【三角形内切圆】内心：角平分线的交点，到三角形三边的距离相等
令$x=(a+b-c)/2, y=(a-b+c)/2, z=(-a+b+c)/2, h=s/p$
则到三点距离和为$\sqrt{x*x+h*h}+\sqrt{y*y+h*h}+\sqrt{z*z+h*h}$

```
TCircle get_in_circle(TTriangle t)
{
    TPoint *p = t.p;
    TCircle tmp;
```

```
    double a,b,c,xa,ya,xb,yb,xc,yc;
    c=dis(p[0],p[1]);
    a=dis(p[1],p[2]);
    b=dis(p[2],p[0]);
    tmp.r=2*triangle_area(t)/(a+b+c);
    xa=p[0].x,xb=p[1].x,xc=p[2].x;
    ya=p[0].y,yb=p[1].y,yc=p[2].y;
    tmp.o.x=(a*xa+b*xb+c*xc)/(a+b+c);
    tmp.o.y=(a*ya+b*yb+c*yc)/(a+b+c);
    return o;
}

/*已知三边求内切圆方法：
1、判断是否有平行边，如果3条边均平行则无内切圆，2条边平行则平行边距离一半为半径.
2、判断是否存在某条边与其他2条边的交点在其延长线上，如果是则不存在内切圆
3、根据内切圆半径乘与周长等于面积的二倍，即可求出内切圆半径.
可以求三边交点，再用上面的程序求,只是加个判断三点能否构成三角形
*/
```

【三角形垂心】垂心：三条高线的交点.则到三点距离和为$3*(c/2/\sqrt{1-cosC*cosC})$

```
TPoint perpencenter(TPoint a, TPoint b, TPoint c) {
    TSegment u, v;
    u.s = c;
    u.e.x = u.s.x - a.y + b.y;
    u.e.y = u.s.y + a.x - b.x;
    v.s = b;
    v.e.x = v.s.x - a.y + c.y;
    v.e.y = v.s.y + a.x - c.x;
    return seg_inter_pnt(u.s, u.e, v.s, v.e);
}
```

【三角形重心】重心：三条中线的交点，到三角形三顶点距离的平方和最小的点,三角形内到三边距离之积最大的点.
则到三点距离和为$\frac{2.0}{3*(\sqrt{(2*(a*a+b*b)-c*c)/4}+\sqrt{(2*(a*a+c*c)-b*b)/4}+\sqrt{(2*(b*b+c*c)-a*a)/4}}$

```
TPoint barycenter(TPoint a, TPoint b, TPoint c) {
    TSegment u, v;
    u.s.x = (a.x + b.x) / 2;
    u.s.y = (a.y + b.y) / 2;
    u.e = c;
    v.s.x = (a.x + c.x) / 2;
    v.s.y = (a.y + c.y) / 2;
    v.e = b;
    return seg_inter_pnt(u.s, u.e, v.s, v.e);
}
```

【三角形费马点】费马点:到三角形三个顶点的距离之和最小
若三角形的三个内角均小于120度，那么该点连接三个顶点形成的三个角均为120度；
若三角形存在一个内角大于120度，则该顶点就是费马点).计算公式如下:
若有一个内角大于120度（这里假设为角C)，则距离为$a+b$
若三个内角均小于120度，则距离为$\sqrt{(a*a+b*b+c*c+4*\sqrt{3.0}*s)}/2$,其中$s$为三角形面积

```
double dis2(TPoint p1, TPoint p2) {
    return (p2.x - p1.x)*(p2.x - p1.x) + (p2.y - p1.y)*(p2.y - p1.y);
}

double get_cos(double a, double b, double c) {
    return (a + b - c) / (2.0 * sqrt(a) * sqrt(b));
}

bool cmp_mat(TPoint p1, TPoint p2)
{
    return p1.x < p2.x || p1.x == p2.x && p1.y < p2.y;
}

TPoint triangle_fermat(TPoint p[])
{
    sort(p, p + 3, cmp_mat);
    double a = dis2(p[0], p[1]);
    double b = dis2(p[1], p[2]);
    double c = dis2(p[2], p[0]);

    double sa = get_cos(b, c, a);
    double sb = get_cos(a, c, b);
    double sc = get_cos(a, b, c);

    if (sa + 0.5 < EPS) return p[2];
    if (sb + 0.5 < EPS) return p[0];
    if (sc + 0.5 < EPS) return p[1];

    TPoint p1 = p[1] - p[0], p2 = p[2] - p[0], res1,  res2, Res;
    if (cross(p1, p2) > 0) {
        res1 = rotate(-PI / 3.0, p1);
        res2 = rotate(PI / 3.0, p2);
    }
    else {
        res1 = rotate(PI / 3.0, p1);
        res2 = rotate(-PI / 3.0, p2);
    }
    res1 = res1 + p[0];
    res2 = res2 + p[0];
    return seg_inter_pnt(res1, p[2], res2, p[1]);;
}

//费马点到点的距离
//若有一个内角大于120度（这里假设为角C），则距离为a + b,(注：角C对应的边最长的)
//若三个内角均小于120度，则距离为sqrt((a * a + b * b + c * c + 4 * sqrt(3.0) * s) /
2),其中S为三角形面积
double Radian(double a, double b, double c)
{
    double tmp = (a * a + b * b - c * c) / (2 * a * b);
    return acos(tmp);
}

double MinTot(double a, double b, double c)
```

```
{
    if (a > c) swap(c, a);
    if (b > c) swap(b, c);
    if (Radian(a, b, c) >= PI * 2.0 / 3.0) return a + b;
    return sqrt((a * a + b * b + c * c + 4.0 * sqrt(3.0) * area(a, b, c)) / 2.0);
}
```

4.5 多边形

【基本定义】

```
struct TPolygon { TPoint p[N]; int n; };
```

【判断是否是简单多边形–】

```
bool is_simple(TPolygon &poly)
{
    if(poly.n < 3) return false;
    poly.p[poly.n] = poly.p[0];
    TSegment seg1, seg2;
    for(int i = 0;i < poly.n - 1;++ i) {
        seg1.s = poly.p[i], seg1.e = poly.p[i + 1];
        for(int j = i + 1;j < poly.n;++ j) {
            seg2.s = poly.p[j], seg2.e = poly.p[j + 1];
            if(j == i + 1)
                if(on_segment(seg2.e, seg1) || on_segment(seg1.s, seg2)) return false;
            else if(j == poly.n - i - 1)
                if(on_segment(seg2.s, seg1) || on_segment(seg1.e, seg2)) return false;
            else if(seg_inter(seg1, seg2)) return false;
        }
    }
    return true;
}
```

【判断多边形是否为凸多边形】
[输出]如果多边形上三点共线，返回2，如果是凸多边形，返回1，如果是凹多边形，返回0

```
int is_convex_poly(int n, TPoint p[]) {
    int t0 = sign(cross(p[n - 1], p[0], p[1]));
    for (int i = 0; i <= n; ++i) {
        int t1 = sign(cross(p[i], p[(i + 1) % n], p[(i + 2) % n]));
        if (t1 * t0 < 0) return 0;
        if (t1 * t0 == 0) return 2;
    }
    return 1;
}
```

【判断多边形的点是顺时针还是逆时针排列】true表示逆时针

```
double is_counter(TPolygon poly)
{
    double ans = 0;
    poly.p[poly.n] = poly.p[0];
```

```
    for(int i = 0;i < poly.n;++ i)
        ans += cross(poly.p[i], poly.p[i+1]);
    return ans > 0;
}

//如果多边形式凸性的，可以使用下面这个
bool is_counter(TPolygon & poly)
{
   return sign(cross(poly.p[1] - poly.p[0], poly.p[2] - poly.p[1])) > 0;
}
```

【简单多边形面积】凸凹边形均可使用,尤其是凸包。

$$Area = \frac{1}{2}\sum_{i=0}^{n-1}\begin{vmatrix} x_i & x_{i+1} \\ y_i & y_{i+1} \end{vmatrix}$$

[输入]多边形顶点集p(按逆时针或逆时针存放),多边形顶点数n
[输出]多边形面积
[注意]防止溢出，要用高精度

```
double area_poly(TPoint p[], int n)
{
    double res = 0; p[n] = p[0];
    for(int i = 0;i < n;++ i) res += cross(p[i], p[i+1]);
    return fabs(res)/2.00;
}
```

【Pick定理(网格中)】
$Area = \frac{1}{2}EdgeDot + InnerDot - 1$
[输入]多边形顶点集p(按逆时针或逆时针存放),多边形顶点数n, 面积A，边界点E，多边形内点I
[输出]多边形面积A, 边界点数E, 内点数I

```
void Pick(TPoint p[], int n, double& A, int& E, int &I)
{
    A = area_poly(p, n), E = 0, p[n] = p[0];
    for(int i = 0;i < n;++ i) {
        int dx = p[i].x - p[i+1].x, dy = p[i].y - p[i+1].y;
        if(dx == 0 || dy == 0) E += abs(dx + dy);
        else E += gcd(abs(dx), abs(dy));
    }
    I = (int)(A + 1 - E / 2.0);
}
```

【多边形重心】已知多边形点坐标，求多边型重心(C_x, C_y)

$$C_x = \frac{1}{6Area}\sum_{i=0}^{n-1}(x_i + x_{i+1})\begin{vmatrix} x_i & x_{i+1} \\ y_i & y_{i+1} \end{vmatrix} \quad C_y = \frac{1}{6Area}\sum_{i=0}^{n-1}(y_i + y_{i+1})\begin{vmatrix} x_i & x_{i+1} \\ y_i & y_{i+1} \end{vmatrix}$$

[输入]输入多边形顶点集p[](按逆时针或逆时针存放),多边形顶点数n
[返回]多边形重心
[注意]防止溢出，要用高精度,p[]要多开个空间，即p[n]的

```
TPoint gravity(TPoint p[], int n)
{
    TPoint grav;
    grav.x=grav.y=0;
    double area = 0;
    p[n] = p[0];
    for(int i = 0;i < n;++ i) {
        double tmp = cross(p[i], p[i+1]);
        grav.x += tmp * (p[i].x + p[i+1].x);
        grav.y += tmp * (p[i].y + p[i+1].y);
        area += tmp;
    }
    grav.x /= 3 * area;
    grav.y /= 3 * area;
    return grav;
}
```

【多边形切割-方向根据点】直线seg切割poly，保留点side一边的多边形
[注意]需保证side不在seg所在直线上
[说明] 可用于半平面交

```
void cut_poly_by_point(TSegment seg, TPolygon & poly, TPoint side)
{
    TPolygon new_poly;
    TPoint interp;
    int i;
    new_poly.n = 0;
    for(i = 0;i < poly.n;++ i) {
        if(same_side(seg.s, seg.e, side, poly.p[i]) >= 0)
            new_poly.p[new_poly.n++] = poly.p[i];
        if (same_side(seg.s,seg.e, poly.p[i],poly.p[i+1]) < 0)
        new_poly.p[new_poly.n++] =
                  seg_inter_pnt(seg.s, seg.e, poly.p[i], poly.p[i+1]);
    }
    poly.n = 0;
    if(new_poly.n == 0) return ;
    poly.p[poly.n++] = new_poly.p[0];
    for(i = 1;i < new_poly.n;++ i)
       if(!same_point(new_poly.p[i], new_poly.p[i-1]))
            poly.p[poly.n++] = new_poly.p[i];
    if(poly.n != 1 && same_point(poly.p[poly.n-1] , poly.p[0])) poly.n --;
    poly.p[poly.n] = poly.p[0];
    //if(poly.n < 3) poly.n = 0;
}
```

【多边形切割-取左右方向】直线seg切割poly
[输出]side=1时poly返回seg.s指向seg.e方向的左边的多边形,side=-1时poly返回右边
[说明] 可用于半平面交

```
void cut_poly_by_side(TSegment seg, TPolygon & poly, int side = 1)
{
    TPolygon new_poly; new_poly.n = 0;
    TPoint interp, dirt = seg.e - seg.s;
```

```
    int i, next, now = sign(cross(dirt, poly.p[0] - seg.e));
    for(i = 0;i < poly.n;++ i, now = next) {
        if(now == 0 || now == side) new_poly.p[new_poly.n++] = poly.p[i];
        next = sign(cross(dirt, poly.p[i+1] - seg.e));
        if (now * next < 0)
            new_poly.p[new_poly.n++] =
               seg_inter_pnt(seg.s, seg.e, poly.p[i], poly.p[i+1]);
    }

    poly.n = 0;
    if(nw_poly.n == 0) return ;
    poly.p[poly.n++] = new_poly.p[0];
    for(i = 1;i < new_poly.n;++ i)
       if(!same_point(new_poly.p[i], new_poly.p[i-1]))
            poly.p[poly.n++] = new_poly.p[i];
    if(same_point(poly.p[poly.n-1] , poly.p[0])) poly.n --;
    poly.p[poly.n] = poly.p[0];
    //if(poly.n < 3) poly.n = 0;
}
```

【判断多边形核是否存在】$O(n^3)$

```
bool core_exist(TPolygon &poly)
 {
    int i, j, k;
    TPoint interp;
    if(!is_counter()) reverse(poly.p, poly.p + poly.n);
    poly.p[poly.n] = poly.p[0];
    for(i = 0;i < poly.n;++ i)
        for(j = i + 1;j < poly.n;++ j)
          if(fabs(cross(poly.p[i], poly.p[i+1], poly.p[j], poly.p[j+1])) > EPS) {
            interp = seg_inter_pnt(poly.p[i], poly.p[i+1], poly.p[j], poly.p[j+1]);
                for(k = 0;k < poly.n;++ k)
                    if(cross(poly.p[k], interp, poly.p[k+1]) > EPS) break;
                if(k == poly.n) return true;
            }
    return false;
}
```

【求多边形的核】$O(n^2)$
[输入] poly 保存需要求核的多边形。
[返回] true 表示存在核，结果存放在*poly_core* 中

```
bool find_core(TPolygon &poly, TPolygon &poly_core)
{
    if(!is_counter(poly))  reverse(poly.p, poly.p+poly.n);
    poly.p[poly.n] = poly.p[0];
    poly_core = poly;
    for(int i = 0;i < poly.n;++ i)
        cut_poly_by_side(TSegment(poly.p[i], poly.p[i+1]), poly_core);
    return poly_core.n > 0;
}
```

【凸多边形面积交】采用半平面交实现
[输出]凸多边形交集面积,p2保存交集边界点，n2表示交集点数
[说明] 可以通过此求得凸多边形的并集面积

```
double convex_poly_inter_area(TPoint p1[], TPoint p2[], int n1, int n2) {
    TPoint tmp[MAXN];
    int tn, i = 0, j, k, next, now;
    p1[n1] = p1[0]; p2[n2] = p2[0];
    for(i = 0;i < n1 && n2 > 2;++ i) {
        now = sign(cross(p1[i+1] - p1[i], p2[0] - p1[i+1]));
        for(j = tn = 0;j < n2;++ j, now = next) {
            if(now >= 0)  tmp[tn++] = p2[j];
            next = sign(cross(p1[i+1] - p1[i], p2[j+1] - p1[i+1]));
            if(now * next < 0)
              tmp[tn++] = seg_inter_pnt(p1[i], p1[i+1], p2[j], p2[j+1]);
        }
        for(j = 0;j < tn;++ j) p2[j] = tmp[j];
        n2 = tn; p2[n2] = p2[0];
    }
    if(n2 < 3) return 0.00;
    return area_poly(p2, n2);
}
```

【简单多边形面积交】
[输出]简单多边形交集面积
[说明] 可以通过此求得简单多边形的并集面积

```
double simple_polygon_inter_area(TPoint p1[], int n1, TPoint p2[], int n2) {
    TPoint tp1[10]={0}, tp2[10]={0};
    int tn1, tn2;
    double ret = 0.00, flag, flag0;
    for(int i = 0;i < n1;++ i) {
        tp1[1] = p1[i]; tp1[2] = p1[i+1]; flag0 = 1.00;
        if(cross(tp1[1] - tp1[0], tp1[2] - tp1[1]) < -eps)
                flag0 = -1.00, swap(tp1[1], tp1[2]);
        for(int j = 0;j < n2;++ j) {
           tp2[1] = p2[j]; tp2[2] = p2[j+1]; flag = flag0;
            if(cross(tp2[1] - tp2[0], tp2[2] - tp2[1]) < -eps)
                flag *= -1.00, swap(tp2[1], tp2[2]);
            ret += flag * convex_poly_inter_area(tp1, tp2, 3, 3);
        }
    }
    return ret;
}
```

【判断点在多边形内】

```
//遇到多边形上的点在射线上时重新取点法
//OFFSET为多边形点x、y坐标最大值者,on_edge为在多边形边上的返回值
int inside_polygon(TPoint & q, TPoint p[], int n, int on_edge = 0)
{
    TPoint q2;
    int i = 0, c;
    p[n] = p[0];
```

```
    while(i < n)
      for(c = 0, i = 0, q2.x = rand() + OFFSET, q2.y = rand() + OFFSET;i < n;++ i)
            if(on_segment(q, p[i], p[i+1])) return on_edge;
            else if(fabs(cross(p[i], q, q2)) < EPS) break;
            else if(seg_inter(q, q2, p[i], p[i+1])) c = !c;
    return c;
}

//遇到多边形上的点在射线上时,判断其附件的点法
int inside_polygon(TPoint & q, TPoint p[], int n, int on_edge = 0)
{
    TPoint q2;
    q2.x = rand() + q.x, q2.y = rand() + 1e20;  //无穷远处的点不能取太大，否则会出现大数吃小数叉乘的时候会出现0
    int i, c = 0;
    for(i = 0;i < 4;++ i) p[n+i] = p[i];
    for(i = 0;i < n;++ i) {
        if(!seg_inter(q, q2, p[i], p[i+1])) continue;
        if(on_segment(q, p[i], p[i+1])) return on_edge;
        if(same_side(q, q2, p[i], p[i+1]) < 0) c = !c;
        else if(on_segment(p[i+1], q, q2) && !on_segment(p[i+2], q, q2)
                && same_side(q, q2, p[i], p[i+2]) == -1)
            c = !c;
        else if(on_segment(p[i+2], q, q2) && same_side(q, q2, p[i], p[i+3]) == -1)
            c = !c;
    }
    return c;
}

//改进弧长法
//若点在多边形边界上，返回2，在内部返回1，在外部返回0
int get_quadrant(const TPoint &p) {
    return sign(p.x) >= 0 ? (sign(p.y) >= 0 ? 0 : 3) : (sign(p.y) >= 0 ? 1 : 2);
}

int point_in_poly(TPoint &t, TPoint q[], int n) {
    TPoint p[N];  //如果有必要，在外面定义
    q[n] = q[0];
    for (int i = 0; i <= n; i++) p[i] = q[i] - t;
    int t1 = get_quadrant(p[0]);
    int sum, i;
    for (sum = 0, i = 1; i <= n; i++) {
        if (sign(p[i].x) == 0 && sign(p[i].y) == 0) return 2;
        int f = sign(cross(p[i - 1], p[i]));
        if (f==0 && sign(p[i-1].x*p[i].x)<=0 && sign(p[i-1].y*p[i].y)<= 0)
            return 2;
        int t2 = get_quadrant(p[i]);
        if (t2 == (t1 + 1) % 4) sum++;
        else if (t2 == (t1 + 3) % 4) sum--;
        else if (t2 == (t1 + 2) % 4) {
            if (f > 0) sum += 2;
            else sum -= 2;
        }
        t1 = t2;
```

```
    }
    return sum != 0;
}
```

【点到多边形最小距离】
[输入]点p坐标,多边形点个数n,点集pt[]
[输出]点到多边形的最小距离, ret保存最近点

```
double min_p2_polygon(TPoint pt[], const int &n, const TPoint &p, TPoint &ret)
{
  double mint = 1e30;
  pt[n] = pt[0];
  for(int i = 0;i < n;++ i) {
    TSegment seg(pt[i], pt[i+1]);
    TPoint ret_p;
    double tmp = dis_p2_seg(p, seg, ret_p);
    if(tmp < mint) mint = tmp, ret = ret_p;
  }
  return mint;
}
```

【两个凸多边形之间的最短距离】通过多边形间的对踵点旋转卡壳
[输出]返回poly1,poly2多边形间的最短距离
[说明] 多边形间最小距离可能是点跟点，点跟边，边跟边直接的距离

```
double dis_poly2_poly(TPolygon &poly1, TPolygon &poly2) {
    int n = poly1.n, m = poly2.n, i, j, x, y;
    TPoint *p = poly1.p, *q = poly2.p;
    //非逆时针的，变成逆时针
    if (sign(cross(p[1] - p[0], p[2] - p[1])) < 0) reverse(p, p + n);
    if (sign(cross(q[1] - q[0], q[2] - q[1])) < 0) reverse(q, q + m);
    //求最高点最低点
    for (i = x = 0; i < n; ++i) if (p[i].y < p[x].y) x = i;
    for (j = y = 0; j < m; ++j) if (q[j].y > q[y].y) y = j;
    p[n] = p[0], q[m] = q[0];
    double ret = 1e20, temp;
    for (i = j = 0; i <= n || j <= m;) {
        if (sign(cross(p[x + 1] - p[x], q[y] - q[y + 1])) > 0) {
            temp = dis_p2_seg(q[y], p[x], p[x + 1]);
            if (ret > temp) ret = temp;
            x = ++x % n, ++i;
        } else {
            temp = dis_p2_seg(p[x], q[y], q[y + 1]);
            if (ret > temp) ret = temp;
            y = ++y % m, ++j;
        }
    }
    return ret;
}
```

【判断线段是否在简单多边形内】
[必要条件]1、线段的两个端点都在多边形内，2、线段和多边形的所有边都不内交;
[用途] 1、判断折线是否在简单多边形内, 2、判断简单多边形是否在另一个简单多边形内

[注意]如果多边形是凸多边形，下面的算法可以化简

【判断圆是否在多边形内】

```
bool circle_in_poly(TPoint &o, const double &r, TPoint p[], int &n)
{
    if (point_in_poly(o, p, n)) {
        p[n] = p[0];
        for (int i = 0; i < n; ++i)
            if (dis_p2_seg(o, p[i], p[i + 1]) + EPS < r) return false;
        return true;
    } else return false;
}
```

【给定一简单多边形，找出一个肯定在该多边形内的点】

[定理]@1、每个多边形至少有一个凸顶点2、顶点数¿=4的简单多边形至少有一条对角线@

[结论]@x坐标最大，最小的点肯定是凸顶点,y坐标最大，最小的点肯定是凸顶点@

```
TPoint a_point_insidepoly(int n, TPoint polygon[]) {
    TPoint v = polygon[0], a, b, r;
    int i, index = 0;
    for (i = 1; i < n; i++) { //寻找一个凸顶点
        if (polygon[i].y < v.y)
            v = polygon[ index = i];
    a = polygon[(index - 1 + n) % n]; //得到v的前一个顶点
    b = polygon[(index + 1) % n];     //得到v的后一个顶点
    TPoint tri[3]={a, v, b}, q;
    double md = INF;
    bool bin = false;
    for (i = 0; i < n; i++) { //寻找在三角形avb内且离顶点v最近的顶点q
        if (i == index)continue;
        if (i == (index - 1 + n) % n)continue;
        if (i == (index + 1) % n)continue;
        if (!inside_polygon(polygon[i],tri,3))continue;
        bin = true;
        if (dist(v, polygon[i]) < md)
            md = dis(v, q = polygon[i]);
    }
    if (!bin){ v = a; q = b; }//没有顶点在三角形avb内，返回线段ab中点
    r.x = (v.x + q.x) / 2; //返回线段vq的中点
    r.y = (v.y + q.y) / 2;
    return r;
    }
}
```

【已知三点求矩形第四点】

已知矩形的三个顶点(a,b,c)，计算第四个顶点d的坐标. 注意：已知的三个顶点可以是无序的

```
TPoint rect4th(TPoint &a, TPoint &b, TPoint &c)
{
    TPoint d;
    if (abs(dot(a, b, c)) < EPS) // 说明c点是直角拐角处
        d.x = a.x + b.x - c.x, d.y = a.y + b.y - c.y;
```

```
    else if (abs(dot(a, c, b)) < EPS) // 说明b点是直角拐角处
        d.x = a.x + c.x - b.x, d.y = a.y + c.y - b.x;
    else if (abs(dot(a, c, b)) < EPS) // 说明b点是直角拐角处
        d.x = c.x + b.x - a.x, d.y = c.y + b.y - a.y;
    return d;
}
```

【矩形包含】矩形2(CD)是否在1(AB)内

```
bool r2inr1(double A, double B, double C, double D)
{
    if(A < B) swap(A, B);
    if(C < D) swap(C, D);
    if(A > C && B > D) return true;
    if(C>A&&D<B&&(C*C-D*D)*sqrt(C*C+D*D-B*B)-A*(C*C+D*D)+2*C*D*B<0)
        return true;
    return false;
}
```

【求从多边形外一点p出发到一个简单多边形的切线】

[输出]如果存在返回切点,其中rp点是右切点,lp是左切点

[注意]p点一定要在多边形外,输入顶点序列是逆时针排列

[说明] 如果点在多边形内肯定无切线;凸多边形有唯一的两个切点,凹多边形就可能有多于两个的切点. 如果polygon是凸多边形，切点只有两个只要找到就可以,可以化简此算法. 如果是凹多边形还有一种算法可以求解:先求凹多边形的凸包,然后求凸包的切线

```
void pointtangentpoly(int n, TPoint polygon[], TPoint p, TPoint &rp, TPoint &lp) {
    TSegment ep, en;
    bool blp, bln;
    lp = rp = polygon[0];
    for (int i = 1; i < n; i++) {
        ep.s = polygon[(i + n - 1) % n];
        en.s = ep.e = polygon[i];
        en.e = polygon[(i + 1) % n];
        blp = cross(ep.e, p, ep.s) >= 0; // p is to the left of pre edge
        bln = cross(en.e, p, en.s) >= 0; // p is to the left of next edge
        // polygon[i] is above rp
        if (!blp && bln && cross(polygon[i], rp, p) > 0) rp = polygon[i];
        // polygon[i] is below lp
        if (blp && !bln && cross(lp, polygon[i], p) > 0)  lp = polygon[i];
    }
}
```

【费马点】

```
//多点的费马点
TPoint p[N];
double sum_pnt(double x0, double y0, int n)
{
    TPoint tp(x0, y0);
    double ret = 0;
    for(int i = 0;i < n;++ i) ret += dis(tp, p[i]);
    return ret;
}
```

```
double fermat_pnt(int n, TPoint &fp)
{
    double tx = 0, ty = 0;
    for(int i = 0;i < n;++ i) tx += p[i].x, ty += p[i].y;
    tx /= n, ty /= n;
    double mint = sum_pnt(tx, ty, n), d = 99999, g;    //d随x，y的范围自己改,一般为最值一半
    while(1) {
        int f = 0;
        if((g = sum_pnt(tx + d, ty, n)) < mint) { mint = g; tx += d; f = 1; }
        if((g = sum_pnt(tx - d, ty, n)) < mint) { mint = g; tx -= d; f = 1; }
        if((g = sum_pnt(tx, ty + d, n)) < mint) { mint = g; ty += d; f = 1; }
        if((g = sum_pnt(tx, ty - d, n)) < mint) { mint = g; ty -= d; f = 1; }
        d *= 0.9;
        if(!f && d < 1e-8) break;
    }
    fp.x = tx, fp.y = ty;
    return mint;
}
```

【扇形重心】设原点通过扇形圆心，x轴为扇形的对称轴，根据对称原理，重心必在x轴上，其横坐标(即距扇形圆心距离)

$$x_c = \frac{\frac{2*R*\sin A}{3}}{A}$$

, 其中R为半径，A为扇形圆心角的一半，取单位为rad（弧度）

4.6 圆、椭圆

【基本定义】

```
struct TCircle { double x, y, r; };
struct TCircle { TPoint o;double r; };
```

【判断点在圆内】

[返回]点p在圆内(包括边界)时，返回true
[用途]因为圆为凸集，所以判断点集，折线，多边形是否在圆内时，只需要逐一判断点是否在圆内即可。

```
bool TPoint_in_circle(TCircle c, TPoint & p)
{
    double d2 = (p.x - c.o.x) * (p.x - c.o.x) + (p.y - c.o.y) * (p.y - c.o.y);
    double r2 = c.r * c.r;
    return d2 < r2 || fabs(r2 - d2) < EPS;
}
```

【计算圆上到点p最近点,如p与圆心重合,返回p本身】

```
TPoint dot_to_circle(TPoint c, double r, TPoint p) {
    TPoint u, v;
    if (dis(p, c) < eps) return p;
    u.x = c.x + r * fabs(c.x - p.x) / dis(c, p);
```

```
    u.y = c.y + r * fabs(c.y - p.y) / dis(c, p)*((c.x - p.x)*(c.y - p.y) < 0 ? -1 : 1);
    v.x = c.x - r * fabs(c.x - p.x) / dis(c, p);
    v.y = c.y - r * fabs(c.y - p.y) / dis(c, p)*((c.x - p.x)*(c.y - p.y) < 0 ? -1 : 1);
    return dis(u, p) < dis(v, p) ? u : v;
}
```

【最小覆盖圆】

[说明] 点集存放在p[]中，m调用时要为0，即MinCircle(n, m = 0)

[注意]递归实现，点多的话可能会爆栈,可现对点集求凸包

```
TPoint p[N], Q[3];
TCircle MinCircle(int n, int &m) // n为点数
{
    TCircle c;
    if(n == 0 || m == 3) {
        if(m == 1) c.o = Q[0], c.r = 0;
        else if(m == 2) {
            c.o.x = (Q[0].x + Q[1].x) * 0.5;
            c.o.y = (Q[0].y + Q[1].y) * 0.5;
            c.r = dis(Q[0], Q[1]) * 0.5;
        }
        else c = get_out_circle(Q[0],Q[1],Q[2]);
        return c;
    }
    if(n == 1 && m == 0) c.o = p[0], c.r = 0;
    else c = MinCircle(n - 1, m);
    if(dis(c.o, p[n-1]) > c.r + EPS) {
        Q[m++] = p[n-1];
        c = MinCircle(n-1, m);
        -- m;
    }
    return c;
}
```

【圆和直线关系】

[输出]相离:0,相切:1,相交:2,rp1和rp2保存交点

```
int cricle_line_inter(TCircle cir, TLine line, TPoint &rp1, TPoint &rp2)
{
    TPoint  p = cir.o;
    double r = cir.r, a = line.a,  b = line.b, c = line.c;
    int res = 0, sga = sign(a), sgb = sign(b);
    c = c + a * p.x + b * p.y;
    if(sga == 0 && sgb == 0) return 0;
    else if(sga == 0 || sgb == 0) {
        double tmp = -c / (sga == 0 ? b : a);
        if(r * r < tmp * tmp) res = 0;
        else if(r * r == tmp * tmp) res = 1, rp1.y = tmp, rp1.x = 0;
        else {
            res = 2;
            rp1.y = rp2.y = tmp;
            rp1.x = sqrt(r * r - tmp * tmp);
            rp2.x = -rp1.x;
        }
```

```
        if(sgb == 0) { swap(rp1.x, rp1.y); swap(rp2.x, rp2.y); }
    }
    else {
        double delta = b * b * c * c - (a * a + b * b)*(c * c - a * a * r * r);
        if (sign(delta) < 0) res = 0;
        else if (sign(delta) == 0) {
            res = 1;
            rp1.y = -b * c / (a * a + b * b);
            rp1.x = (-c - b * rp1.y) / a;
        } else {
            res = 2;
            rp1.y = (-b * c + sqrt(delta)) / (a * a + b * b);
            rp2.y = (-b * c - sqrt(delta)) / (a * a + b * b);
            rp1.x = (-c - b * rp1.y) / a;
            rp2.x = (-c - b * rp2.y) / a;
        }
    }
    rp1.x += p.x; rp1.y += p.y;
    rp2.x += p.x; rp2.y += p.y;
    return res;
}
```

【线段与圆的关系】

[说明] 圆心在原点，对于圆心不在原点的可以通过向量平移得到

[输入]线段端点a,b,圆半径r

[输出]函数返回0则没有交点，返回1则有一个交点为$p1$,返回2则有两个交点为$p1,p2$

```
int find_cross_point(TPoint a, TPoint b, double r, TPoint &p1, TPoint &p2) {
    double len = dis(a, b);
    double d = fabs(cross(a, b)) / len;
    if (sign(d - r) >= 0) {
        return 0;
    }
    TPoint vec;
    if (cross(a, b, TPoint(0, 0)) < 0) {
        vec = b - a;
    } else {
        vec = a - b;
    }
    vec = TPoint(-vec.y, vec.x);
    vec.x *= d / len;
    vec.y *= d / len;
    TPoint add = (b - a);
    double tmp = sqrt(r*r-d*d) / len;
    add.x *= tmp;
    add.y *= tmp;
    TPoint tmp1 = vec + add;
    TPoint tmp2 = vec - add;
    if (dist_sqr(a, tmp1) > dist_sqr(a, tmp2)) {
        swap(tmp1, tmp2);
    }
    bool flag1 = (sign(dot(tmp1, a, b)) < 0);
```

```
    bool flag2 = (sign(dot(tmp2, a, b)) < 0);
    if (flag1 && flag2) {
        p1 = tmp1, p2 = tmp2;
        return 2;
    }
    if (flag1) {
        p1 = tmp1;
        return 1;
    }
    if (flag2) {
        p1 = tmp2;
        return 1;
    }
    return 0;
}
```

【求切点】

[输入]p:圆心坐标，r:圆半径，sp:圆外一点

[输出]rp1,rp2—切点坐标

```
void cutpoint(POINT p,double r,POINT sp,POINT &rp1,POINT &rp2)
{
    POINT p2;
    p2.x=(p.x+sp.x)/2;
    p2.y=(p.y+sp.y)/2;

    double dx2,dy2,r2;
    dx2=p2.x-p.x;
    dy2=p2.y-p.y;
    r2=sqrt(dx2*dx2+dy2*dy2);
    c2point(p,r,p2,r2,rp1,rp2);
}
```

【两圆关系】

[输出]相等:0;相离:1；外切：2；相交：3；内切：4；内含：5；

```
int CircleRelation(TCircle c1, TCircle c2)
{
    TPoint p1 = c1.o, p2 = c2.o;
    double r1 = c1.r, r2 = c2.r;
    double d = sqrt((p1.x - p2.x)*(p1.x - p2.x)+(p1.y - p2.y)*(p1.y - p2.y));
    if(fabs(p1.x - p2.x) < EPS && fabs(p1.y - p2.y) < EPS && fabs(r1 - r2) < EPS)
        return 0;
    // 必须保证前两个if先被判定
    if (fabs(d - r1 - r2) < EPS) return 2;
    if (fabs(d - fabs(r1 - r2)) < EPS) return 4;
    if (d > r1 + r2) return 1;
    if (d < fabs(r1 - r2)) return 5;
    if (fabs(r1 - r2) < d && d < r1 + r2) return 3;
    return -1; // indicate an error!
}
```

【两圆交点】

[说明] 默认两圆已经相交(相切)

```
void c2point(TPoint & p1, double r1, TPoint & p2, double r2, TPoint &rp1, TPoint &rp2)
void c2point(TCircle c1, TCircle c2, TPoint &rp1, TPoint &rp2)
{
    TPoint p1 = c1.o, p2 = c2.o;
    double r1 = c1.r, r2 = c2.r;
    double a = p2.x - p1.x, b = p2.y - p1.y,
           r = (a * a + b * b + r1 * r1 - r2 * r2) / 2;
    if (a == 0 && b != 0) {
        rp1.y = rp2.y = r / b;
        rp1.x = sqrt(r1 * r1 - rp1.y * rp1.y);
        rp2.x = -rp1.x;
    } else if (a != 0 && b == 0) {
        rp1.x = rp2.x = r / a;
        rp1.y = sqrt(r1 * r1 - rp1.x * rp2.x);
        rp2.y = -rp1.y;
    } else if (a != 0 && b != 0) {
        double delta = b * b * r * r - (a * a + b * b)*(r * r - r1 * r1 * a * a);
        rp1.y = (b * r + sqrt(delta)) / (a * a + b * b);
        rp2.y = (b * r - sqrt(delta)) / (a * a + b * b);
        rp1.x = (r - b * rp1.y) / a;
        rp2.x = (r - b * rp2.y) / a;
    }
    rp1.x += p1.x;    rp1.y += p1.y;
    rp2.x += p1.x;    rp2.y += p1.y;
}
```

【计算两圆的公共面积】

```
double circle_intersect(TCircle a, TCircle b)
{
    double d, s, A, B;
    if (a.r > b.r) { d = a.r; a.r = b.r; b.r = d; }
    d = sqrt((a.o.x-b.o.x)*(a.o.x-b.o.x)+(a.o.y-b.o.y)*(a.o.y-b.o.y));
    if(d >= a.r + b.r) return 0.0;
    if(d <= b.r - a.r) return PI * a.r * a.r;
    s = (a.r+b.r+d)/2.0;
    A = acos((a.r*a.r+d*d-b.r*b.r)/2.0/a.r/d);
    B = acos((b.r*b.r+d*d-a.r*a.r)/2.0/b.r/d);
    s = sqrt(s*(s-a.r)*(s-b.r)*(s-d));
    return A *a.r*a.r + B *b.r*b.r - 2.0*s;
}
```

【圆和简单多边形的交】

[说明] $poly_common_area$函数返回多边形与圆交的面积,dt数组保存多边形顶点,n为顶点个数,C为圆心坐标,r为圆半径

```
double dist_sqr(const TPoint &a, const TPoint &b) {
    return (a.x - b.x)*(a.x - b.x) +(a.y - b.y)*(a.y-b.y);
}
double cal_sita(double a, double b, double c) {
    double ans = (a + b - c) / (2 * sqrt(a) * sqrt(b));
    if (ans > 1) {
        ans = 1;
    }
```

```
    if (ans < -1) {
        ans = -1;
    }
    return acos(ans);
}
double sector(const double &r, const double &sita) {
    return 0.5 * r * r * sita ;
}
double common_area(TPoint a, TPoint b, double r) {
    double lena = a.x*a.x+a.y*a.y, lenb = b.x*b.x+b.y*b.y;
    double rr = r * r;
    if (sign(lena - rr) <= 0 && sign(lenb  - rr) <= 0) {
        return fabs(0.5 * cross(a, b));
    }
    TPoint p1, p2;
    int cnt = find_cross_point(a, b, r, p1, p2);
    if (cnt == 0) {
        return sector(r, cal_sita(lena, lenb, dist_sqr(a, b)));
    }
    if (cnt == 1) {
        if (lena < lenb) {
            swap(a, b);
            swap(lena, lenb);
        }
    return sector(r, cal_sita(lena, rr, dist_sqr(a, p1))) + 0.5 * fabs(cross(p1, b));
    }
    if (cnt == 2) {
    return fabs(0.5 * cross(p1, p2)) + sector(r, cal_sita(lena, rr, dist_sqr(a, p1)))
        + sector(r, cal_sita(lenb, rr, dist_sqr(b, p2)));
    }
    return 0;
}
double poly_common_area(TPoint dt[],TPoint C,double r,int n)
{
    double ans = 0;
    dt[n] = dt[0];
    dt[0] = dt[0] - C;
    for (int i = 0; i < n; ++i) {
        dt[i+1] = dt[i+1] - C ;
        ans += common_area(dt[i], dt[i + 1], r) * sign(cross(dt[i], dt[i + 1]));
    }
    return fabs(ans);
}
```

【圆锥曲线方程】平面上圆锥曲线（椭圆、双曲线、抛物线）的一般方程为

$$a_1x^2 + a_2xy + a_3y^2 + a_4x + a_5y + a_6 = 0$$

这个方程含有六个待定系数，用它们之中不为零的任意一个系数去除其它系数，实际上此方程只有五个独立的待定系数。

经过五个不同的点：(x_1,y_1)、(x_2,y_2)、(x_3,y_3)、(x_4,y_4)及(x_5,y_5)的一般圆锥曲线方程为：

$$\begin{vmatrix} x^2 & xy & y^2 & x & y & 1 \\ x_1^2 & x_1y_1 & y_1^2 & x_1 & y_1 & 1 \\ x_2^2 & x_2y_2 & y_2^2 & x_2 & y_2 & 1 \\ x_3^2 & x_3y_3 & y_3^2 & x_3 & y_3 & 1 \\ x_4^2 & x_4y_4 & y_4^2 & x_4 & y_4 & 1 \\ x_5^2 & x_5y_5 & y_5^2 & x_5 & y_5 & 1 \end{vmatrix}$$

下面是几个特定曲线简化完后的方程。

[圆]设圆上各不相同的三点为(x_1,y_1)、(x_2,y_2)和(x_3,y_3)，则有圆方程

$$\begin{vmatrix} x^2+y^2 & x & y & 1 \\ x_1^2+y_1^2 & x_1 & y_1 & 1 \\ x_2^2+y_2^2 & x_2 & y_2 & 1 \\ x_3^2+y_3^2 & x_3 & y_3 & 1 \end{vmatrix}$$

[椭圆]设椭圆上各不相同的四点为(x_1,y_1)、(x_2,y_2)、(x_3,y_3)以及(x_4,y_4)，则有椭圆方程

$$\begin{vmatrix} x^2 & y^2 & x & y & 1 \\ x_1^2 & y_1^2 & x_1 & y_1 & 1 \\ x_2^2 & y_2^2 & x_2 & y_2 & 1 \\ x_3^2 & y_3^2 & x_3 & y_3 & 1 \\ x_4^2 & y_4^2 & x_4 & y_4 & 1 \end{vmatrix}$$

4.7 三维几何

【基本定义】

```
#define eps 1e-8
#define zero(x) (((x)>0?(x):-(x))<eps)
struct point3 { double x, y, z; };
struct line3 { point3 a, b; };
struct plane3 { point3 a, b, c;};
struct sphere3 { point3 o; double r; };  //球
```

【计算cross product U x V】

```
point3 xmult(point3 u, point3 v) {
    point3 ret;
    ret.x = u.y * v.z - v.y * u.z;
    ret.y = u.z * v.x - u.x * v.z;
    ret.z = u.x * v.y - u.y * v.x;
    return ret;
}
```

【计算dot product U . V】

```
double dmult(point3 u, point3 v) {
    return u.x * v.x + u.y * v.y + u.z * v.z;
}
```

【矢量差U - V】

```
point3 subt(point3 u, point3 v) {
    point3 ret;
    ret.x = u.x - v.x;
    ret.y = u.y - v.y;
    ret.z = u.z - v.z;
    return ret;
}
```

【取平面法向量】

```
point3 pvec(plane3 s) {
    return xmult(subt(s.a, s.b), subt(s.b, s.c));
}

point3 pvec(point3 s1, point3 s2, point3 s3) {
    return xmult(subt(s1, s2), subt(s2, s3));
}
```

【两点距离,单参数取向量大小】

```
double distance(point3 p1, point3 p2) {
    return sqrt((p1.x - p2.x)*(p1.x - p2.x)+(p1.y - p2.y)*(p1.y - p2.y)
            +(p1.z - p2.z)*(p1.z - p2.z));
}
```

【向量大小】

```
double vlen(point3 p) {
    return sqrt(p.x * p.x + p.y * p.y + p.z * p.z);
}
```

【判断三个向量是否共面】只需计算三个向量的坐标组成的行列式是否为0来判断，为0则共面.

$$\begin{vmatrix} x1 & y1 & z1 \\ x2 & y2 & z2 \\ x3 & y3 & z3 \end{vmatrix} = 0$$

【判三点共线】

```
int dots_inline(point3 p1, point3 p2, point3 p3) {
    return vlen(xmult(subt(p1, p2), subt(p2, p3))) < eps;
}
```

【判四点共面】

```
int dots_onplane(point3 a, point3 b, point3 c, point3 d) {
    return zero(dmult(pvec(a, b, c), subt(d, a)));
}
```

【判断一个点集中的点是否全部共面】

```
bool coplanar(point3 points, int n)
{
    if (n < 4) return true;
    point3 p = xmult(subt(points[2], points[0]) , subt(points[1], points[0]));
    for (int i = 3; i < n; i++) if (!zero(dmult(p , points[i]))) return false;
    return true;
}
```

【判点是否在线段上,包括端点和共线】

```
int dot_online_in(point3 p, line3 l) {
    return zero(vlen(xmult(subt(p, l.a), subt(p, l.b))))
      && (l.a.x - p.x)*(l.b.x - p.x) < eps
      && (l.a.y - p.y)*(l.b.y - p.y) < eps
      && (l.a.z - p.z)*(l.b.z - p.z) < eps;
}

int dot_online_in(point3 p, point3 l1, point3 l2) {
    return zero(vlen(xmult(subt(p, l1), subt(p, l2))))
      && (l1.x - p.x)*(l2.x - p.x) < eps
      && (l1.y - p.y)*(l2.y - p.y) < eps
      && (l1.z - p.z)*(l2.z - p.z) < eps;
}
```

【判点是否在线段上,不包括端点】

```
int dot_online_ex(point3 p, line3 l) {
    return dot_online_in(p, l)
      && (!zero(p.x - l.a.x) || !zero(p.y - l.a.y) || !zero(p.z - l.a.z))
      && (!zero(p.x - l.b.x) || !zero(p.y - l.b.y) || !zero(p.z - l.b.z));
}
```

```
int dot_online_ex(point3 p, point3 l1, point3 l2) {
    return dot_online_in(p, l1, l2)
      && (!zero(p.x - l1.x) || !zero(p.y - l1.y) || !zero(p.z - l1.z))
      && (!zero(p.x - l2.x) || !zero(p.y - l2.y) || !zero(p.z - l2.z));
}
```

【判点是否在空间三角形上, 包括边界, 三点共线无意义】

```
int dot_inplane_in(point3 p, plane3 s) {
    return zero(vlen(xmult(subt(s.a, s.b), subt(s.a, s.c)))
      - vlen(xmult(subt(p, s.a), subt(p, s.b)))
      - vlen(xmult(subt(p, s.b), subt(p, s.c)))
      - vlen(xmult(subt(p, s.c), subt(p, s.a))));
}

int dot_inplane_in(point3 p, point3 s1, point3 s2, point3 s3) {
    return zero(vlen(xmult(subt(s1, s2), subt(s1, s3)))
      - vlen(xmult(subt(p, s1), subt(p, s2)))
      - vlen(xmult(subt(p, s2), subt(p, s3)))
      - vlen(xmult(subt(p, s3), subt(p, s1))));
}
```

【判点是否在空间三角形上, 不包括边界, 三点共线无意义】

```
int dot_inplane_ex(point3 p, plane3 s) {
    return dot_inplane_in(p, s) && vlen(xmult(subt(p, s.a), subt(p, s.b))) > eps
      && vlen(xmult(subt(p, s.b), subt(p, s.c))) > eps
      && vlen(xmult(subt(p, s.c), subt(p, s.a))) > eps;
}

int dot_inplane_ex(point3 p, point3 s1, point3 s2, point3 s3) {
    return dot_inplane_in(p, s1, s2, s3) && vlen(xmult(subt(p, s1), subt(p, s2))) > eps
      && vlen(xmult(subt(p, s2), subt(p, s3))) > eps
      && vlen(xmult(subt(p, s3), subt(p, s1))) > eps;
}
```

【判两点在线段同侧, 点在线段上返回0, 不共面无意义】

```
int same_side(point3 p1, point3 p2, line3 l) {
    return dmult(xmult(subt(l.a, l.b), subt(p1, l.b)),
      xmult(subt(l.a, l.b), subt(p2, l.b))) > eps;
}

int same_side(point3 p1, point3 p2, point3 l1, point3 l2) {
    return dmult(xmult(subt(l1, l2), subt(p1, l2)),
      xmult(subt(l1, l2), subt(p2, l2))) > eps;
}
```

【判两点在线段异侧, 点在线段上返回0, 不共面无意义】

```
int opposite_side(point3 p1, point3 p2, line3 l) {
    return dmult(xmult(subt(l.a, l.b), subt(p1, l.b)),
      xmult(subt(l.a, l.b), subt(p2, l.b)))<-eps;
}
```

```
int opposite_side(point3 p1, point3 p2, point3 l1, point3 l2) {
    return dmult(xmult(subt(l1, l2), subt(p1, l2)),
      xmult(subt(l1, l2), subt(p2, l2)))<-eps;
}
```

【判两点在平面同侧,点在平面上返回0】

```
int same_side(point3 p1, point3 p2, plane3 s) {
    return dmult(pvec(s), subt(p1, s.a))
      * dmult(pvec(s), subt(p2, s.a)) > eps;
}

int same_side(point3 p1, point3 p2, point3 s1, point3 s2, point3 s3) {
    return dmult(pvec(s1, s2, s3), subt(p1, s1))
      * dmult(pvec(s1, s2, s3), subt(p2, s1)) > eps;
}
```

【判两点在平面异侧,点在平面上返回0】

```
int opposite_side(point3 p1, point3 p2, plane3 s) {
    return dmult(pvec(s), subt(p1, s.a))
      * dmult(pvec(s), subt(p2, s.a))<-eps;
}

int opposite_side(point3 p1, point3 p2, point3 s1, point3 s2, point3 s3) {
    return dmult(pvec(s1, s2, s3), subt(p1, s1))
      * dmult(pvec(s1, s2, s3), subt(p2, s1))<-eps;
}
```

【判两直线平行】

```
int parallel(line3 u, line3 v) {
    return vlen(xmult(subt(u.a, u.b), subt(v.a, v.b))) < eps;
}

int parallel(point3 u1, point3 u2, point3 v1, point3 v2) {
    return vlen(xmult(subt(u1, u2), subt(v1, v2))) < eps;
}
```

【判两平面平行】

```
int parallel(plane3 u, plane3 v) {
    return vlen(xmult(pvec(u), pvec(v))) < eps;
}

int parallel(point3 u1, point3 u2, point3 u3, point3 v1, point3 v2, point3 v3) {
    return vlen(xmult(pvec(u1, u2, u3), pvec(v1, v2, v3))) < eps;
}
```

【判直线与平面平行】

```
int parallel(line3 l, plane3 s) {
    return zero(dmult(subt(l.a, l.b), pvec(s)));
}
```

```
int parallel(point3 l1, point3 l2, point3 s1, point3 s2, point3 s3) {
    return zero(dmult(subt(l1, l2), pvec(s1, s2, s3)));
}
```

【判两直线垂直】

```
int perpendicular(line3 u, line3 v) {
    return zero(dmult(subt(u.a, u.b), subt(v.a, v.b)));
}

int perpendicular(point3 u1, point3 u2, point3 v1, point3 v2) {
    return zero(dmult(subt(u1, u2), subt(v1, v2)));
}
```

【判两平面垂直】

```
int perpendicular(plane3 u, plane3 v) {
    return zero(dmult(pvec(u), pvec(v)));
}

int perpendicular(point3 u1, point3 u2, point3 u3, point3 v1, point3 v2, point3 v3) {
    return zero(dmult(pvec(u1, u2, u3), pvec(v1, v2, v3)));
}
```

【判直线与平面平行】

```
int perpendicular(line3 l, plane3 s) {
    return vlen(xmult(subt(l.a, l.b), pvec(s))) < eps;
}

int perpendicular(point3 l1, point3 l2, point3 s1, point3 s2, point3 s3) {
    return vlen(xmult(subt(l1, l2), pvec(s1, s2, s3))) < eps;
}
```

【判两线段相交,包括端点和部分重合】

```
int intersect_in(line3 u, line3 v) {
    if (!dots_onplane(u.a, u.b, v.a, v.b))
        return 0;
    if (!dots_inline(u.a, u.b, v.a) || !dots_inline(u.a, u.b, v.b))
        return !same_side(u.a, u.b, v) && !same_side(v.a, v.b, u);
    return dot_online_in(u.a, v) || dot_online_in(u.b, v)
      || dot_online_in(v.a, u) || dot_online_in(v.b, u);
}

int intersect_in(point3 u1, point3 u2, point3 v1, point3 v2) {
    if (!dots_onplane(u1, u2, v1, v2))
        return 0;
    if (!dots_inline(u1, u2, v1) || !dots_inline(u1, u2, v2))
        return !same_side(u1, u2, v1, v2) && !same_side(v1, v2, u1, u2);
    return dot_online_in(u1, v1, v2) || dot_online_in(u2, v1, v2)
      || dot_online_in(v1, u1, u2) || dot_online_in(v2, u1, u2);
}
```

【判两线段相交,不包括端点和部分重合】

```
int intersect_ex(line3 u, line3 v) {
    return dots_onplane(u.a, u.b, v.a, v.b)
      && opposite_side(u.a, u.b, v)
      && opposite_side(v.a, v.b, u);
}

int intersect_ex(point3 u1, point3 u2, point3 v1, point3 v2) {
    return dots_onplane(u1, u2, v1, v2)
      && opposite_side(u1, u2, v1, v2)
      && opposite_side(v1, v2, u1, u2);
}
```

【判线段与空间三角形相交,包括交于边界和(部分)包含】

```
int intersect_in(line3 l, plane3 s) {
    return !same_side(l.a, l.b, s) && !same_side(s.a, s.b, l.a, l.b, s.c) &&
           !same_side(s.b, s.c, l.a, l.b, s.a)
           && !same_side(s.c, s.a, l.a, l.b, s.b);
}

int intersect_in(point3 l1, point3 l2, point3 s1, point3 s2, point3 s3) {
    return !same_side(l1, l2, s1, s2, s3) && !same_side(s1, s2, l1, l2, s3) &&
           !same_side(s2, s3, l1, l2, s1) && !same_side(s3, s1, l1, l2, s2);
}
```

【判线段与空间三角形相交,不包括交于边界和(部分)包含】

```
int intersect_ex(line3 l, plane3 s) {
    return opposite_side(l.a, l.b, s) && opposite_side(s.a, s.b, l.a, l.b, s.c)
      && opposite_side(s.b, s.c, l.a, l.b, s.a)
      && opposite_side(s.c, s.a, l.a, l.b, s.b);
}

int intersect_ex(point3 l1, point3 l2, point3 s1, point3 s2, point3 s3) {
    return opposite_side(l1, l2, s1, s2, s3) && opposite_side(s1, s2, l1, l2, s3) &&
           opposite_side(s2, s3, l1, l2, s1) && opposite_side(s3, s1, l1, l2, s2);
}
```

【计算两直线交点,注意事先判断直线是否共面和平行!】

[说明] 线段交点请另外判线段相交(同时还是要判断是否平行!)

```
point3 intersection(line3 u, line3 v)
{
    point3 ret = u.a;
    double t = ((u.a.x - v.a.x)*(v.a.y - v.b.y)-(u.a.y - v.a.y)*(v.a.x - v.b.x))
        / ((u.a.x - u.b.x)*(v.a.y - v.b.y)-(u.a.y - u.b.y)*(v.a.x - v.b.x));
    ret.x += (u.b.x - u.a.x) * t;
    ret.y += (u.b.y - u.a.y) * t;
    ret.z += (u.b.z - u.a.z) * t;
    return ret;
}
point3 intersection(point3 u1, point3 u2, point3 v1, point3 v2)
{
    point3 ret = u1;
```

```
    double t = ((u1.x - v1.x)*(v1.y - v2.y)-(u1.y - v1.y)*(v1.x - v2.x))
            / ((u1.x - u2.x)*(v1.y - v2.y)-(u1.y - u2.y)*(v1.x - v2.x));
    ret.x += (u2.x - u1.x) * t;
    ret.y += (u2.y - u1.y) * t;
    ret.z += (u2.z - u1.z) * t;
    return ret;
}
```

【计算直线与平面交点,注意事先判断是否平行,并保证三点不共线!】

[说明] 线段和空间三角形交点请另外判断

```
point3 intersection(line3 l, plane3 s) {
    point3 ret = pvec(s);
    double t = (ret.x * (s.a.x - l.a.x) + ret.y * (s.a.y - l.a.y)
      + ret.z * (s.a.z - l.a.z)) /
            (ret.x * (l.b.x - l.a.x) + ret.y * (l.b.y - l.a.y)
      + ret.z * (l.b.z - l.a.z));
    ret.x = l.a.x + (l.b.x - l.a.x) * t;
    ret.y = l.a.y + (l.b.y - l.a.y) * t;
    ret.z = l.a.z + (l.b.z - l.a.z) * t;
    return ret;
}

point3 intersection(point3 l1, point3 l2, point3 s1, point3 s2, point3 s3) {
    point3 ret = pvec(s1, s2, s3);
    double t = (ret.x * (s1.x - l1.x) + ret.y * (s1.y - l1.y)
       + ret.z * (s1.z - l1.z)) /
         (ret.x * (l2.x - l1.x) + ret.y * (l2.y - l1.y)
       + ret.z * (l2.z - l1.z));
    ret.x = l1.x + (l2.x - l1.x) * t;
    ret.y = l1.y + (l2.y - l1.y) * t;
    ret.z = l1.z + (l2.z - l1.z) * t;
    return ret;
}
```

【计算两平面交线,注意事先判断是否平行,并保证三点不共线!】

```
line3 intersection(plane3 u, plane3 v) {
    line3 ret;
    ret.a = parallel(v.a, v.b, u.a, u.b, u.c) ?
      intersection(v.b, v.c, u.a, u.b, u.c) : intersection(v.a, v.b, u.a, u.b, u.c);
    ret.b = parallel(v.c, v.a, u.a, u.b, u.c) ?
      intersection(v.b, v.c, u.a, u.b, u.c) : intersection(v.c, v.a, u.a, u.b, u.c);
    return ret;
}

line3 intersection(point3 u1, point3 u2, point3 u3, point3 v1, point3 v2, point3 v3) {
    line3 ret;
    ret.a = parallel(v1, v2, u1, u2, u3) ?
      intersection(v2, v3, u1, u2, u3) : intersection(v1, v2, u1, u2, u3);
    ret.b = parallel(v3, v1, u1, u2, u3) ?
      intersection(v2, v3, u1, u2, u3) : intersection(v3, v1, u1, u2, u3);
    return ret;
}
```

【点到线段距离】三分法,返回线段上距离最近的点d则$distance(a,d)$即为距离

```
point3 p2seg3(point3 a, point3 b, point3 c) {
    point3 d, e;
    while (distance(b, c) > eps) {
        d = b + c; d.x *= 0.5; d.y *= 0.5; d.z *= 0.5;
        e = d + b; e.x *= 0.5; e.y *= 0.5; e.z *= 0.5;
        if (distance(a , d) <= distance(a , e)) b = e;
        else c = d;
    }
    return b;
}
```

【点到直线距离】

```
double ptoline(point3 p, line3 l) {
    return vlen(xmult(subt(p, l.a), subt(l.b, l.a))) / distance(l.a, l.b);
}

double ptoline(point3 p, point3 l1, point3 l2) {
    return vlen(xmult(subt(p, l1), subt(l2, l1))) / distance(l1, l2);
}
```

【点到平面距离】

```
double ptoplane(point3 p, plane3 s) {
    return fabs(dmult(pvec(s), subt(p, s.a))) / vlen(pvec(s));
}

double ptoplane(point3 p, point3 s1, point3 s2, point3 s3) {
    return fabs(dmult(pvec(s1, s2, s3), subt(p, s1))) / vlen(pvec(s1, s2, s3));
}
```

【直线到直线距离】

```
double linetoline(line3 u, line3 v) {
    point3 n = xmult(subt(u.a, u.b), subt(v.a, v.b));
    return fabs(dmult(subt(u.a, v.a), n)) / vlen(n);
}

double linetoline(point3 u1, point3 u2, point3 v1, point3 v2) {
    point3 n = xmult(subt(u1, u2), subt(v1, v2));
    return fabs(dmult(subt(u1, v1), n)) / vlen(n);
}
```

【两直线夹角cos值】

```
double angle_cos(line3 u, line3 v) {
    return dmult(subt(u.a, u.b), subt(v.a, v.b))
      / vlen(subt(u.a, u.b)) / vlen(subt(v.a, v.b));
}

double angle_cos(point3 u1, point3 u2, point3 v1, point3 v2) {
    return dmult(subt(u1, u2), subt(v1, v2))
      / vlen(subt(u1, u2)) / vlen(subt(v1, v2));
}
```

【两平面夹角cos值】

```
double angle_cos(plane3 u, plane3 v) {
    return dmult(pvec(u), pvec(v)) / vlen(pvec(u)) / vlen(pvec(v));
}

double angle_cos(point3 u1, point3 u2, point3 u3, point3 v1, point3 v2, point3 v3) {
    return dmult(pvec(u1, u2, u3), pvec(v1, v2, v3))
      / vlen(pvec(u1, u2, u3)) / vlen(pvec(v1, v2, v3));
}
```

【直线平面夹角sin值】

```
double angle_sin(line3 l, plane3 s) {
    return dmult(subt(l.a, l.b), pvec(s)) / vlen(subt(l.a, l.b)) / vlen(pvec(s));
}

double angle_sin(point3 l1, point3 l2, point3 s1, point3 s2, point3 s3) {
    return dmult(subt(l1, l2), pvec(s1, s2, s3))
      / vlen(subt(l1, l2)) / vlen(pvec(s1, s2, s3));
}
```

【两向量夹角】

```
double angle(point3 a, point3 b) {
    return acos(dmult(a, b)/vlen(a)/vlen(b));
    }
```

【球面2点圆心角】

[输入]圆心角lat表示纬度,lng表示经度$-90 <= w <= 90$,如果是西半球lng为负，南半球lat为负

[输出]两点所在大圆劣弧对应圆心角,$0 <= angle <= pi$

```
double cen_angle(double lng1, double lat1, double lng2, double lat2) {
    double dlng = fabs(lng1 - lng2) * pi / 180;
    while (dlng >= pi + pi) dlng -= pi + pi;
    if (dlng > pi) dlng = pi + pi - dlng;
    lat1 *= pi / 180, lat2 *= pi / 180;
    return acos(cos(lat1) * cos(lat2) * cos(dlng) + sin(lat1) * sin(lat2));
}
```

【球面上两点球面距离】

[说明] A、B两点的球面距离L = V * R , V为向量OA与向量OB的夹角, R为半径

```
double sphere_dist(point3 a, point3 b, sphere3 sp) {
    return angle(subt(a, sp.o), subt(b, sp.o)) * sp.r;
}
//计算球面距离,r为球半径,如果是西半球lng为负，南半球lat为负
double sphere_dist(double r, double lng1, double lat1, double lng2, double lat2) {
    return r * cen_angle(lng1, lat1, lng2, lat2);
}
```

【地球上一些计算】假设地球是球体.

某点地球上度为A,分为B,则角度(PI进制下为)$P = (A + B/60.0) * PI/180$;

设地球上某点的经度为lng ,纬度为lat(其中lng和lat均为PI进制，如果是90度进制的需转换)

则这点的空间坐标是$x = cos(lat) * cos(lng), y = cos(lat) * sin(lng), z = sin(lat)$
如果是西半球lng为负，南半球lat为负,即坐标的正负与经纬度的半球有关。球面距离:$arc = R * acos(ax * bx + ay * by + az * bz)$; 直接距离:$dist = R * sqrt((ax - bx) * (ax - bx) + (ay - by) * (ay - by) + (az - bz) * (az - bz))$;

4.8 四面体

【已知点坐标求体积】
【已知边长度求体积】
|AB| = a, |AC| = b, |AD| = d, |BC| = c', |BD|=b', |CD| = a'.则公式为

$$V^2 = \frac{1}{36}\begin{bmatrix} a^2 & \frac{a^2+b^2-c'^2}{2} & \frac{a^2+c^2-b'^2}{2} \\ \frac{a^2+b^2-c'^2}{2} & b^2 & \frac{b^2+c^2-a'^2}{2} \\ \frac{a^2+c^2-b'^2}{2} & \frac{b^2+c^2-a'^2}{2} & c^2 \end{bmatrix}$$

设$M = b^2 + c^2 - a'^2, N = a^2 + c^2 - b'^2, P = a^2 + b^2 - c'^2$, 则
$V = \frac{1}{6}\sqrt{(abc)^2 - \frac{a^2M^2+b^2N^2+c^2P^2-MNP}{4}}$
[输入] 对应边长
[返回]四面体体积
[注意]边与边的参数对应关系

```
double volume(double a, double b, double c, double al, double bl, double cl)
{
    double aa = a * a, bb = b * b, cc = c * c;
    double m = bb + cc - al * al, n = aa + cc - bl * bl,
        p = aa + bb - cl * cl;
    double v = aa*bb*cc - ((aa*m*m)+(bb*n*n)+(cc*p*p)-m*n*p)/4.0;
    return sqrt(v)/6.0;
}
```

【已知三边及夹角求体积】
若四面体由一个顶点出发的三条棱长分别是a，b，c，其中每两条棱的夹角分别为α, β, γ,则这个四面体的体积

$$V = \frac{abc}{6}\sqrt{1 - \cos\alpha^2 - \cos\beta^2 - \cos\gamma^2 + 2\cos\alpha\cos\beta\cos\gamma}$$

4.9 注意事项

1. 如果没有特殊说明，所有数组均从p[0]开始，p[]的大小应至少比元素个数大一个，既$p[0] \cdots p[n-1]$。
2. 整数几何注意crsoss和dot是否会出界;符点几何注意eps的使用.
3. 计算几何类题目要注意边界条件的处理，如只有一个点、一条边、多点共线等,多测试不对称数据.。

4. 设计算法时尽量避免浮点运算，例如极角排序过程可以通过向量的叉乘大小排序，无需求出角度后再排序。浮点数的大小比较要用 $a > b + \epsilon$ 的形式来判断。
5. 设计算法时尽量避免求斜率，提倡使用向量运算。求斜率推荐使用 `tan2(y, x)` ,注意: `atan2(0,0)=0, atan2(1,0)=pi/2, atan2(-1,0)=-pi/2, atan2(0,1)=0, atan2(0,-1)=pi.`
6. 浮点数输出不能遗漏小数点: `printf("%.0lf\n", 0.);`
7. 定义π: `#define pi acos(-1.)` 或直接`#define pi 3.14159265358979323846264338327795`
8. 如果程序中用到数学函数，一定要`#include<math.h>`，否则虽然编译能通过，但无法得到正确结果
9. 注意舍入方式(0.5的舍入方向),结果有时会出现- 0.00可以将其加上eps
10. 有些一句话的函数可以写成`#define ()()`的预编译语句，这样可以将函数调用工作转移到编译工程中完成，减少运行时函数调用的消耗
11. 判断同一个$2 * PI$域内两角度差应该是$abs(a1 - a2) < beta||abs(a1 - a2) > pi + pi - beta$, 相等应该是$abs(a1 - a2) < eps||abs(a1 - a2) > pi + pi - eps$

第五章 图论

5.1 图的表示

【图的表示】邻接矩阵、邻接表、边表。在有些情况下，图的某些顶点（比如拆点后得到的顶点）的相邻节点以及边权很容易及时求出，那么在这种情况下，就没必要将这些节点保存下来了。

【邻接表】一般直接使用vector<int>, 当vector<int>太慢时使用这个.使用前请先设置$tot = ENDFLAG + 1$; 遍历第k个节点的链表方法是:

for (int i = tree.start[k]; i != ENDFLAG; i = tree.edge[i][1]) {int next = tree.edge[i][0]; }

```
const int N = 100005; //图的最大点数
const int EDGENUM = 300005; //图的最大边数
const int ENDFLAG = 0;
struct EDGELIST {
    int start[N];
    int last[N];
    int edge[EDGENUM][2]; //pos,listnext
    int tot;

    void clear() {
        tot = ENDFLAG + 1;
        memset(last, ENDFLAG, sizeof (last));
        memset(start, ENDFLAG, sizeof (start));
    }
    //添加点s到点t的单向边
    //如果要添加双向边，则tree.push(s, t); if(s != t) tree.push_back(t, s);
    void push_back(int s, int t) {
        edge[tot][0] = t;
        edge[tot][1] = ENDFLAG;
        if (last[s] != ENDFLAG) edge[ last[s] ][1] = tot;
        else start[s] = tot;
        last[s] = tot ++;
    }
} tree;
```

【二分图邻接表转化为二分图矩阵】邻接表g[n][n]，g[i][0]表示个数。输出二分图矩阵G[a][b]

```
int G[N][N],g[N][N];
int mark[N],num[N],a,b;
void dfs(int p,int k)
{
  int i;
```

```
  for (i=1;i<=g[p][0];i++) if (!mark[g[p][i]]) {
    mark[g[p][i]]=k;dfs(g[p][i],3-k);}
}
memset(mark,0,sizeof(mark));
memset(G,0,sizeof(G));
a=0;b=0;
for (i=0;i<n;i++)
if (!mark[i]) {mark[i]=1;dfs(i,2);} //标号后还会剩下没有边的节点无法标号
for (i=0;i<n;i++) {
  if (mark[i]==1) num[i]=a++;
  if (mark[i]==2) num[i]=b++;
}
for (i=0;i<n;i++) {
  if (mark[i]==1) for (j=1;j<=g[i][0];j++) G[num[i]][num[g[i][j]]]=1;
  if (mark[i]==2) for (j=1;j<=g[i][0];j++) G[num[g[i][j]]][num[i]]=1;
}
```

5.2 图的基本算法

【欧拉公式】$V-E+F=2$
【欧拉回路】stack中保存一条欧拉回路

```
int graph[][N];
int stack[M];
void euler(int i) {
  for (int j = 0; j < n; ++j)
    if (graph[i][j] > 0) {
      graph[i][j]--;
      graph[j][i]--;
      euler(j);
    }
  stack[top++] = i;
}
```

【拓扑排序】邻接阵形式,复杂度$O(n^2)$
[输出]如果无法完成排序,返回0,否则返回1,ret返回有序点列
[说明] 传入图的大小n和邻接阵mat,不相邻点边权0

```
#define MAXN 100

int toposort(int n, int mat[][MAXN], int* ret){
   int d[MAXN], i, j, k;
   for (i = 0; i < n; i++)
      for (d[i] = j = 0; j < n; d[i] += mat[j++][i]) ;
   for (k = 0; k < n; ret[k++] = i){
      for (i = 0; d[i] && i < n; i++) ;
      if (i == n) return 0;
      for (d[i] = -1,j = 0; j < n; j++)
         d[j] -= mat[i][j];
   }
   return 1;
```

```
}
```

5.3 树的优化算法

【最大顶点独立集】

```
int max_node_independent(int n, int* pre, int* set){
   int c[MAXN], i, ret = 0;
   for (i = 0; i < n; i++)
      c[i] = set[i] = 0;
   for (i = n - 1; i >= 0; i--)
      if (!c[i]){
         set[i] = 1;
         if (pre[i] != -1)
            c[pre[i]] = 1;
         ret++;
      }
   return ret;
}
```

【最大边独立集】

```
int max_edge_independent(int n, int* pre, int* set){
   int c[MAXN], i, ret = 0;
   for (i = 0; i < n; i++)
      c[i] = set[i] = 0;
   for (i = n - 1; i >= 0; i--)
      if (!c[i] && pre[i] != -1 && !c[pre[i]]){
         set[i] = 1;
         c[pre[i]] = 1;
         ret++;
      }
   return ret;
}
```

【最小顶点覆盖集】

```
int min_node_cover(int n, int* pre, int* set){
   int c[MAXN], i, ret = 0;
   for (i = 0; i < n; i++)
      c[i] = set[i] = 0;
   for (i = n - 1; i >= 0; i--)
      if (!c[i] && pre[i] != -1 && !c[pre[i]]){
         set[i] = 1;
         c[pre[i]] = 1;
         ret++;
      }
   return ret;
}
```

【最小顶点支配集】

```
int min_node_dominant(int n, int* pre, int* set){
   int c[MAXN], i, ret = 0;
   for (i = 0; i < n; i++)
      c[i] = set[i] = 0;
   for (i = n - 1; i >= 0; i--)
      if (!c[i] && (pre[i] == -1 || !set[pre[i]])){
         if (pre[i] != -1){
            set[pre[i]] = 1;
            c[pre[i]] = 1;
            if (pre[pre[i]] != -1)
               c[pre[pre[i]]] = 1;
         }
         else
            set[i] = 1;
         ret++;
      }
   return ret;
}
```

5.4 生成树

【最小生成树Prim $O(n^2)$】
[输入]n表示当前顶点数，cost[N][N]顶点之间边长度,N为最大顶点数。
[输出]返回最小生成树的边总长度，pre[u]–u是生成树的边。
[说明]cost[N][N]不连通时权值无穷大。
[注意]顶点间边的长度类型是int 还是float。

```
int prim(int n, int cost[][N], int pre[]) {
  int d[N], tmp, result = 0;
  int q[N], i, j, top = n - 1, u, best_i;
  for (i = 1; i < n; ++i) d[i] = cost[0][i], q[i - 1] = i, pre[i] = 0;
  for (i = 1; i < n; ++i) {
    for (best_i = 0, j = 1; j < top; ++j)
      if (d[q[j]] < d[q[best_i]]) best_i = j;
    u = q[best_i];
    result += cost[pre[u]][u];
    //pre[u]--u 是生成树的边
    q[best_i] = q[--top];
    for (j = 0; j < top; ++j)
      if (d[q[j]] >  cost[q[j]][u])
        d[q[j]] = cost[q[j]][u], pre[q[j]] = u;
  }
  return result;
}
```

【次小生成树–(程序有误)】复杂度$O(n^2)$, 传入邻接阵mat, 不存在边权inf, 返回次小生成树长度和树的构造pre[], 如返回inf则不存在次小生成树, 必须保证图的连通

```
const int maxn = 100;
const int inf = 1000000000;
```

```
typedef int elem_t;

elem_t prim(int n, elem_t mat[][maxn], int *pre) {
    elem_t min[maxn], ret = 0;
    int v[maxn], i, j, k;
    for (i = 0; i < n; ++i)
        min[i] = inf, v[i] = 0, pre[i] = -1;
    for (min[j = 0] = 0; j < n; ++j) {
        for (k = -1, i = 0; i < n; ++i)
          if (!v[i] && (k == -1 || min[i] < min[k]))
            k = i;
        for (v[k] = 1, ret += min[k], i = 0; i < n; ++i)
            if (!v[i] && mat[k][i] < min[i])
                min[i] = mat[pre[i] = k][i];
    } return ret;
}

elem_t sbmst(int n, elem_t mat[][maxn], int* pre) {
    elem_t min = inf, t, ret = prim(n, mat, pre);
    int i, j, ti, tj;
    for (i = 0; i < n; ++i)
      for (j = 0; j < n && pre[i] != -1; ++j)
        if (i != j && pre[i] != j && pre[j] != i)
         if (mat[j][i] < inf &&
            (t = mat[j][i] - mat[pre[i]][i]) < min)
            min = t, ti = i, tj = j;
    pre[ti] = tj;
    return ret + min;
}
```

【最小树形图】edmonds算法,邻接阵形式,复杂度$O(n^3)$, 返回最小生成树的长度,构造失败返回负值,传入图的大小n和邻接阵mat,不相邻点边权inf,可更改边权的类型,pre[]返回树的构造,用父结点表示,传入时pre[]数组清零,用-1标出源点.
[注意]**注意需要先判断根是否可以到达传入图的每一个顶点再使用**
[例题] UVA 11865 Stream My Contest

```
#include <string.h>
#define MAXN 120
#define inf 1000000000
typedef int elem_t;

elem_t edmonds(int n,elem_t mat[][MAXN*2],int* pre){
  elem_t ret=0;
  int c[MAXN*2][MAXN*2],l[MAXN*2],p[MAXN*2],m=n,t,i,j,k;
  for (i=0;i<n;l[i]=i,i++);
  do{
    memset(c,0,sizeof(c)),memset(p,0xff,sizeof(p));
    for (t=m,i=0;i<m;c[i][i]=1,i++);
    for (i=0;i<t;i++)
      if (l[i]==i&&pre[i]!=-1){
        for (j=0;j<m;j++)
          if (l[j]==j&&i!=j&&mat[j][i]<inf&&
              (p[i]==-1||mat[j][i]<mat[p[i]][i]))
```

```
            p[i]=j;
        if ((pre[i]=p[i])==-1)
          return -1;
        if (c[i][p[i]]){
          for (j=0;j<=m;mat[j][m]=mat[m][j]=inf,j++);
          for (k=i;l[k]!=m;l[k]=m,k=p[k])
            for (j=0;j<m;j++)
              if (l[j]==j){
                if (mat[j][k]-mat[p[k]][k]<mat[j][m])
                  mat[j][m]=mat[j][k]-mat[p[k]][k];
                if (mat[k][j]<mat[m][j])
                  mat[m][j]=mat[k][j];
              }
          c[m][m]=1,l[m]=m,m++;
        }
        for (j=0;j<m;j++)
          if (c[i][j])
            for (k=p[i];k!=-1&&l[k]==k;c[k][j]=1,k=p[k]);
      }
  }
  while (t<m);
  for (;m-->n;pre[k]=pre[m])
    for (i=0;i<m;i++)
      if (l[i]==m){
        for (j=0;j<m;j++)
          if (pre[j]==m&&mat[i][j]==mat[m][j])
            pre[j]=i;
        if (mat[pre[m]][m]==mat[pre[m]][i]-mat[pre[i]][i])
          k=i;
      }
  for (i=0;i<n;i++)
    if (pre[i]!=-1)
      ret+=mat[pre[i]][i];
  return ret;
}
```

5.5 最短路

【**Dijkstra** $O(n^2)$】所有边的权非负，不连通时权值无穷大。顶点从0开始计数。要注意输入数据的s,t是否也是从0开始的。path[i], 表示图中最短路是从path[i] 到i。

```
const int Inf = 0x3fffffff;

int Dijkstra(int n, int s, int t, int c[][N], int path[]) {
    int i, d[N], q[N], path[N], best_i, u;
    int top = n;
    for (i = 0; i < n; ++i)
        d[i] = Inf, q[i] = i;
    d[s] = 0;
    path[s] = s;
    for (i = 0; i < n; ++i) {
```

```
        best_i = 0;
        for (j = 0; j < top; ++j) if (d[q[j]] < d[q[best_i]]) best_i = j;
        u = q[best_i];
        if (u == t) return d[t];
        q[best_i] = q[--top];
        for (j = 0; j < top; ++j) {
            if (d[q[j]] > c[u][q[j]] + d[u])
                d[q[j]] = c[u][q[j]] + d[u], path[q[j]] = u;
        }
    }
    return d[t];
}
```

【Bellman Ford $O(nm)$】如果存在一个从源点可达的权为负的回路则**false**

```
struct eagle {
    int s, t, l;
} e[M];
bool Bellman_Ford(int n, int m, int d[], int s, eagle e[]) {
    int i, j;
    bool cc;
    for (i = 0; i < n; ++i) d[i] = Inf;
    d[s]=0;
    for (i = 0; i < n; ++i) {
        cc = false;
        for (j = 0; j < m; ++j) {
            if (d[e[j].t] > d[e[j].s] + e[j].l)
                d[e[j].t] = d[e[j].s] + e[j].l, cc = true;
        }
        if (!cc) break;
    }
    for (i = 0; i < m; ++i) {
        if (d[e[i].t] > d[e[i].s] + e[i].l) return false;
    }
    return true;
}
```

【spfa】[说明] bellman加速版用queue加速就是spfa ,还有就是二分的思想

```
#define N 1009
#define M 5009
#define inf 1e9

struct node {
    int t;
    double time;
    node *next;
} g[M];
int cnt = 0;
node *bian[N];
double fun[N];

void insert(int a, int b, double time) {

    g[cnt].t = b;
```

```
    g[cnt].time = time;
    g[cnt].next = bian[a];
    bian[a] = &g[cnt];
    ++cnt;
}
int n, m;
double dis[N];
int out[N];
queue<int> que;

int bellman(double k)// 顶点的标号是从到1n 求最小距离
{
    while (!que.empty()) que.pop();
    memset(out, 0, sizeof (out));
    int i;
    dis[1] = 0;
    for (i = 2; i <= n; ++i) dis[i] = inf;
    que.push(1);
    int u, v;
    node *p;
    while (!que.empty()) {
        u = que.front();
        que.pop();
        out[u]++;
        if (out[u] > n)//判断有没有负环
            return 1;
        p = bian[u];
        while (p) {
            v = p->t;
            if (dis[v] > dis[u] + k * p->time - fun[v]) {
                dis[v] = dis[u] + k * p->time - fun[v];
                que.push(v);
            }
            p = p->next;
        }
    }
    return 0;
}
```

【Karp算法求有向图最优比率圈， $O(nm)$**】**注意s可以到达图所有点，如果不存在这样的点可以增加一个虚拟点，它到其他点的距离都是0, 如果success 返回false 表示找不到圈

```
double Karp(int n, int m, int s,
eagle e[], bool &success) {
  int i, j, k;
  double mmin = Inf, mmax;
  for (i = 0; i <= n; ++i)
    for (j = 0; j < n; ++j) d[i][j] = Inf;
  for (d[0][s] = 0, i = 0; i < n; ++i)
    for (j = 0; j < m; ++j)
      if (d[i][e[j].a] < Inf &&
          d[i + 1][e[j].b] > d[i][e[j].a] + e[j].l)
        d[i + 1][e[j].b] = d[i][e[j].a] + e[j].l;
```

```
  for (i = 0; i < n; ++i)
    if (d[n][i] < Inf) {
      for (mmax = -Inf, j = 0; j < n; ++j) {
        double tmp = double(d[n][i] - d[j][i]) / (n - j);
        if (mmax < tmp) mmax = tmp;
      }
      if (mmin > mmax) mmin = mmax;
    }
  if (mmin >= Inf - eps) success = false;else success = true;
  return mmin;
}
```

【第k短路】按边存图，n为节点数，m为边数，s 起点，t 终点,findkmin返回第k短路的长度，注意这个是允许圈和重复边的

```
int n, m; struct edge{ //存边的起点终点v1,v2 权v
  int v1,v2,v;
} e[M];

struct node{
  int v,len;
}; std::queue<struct node> Q;

int top[N]; std::vector<int> heap[N]; int F[N];

bool cmp(const edge &a, const edge &b) {
  return a.v1 < b.v1;
}

void Insert(int num, int x) {
  heap[num].push_back(x);
  std::push_heap(heap[num].begin(), heap[num].end());
} int findkmin(int s, int t, int k, int n, int m) {
  int i, qt = 0, qh = 1;
  struct node x, tmp;
  std::sort(e, e + m, cmp);
  memset(F, -1, sizeof(F));
  F[e[0].v1] = 0;
  for (i = 1; i < m; ++i) if(e[i].v1 != e[i - 1].v1) F[e[i].v1] = i;
  tmp.v = s, tmp.len = 0;
  Q.push(tmp);
  for(; !Q.empty();) {
    x = Q.front();
    Q.pop();
    for (i = F[x.v];i != -1 && i < m && e[i].v1 == x.v; ++i) {
      tmp.v = e[i].v2;
      tmp.len = x.len + e[i].v;
      if (heap[tmp.v].size() < k) {
        Insert(tmp.v, tmp.len);
        Q.push(tmp);
      } else if (tmp.len < heap[tmp.v][0]) {
              std::pop_heap(heap[tmp.v].begin(), heap[tmp.v].end());
              *(heap[tmp.v].rbegin()) = tmp.len;
              std::push_heap(heap[tmp.v].begin(), heap[tmp.v].end());
```

```
                Q.push(tmp);
              }
    }
  }
  std::sort_heap(heap[t].begin(), heap[t].end());
  if (heap[t].size() < k) return -1;
  return heap[t][k-1];
}
```

【线性差分约束系统】差分约束系统是指形如$\boldsymbol{Ax} \leq \boldsymbol{b}$的一个不等式组，其中$\boldsymbol{A}$的每行恰有一个1和一个$-1$，其它元素都是0。这样的系统可以转化为图论模型，可以很巧妙的解决一些很难的题目。首先，构造一个有$n+1$个顶点的有向图G，顶点编号为$0 \cdots n$。若有不等式$x_i + C \leq x_j$，则添加一条从j指向i的有向边，权为C。若有不等式$x_i + C < x_j$，则添加一条从j指向i的有向边，权为$C+1$。若有不等式$x_i + C \geq x_j$，则添加一条从i指向j的有向边，权为C。若有不等式$x_i + C > x_j$，则添加一条从i指向j的有向边，权为$C-1$。另作从0到其他顶点的有向边，权为0，表示$x_i \leq 0$。问题转化为以顶点0为源点的最短路或拓扑排序问题。

5.6 连通性

【块的定义】没有割点的无向图称为块。把每块缩成一个点，得到一棵树，它得边是桥。从任意一个块经任意一条简单路走到另一个块所经过得桥的集合是相同的。

【求割点和桥】

```
int c[MAX];           //遍历标志,c[i]=0未遍历,1已遍历未检查,2已遍历已检查
int depth[MAX];       //深度
int graph[MAX][MAX]; //邻接矩阵
int Cut[MAX];         //Cut[i]==1 割点
int Brige[MAX][MAX]; //Brige[i][j]==1 (i,j)为桥
int Root;             //根节点
int Ancestor[MAX];//Ancestor[k]为k及k的子孙相连的辈分最高的祖先

void findnode(int k,int kfather,int deep)
{
  int i,tot; //tot为顶点k的儿子数量
  c[k]=1;depth[k]=deep;
  Ancestor[k]=deep;tot=0;
  for (i=0;i<n;i++) {
    if (graph[i][k]&&i!=kfather&&c[i]==1)
      Ancestor[k]=min(Ancestor[k],depth[i]);
    if (graph[i][k] && c[i]==0) {
      findnode(i,k,deep+1);
      tot++;
      Ancestor[k]=min(Ancestor[k],Ancestor[i]);
      //求割点
      if ((k==Root&&tot>1)||(k!=Root&&Ancestor[i]>=depth[k])) Cut[k]=1;
      //求桥的
      if (Ancestor[i]>depth[k]) Brige[k][i]=1;
    }
  }
  c[k]=2;
}
```

【有向图极大连通子图】邻接表形式，$mark[i]$相同的在同一个连通子图中，有些题目可以根据这个把一个连通子图缩成一个点，使图变成有向无环图。$color$从0开始染色。

```
//Gabow算法dfsn[v] == -1表示v未被访问过
vector<int> graph[N];
int sstack[N], pathstack[N], dfsn[N] ;
int ptop , cnt , stop ;
int mark[N];
int color;

void init() {
    memset(dfsn,-1,sizeof(dfsn));
    memset(mark,-1,sizeof(mark));
    ptop=cnt=stop=color=0;
}

void Gabow(int w) {
    int v, i;
    dfsn[w] = cnt++;
    sstack[stop++] = pathstack[ptop++] = w;
    for (i = 0; i < graph[w].size(); ++i) {
        v = graph[w][i];
        if (dfsn[v] == -1) Gabow(v);
        else if (mark[v] == -1)
            while (dfsn[pathstack[ptop - 1]] > dfsn[v]) ptop--;
    }
    if (pathstack[ptop - 1] == w) ptop--;
    else return;
    do { mark[v = sstack[--stop]] = color; } while (v != w);
    color++;
}

//Kosaraju算法, 通过2次DFS实现
//conn[]图的邻接表，rconn[]图的有向边的反向边邻接表
vector<int> conn[MAXN], rconn[MAXN];
bool visited[MAXN];
int sta[MAXN], top, mark[MAXN], color;

void dfs(int idx) {
    visited[idx] = true;
    for (int i = 0; i < conn[idx].size(); ++i)
        if (!visited[conn[idx][i]])
            dfs(conn[idx][i]);
    sta[top++] = idx;
}

void rdfs(int idx) {
    visited[idx] = true;
    mark[idx] = color;
    for (int i = 0; i < rconn[idx].size(); ++i)
        if (!visited[rconn[idx][i]])
            rdfs(rconn[idx][i]);
```

```
}
//n节点数，从0开始标号
void Kosaraju(int n) {
    int i;
    memset(visited, 0, sizeof (bool) * n);
    for (i = top = 0; i < n; ++i) if (!visited[i]) dfs(i);
    memset(visited, 0, sizeof (bool) * n);
    for (i = top - 1, color = 0; i >= 0; --i)
        if (!visited[sta[i]]) rdfs(sta[i]), ++color;
}
```

【无向图的最小边割】

```
const int Inf = 0x3fffffff;
int minCut(int n, int graph[][N]) {
  int h[N], vertex[N], d[N];
  for (int i = 0; i < n; ++i) vertex[i] = i;
  int last_i, best_j, v, top, result = Inf;
  while (n > 1) {
    d[last_i = vertex[0]] = -1;
    for(int i = 1; i < n; i++)
      d[vertex[h[i] = i]] = graph[vertex[i]][vertex[0]];
    top = h[0] = n - 1;
    for(int i = 1; i < n; ++i){
      best_j = 0;
      for (int j = 1; j < top; ++j)
        if (d[vertex[h[j]]] > d[vertex[h[best_j]]]) best_j = j;
      v = h[best_j];
      h[best_j] = h[--top];
      if (i == n - 1) {
        if(d[vertex[v]] < result) result = d[vertex[v]];
        for(int j = 0; j < n; ++j)
          graph[vertex[j]][last_i] += graph[vertex[v]][vertex[j]],
          graph[last_i][vertex[j]] = graph[vertex[j]][last_i];

        vertex[v] = vertex[--n];
        break;
      }
      last_i = vertex[v];
      for(int j = 0; j < top; ++j)
        d[vertex[h[j]]] += graph[vertex[h[j]]][last_i];
    }
  }
  return result;
}
```

【最佳(小)割边集】复杂度$O(E*MAX_FLOW)$,E表示边数,MAX_FLOW表示求最大流复杂度

[输入]点数n,源点s,汇点t,边数$edge_num$,边集合$edge[][3]$

[输出]返回割边集中边数,最佳割边集$edge_cut[]$,,最小割$mincost$,$edge[edge_cut[k]]$即为割边集中的边

[说明] $edge[][0]$到$edge[][1]$有一条权值为$edge[][2]$的边如,如果要求最小割边集，令$edge[][2] = 1$即可

```
int best_edge_cut(int n, int s, int t, int edge_num,
                  int edge[][3], int edge_cut[], int& mincost) {
    if (s == t) return -1;
    int  m0[N][N], m[N][N], i, j, k, l, ret = 0, last;
    #define FOR(i, j, n) for(i = 0;i < n;++ i)for(j = 0;j < n;++ j)
    //边集转化成邻接矩阵
    FOR(i, j, n) m0[i][j] = 0;
    for(k = 0;k < edge_num;++ k)
        m0[edge[k][0]][edge[k][1]] += edge[k][2];

    FOR(i, j, n) m[i][j] = m0[i][j];
    mincost = last = max_flow(n, m, s, t);

    //检查是否有最大流通过单边
    for (k = 0; k < edge_num && last; ++k) {
        if (edge[k][2] != last) continue;
        FOR(i, j, n) m[i][j] = m0[i][j];
        m[edge[k][0]][edge[k][1]] -= edge[k][2];
        if (max_flow(n, m, s, t) == 0) {
            edge_cut[0] = k;
            return 1;
        }
    }
    //最大流通过多边
    for (k = 0; k < edge_num && last; ++k) {
        FOR(i, j, n) m[i][j] = m0[i][j];
        m[edge[k][0]][edge[k][1]] -= edge[k][2];
        if (max_flow(n, m, s, t) == last - edge[k][2]) {
            edge_cut[ret++] = k;
            m0[edge[k][0]][edge[k][1]] -= edge[k][2];
            last -= edge[k][2];
        }
    }
    return ret;
}
```

【最佳(小)割点集】复杂度$O(n*MAX_FLOW)$,其中n为点数，MAX_FLOW表示求最大流复杂度
[输入]点数n,邻接阵$ma[][]$,点权值$cost[]$,源点s,汇点t,
[输出]最佳(小)割点集vex_cut,最小割$mincost$
[说明] 如果要求最小割点集$cost[]$值为1即可

```
int best_vertex_cut(int n, int mat[][MAXN], int* cost,
                    int s, int t, int* vex_cut, int& mincost) {
    if (s == t || mat[s][t]) return -1;
    int m0[MAXN][MAXN], m[MAXN][MAXN], i, j, k, ret = 0, last;
    #define FOR(i, j, n)  for (i = 0; i < (n); i++)for (j = 0; j < (n); j++)
    FOR(i, j, n + n) m0[i][j] = 0;
    FOR(i, j, n)  if (mat[i][j]) m0[i][n + j] = inf;
    for (i = 0; i < n; i++)
        m0[n + i][i] = cost[i];
    FOR(i, j, n + n) m[i][j] = m0[i][j];
    mincost = last = max_flow(n + n, m, s, n + t);
```

```
    for (k = 0; k < n && last; k++)
        if (k != s && k != t) {
            FOR(i, j, n + n) m[i][j] = m0[i][j];
            m[n + k][k] = 0;
            if (max_flow(n + n, m, s, n + t) == last - cost[k]) {
                vex_cut[ret++] = k;
                m0[n + k][k] = 0;
                last -= cost[k];
            }
        }
    return ret;
}
```

算法描述:

1. 构造一个网络 N
   若G为无向图;

   (a) 原G图中的每个顶点V变成N网中的两个顶点V'和V''，顶点V'至V''有一条弧连接，弧容量为1;

   (b) 原G图中的每条边$e = UV$，在N网中有两条弧$e' = U''V', e'' = V''U'$与之对应，e'弧容量为∞, e''弧容量为∞;

   (c) A''为源顶点，B'为汇顶点

   若G为有向图:

   (a) 原G团中的每个顶点V变成N网中的两个顶点V'和V''.顶点V'至V'有连接，弧容量为1;

   (b) 原G固中的每条弧$e = UV$变成一条有向轨$U'U''V'V''$，其中轨上的弧$U''V'$的容量为∞;

   (c) A''为源顶点，B'为汇顶点

2. 求A''到B'的最大流F

3. 流出A''一切弧的流量和$\sum F(e)$，即$P(A, B)$。所有具有流量1的弧(V', V'')对应的V顶点组成一个割顶集，在G固中去掉这些顶点则G图变成不连通。

有了求 $P(A, B)$ 的算法基础．我们不难得出 $k(G)$ 的求解思路：首先设$k(G)$的初始值为∞，然后分析图的每一对顶点。如果顶点A, B不相邻，则用求最大流的方法得出$P(A, B)$和对应的割顶集。若$P(A, B)$为目前最小，则记当前的$k(G)$值为$P(A, B)$，保存其割顶集。直至所有不相邻的顶点对分析完为止，就可得出图的顶连通度$k(G)$和最小割顶集了。

5.7 网络流

【最大流】输入图容量矩阵graph[MAXN][MAXN],取非零值表示有边，s为源点，t为汇点。
[说明] 复杂度上，V为点数，E为边数,C为边最大容量

```
const int Inf = 0x7fffffff;
const MAXN = 500;

//最近距离 O(VE^2)
int max_flow(int n, int graph[][MAXN], int s, int t) {
    int cap[MAXN], pre[MAXN], q[MAXN], vis[MAXN];
    bool mark[MAXN];
    int i, u, v, top, bot, flow = 0, visn = n;
    for(;;) {
        memset(mark, 0, sizeof(bool) * n);
        cap[s] = Inf, mark[s] = true, cap[t] = 0;
        for(q[bot = top = 0] = s;bot <= top;) {
            for(u = q[bot++], v = 0;v < n;++ v)
                if(v != u && !mark[v] && graph[u][v] > 0) {
                    q[++top] = v, pre[v] = u, mark[v] = true;
                    cap[v] = min(cap[u], graph[u][v]);
                    if(v == t) break;
                }
            if(v == t) break;
        }
        if(cap[t] == 0) return flow;
        for(flow += cap[t], i = t;i != s;i = pre[i])
            graph[i][pre[i]] += cap[t], graph[pre[i]][i] -= cap[t];
    }
    return flow;
}

//最大容量 O(VE log(C))
int max_flow(int n, int graph[][MAXN], int s, int t) {
    int cap[MAXN], pre[MAXN], vis[MAXN];
    int i, best_i, u, visn, flow = 0 ;
    for(;;) {
        for(i = 0;i < n;++ i) vis[i] = i, cap[i] = 0;
        for(cap[s] = Inf, visn = n;;) {
            for(i = best_i = 0;i < visn;++ i)
                if(cap[vis[i]] > cap[vis[best_i]]) best_i = i;
            u = vis[best_i]; vis[best_i] = vis[--visn];
            if(cap[u] == 0) return flow;
            if(u == t) break;
            for(i = 0;i < visn;++ i)
                if(cap[vis[i]] < (best_i = min(cap[u], graph[u][vis[i]])))
                    cap[vis[i]] = best_i, pre[vis[i]] = u;
        }
        if(cap[t] == 0) return flow;
        for(flow += cap[t], i = t;i != s;i = pre[i])
            graph[i][pre[i]] += cap[t], graph[pre[i]][i] -= cap[t];
    }
    return flow;
}

//带标号最近距离 O(V^2E)
int max_flow(int n, int graph[][MAXN], int s, int t) {
    int d[MAXN], q[MAXN], pre[MAXN];
```

```
    bool mark[MAXN];
    int top, bot, u, v, flow = 0;
    memset(mark, false, sizeof(mark[0]) * n);
    mark[t] = true, q[top = bot = 0] = t, d[t] = 0;
    while(top <= bot)
        for(u = q[top++], v = 0;v < n;++ v)
            if(u != v && !mark[v] && graph[v][u] > 0)
                d[v] = d[u] + 1, q[++bot] = v, mark[v] = true;
    for(pre[u = s] = s, q[s] = Inf;d[s] < n;) {
          for(v = 0;v < n && (d[u] != d[v] + 1 || graph[u][v] <= 0);++ v);
          if(v < n)  q[v] = min(graph[u][v] , q[u]), pre[v] = u, u = v;
          else  {
              for(bot = n, v = 0;v < n;++ v)
                  if(bot > d[v] && graph[u][v] > 0) bot = d[v];
              d[u] = bot + 1, u = pre[u];
          }
          if(u == t) for(flow += q[t];u != s;u = pre[u])
              graph[pre[u]][u] -= q[t], graph[u][pre[u]] += q[t];
    }
    return flow;
}
```

//最高标号预流推进n^3

```
int max_flow(int n, int graph[][N], int s, int t) {
  int q[N], pre[N], e[N] = {0}, d[N], top, bot, u, v, x, push_flow;;
  bool mark[N] = {0};
  for (int i = 0; i < n; ++i) d[i] = n;
  mark[t] = true, q[top = bot = 0] = t, d[t] = 0;
  for (; top <= bot; ++top) for (u = q[top], v = 0; v < n; ++v)
      if (u != v && graph[v][u] > 0 && !mark[v])
        mark[q[++bot] = v] = true, d[v] = d[u] + 1;
  memset(mark, false, sizeof(mark));
  e[s] = Inf; mark[s] = true;
  for (top = 1, q[0] = s;top != 0;)  {
    for (x = 0, v = 1; v < top; ++v) if (d[q[v]] > d[q[x]]) x = v;
    do {
    for (u = q[x], v = 0; v < n && e[u] != 0; ++v) {
      if (d[u] == d[v] + 1 && graph[u][v] > 0) {
        if (graph[u][v] < e[u]) push_flow = graph[u][v];
          else push_flow = e[u];
        e[v] += push_flow, e[u] -= push_flow,
        graph[u][v] -= push_flow, graph[v][u] += push_flow;
        if (!mark[v] && e[v] > 0 && d[v] < n && d[v] > 0)
          mark[v] = true, q[top++] = v;
      }
    }
    if (e[u] != 0) {
      for (bot = n, v = 0; v < n; ++v)
        if (bot > d[v] && graph[u][v] > 0) bot = d[v];
      if ((d[u] = bot + 1) >= n ) q[x] = q[--top], mark[u] = false;
    } else q[x] = q[--top], mark[u] = false;
   } while (e[u] != 0 && d[u] <n);
```

```
  }
  return e[t];
}



//邻接表结合邻接矩阵
struct edge {
    int v;
    edge * next;
} edges[2 * maxn*maxn], *net_g[maxn];
int edgen;
int net[maxn][maxn];
int level[maxn];

void graphinit(int n) {
    memset(net_g, 0, sizeof (net_g[0]) * n);
    edgen = -1;
    memset(net, 0, sizeof (net[0][0]) * maxn * n);
}

void addedge(int a, int b, int c, edge * net_g[], int net[][maxn]) {
    if (net[a][b] == 0) {
        edges[++edgen].v = b;
        edges[edgen].next = net_g[a];
        net_g[a] = & edges[edgen];

        edges[++edgen].v = a;
        edges[edgen].next = net_g[b];
        net_g[b] = & edges[edgen];
    }
    net[a][b] += c;
}

void bfs(int s, int n, edge * graph[], int net[][maxn]) {
    int front, rear, tmp, q[maxn];
    edge * p;
    memset(level, -1, sizeof (level[0]) * n);
    for (level[s] = 0, q[front = rear = 0] = s; front <= rear; ++front)
        for (tmp = q[front], p = graph[tmp]; p; p = p->next)
            if (net[tmp][p->v] && level[p->v] == -1)
                level[p->v] = level[tmp] + 1, q[++rear] = p->v;
}

int dinic(int n, int s, int t, edge * graph[], int net[][maxn]) {
    int i, k, st[maxn], top, cur, maxflow = 0;
    edge * p;
    while (1) {
        bfs(s, n, graph, net);
        if (level[t] == -1) break;
        st[top = 0] = cur = s;
        while (1) {
            if (cur == t) {
```

```
                int minc = 0x7fffffff;
                for (i = 0; i < top; ++i)
                    if (minc > net[st[i]][st[i + 1]])
                        minc = net[st[i]][st[i + 1]], k = i;
                for (i = 0; i < top; ++i)
                    net[st[i]][st[i + 1]] -= minc, net[st[i + 1]][st[i]] += minc;
                maxflow += minc, cur = st[top = k];
                continue;
            }
            for (p = graph[cur]; p; p = p->next)
                if (net[cur][p->v] && level[cur] + 1 == level[p->v])
                    break;
            if (p)
                cur = st[++top] = p->v;
            else {
                if (top == 0) break;
                level[cur] = -1, cur = st[--top];
            }
        }
    }
    return maxflow;
}

//邻接表版无法判断重边，用于实现顶点很多的情况
#define N 20009

struct node {
    int t;
    int flow;
    int cap;
    int index;
};
vector<node> g[N];
vector<int> e[N];
int rank[N];

void init(int n) {
    memset(rank, -1, sizeof (rank));
    int i;
    for (i = 0; i < n; ++i) {
        g[i].clear();
        e[i].clear();
    }
}

void addedge(int a, int b, int c) {
    node temp;
    temp.t = b;
    temp.cap = c;
    temp.flow = 0;
    temp.index = g[b].size();
    g[a].push_back(temp);
```

```
    temp.t = a;
    temp.cap = 0;
    temp.index = g[a].size() - 1;
    g[b].push_back(temp);
}

void addedge1(int a, int b, int c) {
    node temp;
    temp.t = b;
    temp.cap = c;
    temp.flow = 0;
    temp.index = g[b].size();
    g[a].push_back(temp);

    temp.t = a;
    temp.index = g[a].size() - 1;
    g[b].push_back(temp);
}

int bfs(int s, int t, int n) {
    memset(rank, -1, sizeof (rank));
    int i;
    for (i = 0; i < n; ++i)
        e[i].clear();
    int que[N];
    int u, v;
    int front = 0, rear = 0;
    rank[s] = 0;
    rear++;
    que[front] = s;
    for (; rear > front;) {
        u = que[front++];
        for (i = 0; i < g[u].size(); ++i) {
            v = g[u][i].t;
            if (rank[v] == -1 && g[u][i].cap) {
                que[rear++] = v;
                rank[v] = rank[u] + 1;
            }
            if (rank[v] == rank[u] + 1)
                e[u].push_back(i);
        }
    }
    return (rank[t] != -1);
}

int dinic(int s, int t, int n) {
    int st[N], maxflow = 0;
    int aux[N];
    int top, cur;
    int p, i, k;
    while (1) {
        if (bfs(s, t, n) == 0) break;
        top = 0;
```

```
        st[top] = s;
        cur = s;
        while (1) {
            if (cur == t) {
                int minc = 0x7fffffff;
                for (i = 0; i < top; ++i) {
                    p = aux[i + 1];
                    if (minc > g[st[i]][p].cap)
                        minc = g[st[i]][p].cap, k = i;
                }
                for (i = 0; i < top; ++i) {
                    p = aux[i + 1];
                    g[st[i]][p].cap -= minc;
                    g[g[st[i]][p].t][g[st[i]][p].index].cap += minc;

                }
                maxflow += minc;
                cur = st[top = k];
                //  continue;
            }
            int len = e[cur].size();
            while (len) {
                p = e[cur][len - 1];
                if (g[cur][p].cap && rank[g[cur][p].t] == rank[cur] + 1)
                    break;
                else {
                    len--;
                    e[cur].pop_back();
                }
            }
            if (len) {
                cur = st[++top] = g[cur][p].t;
                aux[top] = p;
            } else {
                if (top == 0) break;
                rank[cur] = -1;
                cur = st[--top];
            }
        }
    }
    return maxflow;
}

//能够求每条边流量的最大流
int _max_flow(int n, int mat[][MAXN], int s, int t, int flow[][MAXN]) {
    int pre[MAXN], que[MAXN], d[MAXN], p, q, t, i, j;
    if (s == t) return Inf;
    for (i = 0; i < n; i++)
        for (j = 0; j < n; flow[i][j++] = 0);
    for (;;) {
        for (i = 0; i < n; pre[i++] = 0);
        pre[t = s] = s + 1, d[t] = inf;
        for (p = q = 0; p <= q && !pre[t]; t = que[p++])
```

```
            for (i = 0; i < n; i++)
                if (!pre[i] && j = mat[t][i] - flow[t][i])
                    pre[que[q++] = i] = t + 1, d[i] = d[t] < j ? d[t] : j;
                else if (!pre[i] && j = flow[i][t])
                    pre[que[q++] = i] = -t - 1, d[i] = d[t] < j ? d[t] : j;
        if (!pre[t]) break;
        for (i = t; i != s;)
            if (pre[i] > 0) flow[pre[i] - 1][i] += d[t], i = pre[i] - 1;
            else flow[i][-pre[i] - 1] -= d[t], i = -pre[i] - 1;
    }
    for (j = i = 0; i < n; j += flow[s][i++]);
    return j;
}
```

【无源汇网络容量有上下界的可行流】

[输入]网络节点数n,容量mat,流量下界bf
[输出]返回最大(小)流量,-1表示无可行流,flow返回每条边的流量
[注意]MAXN应比最大结点数多2,无可行流返回-1时mat未复原!

```
int limit_flow(int n, int mat[][MAXN], int bf[][MAXN], int flow[][MAXN]) {
    int i, j, sk, ks;
    mat[n][n + 1] = mat[n + 1][n] = mat[n][n] = mat[n + 1][n + 1] = 0;
    for (i = 0; i < n;i++)
        for (mat[n][i] = mat[i][n] = mat[n + 1][i] = mat[i][n + 1] = j = 0; j < n; j++)
            mat[i][j] -= bf[i][j], mat[n][i] += bf[j][i], mat[i][n + 1] += bf[i][j];
    for (i = 0; i < n + 2; i++)
        for (j = 0; j < n + 2; flow[i][j++] = 0) ;
    //最大流
    _max_flow(n + 2, mat, n, n + 1, flow);
    //最小流
    _max_flow(n + 2, mat, n + 1, n, flow);
    for (i = 0; i < n; i++)
        if (flow[n][i] < mat[n][i]) return -1;
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            mat[i][j] += bf[i][j], flow[i][j] += bf[i][j];
    for (j = i = 0; i < n; j += flow[n][i++]);
    return j;
}
```

【有源汇容量有上下界的可行流】

[输入]网络节点数n,容量mat,流量下界bf,源点s,汇点t
[输出]返回最大(小)流量,-1表示无可行流,flow返回每条边的流量
[注意]MAXN应比最大结点数多2,无可行流返回-1时mat未复原!

```
int limit_flow(int n, int mat[][MAXN], int bf[][MAXN], int s, int t, int flow[][MAXN]) {
    int i, j, sk, ks;
    if (s == t) return inf;
    mat[n][n + 1] = mat[n + 1][n] = mat[n][n] = mat[n + 1][n + 1] = 0;
    for (i = 0; i < n; i++)
        for (mat[n][i] = mat[i][n] = mat[n + 1][i] = mat[i][n + 1] = j = 0; j < n; j++)
            mat[i][j] -= bf[i][j], mat[n][i] += bf[j][i], mat[i][n + 1] += bf[i][j];
    sk = mat[s][t], ks = mat[t][s], mat[s][t] = mat[t][s] = inf;
```

```
    for (i = 0; i < n + 2; i++)
        for (j = 0; j < n + 2; flow[i][j++] = 0);
    _max_flow(n + 2, mat, n, n + 1, flow);
    for (i = 0; i < n; i++)
        if (flow[n][i] < mat[n][i]) return -1;
    flow[s][t] = flow[t][s] = 0, mat[s][t] = sk, mat[t][s] = ks;
    //最大流
    max_flow(n, mat, s, t, flow);
    //最小流
    _max_flow(n, mat, t, s, flow);
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            mat[i][j] += bf[i][j], flow[i][j] += bf[i][j];
    for (j = i = 0; i < n; j += flow[s][i++]);
    return j;
}
```

算法描述:

1. 建立附加网络
   (a) 新增两个顶点附加源s'和附加汇t'
   (b) 将原网络每个顶点u与t'相连，容量为指向该顶点的边容量下限之和
   (c) 将s'与原网络每个顶点u，容量为由u出发的边的容量下限之和
   (d) 原网络每条边容量修正为上界- 下界
   (e) s,t相连，t,s相连，容量无限大。
2. 对附加网络求最大流。如果能使所有从s'流出的边流量达到容量，则原网络可行流=附加网络流+ 下界；否则无解。
3. 用最大流算法改进原网络可行流。

【最小费用最大流】Edmonds Karp对偶算法,复杂度 $O(V^3E^2)$。
[输入]容量矩阵mat[][],非零边表示有边,费用矩阵cost[][],s为源点，t为汇点
[输出]返回最大流, netcost返回最小费用，flow[][]返回流量矩阵

```
int DualityEdmondsKarp(int n, int mat[][MAXN], int cost[][MAXN],
                       int s, int t, int flow[][MAXN], int &netcost) {
    int i, j, k, c, quit, max_flow = 0;
    int best[MAXN], prev[MAXN];
    for (i = 0; i < n; i++)
        for (j = 0; j < n; flow[i][j++] = 0)
            if (mat[i][j]) mat[j][i] = 0, cost[j][i] = -cost[i][j];
    for(netcost = 0;;max_flow += c, netcost += c * best[t]) {
        for (i = 0; i < n; best[i++] = Inf);
        for (best[s] = quit = 0;!quit;)
            for (quit = 1, i = 0; i < n; i++)
                if (best[i] < Inf)
                    for (j = 0; j < n; j++)
                        if (flow[i][j] < mat[i][j] && best[i] + cost[i][j] < best[j]) {
                            best[j] = best[i] + cost[i][j];
```

```
                            prev[j] = i;
                            quit = 0;
                        }
        if (best[t] >= Inf) break;
        for(c = Inf, j = t;j != s;j = i) {
            i = prev[j];
            if (c > mat[i][j] - flow[i][j]) c = mat[i][j] - flow[i][j];
        }
        for(j = t;j != s;j = i) {
            i = prev[j];
            flow[i][j] += c;
            flow[j][i] = -flow[i][j];
        }
    }
    return max_flow;
}
```

【判断给定流是否最小费用最大流】Klein圈迭加算法。找残留网络r[n][n]中负圈(Floyd算法)，如果没有说明是最小费用；如果存在负圈，则改进该圈的流量，可以增流。

【最优比率割】输入图容量矩阵g[n][n],取非零值表示有边，s为源点，t为汇点。

```
const double eps = 1e-8;
int n,m;
double g[N][N];
struct st {
  int a,b;
}Line[M];

double findmin(double x) {
  double c[N][N];
  bool mark[N], h[M];
  double t=0;
  int tn=0;
  memset(h,false,sizeof(h));
  memset(c,0,sizeof(c));
  for ( int i=0;i<m;++i) {
    int a=Line[i].a,b=Line[i].b;
    if ( g[a][b]>=x+eps )
      c[a][b]=c[b][a]=g[a][b]-x;
      else t+=g[a][b],tn++,h[i]=true;
  }
  //mark为被最小割分成的2部分中的一个, mark[a] == mark[b] a,b在最小割分割后的其中的一个
  network_max_flow(n, c, 0,n-1,mark);
  for ( int i=0;i<m;++i) if (!h[i]) {
    int a=Line[i].a,b=Line[i].b;
    if ( mark[a]!=mark[b] ) t+=g[a][b],tn++;
  }

  if ( tn==0 ) return 1e10;
  return t/tn;

}
```

【求最小割】基于增广路的最大流算法最后一次的增广（失败的那次）可以确定最小割。已经被访问过的点（不一定是最后队列里的点）属于A^*，其他点属于A^{*-}。

5.8 二分图匹配

【二分图最大匹配】Hungary算法, 邻接阵g[n][m], 复杂度 $O(nm)$, 二分图大小n,m, g[i][j]==1表示i,j有边, 函数match()返回最大匹配数, 匹配结果为(mat[i],i), 未匹配顶点mat[i]为-1.

```
int mat[M],tmat[M];
int n,m;
int hungry_aug(int i) {
    int v,j;
    for (j = 0 ; j < m; j++)
      if ( g[i][j] && tmat[j]==0) {
        tmat[j]=1;v=mat[j];mat[j]=i;
        if (v==-1 || hungry_aug(v)) return 1;
            mat[j]=v;
      }
 return 0;
}

int hungry() {
    int i,k=0;
    memset(mat,-1,sizeof(mat));
    for (i = 0; i < n; i++) {
        memset(tmat,0,sizeof(tmat));
        k+=hungry_aug(i);
    }
    return k;
}
```

【带权二分图最大匹配KM算法】邻接阵graph[n][m], 返回带权最大匹配数, mat[i]表示匹配边graph[mat[i]][i]

```
int graph[N][N], mat[N], lx[N], ly[N];
int x[N], y[N];
int n, m;

bool KM_aug(int v) {
  int i;
  for (x[v] = true, i = 0; i < m; ++i)
    if (!y[i] && lx[v] + ly[i] == graph[v][i]) {
      y[i] = true;
      if (mat[i] == -1 || KM_aug(mat[i]))
       { mat[i] = v; return true; }
    }
  return false;
}

int KM() {
  int i, j, k, d, nn, mm, result = 0;
  memset(mat, -1, sizeof(mat));
```

```
  memset(ly, 0, sizeof(ly));
  if (n > m) nn = n, mm = m; else nn = m, mm = n;
  for (i = 0; i < n; ++i) {
    lx[i] = graph[i][0];
    for (j = 1; j < m; ++j)
      if (graph[i][j] > lx[i]) lx[i] = graph[i][j];
  }
  for (k = 0; k < nn; ++k) {
    while (true) {
      memset(x, false, sizeof(x));
      memset(y, false, sizeof(y));
      if (KM_aug(k)) break;
      for (d = Inf, i = 0; i < n; ++i) if (x[i])
          for (j = 0; j < m; ++j) if (!y[j])
              if (lx[i] + ly[j] - graph[i][j] < d)
                d = lx[i] + ly[j] - graph[i][j];
      for (i = 0; i < n; ++i) if (x[i]) lx[i] -= d;
      for (j = 0; j < m; ++j) if (y[j]) ly[j] += d;
    }
  }
  for (i = 0;i < mm; ++i) if(mat[i] >= 0) result += graph[mat[i]][i];
  return result;
}
```

【二分图匹配Hopcroft-Karp算法O(sqrt(n)*e)】[注意]自己写build在里面给ns赋值才能init，ns为某一边点数

```
class Graph {
public:
   const static int N = 6000;     //最大边数
   bool build();
   int match();
   void init() {for(int i=0;i<ns;++i) g[i].clear();};
private:
   const static int INF = 1 << 28;
   vector<int> g[N];
   int ns, xmate[N], ymate[N], ds[N], dt[N], dis;
   bool vst[N];
   bool search();
   bool dfs(int);
};

bool Graph::build()
{
}

bool Graph::search() {
   memset(ds, -1, sizeof(ds)); memset(dt, -1, sizeof(dt));
   queue<int> Q; dis = INF;
   for(int i = 0; i < ns; i++)
      if(xmate[i] == -1) { Q.push(i); ds[i] = 0; }
   while(!Q.empty()) {
      int u = Q.front(); Q.pop();
      if(ds[u] > dis) break;
```

```
      for(int i = g[u].size()-1; i >= 0; i--) {
         int v = g[u][i];
         if(dt[v] != -1) continue;
         dt[v] = ds[u]+1;
         if(ymate[v] == -1) dis = dt[v];
         else { ds[ymate[v]] = dt[v]+1; Q.push(ymate[v]); }
      }
   }
   return (dis != INF);
}

bool Graph::dfs(int u) {
   for(int i = g[u].size()-1; i >= 0; i--) {
      int v = g[u][i];
      if(vst[v] || dt[v] != ds[u]+1) continue;
      int ut = ymate[v], vs = xmate[u];
      vst[v] = true;
      if(ut != -1 && dt[v] == dis) continue;
      ymate[v] = u; xmate[u] = v;
      if(ut == -1 || dfs(ut)) return true;
      ymate[v] = ut;  xmate[u] = vs;
   }
   return false;
}

int Graph::match() {
   int mth = 0;
   memset(xmate, -1, sizeof(xmate)); memset(ymate, -1, sizeof(ymate));
   while(search()) {
      memset(vst, false, sizeof(vst));
      for(int i = 0; i < ns; i++)
         if(ds[i] == 0 && dfs(i)) mth++;
   }
   return mth;
}
```

【一般图基数最大匹配花算法By mmd】

```
struct Graph {
    int n, match[maxn];
    bool adj[maxn][maxn];
    void clear() {
        memset(adj, 0, sizeof(adj));
        n = 0;
    }
    void insert(const int &u, const int &v) {
        get_max(n, max(u, v) + 1);
        adj[u][v] = adj[v][u] = 1;
    }
    int max_match() {
        memset(match, -1, sizeof(match));
        int ans = 0;
        for (int i = 0; i < n; ++i) {
            if (match[i] == -1) {
```

```
            ans += bfs(i);
        }
    }
    return ans;
}
int Q[maxn], pre[maxn], base[maxn];
bool hash[maxn];
bool in_blossom[maxn];
int bfs(int p) {
    memset(pre, -1, sizeof(pre));
    memset(hash, 0, sizeof(hash));
    for (int i = 0; i < n; ++i) {
        base[i] = i;
    }
    Q[0] = p;
    hash[p] = 1;
    for (int s = 0, t = 1; s < t; ++s) {
        int u = Q[s];
        for (int v = 0; v < n; ++v) {
            if (adj[u][v] && base[u] != base[v] && v != match[u]) {
                if (v == p || (match[v] != -1 && pre[match[v]] != -1)) {
                    int b = contract(u, v);
                    for (int i = 0; i < n; ++i) {
                        if (in_blossom[base[i]]) {
                            base[i] = b;
                            if (hash[i] == 0) {
                                hash[i] = 1;
                                Q[t++] = i;
                            }
                        }
                    }
                } else if (pre[v] == -1) {
                    pre[v] = u;
                    if (match[v] == -1) {
                        argument(v);
                        return 1;
                    } else {
                        Q[t++] = match[v];
                        hash[match[v]] = 1;
                    }
                }
            }
        }
    }
    return 0;
}
void argument(int u) {
    while (u != -1) {
        int v = pre[u];
        int k = match[v];
        match[u] = v;
        match[v] = u;
        u = k;
```

```
        }
    }
    void change_blossom(int b, int u) {
        while (base[u] != b) {
            int v = match[u];
            in_blossom[base[v]] = in_blossom[base[u]] = true;
            u = pre[v];
            if (base[u] != b) {
                pre[u] = v;
            }
        }
    }
    int contract(int u, int v) {
        memset(in_blossom, 0, sizeof(in_blossom));
        int b = find_base(base[u], base[v]);
        change_blossom(b, u);
        change_blossom(b, v);
        if (base[u] != b) {
            pre[u] = v;
        }
        if (base[v] != b) {
            pre[v] = u;
        }
        return b;
    }
    int find_base(int u, int v) {
        bool in_path[maxn] = {};
        while (true) {
            in_path[u] = true;
            if (match[u] == -1) {
                break;
            }
            u = base[pre[match[u]]];
        }
        while (!in_path[v]) {
            v = base[pre[match[v]]];
        }
        return v;
    }
};
```

【一般图基数最大匹配花算法】graph[N][N]为邻接矩阵，match[] 为匹配。graph_match 返回匹配数

```
int aug(int n,int mat[][MAXN],int match[],int v[],int now){
    int i,ret=0;
    v[now]=1;
    for (i=0;i<n;i++)
        if (!v[i]&&mat[now][i]){
            if (match[i]<0)
                match[now]=i,match[i]=now,ret=1;
            else{
                v[i]=1;
                if (aug(n,mat,match,v,match[i]))
```

```
                    match[now]=i,match[i]=now,ret=1;
                v[i]=0;
            }
            if (ret)
                break;
        }
    v[now]=0;
    return ret;
}

int graph_match(int n,int mat[][MAXN],int match[]){
    int v[MAXN],i,j;
    for (i=0;i<n;i++)
      v[i]=0,match[i]=-1;
    for (i=0,j=n;i<n&&j>=2;)
        if (match[i]<0&&aug(n,mat,match,v,i))
            i=0,j-=2;
        else
            i++;
    for (i=j=0;i<n;i++)
        j+=(match[i]>=0);
    return j/2;
```

【最大团算法】

返回最大团大小和一个方案,传入图的大小n和邻接阵mat, mat[i][j]为布尔量

```
#define MAXN 60
void clique(int n, int* u, int mat[][MAXN],
            int size, int& max, int& bb,
            int* res, int* rr, int* c) {
  int i, j, vn, v[MAXN];
  if (n) {
    if (size + c[u[0]] <= max) return;
    for (i = 0; i < n + size - max && i < n; ++ i) {
      for (j = i + 1, vn = 0; j < n; ++ j)
        if (mat[u[i]][u[j]])
          v[vn ++] = u[j];
      rr[size] = u[i];
      clique(vn, v, mat, size + 1, max, bb, res, rr, c);
      if (bb) return;
    }
  } else if (size > max) {
    max = size;
    for (i = 0; i < size; ++ i)
      res[i] = rr[i];
    bb = 1;
  }
}
int maxclique(int n, int mat[][MAXN], int *ret) {
  int max = 0, bb, c[MAXN], i, j;
  int vn, v[MAXN], rr[MAXN];
  for (c[i = n - 1] = 0; i >= 0; -- i) {
    for (vn = 0, j = i + 1; j < n; ++ j)
      if (mat[i][j])
```

```
            v[vn ++] = j;
        bb = 0;
        rr[0] = i;
        clique(vn, v, mat, 1, max, bb, ret, rr, c);
        c[i] = max;
      }
      return max;
}

//另外一个最大团
#define N 120
vector<int> color[N];
int map[N][N], visit[N], dp[N];
int Max, finded, n, k;

int find(int start, int flag[]) {
    int i;
    for (i = start; i < n; ++i)
        if (flag[i]) return i;
    return -1;
}

void dfs(int start, int visit[], int depth) {
    int flag[N], set[N];
    int first;
    memcpy(flag, visit, n * 4);
    memcpy(set, visit, n * 4);
    first = find(start, flag);
    if (first == -1) {
        if (depth > Max) {
            Max = depth;
            finded = 1;
        }
        return;
    }
    int i;
    while (first != -1) {
        //color[k].push_back(first放这里是错误的);
        if (depth + n - start <= Max) return;
        if (depth + dp[first] <= Max) return;
        color[k].push_back(first); //保存结点
        set[first] = 0;
        flag[first] = 0;
        for (i = first + 1; i < n; ++i) {
            if (flag[i] && map[first][i])
                set[i] = 1;
            else set[i] = 0;
        }
        dfs(first, set, depth + 1);
        if (finded) return;
        first = find(first, flag);
        color[k].pop_back(); //弹出结点
    }
```

```
}

//使用前先调用init()
void init() {
    Max = 0;
    for (i = n - 1; i >= 0; --i) {
        k = i;
        finded = 0;
        memcpy(visit, map[i], 4 * n);
        dfs(i, visit, 1);
        dp[i] = Max;
    }
}
//图G的最大独立集= G的补图（G'）的最大团值
//图G的最小顶点覆盖=顶点数（V）- G的补图（G'）的最大团值
//做补图的时候注意去掉顶点到自己的边
```

5.9 独立集与支配集

定义5.9.1 (最小支配集) 支配集$D \subset V$，满足$\forall v \in G$，或者$v \in D$，或者v与D中一个顶点相邻。最小支配集的顶点个数，称为支配数 $\gamma(G)$。

定义5.9.2 (最大独立集) 独立集$I \subset V$，I中任意两个顶点不相邻。最大独立集的顶点个数，称为独立数 $\beta(G)$。

定义5.9.3 (最小覆盖集) 覆盖$K \subset V$，且G的每一条边至少有一个端点属于K，即顶点覆盖全体边。最小覆盖的顶点个数，称为覆盖数 $\alpha(G)$。

定义5.9.4 (最小边覆盖) 边覆盖$U \subset E$，U覆盖所有顶点，即边覆盖全体顶点。
最小边覆盖=最大匹配+未盖点(任取一条与之相连的点作覆盖边) 。

定义5.9.5 (最大团) 完全子图$U \subset V$，满足U中任意两个顶点均相邻。完全子图U是团，当且仅当U不包含在G的更大完全子图中。最大团是G中包含顶点最多的团。

定义5.9.6 (最小路径覆盖) 有向图$G = (V, E)$的路径覆盖是一个其节点不相交的路径集合p，图中每个节点仅分别包含于p中的一条路径。路径可从任意节点开始和结束，且长度也为任意值，包括0。最小路径覆盖是包含路径数最少的覆盖。

【关系】

1. 一个独立集是极大独立集，当且仅当它是一个支配集
2. I是独立集，当且仅当$V - I$是覆盖集。即$\alpha(G) + \beta(G) = |V|$
3. U是$G = (V, E)$最大团，当且仅当U是$G = (V, \neg E)$最大独立集
4. 极大独立集必为极小支配集；但反之未必。
5. 二部图中，最大匹配=最小覆盖集($K\ddot{o}ing$定理)

6. 二部图中，最小边覆盖=最大独立集=n-最大匹配=未盖点+最大匹配

7. 有向无环图的最小路径覆盖。如果给定的问题是有向无环图，我们把图中的每个顶点拆成两个顶点x_i和y_i，若原图中点i到点j有边的话，则新图中在x_i和y_j间连一条边。可以看到，新图是一张二分图。由于题目保证了图中不存在圈，所以最小路径覆盖数=顶点数(原图)-最大匹配数。

5.10 注意事项

1. 要注意输入数据的节点标号是否也是从0开始的。

2. 要注意输入数据是否有重边存在。

3. 在不同情况下顶点不连通的取值约定是不同的。

4. 二分图匹配转化为最大流求解时要注意将原图中的每一条边改为相应的由X指向Y的有向边

5. 调用图论算法时注意顶点数n的值。

6. 一个点含有多种状态时，可以将点拆为多个。这是方程维数增加在几何上的体现。例如每个点只能访问一次，可将点拆成一进一出两个点，两点之间连容量为1的边。

7. 高维的动态规划问题可以转化为最小费用流来解决。

8. 最小割常用于求满足某种最小关系的顶点集

第六章 数论

6.1 欧几里德算法

```
BIGNUM gcd(BIGNUM m, BIGNUM n) {
    while (true) {
        if ((m = m % n) == 0) return n;
        if ((n = n % m) == 0) return m;
    }
}
```

【扩展欧几里德算法】gcd(a, b) = a*x+b*y
[输出]x,y，函数返回GCD(a, b)最大公约数

```
BIGNUM ext_gcd(BIGNUM a, BIGNUM b, BIGNUM &x, BIGNUM &y)
{
    BIGNUM t, d;
    if (b == 0) { x = 1; y = 0; return a; }
    d = ext_gcd(b, a % b, x, y);
    t = x;
    x = y;
    y = t - a / b*y;
    return d;
}
```

【乘法逆元】
$ax \equiv 1 \pmod{n},\ n > 0$
[注意]a与n必须互质
[输出]函数求a对n的乘法逆元，若不存在则返回-1

```
int Invmod(int a,int n){
    int x,y;
    if (ext_gcd(a,n,x,y)!=1)return -1;
    return (x%n+n)%n;
}
```

6.2 模线性方程

$ax \equiv b \pmod{n},\ n > 0$
[说明] 返回-1说明无解，否则返回的是最小正解, X[]存放的是所有可以的解

[注意]n必须大于0, a和b必须大于等于0

```
BIGNUM modequ(BIGNUM a, BIGNUM b, BIGNUM n, BIGNUM *X)
{
    BIGNUM e, i, d, x, y, step;
    d = ext_gcd(a, n, x, y);  //d解的个数
    if (b % d != 0) return -1;
    else {
        e = (x * (b / d)) % n;
        for (i = 0; i < d; i++) //可能x<0
            X[i] = (e + i * (n / d)) % n; //就是解X
        //最小正解
        step = n / d;
        e = (e - e / step * step + step) % step;
    }
    return e;
}
```

6.3 中国剩余定理

$$\begin{cases} a \equiv b_1 \pmod{w_1} \\ a \equiv b_2 \pmod{w_2} \\ \vdots \\ a \equiv b_n \pmod{w_n} \end{cases}$$

其中$\boldsymbol{w}, \boldsymbol{b}$已知,$w[i] > 0$且$w[i]$与$w[j]$互质,求$a$.解的范围$1..n, n = w[0] * w[1] * ... * w[k-1]$

```
int China(int b[], int w[], int k) {
    int i;
    int d, x, y, a = 0, m, n = 1;
    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;
    }
    if (a > 0) return a;
    else return (a + n);
}
```

6.4 模方程合并

$$\begin{cases} x \equiv c_1 \pmod{b_1} \\ x \equiv c_2 \pmod{b_2} \\ \vdots \\ x \equiv c_n \pmod{b_n} \end{cases}$$

其中$\boldsymbol{w}, \boldsymbol{b}$已知,$b[i] > 0$且$b[i]$与$b[j]$可以不互质,求$x$.解的范围$1..n, n = lcm(b[0], b[1], ..., b[k-1])$

```
bool mergef(BIGNUM b1,BIGNUM c1,BIGNUM b2,BIGNUM c2,BIGNUM &b,BIGNUM &c) {
    BIGNUM tb1=b1,tb2=b2;
    c=((c2-c1)%b2+b2)%b2;
    BIGNUM G=gcd(b1,b2);
    if(c%G)return false;
    c/=G;
    b1/=G;
    b2/=G;
    c=c*Invmod(b1,b2)%b2;
    c=c*tb1+c1;
    b=tb1*tb2/G;
    c%=b;
    return true;
}
BIGNUM merge(BIGNUM b[],BIGNUM c[],int n) {
    BIGNUM bb=b[0],cc=c[0],bbb,ccc;
    int i;
    for(i=1;i<n;++i)
    {
        if(!mergef(bb,cc,b[i],c[i],bbb,ccc))
            return -1;
        bb=bbb;
        cc=ccc;
    }
    return cc;
}
```

6.5 $a^b \bmod c$

[注意]中间变量是否会超过I64的范围

```
BIGNUM product_mod(BIGNUM a, BIGNUM b, BIGNUM c) { //(a*b)%c
    BIGNUM r = 0, d = a;
    while (b > 0) {
        if (b & 1) r = (r + d) % c;
        d = (d + d) % c;
        b >>= 1;
    }
    return r%c;
}

BIGNUM PowerMod(BIGNUM a, BIGNUM b, BIGNUM c) { //(a^b)%c
    BIGNUM r = 1, d = a;
    while (b > 0) {
        if (b & 1) r = product_mod(r, d, c);
        d = product_mod(d, d, c);
        b >>= 1;
    }
    return r%c;
```

```
}

//anther one
int exp_mod(int n,int m, int p)  //(n^m)%p
{
    if (m==0) return 1%p;
    if (m==1) return n%p;
    int temp=exp_mod(n,m/2, p)%p;
    if (m%2!=0) return ((temp*temp)%p*(n%p))%p;  //temp * temp < LONG_LONG_MAX
    else return(temp*temp) %p;
}
```

6.6 Baby-step giant-step

[说明] 参考题目FOJ 1493
baby-step giant-step解方程$B^x = N(modP)$中最小x
根据鸽巢原理,可以把"无穷大"置为$P+1$.
若返回-1或者返回值大于等于P则无解.

```
typedef long long llong;
#define BUF 65600//sqrt(P)
#define SIZE 131071
struct BabyHash{
int a,b,next;
BabyHash(){};
BabyHash(int _a,int _b){a=_a;b=_b;next=-1;}
}hash[SIZE+BUF];
static bool hflag[SIZE];
static int top;
void insBaby(int a,int b)
{
    int key=b&SIZE;
    if(!hflag[key])
    {
        hflag[key]=true;
        hash[key]=BabyHash(a,b);
        return;
    }
    while(hash[key].next!=-1)
    {
        if(hash[key].b==b)return;
        key=hash[key].next;
    }
    hash[key].next=++top;
    hash[top]=BabyHash(a,b);
}
int findBaby(int b)
{
    int key=b&SIZE;
    if(!hflag[key])return -1;
```

```
    while(key!=-1){if(hash[key].b==b)return hash[key].a;key=hash[key].next;}
    return -1;
}
int BabyStep(int a,int b,int c)//a^x mod b = c,x>=0
{
    if(!a)
    {
        if(b==1)
            return c?-1:0;
        return c==1?0:(c==0?1:-1);
    }
    memset(hflag,false,sizeof(hflag));
    top=SIZE;
    int G;
    int M,i,j,diff=0;
    llong B,P=1%b,sigma=1%b,ob=b,oc=c;
    while(1)
    {
        if(sigma==oc)return diff;
        G=gcd(a,b);
        if(G==1||c%G)break;
        b/=G;
        c/=G;
        P=P*(a/G)%b;
        sigma=sigma*a%ob;
        ++diff;
    }
    M=ceil(sqrt((double)b));
    B=1%b;
    for(i=0;i<=M;++i)insBaby(i,B),B=B*a%b;
    llong tmp=PowerMod(a,M,b);
    B=P;
    for(i=0;i<=M;++i)
    {
        G=modequ(B,c,b);
        j=findBaby(G);
        if(j!=-1)return i*M+j+diff;
        B=B*tmp%b;
    }
    return -1;
}
```

6.7 素数判定随机算法

[说明] 随机次数times可以更改

```
BIGNUM myrand() {
    BIGNUM a = rand();
    a *= rand();
    return a;
}
```

```
bool Rabin_Miller(BIGNUM n, int times = 20) {
    BIGNUM k = 0, i, j, m, a;
    if (n < 2) return false;
    if (n == 2) return true;
    if (!(n & 1)) return false;
    for (m = n - 1; !(m & 1); (m >>= 1), ++k);
    for (i = 0; i < times; ++i) {
        a = PowerMod(myrand() % (n - 2) + 2, m, n);
        if (a == 1)
            continue;
        for (j = 0; j < k && a != n - 1; ++j)
            a = mod_mul(a, a, n);
        if (j >= k) return false;
    }
    return true;
}
```

6.8 分解质因数随机算法Pollard

[输出]分解的质因子存放在ans[]中,且是无序,可以对ans排序按从小到大排放
[说明] 当n不大时，用随机算法速度反倒会慢.

```
BIGNUM ans[100], ansn; //全局变量ans初始化,ansn = 0;

BIGNUM func(BIGNUM x, BIGNUM n) {
    return (mod_mul(x, x, n) + 1) % n;
}

void Pollard(BIGNUM n, int times = 20) {
    BIGNUM i, x, y, p;
    if (Rabin_Miller(n)) {
        ans[ansn++] = n;
        return;
    }
    if (!(n & 1)) {
        Pollard(2);
        Pollard(n >> 1);
        return;
    }
    for (i = 1; i < times; ++i) {
        x = i;
        y = func(x, n);
        p = gcd(y - x, n);
        while (p == 1) {
            x = func(x, n);
            y = func(func(y, n), n);
            p = gcd((y - x + n) % n, n) % n;
        }
        if (p == 0 || p == n) continue;
        Pollard(p);
        Pollard(n / p);
```

```
            return;
        }
    }
```

[输出]1..n-1中与n互质数的个数
[注意]当n == 1的时候个数为1个

6.9 欧拉函数

求$1..n-1$中与n互质的数的个数
[说明] 欧拉公式$Euler(n) = n(1-1/p_1)(1-1/p_2)(,,).$
【求单个欧拉函数值】

```
int euler(int n) {
    int ans = 1;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            n /= i;
            ans *= i - 1;
            while (n % i == 0) {
                n /= i;
                ans *= i;
            }
        }
    }
    if (n > 1) ans *= n - 1;
    return ans;
}
```

【利用筛素数求法,可以求出$1...n$的欧拉函数值】

```
const int mx = 10000000;
int E[mx], prime[10000], pn;
bool bprime[mx];

int euler_all(int n)
{
    memset(bprime, 0, sizeof(bool) * (n + 1));
    E[0] = E[1] = 1;
    int i, j;
    for(i = 2, pn = 0;i <= n;++ i) {
        if(bprime[i] == 0) {
            prime[pn ++] = i;
            E[i] = i - 1;
        }
        for(j = 1;j < pn && prime[j] * i <= n;++ j) {
            bprime[prime[j] * i] = true;
            if(i % prime[j] == 0) {
                E[prime[j] * i] = E[i] * prime[j];
                break;
            }
            else {
```

```
                E[prime[j] * i] = E[i] * (prime[j] - 1);
            }
        }
    }
    return E[n];
}
```

【另一种写法筛法求出1... n的欧拉函数值】
顺便还产生了1..N内素数if(phi[i]==i-1)so i is a prime

```
int phi[N];
void mkphilist()
{
    int i,j;
    phi[1]=1;
    for(i=2;i<N;++i)
     if(!phi[i])
      for(j=i;j<N;j+=i)
      {
          if(!phi[j])
              phi[j]=j;
          phi[j]-=phi[j]/i;
      }
}
```

6.10 欧拉定理解方程$A^xmodB=1$中的x

```
bool p[100001]={1,1};
int P[50001],plen=0;
int fac[32][2];
int len;
bool flag;
llong ans;
int A,B1,B2;
llong B,E;
void mklist()
{
   int i,j;
   for(i=2;i<=400;++i)if(!p[i])for(j=i;j*i<=100000;++j)p[i*j]=true;
   for(i=2;i<=100000;++i)if(!p[i])P[plen++]=i;
}
void split(llong n)
{
   int i;
   len=0;
   for(i=0;i<plen;++i)
   {
       if(n%P[i]==0)
       {
           fac[len][0]=P[i];
           fac[len][1]=0;
```

```
            while(n%P[i]==0)
                n/=P[i],++fac[len][1];
            ++len;
            if(n==1||(n<=100000&&!p[n]))break;
        }
    }
    if(n!=1){fac[len][0]=n;fac[len][1]=1;++len;}
    return;
}
void dfs(int dep,llong x)
{
    if(flag)return;
    if(dep==len)
    {
        if(PowMod(A,x,B)==1)
            if(ans>x)
                ans=x;
        if(ans*ans<=E)flag=true;
        return;
    }
    int i;
    for(i=0;i<=fac[dep][1];++i)
    {
        dfs(dep+1,x);
        if(flag)return;
        x*=fac[dep][0];
    }
}
int solvemod(int A,int B)
{
    if(gcd(A,B)!=1)return -1;
    llong E=eular(B);
    ans=E;
    split(E);
    flag=false;
    dfs(0,1);
    return ans;
}
```

6.11 筛法生成质数表

```
void sift_prime(bool bprime[], int prime[], int &, pnint N) {
    int i, j;
    pn = 0;
    memset(bprime,false,sizeof(bool)*N);
    bprime[0] = bprime[1] = true;
    for (i = 2; i < N; ++i) {
        if (!bprime[i]) prime[pn++] = i;
        for (j = 0; i * prime[j] < N && j < len; ++j) {
            bprime[i * prime[j]] = 1;
            if (i % prime[j] == 0) break; //speed up linear time
```

```
        }
    }
}
```

【2进制压缩筛素数】

[说明]

-生成$[1,n]$内所有素数...

N -素数表大小...

M -最大的整数,满足$(2*(M-1)+1)^2<N$...

L -保证$2*L+1<N$...

$p[],plen$ -分别保存生成的素数和素数表长度...

-效率为普通线性优化筛法的2倍...

```
int N=13000005;
int M=1804;
int L=6500002;
int p[1000005],plen=0;
bool ss[6500002];
void mklist()
{
    int i,j,buf;
    p[plen++]=2;
    for(i=1;i<M;++i)
        if(!ss[i])
            for(buf=(i<<1)+1,j=i+buf;j<L;ss[j]=1,j+=buf);
    for(i=1;i<L;++i)if(!ss[i])p[plen++]=(i<<1)+1;
}
```

【sieve筛选法求一段区间内素数】

[说明]

-生成$[1,n]$内所有素数...

$maxn$ -区间最大长度...

L -区间左边界...

U -区间右边界...

$p[]$ -保存区间内素数

$flag[]$ -表示区间内某个数是否是素数,i-L查询

sz -区间内素数个数

```
#define maxn 1024000 int L,U; int p[maxn],sz; bool flag[maxn]; void
sieve(int L,int U){
    int d=U-L+1,i,limit=(int)sqrt((double)U);
    for(i=0;i<d;i++)flag[i]=true;
    for(i=L&0x1;i<d;i+=2)flag[i]=false;
    for(i=3;i<=limit;i+=2){
        if(i>L&&flag[i-L]==false)continue;
        int j=L/i*i;
        if(j<L)j+=i;
        if(j==i)j+=i;
        j=j-L;
        for(;j<d;j+=i)flag[j]=false;
```

```
    }
    if(L<=1)flag[1-L]=false;
    if(L<=2)flag[2-L]=true;
    sz=0;
    for(i=0;i<d;i++)if(flag[i]==true)p[sz++]=i+L;
}
```

6.12 Pell方程

```
//resolve x^2-d*y^2=c, c=1 or -1
//return 0 if there's no solution
int Pell(int d, int c,bignum &x, bignum &y)
{
  bignum x1, y1, t;
  int sd, P, Q, a, odd = 0;
  sd = (int)sqrt((double)d);
  if (sd*sd == d) return 0;
  x1 = y = P = 0; x = y1 = Q = 1;
  while ((a=(sd+P)/Q) != sd*2)
  {
    odd ^= 1; P = a*Q-P; Q = (d-P*P)/Q;
    t = x; x = x*a+x1; x1 = t;
    t = y; y = y*a+y1; y1 = t;
  }
  if (!odd && c==-1) return 0;
  if (odd && c==1)
  {
    t = x; x = t*t+y*y*d; y = t*y*2;
  }
  return 1;
}
```

[说明] 求解$x^2 - n * y^2 = 1$的最小整数解
如果n为完全平方数,则返回一组平凡解$(x = 1, y = 0)$
否则返回一组最小正解$x, y(x, y > 0)$
注意如果n比较大(几百)那么x,y可能需要高精度

```
typedef long long llong;
int Pell(llong n,llong& x,llong& y)
{
    llong aa=(llong)sqrt((double)n),a=aa;
    x=1;
    y=0;
    if(aa*aa==n)return 0;
    llong p1=1,p2=0,q1=0,q2=1,g=0,h=1;
    while(true)
    {
        g=-g+a*h;
        h=(n-g*g)/h;
```

```
        x=a*p1+p2;
        y=a*q1+q2;
        if(x*x-n*y*y==1)
            return 1;
        p2=p1;
        q2=q1;
        p1=x;
        q1=y;
        a=(g+aa)/h;
    }
    return 1;
}
```

6.13 分数逼近

[说明] 经典例题:*Asia Phuket* 2009 *Rating Hazard*

```
typedef long long LL;
#define mp make_pair
typedef pair<LL,LL> PLL;

pair<PLL,PLL> BG(PLL A,LL n){ //返回最接近A的左右边界,分子分母<= n,结果保证分子分母互质
    LL a,b;
    LL x1,y1,x2,y2,x3,y3,l,r,m,k1,k2;
    a = A.first; b = A.second;
    if(b <= n) return mp(A, A) ;
    x1 = 0, y1 = 1, x3 = 1, y3 = 1;
    while(1){
        x2 = x1 + x3, y2 = y1 + y3;
        if(y2 > n) break;
        l = 1; r = n;
        if(a * y2 < b * x2){
            while(l < r){
                m = (l + r + 1) / 2;
                k1 = m * x1 + x3, k2 = m * y1 + y3;
                if(m * y1 + y3 > n || k1 * b < k2 * a) r = m - 1;
                else l = m;
            }
            x3 = l * x1 + x3, y3 = l * y1 + y3;
        }else{
            while(l < r){
                m = (l + r + 1) / 2;
                k1 = x1 + m * x3, k2 = y1 + m * y3;
                if(y1 + m * y3 > n || k1 * b > k2 * a) r = m - 1;
                else l = m;
            }
            x1 = x1 + l * x3, y1 = y1 + l * y3;
        }
    }
    return mp( mp(x1,y1), mp(x3,y3));
}
```

6.14 初等数论

【取整函数】 $x-1<\lfloor x\rfloor\le x\le\lceil x\rceil<x+1,\quad \lfloor -x\rfloor=-\lceil x\rceil$
【mod函数】 当$y\neq 0$时，$x \bmod y=x-y\left\lfloor\frac{x}{y}\right\rfloor$; $x \bmod 0=x$

定理6.14.1 (R.J.McEliece) 令$f(x)$为区间A上的连续、严格递增函数，且$\forall x\in A$, 当$f(x)$为整数时, x也为整数。则$\forall x\in A$且$\lfloor x\rfloor,\lceil x\rceil\in A$, 有$\lfloor f(x)\rfloor=\lfloor f(\lfloor x\rfloor)\rfloor$, $\lceil f(x)\rceil=\lceil f(\lceil x\rceil)\rceil$.

【同余】 $x\equiv y\pmod z\iff x \bmod z=y \bmod z\iff z|x-y\iff (x,z)=(y,z).\quad (x,y\in Z)$

性质A. $a\equiv b,x\equiv y\Rightarrow a\pm x\equiv b\pm y,ax\equiv by\pmod m$

性质B. $ax\equiv by,a\equiv b,a\perp m\Rightarrow x\equiv y\pmod m$

性质C. $a\equiv b\pmod m\iff an\equiv bn\pmod{mn}.(n\neq 0)$

性质D. $a\equiv b\pmod{[r,s]}\iff a\equiv b\pmod r,a\equiv b\pmod s$

性质E. **(Law of inverses)** $n\perp m\Rightarrow$ ∃整数n'使$nn'\equiv 1\pmod m$

【注】 当$x,y\in R$时，性质*A,C*仍成立

定理6.14.2 (Frobenius问题) 给定$n(n\ge 2)$个互质的正整数$a_1,a_2,\cdots,a_n$，求使不定方程$\sum_i a_ix_i=F$没有非负整数解的最大正整数$F(a_1,a_2,\cdots,a_n)$. 当$n=2$时，$F(a_1,a_2)=a_1a_2-a_1-a_2$; $n=3$时，当$a=pq,b=qw,c=wp,(p,q)=(q,w)=(w,p)=1$时$F(a,b,c)=2qpw-a-b-c$; 当$(a,b,c)=1,c>\frac{ab}{(a,b)^2}-\frac{a}{(a,b)}-\frac{b}{(a,b)}$时, $F(a,b,c)=\frac{ab}{(a,b)}+c(a,b)-a-b-c$.

定理6.14.3 (费马小定理) 当p为质数时$a^p\equiv a\pmod p$

证明： i)当a是p的倍数时$a^p\equiv a\pmod p$.
ii)当$a \bmod p\neq 0$时, 数列$0 \bmod p,a \bmod p,2a \bmod p,\cdots,(p-1)a \bmod p$是$p$个不同的数(因为如果$ax \bmod p=ay \bmod p$即$ax\equiv by\pmod p,a\perp p$, 由性质B得$x\equiv y\pmod p$导致矛盾) 这$p$个数中，第1个数是0, 其余的是$1,2,\ldots,(p-1)$的某种排列, 由性质1得:

$$a\cdot(2a)\ldots\big((p-1)a\big)\equiv 1\times 2\times\cdots\times(p-1)\text{即}a^p\equiv a\pmod p$$

□

定义6.14.1 (欧拉函数) n为正整数, 定义$\varphi(n)$为$\{0,1,\cdots,n-1\}$中与n互质的数的个数。特别地，$\varphi(1)=1,\varphi(3)=2$, 当$p$为质数时$\varphi(p)=p-1,\varphi(p^e)=p^e-p^{e-1}$

$\varphi(n)$为可积函数. 一般地，当$n=p_1^{e_1}\cdots p_r^{e_r}$则

$$\varphi(n)=\big(p_1^{e_1}-p_1^{e_1-1}\big)\cdots\big(p_r^{e_r}-p_r^{e_r-1}\big)=n\prod_{p\text{为}n\text{的质因数}}\left(1-\frac{1}{p}\right)$$

定理6.14.4 (欧拉定理) $\forall$正整数$m,a\perp m\Rightarrow a^{\varphi(m)}\equiv 1\pmod m$.

例6.14.1 求证性质E: $n\perp m\Rightarrow$ ∃整数n'使$nn'\equiv 1\pmod m$

证明： 与证明定理6.14.3类似, $0 \bmod m,n \bmod m,2n \bmod m,\cdots,(m-1)n \bmod m$是$m$个不同的数且为$0\sim m-1$的某种排列。特别地必存在$n'\in[0,m-1]$使$nn'\equiv 1\pmod m$. □

例6.14.2 已知$x \bmod 3 = 2, x \bmod 5 = 3$, 求$x \bmod 15$.

解： $\because x \equiv 2 \pmod 3, x \equiv 3 \pmod 5$
为了利用性质D, 设$2 + 3m = 3 + 5n \Rightarrow \begin{cases} m = 2 \\ n = 1 \end{cases}$
$\Rightarrow x \equiv 2 \equiv 8 \pmod 3, x \equiv 3 \equiv 8 \pmod 5 \Rightarrow x \bmod 15 = 8$

例6.14.3 求1997^{1997}末尾两位数.

解： 本题相当于求$1997^{1997} \bmod 100$.但由于100太大不便于计算，考虑将其分解为4×25.

$\therefore 1997 \equiv -3 \Rightarrow 1997^{1997} \equiv -3^{1997} \pmod{100}$
$\therefore -3^{1997} \equiv -(-1)^{1997} \equiv 1 \pmod 4$
又$\because 3^3 = 27 \equiv 2 \Rightarrow 3^9 \equiv 2^3 = 8,\ 3^{10} \equiv 3 \times 8 \equiv -1 \pmod{25}$
$\therefore -3^{1997} = -3^7 \times (3^{10})^{199} \equiv -3^7 \times (-1)^{199} = 3^7 \equiv 3 \times (3^3)^2 \equiv 3 \times 2^2 \equiv 12 \pmod{25}$
$\therefore \begin{cases} 1997^{1997} \equiv 1 \equiv 37 \pmod 4 \\ 1997^{1997} \equiv 12 \equiv 37 \pmod{25} \end{cases} \Rightarrow 1997^{1997}$末尾两位数为37.

例6.14.4 求证$n = \overline{a_1 a_2 \cdots a_n}$能否被3整除可通过各位数字和$a_1 + a_2 + \cdots + a_n$能否被3整除来验证。

证明： $10 \equiv 1 \pmod 3 \Rightarrow 10^n \equiv 1 \pmod 3$
$\therefore n = a_0 + a_1 \times 10 + a_2 \times 10^2 + \cdots + a_n \times 10^n \equiv a_1 + a_2 + \cdots + a_n \pmod 3$ □

【推广】

a) $10 \equiv -1 \Rightarrow 10^n \equiv (-1)^n \pmod{11}$
$\therefore n \equiv a_0 - a_1 + a_2 + \cdots + (-1)^n a_n \pmod{11}$

b) $100 \equiv 2 \pmod 7 \Rightarrow 10^n \equiv 2 \times 10^{n-2} \pmod 7$
即得**被7整除数的判断方法：**把最高位数字×2, 然后向后退两位与原数(除去最高位)相加，重复该过程，所得的每个数都与原数同余。例如，对于$\underline{1}23456 \overset{+2}{\to} \underline{2}5456 \overset{+4}{\to} \underline{5}856 \overset{+10}{\to} \underline{9}56 \overset{+18}{\to} 74 \Rightarrow 74 \bmod 7 = 4, \quad \therefore 123456 \bmod 7 = 4$

【阶乘的数论性质】记$\mu = \left\lfloor \frac{n}{p} \right\rfloor + \left\lfloor \frac{n}{p^2} \right\rfloor + \cdots = \sum_{k>0} \left\lfloor \frac{n}{p^k} \right\rfloor$, 则$p^\mu | n!$且$p^{\mu+1} \nmid n!$.

证明： $\left\lfloor \frac{n}{p^k} \right\rfloor$表示$\{1, 2, \ldots, n\}$中是$p^k$倍数的元素个数，因此$\forall a \in N, p^j | a, p^{j+1} \nmid a$, 恰好被计数了$j$次，依次在$\left\lfloor \frac{n}{p} \right\rfloor, \left\lfloor \frac{n}{p^2} \right\rfloor, \cdots, \left\lfloor \frac{n}{p^j} \right\rfloor$中。 □

【注】根据定理*6.14.1*得$\left\lfloor \frac{n}{p^{k+1}} \right\rfloor = \left\lfloor \frac{\left\lfloor \frac{n}{p^k} \right\rfloor}{p} \right\rfloor$可由前一项结果算出下一项，减少计算量。

定理6.14.5 (Wilson定理) p为质数，则$(p-1)! \bmod p = p - 1$.

证明： i)当$p=2$时$1 \bmod 2=1$成立。

ii)当p为奇质数时，在$(p-1)!$中，$\forall a \in [1,p-1], \exists a'$使$aa' \equiv 1 \pmod p$且整数$a' \in [1,p-1]$是由$a$唯一确定的(否则$aa' \equiv aa''$则$a' \equiv a''$且$a', a'' \in [1,p-1]$必有$a'=a''$). 所以$(p-1)!$中每个因子可成对分组，每组两个数相乘均与$1 \bmod p$等价。但应注意有可能出现$a=a'$的情况，这时$a^2 \equiv 1 \pmod p$即$p|a^2-1=(a+1)(a-1) \Rightarrow p|a-1$或$p|a+1$, 满足$p|a-1$的只有1，满足$p|a+1$的只有$p-1$. 于是，除去$1, p-1$之后的每组两数积与1同余。

$$\therefore (p-1)! \equiv 1 \cdot (p-1) \equiv p-1 \pmod p \Rightarrow (p-1)! \bmod p = p-1$$

□

第七章 组合数学

7.1 常用公式

定理7.1.1 ($P\acute{o}lya$ 定理) 设G是p个对象的一个置换群，用m种颜色涂染p个对象，则不同染色方案为：

$$L=\frac{1}{|G|}\sum_{i=1}^{|G|}m^{c(g_i)}$$

【错排公式】

$$D_n=(n-1)(D_{n-1}+D_{n-2})\Rightarrow D_n=n!\left(1-1+\frac{1}{2!}-\frac{1}{3!}+\cdots\right)=\sum_{i=0}^{n}(-1)^i\binom{n}{i}(n-i)!$$

【Catalan数】

$$B_n=\begin{cases}1 & n=0\\ \sum_{i=0}^{n-1}B_iB_{n-i-1} & n\geq 1\end{cases}\Rightarrow B_n=\frac{1}{n+1}\binom{2n}{n}$$

【重组合】从n个对象中选取k个允许重复的选法为$H_n^k=\binom{n+k-1}{k}$.

证明： 本题等价于满足不等式$1\leq a_1\leq a_2\cdots\leq a_k\leq n$的整数解的个数。它等价于$0<a_1<a_2+1<\cdots<a_k+k-1<n+k\iff 0<b_1<b_2\cdots<b_k<n+k\iff$从$n+k-1$个不同的数$\{1,2,\cdots,n+k-1\}$中选出$k$个数的组合，即$\binom{n+k-1}{k}$. □

【组合数系统】$\forall$整数$n,\exists 0\leq a<b<c<d$, 使$n=\binom{a}{1}+\binom{b}{2}+\binom{c}{3}+\binom{d}{4}$成立。可用贪心算法依序求出$d,c,b,a$.

【第一类Stirling数】$\begin{bmatrix}n\\m\end{bmatrix}$表示将$n$个元素分为$m$个圈的排列数。

【第二类Stirling数】$\begin{Bmatrix}n\\m\end{Bmatrix}$表示将$n$个元素分为$m$个互不相交的子集的方法数。

$$\begin{Bmatrix}n\\m\end{Bmatrix}=\begin{Bmatrix}n-1\\m-1\end{Bmatrix}+k\begin{Bmatrix}n-1\\m\end{Bmatrix}\Rightarrow\begin{Bmatrix}n\\m\end{Bmatrix}=\frac{1}{m!}\sum_{i=0}^{m}(-1)^i\binom{i}{m}(m-i)^n$$

证明： 令$f(n,m)$为将$\{1,2,\cdots,n\}$分为m个非空集合的方法数。显然$f(1,m)=\delta_{1m}$. 当$n>1$时，对元素n有两种可能性：a)将n单独作为一个子集，有$f(n-1,m-1)$种方法。b)将n放入一个已存在的子集中，$\forall m$-划分都有m种方法，共有$mf(n-1,m)$种方法。
$\therefore f(n,m)=f(n-1,m-1)+mf(n-1,m)$由归纳法可知$f(n,m)=\begin{Bmatrix}n\\m\end{Bmatrix}$. □

【经典计数模型】

- n个球有区别，m个盒子有区别，有空盒时方案计数为m^n
- n个球有区别，m个盒子有区别，无空盒时方案计数为$m!\left\{ {n \atop m} \right\}$
- n个球有区别，m个盒子无区别，有空盒时方案计数为$\sum_{i=1}^{m}\left\{ {n \atop i} \right\}$
- n个球有区别，m个盒子无区别，无空盒时方案计数为$\left\{ {n \atop m} \right\}$
- n个球无区别，m个盒子有区别，有空盒时方案计数为$\binom{n+m-1}{n}$
- n个球无区别，m个盒子有区别，无空盒时方案计数为$\binom{n-1}{m-1}$
- n个球无区别，m个盒子无区别，有空盒时方案计数相当于n用$\{1,2,\cdots,m\}$拆分的拆分数，即$\frac{1}{(1-x)(1-x^2)\cdots(1-x^m)}$的$x^n$的系数
- n个球无区别，m个盒子无区别，无空盒时方案计数相当于$n-m$用$\{1,2,\cdots,m\}$拆分的拆分数，即$\frac{1}{(1-x)(1-x^2)\cdots(1-x^m)}$的$x^{n-m}$的系数
- 把k_1个a_1,k_2个$a_2,\cdots k_r$个a_r放入n个不同盒子，每盒可空的方案计数为$\prod_{i=1}^{r}\binom{n+k_i-1}{k_i}$
- 把k_1个a_1,k_2个$a_2,\cdots k_r$个a_r放入n个不同盒子，每盒不空的方案计数为

$$\prod_{j=0}^{n-1}\left((-1)^j\binom{n}{j}\prod_{i=0}^{r}\binom{n-j+k_i-1}{k_i}\right)$$

【差分多项式】

如果$p(x)$是n次多项式且差分表左边沿等于$c_0,c_1,\cdots,c_n,0,0,\cdots$,则

$$p(x)=c_0\binom{x}{0}+c_1\binom{x}{1}+\cdots+c_n\binom{x}{n}$$

如果$p(x)$是n次多项式且差分表左边沿等于$c_0,c_1,\cdots,c_n,0,0,\cdots$,则

$$\sum_{t=0}^{m}p(t)=c_0\binom{m+1}{1}+c_1\binom{m+1}{2}+\cdots+c_n\binom{m+1}{n+1}$$

【基本波利亚定理】求置换的循环节,polya原理.perm[0..n-1]为0..n-1的一个置换(排列).返回置换最小周期,num返回循环节个数

```
int gcd(int a,int b){
    return b?gcd(b,a%b):a;
}

int polya(int* perm,int n,int& num){
    int i,j,p,v[MAXN]={0},ret=1;
    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;
}
```

【项链染色问题】m 颜色数, n 珠子数

1. 只考虑旋转$\frac{\sum_{0<=k<n} m^{gcd(n,k)}}{n}$
2. 只考虑翻转
   - n为偶数$(n/2 * m^{(n/2)} + n/2 * m^{(n/2+1)})/n$
   - n为奇数$(n * m^{((n+1)/2)})/n$
   - 综合考虑..../$(2 * n)$

[说明] 旋转和翻转视为相同，用c种颜色染n个珠子，构成不同项链的种数。 [说明] 利用到的有:
如果gcd(n,m)=k,并有,1¡=m¡=n,那么显然可以得到满足条件的m的个数等价
于gcd(n/k,m/k)=1的个数.于是可以利用欧拉函数来解.
如果mod的数字比较大，可以考虑使用乘法逆元(如果模数是素数或square-free数),
然后可以解决(中国剩余定理).

```
typedef long long llong;
llong hpow(llong a,llong b)
{
    llong ret=1;
    while(b)
    {
        if(b&0x1)
            ret=ret*a;
        a=a*a;
        b>>=1;
    }
    return ret;
}
llong polya(llong c,llong n)
{
    llong i;
    llong t=hpow(c,n)+(n!=1?Euler(n)*c:0);
    for(i=2;i*i<n;++i)
        if(n%i==0)
            t+=Euler(n/i)*hpow(c,i)+Euler(i)*hpow(c,n/i);
    if(i*i==n)t+=Euler(i)*hpow(c,i);
    t/=n;
    if(n&0x1)
        t+=hpow(c,(n>>1)+1);
    else
        t+=hpow(c,n>>1)*(c+1)>>1;
    return (t>>1);
}
```

【计算置换群的m次方, n 为置换群长度标记从1到n】

```
#define N 230
int change[N], temp[N], ans[N], flag[N];
int m, n, cnt;

void rotate(int k) {
    int i;
    for (i = 1; i <= n; ++i) if (flag[i] == k) temp[i] = change[ans[i]];
    for (i = 1; i <= n; ++i) if (flag[i] == k) ans[i] = temp[i];
}

void circulate() {
    int i, j;
    for (i = 1; i <= n; ++i) {
        if (!flag[i]) {
            int len = 0, now = i;
            while (!flag[now]) {
                flag[now] = cnt;
                now = change[now];
                ++len;
            }
            cnt++;
            int k = m % len;
            for (j = 0; j < k; ++j)
                rotate(cnt - 1);
        }
    }
}
```

7.2 Fibonacci数

A. $F_{n+1}F_{n-1} - F_n^2 = (-1)^n$ **【推论】** $F_n \perp F_{n+1}$.

证明：

$$\det\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \det\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \Rightarrow F_{n+1}F_{n-1} - F_n^2 = (-1)^n$$

□

B. $F_{n+m} = F_mF_{n+1} + F_{m-1}F_n$ **【推论】** $F_n | F_{nk}$.

定理7.2.1 (É.Lucas)

$$\gcd(F_m, F_n) = F_{\gcd(m,n)} \tag{7.2.1}$$

证明： 由性质B得F_m, F_n的公因数必为F_{n+m}的因数，相反地，F_{m+n}, F_n的公因数必为F_mF_n+1的因数。由F_n, F_{n+1}互质得F_{n+m}与F_n的公因数为F_m的因数。即$d|F_{m+n}, d|F_n \iff d|F_m$, 归纳可得$\gcd(F_m, F_n) = d \iff \gcd(F_{m \bmod n}, F_n) = d$, 由gcd算法可得$\gcd(F_m, F_n) = \cdots = \gcd(F_0, F_{\gcd(m,n)})$. □

【F_n的母函数】设$G(z)=F_0+F_1z+F_2z^2+\cdots=z+z^2+2z^3+3z^4+\cdots$

$$\begin{array}{rrr} & G(z)= & F_0+F_1z+F_2z^2+F_3z^3+\cdots \\ & zG(z)= & F_0z+F_1z^2+F_2z^3+\cdots \\ - & z^2G(z)= & F_0z^2+F_1z^3+\cdots \\ \hline & (1-z-z^2)G(z)= & F_0+(F_1-F_0)z+0\cdot z^2+\cdots \end{array}$$

$$\therefore G(z)=\frac{z}{1-z-z^2}=\frac{1}{\sqrt{5}}\left(\frac{1}{1-\phi z}-\frac{1}{1-\Phi z}\right)\quad(\Phi=1-\phi=\frac{1-\sqrt{5}}{2})$$

$$\therefore G(z)=\frac{1}{\sqrt{5}}(1+\phi z+\phi^2z^2+\cdots-1-\Phi z-\Phi^2z^2\cdots)\quad\text{(通过无穷等比数列和公式)}$$

$$\therefore G(z)\text{中}z^n\text{的系数即为}F_n=\frac{1}{\sqrt{5}}(\phi^n-\Phi^n)$$

【推论】

1. $\phi^n=F_n\phi+F_{n-1};\quad \Phi^n=F_n\Phi^n+F_{n-1}$
2. 由于$|\Phi|<1$, 即$\frac{\Phi^n}{\sqrt{5}}$足够小，以至于$F_n=\left(\frac{\phi^n}{\sqrt{5}}\text{舍入最接近的整数}\right)$
3. $G(z)^2=\frac{1}{5}\left(\frac{1}{(1-\phi z)^2}+\frac{1}{(1-\Phi z)^2}-\frac{2}{1-z-z^2}\right)$, z^n的系数为$\sum_{k=0}^{n}F_kF_{n-k}$.
   $\therefore\sum_{k=0}^{n}F_kF_{n-k}=\frac{1}{5}\Big((n+1)(\phi^n+\Phi^n)-2F_{n+1}\Big)=\frac{1}{5}\Big((n+1)(F_n+2F_{n-1})-2F_{n+1}\Big)=\frac{1}{5}(n-1)F_n+\frac{2}{5}nF_{n-1}$.
4. $a_0=r,a_1=s,a_{n+2}=a_{n+1}+a_n$则$a_n=rF_{n-1}+sF_n$.

【注】 当n为负数时除了推论2外其余的结论均成立。其正确性可通过假设$k\geq n$成立，证明$k-1$成立来得到。

【求和公式】

1. $\sum_{k=0}^{n}F_k=F_{n+2}-1$
2. $\sum_{k=0}^{n}\binom{n-k}{k}=F_{n+1}$
3. $\sum_k\binom{n}{k}F_{m+k}=F_{m+2n}$
   【推广】$\sum_k\binom{n}{k}F_t^kF_{t-1}^{n-k}F_{m+k}=F_{m+tn}$
4. $\sum_{k=1}^{n}k=\frac{n\,(n+1)}{2}$

5. $\sum_{k=1}^{n} 2k-1 = n^2$

6. $\sum_{k=1}^{n} k^2 = \frac{n\left(n+1\right)\left(2n+1\right)}{6}$

7. $\sum_{k=1}^{n} \left(2k-1\right)^2 = \frac{n\left(4n^2-1\right)}{3}$

8. $\sum_{k=1}^{n} k^3 = \left(\frac{n\left(n+1\right)}{2}\right)^2$

9. $\sum_{k=1}^{n} \left(2k-1\right)^3 = n^2\left(2n^2-1\right)$

10. $\sum_{k=1}^{n} k^4 = \frac{n\left(n+1\right)\left(2n+1\right)\left(3n^2+3n-1\right)}{30}$

11. $\sum_{k=1}^{n} k^5 = \frac{n^2\left(n+1\right)^2\left(2n^2+2n-1\right)}{12}$

12. $\sum_{k=1}^{n} k\left(k+1\right) = \frac{n\left(n+1\right)\left(n+2\right)}{3}$

13. $\sum_{k=1}^{n} k\left(k+1\right)\left(k+2\right) = \frac{n\left(n+1\right)\left(n+2\right)\left(n+3\right)}{4}$

14. $\sum_{k=1}^{n} k\left(k+1\right)\left(k+2\right)\left(k+3\right) = \frac{n\left(n+1\right)\left(n+2\right)\left(n+3\right)\left(n+4\right)}{5}$

【Finbonacci数系统】令$k \gg m$表示$k \geq m+2$,则$\forall n \in Z^+, n$能被唯一表示为$n = F_{k_1} + F_{k_2} + \cdots + F_{k_r}(k_1 \gg k_2 \gg \cdots \gg K_r \gg 0)$. F_{k_i}可通过贪心算法取得，因为当F_{k_1}取与n最接近的F_n时$n - F_{k_1} < F_{k_1-1}$归纳可得结果，唯一性的证明与唯一分解定理相似。
【ϕ-进制系统】规定ϕ-进制中不能出现连续相邻的1，不能出现无穷序列，则$\forall n \in Z^+$能被以ϕ为基的ϕ-进制系统唯一表示。如$1 = (.11)_\phi = (.011\cdots)_\phi$但它们均不符合规定。$n$对应的$\phi$-进制数可通过贪心算法求出。

例7.2.1 (Fibonacci Strings) 令S_1 =“a”,S_2 =“b”,$S_{n+2} = S_{n+1}S_n$即S_{n+2}由S_{n+1}在左，S_n在右合成，例如S_3 =“ba”,S_4 =“bab”. 试探求S_n的性质。

【性质】

- S_∞中没有连续2个a，没有连续3个b.
- S_n中包含F_{n-2}个a,F_{n-1}个b.

- 如果将$m-1$表示为Finbonacci进制数，则S_∞中第m个字符为a，当且仅当$k_r=2$.
- S_∞中第k个字符为b, 当且仅当$\lfloor(k+1)\phi^{-1}\rfloor-\lfloor k\phi^{-1}\rfloor=1$.
- S_∞中第k个字符为b, 当且仅当$k=\lfloor m\phi\rfloor$ (m为正整数).
- S_∞前k个字符中b的个数为$\lfloor(k+1)\phi^{-1}\rfloor$.

例7.2.2 (取石子) 有n个石子，A,B两个人轮流取石子。第一次A可以取仍一个石子，但不能取光。此后，后一个人取的石子数可为1至对手取石子数的2倍。取走最后一个石子的人获胜。求A的必胜策略。

解： 将n表示为Fibonacci进制数$n=F_{k_1}+F_{k_2}+\cdots+F_{k_r}$, 令$\mu(n)=F_{k_r},\mu(0)=\infty$,可得：

(A) 当$n>0$时,$\mu\big(n-\mu(n)\big)>2\mu(n)$.

证明： $\mu\big(n-\mu(n)\big)=F_{k_{r-1}}\geq F_{k_r+2}>2F_{k_r}$ □

(B) 当$0<m<F_k$时，$\mu(m)\leq 2(F_k-m)$.

证明： 令$\mu(m)=F_j$; $m\leq F_{k-1}+F_{k-3}+\cdots+F_{j+(k-1-j)\bmod 2}=-F_{j-1+(k-1-j)\bmod 2}+F_k\leq -\frac{1}{2}F_j+F_k$ □

(C) 当$0<m<\mu(n)$时，$\mu\big(n-\mu(n)+m\big)\leq 2\mu\big(\mu(m)-m\big)$.

证明： 由(B)直接推得. □

(D) 当$0<m<\mu(n)$时，$\mu(n-m)\leq 2m$.

证明： 令$m=\mu(n)-m$代入(C)可得. □

【必胜策略】当本轮有n个石子，最多可取q个石子时，当且仅当$\mu(n)\leq q$时有必胜策略：取$\mu(n)$个石子。

证明： 1. 当$\mu(n)>q$时，所有移动方案将得到n',q'使$\mu(n')\leq q'$[由(D)可得].

2. 当$\mu(n)\leq q$时，我们可以取$\mu(n)$个(当$q\geq n$), 其它取法将得到n',q'使$\mu(n')>q'$.
由此可得必胜序列$F_{k_j}+\cdots+F_{k_r}$ $\big(1\leq j\leq r$且$j=1$或$F_{k_{j-1}}>2(F_{k_j}+\cdots+F_{k_r})\big)$ □

7.3 母函数

【性质】设$G(z)$为a_n的母函数；$H(z)$为b_n的母函数。

A. 线性组合 $\alpha G(z)+\beta H(z)$是$< \alpha a_n+\beta b_n >$的母函数，即$\alpha\sum_{n\geq 0}a_nz^n+\beta\sum_{n\geq 0}b_nz^n=\sum_{n\geq 0}(\alpha a_n+\beta b_n)z^n$

B. 变阶 $z^mG(z)$是$< a_{n-m} >=0,\ldots,0,a_0,a_1,\ldots$的母函数；$z^{-m}G(z)$是$< a_{n+m} >=a_m,a_{m+1},\cdots$的母函数。

由性质A,B可得线性递归方程$a_n=c_1a_{n-1}+\cdots+c_ma_{n-m}$的解$a_n=\frac{p(n)}{1-c_1z-\cdots-c_mz^m}$

【特例】设$G(z)$为常数列$1,1,\cdots$的母函数，则$zG(z)$代表$< a_{n-1} >$即$0,1,1,\cdots$，因此$(1-z)G(z)=1$，由此可得重要公式

$$\frac{1}{1-z}=1+z+z^2+\cdots \tag{7.3.1}$$

C. 乘积 $G(z)H(z)$为$\sum_{k=0}^{n}a_kb_{n-k}$的母函数。

【特例】

a) 当$b_n=[n=m]$时即得性质B

b) 当$b_n=1$时$G(z)H(z)$为$\sum_{k=0}^{n}a_k$的母函数.

c) 当递归关系中含有组合数时，通常用$< \frac{a_n}{n!} >$作母函数。例如$c_n=\sum_k\binom{n}{k}a_nb_{n-k}$则设$G(z)$为$\frac{a_n}{n!}$，$H(z)$为$\frac{b_n}{n!}$则$G(z)H(z)$为$\sum_{k=0}^{n}\frac{a_nb_{n-k}}{k!(n-k)!}=\frac{c_n}{n!}$的母函数。

【推广】任意个母函数乘积是$\sum_{\substack{k_0,k_1,\cdots\geq 0\\ k_0+k_1+\cdots=n}}a_{0k_0}a_{1k_1}\cdots$的母函数，即$\prod_{j\geq 0}\sum_{k\geq 0}a_{jk}z^k=\sum_{n\geq 0}z^n\sum_{\substack{k_0,k_1,\cdots\geq 0\\ k_0+k_1+\cdots=n}}a_{0k}$

特别地，3个母函数相乘得到$\sum_{\substack{i,j,k\geq 0\\ i+j+k=n}}a_ib_jc_k$的母函数。(常逆用本性质将复杂的求和转化为若干母函数的乘积)

D. 换元 $G(cz)$为$< c^ka_k >$的母函数。特别地$1,c,c^2,\cdots$的母函数是$\frac{1}{1-cz}$

要抽出级数的交替的项，有一个熟知的技巧：

$$\begin{cases}\frac{1}{2}\Big(G(z)+G(-z)\Big)=a_0+a_2z^2+a_4z^4+\cdots\\ \frac{1}{2}\Big(G(z)-G(-z)\Big)=a_1z+a_3z^3+a_5z^5+\cdots\end{cases}$$

使用单位复根，我们得到抽出级数的第km项的方法。令$\omega = e^{2\pi i/m} = \cos\left(\frac{2\pi}{m}\right) + i\sin\left(\frac{2\pi}{m}\right)$则

$$\sum_{n \bmod m=r} a_n z^n = \frac{1}{m}\sum_{0\le k<m} \omega^{-kr} G(\omega^k z) \quad (0 \le r < m)$$

例如$m=3, r=1$时$\omega = -\frac{1}{2} + \frac{\sqrt{3}}{2}i$, 则$a_1 z + a_4 z^4 + a_7 z^7 + \cdots = \frac{1}{3}\Big(G(z) + \omega^{-1}G(\omega z) + \omega^{-2}G(\omega^2 z)\Big)$

E. 微分与积分 由$G'(z) = a_1 + 2a_2 z + 3a_3 z^2 + \cdots = \sum_{k\ge 0}(k+1)a_{k+1}z^k$可得$zG'(z)$是$< na_n >$的母函数。

相反的，$\int_0^z G(t)\,dt = a_0 z + \frac{1}{2}a_1 z^2 + \frac{1}{3}a_2 z^3 + \cdots = \sum_{k\ge 1}\frac{1}{k}a_{k-1}z^k$

【特例】设$G(z) = \frac{1}{1-z}$则

$$\begin{aligned} G'(z) &= \frac{1}{(1-z)^2} = 1 + 2z + 3z^2 + \cdots = \sum_{k\ge 0}(k+1)z^k \\ \int_0^z G(t)\,dt &= \ln\frac{1}{1-z} = z + \frac{1}{2}z^2 + \frac{1}{3}z^3 + \cdots = \sum_{k\ge 1}\frac{1}{k}z^k \\ F(z) = G(z)\int_0^z G(t)\,dt &= \frac{1}{1-z}\ln\frac{1}{1-z} = z + \frac{3}{2}z^2 + \frac{11}{6}z^3 + \cdots = \sum_{k\ge 0} H_k z^k \end{aligned}$$

对上式求导得$F'(z) = \frac{1}{(1-z)^2} + \frac{\ln\frac{1}{1-z}}{(1-z)^2}$

$\because\ F'(z) =< (n+1)H_{n+1} >,\quad \frac{1}{(1-z)^2} =< \sum_{k=0}^{n} 1 >,\quad \frac{\ln\frac{1}{1-z}}{(1-z)^2} =< \sum_{k=1}^{n} H_k >\quad \therefore \sum_{k=1}^{n-1} H_k = nH_n - n$

F. 已知的母函数

i) **二项式定理** $(1+z)^r = 1 + rz + \frac{r(r-1)}{2}z^2 + \cdots = \sum_{k\ge 0}\binom{r}{k}z^k$

【特例】当r为负整数时$\frac{1}{(1-z)^{n+1}} = \sum_{k\ge 0}\binom{-n-1}{k}(-z)^k = \sum_{k\ge 0}\binom{n+k}{n}z^k$

ii) **指数级数** $e^z = 1 + z + \frac{1}{2!}z^2 + \cdots = \sum_{k\ge 0}\frac{1}{k!}z^k$

【推广】$(e^z - 1)^n = z^n + \frac{1}{n+1}\left\{{n+1 \atop n}\right\} z^{n+1} + \cdots = n!\sum_k \left\{{k \atop n}\right\}\frac{z^k}{k!}$

iii) **对数级数**

$$\begin{aligned}\ln(1+z) &= z-\frac{1}{2}z^2+\frac{1}{3}z^3-\cdots=\sum_{k\geq 1}\frac{(-1)^{k+1}}{k}z^k\\ \frac{1}{(1-z)^{m+1}}\ln\left(\frac{1}{1-z}\right) &= \sum_{k\geq 1}(H_{m+k}-H_m)\binom{m+k}{k}z^k\end{aligned}$$

$$\left(\ln\frac{1}{1-z}\right)^n=z^n+\frac{1}{n+1}\begin{bmatrix}n+1\\ n\end{bmatrix}z^{n+1}+\cdots=n!\sum_k\begin{bmatrix}k\\ n\end{bmatrix}\frac{z^k}{k!}$$

iv) **其它**

$$\begin{aligned}z(z+1)\cdots(z+n-1) &= \sum_k\begin{bmatrix}n\\ k\end{bmatrix}z^k\\ \frac{z^n}{(1-z)(1-2z)\cdots(1-nz)} &= \sum_k\begin{Bmatrix}k\\ n\end{Bmatrix}z^k\\ \frac{z}{e^z-1} &= \sum_{k\geq 0}\frac{B_k z^k}{k!}\end{aligned}$$

G. 分离系数 $[z^n]G(z)=a_n$

【推论】 $\dfrac{[z^n]G(z)}{1-z}=\sum_{k=0}^{n}a_k;\quad [z^n]G(z)=\dfrac{1}{2\pi i}\oint_{|z|=r}\dfrac{G(z)\,dz}{z^{n+1}}$

【推广】 $[f(z)]G(z)=f\left(\frac{1}{z}\right)G(z)$的常数项。例如$[z^2-2z^5]G(z)=\left(\frac{1}{z^2}-\frac{2}{z^5}\right)G(z)$的常数项$=a_2-2a_5$

例7.3.1 已知n个数$x_1,x_2,\cdots,x_n$,设$h_m=\sum_{1\leq j_1\leq\cdots\leq j_m\leq n}x_{j_1}\cdots x_{j_m}$, $S_j=\sum_{k=1}^{n}x_k^j$. 将h_m用S_j表出。

例如,$h_2=\frac{1}{2}S_1^2+\frac{1}{2}S_2, h_3=\frac{1}{6}S_1^3+\frac{1}{2}S_1S_2+\frac{1}{3}S_3$.

解： 设$G(z)=\sum_{k\geq 0}h_kz^k=(1+x_1z+x_1^2z^2+\cdots)\cdots(1+x_nz+x_n^2z^2+\cdots)=\dfrac{1}{(1-x_1z)(1-x_2z)\cdots(1-x_nz)}$

$G(z)$是多项式的倒数，此时取对数往往可简化计算。

$$\ln G(z)=\ln\frac{1}{1-x_1z}+\cdots+\ln\frac{1}{1-x_nz}=\left(\sum_{k\geq 1}\frac{x_1^kz^k}{k}\right)+\cdots+\left(\sum_{k\geq 1}\frac{x_n^kz^k}{k}\right)=\sum_{k\geq 1}\frac{S_kz^k}{k}$$

$$\therefore G(z)=e^{\ln G(z)}=exp\left(\sum_{k\geq 1}\frac{S_kz^k}{k}\right)=\prod_{k\geq 1}e^{S_kz^k/k}=(1+S_1z+\frac{S_1^2z^2}{2!}+\cdots)(1+\frac{S_2z^2}{2}+$$

$$\frac{S_2^2z^4}{2^2\dot{2}!}+\cdots)\cdots=\sum_{m\geq 0}\left(\sum_{\substack{k_1,k_2,\cdots,k_m\geq 0\\ k_1+2k_2+\cdots+mk_m=m}}\frac{S_1^{k_1}}{1^{k_1}k_1!}\frac{S_2^{k_2}}{2^{k_2}k_2!}\cdots\frac{S_m^{k_m}}{m^{k_m}k_m!}\right)z^m$$

【注】 事实上h_m的表达式并不复杂,h_m的项数即为m的整数拆分数。例如$m=12$时,*12*的一种

拆分方法为$12=5+2+2+2+1$,它对应$k_1+2k_2+\cdots+12k_{12}=12$的一个解,其中$k_j$为$j$在拆分方案中的个数，本例中$k_1=1,k_2=3,k_5=1$,其余的$k_j=0$, 所以$\frac{S_1}{1^1 1!}\frac{S_2^3}{2^3 3!}\frac{S_5}{5^1 1!}=\frac{1}{240}S_1S_2^3S_5$为$h_{12}$中的一项。
将$\ln G(z)$求导易得h_n的递归关系：$h_n=\frac{1}{n}(S_1h_{n-1}+S_2h_{n-2}+\cdots+S_nh_0)\quad(n\geq 1)$

例7.3.2 已知S_m可用h_j表出，例如$S_1=h_1,S_2=2h_2-h_1^2,\cdots$,求$S_m$的此种表示中$h_1^{k_1}h_2^{k_2}\cdots h_m^{k_m}$ $(k_1+2k_2+\cdots+mk_m=m)$项的系数。

解： 由$\displaystyle\sum_{m\geq 1}\frac{S_mz^m}{m}=\ln G(z)=\sum_{k\geq 1}(-1)^{k-1}\frac{(h_1z+h_2z^2+\cdots)^k}{k}$得
系数为$\dfrac{(-1)^{k_1+k_2+\cdots+k_m-1}m(k_1+k_2+\cdots+k_m-1)!}{k_1!k_2!\cdots k_m!}$

例7.3.3 求双下标数列$a_{mn}=\binom{n}{m}$的母函数。

解： $\displaystyle\sum_{m,n\geq 0}a_{mn}w^mz^n=\sum_{m,n\geq 0}\binom{n}{m}w^mz^n=\sum_{n\geq 0}(1+w)^nz^n=\frac{1}{1-z-wz}$

例7.3.4 给定正整数n,r.求

$$\text{(a)}\sum_{1\leq k_1<k_2<\cdots<k_r\leq n}k_1k_2\cdots k_r\quad\text{(b)}\sum_{1\leq k_1\leq k_2\leq\cdots\leq k_r\leq n}k_1k_2\cdots k_r$$

解： (a)当n固定时$G_n(z)=(1+z)(1+2z)\cdots(1+nz)$
$\Rightarrow G_n(z)=z^{n+1}\left(\frac{1}{z}\right)\left(\frac{1}{z}+1\right)\left(\frac{1}{z}+2\right)\cdots\left(\frac{1}{z}+n\right)=\sum_k\begin{bmatrix}n+1\\k\end{bmatrix}z^{n+1-k}\Rightarrow\begin{bmatrix}n+1\\n+1-r\end{bmatrix}$
(b)类似的，$G_n(z)=\dfrac{1}{1-z}\cdot\dfrac{1}{1-2z}\cdots\dfrac{1}{1-nz}=\displaystyle\sum_k\begin{Bmatrix}k\\n\end{Bmatrix}z^{k-n}\Rightarrow\begin{Bmatrix}n+r\\n\end{Bmatrix}$

例7.3.5 求$\displaystyle\sum_{k_0+2k_1+4k_2+8k_3+\cdots=n}\binom{r}{k_0}\binom{r}{k_1}\binom{r}{k_2}\binom{r}{k_3}\cdots$

解： $G(z)=(1+z)^r(1+z^2)^r(1+z^4)^r\cdots=(1-z)^{-r}=\displaystyle\sum_n\binom{r+n-1}{n}z^n\Rightarrow\binom{r+n-1}{n}$

第八章　数值计算

8.1　解线性方程组

【高斯消元法】

```
#define MAXN 100
#define fabs(x) ((x)>0?(x):-(x))
#define eps 1e-10

//列主元消去求解gaussa[][]x[]=c[]
//序数矩阵a[n][n+1], 存放在ca[][n列+1]
//返回是否有唯一解若有解在,x中[]
int gauss_cpivot(double a[][MAXN], int n, double x[]) {
    int i, j, k, p;
    for (j = 0; j < n; ++j) {
        for (i = j + 1, p = j; i < n; ++i)
            if (fabs(a[i][j]) > fabs(a[p][j]))
                p = i;
        if (fabs(a[p][j]) < eps) return 0;
        if (p != j)
            for (k = j; k <= n; ++k)
                swap(a[j][k], a[p][k]);
        for (i = j + 1; i < n; ++i)
            for (k = n; k >= j; --k)
                a[i][k] -= a[j][k] * a[i][j] / a[j][j];
    }
    for (j = n - 1; j >= 0; --j) {
        x[j] = a[j][n] / a[j][j];
        for (i = j - 1; i >= 0; --i)
            a[i][n] -= a[i][j] * x[j];
    }
    return 1;
}

//全主元消去解gaussa[][]x[]=b[]
//返回是否有唯一解若有解在,b中[]
int gauss_tpivot(int n, double a[][MAXN], double b[]) {
    int i, j, k, row, col, index[MAXN];
    double maxp, t;
    for (i = 0; i < n; i++) index[i] = i;
    for (k = 0; k < n; k++) {
        for (maxp = 0, i = k; i < n; i++)
            for (j = k; j < n; j++)
```

```
                if (fabs(a[i][j]) > fabs(maxp))
                    maxp = a[row = i][col = j];
        if (fabs(maxp) < eps) return 0;
        if (col != k) {
            for (i = 0; i < n; i++)
                t = a[i][col], a[i][col] = a[i][k], a[i][k] = t;
            j = index[col], index[col] = index[k], index[k] = j;
        }
        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];
    for (k = 0; k < n; k++) a[0][index[k]] = b[k];
    for (k = 0; k < n; k++) b[k] = a[0][k];
    return 1;
}
```

8.2 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(double a[], double b[], double c[], double x[])
{
  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];
}
```

8.3 矩阵运算

```
#define MAXN 100

#define fabs(x) ((x)>0?(x):-(x))
#define zero(x) (fabs(x)<1e-10)

struct mat{
  int n,m;
  double data[MAXN][MAXN];
};
//矩阵乘
int mul(mat& c,const mat& a,const mat& b){
  int i,j,k;
  if (a.m!=b.n)
    return 0;
  c.n=a.n,c.m=b.m;
  for (i=0;i<c.n;i++)
    for (j=0;j<c.m;j++)
      for (c.data[i][j]=k=0;k<a.m;k++)
        c.data[i][j]+=a.data[i][k]*b.data[k][j];
  return 1;
}
//矩阵的逆
int inv(mat& a){
  int i,j,k,is[MAXN],js[MAXN];
  double t;
  if (a.n!=a.m)
    return 0;
  for (k=0;k<a.n;k++){
    for (t=0,i=k;i<a.n;i++)
      for (j=k;j<a.n;j++)
        if (fabs(a.data[i][j])>t)
          t=fabs(a.data[is[k]=i][js[k]=j]);
    if (zero(t))
      return 0;
    if (is[k]!=k)
      for (j=0;j<a.n;j++)
        t=a.data[k][j],a.data[k][j]=a.data[is[k]][j],a.data[is[k]][j]=t;
    if (js[k]!=k)
```

```
      for (i=0;i<a.n;i++)
        t=a.data[i][k],a.data[i][k]=a.data[i][js[k]],a.data[i][js[k]]=t;
    a.data[k][k]=1/a.data[k][k];
    for (j=0;j<a.n;j++)
      if (j!=k)
        a.data[k][j]*=a.data[k][k];
    for (i=0;i<a.n;i++)
      if (i!=k)
        for (j=0;j<a.n;j++)
          if (j!=k)
            a.data[i][j]-=a.data[i][k]*a.data[k][j];
    for (i=0;i<a.n;i++)
      if (i!=k)
        a.data[i][k]*=-a.data[k][k];
  }
  for (k=a.n-1;k>=0;k--){
    for (j=0;j<a.n;j++)
      if (js[k]!=k)
        t=a.data[k][j],a.data[k][j]=a.data[js[k]][j],a.data[js[k]][j]=t;
    for (i=0;i<a.n;i++)
      if (is[k]!=k)
        t=a.data[i][k],a.data[i][k]=a.data[i][is[k]],a.data[i][is[k]]=t;
  }
  return 1;
}
//行列式的值注意n = m
double det(const mat& a){
  int i,j,k,sign=0;
  double b[MAXN][MAXN],ret=1,t;
  if (a.n!=a.m)
    return 0;
  for (i=0;i<a.n;i++)
    for (j=0;j<a.m;j++)
      b[i][j]=a.data[i][j];
  for (i=0;i<a.n;i++){
    if (zero(b[i][i])){
      for (j=i+1;j<a.n;j++)
        if (!zero(b[j][i]))
          break;
      if (j==a.n)
        return 0;
      for (k=i;k<a.n;k++)
        t=b[i][k],b[i][k]=b[j][k],b[j][k]=t;
      sign++;
    }
    ret*=b[i][i];
    for (k=i+1;k<a.n;k++)
      b[i][k]/=b[i][i];
    for (j=i+1;j<a.n;j++)
      for (k=i+1;k<a.n;k++)
        b[j][k]-=b[j][i]*b[i][k];
  }
  if (sign&1)
```

```
    ret=-ret;
  return ret;
}
```

8.4 Romberg积分

注意f(x),要先定义。有限区间内的单值、有界、连续函数(把积分变量和积分界取倒数,可以积单侧无穷, 当然前提也是要有界). 在通常的实现里面, 最大迭代次数是预先给定的, 如果超过这个次数还达不到收敛条件($R[0] - R[1] <= eps$), 就报错结束. 这个也可以看作是对积分格式的限制. 但是它是后验的, 积分格式复杂的时候貌似没有什么办法可以预先知道.

```
double f(double x) {//函数方程写在这里 }

double romberg(double a, double b) {
   std::vector<double> R;
   int k = -1;
   double r = 0.5 * (b - a) * (f(a) + f(b));
   R.push_back(r);
   do  {
      k += 1;
      r = 0.0;
      for (int i = 0; i < pow(2, k); i++) {
         r += f(a + (b - a) * (i + 0.5) / pow(2, k));
      }
      r *= (b - a) / pow(2, k + 1);
      r += 0.5 * R[k];
      R.push_back(r);
      for (int m = 0; m <= k; m++) {
         R[k - m] = (pow(4, m + 1) * R[k + 1 - m] - R[k - m]) /
            (pow(4, m + 1) - 1);
      }
   } while (fabs(R[0] - R[1]) > eps);
   return R[0];
}
```

8.5 线性规划单纯型算法linear programming simplex algorithm

线性规划LP :
min (CT)x
s.t.
Ax = B
x >= 0
(C, B, x为列向量, CT为c的转置)

[说明] 基可行解的计算采用的是大M单纯型法。未处理循环问题

```
const int DM = 128;
const double M = 1e8; // 相对无穷大
```

```
const double eps = 1e-10;

bool LP(int n, int m, double C[], double A[][DM],
        double B[], double x[], double &f) {
    int i, j;
    double sgm[DM], b[DM], a[DM][DM], c[DM];
    bool base[DM];
    for(i = 0; i < n; i++) c[i] = C[i];
    for(i = 0; i < m; i++) c[n+i] = M; //无穷大, 惩罚人工变量不要取的太大
    memset(b, 0, sizeof(b));
  for(i = 0; i < m; i++) b[i + n] = B[i];
  for(i = 0; i < m; i++)
     for(j = 0; j < n; j++) a[n + i][j] = A[i][j];
    for(i = 0; i < m; i++) a[n+i][n+i] = 1;
    memset(base, false, sizeof(base));
   for(i = 0; i < m; i++) base[n+i] = true; // 初始化基向量
   memset(x, 0, sizeof(x));
   for(i = 0; i < m+n; i++) x[i] = b[i]; // 初始化基可行解
   memset(sgm, 0, sizeof(sgm));
   for(i = 0; i < n; i++) {
      sgm[i] = c[i];
      for(j = 0; j < m; j++) sgm[i] -= a[n+j][i]*c[n+j];
   } // 初始化判别数
   f = 0;
   for(i = 0; i < m+n; i++) f += x[i] * c[i];
   while(true) {
      double minsgm = M; int k;
      for(i = 0; i < n; i++)
         if(minsgm > sgm[i]) { minsgm = sgm[i]; k = i; }
      if(minsgm > -eps) {
         for(i = 0; i < m; i++)
            if(fabs(x[i+n]) > eps) return false; // 算法结束, 此线性规划不存在最优解;
         if(i == m) return true;
      }
      int l = -1; x[k] = M;
      for(i = 0; i < m+n; i++) {
         if(!base[i] || a[i][k] < eps) continue;
         if(x[k] > b[i]/a[i][k]) { x[k] = b[i]/a[i][k]; l = i; }
      }
      if(l == -1) return false; // 算法结束, 此线性规划不存在最优解;
      for(i = 0; i < m+n; i++)
         if(base[i]) x[i] = b[i] - a[i][k] * x[k];
      base[l] = false; base[k] = true; f += x[k] * sgm[k];
      double na[DM][DM] = { 0 }, sk = sgm[k];
      for(i = 0; i < n+m; i++) {
         sgm[i] -= a[l][i] / a[l][k] * sk;
         if(!base[i]) continue;
         if(i == k) b[i] = b[l] / a[l][k];
         else b[i] -= b[l]/a[l][k]*a[i][k];
          for(j = 0; j < n+m; j++)
            if(i == k) na[i][j] = a[l][j] / a[l][k];
            else na[i][j] = a[i][j] - a[l][j]/a[l][k]*a[i][k];
      }
```

```
      memcpy(a, na, sizeof(a));
    }
    return true;
}
```

8.6 判线性相关(正交化)

[输入]m个n维向量

```
#define MAXN 100
#define eps 1e-10

int linear_dependent(int m, int n, double vec[][MAXN]) {
    double ort[MAXN][MAXN], e;
    int i, j, k;
    if (m > n) return 1;
    for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++) ort[i][j] = vec[i][j];
        for (k = 0; k < i; k++) {
            for (e = j = 0; j < n; j++) e += ort[i][j] * ort[k][j];
            for (j = 0; j < n; j++) ort[i][j] -= e * ort[k][j];
            for (e = j = 0; j < n; j++) e += ort[i][j] * ort[i][j];
            if (fabs(e = sqrt(e)) < eps) return 1;
            for (j = 0; j < n; j++) ort[i][j] /= e;
        }
    }
    return 0;
}
```

8.7 牛顿法解多项式的根

输入多项式系数c[]，多项式度数n，求在[a,b]间的根, 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;
}
```

8.8 高精度计算

如果对Java熟悉可以用Java解决。注意是用C++的输入输出是用cin,cout。对于几个高精度数和整型运算的函数要注意整型溢出的问题，必要时可以将DIGIT和DEPTH的值改小

```
typedef long long llong;
const int DIGIT = 4; //位数
const int BASE = 10000; //大数的进制
```

```
const int MAX = 30; //数组大小不能只计算当前，注意乘的结果位数会增加,
char buf[MAX * DIGIT];

struct BigInt {
    int Data[MAX], Len, sgn;
    BigInt(int v = 0){
        if(v == 0) { Len = 1; Data[1] = 0; }
        for( Len = 0;v;v /= BASE) Data[Len++] = v % BASE;
    }
    BigInt(const BigInt &V):Len(V.Len) {
        memcpy(Data, V.Data, Len * sizeof(int));
    }
    BigInt& operator=(int v) {
        if(v == 0) { Len = 1; Data[1] = 0; return *this;}
        for( Len = 0;v;v /= BASE) Data[Len++] = v % BASE;
        return *this;
    }
    BigInt &operator=(const BigInt &V) {
        Len = V.Len;
        memcpy(Data, V.Data, Len * sizeof(int));
        return * this;
    }
    int &operator[](int index) { return Data[index]; }
    int operator[](int index) const { return Data[index]; }
};

int comp(const BigInt &A, const BigInt &B) {
    if (A.Len != B.Len) return A.Len > B.Len ? 1 : -1;
    int i;
    for (i = A.Len - 1; i > 0 && A[i] == B[i]; i--);
    return A[i] - B[i];
}

BigInt operator+(const BigInt &A, const BigInt &B) {
    int i, Carry(0);
    BigInt R;
    for (i = 0; i < A.Len || i < B.Len || Carry > 0; i++) {
        if (i < A.Len) Carry += A[i];
        if (i < B.Len) Carry += B[i];
        R[i] = Carry % BASE;
        Carry /= BASE;
    }
    R.Len = i;
    return R;
}

BigInt operator-(const BigInt &A, const BigInt &B) {
    int i, Carry(0);
    BigInt R;
    R.Len = A.Len;
    for (i = 0; i < R.Len; i++) {
        R[i] = A[i] - Carry;
        if (i < B.Len) R[i] -= B[i];
```

```
        if (R[i] < 0) Carry = 1, R[i] += BASE;
        else Carry = 0;
    }
    while (R.Len > 1 && R[R.Len - 1] == 0) R.Len--;
    return R;
}

BigInt operator*(const BigInt &A, const int &B) {
    int i;
    llong Carry(0);
    BigInt R;
    for (i = 0; i < A.Len || Carry > 0; i++) {
        if (i < A.Len) Carry += llong(A[i]) * B;
        R[i] = Carry % BASE;
        Carry /= BASE;
    }
    R.Len = i;
    return R;
}

BigInt operator*(const BigInt &A, const BigInt &B) {
    BigInt inc(1), ret(0);
    int i, j;
    llong Carry(0);
    for (i = 0; i < A.Len; ++i) {
        for (j = Carry = 0; j < B.Len || Carry > 0; ++j, Carry /= BASE) {
            if (j < B.Len) Carry += llong(A[i]) * B[j];
            if (i + j < ret.Len) Carry += ret[i + j];
            if (i + j >= ret.Len) ret[ret.Len++] = Carry % BASE;
            else ret[i + j] = Carry % BASE;
        }
    }
    return ret;
}

//另一种*的重载
BigInt operator*(const BigInt &A, const BigInt &B) {
    BigInt inc(1), ret(0);
    for (int i = 0; i < B.Len; ++i) {
        inc = A * B.Data[i];
        for (int j = 0; j < i; ++j) inc = inc * BASE;
        ret = ret + inc;
    }
    return ret;
}

BigInt operator/(const BigInt &A, const BigInt &B) {
    BigInt tmp, mod(0), ret;
    int i, lf, rg, mid;
    for (i = A.Len - 1; i >= 0; --i) {
        mod = mod * BASE + A[i];
        for (lf = 0, rg = BASE - 1; lf - rg;) {
            mid = (lf + rg + 1) >> 1;
```

```
            if (comp(B * mid, mod) <= 0) lf = mid;
            else rg = mid - 1;
        }
        ret[i] = lf;
        mod = mod - B * lf;
    }
    ret.Len = A.Len;
    while (ret.Len > 1 && ret[ret.Len - 1] == 0)--ret.Len;
    return ret; //return mod 就是%运算
}

BigInt operator/(const BigInt& A, const int B)
{
    BigInt R(0);
    int I;
    llong C(0);
    for (I = A.Len - 1; I >= 0; I--)  {
        C = C * Base + A[I];
        R[I] = C / B;
        C %= B;
    }
    R.Len = A.Len;
    while (R.Len > 1 && R[R.Len - 1] == 0) R.Len--;
    return R;
}


int digits(const BigInt &A) {
    if (A.Len == 0) return 0;
    int l = (A.Len - 1) * DIGIT;
    for (int t = A[A.Len - 1]; t; ++l, t /= 10);
    return l;
}

istream & operator>>(istream &In, BigInt &V) {
    char Ch;
    for (V = 0; In >> Ch;) {
        V = V * 10 + (Ch - '0');
        if (In.peek() <= '␣') break;
    }
    return In;
}

ostream & operator<<(ostream &Out, const BigInt &V) {
    int i;
    Out << (V.Len == 0 ? 0 : V[V.Len - 1]);
    for (i = V.Len - 2; i >= 0; i--)
        for (int j = BASE / 10; j > 0; j /= 10) Out << V[i] / j % 10;
    return Out;
}

int read(BigInt &A, char buf[]) { //读取失败返回0
    if (1 != scanf("␣%s", buf)) return 0;
```

```
    int w, u, v, ln = strlen(buf);
    memset(&A, 0, sizeof (BigInt));
    for (v = 0; v < ln - 1 && buf[v] == '0'; ++v);
    if (buf[v] == '-') A.sgn = -1, ++v;
    else A.sgn = 1;

    for (w = 1, u = 0; ln > v;) {
        u += (buf[--ln] - '0') * w;
        if (w * 10 == BASE) A[A.Len++] = u, u = 0, w = 1;
        else w *= 10;
    }
    if (w != 1) A[A.Len++] = u;
    return 1;
}

void write(const BigInt &A) {
    int i;
    printf("%d", A[A.Len - 1]);
    for (i = A.Len - 2; i >= 0; --i) printf("%0*d", DIGIT, A[i]);
}
```

8.9 分数运算

```
struct frac {
    int num, den;
};

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

int gcd(int a, int b) {
    int t;
    if (a < 0) a = -a;
    if (b < 0) b = -b;
    if (!b) return a;
    while (t = a % b) a = b, b = t;
    return b;
}

void simplify(frac& f) {
    int t;
    if (t = gcd(f.num, f.den)) f.num /= t, f.den /= t;
    else f.den = 1;
}

frac f(int n, int d, int s = 1) {
    frac ret;
    if (d < 0) ret.num = -n, ret.den = -d;
    else ret.num = n, ret.den = d;
    if (s) simplify(ret);
```

```
    return ret;
}

frac convert(double x) {
    frac ret;
    for (ret.den = 1; fabs(x - int(x)) > 1e-10; ret.den *= 10, x *= 10);
    ret.num = (int) x;
    simplify(ret);
    return ret;
}

int fraqcmp(frac a, frac b) {
    int g1 = gcd(a.den, b.den), g2 = gcd(a.num, b.num);
    if (!g1 || !g2) return 0;
    return b.den / g1 * (a.num / g2) - a.den / g1 * (b.num / g2);
}

frac add(frac a, frac b) {
    int g1 = gcd(a.den, b.den), g2, t;
    if (!g1) return f(1, 0, 0);
    t = b.den / g1 * a.num + a.den / g1 * b.num;
    g2 = gcd(g1, t);
    return f(t / g2, a.den / g1 * (b.den / g2), 0);
}

frac sub(frac a, frac b) {
    return add(a, f(-b.num, b.den, 0));
}

frac mul(frac a, frac b) {
    int t1 = gcd(a.den, b.num), t2 = gcd(a.num, b.den);
    if (!t1 || !t2) return f(1, 1, 0);
    return f(a.num / t2 * (b.num / t1), a.den / t1 * (b.den / t2), 0);
}

frac div(frac a, frac b) {
    return mul(a, f(b.den, b.num, 0));
}
```

8.10 行列式$mod\ m$的值

[说明] Guass消元法+欧几里得:求解矩阵行列式$mod\ m$的值
复杂度$O(n^3logn)$
m可以为任何数$(m > 0)$
注意可能出现的溢出的情况
$mat[][]$ -方阵数据
n -方阵大小
m -取mod的数

```
typedef long long llong;
template <class T>inline T MOD(T A,T B){A%=B;if(A<0)A+=B;return A;}
template <class T>inline void SWAP(T &A,T &B){T buf=A;A=B;B=buf;}
#define N 210 int mat[N][N];
int guass(int mat[][N],int n,int m) {
    if(m==1)return 0;
    llong ans=1,t,tmp;
    int i,j,k;
    for(i=0;i<n;++i)
        for(j=0;j<n;++j)
            mat[i][j]=MOD(mat[i][j],m);
    for(i=0;i<n;++i)
    {
        for(j=i+1;j<n;++j)
        {
            while(mat[j][i])
            {
                t=mat[i][i]/mat[j][i];
                for(k=0;k<n;++k)
                {
                    tmp=mat[i][k];
                    tmp-=t*mat[j][k];
                    tmp=MOD(tmp,(llong)m);
                    mat[i][k]=tmp;
                }
                for(k=0;k<n;++k)SWAP(mat[i][k],mat[j][k]);
                ans=-ans;
            }
        }
        if(mat[i][i]==0)return 0;
        ans=(ans*mat[i][i]%m+m)%m;
    }
    return ans;
}
```

8.11 FFT a*b

[说明] $a*b$的FFT 代码

```
#include<cmath>
#include<cstdio>
#include<cstring>
using namespace std;
const int BASE = 100000;
const int N_DIGIT = 5;
const int N = 32768*16;
const double PI = acos(-1.0);
struct Complex {
    double real, imag;
};
```

```
Complex omega[N>>1];
Complex a[N];
Complex b[N];
char s[1000003];
int d[N], len;
void bitReverse(Complex a[]) {
    int i, j = 1, k;
    Complex t;
    for (i = 1; i < len; ++i) {
        if (i < j) {
            t.real = a[j-1].real;
            t.imag = a[j-1].imag;
            a[j-1].real = a[i-1].real;
            a[j-1].imag = a[i-1].imag;
            a[i-1].real = t.real;
            a[i-1].imag = t.imag;
        }
        k = len >> 1;
        while (k < j) {
            j -= k;
            k >>= 1;
        }
        j += k;
    }
}
void calOmega() {
    double unit = 2 * PI / len;
    int n = len >> 1;
    for (int i = 0; i < n; ++i) {
        double t = unit * i;
        omega[i].real = cos(t);
        omega[i].imag = sin(t);
    }
}
void fft(Complex a[], bool inverse = false) {
    bitReverse(a);
    int s = len >> 1;
    int m, k, j;
    int up, t, step;
    int i1, i2;
    Complex tmp;
    if (inverse)for (j = 0; j < s; ++j)omega[j].imag = -omega[j].imag;
    s = 1;
    for (m = 2; m <= len; m <<= 1) {
        up = m >> 1, t = len >> s;// 2^(log2(n) - s) != n - 2^s  !!!!!!
        for (k = 0; k < len; k += m) {
            step = 0;
            for (j = 0; j < up; ++j) {
                i1 = k + j;
                i2 = i1 + up;
                tmp.real = omega[step].real * a[i2].real - omega[step].imag * a[i2].imag;
                tmp.imag = omega[step].real * a[i2].imag + omega[step].imag * a[i2].real;
                a[i2].real = a[i1].real - tmp.real;
```

```
                a[i2].imag = a[i1].imag - tmp.imag;
                a[i1].real += tmp.real;
                a[i1].imag += tmp.imag;
                step += t;
            }
        }
        ++s;
    }
    if (inverse) {
        double t = 1.0 / len;
        for (j = 0; j < len; ++j)a[j].real *= t;
    }
}
int convert(int d[], char s[]) {
    int sLen = strlen(s);
    int dLen = ((sLen - 1) / N_DIGIT) + 1, i = 0, n;
    char *pRight = s + sLen - 1, *pLeft = pRight - (N_DIGIT - 1);
    memset(d, 0, sizeof(int)*dLen);
    while (i<dLen && pRight >= s) {
        if (pLeft < s) pLeft = s;
        n = 0;
        while (pLeft <= pRight) {
            n = n * 10 + (*pLeft & 15);
            ++pLeft;
        }
        d[i++] = n;
        pRight -= N_DIGIT;
        pLeft = pRight - (N_DIGIT - 1);
    }
    return dLen;
}
bool init() {
    int i, j;//read a
    if (scanf("%s", s) != 1)return false;
    int aLen = convert(d, s);//length of a
    for (i = 0; i < aLen; ++i) {
        a[i].real = d[i];
        a[i].imag = 0;
    }//read b
    scanf("%s", s);
    int bLen = convert(d, s);//length of b
    for (j = 0; j < bLen; ++j) {
        b[j].real = d[j];
        b[j].imag = 0;
    }
    len = 1;//length of product who uses int
    while (len < aLen + bLen) len <<= 1;
    memset(a + i, 0, sizeof(Complex) *(len - i));
    memset(b + j, 0, sizeof(Complex) *(len - j));
    calOmega();
    return true;
}
void mul() {
```

```
    for (int i = 0; i < len; ++i) {
        double real = a[i].real * b[i].real - a[i].imag * b[i].imag;
        double imag = a[i].real * b[i].imag + a[i].imag * b[i].real;
        a[i].real = real;
        a[i].imag = imag;
    }
}
void print() {
    double carry = 0, t;
    static char format[10];
    int i;
    for (i = 0; i < len; ++i) {
        t = carry + a[i].real;
        carry = floor((t + 0.5) / BASE);
        d[i] = int(t - carry * BASE + 0.5);
    }
    for (i = len - 1; i > 0 && d[i] == 0; --i);
    sprintf(format, "%%.%dd", N_DIGIT);
    printf("%d", d[i]);
    for (--i; i >= 0; --i)printf(format, d[i]);
    printf("\n");
}
int main() {
    int t;
    scanf("%d", &t);
    while (init()) {
        fft(a);
        fft(b);
        mul();
        fft(a, true);
        print();
    }
    return 0;
}
```

第九章 博弈论

9.1 经典博弈

【巴什博弈】只有一堆n个物品，两个人轮流从这堆物品中去物，规定每次最少取一个，最多取m个。最后取光者得胜。当$n\%\left(m+1\right)\neq 0$时，先手必胜。当$n\%\left(m+1\right)==0$时，先手必败。

【威佐夫博弈】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。(poj 1067 取石子游戏) 设两堆石子个数为$a,b(a\geq b)$ 则：

```
double alpha = (1.0 + sqrt(5.0)) / 2.0 , beta = (3.0 + sqrt(5.0)) / 2.0;
int k = ceil(a / beta) , num1 = alpha * k , num2 = beta * k;
if(num1 == b&&num2 == a) 先手必败。
else 先手必胜。
```

9.2 Sprague－Grundy 函数

$$g\left(x\right)=\min\{n\geq 0:n\neq g\left(y\right)for\ y\in \mathbf{F}\left(x\right)\} \tag{9.2.1}$$

[说明] 求SG函数递归写法 Take-Away Game

```
int sg[N];    //初始化都为-1
int lim[M];   //限制取的石子数
int n,m;   //石子数,可取石子集合大小

int mex(int pos) {
    if (sg[pos]!=-1) return sg[pos];
    int &res=sg[pos];
    bool flag[M]={0};
    int i;
    for (i=0;i<m;i++)
    {
        if (pos-lim[i]<0) continue;
        flag[mex(pos-lim[i])]=true;
    }
    for (i=0;flag[i];i++);
    return res=i;
}
```

[说明] 非递归写法

```
//非递归,状态从小到大枚举
//亦可以不初始化为-1

int sg[N], lim[M],flag[M], i,j;

memset(sg,-1,sizeof(sg));
memset(flag,-1,sizeof(flag));

for (i=0;i<N;i++) {
    for (j=0;j<M;j++)
    {
        if (i-lim[j]<0) continue;
        flag[sg[i-lim[j]]]=i; //当前标记
    }
    for (j=0;flag[j]==i;j++);
    sg[i]=j;
}
```

[注意]如果数据范围较大,可以对据定义小规模暴力求sg值,猜测规律

9.3 Nim-Multiplication

[说明] Nim积性质

- $0 \otimes x = x \otimes 0 = 0$
- $1 \otimes x = x \otimes 1 = x$
- $x \otimes y = y \otimes x$
- $x \otimes (y \otimes z) = (x \otimes y) \otimes z$
- $x \otimes (y \oplus z) = (x \otimes y) \oplus (y \otimes z)$
- 一个$Fermat\ 2-power$和一个比它小的数的Nim积就是它们在一般意义下的乘积。
- 一个$Fermat\ 2-power$和它自己的Nim积是一般意义下的乘积的3/2

[说明] 通常成2^{2^n}这种形式的数为$Fermat\ 2-power$。

```
int sg[32][32];
int sgProd(int x,int y);
int sgProd2(int x,int y) {
    if(x < y) swap(x , y);
    if(sg[x][y] != -1) return sg[x][y];
    if((x&(x-1)) == 0) {
        if(x == y) return sg[x][y] = (1<<x) * 3 / 2;
        return sg[x][y] = (1<<x) * (1<<y);
    }
    int ret = 1;
    for (int i = 1 ; i <= x; i <<= 1) {
```

```
        if((x&i) && (y&i)) {
            ret = sgProd((1<<i) * 3 / 2 , ret);
        } else if((x&i) || (y&i)){
            ret = sgProd(1<<i , ret);
        }
    }
    return sg[x][y] = ret;
}
int sgProd(int x,int y)  //调用sgProd(x,y)求x,y的nim积
 {
    if(x == 0 || y == 0) return 0;
    if(x == 1 || y == 1) return x + y - 1;
    int ret = 0;
    for (int i = 0 ; 1<<i <= x ; i ++) if(x & (1<<i)) {
        for (int j = 0 ; 1<<j <= y ; j ++) if(y & (1<<j)) {
            ret ^= sgProd2(i,j);
        }
    }
    return ret;
}
```

[注意]注意sg值是否会超过int,为保险起见可以开$long\ long\ sg$[64][64]

9.4 常见限制条件下的SG值

- 最多取m个：$SG(n) = n\%(m+1)$
- 只能取奇数个：$SG(n) = n\%2$
- 只能取2^i个：$SG(n) = n\%3$
- 只能取p^i个(p为大于2的素数)：$SG(n) = n\%2$

第十章　经典问题

【魔方旋转-置换群-pku 1955】

```
//魔方面顺序为
/// 4
//0 1 2 3
/// 5

 #include <iostream.h>
 #include <string.h>

int d1[6][20]={
    {12,24,36,35,34,22,10,11,52,49,46,37,25,13,7,4,1,21,33,45},
    {15,14,27,39,38,37,25,13,46,47,48,40,28,16,9,8,7,12,24,36},
    {18,30,42,41,40,28,16,17,48,51,54,43,31,19,3,6,9,15,27,39},
    {19,20,21,33,45,44,43,31,54,53,52,34,22,10,1,2,3,18,30,42},
    {1,2,3,6,9,8,7,4,13,14,15,16,17,18,19,20,21,10,11,12},
    {46,47,48,51,54,53,52,49,45,44,43,42,41,40,39,38,37,36,35,34}
};
int d2[6][20]={
    {10,11,12,24,36,35,34,22,37,25,13,7,4,1,21,33,45,52,49,46},
    {13,25,14,15,27,39,38,37,40,28,16,9,8,7,12,24,36,46,47,48},
    {16,17,18,30,42,41,40,28,43,31,19,3,6,9,15,27,39,48,51,54},
    {43,31,19,20,21,33,45,44,34,22,10,1,2,3,18,30,42,54,53,52},
    {7,4,1,2,3,6,9,8,16,17,18,19,20,21,10,11,12,13,14,15},
    {52,49,46,47,48,51,54,53,42,41,40,39,38,37,36,35,34,45,44,43}
};

char cube[55],tmp[55];

int main() {
    int t,c=1,i,j,p,face,turn; cin>>t;
    while(t--) {
        for(i=1;i<=54;i++) cin>>cube[i];
        cin>>p;
        for(i=0;i<p;i++) {
            cin>>face>>turn;
            memcpy(tmp,cube,55*sizeof(char));
            if(turn>0) //第face面正转90度
                for(j=0;j<20;j++) cube[d1[face][j]]=tmp[d2[face][j]];
            else  //第face面反转90度
                for(j=0;j<20;j++) cube[d2[face][j]]=tmp[d1[face][j]];
        }
        cout<<"Scenario␣#"<<c<<":\n";
```

```
        c++;
        for(i=1;i<=9;i++) {
            if((i-1)%3==0) cout<<"␣␣␣␣␣";
            cout<<"␣"<<cube[i];
            if(i%3==0) cout<<endl;
        }
        for(i=10;i<=45;i++) {
            if(i!=10 && i!=22 && i!=34) cout<<"␣";
            cout<<cube[i];
            if(i==21 || i==33 || i==45) cout<<"\n";
        }
        for(i=46;i<=54;i++) {
            if((i-1)%3==0) cout<<"␣␣␣␣␣";
            cout<<"␣"<<cube[i];
            if(i%3==0) cout<<endl;
        }
        cout<<"\n";
    }
    return 0;
}
```

【 两堆点，求一平面，将这二堆点分开- pku 3643】

迭代方法

$if(a * x[i] + b * y[i] + c * z[i] + d > 0)$

如果$x[i]$ 为正数，则 $a-=x[i]，a$减小，则$a * x[i]$ 减小

如果$x[i]$ 为负数，则 $a-=x[i]，a$增加，反而 $a * x[i]$ 减小

同样道理对于$if(a * x[i] + b * y[i] + c * z[i] + d <= 0)$

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

int main() {
    double x[202], y[202], z[202];
    double a, b, c, d;
    int r, e, n, i, j, k, flag;
    while (scanf("%d", &r) != EOF && r != -1) {
        for (i = 0; i < r; ++i)
            scanf("%lf␣%lf␣%lf", &x[i], &y[i], &z[i]);
        scanf("%d", &e);
        //第一堆点r个，第二堆点e个
        n = e + r;
        for (i = r; i < n; ++i)
            scanf("%lf␣%lf␣%lf", &x[i], &y[i], &z[i]);
        a = b = c = d = 0;
        flag = 0;
        for (k = 0; k < 50000 && !flag; ++k) {
            //50000,迭代次数,可根据时间调整
            flag = 1;
            for (i = 0; i < r; ++i)
                if (a * x[i] + b * y[i] + c * z[i] + d > 0)
                    a -= x[i], b -= y[i], c -= z[i], d -= 1, flag = 0;
            for (i = r; i < n; ++i)
```

```
                if (a * x[i] + b * y[i] + c * z[i] + d <= 0)
                    a += x[i], b += y[i], c += z[i], d += 1, flag = 0;
        }
        printf("%lf␣%lf␣%lf␣%lf\n", a, b, c, d);

    }
}
```

【求多边形最小切圆– pku 3525, sgu 332】通过二分答案再加上半平面交方面求得其半径长度$O(n^2logn)$
也可以枚举多边形上的三条边，求其三角形内切圆，取最小者即为半径。但这么做是$O(n^3)$
此程序sgu过不了－_

```
TPoint poly[N], new_poly[N], temp_poly[N];
int main() {
    int i, j, k, nn, n, t;
    double up, low, mid;
    TPoint st, ed, interp;
    TSegment seg0, seg1, seg2;
    while (scanf("%d", &n) != EOF && n != 0) {
        up = -1e8; low = 1e8;
        for (i = 0; i < n; ++i) {
            scanf("%lf␣%lf", &poly[i].x, &poly[i].y);
            if (up < poly[i].x) up = poly[i].x;
            if (low > poly[i].x) low = poly[i].x;
        }
        up = (up < low ? (up * 0.5) : (low * 0.5)) + EPS;
        low = 0.000;
        poly[n] = poly[0];

        memcpy(new_poly, poly, sizeof (poly[0]) * (n + 1));
        while (up - low > EPS) {
            memcpy(new_poly, poly, sizeof (poly[0]) * (n + 1));
            nn = n;
            mid = (up + low + EPS) * 0.5;
            for (i = 0; i < n; ++i) {
                seg0.s = poly[i], seg0.e = poly[i + 1];
                get_dis_parall(mid, seg0, seg1, seg2);
                if (cross(poly[i + 1] - poly[i], seg1.s - poly[i]) > -EPS)
                    st = seg1.s, ed = seg1.e;
                else st = seg2.s, ed = seg2.e;
                t = 0;
              for (j = k = 0; j < nn; ++j) {
                if (cross(ed - st, new_poly[j] - st) > -EPS)
                    temp_poly[k++] = new_poly[j];
                if (t < 2 && fabs(cross(ed - st, new_poly[j + 1] - new_poly[j])) > EPS) {
                    interp = seg_inter_pnt(st, ed, new_poly[j], new_poly[j + 1]);
                      if (on_segment(interp, new_poly[j], new_poly[j + 1]))
                       temp_poly[k++] = interp, t++;
                      }
                }
                if (area(temp_poly, k) < 1e-10) break;
                memcpy(new_poly, temp_poly, k * sizeof (poly[0]));
```

```
                nn = k;
                new_poly[nn] = new_poly[0];
            }
            if (i < n) up = mid;
            else low = mid + EPS;
        }
        printf("%.6lf\n", low + EPS);
    }
    return 0;
}
```

【多边形面积并-pku 3695】

```
//暴力版O(n^2)
const int UP = 1, DOWN = -1;

struct TSeg {
    int l, r, y;
    int dirt;
    bool operator<(const TSeg b) const {
        return y < b.y;
    }
    void set(int ll, int rr, int yy, int d) {
        l = ll; r = rr; y = yy; dirt = d;
    }
};

struct TRect {
    int l, r, b, t;
    void set(int l0, int r0, int b0 , int t0) {
        l = l0; r = r0; b = b0; t = t0;
    }
};
TRect rect[25];
TSeg seg[50];
int x[50];

int main()
{
    int n, m, i, j, k, q, cas = 1, segn, xn, t;
    while(scanf("%d␣%d", &n, &m) != EOF) {
        if(n == 0 && m == 0) break;
        printf("Case␣%d:\n", cas++);
        for(i = 1;i <= n;++ i)
            scanf("%d␣%d␣%d␣%d", &rect[i].l, &rect[i].b, &rect[i].r, &rect[i].t);
        for(q = 1;q <= m;++ q) {
            scanf("%d", &t);
            xn = segn = 0;
            for(i = 0;i < t;++ i) {
                scanf("%d", &k);
                seg[segn++].set(rect[k].l, rect[k].r, rect[k].t, UP);
                seg[segn++].set(rect[k].l, rect[k].r, rect[k].b, DOWN);
                x[xn++] = rect[k].l, x[xn++] = rect[k].r;
            }
```

```
            sort(x, x + xn);
            for(i = j = 0;i < xn;++ j) {
                k = i;
                while(k < xn && x[i] == x[k+1]) ++ k;
                x[j] = x[i];
                i = k + 1;
            }
            xn = j;
            sort(seg, seg + segn);
            int area = 0;
            for(i = 0;i < xn - 1;++ i) {
                int cnt = 0;
                int lowy;
                for(j = 0;j < segn;++ j) {
                    if(seg[j].l <= x[i] && seg[j].r >= x[i+1]) {
                        if(cnt == 0) lowy = seg[j].y;
                        cnt += seg[j].dirt;
                        if(cnt == 0) area += (seg[j].y - lowy) * (x[i+1] - x[i]);
                    }
                }
            }
            printf("Query␣%d:␣%d\n", q, area);
        }
        puts("");
    }
    return 0;
}

//线段树版本O(nlogn)
const int UP = 1, DOWN = -1;
const int N = 200;

struct TRect { int l, r, t, b; };
struct NODE { int bot, top, len, cover;};
struct TSeg {
    int x, top, bot, dirt;
    void set(int x0, int top0, int bot0, int dirt0) {
        x = x0; top = top0; bot = bot0; dirt = dirt0;
    }
    bool operator<(const TSeg &b) const {
        return x < b.x;
    }
};

TRect rect[N];
TSeg seg[N];
NODE node[N];
int ys[N], cnt;

void calc_len(int idx)
{
    if(node[idx].cover > 0) node[idx].len = node[idx].top - node[idx].bot;
    else {
```

```
        int t = (idx << 1) + 2, b = (idx << 1) + 1;
        node[idx].len = node[t].len + node[b].len;
    }
}

void build_tree(int idx, int bot, int top)
{
    memset(node + idx, 0, sizeof(node[0]));
    node[idx].top = ys[top];
    node[idx].bot = ys[bot];
    if(bot + 1 < top) {
        int mid = (bot + top)>>1;
        build_tree((idx << 1) + 1, bot, mid);
        build_tree((idx << 1) + 2, mid, top);
    }
}

void update(TSeg &seg, int idx)
{
    if(seg.top == node[idx].top && seg.bot == node[idx].bot) {
        node[idx].cover += seg.dirt;
        calc_len(idx);
        return ;
    }
    int t = (idx << 1) + 2, b = (idx << 1) + 1;
    if(seg.bot < node[b].top) {
        if(node[b].top >= seg.top) update(seg, b);
        else {
            TSeg t_seg = seg;
            t_seg.top = node[b].top;
            update(t_seg, b);
            t_seg = seg;
            t_seg.bot = node[t].bot;
            update(t_seg, t);
        }
    }
    else update(seg, t);
    calc_len(idx);
}

int main()
{
    int n, m, i, j, k, q, cas = 1, ans, r;
    while(scanf("%d␣%d", &n, &m) != EOF) {
        if(n == 0 && m == 0) break;
        for(i = 1;i <= n;++ i) {
            scanf("%d␣%d␣%d␣%d", &rect[i].l, &rect[i].b, &rect[i].r, &rect[i].t);
            if(rect[i].l > rect[i].r) swap(rect[i].l, rect[i].r);
            if(rect[i].b > rect[i].t) swap(rect[i].b, rect[i].t);
        }
        printf("Case␣%d:\n", cas ++);
        for(q = 1;q <= m;++ q) {
            scanf("%d", &r);
```

```
            cnt = ans = 0;
            for(j = 0;j < r;++ j) {
                scanf("%d", &i);
                seg[cnt].set(rect[i].l, rect[i].t, rect[i].b, UP);
                seg[cnt+1].set(rect[i].r, rect[i].t, rect[i].b, DOWN);
                ys[cnt++] = rect[i].b;
                ys[cnt++] = rect[i].t;
            }
            sort(seg, seg + cnt);
            sort(ys, ys + cnt);

            build_tree(0, 0, cnt - 1);
            update(seg[0], 0);
            for(i = 1;i < cnt;++ i) {
                ans += (seg[i].x - seg[i-1].x) * node[0].len;
                update(seg[i], 0);
            }
            printf("Query␣%d:␣%d\n", q, ans);
        }
        puts("");
    }
    return 0;
}
```

【多边形并求周长-pku 1177】

```
const int IN = 1, OUT = -1;
struct TSeg{
    int x, y1, y2, dirt;
    void set(int x0, int y10, int y20, int d0) {
        x = x0; y1 = y10; y2 = y20; dirt = d0;
    }
    bool operator<(const TSeg b) const {
        return x < b.x;
    }
};
TSeg sg[10005];
int rect[5003][4];
int dx[10005];

int main()
{
    int n, i, j, k, x1, y1, x2, y2, m, cnt;
    while(scanf("%d", &n) != EOF) {
        for(i = j = k = 0;i < n;++ i) {
            scanf("%d␣%d␣%d␣%d", &rect[i][0], &rect[i][1], &rect[i][2], &rect[i][3]);
            sg[j++].set(rect[i][1], rect[i][0], rect[i][2], IN);
            sg[j++].set(rect[i][3], rect[i][0], rect[i][2], OUT);
            dx[k++] = rect[i][0]; dx[k++] = rect[i][2];
        }
        sort(dx, dx + k);
        sort(sg, sg + k);

        for(i = m = 0;i < k;++ i)
```

```
            if(!i || dx[i] != dx[i-1])
               dx[m++] = dx[i];


        int ans = 0;
        for(cnt = 0, i = 1;i < m;++ i) {
            for(j = 0;j < k;++ j) {
                if(sg[j].y1 <= dx[i-1] && sg[j].y2 >= dx[i]) {
                    if(cnt == 0) ans += dx[i] - dx[i-1];
                    cnt += sg[j].dirt;
                }
            }
        }

        for(i = j = k = 0;i < n;++ i) {
            sg[j++].set(rect[i][0], rect[i][1], rect[i][3], IN);
            sg[j++].set(rect[i][2], rect[i][1], rect[i][3], OUT);
            dx[k++] = rect[i][1]; dx[k++] = rect[i][3];
        }
        sort(dx, dx + k);
        sort(sg, sg + k);
         for(i = m = 0;i < k;++ i)
            if(!i || dx[i] != dx[i-1])
               dx[m++] = dx[i];
        for(cnt = 0, i = 1;i < m;++ i) {
            for(j = 0;j < k;++ j) {
                if(sg[j].y1 <= dx[i-1] && sg[j].y2 >= dx[i]) {
                    if(cnt == 0) ans += dx[i] - dx[i-1];
                    cnt += sg[j].dirt;
                }
            }
        }
        printf("%d\n", ans*2);
    }
    return 0;
}

//线段树版本,抄别人的，写得不是很好

const int MAX = 10010;
using namespace std;

struct NODE {
    int left, right;
    NODE *lchild, * rchild;
    int len, cover;
    int m, line;
    bool lcover, rcover;
    NODE() {
        lchild = rchild = NULL;
        lcover = rcover = false;
        m = line = 0;
    }
```

```
} root;

struct LINE {
    int top, end, x;
    bool tag;
    bool operator<(const LINE b) const {
        return x < b.x;
    }
} line[MAX + 1];

int temp[MAX + 1], index[MAX + 1], len;

void getm(NODE* node) {
    if (node->cover > 0) node->m = node->len;
    else {
        if (node->right - node->left > 1) {
            node->m = node->lchild->m + node->rchild->m;
        } else {
            node->m = 0;
        }
    }
    return;
}

void getline(NODE* node) {
    if (node->cover > 0) {
        node->lcover = node->rcover = true;
        node->line = 1;
    } else if (node->right - node->left > 1) {
        node->lcover = node->lchild->lcover;
        node->rcover = node->rchild->rcover;
        node->line = node->lchild->line + node->rchild->line -
                node->rchild->lcover * node->lchild->rcover;
    } else {
        node->lcover = node->rcover = false;
        node->line = 0;
    }
    return;
}

int getindex(int y) {
    int i;
    for (i = 0; i < len; ++i)
        if (y == index[i])return i;
}

void build(NODE* node, int l, int r) {
    node->right = r, node->left = l;
    node->len = index[r] - index[l];
    node->cover = 0;
    if (node->right - node->left > 1) {
        node->lchild = new NODE;
        node->rchild = new NODE;
```

```
        build(node->lchild, l, (l + r) / 2);
        build(node->rchild, (l + r) / 2, r);
    }
    return;
}

void insert(NODE* node, int l, int r) {
    if (l <= node->left && r >= node->right) node->cover++;
     else {
        int mid = (node->left + node->right) / 2;
        if (l < mid)insert(node->lchild, l, r);
        if (r > mid)insert(node->rchild, l, r);
    }
    getm(node);
    getline(node);
    return;
}

void del(NODE* node, int l, int r) {
    if (l <= node->left && r >= node->right) node->cover--;
    else {
        int mid = (node->left + node->right) / 2;
        if (l < mid)del(node->lchild, l, r);
        if (r > mid)del(node->rchild, l, r);
    }
    getm(node);
    getline(node);
    return;
}


int main() {
    int n, i, j;
    int x1, y1, x2, y2;
    int ans = 0, m = 0;
    scanf("%d", &n);
    for (i = 0; i < n; ++i) {
        scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
        line[2 * i].x = x1, line[2 * i + 1].x = x2;
        line[2 * i].top = line[2 * i + 1].top = y1;
        line[2 * i].end = line[2 * i + 1].end = y2;
        line[2 * i].tag = true, line[2 * i + 1].tag = false;
        temp[2 * i] = y1, temp[2 * i + 1] = y2;
    }
    sort(temp, temp + 2 * n);
    sort(line, line + 2 * n);
    index[0] = temp[0];
    for (i = 1, len = 1; i < 2 * n; ++i) {
        if (temp[i] != temp[i - 1])
            index[len++] = temp[i];
    }
    build(&root, 0, len - 1);
    for (i = 0; i < 2 * n - 1; ++i) {
```

```
        if (line[i].tag == true) {
            insert(&root, getindex(line[i].top), getindex(line[i].end));
        } else {
            del(&root, getindex(line[i].top), getindex(line[i].end));
        }

        ans += root.line * (line[i + 1].x - line[i].x)*2;
        ans += abs(root.m - m);
        m = root.m;
    }

    del(&root, getindex(line[i].top), getindex(line[i].end));
    ans += abs(root.m - m);
    printf("%d\n", ans);

    return 0;
}
```

【幻方构造】幻方，即将自然数1，2，3，……n*n排列成一个n*n方阵，使得每行、每列以及两对角线上的各个数之和都相等，等于n/2*(n*n+1)
幻方构造(l!=2), 只有偶数才有幻方

```
#define MAXN 100

void dllb(int l, int si, int sj, int sn, int d[][MAXN]) {
    int n, i=0, j=l/2;
    for(n=1;n<=l*l;++n) {
        d[i+si][j+sj] = n + sn;
        if(n % l) { i=(i)?(i-1):(l-1);j=(j==l-1)?0:(j+1); }
        else i=(i==l-1)?0:(i+1);
    }
}
void magic_odd(int l,int d[][MAXN]){    dllb(l,0,0,0,d); }

void magic_4k(int l,int d[][MAXN]){
    int i,j;
    for (i=0;i<l;i++)
        for (j=0;j<l;j++)
            d[i][j]=((i%4==0||i%4==3)&&(j%4==0||j%4==3)||
        (i%4==1||i%4==2)&&(j%4==1||j%4==2))?(l*l-(i*l+j)):(i*l+j+1);
}

void magic_other(int l,int d[][MAXN]){
    int i,j,t;
    dllb(l/2,0,0,0,d);
    dllb(l/2,l/2,l/2,l*l/4,d);
    dllb(l/2,0,l/2,l*l/2,d);
    dllb(l/2,l/2,0,l*l/4*3,d);
    for (i=0;i<l/2;i++)
        for (j=0;j<l/4;j++)
            if (i!=l/4||j)
                t=d[i][j],d[i][j]=d[i+l/2][j],d[i+l/2][j]=t;
    t=d[l/4][l/4],d[l/4][l/4]=d[l/4+l/2][l/4],d[l/4+l/2][l/4]=t;
    for (i=0;i<l/2;i++)
```

```
        for (j=l-l/4+1;j<l;j++)
            t=d[i][j],d[i][j]=d[i+l/2][j],d[i+l/2][j]=t;
}

void generate(int l,int d[][MAXN]){
    if (l%2) magic_odd(l,d);
    else if (l%4==0) magic_4k(l,d);
    else magic_other(l,d);
}
```

【度限制生成树- pku 1639】希望哪位勇士能够把这个程序写成规范化的模板

```
#define N 40
#define inf 0x7fffffff
map<string, int> mymap;
map<string, int>::iterator z;
bool flag[N];
int g[N][N];
int tree[N][N];
int dis[N];
int father[N];
int MAX[N];
int POS[N];

int max(int a, int b) {
    return a > b ? a : b;
}

void prim(int n, int s) {
    int i, j;
    for (dis[s] = i = 0; i < n - 1; ++i) {
        int MIN = inf, pos = -1;
        for (j = 1; j < n; ++j) {
            if (MIN > dis[j] && flag[j] == 0) {
                MIN = dis[j];
                pos = j;
            }
        }
        if (pos != -1) {
            for (flag[pos] = j = 1; j < n; ++j) {
                if (g[pos][j] != -1 && g[pos][j] < dis[j] && flag[j] == 0) {
                    father[j] = pos;
                    dis[j] = g[pos][j];
                }
            }
        } else goto loop;
    }
loop:
    for (i = 1; i < n; ++i)
        if (father[i] != -1)
            tree[i][father[i]] = tree[father[i]][i] = 1;
}

int MIN, pos;
```

```
void dfs(int u, int n) {
    int i;
    for (flag[u] = i = 1; i < n; ++i)
        if (flag[i] == 0 && tree[u][i] != 0) {//保证在最小生成树上是连接的
            if (MIN > g[0][i] && g[0][i] != -1) {
                MIN = g[0][i];
                pos = i;
            }
            dfs(i, n);
        }
}

void dfs1(int u, int n) {
    int i;
    for (flag[u] = i = 1; i < n; ++i)
        if (flag[i] == 0 && tree[u][i] != 0)   {
            if (u == 0) MAX[i] = 0,  POS[i] = 0;
            else {
                MAX[i] = max(MAX[u], g[i][u]);
                if (MAX[i] == MAX[u]) POS[i] = POS[u];
                else POS[i] = u;
            }
            dfs1(i, n);
        }
}

int main() {
    int n, contain;
    string s;
    while (cin >> n) {
        int a, b;
        mymap.clear();
        memset(g, -1, sizeof (g));
        memset(tree, 0, sizeof (tree));
        memset(flag, 0, sizeof (flag));
        memset(father, -1, sizeof (father));
        mymap["Park"] = 0;
        int i, j, cnt = 1, quan, k = 0;
        for (i = 0; i < n; ++i) {
            cin >> s;
            z = mymap.find(s);
            if (z == mymap.end()) mymap[s] = cnt++;
            a = mymap[s];
            cin >> s;
            z = mymap.find(s);
            if (z == mymap.end()) mymap[s] = cnt++;
            b = mymap[s];
            cin >> quan;
            g[a][b] = g[b][a] = quan;
        }
        n = cnt;
        for (i = 0; i < cnt; dis[i++] = inf);
        cin >> contain;
```

```
        for (i = 1; i < n; ++i)
            if (flag[i] == 0) {
                k++;
                prim(n, i);
            }
        memset(flag, 0, sizeof (flag));
        for (i = 1; i < cnt; ++i)
            if (flag[i] == 0) {
                if (g[0][i] != -1) {
                    MIN = g[0][i];
                    pos = i;
                } else MIN = inf;
                dfs(i, n);
                tree[0][pos] = tree[pos][0] = 1;
            }
        memset(MAX, 0, sizeof (MAX));
        memset(flag, 0, sizeof (flag));
        MAX[0] = 0;
        dfs1(0, n);
        int least = inf;
        int sum = 0;
        for (i = 0; i < n; ++i) {
            for (j = 0; j < n; ++j)
                if (tree[i][j])
                    sum += g[i][j];
        }
        sum /= 2;
        if (sum < least) least = sum;
        for (int p = 0; p < contain - k; ++p) {
            MIN = inf;
            for (i = 1; i < n; ++i) {
                if (tree[0][i] == 0)
                    if (g[0][i] != -1 && g[0][i] - MAX[i] < MIN) {
                        MIN = g[0][i] - MAX[i];
                        pos = i;
                    }
            }
            tree[0][pos] = 1;
            tree[pos][0] = 1;
            tree[pos][POS[pos]] = 0;
            tree[POS[pos]][pos] = 0;
            if (MIN != inf) {
                sum += MIN;
                if (sum < least) least = sum;//如果已经是最大了 那就不用再加边了
                else break;
            }else break; //保证不存在
            memset(flag, 0, sizeof (flag));
            dfs1(0, n);
        }
        cout << "Total␣miles␣driven:␣" << least << endl;
    }
    return 0;
}
```

【最优比例割zju - 2676】这个有点乱,没有整理

```
const int N = 105;
const int M = 405;
int l[M];
const int Inf = 0x7fffffff;
const double eps = 1e-8;
int n, m;
double g[N][N];

struct st {
    int a, b;
} Line[M];

double max_flow(int n, double graph[][N], int s, int t, bool mark[]) {
    int vis[N], pre[N];
    int i, top, bot, best_i, u;
    double cap[N], flow = 0;
    while (true) {
        for (i = 0; i < n; ++i) mark[i] = false;
        cap[s] = Inf, top = bot = 0,
                mark[vis[0] = pre[s] = s] = true, cap[t] = 0;
        while (top <= bot) {
            best_i = vis[top];
            for (i = 0; i < n; ++i)
                if (best_i != i && graph[best_i][i] > 0 && !mark[i]) {
                    vis[++bot] = i, pre[i] = best_i, mark[i] = true;
                    if (cap[best_i] > graph[best_i][i])
                        cap[i] = graph[best_i][i];
                    else cap[i] = cap[best_i];
                    if (i == t) break;
                }
            if (i == t) break;
            top++;
        }
        if (cap[t] == 0) return flow;
        for (i = t; i != s; i = pre[i]) {
            graph[pre[i]][i] -= cap[t];
            graph[i][pre[i]] += cap[t];
        }
        flow += cap[t];
    }
    return flow;
}

double findmin(double x) {
    double c[N][N];
    bool mark[N], h[M];
    double t = 0;
    int tn = 0;
    memset(h, false, sizeof (h));
    memset(c, 0, sizeof (c));
    for (int i = 0; i < m; ++i) {
        int a = Line[i].a, b = Line[i].b;
```

```
        if (g[a][b] >= x + eps)
            c[a][b] = c[b][a] = g[a][b] - x;
        else t += g[a][b], tn++, h[i] = true, l[i] = 1;
    }
    //为被最小割分成的部分中的一个mark2, mark[a] == mark[b] a,在最小割分割后的其中的一个b
    max_flow(n, c, 0, n - 1, mark);
    for (int i = 0; i < m; ++i) if (!h[i]) {
            int a = Line[i].a, b = Line[i].b;
            if (mark[a] != mark[b]) t += g[a][b], tn++, l[i] = 1;
        }
    if (tn == 0) return 1e10;
    return t / tn;
}

int main()
{
    double sum, v, t1, t2, tmp;
    int i, x, y, j, cnt, k = 0;
    int t[M];
    while(scanf("%d␣%d", &n, &m) != EOF) {
        if(k) printf("\n");
        else k = 1;
        sum = 0.0;
        for(i = 0;i< m;++ i) {
            scanf("%d␣%d␣%lf", &x, &y, &v);
            -- x, -- y;
            Line[i].a = x;
            Line[i].b = y;
            g[x][y] = g[y][x] = v;
            sum += v;
        }
        t1 = t2 = 0;
        while(1) {
            tmp = t1;t1 = t2;
            memset(l, 0, sizeof(l));
            t2 = findmin(t1);
            if(t2 == 1e10) break;
            if(tmp == t2) break;
            if(fabs(t1 - t2) < 1e-5) break;
            for(i = 0;i < m;t[i] = l[i], ++ i);
        }
        for(i = cnt = 0;i < m;++ i) if(t[i]) ++ cnt;
        printf("%d\n", cnt);
        for(i = 0;i< m;++ i)
            if(t[i]) {printf("%d", i + 1); break;}
        for(j = i + 1;j < m;++ j)
            if(t[j]) printf("␣%d", j + 1);
        puts("");
    }
    return 0;
}
```

【完美覆盖Dancing Links-hust1017】

```
#include <stdio.h>
#include <string.h>
#include <stack>
using namespace std;
#define N 1005
int U[N*N],D[N*N],L[N*N],R[N*N],S[N];
int DL[N][N];
int Col[N*N],Row[N*N];
stack<int>V;
void Remove(int c)
{
    L[R[c]] = L[c];
    R[L[c]] = R[c];
    for(int i=D[c];i!=c;i=D[i])
        for(int j=R[i];j!=i;j=R[j]){
        U[D[j]] = U[j];
        D[U[j]] = D[j];
        --S[Col[j]];
    }
}
void Resume(int c)
{
    for(int i=U[c];i!=c;i=U[i])
        for(int j=L[i];j!=i;j=L[j])
        {
        ++S[Col[j]];
        U[D[j]] = j;
        D[U[j]] = j;
        }
    L[R[c]] = c;
    R[L[c]] = c;
}
bool dfs(int deep)
{
    if(R[0]==0) return true;
    int i,j,c,min = 10000000;
    for(i=R[0];i!=0;i=R[i]){
        if(S[i]<min){
            min = S[i];
            c = i;
        }
    }
    Remove(c);
    for(i=D[c];i!=c;i=D[i]){
        V.push(Row[i]);
        for(j=R[i];j!=i;j=R[j]) Remove(Col[j]);
        if(dfs(deep+1)){
            return true;
        }
        V.pop();
        for(j=L[i];j!=i;j=L[j]) Resume(Col[j]);

    }
```

```
    Resume(c);
    return false;
}
void init(int n,int m)
{
    int i,j,first;
    while(!V.empty()) V.pop();
    for(i=0;i<=m;i++){
        U[i] = D[i] = i;
        R[i] = i + 1;
        L[i] = i - 1;
    }
    R[m] = 0;
    L[0] = m;
    int len = m + 1;
    memset(S,0,sizeof(S));
    for(i=0;i<n;i++){
        first = -1;
        for(j=1;j<=m;j++){
            if(DL[i][j]==1){
                if(first == -1){
                    first = len;
                }
                else{
                    R[len-1] = len;
                    L[len] = len -1;
                }
                S[j] ++;
                D[U[j]] = len;
                D[len] = j;
                U[len] = U[j];
                U[j] = len;
                Row[len] = i;
                Col[len++] = j;
            }
        }
        if(first != -1)
        {
            L[first] = len -1;
            R[len-1] = first;
        }

    }
}
int main()
{
    int n,m,i,j,t,tt;
    //freopen("input.txt","r",stdin);
    while(scanf("%d␣%d",&n,&m)!=EOF)
    {
        memset(DL,0,sizeof(DL));
        for(i=0;i<n;i++)
        {
```

```
            scanf("%d",&t);
            for(j=0;j<t;j++)
            {
                scanf("%d",&tt);
                DL[i][tt] = 1;
            }
        }
        init(n,m);
        dfs(0);
        if(V.empty()) puts("NO");
        else{
            int tot = V.size();
            printf("%d␣",tot);
            for(i=0;i<tot;i++){
                printf("%d%c",V.top()+1,i==tot-1?'\n':'␣');
                V.pop();
            }
        }

    }
    return 0;
}
```

【三维凸包-HDU 3662 3D Convex Hull】输入点数n (n <= 500),之后输入n个三维点,求面数.注意凸包表示为若干极三角形,从外面看点的顺序是逆时针

```
#include<cstdio>
#include<math.h>
#include<string.h>
#include<vector>
#include<iostream>
#include<algorithm>
using namespace std;
const int maxn = 505;
typedef double db;
const db eps = 1e-8;
int sign( db x){return x < -eps ? -1 : x > eps;}
db sqr(db x){return x * x;}
struct TPoint3{
    db x, y, z;
    TPoint3(){}
    TPoint3(db x, db y, db z): x(x), y(y), z(z){}
    TPoint3 operator - (const TPoint3 p){return TPoint3(x - p.x, y - p.y, z - p.z);}
    TPoint3  X (TPoint3 p){return TPoint3(y*p.z-z*p.y, z*p.x-x*p.z, x*p.y-y*p.x);}
    db  O (TPoint3 p){return x*p.x+y*p.y+z*p.z;}
    db len(){return sqrt(sqr(x) + sqr(y) + sqr(z));}
    bool operator == (TPoint3 p){
        return fabs(p.x -x) < eps && fabs(p.y - y) < eps && fabs(p.z - z) < eps;
    }
    void get(){scanf("%lf%lf%lf", &x, &y, &z);}
};

struct Convex{
    // 剖分出来的三角面的点逆时针存储...
```

```
struct fac{
    int a, b, c;
    bool ok;
} add, F[maxn*4];
TPoint3  P[maxn] ;
int n, cnt, cou;
int to[maxn][maxn];
bool v[ maxn * 4];
// 请在使用前先初始化....
void INIT(){
    cnt = cou = 0;
    memset(to ,0 ,sizeof to);
    memset(v, 0, sizeof v) ;
}
bool dotsInLine(TPoint3 p1,TPoint3 p2,TPoint3 p3){
    return (p2-p1).X(p3-p1).len() <eps;
}
db ptof(TPoint3 &p, fac &f) {
    TPoint3 m = P[f.b]-P[f.a], n = P[f.c]-P[f.a], t = p-P[f.a];
    return m .X( n) .O(t);
}

void deal(int p, int a, int b){
    int f = to[a][b];
    if (F[f].ok){
        if (ptof(P[p], F[f]) > -eps) dfs(p, f);
        else{
            add.a = b, add.b = a, add.c = p, add.ok = 1;
            to[p][b] = to[a][p] = to[b][a] = cnt;
            F[cnt++] = add;
        }
    }
}

void dfs(int p, int cur){
    F[cur].ok = 0;
    deal(p, F[cur].b, F[cur].a);
    deal(p, F[cur].c, F[cur].b);
    deal(p, F[cur].a, F[cur].c);
}

void init(){
    int i;
    for(cou = i = 1; i < n; i++)switch(cou){
            case 1:
                if(P[0] == P[i])break;
                swap(P[cou++], P[i]);
                break;
            case 2:
                if(dotsInLine(P[0], P[1], P[i]))break;
                swap(P[cou++], P[i]);
                add.a = 0, add.b = 1, add.c = 2;
                break;
```

```
            case 3:
                if( sign(ptof(P[i], add))==0 )break;
                swap(P[cou++], P[i]);
                break;
            default:;
        }
}

void convex(){
    int i, j;
    init();
    if(cou < 4){/*....deal...*/}

    for(cnt = i = 0; i < 4; i++){
        add.a = (i+1)%4, add.b = (i+2)%4, add.c = (i+3)%4, add.ok = 1;
        if (ptof(P[i], add) > 0) swap(add.b, add.c);
        to[add.a][add.b] = to[add.b][add.c] = to[add.c][add.a] = cnt;
        F[cnt++] = add;
    }
    for (i = 4; i < n; i++)
        for (j = 0; j < cnt; j++)
            if (F[j].ok && ptof(P[i], F[j]) > eps){
                dfs(i, j);
                break;
            }
    for(i = j = 0; i < cnt; i++) if(F[i].ok)F[j++] = F[i]; cnt = j;
}
// 返回三维凸包表面积...
db get_sur() {
    db ans = 0.0;
    for(int i = 0; i < cnt; i++)
      ans += fabs((P[F[i].b] - P[F[i].a]).X(P[F[i].c] - P[F[i].a]).len()) / 2.0 ;
    return ans ;
}
vector<TPoint3> getvec(int x){ //保证传入的为顺时针...
    vector< TPoint3> ret;
    ret.push_back(P [F[x] . c]) ;
    ret.push_back(P [F[x] . b]) ;
    ret.push_back(P [F[x] . a]) ;
    return ret;
}
db get_vol_f( vector<TPoint3> p) {
    double vol = 0;
    for (int i = 0; i < p.size(); ++i) {
        TPoint3 p1 = p[i] - p.front();
        TPoint3 p2 = (i + 1 == p.size() ? p.front() : p[i + 1]) - p.front();
        vol += p1 .X( p2) .O( p.front() );
    }
    return vol / 6.0;
}
//返回三维凸包体积...
db get_vol() {
    double vol = 0;
```

```
        for (int i = 0; i < cnt; ++i)
            vol += get_vol_f( getvec(i) );
        return fabs(vol);
    }

    void show(){
        for(int i = 0; i < cnt; i++)  //返回的是凸包上的三角形(点的编号) ...
        printf("%d␣%d␣%d\n",F[i].a, F[i].b, F[i].c);
    }
    void Read(){
        int i;
        for(i = 0; i < n; ++ i) P[i].get() ;
    }
    bool same(int x, int y){
        return sign(ptof(P[ F[x].a ], F[y])) == 0 &&
                    sign(ptof(P[ F[x].b ], F[y])) == 0 &&
                    sign(ptof(P[ F[x].c ], F[y])) == 0 ;
    }
    // 获得面数...
    int get_fac_cnt(){
        int ans = 0;
        for (int i = 0; i < cnt; i++){
            bool nb = 1;
            for (int j = 0; j < i; j++){
                if (same(i, j)){
                    nb = 0;
                    break;
                }
            }
            ans += nb;
        }
        return ans;
    }
};
Convex C;
int main(){
    while(~ scanf("%d", &C.n)) {
        C.INIT();
        C.Read();
        C.convex();
        printf("%d\n", C.get_fac_cnt() );
    }
    return 0;
}
```

【多个圆求并】

```
#include<iostream>
#include<stdiO.h>
#include<string.h>
#include<cmath>
#include<ctime>
#include<algorithm>
using namespace std;
```

```
typedef double db;
const int maxn = 1005 ;
const db EPS = 1e-8;
const db PI = acos(- 1.0);
int sign(db x){return x < - EPS ? - 1 : x > EPS;}
db sqr(db x){return x * x;}

struct TPoint{
    db x,y;
    TPoint(){}
    TPoint(db xx,db yy):x(xx),y(yy){}
    TPoint operator+(const TPoint P){return TPoint(x + P.x, y + P.y) ;}
    TPoint operator-(const TPoint P){return TPoint(x - P.x, y - P.y) ;}
    TPoint operator/(const db k){return TPoint(x / k, y / k);}
    bool operator<(const TPoint P)const{
        return sign(x - P.x) < 0 || (sign(x - P.x) == 0 && sign(y - P.y) < 0);
    }
    db len(){return sqrt(sqr(x) + sqr(y));}
    db X(TPoint P){return x * P.y - y * P.x;}
    void get(){scanf("%lf%lf",&x,&y);}
    void out(){cout <<'('<<x<<','<<y<<')'<<endl;}
};

struct TCircle{
    TPoint O; db r;
    db area(){return PI * sqr(r) ;}
    db get_area(db k){
        return sqr(r) * (k - sin(k)) ;
    }
    bool operator<(const TCircle C)const{return sign(r - C.r) > 0;}
    void get(){O.get(); scanf("%lf",&r);}
    int Rel(TCircle C){
        TPoint p1 = O, p2 = C.O;
        db r1 = r, r2 = C.r;
        db d = (p1 - p2).len();
        if(sign(p1.x - p2.x) == 0 && sign(p1.y - p2.y) == 0 && sign(r1 - r2) == 0)
          return 0;
        if(sign(d - r1 - r2) == 0) return 2;
        if(sign(d - fabs(r1 - r2)) == 0) return 4;
        if(sign(d -r1 - r2) > 0) return 1;
        if(sign(d - fabs(r1 - r2)) < 0) return 5;
        if(sign(fabs(r1 - r2) - d) < 0 && sign(d - r1 - r2) < 0) return 3;
        return - 1;
    }
    void get_jiao(TCircle C,TPoint &rp1,TPoint &rp2){
        TPoint p1 = O, p2 = C.O;
        db r1 = r, r2 = C.r;
        db a = p2.x - p1.x, b = p2.y - p1.y;
        db r = (a * a + b * b + r1 * r1 - r2 * r2) / 2.0;
        if(sign(a) == 0 && sign(b)){
            rp1.y = rp2.y = r / b;
            rp1.x = sqrt(r1 * r1 - sqr(rp1.y));
```

```
            rp2.x = - rp1.x;
        }else if(sign(a) && sign(b) == 0){
            rp1.x = rp2.x = r / a;
            rp1.y = sqrt(r1 * r1 - sqr(rp1.x));
            rp2.y = - rp1.y;
        }else if(sign(a) && sign(b)){
            db k = sqr(b * r) - (a * a + b * b)*(r * r - sqr(r1 * a));
            rp1.y = (b * r + sqrt(k)) / (a * a + b * b);
            rp2.y = (b * r - sqrt(k)) / (a * a + b * b);
            rp1.x = (r - b * rp1.y) / a;
            rp2.x = (r - b * rp2.y) / a;
        }
        rp1 = rp1 + p1;
        rp2 = rp2 + p1;
    }
};

db tonormal(db x){return sign(x) < 0 ? x + 2.0 * PI : x;}

struct Event{
    db l,r;
    TPoint Pl,Pr ;
    bool operator<(const Event n)const{
        return sign(l - n.l) < 0 || (sign(l - n.l) == 0 && sign(r - n.r) < 0);
    }
    void add(db ll,db rr,TPoint ppl,TPoint ppr){
        l = ll; r = rr; Pl = ppl; Pr = ppr;
    }
}s[maxn];

TCircle C[maxn],tC[maxn];
TPoint P[maxn], tP[maxn];
db Cir_union(TCircle C[], int n){
    int i,j,k = 0;
    db ans = 0.0;
    sort(C, C + n) ;
    for(i = 0; i < n; ++ i){
        for(j = 0;  j < k; ++ j){
            int flg = C[i].Rel(tC[j]);
            if(flg == 0 || flg == 4 || flg == 5) break ;
        }
        if(j == k) tC[k ++] = C[i] ;
    }
    n = k;
    for(i = 0;  i < n; ++ i) C[i] = tC[i];
    for( i = 0; i < n; ++ i){
        int len = 0;
        TPoint c =TPoint(C[i].r, 0) + C[i]. O;
        for(j = 0; j < n; ++ j) if(i ^ j){
                if(C[i].Rel(C[j]) == 1) continue;
                TPoint a,b ;
                C[i].get_jiao( C[j],a,b);
                TPoint aa = a - C[i].O, bb = b - C[i].O;
```

```
                db jL = tonormal(atan2(aa.y, aa.x)), jR = tonormal(atan2(bb.y, bb.x));
                if((a - C[i].O).X(C[j].O - C[i].O) < 0 )swap(jL, jR),swap(a,b);
                if(sign(jL - jR) > 0)
                  s[len++].add(0,jR,c,b),s[len++].add(jL,2.0 * PI,a, c);
                else s[len++].add(jL, jR, a, b) ;
        }
        if(len) sort(s, s + len);s[len++].add(2.0 * PI, 2.0 * PI, c, c);
        db now = 0.0;
        for(j = 0; j < len; ++ j){
            ans += C[i].get_area(s[j].l - now) + c.X(s[j].Pl);
            for(now = s[j].r,c = s[j].Pr; j < len && sign(s[j].l - now) <= 0; j ++)
              if(sign(now - s[j].r) < 0) now = s[j].r, c = s[j].Pr;-- j;
        }
    }
    return ans / 2.0;
}
int n ;
bool get(){
    if(EOF == scanf("%d",&n)) return 0;
    int i ;
    for(i = 0;  i < n; ++ i) C[i].O.get(),scanf("%lf",&C[i].r);
    return 1;
}
void work(){
    cout<<"the␣Union␣of␣circle␣'s␣area␣is␣"<<  Cir_union(C, n) <<endl;
}
int main(){
    //freopen("D:\\in.txt","r",stdin);
    //freopen("D:\\acout.txt","w",stdout);
    while(get()) work();
    return 0;
}
```

【多个圆求交】

```
#include<iostream>
#include<stdiO.h>
#include<string.h>
#include<cmath>
#include<ctime>
#include<algorithm>
using namespace std;

typedef double db;
const int maxn = 1005 ;
const db EPS = 1e-8;
const db PI = acos(- 1.0);
int sign(db x){return x < - EPS ? - 1 : x > EPS;}
db sqr(db x){return x * x;}

struct TPoint{
    db x,y;
    TPoint(){}
    TPoint(db xx,db yy):x(xx),y(yy){}
```

```
    TPoint operator+(const TPoint P){return TPoint(x + P.x, y + P.y) ;}
    TPoint operator-(const TPoint P){return TPoint(x - P.x, y - P.y) ;}
    TPoint operator/(const db k){return TPoint(x / k, y / k);}
    bool operator<(const TPoint P)const{
        return sign(x - P.x) < 0 || (sign(x - P.x) == 0 && sign(y - P.y) < 0);
    }
    db len(){return sqrt(sqr(x) + sqr(y));}
    db X(TPoint P){return x * P.y - y * P.x;}
    void get(){scanf("%lf%lf",&x,&y);}
};

struct TCircle{
    TPoint O; db r;
    db area(){return PI * sqr(r) ;}
    db get_area(db k){
        return sqr(r) * (k - sin(k)) * 0.5 ;
    }
    bool operator<(const TCircle C)const{return r < C.r;}
    void get(){O.get(); scanf("%lf",&r);}
    int Rel(TCircle C){
        TPoint p1 = O, p2 = C.O;
        db r1 = r, r2 = C.r;
        db d = (p1 - p2).len();
        if(sign(p1.x - p2.x) == 0 && sign(p1.y - p2.y) == 0 && sign(r1 - r2) == 0)
           return 0;
        if(sign(d - r1 - r2) == 0) return 2;
        if(sign(d - fabs(r1 - r2)) == 0) return 4;
        if(sign(d -r1 - r2) > 0) return 1;
        if(sign(d - fabs(r1 - r2)) < 0) return 5;
        if(sign(fabs(r1 - r2) - d) < 0 && sign(d - r1 - r2) < 0) return 3;
        return - 1;
    }
    void get_jiao(TCircle C,TPoint &rp1,TPoint &rp2){
        TPoint p1 = O, p2 = C.O;
        db r1 = r, r2 = C.r;
        db a = p2.x - p1.x, b = p2.y - p1.y;
        db r = (a * a + b * b + r1 * r1 - r2 * r2) / 2.0;
        if(sign(a) == 0 && sign(b)){
            rp1.y = rp2.y = r / b;
            rp1.x = sqrt(r1 * r1 - sqr(rp1.y));
            rp2.x = - rp1.x;
        }else if(sign(a) && sign(b) == 0){
            rp1.x = rp2.x = r / a;
            rp1.y = sqrt(r1 * r1 - sqr(rp1.x));
            rp2.y = - rp1.y;
        }else if(sign(a) && sign(b)){
            db k = sqr(b * r) - (a * a + b * b)*(r * r - sqr(r1 * a));
            rp1.y = (b * r + sqrt(k)) / (a * a + b * b);
            rp2.y = (b * r - sqrt(k)) / (a * a + b * b);
            rp1.x = (r - b * rp1.y) / a;
            rp2.x = (r - b * rp2.y) / a;
        }
        rp1 = rp1 + p1;
```

```
        rp2 = rp2 + p1;
    }
};

db tonormal(db x){return sign(x) < 0 ? x + 2.0 * PI : x;}

struct Event{
    db x;
    int ty;
    TPoint P ;
    bool operator<(const Event n)const{
        return sign(x - n.x) < 0 || (sign(x - n.x) == 0 && ty  < n.ty);
    }
    void add(db xx,int tty, TPoint pp){
        x = xx; ty = tty; P = pp;
    }
}s[maxn];

db R;
TCircle C[maxn],tC[maxn];
TPoint P[maxn], tP[maxn];
db Cir_jiao(TCircle C[], int n, TPoint &jiao){
    int i,j,k = 0;
    for(i = 0; i < n; ++ i) if(sign(C[i].r) == 0) return 0.0;
    db ans = 0.0,parea = 0.0;
    sort(C, C + n) ;
    for(i = 0; i < n; ++ i){
        for(j = 0;  j < k; ++ j){
            int flg = C[i].Rel(tC[j]);
            if( flg == 1) return 0.0;
            if(flg == 0 || flg == 4 || flg == 5) break ;
        }
        if(j == k) tC[k ++] = C[i] ;
    }
    n = k;
    for(i = 0;  i < n; ++ i) C[i] = tC[i];
    if(n == 1) return C[0].area() ;
    for( i = 0; i < n; ++ i){
        int len = 0;
        s[len].x = 0.0;
        s[len].P = TPoint(C[i].r, 0) + C[i]. O ;
        s[len ++].ty = 0;
        s[len].x = 2.0 * PI;
        s[len].P = s[len - 1].P ;
        s[len ++].ty = 1;
        for(j = 0; j < n; ++ j) if(i ^ j){
                TPoint a,b,c =TPoint(C[i].r, 0) + C[i]. O;
                C[i].get_jiao( C[j],a,b);
                TPoint aa = a - C[i].O, bb = b - C[i].O;
                db jL = tonormal(atan2(aa.y, aa.x)), jR = tonormal(atan2(bb.y, bb.x)) ;
                if((a - C[i].O).X(C[j].O - C[i].O) < 0)swap(jL, jR),swap(a,b);
                if(sign(jL - jR) > 0) {
                    s[len++].add(0, 0, c);
```

```
                    s[len++].add(jR,1,b);
                    s[len++].add(jL,0,a);
                    s[len++].add(2.0 * PI,1,c);
                }else{
                    s[len++].add(jL,0,a);
                    s[len++].add(jR,1,b);
                }
            }
            sort(s, s + len);
            db B[maxn];
            int cnt = 0,l = 0;
            for(j = 0; j < len; ++ j){
                if(s[j].ty == 0) ++ cnt; else -- cnt;
                if(cnt == n - s[j].ty)  P[l] = s[j].P,B[l ++] = s[j].x;
            }
            for(j = 0; j + 1< l; j += 2)
             ans += C[i].get_area(B[j + 1] - B[j]) ,parea += P[j].X(P[j + 1]);
    }
    return ans +  parea/ 2.0 ;
}
int n = 4;
TPoint A[5], B[5];
bool get(){
    int i,j;
    for(i = 0; i < 2; ++ i) A[i].get();
    for(i = 0; i < 2; ++ i) B[i].get();
    if(sign(A[0].x) == 0 && sign(A[0].y) == 0 && sign(A[1].x) == 0 && sign(A[1].y) == 0
    && sign(B[0].x) == 0 && sign(B[0].y) == 0 && sign(B[1].x) == 0 && sign(B[1].y) == 0)
      return 0;
    int cc = 0;
    for(i = 0; i < 2; ++ i)
        for(j = 0; j < 2; ++ j)
            C[cc].O = (A[i] + B[j]) / 2.0,C[cc].r = (A[i] - B[j]).len() / 2.0, ++ cc;
    return 1;
}

TCircle CC[maxn];
int cas = 0;
void work(){
    ++ cas;
    TPoint jiao;
    db area = 0.0;
    int i,j,k;
    for(i = 0; i < 4; ++ i) area += C[i].area();
    for(i  =0;  i< 4; ++ i)
        for(j = i + 1; j < 4; ++ j){
            CC[0] = C[i]; CC[1] = C[j];
            area -= Cir_jiao(CC, 2, jiao);
        }
    for(i = 0; i < 4; ++ i)
        for(j = i + 1; j < 4; ++ j)
            for(k = j + 1; k < 4; ++ k) {
                CC[0] = C[i]; CC[1] = C[j]; CC[2] = C[k];
```

```
                area += Cir_jiao(CC, 3, jiao);
            }
    area -= 2.0 * Cir_jiao(C, 4, jiao);
    printf("Case␣%d:␣%.3f\n\n",cas,fabs(area));
}

int main(){
    while(get()) work();
    return 0;
}
```

【树链剖分例题】

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

using namespace std;

const int maxn = 30005;
const int maxm = 65537;
const int Limit = 15;
int head[maxn],vpt[maxm],next[maxm],l = 1; //链表
int num[maxn];//点权
int L[maxm],R[maxm],V[maxm],S[maxm],val[maxn];// 线段树
//树链剖分whead[i]-i对应的重路径的深度最小点,ID[i] - i点的重边的点编号
int dep[maxn],whead[maxn],cnt[maxn],ID[maxn];
int f[maxn][15];//LCA
void mktree(int l,int r,int x){
    L[x]=l; R[x] = r;
    if(l == r - 1)return;
    int mid = (l + r) >> 1;
    mktree(l,mid,x << 1);
    mktree(mid,r,x << 1 | 1);
}

void Insert(int l,int r,int x,int v){
    if(l <= L[x] && R[x] <= r){
        V[x] = v;
        S[x] = v;
        return ;
    }
    int mid = (L[x] + R[x]) >> 1;
    if(r <= mid) Insert(l,r,x << 1,v);else
        if(l >= mid) Insert(l,r,x << 1 | 1,v); else
            Insert(l,mid,x << 1,v),Insert(mid,r,x << 1 | 1,v);
    V[x] = max(V[x << 1],V[x << 1 | 1]);
    S[x] = S[x << 1] + S[x << 1 | 1];
}
int Query(int l,int r,int x){
    if(l <= L[x] && R[x] <= r) return V[x];
    int mid = (L[x] + R[x]) >> 1;
    if(r <= mid) return Query(l,r,x << 1);
    if(l >= mid) return  Query(l,r,x << 1 | 1);
    return max(Query(l,mid,x << 1),Query(mid,r,x<< 1 | 1));
```

```
}
int QueryS(int l,int r,int x){
    if(l <= L[x] && R[x] <= r) return S[x];
    int mid = (L[x] + R[x]) >> 1;
    if(r <= mid) return QueryS(l,r,x << 1);
    if(l >= mid) return  QueryS(l,r,x << 1 | 1);
    return QueryS(l,mid,x << 1) + QueryS(mid,r,x<< 1 | 1);
}
void add(int a,int b){
    vpt[++l]=b;next[l]=head[a];head[a]=l;
    vpt[++l]=a;next[l]=head[b];head[b]=l;
}

int hash[maxn];

void init(int n){
     l = 1;
     memset(head,0,sizeof head);
     memset(hash,0,sizeof hash);
     memset(f,-1,sizeof f);
     for(int i = 0; i < n; ++ i) whead[i] = i; //to self...
}
void dfs(int x,int d){
    cnt[x] = hash[x] = 1;
    dep[x] = d;
    hash[x] = 1;
    int i;
    for(i = 1; i < Limit; ++ i)
      if(f[x][i - 1] != - 1)f[x][i] = f[f[x][i - 1]][i - 1];
    for(i = head[x];i;i=next[i]){
        int y = vpt[i];
        if(hash[y]) continue;
        f[y][0] = x;
        val[y] = num[y];
        dfs(y,d + 1);
        cnt[x] += cnt[y];
    }
}
int idx;
//编号重路径,从深度小的到深度大的递增
void dfs_edge(int x,bool w){
    hash[x] =  1;
    int i,j = - 1;
    for(i = head[x];i;i=next[i]){
        int  y = vpt[i];
        if(hash[y]) continue;
        if(j == -1 || cnt[y] > cnt[j]) j = y;
    }
    if(j != -1)
         whead[j] = whead[x],ID[x] = ++ idx,dfs_edge(j,true);
    else if(w) ID[x] = ++ idx;
    for(i = head[x];i;i=next[i]){
        int  y = vpt[i];
```

```
        if(hash[y] || y == j) continue;
        dfs_edge(y,false);
    }
}

int LCA(int a,int b){
    if(dep[a] < dep[b]) swap(a,b);
    int i,j;
    for(i=dep[a] - dep[b],j=0;i;i>>=1,++j) if(i & 1) a = f[a][j];
    i = Limit - 1;
    while(a ^ b){
        for(;i && f[a][i] == f[b][i]; -- i);
        a=f[a][i];b=f[b][i];
    }
    return a;
}

int n;
void get(){
    int i;
    scanf("%d",&n);
    init(n + 1);
    add(0,1);
    for(i = 0; i < n - 1; ++ i) {
        int a,b;
        scanf("%d%d",&a,&b);
        add(a,b);
    }
    for(i = 1; i <= n; ++ i) scanf("%d",&num[i]);
    ++ n;
}

int Qmax(int a,int b){
    int ans = val[a];
    while(a ^ b){
        if(whead[a] == a) {
            ans = max(ans,val[a]);
            a = f[a][0];
        }else{
            int c = whead[a];
            if(dep[c] < dep[b]){
                ans = max(ans,Query(ID[b],ID[a],1));
                a = b;
            }else{
                ans = max(ans,Query(ID[c],ID[a],1));
                a = c;
            }
        }
    }
    return ans;
}
int Qsum(int a,int b){
    int ans = 0;
```

```
    while(a ^ b){
        if(whead[a] == a) {
            ans += val[a];
            a = f[a][0];
        }else{
            int c = whead[a];
            if(dep[c] < dep[b]){
                ans += QueryS(ID[b],ID[a],1);
                a = b;
            }else{
                ans += QueryS(ID[c],ID[a],1);
                a = c;
            }
        }
    }
    return ans;
}
char op[10];
void work(){
    int i,j,Q;
    dfs(0,0);
    idx = 0;
    memset(hash,0,sizeof hash);
    dfs_edge(0,true);
    mktree(1,idx,1);
    for(i = 0; i < n; ++ i) if(ID[i])
        for(j=head[i];j;j=next[j])
            if(ID[vpt[j]] == ID[i] + 1) {
                Insert(ID[i],ID[i] + 1 ,1,val[vpt[j]]);
                break;
            }
    scanf("%d",&Q);
    while(Q --){
        int a,b,c;
        scanf("%s%d%d",op,&a,&b);
        if(strcmp(op,"CHANGE") == 0){
            if(ID[f[a][0]] && ID[a] ==  ID[f[a][0]] + 1) Insert(ID[a]  - 1,ID[a],1, b);
            val[a] = b;
        }
        if(strcmp(op,"QMAX") == 0){
            c = LCA(a,b); c=  f[c][0];
            int ans = max(Qmax(a,c),Qmax(b,c));
            printf("%d\n",ans);
        }
        if(strcmp(op,"QSUM") == 0){
            int cc;
            c = LCA(a,b);cc = c;c = f[c][0];
            int ans =Qsum(a,c) + Qsum(b,c) - val[cc];
            printf("%d\n",ans);
        }
    }
}
int main(){
```

```
    //freopen("D:\\in.txt","r",stdin);
    get();
    work();
    return 0;
}
```

【最小圆覆盖】

```
#include <iostream>
#include <math.h>
using namespace std;
#define MAX 1000005

const double eps = 1e-9;
const double oo = 1e100;
struct point{
    double x,y;
    void getP(){scanf("%lf%lf",&x,&y);}
    void outP(){printf("%f%f",&x,&y);}
    point(){}
    point(double nx,double ny):x(nx),y(ny){}
}a[MAX];
int n;
int sign(double x){return(x<-eps)?-1:(x>eps);}
double dis(point &a,point &b){
    return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));
}
double cross(point &a,point &b){return a.x*b.y-a.y*b.x;}

point crossLine(point &a,point &b,point &c,point &d){
    double a1 = b.y-a.y,b1 = a.x-b.x,c1 = cross(b,a);
    double a2 = d.y-c.y,b2 = c.x-d.x,c2 = cross(d,c);
    point cP;
    if (sign(a1) == 0){
        cP.y = -c1 / b1;
        cP.x = -(b2*cP.y+c2) / a2;
    }else{
        cP.y = (a1*c2-a2*c1) / (a2*b1-a1*b2);
        cP.x = -(b1*cP.y+c1) / a1;
    }
    return cP;
}

double dot(point &a,point &b,point &c){
    return (b.x-a.x)*(c.x-a.x)+(b.y-a.y)*(c.y-a.y);
}
double cross(point &a,point &b,point &c){
    return (b.x-a.x)*(c.y-a.y)-(b.y-a.y)*(c.x-a.x);
}
double between(point &a,point &b,point &c){
    return sign(dot(c,a,b)) <= 0;
}
/*
point crossLine(point &a,point &b,point &c,point &d){
```

```
    double s1 = cross(a,b,c),s2 = cross(a,b,d);
    point cP;
    cP.x = (s2*c.x-s1*d.x) / (s2-s1);
    cP.y = (s2*c.y-s1*d.y) / (s2-s1);
    return cP;
}
*/
void getCircle(point &a,point &b,point &c,point &o,double &r){
    point d1 = point((a.x+b.x) / 2,(a.y+b.y) / 2);
    point d2 = point((b.x+c.x) / 2,(b.y+c.y) / 2);
    point d3 = point((c.x+a.x) / 2,(c.y+a.y) / 2);
    if (sign(dot(a,b,c)) <= 0)
        o = d2,r = dis(b,c) / 2;
    else if (sign(dot(c,a,b)) <= 0)
        o = d1,r = dis(a,b) / 2;
    else if (sign(dot(b,c,a)) <= 0)
        o = d3,r = dis(c,a) / 2;
    else{
        point e1 = point(d1.x+a.y-b.y,d1.y+b.x-a.x);
        point e2 = point(d2.x+b.y-c.y,d2.y+c.x-b.x);
        o = crossLine(d1,e1,d2,e2);
        r = dis(o,a);
    }

}
bool check(point &o,double &r,point &cp){
    bool f = true;
    double cd,md;
    int i,mdi;
    md = 0.0;
    for (i = 0;i < n;i++){
        cd = dis(a[i],o);
        if (sign(cd-r) > 0) f = false;
        if (sign(cd-md) > 0)
            md = cd,mdi = i;
    }
    cp = a[mdi];
    return f;
}
int main(){
    point p[3],cp,o,mo;
    double r,mr;
    int i,mi;
    scanf("%d",&n);
    for (i = 0;i < n;i++)
        a[i].getP();
    p[0] = a[0],p[1] = a[1],p[2] = a[2];
    getCircle(p[0],p[1],p[2],o,r);
    while (!check(o,r,cp)){
        mr = oo;
        for (i = 0;i < 3;i++){
            point tp = p[i];
            p[i] = cp;
```

```
            getCircle(p[0],p[1],p[2],o,r);
            if (sign(dis(o,tp)-r) <= 0)
                if (sign(mr-r) >= 0)
                    mr = r,mi = i,mo = o;
            p[i] = tp;
        }
        r = mr,p[mi] = cp,o = mo;
    }
    printf("%.3f\n",r);
    return 0;
}
```

【最小球覆盖】

```
#include <stdio.h>
#include <math.h>
#define MAX 35
#define mid(a,b) point((a.x+b.x) / 2,(a.y+b.y) / 2,(a.z+b.z) / 2)
#define vec(a,b) point(b.x-a.x,b.y-a.y,b.z-a.z)

const double eps = 1e-9;
const double oo = 1e300;
struct point{
    double x,y,z;
    point(){}
    point(double nx,double ny,double nz):x(nx),y(ny),z(nz){}
    void getP(){scanf("%lf%lf%lf",&x,&y,&z);}
}a[MAX];
int n;
int sign(double x){return(x<-eps)?-1:(x>eps);}
double dis(point &a,point &b){
    return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y)+(a.z-b.z)*(a.z-b.z));
}
double dot(point a,point b){return a.x*b.x+a.y*b.y+a.z*b.z;}
double dot(point &a,point &b,point &c){
    return (b.x-a.x)*(c.x-a.x)+(b.y-a.y)*(c.y-a.y)+(b.z-a.z)*(c.z-a.z);
}
point cross(point &a,point &b){
    return point(a.y*b.z-a.z*b.y,a.z*b.x-a.x*b.z,a.x*b.y-a.y*b.x);
}
void planeCross(point &n1,double &d1,point &n2,double &d2,point &n3,double &d3,point &o){
    point p12 = cross(n1,n2),p23 = cross(n2,n3),p31 = cross(n3,n1);
    double dd = dot(n1,p23);
    o.x = (d1*p23.x+d2*p31.x+d3*p12.x) / dd;
    o.y = (d1*p23.y+d2*p31.y+d3*p12.y) / dd;
    o.z = (d1*p23.z+d2*p31.z+d3*p12.z) / dd;
}
void getCircle(point &a,point &b,point &c,point &o,double &r){
    point ab = mid(a,b),bc = mid(b,c),ca = mid(c,a);
    if (sign(dot(a,b,c)) <= 0)
        o = bc,r = dis(b,bc);
    else if (sign(dot(b,c,a)) <= 0)
        o = ca,r = dis(c,ca);
    else if (sign(dot(c,a,b)) <= 0)
```

```
        o = ab,r = dis(a,ab);
    else{
        point n0,n1 = vec(a,b),n2 = vec(b,c);
        n0 = cross(n1,n2);
        double d0 = dot(n0,ca),d1 = dot(n1,ab),d2 = dot(n2,bc);
        planeCross(n0,d0,n1,d1,n2,d2,o);
        r = dis(o,a);
    }
}
void getSphere(point &a,point &b,point &c,point &d,point &o,double &r){
    point n1 = vec(a,b),n2 = vec(a,c),n3 = vec(a,d);
    point p[4] = {a,b,c,d},tp,oOo;
    double rr = oo;
    for (int i = 0;i < 4;i++){
        tp = p[i],p[i] = d;
        getCircle(p[0],p[1],p[2],o,r);
        if (sign(dis(o,tp)-r) <= 0 && sign(r-rr) < 0)
            oOo = o,rr = r;
        p[i] = tp;
    }
    if (sign(dot(n1,cross(n2,n3))) != 0){
        point ab = mid(a,b),ac = mid(a,c),ad = mid(a,d);
        double d1 = dot(ab,n1),d2 = dot(ac,n2),d3 = dot(ad,n3);
        planeCross(n1,d1,n2,d2,n3,d3,o);
        r = dis(o,a);
        if (sign(r-rr) < 0)
            oOo = o,rr = r;
    }
    o = oOo,r = rr;
}
bool check(point &o,double &r,point &cp){
    double cd,maxd = 0.0;
    bool f = true;
    int i,maxi;
    for (i = 0;i < n;i++){
        cd = dis(o,a[i]);
        if (sign(cd-r) > 0) f = false;
        if (sign(cd-maxd) > 0)
            maxd = cd,maxi = i;
    }
    cp = a[maxi];
    return f;
}
int main(){
    point p[4],cp,tp,o,oOo;
    double r,rr;
    int i,ii;
    while (scanf("%d",&n),n){
        for (i = 0;i < n;i++)
            a[i].getP();
        getSphere(p[0] = a[0],p[1] = a[1],p[2] = a[2],p[3] = a[3],o,r);
        while (!check(o,r,cp)){
            rr = oo;
```

```
            for (i = 0;i < 4;i++){
                tp = p[i],p[i] = cp;
                getSphere(p[0],p[1],p[2],p[3],o,r);
                if (sign(dis(o,tp)-r) <= 0 && sign(r-rr) < 0)
                    ii = i,oOo = o,rr = r;
                p[i] = tp;
            }
            p[ii] = cp,o = oOo,r = rr;
        }
        printf("%.5f\n",r);
    }
}
```

【多面体体积】

```
//clockwise
bool solve() {
    scanf("%d", &n);
    if (n == 0)
        return false;
    for (int i = 0; i < n; ++i)
        planes[i].input();
    printf("%.4lf\n", get_vol());
    return true;
}

double get_vol() {
    double vol = 0;
    for (int i = 0; i < n; ++i)
        vol += get_vol(planes[i].p);
    return abs(vol);
}

double get_vol(const vector<point>& p) {
    double vol = 0;
    for (vector<point>::const_iterator i = p.begin(); i != p.end(); ++i) {
        point p1 = *i - p.front();
        Point p2 = (i + 1 == p.end() ? p.front() : *(i + 1)) - p.front();
        vol += (p1 * p2) ^ p.front();
    }
    return vol / 6.0;
}
```