福州大学 ACM/ICPC 集训队资料

Implemented in C

陈 研 包能辉

2005年10月7日

目 录

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
  - `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 **)mallo`
  - `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（VC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
- ispunct(c) 非字母数字空格的可打印字符
- isprint(c) 含空格可打印
- isspace(c) 空格,Tab,CR
- islower(c) 小写字母
- 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>中的字符串流更容易转化为char数组)：

```
#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;
```

- 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

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

对于vector(数组) ,deque(双向数组) stack(栈) ,queue(队列)对效率的影响比较大不建议使用，不过由于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 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

1.4 注意事项

1. **Windows下的有些GCC编译器用scanf读64位整数有bug, 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,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。
    (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。

第二章 数据结构

2.1 Hash

Hash表在搜索中运用很广。如果hash函数的值太大，可以用二维数组存储。

```
#define Size 1001  //Size要用大质数,1811,5087,10657,50503,100109,500119
int hash[Size*Size][2]; //此处以2维数组为例
int first[Size][Size];  //初始化置0
int next[Size*Size];    //初始化置0
int entry=1;            //entry初始化为1
int insert_hash()
{
  long long a,b;
  a=前半部分的值(可按位存储);b=后半部分;
  c=a%Size; d=b%Size;
  e=first[c][d];
  while (e!=0) {
    if (hash[e][0]==a&&hash[e][1]==b) return 0;
    e=next[e];
  }
  hash[entry][0]=a;
  hash[entry][1]=b;
  next[entry]=first[c][d];
  first[c][d]=entry;
  entry++;
  return 1;
}
```

搜索时加入: `if (insert_hash()) a_new_node; else already_in_the_hash;`

```
\\字符串用的Hash函数
int ELFhash(char *key)
{
    unsigned long h=0;
    while (*key)
    {
        h=(h<<4)+(*key++);
        unsigned long g=h & 0xf0000000L;
        if (g) h^=g>>24;
        h&=~g;
    }
    return (h%Size);//size要用大质数
}
```

2.2 树形并查集

```
int parent[N],rank[N];
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 union(int a,int b)
{ 如果空间不够rank数组可以不用,此时union(a,b)等价于parent[find(a)]=find(b);
  int i,j;
  i=find(a); j=find(b);
  if (rank[i]>rank[j]) parent[j]=i;
  else {
    parent[i]=j;
    if (rank[i]==rank[j]) rank[j]++;
  }
}
for (i=0;i<n;i++) {parent[i]=i;rank[i]=0;} //初始化
for (i=0;i<m;i++) if (find(a[i])!=find(b[i]))
  union(a[i],b[i]); //合并a[i],b[i]
for (i=0;i<k;i++) if (find(c[i])==find(d[i])) {c[i],d[i]属于同一集合的处理}
```

2.3 最大堆

主要用在需要对于堆进行直接操作的情况下，一般情况使用STL。

```
int heap[N]; //堆用heap[1..n]存储,不是从heap[0]开始
void insert(int x)
{
  p=++n;
  while (p>1&&!(heap[p/2]>=x) {
    heap[p]=heap[p/2]; p=p/2; }
  heap[p]=x;
}
int extract(void)
{
  x=heap[1];heap[1]=h[n--];
  heapify(1);
  return x;
}
void heapify(int r)
{
  p=r;
  if (2*r<=n && heap[2*r]>=heap[p]) p=2*r;
  if (2*r+1<=n && heap[2*r+1]>=heap[p]) p=2*r+1;
  if (p!=r) {
```

```
    tmp=heap[r];heap[r]=heap[p];heap[p]=tmp;
    heapify(p);
}
```

2.4 排列与阶乘进制

【排列转化为阶乘进制】排列在数组a[1..n]中。

```
inline long long KnuthPEncode() {
  long long place = n, encoding = 0;
  int i, j;
  for (i=1; i<=n; i++) {
    int this_letter = a[i];
    encoding *= place--;
    for (j=i+1; j<=n; j++) {
      a[j] < this_letter && encoding++;
    }
  }
  return encoding;
}
```

【阶乘进制转化为排列】

```
inline void transback(long long perm)
{
  int i,j,k,p[N],mark[N];
  memset(p,0,sizeof(p));memset(mark,0,sizeof(mark));
  i=1;
  while (perm>0) {
    p[i]=perm%(i+1);
    perm/=(++i);
  }
  for (i=n-1;i>=1;i--) {
    j=n,k=0;
    while (k<=p[i]) {
      if (!mark[j--]) k++;
    }
    j++;
    a[j]=i+1;
    mark[j]=1;
  }
  j=1;while (mark[j]) j++;a[j]=1;
}
```

2.5 基于空间分割的数据结构

【败者树】用败者树求a[0..n-1]中的最值，建树$O(n)$,查询$O(\log n)$。`findmax(s,t)`返回s,t中的最大/最小值

```
int N=1<<16; // N大于n,并为2的次方;
void builltree(int n,int f[2*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/=2,j/=2) {
     for (int k=i;k<j;k+=2) f[k/2]=f[k]>f[k+1]?f[k]:f[k+1];
     // 求的是最大值,可改求最小值
     if (j%2==0) f[j/2]=f[j];
  }
}
int findmax(int s,int t)
//查询s,t中的最大/最小值,求的是最大值,可改求最小值
{
    int r=f[s]>=f[t]?s:t;
    for (int i=s,j=t;i<j;i>>=1,j>>=1)
    {
        if (!(i&1) && i+1<j && f[i+1]>f[r]) r=i+1;
        if ((j&1) && j-1>i && f[j-1]>f[r]) r=j-1;
    }
    return f[r];
    //以下程序可求a[s..t] 最大/最小值的下标
    while (r<N){
    if (f[r]==f[r*2]) r=r*2;
    else if (f[r]==f[r*2+1])  r=r*2+1;
}
```

【树状数组】支持元素动态改变的求a[0..r]的和。运算符必须满足减法原则。效率是线段树的3倍。数状数组存在in[1..n]中，初始值均为0。

```
int lowbit(int t)
{ return t & (t^(t-1)); }
void modify(int pos,int num) //a[pos]+=num
{
  while (pos<=n) {
    in[pos]+=num;
    pos+=lowbit(pos);
  }
}
int query(int end) //a[0]+...+a[end]
{
  int sum=0;
  while (end>0) {
    sum+=in[end];
    end-=lowbit(end);
  }
  return sum;
}
//初始化
memset(in,0,sizeof(in));
for (i=0;i<n;i++) modify(i+1,a[i]);
```

第三章　基础算法

3.1　搜索算法

【深度优先搜索】

```
#include<time.h>  //用于卡时
time_t t;
void search(int t) //排列生成算法
{
  int i;
  if (t>=n) {output(x);return;}
  if (time(NULL)-t>=5) {output(x);return;} //5秒卡时
  for (i=t;i<n;i++) {
    swap(x[t],x[i]);
    if (constraint(t)&&bound(t)) search(t+1);
    swap(x[t],x[i]);
  }
}
int main()
{
  for (i=0;i<n;i++) x[i]=i; //初始化
  t=time(NULL);
  search(0);
}
```

【广度优先搜索】

```
//定义：
typedef struct {
  T t //当前节点状态;
  int level,fa; //当前节点深度、父节点}CSta;
CSta list[max];
int op,cl //op总结点数，cl待扩展节点编号
CSta nst; //新状态
//初始化：
cl=1; op=1;
list[1]=初始状态; list[1].level=0;
//主程序
while (op>=cl && op<max) {
  for (i=0;i<TotalExpandMethod;i++) {
    nst=生成新节点;
    if (nst不合法) continue;
    if (nst在散列表内) continue;
```

```
    op++;
    list[op]=nst; list[op].fa=cl; list[op].level=list[cl].level+1;
    if (找到解) {output(op); return; }
  }
  cl++;
}
//输出函数:
void output(int op)
{
  if (op==1) {
    printf("(%d,%d)",list[op].x,list[op].y);
  }
  else {
    output(list[op].fa);
    printf("->(%d,%d)",list[op].x,list[op].y);
  }
}
```

【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

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

```
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】

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

3.3 解线性方程组

【高斯消元法】系数矩阵`double a[n][n+1]`, 结果存在a[i][n]中

```
int i,j,k;
double div,tmp[n+1];
for (k=0;k<n;k++) {
  for (i=0;i<n;i++) if (i!=k) {
    major=0.;
```

```
    for (j=k;j<n;j++) if (fabs(a[j][k])>=fabs(major)) {
      major=a[j][k];mx=j;
    }
    j=mx;
    memcpy(tmp,a[k],sizeof(tmp));
    memcpy(a[k],a[j],sizeof(tmp));
    memcpy(a[j],tmp,sizeof(tmp));
    if (a[k][k]==0) return 无唯一解
    div=-a[i][k]/a[k][k];
    a[i][k]=0.;
    for (j=0;j<n+1;j++) if (j!=k)
      a[i][j]+=a[k][j]*div;
  }
}
for (i=0;i<n;i++) {
  if (!a[i][i]) return 无唯一解
  a[i][n]/=a[i][i];
  if (fabs(a[i][n])<1e-8) a[i][n]=0.;
}
```

3.4 LIS 最长非降序列

二分查找$O(n\log n)$ 函数`LISans(int a[])`返回a的最长非降序列长度，b是最长的序列。

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 a[N],b[N];
int find(int p)
{
      j=1,k=ans;
      while (j<=k) {
        m=(j+k)>>1;
        if (b[m]>p) j=m+1;
        else k=m-1;
      }
}
int LISans(int a[])
{
      int ans=0;
      memset(b,0,sizeof(b));
      for (i=0;i<n;i++)
      {
         t=find(a[i])+1;
         if(t>ans) ans=t;
         b[t]=a;         //b数组存放的就是所要求的序列
      }
      return ans;
}
```

3.5 用归并排序求逆序数

最后的结果中数组a是从小到大排序的

```
int temp[2*N];
int msort(int a[],int l,int r) //注意逆序数很容易超过int,必要时用long long
{

        if (l==r) return 0;
        int i,j,k,m=(l+r)>>1;
        int t=msort(a,l,m);
        t+=msort(a,m+1,r);
        i=l;j=m+1;
        for (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。a.y=0是公元前1年,a.y=-1是公元前2年。

```
typedef struct { int y,m,d; }DateType; //日期类型(年,月,日)

int GToJD(DateType a)
//将格林历转成儒略历
{
     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;
}

DateType JDToG(int jd)
//将儒略历转成格林历
{
        DateType a;
        int l,m,i,j,d,n,y;
        l=jd+68569;
        n=(4*l)/146097;
        l=l-(146097*n+3)/4;
        i=(4000*(l+1))/1461001;
        l=l-(1461*i)/4+31;
        j=(80*l)/2447;
        a.d=l-(2447*j)/80;
        l=j/11;
```

```
        a.m=j+2-(12*l);
        a.y=100*(n-49)+i+l;
        return a;
}
```

3.7 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

```
#define DIGIT   4           //一个数组元素存放的位数个数
#define DEPTH   10000       //DEPTH = 10^DIGIT
#define MAX     500         //数组的长度

typedef int bignum_t[MAX+1];
//输入
int read(bignum_t a,istream& is=cin){
  char buf[MAX*DIGIT+1],ch;
  int i,j;
  memset((void*)a,0,sizeof(bignum_t));
  if (!(is>>buf)) return 0;
  for (a[0]=strlen(buf),i=a[0]/2-1;i>=0;i--)
        ch=buf[i],buf[i]=buf[a[0]-1-i],buf[a[0]-1-i]=ch;
  for (a[0]=(a[0]+DIGIT-1)/DIGIT,j=strlen(buf);j<a[0]*DIGIT;buf[j++]='0');
  for (i=1;i<=a[0];i++)
    for (a[i]=0,j=0;j<DIGIT;j++)
      a[i]=a[i]*10+buf[i*DIGIT-1-j]-'0';
  for (;!a[a[0]]&&a[0]>1;a[0]--);
  return 1;
}
//输出
void write(const bignum_t a,ostream& os=cout){
  int i,j;
  for (os<<a[i=a[0]],i--;i;i--)
    for (j=DEPTH/10;j;j/=10)
        os<<a[i]/j%10;
}
//比较
int comp(const bignum_t a,const bignum_t b){
  int i;
  if (a[0]!=b[0])
        return a[0]-b[0];
  for (i=a[0];i;i--)
        if (a[i]!=b[i])
                return a[i]-b[i];
  return 0;
}

int comp(const bignum_t a,const int b){//高精度数与整型比较
```

```
  int c[12]={1};
  for(c[1]=b;c[c[0]]>=DEPTH;c[c[0]+1]=c[c[0]]/DEPTH,c[c[0]]%=DEPTH,c[0]++);
  return comp(a,c);
}
//加法
void add(bignum_t a,const bignum_t b){//高精度数a+b,结果在a中
  int i;
  for (i=1;i<=b[0];i++)
    if ((a[i]+=b[i])>=DEPTH)
        a[i]-=DEPTH,a[i+1]++;
  if (b[0]>=a[0])
    a[0]=b[0];
  else
    for (;a[i]>=DEPTH&&i<a[0];a[i]-=DEPTH,i++,a[i]++);
  a[0]+=(a[a[0]+1]>0);
}

void add(bignum_t a,const int b){//高精度数与整型相加
  int i=1;
  for(a[1]+=b;a[i]>=DEPTH&&i<a[0];a[i+1]+=a[i]/DEPTH,a[i]%=DEPTH,i++);
  for(;a[a[0]]>=DEPTH;a[a[0]+1]=a[a[0]]/DEPTH,a[a[0]]%=DEPTH,a[0]++);
}
//减法
void sub(bignum_t a,const bignum_t b){//高精度数a-b,结果在a中
  int i;
  for (i=1;i<=b[0];i++)
    if ((a[i]-=b[i])<0)
        a[i+1]--,a[i]+=DEPTH;
  for (;a[i]<0;a[i]+=DEPTH,i++,a[i]--);
  for (;!a[a[0]]&&a[0]>1;a[0]--);
}

void sub(bignum_t a,const int b){//高精度数减整型
  int i=1;
  for(a[1]-=b;a[i]<0;a[i+1]+=(a[i]-DEPTH+1)/DEPTH,
      a[i]-=(a[i]-DEPTH+1)/DEPTH*DEPTH,i++);
  for (;!a[a[0]]&&a[0]>1;a[0]--);
}

//乘法
void mul(bignum_t c,const bignum_t a,const bignum_t b){//高精度数a*b,结果
在c中
  int i,j;
  memset((void*)c,0,sizeof(bignum_t));
  for (c[0]=a[0]+b[0]-1,i=1;i<=a[0];i++)
    for (j=1;j<=b[0];j++)
      if ((c[i+j-1]+=a[i]*b[j])>=DEPTH)
        c[i+j]+=c[i+j-1]/DEPTH,c[i+j-1]%=DEPTH;
  for (c[0]+=(c[c[0]+1]>0);!c[c[0]]&&c[0]>1;c[0]--);
}

void mul(bignum_t a,const int b){ //高精度数a与整型b相乘,结果在a中
  int i;
  for (a[1]*=b,i=2;i<=a[0];i++){
```

```
    a[i]*=b;
    if (a[i-1]>=DEPTH)
      a[i]+=a[i-1]/DEPTH,a[i-1]%=DEPTH;
  }
  for (;a[a[0]]>=DEPTH;a[a[0]+1]=a[a[0]]/DEPTH,a[a[0]]%=DEPTH,a[0]++);
  for (;!a[a[0]]&&a[0]>1;a[0]--);
}

void mul(bignum_t b,const bignum_t a,const int c,const int d){
  int i;
  memset((void*)b,0,sizeof(bignum_t));
  for (b[0]=a[0]+d,i=d+1;i<=b[0];i++)
    if ((b[i]+=a[i-d]*c)>=DEPTH)
      b[i+1]+=b[i]/DEPTH,b[i]%=DEPTH;
  for (;b[b[0]+1];b[0]++,b[b[0]+1]=b[b[0]]/DEPTH,b[b[0]]%=DEPTH);
  for (;!b[b[0]]&&b[0]>1;b[0]--);
}

//除法
//除法用的比较
int comp(const bignum_t a,const int c,const int d,const bignum_t b){
  int i,t=0,O=-DEPTH*2;
  if (b[0]-a[0]<d&&c)
    return 1;
  for (i=b[0];i>d;i--){
    t=t*DEPTH+a[i-d]*c-b[i];
    if (t>0) return 1;
    if (t<O) return 0;
  }
  for (i=d;i;i--){
    t=t*DEPTH-b[i];
    if (t>0) return 1;
    if (t<O) return 0;
  }
  return t>0;
}
//除法用的减法
void sub(bignum_t a,const bignum_t b,const int c,const int d){
  int i,O=b[0]+d;
  for (i=1+d;i<=O;i++)
    if ((a[i]-=b[i-d]*c)<0)
      a[i+1]+=(a[i]-DEPTH+1)/DEPTH,a[i]-=(a[i]-DEPTH+1)/DEPTH*DEPTH;
  for(;a[i]<0;a[i+1]+=(a[i]-DEPTH+1)/DEPTH,a[i]-=(a[i]-DEPTH+1)/DEPTH*DEPTH,i++);
  for(;!a[a[0]]&&a[0]>1;a[0]--);
}

void div(bignum_t c,bignum_t a,const bignum_t b){// a/b,结果在c中,余数是a
  int h,l,m,i;
  memset((void*)c,0,sizeof(bignum_t));
  c[0]=(b[0]<a[0]+1)?(a[0]-b[0]+2):1;
  for (i=c[0];i;sub(a,b,c[i]=m,i-1),i--)
    for (h=DEPTH-1,l=0,m=(h+l+1)>>1;h>l;m=(h+l+1)>>1)
        if (comp(b,m,i-1,a)) h=m-1;
```

```
        else l=m;
  for (;!c[c[0]]&&c[0]>1;c[0]--);
  c[0]=c[0]>1?c[0]:1;
}

void div(bignum_t a,const int b,int& c){// a/b,结果在a中,余数是c
  int i;
  for(c=0,i=a[0];i;c=c*DEPTH+a[i],a[i]=c/b,c%=b,i--);
  for (;!a[a[0]]&&a[0]>1;a[0]--);
}
//开方
void sqrt(bignum_t b,bignum_t a){ //开方a ,结果在b
  int h,l,m,i;
  memset((void*)b,0,sizeof(bignum_t));
  for (i=b[0]=(a[0]+1)>>1;i;sub(a,b,m,i-1),b[i]+=m,i--)
    for (h=DEPTH-1,l=0,b[i]=m=(h+l+1)>>1;h>l;b[i]=m=(h+l+1)>>1)
      if (comp(b,m,i-1,a)) h=m-1;
      else l=m;
  for (;!b[b[0]]&&b[0]>1;b[0]--);
  for (i=1;i<=b[0];b[i++]>>=1);
}
```

第四章　图论

【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。

【欧拉公式】$V-E+F=2$

【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;
}
```

4.1　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。

```
int Dijkstra(int n,int s,int t, int path[])
{
  int i,j,w,minc,d[N],mark[N];
  memset(mark,0,sizeof(mark));
  for (i=0;i<n;i++) {d[i]=g[s][i];path[i]=s;}
  mark[s]=1;path[s]=-1;d[s]=0;
```

```
  for (i=1;i<n;i++) {
    minc=maxint;
    for (j=0;j<n;j++)
      if ((mark[j]==0)&&(minc>=d[j])) {minc=d[j];w=j;}
    mark[w]=1;
    for (j=0;j<n;j++)
      if (mark[j]==0&&d[j]>d[w]+g[w][j]) {
        d[j]=d[w]+g[w][j];
        path[j]=w;
      }
  }
  return d[t];
```

【**Bellman Ford** $O(n^2)$】如果存在一个从源点可达的权为负的回路则success=false

```
int Bellman_Ford(int n,int s,int t,int path[],int success)
{
  int i,j,k,d[N];
  for (i=0;i<n;i++) {d[i]=maxint;path[i]=0;}
  d[s]=0;
  for (k=1;k<n;k++)
    for (i=0;i<n;i++)
      for (j=0;j<n;j++)
        if (d[j]>d[i]+G[i][j]) {d[j]=d[i]+G[i][j];path[j]=i;}
  success=0;
  for (i=0;i<n;i++)
    for (j=0;j<n;j++)
      if (d[j]>d[i]+G[i][j]) return 0;
  success=1;
  return d[t];
}
```

【迭代法】点多边少时求最短路用迭代法。

```
for (i=0;i<n;i++) dis[i]=maxint;
dis[起始点]=0;
do {
  quit=1;
  for (i=0;i<n;i++) if (dis[i]<max) {
    for (j=0;j<n;j++) if (g[i][j]有边)
      if (dis[i]+g[i][j]<dis[j]) {
        dis[j]=dis[i]+g[i][j];
        prev[j]=i;
        quit=0;
       }
  }
}while(!quit);
```

【**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

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

```
struct kk{      //队列长度
        int v,len;
}Q[500000];
int top[N];
int heap[N][20];
int F[N];

int cmp(const void *a,const void *b)
{
        edge a1=*(edge *)a;
        edge b1=*(edge *)b;
        return a1.v1-b1.v1;
}
void Insert(int num,int x)
{
        int p=top[num]++;
        while(p&&heap[num][(p-1)/2]<x){
                heap[num][p]=heap[num][(p-1)/2];
                p=(p-1)/2;
        }
        heap[num][p]=x;
}
int pop(int num)
{
        int ans=heap[num][0];
        int t=heap[num][--top[num]];
        int p=0;
        int s=2*p+1;
        while(s<top[num]){
                if(s+1<top[num]&&heap[num][s+1]>heap[num][s]) s++;
                if(heap[num][s]<=t) break;
                heap[num][p]=heap[num][s];
                p=s;
                s=2*p+1;
        }
        heap[num][p]=t;
        return ans;
}

int findkmin(int s,t,k)
{
        int i,qt=0,qh=1;
        kk x,tmp;
        qsort((void *)e,m,sizeof(e[0]),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;
        memset(top,0,sizeof(top));
        Q[0].v=s;
        Q[0].len=0;
        for(;qt<qh;qt++){
               x = Q[qt];
               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(top[tmp.v]<k) {
                          Insert(tmp.v,tmp.len);
                          Q[qh++]=tmp;
                }
                else if(tmp.len<heap[tmp.v][0]) {
                     pop(tmp.v);
                     Insert(tmp.v,tmp.len);
                     Q[qh++]=tmp;
                     }
                 }
        }

        if (top[t]<k) return -1;
        return heap[t][0];
}
```

【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。

4.2 最小生成树

【Prim $O(n^2)$】fa[N]用来记录每个节点的父节点，g[N][N]不连通时权值无穷大。返回最小值。

```
int prim(int n,int fa[])
{
  int i,j,k;
  int cost[N],close[N],used[N];
  memset(used,0,sizeof(used));
  memset(close,0,sizeof(close));
  for (i=0;i<n;i++) {
    cost[i]=g[0][i]; fa[i]=-1; //-1表示没有父节点
  }
  used[0]=1;
  for (i=1;i<n;i++) {
    j=0;
    while (used[j]) j++;
    for (k=0;k<n;k++)
      if ((!used[k])&&(cost[k]<cost[j])) j=k;
    fa[j]=close[j];
    used[j]=1;
```

```
    for (k=0;k<n;k++)
      if ((!used[k])&&(g[j][k]<cost[k])) {
        cost[k]=g[j][k];
        close[k]=j;
      }
  }
  for (i=0;i<n;i++) if (fa[i]>=0) min+=g[fa[i]][i];
  return min;
}
```

【最小图形树】g[N][N]为0时不连通，并设根结点是$V1$

```
void MGT(int n,int g[][N])
{
  int cc[maxn][maxn],              /*圈内顶点序号*/
      l[maxn],                     /*D1顶点在D0中的序号*/
      ft[maxn];                    /*D0圈的最小图形树H0*/
  int i,j,k,t,x,y,v,u,
      ft1[maxn],                   /*D1对应的树形图H1*/
      more,                        /*收缩标志*/
      ss[maxn];                    /*H1中的圈集合*/
  for (i=1;i<=n;i++) l[i]=i;
  for (i=1;i<=maxn;i++) ft[i]=0;
  do {
     more=0;
     for (i=1;i<=maxn;i++) {
       ft1[i]=0;
       for(j=1;j<=maxn;j++) cc[i][j]=0;
     }
     for (i=1;i<=n;i++) cc[i][i]=1;
     for (v=2;v<=n;v++) if (l[v]==v) {          /*V顶点目前未收缩*/
       k=maxint;
       for (u=2;u<=n;u++) if(l[u]==v)       /*若u顶点被收缩成v*/
         for (j=1;j<=n;j++)      /*在以u为终点的弧集中,取最短弧的起点t*/
           if (l[j]!=v&&gh[j][u]>0&&gh[j][u]<k) {k=gh[j][u];t=j;i=u;}
       if (k==maxint) {printf("no answer\n");return;}
       else {
         ft[i]=t;ft1[v]=l[t];  /*t顶点进入H0，对应的L[t]进入H1*/
         if (cc[v][l[t]]) {    /*若l[t]在以v顶点为收缩点所在的圈内*/
           more=1;k=l[t];      /*圈内顶点D1序号设为v*/
           do {
             l[k]=v;k=ft1[k];
           }while (l[k]!=v);
           for (x=1;x<=n;x++) if (l[x]!=v)  /*修正弧长*/
             for (y=1;y<=n;y++)
               if (l[y]==v&&gh[x][y]>0)
                 gh[x][y]=gh[x][y]-gh[ft[y]][y];
         }
         for (i=1;i<=maxn;i++) ss[i]=0;
         k=v;
         do {
           for (i=1;i<=maxn;i++)   /*设置圈内各顶点为收缩点的各圈集合*/
           if (cc[v][i]) cc[ft1[k]][i]=1;
           ss[k]=1;k=ft1[k];
```

```
          }while (ft1[k]&&!ss[k]);
        }
      }
  }while(more);    /*直到不含有向圈为止*/
  for (i=2;i<=n;i++) printf("%d-->%d\n",ft[i],i);
}
```

4.3 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][j]&&i!=kfather&&c[i]==1)
      Ancestor[k]=min(Ancestor[k],depth[i]);
    if (graph[i][k] && c[i]==1) {
      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;
}
```

【求割顶集】

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$P(A,B)$和对应的割项集。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。

4.4 网络流算法

【最大流】Edmonds Karp算法求最大流,复杂度 $O(VE^2)$。返回最大流,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，f[n][n]返回流量矩阵。

```
int EdmondsKarp(int n,int s,int t)
{
  int i,j,k,c,head,tail,flow=0;
  int r[N][N];
  int prev[N],visit[N],q[N];
  for (i=0;i<n;i++) for (j=0;j<n;j++)
    {f[i][j]=0;r[i][j]=g[i][j];}  //初始化流量网络和残留网络
  while (1) {  //在残留网络中找到一条s到t的最短路径
    head=tail=0;
    memset(visit,0,sizeof(visit));
    q[tail++]=s;
    prev[s]=-1; visit[s]=1;
    while(head<tail) {  //宽度优先搜索从s到t的最短路径
      k=q[head++];
      for (i=0;i<n;i++)
        if (!visit[i] && r[k][i]>0) {
          visit[i]=1;
          prev[i]=k;
          if (i==t) goto next;
          q[tail++]=i;
        }
```

```
    }
next:
    if (!visit[t]) break; //流量已达到最大
    c=max;
    j=t;
    while (j!=s) {
      i=prev[j];
      if (c>r[i][j]) c=r[i][j];
      j=i;
    }
    //下面改进流量
    j=t;
    while (j!=s) {
      i=prev[j];
      f[i][j]+=c;
      f[j][i]=-f[i][j];
      r[i][j]=g[i][j]-f[i][j];
      r[j][i]=g[j][i]-f[j][i];
      j=i;
    }
    flow+=c;
  }
  return flow;
}
```

【求最小割】从最大流算法最后一次的标号mark，可以确定最小割。有标号的点属于A^*，未标号的点属于A^{*-}。

【容量有上下界的最大流】

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。

【容量有上下界的最小流】先求附加网络最大流，得到原网络可行流。以t为源点，s为汇点，求出的最大流即为s到t最小流。

【最小费用最大流】Edmonds Karp对偶算法,复杂度$O(V^3E^2)$。返回最大流,输入图容量矩阵g[n][n],费用矩阵w[n][n],取非零值表示有边，s为源点，t为汇点，f[n][n]返回流量矩阵，minw返回最小费用。

```
int DualityEdmondsKarp(int n,int s,int t)
{
```

```
  int i,j,k,c,quit,flow=0;
  int best[N],prev[N];
  minw=0;
  for (i=0;i<n;i++) for (j=0;j<n;j++) {
    f[i][j]=0;
    if (g[i][j]) {g[j][i]=0;w[j][i]=-w[i][j];}
  }
  while (1) {
    for (i=0;i<n;i++) best[i]=max;
    best[s]=0;
    do {
      quit=1;
      for (i=0;i<n;i++) if (best[i]<max) {
        for (j=0;j<n;j++)
          if (f[i][j]<g[i][j]&&best[i]+w[i][j]<best[j]) {
            best[j]=best[i]+w[i][j];
            prev[j]=i;
            quit=0;
          }
      }
    }while(!quit);
    if (best[t]>=max) break;
    c=max;
    j=t;
    while (j!=s) {
      i=prev[j];
      if (c>g[i][j]-f[i][j]) c=g[i][j]-f[i][j];
      j=i;
    }
    j=t;
    while (j!=s) {
      i=prev[j];
      f[i][j]+=c;
      f[j][i]=-f[i][j];
      j=i;
    }
    flow+=c; minw+=c*best[t];
  }
  return 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。

4.5 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[N];
int n,m;
int find(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 || find(v)) return 1;
                mat[j]=v;
          }
 return 0;
}

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

【带权二分图最大匹配】邻接阵mat[m][n], 返回带权最大匹配数, 传入二分图大小m,n和邻接阵mat, 取非零值表示有边match1[],match2[]返回一个最大匹配,未匹配顶点值为-1

```
#define _clr(x) memset(x,0xff,sizeof(x))
int kuhn_munkras(int m,int n,int mat[][MAXN],int* match1,int* match2){
    int s[MAXN],t[MAXN],l1[MAXN],l2[MAXN],p,q,ret=0,i,j,k;
    for (i=0;i<m;i++)
        for (l1[i]=-inf,j=0;j<n;j++)
            l1[i]=mat[i][j]>l1[i]?mat[i][j]:l1[i];
    for (i=0;i<n;l2[i++]=0);
    for (_clr(match1),_clr(match2),i=0;i<m;i++){
        for (_clr(t),s[p=q=0]=i;p<=q&&match1[i]<0;p++)
            for (k=s[p],j=0;j<n&&match1[i]<0;j++)
                if (l1[k]+l2[j]==mat[k][j]&&t[j]<0){
                    s[++q]=match2[j],t[j]=k;
                    if (s[q]<0)
                        for (p=j;p>=0;j=p)
                            match2[j]=k=t[j],p=match1[k],match1[k]=j;
                }
        if (match1[i]<0){
            for (i--,p=inf,k=0;k<=q;k++)
                for (j=0;j<n;j++)
                    if (t[j]<0&&l1[s[k]]+l2[j]-mat[s[k]][j]<p)
                        p=l1[s[k]]+l2[j]-mat[s[k]][j];
            for (j=0;j<n;l2[j]+=t[j]<0?0:p,j++);
```

```
            for (k=0;k<=q;l1[s[k++]]-=p);
        }
    }
    for (i=0;i<m;i++)
        ret+=mat[i][match1[i]];
    return ret;
}
```

4.6 独立集与支配集

定义4.6.1 (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4.6.2 (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4.6.3 (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4.6.4 (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4.6.5 (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4.6.6 (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。路径可从任意节电开始和结束，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。

4.7 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

第五章 计算几何

5.1 基础知识

【类型定义】`typedef struct {double x;double y;}Point;`

【符号函数】为避免误差判断浮点数的符号,Precision 可取一个很小的数如1E-6

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

【距离】系统提供的`hypot(double x,double y)`速度较慢, 如果Point 的点为整数,小心溢出.

```
double dis(Point p1,Point p2)
{ return (sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y))); }
```

【过 $(x_1,y_1),(x_2,y_2)$ 直线方程】 $Ax+By+C=0$，其中 $A=y_2-y_1, B=x_1-x_2, C=x_2y_1-x_1y_2$

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

```
double mul(Point p1,Point p2,Point p0)
{ return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y); }
```

$\boldsymbol{u}\times\boldsymbol{v}=\begin{vmatrix} u_x & u_y \\ v_x & v_y \end{vmatrix}$，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`(p1.x-p0.x)*(p2.x-p0.x或)>0``(p1.y-p0.y)*(p2.y-p0.y)>0`，则AB,AC同向
  - 若`(p1.x-p0.x)*(p2.x-p0.x或)<0``(p1.y-p0.y)*(p2.y-p0.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】$AB\bullet AC=|AB||AC|cos\alpha=(x_B-x_A)(x_C-x_A)+(y_B-y_A)(y_C-y_A)$

— $AB \bullet AC < 0$，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

【比较向量极角大小】

```
bool cmp2(Point p1,Point p2)
{
  if (p1.y>0&&p2.y<=0) return false;
  else if (p1.y<=0&&p2.y>0) return true;
  else if (p1.y==0&&p2.y==0) {
    if (p1.x<0&&p2.x>0) return true;
    if (p1.x>0&&p2.x<0) return false;
    else if (abs(p1.x)<abs(p2.x)) return true;
    else if (abs(p1.x)>abs(p2.x)) return false;
    else return false; //equal
  } else {
    int a=mul(p1,p2);
    if (a>0) return true;
    else if (a<0) return false;
    else {
      int x=abs(p1.x)-abs(p2.x),y=abs(p1.y)-abs(p2.y);
      if (x<0||y<0) return true;
      else if (x>0||y>0) return false;
      else return false; //the same point
    }
  }
}
```

【点是否在线段AB上(包括端点)】

```
int xyCmp(double p,double mini,double maxi) //坐标比较
{ return dblcmp(p-mini)*dblcmp(p-maxi);}

//  0 C 在AB端点上
//  1 C 不在AB上
// -1 C 在AB上,但不在端点上
int betweenCmp(Point c,Point a,Point b)
//判断c是否在线段ab范围内
{
  if (fabs(b.x-c.x)>fabs(b.y-c.y)) //为避免误差选择选择跨度大的分量
      retrun xyCmp(a.x,min(b.x,c.x),max(b.x,c.x));
  else
      return xyCmp(a.y,min(b.y,c.y),max(b.y,c.y));
}
```

【判断线段相交】下面的代码可以判断线段相交，并可求交点

```
// 0 不相交
// 1 规范相交
// 2 非规范相交(端点为交点,线段重合)
```

```
// a,b为第一条线段的两端点
// c,d为第二条线段的两端点
// p为交点
int segcross(Point a,Point b,Point c,Point d,Point & p)
{
     double s1,s2,s3,s4;
     int d1,d2,d3,d4;

     d1=dblcmp(s1=mul(a,b,c));
     d2=dblcmp(s2=mul(a,b,d));
     d3=dblcmp(s3=mul(c,d,a));
     d4=dblcmp(s4=mul(c,d,b));

     //若规范相交则求交点的代码
     if ( (d1^d2)==-2 && (d3^d4)==-2)
     {
        p.x=(c.x*s2-d.x*s1)/(s2-s1);
        p.y=(c.y*s2-d.y*s1)/(s2-s1);
        return 1;
     }
     //判定非规范相交
     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 2;

     return 0;
}
```

【点P到直线AB距离】

$$\frac{|(P-A)\times(B-A)|}{|B-A|}$$

【多边形面积公式】$Area=\frac{1}{2}\sum\begin{vmatrix} x_i & x_{i+1} \\ y_i & y_{i+1} \end{vmatrix}$

【多边形重心公式】多边形重心(C_x,C_y)

$$C_x=\frac{1}{6Area}\sum(x_i+x_{i+1})\begin{vmatrix} x_i & x_{i+1} \\ y_i & y_{i+1} \end{vmatrix} C_y=\frac{1}{6Area}\sum(y_i+y_{i+1})\begin{vmatrix} x_i & x_{i+1} \\ y_i & y_{i+1} \end{vmatrix}$$

【**Pick**定理(网格中)】$Area=\frac{1}{2}EdgeDot+InnerDot-1$

5.2 几何图形的包含关系

【点q是否在多边形$p[n]$内部】

【算法一】精度高，但不适用于自交的病态多边形，不包括边界

```
int Inside(Point q)
{
  int i,j,c=0;
  for (i=0,j=n-1;i<n;j=i++) {
    if ((((p[i].y<=q.y)&&(q.y<p[j].y))||
```

```
          ((p[j].y<=q.y)&&(q.y<p[i].y)))&&
         (q.x<(p[j].x-p[i].x)*(q.y-p[i].y)/(p[j].y-p[i].y)+p[i].x))
               c=!c;
  }
  return c;
}
```

【算法二】可能产生浮点误差，但适用于自交的多边形

```
double getaix(double a,double b,double c)
{ return acos((a*a+b*b-c*c)/(2*a*b)); }
int Inside(Point q)
{
  double totalaix=0;
  int i;
  for(i=0;i<n;i++)
    totalaix+=getaix(dis(q,p[i]),dis(q,p[(i+1)%n]),dis(p[i],p[(i+1)%n]));
  if(fabs(totalaix-2*acos(-1))<1e-4) return 1;
  return 0;
}
```

【判断线段是否在多边形内】

```
1. if 线端PQ的端点不都在多边形内
2.   then return false;
3. 点集pointSet初始化为空;
4. for 多边形的每条边s
5.   do if 线段的某个端点在s上
6.         then 将该端点加入pointSet;
7.      else if s的某个端点在线段PQ上
8.         then 将该端点加入pointSet;
9.      else if s和线段PQ相交// 这时候可以肯定是内交
10.        then return false;
11. 将pointSet中的点按照X-Y坐标排序，X坐标小的排在前面，对于X坐标相同的点，Y坐标
小的排在前面;
12. for pointSet中每两个相邻点pointSet[i] , pointSet[ i+1]
13.    do if pointSet[i] , pointSet[ i+1] 的中点不在多边形中
14.         then return false;
15. return true;
```

【判断线段、折线、矩形、多边形是否在圆内】

因为圆是凸集，所以只要判断是否每个顶点都在圆内即可。

5.3 求凸包(Graham 算法)

【说明】输入点集p[n]，输出凸包点集ch[top]

【注意】注意一些特殊情况：$n = 1, 2; top = 1, 2$

```
int cmp(const void *e1,const void *e2)
{
  Point *a,*b;
  a=(Point*)e1;b=(Point*)e2;
  if (mul(*a,*b,p[0])>0) return -1;
```

6.4 $a^b \bmod c$

```
int PowerMod(int a,int b,int c)
{
  int r=1;
  while(b) {
    if(b&1) r=(r*a)%c;
    b>>=1;
    a=(a*a)%c;
  }
  return r;
}
```

6.5 素数判定随机算法

```
int64 mod_mul(int64 x,int64 y,int n)
{
  if(!x) return 0;
  return (((x&1)*y)%n+(mod_mul(
        x>>1,y,n)<<1)%n)%n;
}
long long myrand()
{
  long long a=rand();
  a*=rand();
  return abs(a);
}
int Rabin_Miller(long long n)
{
  long long k=0,i,j,m,a;
  if(n<2) return 0;
  if(n==2) return 1;
  if(!(n&1)) return 0;
  m=n-1;
  while(!(m&1)) m>>=1,k++;
  for(i=0;i<20;i++) {
    a=myrand()%(n-2)+2;
    a=PowerMod(a,m,n);
    if(a==1) continue;
    for(j=0;j<k;j++) {
      if(a==n-1) break;
      a=mod_mul(a,a,n);
    }
    if(j<k) continue;
    return 0;
  }
  return 1;
}
```

6.6 分解质因数随机算法Pollard

```
long long func(int64 x,int64 n)
{
  return (mod_mul(x,x,n)+1)%n;
}
void Pollard(long long n)
{
  long long 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<20;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);  //p为n的约数
    Pollard(n/p);
    return;
  }
}
```

6.7 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6.7.1 (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】 $x \equiv y \pmod z \iff x \bmod z = y \bmod z \iff z|x-y \iff (x,z) = (y,z)$. $(x, y \in Z)$

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6.7.2 (Frobenius问题) 给定$n(n \geq 2)$个互质的正整数$a_1, a_2, \cdots, a_n$，求使不定方程$\sum_i a_i x_i = F$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，$F(a_1, a_2) = a_1 a_2 - a_1 - a_2$; $n = 3$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6.7.3 (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6.7.1 (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

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

定理6.7.4 (欧拉定理) $\forall$正整数$m, a \perp m \Rightarrow a^{\varphi(m)} \equiv 1 \pmod m$.

例6.7.1 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6.7.3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6.7.2 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6.7.3 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6.7.4 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$\underline{1}23456 \overset{+2}{\rightarrow} \underline{2}5456 \overset{+4}{\rightarrow} \underline{5}856 \overset{+10}{\rightarrow} \underline{9}56 \overset{+18}{\rightarrow} 74 \Rightarrow 74 \bmod 7 = 4, \quad \therefore 123456 \bmod 7 = 4$

【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*6.7.1*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6.7.5 (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】

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

【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$f(n,m)=\left\{{n \atop m}\right\}$. □

【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

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

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. $\phi^n=F_n\phi+F_{n-1}$; $\Phi^n=F_n\Phi^n+F_{n-1}$
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】$\sum_k \binom{n}{k} F_t^k F_{t-1}^{n-k} F_{m+k} = F_{m+tn}$

【**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$F_{k_j}+\cdots+F_{k_r}\quad\big(1\leq j\leq r$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$\sum_{k=0}^{n} a_k b_{n-k}$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$c_n = \sum_k \binom{n}{k} a_n b_{n-k}$则设$G(z)$为$\frac{a_n}{n!}$，$H(z)$为$\frac{b_n}{n!}$则$G(z)H(z)$为$\sum_{k=0}^{n} \frac{a_n b_{n-k}}{k!(n-k)!} = \frac{c_n}{n!}$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$\prod_{j\geq 0} \sum_{k\geq 0} a_{jk} z^k = \sum_{n\geq 0} z^n \sum_{\substack{k_0,k_1,\cdots\geq 0 \\ k_0+k_1+\cdots=n}} a_{0k_0} a_{1k_1} \cdots$。特别地，3个母函数相乘得到$\sum_{\substack{i,j,k\geq 0 \\ i+j+k=n}} a_i b_j c_k$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$< c^k a_k >$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：

$$\begin{cases} \frac{1}{2}\Big(G(z) + G(-z)\Big) = a_0 + a_2 z^2 + a_4 z^4 + \cdots \\ \frac{1}{2}\Big(G(z) - G(-z)\Big) = a_1 z + a_3 z^3 + a_5 z^5 + \cdots \end{cases}$$

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

$$\sum_{n \bmod m = r} a_n z^n = \frac{1}{m} \sum_{0\leq k<m} \omega^{-kr} G(\omega^k z) \quad (0 \leq r < m)$$

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$G'(z) = a_1 + 2a_2 z + 3a_3 z^2 + \cdots = \sum_{k\geq 0} (k+1) a_{k+1} z^k$可得$zG'(z)$是$< na_n >$的母函数。

相反的，$\int_0^z G(t)\, dt = a_0 z + \frac{1}{2} a_1 z^2 + \frac{1}{3} a_2 z^3 + \cdots = \sum_{k\geq 1} \frac{1}{k} a_{k-1} z^k$

【特例】设$G(z)=\frac{1}{1-z}$则

$$\begin{aligned} G'(z) &= \frac{1}{(1-z)^2}=1+2z+3z^2+\cdots=\sum_{k\geq 0}(k+1)z^k \\ \int_0^z G(t)\,dt &= \ln\frac{1}{1-z}=z+\frac{1}{2}z^2+\frac{1}{3}z^3+\cdots=\sum_{k\geq 1}\frac{1}{k}z^k \\ F(z)=G(z)\int_0^z G(t)\,dt &= \frac{1}{1-z}\ln\frac{1}{1-z}=z+\frac{3}{2}z^2+\frac{11}{6}z^3+\cdots=\sum_{k\geq 0}H_k z^k \end{aligned}$$

对上式求导得$F'(z)=\dfrac{1}{(1-z)^2}+\dfrac{\ln\frac{1}{1-z}}{(1-z)^2}$

$\because F'(z)=<(n+1)H_{n+1}>,\quad \dfrac{1}{(1-z)^2}=<\sum_{k=0}^{n}1>,\quad \dfrac{\ln\frac{1}{1-z}}{(1-z)^2}=<\sum_{k=1}^{n}H_k>$

$\therefore \sum_{k=1}^{n-1}H_k=nH_n-n$

F. 已知的母函数

i) **二项式定理** $(1+z)^r=1+rz+\dfrac{r(r-1)}{2}z^2+\cdots=\sum_{k\geq 0}\binom{r}{k}z^k$

【特例】当r为负整数时$\dfrac{1}{(1-z)^{n+1}}=\sum_{k\geq 0}\binom{-n-1}{k}(-z)^k=\sum_{k\geq 0}\binom{n+k}{n}z^k$

ii) **指数级数** $e^z=1+z+\dfrac{1}{2!}z^2+\cdots=\sum_{k\geq 0}\dfrac{1}{k!}z^k$

【推广】$(e^z-1)^n=z^n+\dfrac{1}{n+1}\left\{{n+1\atop n}\right\}z^{n+1}+\cdots=n!\sum_k\left\{{k\atop n}\right\}\dfrac{z^k}{k!}$

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$G(z) = \sum_{k\geq 0} h_k z^k = (1 + x_1z + x_1^2z^2 + \cdots)\cdots(1 + x_nz + x_n^2z^2 + \cdots) = \dfrac{1}{(1-x_1z)(1-x_2z)\cdots(1-x_nz)}$

$G(z)$是多项式的倒数，此时取对数往往可简化计算。

$$\ln G(z) = \ln\frac{1}{1-x_1z} + \cdots + \ln\frac{1}{1-x_nz} = \left(\sum_{k\geq 1}\frac{x_1^k z^k}{k}\right) + \cdots + \left(\sum_{k\geq 1}\frac{x_n^k z^k}{k}\right) = \sum_{k\geq 1}\frac{S_k z^k}{k}$$

$$\therefore G(z) = e^{\ln G(z)} = exp\left(\sum_{k\geq 1}\frac{S_k z^k}{k}\right) = \prod_{k\geq 1} e^{S_k z^k/k} = (1 + S_1z + \frac{S_1^2z^2}{2!} + \cdots)(1 + \frac{S_2z^2}{2} + \frac{S_2^2z^4}{2^2\dot{2}!} + \cdots)\cdots = \sum_{m\geq 0}\left(\sum_{\substack{k_1,k_2,\cdots,k_m\geq 0 \\ k_1+2k_2+\cdots+mk_m=m}} \frac{S_1^{k_1}}{1^{k_1}k_1!}\frac{S_2^{k_2}}{2^{k_2}k_2!}\cdots\frac{S_m^{k_m}}{m^{k_m}k_m!}\right) z^m$$

【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$\sum_{m\geq 1}\frac{S_m z^m}{m}=\ln G(z)=\sum_{k\geq 1}(-1)^{k-1}\frac{(h_1z+h_2z^2+\cdots)^k}{k}$得
系数为$\frac{(-1)^{k_1+k_2+\cdots+k_m-1}m(k_1+k_2+\cdots+k_m-1)!}{k_1!k_2!\cdots k_m!}$

例7.3.3 求双下标数列$a_{mn}=\binom{n}{m}$的母函数。

解： $\sum_{m,n\geq 0}a_{mn}w^m z^n=\sum_{m,n\geq 0}\binom{n}{m}w^m z^n=\sum_{n\geq 0}(1+w)^n z^n=\frac{1}{1-z-wz}$

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，$G_n(z)=\frac{1}{1-z}\cdot\frac{1}{1-2z}\cdots\frac{1}{1-nz}=\sum_k\begin{Bmatrix}k\\n\end{Bmatrix}z^{k-n}\Rightarrow\begin{Bmatrix}n+r\\n\end{Bmatrix}$

例7.3.5 求$\sum_{k_0+2k_1+4k_2+8k_3+\cdots=n}\binom{r}{k_0}\binom{r}{k_1}\binom{r}{k_2}\binom{r}{k_3}\cdots$

解： $G(z)=(1+z)^r(1+z^2)^r(1+z^4)^r\cdots=(1-z)^{-r}=\sum_n\binom{r+n-1}{n}z^n\Rightarrow$
$\binom{r+n-1}{n}$