Include Notion 数论 数列 分数规划 经典问题

Include

```
#include <vector>
#include <list>
#include <map>
#include <set>
#include <deque>
#include <stack>
#include <algorithm>
#include <sstream>
#include <iostream>
#include <cstdio>
#include <cmath>
#include <cstdlib>
#include <cstring>
using namespace std;
```

```
#include <vector>
#include <list>
#include <map>
#include <set>
#include <deque>
#include <stack>
#include <bitset>
#include <algorithm>
#include <functional>
#include <numeric>
#include <utility>
#include <sstream>
#include <iostream>
#include <iomanip>
#include <cstdio>
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <string.h>
#include <limits>
using namespace std;
```

Notion

壹.zzy的半平面交，如果核为一个点，calCore也会返回true，只不过n=3 且三个点相同而已（面积为零）
2.树状数组arr[i]=lowbit(i)。i=壹...N，代表每个位置都赋制为壹
3.二维树状数组arr[i][j] = lowbit(i) * lowbit(j)。i=壹..N,j=壹..N，代表每个位置都赋制为壹三维类似
4.关注Formatter类库
5.一个整数，所有数位递归相加，公式：(x+8)%9+壹　　其实就是x%9，然后返回[壹,9]区间上的值
6.稠密图上的prim求MST不要用heap！
7.注意原始模板中的LCA转RMQ中询问的时候可能是区间颠倒。。。
8.Stirling公式求位数最好先转int再+壹
9.最小路径覆盖、二分图都是以 0 开始。。
壹 0.二分图最大匹配中，如果仅仅判断 V 壹中的点是否能完全匹配，则可以这样写 solve

```
bool solve() {
    memset(vlink, 0, sizeof(vlink));
    memset(link, 255, sizeof(link));
    int ans = 0;
    for(int i = 0; i < n; i ++) {
        memset(vis, 0, sizeof(vis));
        if(!find(i)) return false;
    }
    return true;
}
```

壹壹.双连通分支中不同连通块之间，如果有边，有且只有一条
壹 2.费马点最好向八个方向搜索，否则可能被死锁

壹 3.kmp 扩展：
整个字符串都被一个连续的字符串覆盖（可能不充满），这样最小的能覆盖整个字符串的串的长度为 len
则 len = (字符串长度) - (kmp 构造的 pi 数组的最后一个值为整个字符串出现的)
如 ABABA, 则 len = n - pi[n-壹] = 2; 因为整个串都可以被 AB 覆盖

```
int k, pi[壹00壹0];
#define KMP_GO(X) while(k>0 && P[k] != X[i])   k  =  pi[k- 壹 ];
if(P[k] == X[i]) k ++
void kmp_match(long long * P, int n) {
    pi[0] = k = 0;
    for(int i = 壹; i < n; i ++) {
        KMP_GO(P);
        pi[i] = k;
    }
}
```

壹 4.两个欧拉定理
欧拉定理跟高斯定理一样，几乎在各个领域都有。
记两个跟几何有关的。

平面几何的欧拉定理
垂心，重心，外心共线，且分割比例是 2/壹

平面图的欧拉定理
平面图有，顶点数-边数+区域数=连通块数+壹

数论

最大公约数

```
int gcd(int a,int b) {//如果a和b有一个是0，返回另一个
    static int t;
    for(; t=b; b=a%b,a=t);
    return a;
}
```

扩展欧几里得

```
/**
    ----------------------
    ax+by=gcd(a, b);
    将x和y写入，并且返回gcd(a, b);

    此(x, y)为一组解，
    全部的解为(x + k*b/gcd, y - k*a/gcd);    (k∈Z)
    即△x = b/gcd,  △y = -a/gcd;

    如果ax + by = n*gcd(a, b)
    解为     x' = n*x;
             y' = n*y;
    然而变化依然为：△x = b/gcd,  △y = -a/gcd;
    其中，x<b的解共有gcd个
    （这样其实就已经解了模线性方程）
*/
int ex_eculid(int a0,int b0,int&x,int&y){
    static int a[壹00],b[壹00],i,t,g;
    for(*a=a0, *b=b0, i=0; b[i]; a[i+壹]=b[i], b[i+壹]=a[i]%b[i], i++);
    for(x=壹, y=0, g=a[i]; i--; t=x-a[i]/b[i]*y, x=y, y=t);
    return g;
}
```

中国余数定理(模线性方程组)

```
/**
    下标从0开始，个数为k
    满足：A % n[i] = a[i];
    返回A(0<=A<N)
    全部解为A+k*N.(k∈Z 且 N = ∏n[i])
*/
int china(int*a, int*n, int k) {
    static int A,N,m,x,y,i;
    for(N=壹, i=0; i<k; N*=n[i++]);
    for(A=0; i--;ex_eculid(m=N/n[i],n[i],x,y), A=(A+a[i]*m*x)%N);
    return A<0?A+N:A;
}
```

返回 a 的逆元（模取 c）

```
//  要求a、c互质
int inv(int a, int c) {
    static int x, y, dx, g;
    g = ex_eculid(a, c, x, y);
    dx = c / g;
    x %= dx;
    return x < 0 ? x + dx : x;
}
```

合并同余方程

```
/**
        x = a壹 mod(n壹)
        x = a2 mod(n2)
```

```
    等价于:
        x = a0 mod(n0)
    ----------------------
    可以用于解 同余方程组（n壹,n2....不一定互质，比china强）
    要求：n壹, n2不同且不为0
*/

void combine(int a壹, int n壹, int a2, int n2, int &a0, int &n0) {
    static int a, b, c, g;

    a = n壹;
    b = (a2 - a壹) % n2;        if(b < 0)   b += n2;
    c = n2;

    g = gcd(gcd(a, b), c);
    a /= g;     b /= g;     c /= g;

    n0 = n壹 * c;
    a0 = (a壹 + n壹 * inv(a, c) * b) % n0;
}
```

模取幂运算

```
/**
    返回(a^b)%w;
    O(lg(b))的效率
*/
int mod_pow(int a,int k,int m) {
    static int r;
    for(r=壹;k;k>>=壹,a=a*a%m)  if(k&壹)r=r*a%m;
    return r;
}
```

素数生成器

```
/**
    prim:   将素数压入此数组中，如2,3,5,7,壹壹,壹3,壹7,壹9...
    isPrime:    下表为i的数，是否为素数
    n:      判断小于等于n的数，prime的上限
    m:      isPrime的上限（n <= m <= n*n）

    返回：素数的个数
*/
int getPrime(int*prime,bool*isPrime,int n,int m){
    int p,i,j;
    memset(isPrime, 壹, m);     //悬!
    p=isPrime[0]=isPrime[壹]=0;
    for(i=2;i<=n;i++)
        if(isPrime[i])
            for(prime[p++]=i,j=2*i;j<=m;j+=i)
                isPrime[j] = false;
    return p;
}
```

整数分解(>=2)

```
Prime做到√maxn就行了,isPrime做到maxn
int resolve(int x, int *arr) {
    int len = 0;
    for(int i = 0; x != 壹; i ++) {
        if(isPrime[x]) {
            arr[len ++] = x;
            break;
        }
        while(x % prime[i] == 0) {
            arr[len ++] = prime[i];
            x /= prime[i];
```

```
        }
    }
    return len;
}
```

下面是没有isPrime 的版本，因此isPrime只需做到prime长度就可以了。

```
int resolve(int x, int *arr) {
    int len = 0;
    for(int i=0;prime[i]*prime[i]<=x;i++){
        while(x%prime[i]==0) {
            arr[len ++] = prime[i];
            x /= prime[i];
        }
    }
    if(x != 壹) arr[len ++] = x;
    return len;
}
```

Baby-step giant-step

In group theory, a branch of mathematics, the **baby-step giant-step** algorithm is a series of well-defined steps to compute the discrete logarithm. The discrete log problem is of fundamental importance to the area of public key cryptography. Many of the most commonly used cryptography systems are based on the assumption that the discrete log is extremely difficult to compute; the more difficult it is, the more security it provides a data transfer. One way to increase the difficulty of the discrete log problem is to base the cryptosystem on a larger group.

Contents

[hide]

[edit] Theory

The algorithm is based on a space-time tradeoff. It is a fairly simple modification of trial multiplication, the naive method of finding discrete logarithms.

Given a cyclic group G of order n, a generator α of the group and a group element β, the problem is to find an integer x such that

The baby-step giant-step algorithm is based on rewriting x as $x = im + j$, with and and . Therefore, we have:

The algorithm precomputes α^j for several values of j. Then it fixes an m and tries values of i in the left-hand side of the congruence above, in the manner of trial multiplication. It tests to see if the congruence is satisfied for any value of j, using the precomputed values of α^j.

[edit] The algorithm

Input: A cyclic group G of order n, having a generator α and an element β.

Output: A value x satisfying $\alpha^x = \beta$.

1. $m \leftarrow \text{Ceiling}(\sqrt{n})$
2. For all j where $0 \leqslant j < m$:
    1. Compute α^j and store the pair (j, α^j) in a table. (See section "In practice")
3. Compute α^{-m}.
4. $\gamma \leftarrow \beta$.
5. For $i = 0$ to $(m - 壹)$:
    1. Check to see if γ is the second component (α^j) of any pair in the table.
    2. If so, return $im + j$.
    3. If not, $\gamma \leftarrow \gamma \cdot \alpha^{-m}$.

[edit] In practice

The best way to speed up the baby-step giant-step algorithm is to use an efficient table lookup scheme. The best in this case is a hash table. The hashing is done on the second component, and to perform the check in step 壹 of the main loop, γ is hashed and the resulting memory address checked. Since hash tables can retrieve and add elements in O(壹) time (constant time), this does not slow down the overall baby-step giant-step algorithm.

The running time of the algorithm and the space complexity is O(), much better than the O(n) running time of the naive brute force calculation.

[edit] Notes

- The baby-step giant-step algorithm is a generic algorithm. It works for every finite cyclic group.
- It is not necessary to know the order of the group G in advance. The algorithm still works if n is merely an upper bound on the group order.
- Usually the baby-step giant-step algorithm is used for groups whose order is prime. If the order of the group is composite then the Pohlig-Hellman algorithm is more efficient.
- The algorithm requires $O(m)$ memory. It is possible to use less memory by choosing a smaller m in the first step of the algorithm. Doing so increases the running time, which then is $O(n/m)$. Alternatively one can use Pollard's rho algorithm for logarithms, which has about the same running time as the baby-step giant-step algorithm, but only a small memory requirement.
- The algorithm was originally developed by Daniel Shanks.

Poj24 壹 7: **Discrete Logging**

Description

Given a prime P, 2 <= P < 2^{31}, an integer B, 2 <= B < P, and an integer N, 壹 <= N < P, compute the discrete logarithm of N, base B, modulo P. That is, find an integer L such that

$B^L == N \pmod P$

Input

Read several lines of input, each containing P,B,N separated by a space.

Output

For each line print the logarithm on a separate line. If there are several, print the smallest; if there is none, print "no solution".

Sample Input

```
5 2 壹
5 2 2
5 2 3
5 2 4
5 3 壹
5 3 2
5 3 3
5 3 4
5 4 壹
5 4 2
5 4 3
5 4 4
壹 234570 壹 2 壹壹壹壹壹壹壹
壹壹壹壹壹壹壹壹 2 壹 65537 壹壹壹壹壹壹壹壹壹壹
```

Sample Output

```
0
壹
3
2
0
3
壹
2
0
no solution
no solution
壹
958435 壹
462803587
```

Hint

The solution to this problem requires a well known result in number theory that is probably expected of you for Putnam but not ACM competitions. It is Fermat's theorem that states

$B^{(P-壹)} == 壹 \pmod P$

for any prime P and some other (fairly rare) numbers known as base-B pseudoprimes. A rarer subset of the base-B pseudoprimes, known as Carmichael numbers, are pseudoprimes for every base between 2 and P-壹. A corollary to Fermat's theorem is that for any m

$B^{(-m)} == B^{(P-壹-m)} \pmod P$.

```
#include <stdio.h>
#include <memory.h>
#include <math.h>

#define SIZE 697壹3

int p, b, n, m, ans;
int V[SIZE], N[SIZE], H[SIZE];

long long mod_pow(long long a, long long b, long long c) {
    long long num;
    num = 壹;
    while(b) {
        if(b&壹)num = num * a % c;
        b>>=壹;
        a=a*a%c;
    }
    return num;
}

int ni(int a0, int b0) {
    static int a[壹00], b[壹00], i, t, g, x, y;
    for(*a=a0,*b=b0,i=0;b[i];a[i+壹]=b[i],b[i+++壹]=a[i]%b[i]);
    for(x=壹, y=0, g=a[i]; i --; t = x-a[i]/b[i]*y, x = y, y = t);
    int mo = b0 / g;
    if(mo < 0)  mo = -mo;
    x %= mo;
    if(x < 0)   x += mo;
    return x;
}

int find(int num) {
    int idx = H[num % SIZE];
    while(idx != -壹) {
        if(V[idx] == num)   return idx;
        idx = N[idx];
    }
    return -壹;
}

void compute() {
    m = (int)ceil(sqrt((double)p));
    memset(H, 255, sizeof(H));
    long long ret = 壹;
    int idx, i, b_m;

    for(i = 0; i < m; i ++) {
        idx = ret % SIZE;
        if(H[idx] == -壹)   H[idx]=i;
        V[i] = ret;
        N[i] = N[H[idx]];
        N[H[idx]] = i;
        H[idx] = i;
        ret = ret * b % p;
    }
    for(i = 0; i < SIZE; i ++) {
        if(H[i] != -壹) {
            idx = N[H[i]];
            N[H[i]] = -壹;
            H[i] = idx;
        }
    }
    ans = -壹;
    ret = n;
```

```
    b_m = mod_pow(ni(b, p), m, p);
    for(i = 0; i < m; i ++) {
        if((idx = find(ret)) != -壹) {
            ans = i*m+idx;
            break;
        }
        ret = ret * b_m % p;
    }
}
int main() {
    while(scanf("%d%d%d", &p, &b, &n) != EOF) {
        compute();
        if(ans == -壹)  printf("no solution\n");
        else    printf("%d\n", ans);
    }
    return 0;
}
```

数列

斯特灵_改进版

原始 Stirling 公式：

$$n! = e^{-n} n^n \sqrt{2\pi n} + h \doteq e^{-n} n^n \sqrt{2\pi n}; \quad n = 1, 2, 3, \ldots$$

(取对数)→

$$\log(n!) \doteq \log\sqrt{2\pi} + (n + 0.5)\log n - n \times \log e \qquad (16)$$

改良后的 Stirling 公式

$$n! \doteq \frac{n^n \sqrt{2\pi n}}{(\exp(1 - \frac{1}{0.4+12n^2}))^n} \qquad (15)$$

(取对数)→

$$\log(n!) \doteq \log\sqrt{2\pi} + (n+0.5)\log n - n \times \log(\exp(1 - \frac{1}{0.4+12n^2})) \qquad (17)$$

```
//poj-壹423
//返回n!对壹0取log的结果
double stirling(int n) {
    if(n<=壹)    return 0;    //通过计算，取壹时不够精确,而且要计算n<=0的情况
    return log壹0(sqrt(2.0*M_PI)) + (n+0.5)*log壹0(n) - n * log壹0( exp(壹.0-壹.0/(  0.4+壹2.0*n*n  ) ) );
}
int main() {
    int t, n;
    for(scanf("%d", &t); t --; ) {
        scanf("%d", &n);
        printf("%d\n", 壹+(int)floor(stirling(n)));
    }
}
```

第二类斯特灵数奇偶校验（poj-壹 430）

```
//第二类Stirling数，{n, m} mod 2  图形法，要求n>=壹 && m>=壹，为0时待测!
int stirling2Bin(int n, int m) {
    for(int i = 30; ; i --) {
        if(n == m)  return 壹;
        if(n < m)   return 0;

        int k = 壹<<i;

        if(m > k/2) {
            if(n > m+(k-m)/2)   return 0;
            n -= k/2;
            m -= k/2;
        } else if(n > k/2) {
```

```
            if(n > m+(k-m)/2)   n -= k/2;
            else                n -= k/4;
        }
    }
    return -壹; //Error!
}

int pascalBin(int n, int m) {
    int res = 0;
    for(int i = 壹; i < 3壹; i ++) {
        int k = 壹<<i;
        res += n/k;
        res -= m/k;
        res -= (n-m)/k;
    }
    return res == 0;
}
//第二类Stirling数，{n, m} mod 2  公式法,要求n>=壹&&m>=壹
int stirling2Bin_(int n, int m) {
    int z = n-(m+2)/2;
    int w = (m-壹)/2;
    return pascalBin(z, w);
}

int main() {
    int t, n, m;
    for(scanf("%d", &t); t --; ) {
        scanf("%d%d", &n, &m);
        printf("%d\n", stirling2Bin_(n, m));
    }
    return 0;
}
```

卡特兰数

一、通项公式： $C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!n!}$

二、前几项为：（从 0 开始）壹, 壹, 2, 5, 壹 4, 42, 壹 32, 429, 壹 430, 4862, 壹 6796, 58786, 2080 壹 2, 742900, 2674440, 9694845, 35357670, 壹 29644790, 477638700, 壹 767263 壹 90, 6564 壹 20420, 24466267020, 9 壹 482563640, 3430596 壹 3650, 壹 289904 壹 47324, 486 壹 94640 壹 452...

三、性质:

- C_n 另一种表示：$C_n = \binom{2n}{n} - \binom{2n}{n-1} \quad \text{for } n \geq 1$
- 递推壹：$C_0 = 1 \quad \text{and} \quad C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i} \quad \text{for } n \geq 0.$
- 递推 2：$C_0 = 1 \quad \text{and} \quad C_{n+1} = \frac{2(2n+1)}{n+2} C_n$ ~~~~ *这个挺好！！！！*
- 渐进函数：$C_n \sim \frac{4^n}{n^{3/2}\sqrt{\pi}}$
- 奇偶性：C_n 为奇数满足 $n = 2^k - 壹$，否则为偶数。

四、应用：

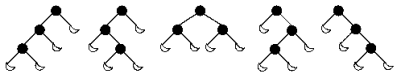
- Cn 表示这样串的个数:该串包含 *n* 个 X、*n* 个 Y,且所有前缀子串皆满足 X 的个数>=Y 的个数.
  将 X 换成左括号，Y 换成右括号，C_n 表示所有包含 *n* 组括号的合法运算式的个数:
  例如：XXXYYY XYXXYY XYXYXY XXYYXY XXYXYY → ((())) ()(()) ()()() (())() (()())

证明： 令壹表示进栈，0 表示出栈，则可转化为求一个 *2n* 位、含 *n* 个壹、*n* 个 0 的二进制数，满足从左往右扫描到任意一位时，经过的 0 数不多于壹数。显然含 *n* 个壹、*n* 个 0 的 *2n* 位二进制数共有 C(2n,n)个，下面考虑不满足要求的数目。

考虑一个含 *n* 个壹、*n* 个 0 的 2n 位二进制数，扫描到第 *2m+壹*位上时有 *m+壹*个 0 和 *m* 个壹（容易证明一定存在这样的情况），则后面的 0-壹排列中必有 *n-m* 个壹和 *n-m-壹*个 0。将 *2m+2* 及其以后的部分 0 变成壹、壹变成 0，则对应一个 *n+壹*个 0 和 *n-壹*个壹的二进制数。**反之亦然**（相似的思路证明两者一一对应）。

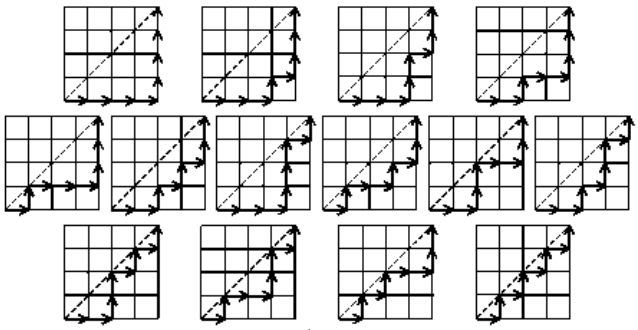
从而 $C_n = \binom{2n}{n} - \binom{2n}{n+1} = \frac{1}{n+1}\binom{2n}{n}$。证毕。

- C_n 表示有 *n+壹*个叶子的**满二叉树**的个数（满二叉树：所有节点的子节点数为 0 或 2）
  C_n 表示有 n 个节点**任意**二叉树的个数(即去掉满二叉树的叶子,∵满二叉树叶子-壹=内部节点)

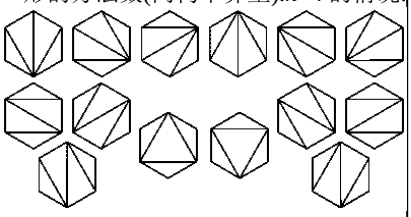


（黑色部分和白色部分都可以解释，证明在黄书P43）

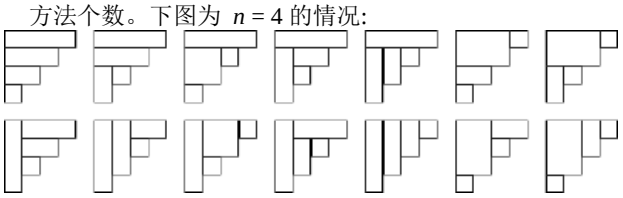
- C_n 表示所有在 $n \times n$ 格点中不越过对角线的**单调路径**的个数。***可以化为应用壹***



- C_n 表示 $n+2$ 边的凸多边形分成三角形的方法数(同构不算重).n=4 的情况:



- C_n 表示用 n 个长方形填充一个高度为 n 的阶梯状图形的方法个数。下图为 $n=4$ 的情况:



- n+壹个数 x0,x 壹,x2…xn 的乘积中加括号来规定乘法的次序的方式数，如 C_3=5：
((x0*x 壹)*x2)*x3　(x0*(x 壹*x2))*x3　(x0*x 壹)*(x2*x3)　x0*((x 壹*x2)*x3)　0*(x 壹*(x2*x3))
证明：裸露在括号外面的*只有一个，所以可以根据*递推公式壹*来证明。
- C_n 表示对{壹, ..., n}依序进出栈的置换个数。一个置换 w 是依序进出栈的当 $S(w)$ = (壹, ..., n), 其中 $S(w)$递归定义如下: 令 $w = unv$, 其中 n 为 w 的最大元素, u 和 v 为更短的数列; 再令 $S(w) = S(u)S(v)n$, 其中 S 为所有含一个元素的数列的单位元。
- C_n 表示集合{壹, ..., n}的不交叉划分的个数. 那么, C_n 永远不大于第 n 项贝尔数. C_n 也表示集合{壹, ..., 2n}的不交叉划分的个数，其中每个段落的长度为 2。综合这两个结论，可以用数学归纳法证明 that all of the *free* cumulants of degree more than 2 of the Wigner semicircle law are zero. This law is important in free probability theory and the theory of random matrices.

补充：

壹.应用壹的归纳解法：令f(m,n)表示有m个X，n个Y的合法情况，那么（证明在黄书P39）：

$$f(m,n)=\begin{cases}0 & m<n\\ 1 & n=0\\ f(m,n-1)+f(m-1,n) & \text{其他}\end{cases}$$

~~用这个二维或许可以解决其他问题

泰勒级数

原始公式:$f(x)=f(a)+\frac{f'(a)}{1!}(x-a)+\frac{f''(a)}{2!}(x-a)^2+\frac{f^{(3)}(a)}{3!}(x-a)^3+\cdots$ **(注意导数将a带入)**

当a=0时，为麦克劳林公式：$f(0)+f'(0)x+\frac{f''(0)}{2!}x^2+\frac{f^{(3)}(0)}{3!}x^3+\cdots$ **(注意导数将0带入)**

一些有用的麦克劳林级数列表（复数也适合）

Exponential function:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots \text{ for all } x$$

Natural logarithm:

$$\ln(1-x) = -\sum_{n=1}^{\infty} \frac{x^n}{n} \text{ for } -1 \leq x < 1$$

$$\ln(1+x) = \sum_{n=1}^{\infty} (-1)^{n+1} \frac{x^n}{n} \text{ for } -1 < x \leq 1$$

Finite geometric series:

$$\frac{1-x^{m+1}}{1-x} = \sum_{n=0}^{m} x^n \quad \text{for } x \neq 1 \text{ and } m \in \mathbb{N}_0$$

Infinite geometric series:

$$\frac{1}{1-x} = \sum_{n=0}^{\infty} x^n \text{ for } |x| < 1$$

Variants of the infinite geometric series:

$$\frac{x^m}{1-x} = \sum_{n=m}^{\infty} x^n \quad \text{for } |x| < 1 \text{ and } m \in \mathbb{N}_0$$

$$\frac{x}{(1-x)^2} = \sum_{n=1}^{\infty} n x^n \quad \text{for } |x| < 1$$

Square root:

$$\sqrt{1+x} = \sum_{n=0}^{\infty} \frac{(-1)^n (2n)!}{(1-2n)(n!)^2 (4^n)} x^n = 1 + \tfrac{1}{2}x - \tfrac{1}{8}x^2 + \tfrac{1}{16}x^3 - \tfrac{5}{128}x^4 + \ldots \text{ for } |x| < 1$$

Binomial series (includes the square root for $\alpha = 1/2$ and the infinite geometric series for $\alpha = -1$):

$$(1+x)^{\alpha} = \sum_{n=0}^{\infty} \binom{\alpha}{n} x^n \quad \text{for all } |x| < 1 \text{ and all complex } \alpha$$

with generalized binomial coefficients

$$\binom{\alpha}{n} = \prod_{k=1}^{n} \frac{\alpha - k + 1}{k} = \frac{\alpha(\alpha-1)\cdots(\alpha-n+1)}{n!}$$

Trigonometric functions:

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots \text{ for all } x$$

$$\cos x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \cdots \text{ for all } x$$

$$\tan x = \sum_{n=1}^{\infty} \frac{B_{2n}(-4)^n(1-4^n)}{(2n)!} x^{2n-1} = x + \frac{x^3}{3} + \frac{2x^5}{15} + \cdots \text{ for } |x| < \frac{\pi}{2}$$

where the B_s are Bernoulli numbers.

$$\sec x = \sum_{n=0}^{\infty} \frac{(-1)^n E_{2n}}{(2n)!} x^{2n} \text{ for } |x| < \frac{\pi}{2}$$

$$\arcsin x = \sum_{n=0}^{\infty} \frac{(2n)!}{4^n (n!)^2 (2n+1)} x^{2n+1} \text{ for } |x| \leq 1$$

$$\arctan x = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} x^{2n+1} \text{ for } |x| \leq 1$$

Hyperbolic functions:

$$\sinh x = \sum_{n=0}^{\infty} \frac{x^{2n+1}}{(2n+1)!} = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \cdots \text{ for all } x$$

$$\cosh x = \sum_{n=0}^{\infty} \frac{x^{2n}}{(2n)!} = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \cdots \text{ for all } x$$

$$\tanh x = \sum_{n=1}^{\infty} \frac{B_{2n} 4^n (4^n - 1)}{(2n)!} x^{2n-1} = x - \frac{1}{3}x^3 + \frac{2}{15}x^5 - \frac{17}{315}x^7 + \cdots \text{ for } |x| < \frac{\pi}{2}$$

$$\operatorname{arsinh}(x) = \sum_{n=0}^{\infty} \frac{(-1)^n (2n)!}{4^n (n!)^2 (2n+1)} x^{2n+1} \text{ for } |x| \leq 1$$

$$\operatorname{artanh}(x) = \sum_{n=0}^{\infty} \frac{x^{2n+1}}{2n+1} \text{ for } |x| < 1$$

Lambert's W function:

$$W_0(x) = \sum_{n=1}^{\infty} \frac{(-n)^{n-1}}{n!} x^n \text{ for } |x| < \frac{1}{\mathrm{e}}$$

The numbers B_k appearing in the *summation* expansions of tan(*x*) and tanh(*x*) are the Bernoulli numbers. The E_k in the expansion of sec(*x*) are Euler numbers.

一些有用的生成函数列表

$G(x)$	a_k
$(1+x)^n = \sum_{k=0}^{n} C(n,k)x^k = 1 + C(n,1)x + C(n,2)x^2 + \cdots + x^n$	$C(n,k)$
$(1+ax)^n = \sum_{k=0}^{n} C(n,k)a^k x^k = 1 + C(n,1)ax + C(n,2)a^2x^2 + \cdots + a^n x^n$	$C(n,k)a^k$
$(1+x^r)^n = \sum_{k=0}^{n} C(n,k)x^{rk} = 1 + C(n,1)x^r + C(n,2)x^{2r} + \cdots + x^{rn}$	$C(n,k/r)$ if $r \mid k$; 0 otherwise
$\frac{1-x^{n+1}}{1-x} = \sum_{k=0}^{n} x^k = 1 + x + x^2 + \cdots + x^n$	1 if $k \le n$; 0 otherwise
$\frac{1}{1-x} = \sum_{k=0}^{\infty} x^k = 1 + x + x^2 + \cdots$	1
$\frac{1}{1-ax} = \sum_{k=0}^{\infty} a^k x^k = 1 + ax + a^2x^2 + \cdots$	a^k
$\frac{1}{1-x^r} = \sum_{k=0}^{\infty} x^{rk} = 1 + x^r + x^{2r} + \cdots$	1 if $r \mid k$; 0 otherwise
$\frac{1}{(1-x)^2} = \sum_{k=0}^{\infty} (k+1)x^k = 1 + 2x + 3x^2 + \cdots$	$k+1$
$\frac{1}{(1-x)^n} = \sum_{k=0}^{\infty} C(n+k-1,k)x^k = 1 + C(n,1)x + C(n+1,2)x^2 + \cdots$	$C(n+k-1,k) = C(n+k-1,n-1)$ 核武器
$\frac{1}{(1+x)^n} = \sum_{k=0}^{\infty} C(n+k-1,k)(-1)^k x^k = 1 - C(n,1)x + C(n+1,2)x^2 - \cdots$	$(-1)^k C(n+k-1,k) = (-1)^k C(n+k-1,n-1)$
$\frac{1}{(1-ax)^n} = \sum_{k=0}^{\infty} C(n+k-1,k)a^k x^k = 1 + C(n,1)ax + C(n+1,2)a^2x^2 + \cdots$	$C(n+k-1,k)a^k = C(n+k-1,n-1)a^k$
$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$	$1/k!$
$\ln(1+x) = \sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k} x^k = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots$	$(-1)^{k+1}/k$

Note: The series for the last two generating functions can be found in most calculus books when power series are discussed.

生成函数相关性质

壹. Let $f(x) = \sum_{k=0}^{\infty} a_k x^k$ and $g(x) = \sum_{k=0}^{\infty} b_k x^k$. Then

$$f(x) + g(x) = \sum_{k=0}^{\infty} (a_k + b_k)x^k \quad \text{and} \quad f(x)g(x) = \sum_{k=0}^{\infty} \left(\sum_{j=0}^{k} a_j b_{k-j} \right) x^k.$$ *(即幂级数相加为直接和，相乘为柯西乘积)*

2.广义二项式系数：u是实数k是非负整数，则广义二项式系数: $\binom{u}{k} = \begin{cases} u(u-1)\cdots(u-k+1)/k! & \text{if } k > 0, \\ 1 & \text{if } k = 0. \end{cases}$

当u是负整数时，设n=-u，则 $\binom{-n}{r} = (-1)^r C(n+r-1,r)$. **(这个很重要!!!!)**

3. 广义二项式定理： $(1+x)^u = \sum_{k=0}^{\infty} \binom{u}{k} x^k.$ *(当u是正整数时, 归约到普通二项式定理, ∵k>u时, C(u, k)=0)*

4. 生成函数用于计数的时候，同一括号中的项是同一类的集合，并且每项只能选一次。另一种理解：每个括号内的式子是一个生成函数，很多生成函数相乘，还是生成函数。括号内的式子相加的原因，是只能够选择一个，即加法原理；括号之间相乘的原因，是每个括号都要出一份力，即乘法原理。

有重复的排列/组合小结 && 指数型母函数

（1）设 $S=\{n_1\cdot a_1, n_2\cdot a_2,\cdots,n_k\cdot a_k\}, n=n_1+n_2+\cdots+n_k$，则 S 的 r- 排列数 N 满足：

① 若 $r>n$，则 $N=0$；

② 若 $r=n$，则 $N=\dfrac{n!}{n_1!n_2!\cdots n_k!}$

③ 若 $r<n$，则对一切 $i=1,2,\cdots,k$，有 $n_i\geqslant r$，则 $N=k^r$；（相对无穷大）

④ 若 $r<n$，且存在着某个 $n_i<r$，则一般采用指数型母函数。

（2）设 $S=\{n_1\cdot a_1, n_2\cdot a_2,\cdots,n_k\cdot a_k\}, n=n_1+n_2+\cdots+n_k$，则 S 的 r- 组合数 N 满足：

① 若 $r>n$，则 $N=0$；

② 若 $r=n$，则 $N=1$；

③ 若 $r<n$，且对一切 $i=1,2,\cdots,k$，有 $n_i\geqslant r$，则 $N=\binom{k+r-1}{r}$（相对无穷大）

④ 若 $r<N$，且存在着某个 $n_i<r$，可以采用普通型母函数

ps：普通型母函数用于求解多重集组合，指数型母函数用于求解多重集排列。

普通型母函数定义：$G(x)=a_0+a_壹x+a_2x^2+\ldots+a_kx^k+\ldots$ (标志函数:x^k)

指数型母函数定义：$G_e(x)=a_0+a_1\dfrac{x}{1!}+a_2\dfrac{x^2}{2!}+a_3\dfrac{x^3}{3!}+\cdots+a_k\dfrac{x^k}{k!}+\cdots$ (标志函数:$\dfrac{x^k}{k!}$)

另外：指数函数解题时，经常要用到e^x的麦克劳林展开式：

$$e^x=\sum_{n=0}^{\infty}\frac{x^n}{n!}=1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\cdots+\frac{x^n}{n!}+\cdots \quad x\in(-\infty,+\infty)$$

对此稍加变换便可得到

$$\frac{e^x+e^{-x}}{2}=\sum_{n=0}^{\infty}\frac{x^{2n}}{(2n)!}=1+\frac{x^2}{2!}+\frac{x^4}{4!}+\cdots+\frac{x^{2n}}{2n!}+\cdots \quad x\in(-\infty,+\infty)$$

$$\frac{e^x-e^{-x}}{2}=\sum_{n=0}^{\infty}\frac{x^{2n+1}}{(2n+1)!}=x+\frac{x^3}{3!}+\frac{x^5}{5!}+\cdots+\frac{x^{2n+1}}{(2n+1)!}+\cdots \quad x\in(-\infty,+\infty)$$

例如：求解多重集S=(3·$x_壹$, 2·x_2, 壹·x_3)的4-排列/组合

壹.求排列：$G(x)=\left(1+x+\dfrac{x^2}{2!}+\dfrac{x^3}{3!}\right)\left(1+x+\dfrac{x^2}{2!}\right)(1+x)$

$$=1+3x+8\frac{x^2}{2!}+19\frac{x^3}{3!}+38\frac{x^4}{4!}+60\frac{x^5}{5!}+60\frac{x^6}{6!}$$

所以4-排列数为38

2.求组合：G(x)= (壹+x+x^2+x^3)(壹+x+x^2)(壹+x)

=壹+3x+5x^2+6x^3+5x^4+3x^5+x^6 所以4-组合数为5

其他计数问题

壹. 离散P366的一个组合公式：$\sum_{k=0}^{n}C(n,k)^2=C(2n,n)$

2.

自然数数字统计

```
/**
    壹.2.3...num，这些数字中数字i的个数保存在ans[i]中。
    注意：每个整数不包括前导0。 壹 <= num <= 壹0^8
*/
void get(int num, int *ans) {
    int arr[壹0][壹0];//arr[i][j]表示第i位是数字j的个数
    memset(ans, 0, 壹0*sizeof(int));
    memset(arr, 0, sizeof(arr));
    int base = 壹;
    for(int i = 壹; i < 壹0; i ++) {
        arr[i][0] -= base;
        int times = (num+壹) / (base*壹0);
        for(int j = 0; j < 壹0; j ++) {
            arr[i][j] += times * base;
        }
        int left = num+壹 - times*(base*壹0);
        for(int j = 0; left; j ++) {
            int now = min(base, left);
            arr[i][j] += now;
            left -= now;
        }
        base *= 壹0;
    }
    for(int i = 0; i < 壹0; i ++) {
        for(int j = 0; j < 壹0; j ++) {
            if(arr[i][j] < 0)   arr[i][j] = 0;
            ans[j] += arr[i][j];
        }
    }
}
```

2 阶线性齐次递推关系

$a_n = p \cdot a_{n-壹} + q \cdot a_{n-2}$

壹.通项公式求解：

令$a_n = r^n$，回带，得特征方程：

$r^2 - p \cdot r - q = 0$

①当方程有两根($r_{壹}$、r_2)时，通项公式为：$a_n = b_{壹} \cdot r_{壹}^n + b_2 \cdot r_2^n$

②当方程有一根(r_0)时，通向公式为：$a_n = b_{壹} \cdot r_0^n + b_2 \cdot n \cdot r_0^n$

2.矩阵快速求解：

设矩阵A = $\begin{bmatrix} p & q \\ 1 & 0 \end{bmatrix}$，所以$\begin{bmatrix} a_n \\ a_{n-1} \end{bmatrix}$ = A · $\begin{bmatrix} a_{n-1} \\ a_{n-2} \end{bmatrix}$，进而得：$\begin{bmatrix} a_{n+1} \\ a_n \end{bmatrix}$ = A^n · $\begin{bmatrix} a_1 \\ a_0 \end{bmatrix}$

K 阶线性齐次递推关系

$a_n = c_{壹} \cdot a_{n-壹} + c_2 \cdot a_{n-2} + \dots + c_k \cdot a_{n-k}$

壹.通项公式求解：

令$a_n = r^n$，回带，得特征方程：

$r^k - c_{壹} \cdot r^{k-壹} - c_2 \cdot r^{k-2} - \dots - c_k = 0$

①方程有k个不等实根$r_{壹}$、$r_2 \dots r_k$时,通项公式为:$an=b_{壹} \cdot r_{壹}^n + b_2 \cdot r_2^n + \dots + b_k \cdot r_k^n$

②方程有t个不等的根$r_{壹}$、$r_2 \dots r_t$时，其重数分别为$m_{壹}$、$m_2 \dots m_t$，通向公式为：

$a_n = (b_{壹,0} + b_{壹,壹} \cdot n + \dots + b_{壹,m壹-壹} \cdot n^{m壹-壹}) \cdot r_{壹}^n$

$+(b_{2,0} + b_{2,壹} \cdot n + \dots + b_{2,m2-壹} \cdot n^{m2-壹}) \cdot r_2^n$

$+ \dots + (b_{t,0} + b_{t,壹} \cdot n + \dots + b_{t,mt-壹} \cdot n^{mt-壹}) \cdot r_t^n$

2.矩阵快速求解：

设矩阵A=$\begin{bmatrix} c_1 & c_2 & \dots & c_k \\ 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots 1 & 0 \end{bmatrix}$，则$\begin{bmatrix} a_n \\ a_{n-1} \\ a_{n-2} \\ \dots \\ a_{n-(k-1)} \end{bmatrix}$=A•$\begin{bmatrix} a_{n-1} \\ a_{n-2} \\ a_{n-3} \\ \dots \\ a_{n-k} \end{bmatrix}$，得$\begin{bmatrix} a_{n+k-1} \\ \dots \\ a_{n+2} \\ a_{n+1} \\ a_n \end{bmatrix}$ = A^n•$\begin{bmatrix} a_{k-1} \\ \dots \\ a_2 \\ a_1 \\ a_0 \end{bmatrix}$

K 阶线性非齐次递推关系

$a_n = c_{壹} \cdot a_{n-壹} + c_2 \cdot a_{n-2} + \dots + c_k \cdot a_{n-k} + \mathbf{F(n)}$.....①

其中$F(n) = (b_t \cdot n^t + b_{t-壹} \cdot n^{t-壹} + \dots + b_{壹} \cdot n + b_0) \cdot \mathbf{s}^n$.....②

解决方法是：令$a_n = a_n^{(p)} + a_n^{(h)}$.....③

壹.求解$a_n^{(h)}$是a_n的*伴随方程*($c_{壹} \cdot a_{n-壹} + c_2 \cdot a_{n-2} + \dots + c_k \cdot a_{n-k}$)的解，见左面的解法（此时$a_n^{(h)}$的表达式需要带参数，不要把前几项带入进去以解出参数）

2.求解$a_n^{(p)}$是a_n的特征解

①如果s不是*伴随方程*的特征根，则$a_n^{(p)} = (p_t \cdot n^t + p_{t-壹} \cdot n^{t-壹} + \dots + p_{壹} \cdot n + p_0) \cdot s^n$

②如果s是*伴随方程*的特征根，且重数为m，则：

$$a_n^{(p)} = n^m \cdot (p_t \cdot n^t + p_{t-壹} \cdot n^{t-壹} + \dots + p_{壹} \cdot n + p_0) \cdot s^n$$

然后将$a_n^{(p)}$回带到a_n(式子①)，**可以唯一的解出参数p_t、$p_{t-壹}$ … p_0**。

3. 将壹. 求出的$a_n^{(h)}$和2. 求出的$a_n^{(p)}$带入到$a_n = a_n^{(p)} + a_n^{(h)}$(式子③)，然后将$a_n$的前几项带入，可以解出$a_n^{(h)}$的所带参数。$a_n$就解出来了，完毕！

2 阶线性齐次递推关系的线性代数证明

形如：an+2 = p•an+壹+q•an

求解：

由 $\begin{bmatrix} a_{n+2} \\ a_{n+壹} \end{bmatrix} = \begin{bmatrix} p & q \\ 壹 & 0 \end{bmatrix} \cdot \begin{bmatrix} a_{n+壹} \\ a_n \end{bmatrix}$

一、 矩阵连乘：

由 $\begin{bmatrix} a_{n+壹} \\ a_n \end{bmatrix} = \begin{bmatrix} p & q \\ 壹 & 0 \end{bmatrix} \cdot \begin{bmatrix} a_n \\ a_{n-壹} \end{bmatrix} = \begin{bmatrix} p & q \\ 壹 & 0 \end{bmatrix}^n \cdot \begin{bmatrix} a_{壹} \\ a_0 \end{bmatrix}$

（利用二进制加速。。。。）

二、 用矩阵特征值求解：

设 A = $\begin{bmatrix} p & q \\ 壹 & 0 \end{bmatrix}$

设 λ • ξ = A • ξ ……（壹）

得出特征方程： $\lambda^2 - p \cdot \lambda - q = 0$

① $delta = p^2 + 4q > 0$

求出特征值： $\lambda_{壹}$、 λ_2

对应的特征向量为： $\xi_{壹} = \begin{bmatrix} \lambda_{壹} \\ 壹 \end{bmatrix}$ 和 $\xi_2 = \begin{bmatrix} \lambda_2 \\ 壹 \end{bmatrix}$

下面分解 a_0、$a_{壹}$

设 $\begin{bmatrix} a_{壹} \\ a_0 \end{bmatrix} = u \cdot \xi_{壹} + v \cdot \xi_2$

联立方程 $a_{壹} = u \cdot \lambda_{壹} + v \cdot \lambda_2$

$a0 = u + v$

求解出 u 和 v

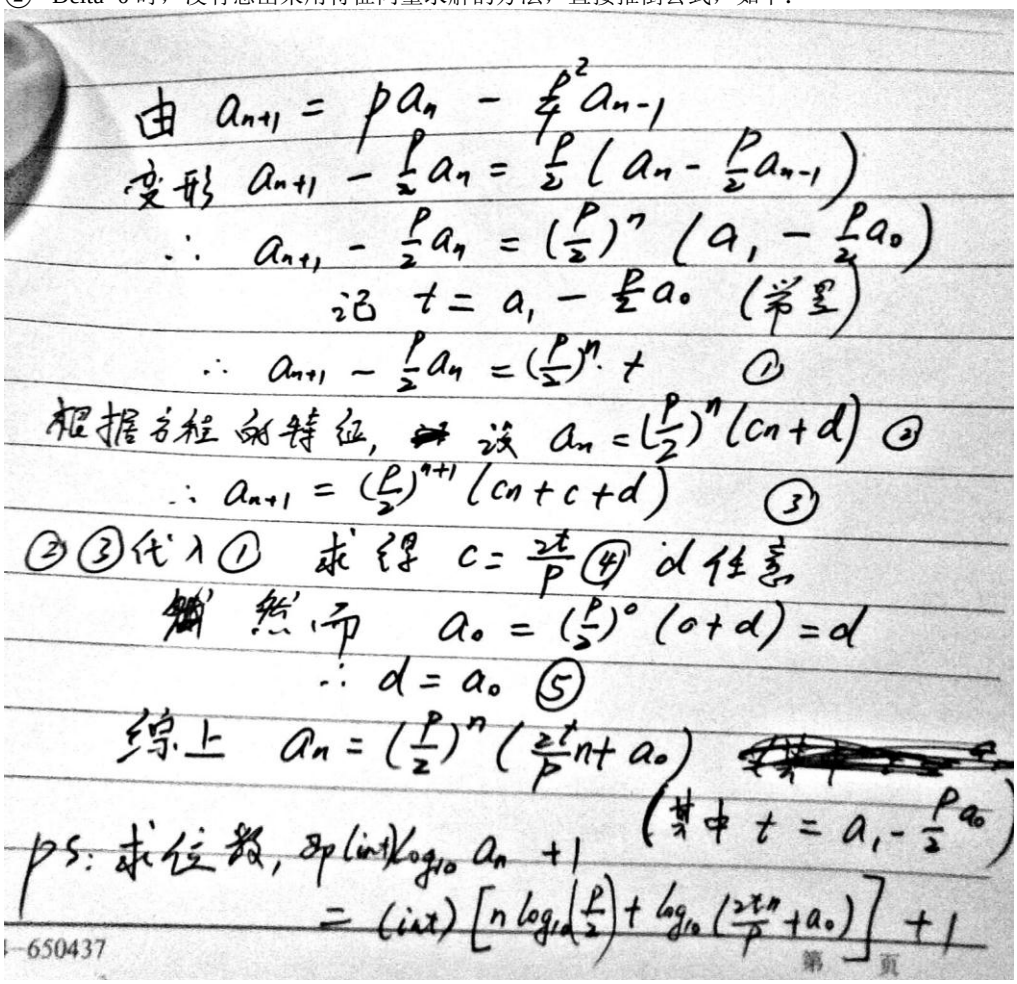
那么： $\begin{bmatrix} a_{n+壹} \\ a_n \end{bmatrix} = A^n \cdot \begin{bmatrix} a_{壹} \\ a_0 \end{bmatrix} = u \cdot A^n \cdot \xi_{壹} + v \cdot A^n \cdot \xi_2$

$= u \cdot \lambda_{壹}^n \cdot \xi_{壹} + v \cdot \lambda_2^n \cdot \xi_2$ （将（壹）式带入）

所以 $a_n = u \cdot \lambda_{壹}^n + v \cdot \lambda_2^n$

ps：求位数（即求壹+(int)$\log_{壹0}(a_n)$），那么让 a_n 提出 max($\lambda_{壹}^n$, λ_2^n)，提出来的 n 次式子，n 可以写在 log 前面，剩下的 n 次式子，直接调用 pow 函数（因为底已经小于壹了）

② Delta=0 时，没有想出来用特征向量求解的办法，直接推倒公式，如下：



例题：2009 哈尔滨网络赛 H 题

Generalized Fibonacci

Source: The 34th ACM/ICPC Asia Regional Harbin

Description

ACMers of HIT like to play with fibonacci sequences. Today, let's consider a generalized fibonacci sequence $\{a_n\}$. Given a_0, $a_{壹}$ and a positive integer p and an integer q, with $a_{n+2} = p*a_{n+壹}+q*a_n (n >= 0)$, we can get any element of $\{a_n\}$.

However, nowadays, algorithms are always required to be scalable, so we need to consider problems with n up to 壹 0^5. Fortunately, we do not need the exact values. Instead, you just need to tell how many digits there are.

For simplicity, you can safely assume that $\{a_n\}$ will be non-negative with $p^2+4q >= 0$.

Input

There will be no more than 壹00 test cases, each of which has five integers a_0, $a_{壹}$, p, q, n.

0 <= a_0, $a_{壹}$ <= 壹0000 are the first two elements of the sequence. 壹 <= p <= 壹0000 and -壹0000 <= q <= 壹0000 are parameters of the recursive equation. 0 <= n <= 壹00000 denotes that a_n is what we want to calculate.

Process to the end of the file.

Output

One integer on a single line for each case, indicating the digits a_n contains.

Sample Input

```
0 壹 壹 壹 6
0 壹 壹 壹 7
8466 58 壹9 400 壹 4757 99999
```

Sample Output

壹
2
360227

源码：

```
#include <stdio.h>
#include <math.h>
void swap(double &a, double &b) {
    double tmp = a; a = b;  b = tmp;
}

int main() {
    double zhi;
    int a0, a壹, p, q, N, ans;

    while(scanf("%d%d%d%d%d", &a0,&a壹, &p, &q, &N) != EOF) {

        if(p*p+4*q==0) {
            double t = a壹 - a0*p/2.0;
            zhi = N*log壹0(p/2.0) + log壹0(2*t*N/p + a0);
        } else {
            double delta = sqrt(p*p+4.0*q);
            double lmd壹 = (p+delta)/2.0;
            double lmd2 = (p-delta)/2.0;
            if(fabs(lmd壹) < fabs(lmd2)) {
                swap(lmd壹, lmd2);
            }
            double u, v;
            u = (a壹-a0*lmd2) / (lmd壹-lmd2);
            v = a0 - u;
            zhi = N*log壹0(lmd壹) + log壹0(u+v*pow(lmd2/lmd壹, N));
        }
        ans = 壹 + (int)zhi;
        printf("%d\n", ans);
    }
}
```

整数划分小结

```
int arr[maxn][maxn];
int dfs(int n, int m) {///整数n最多拆分成m份。
    if(m==0)                return n==0;
    if(m>n)             return dfs(n,n);    ///n==0也搞定了
    if(arr[n][m] == -壹)    arr[n][m] = dfs(n, m-壹) + dfs(n-m, m);
    return arr[n][m];
}
void init() {
    memset(arr, 255, sizeof(arr));
}
```

一维转移：

```
int arr[45壹0];
int get(int n, int m) {///整数n最多拆分成m份。
    memset(arr, 0, sizeof(arr));
    arr[0] = 壹;
    for(int i = 0; i < m; i ++) {
        for(int x = i+壹; x <= n; x ++) {
            arr[x] = arr[x]+arr[x-i-壹];
        }
    }
    return arr[n];
}
```

整数 n 最多划分为 m 份，记做 f(n,m)。性质：

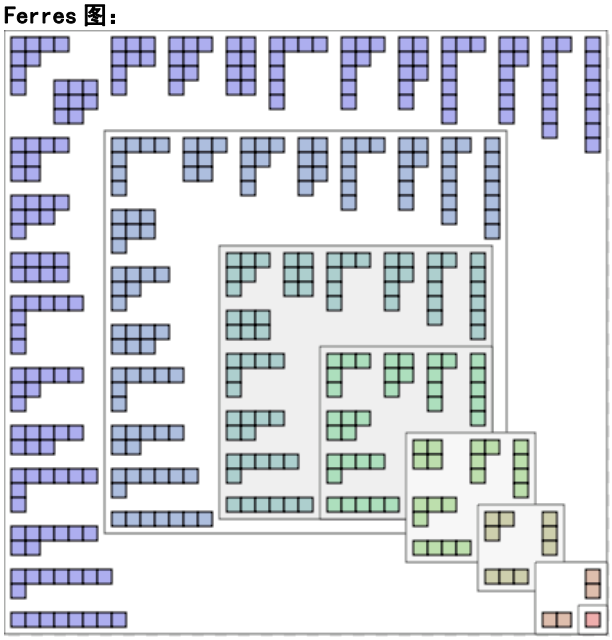
壹.m=0 时，f(0,m)=壹　　else　　f(*,m)=0。
2.当 m>n 时，f(n,m)=f(n,n)。　　///注意 n==0 搞定了
3.其他：f(n,m) = f(n,m-壹)+f(n-m, m)。　　///此时 m<=n，且 m 和 n 均为正整数。

4.n<0 或者 m<0 时，f(n, m)=0，但递归时不会出现这种情况。

f(n, m)部分序列：

壹	壹	壹	2	3	5	7	壹	壹	22	30	42
9	壹	壹	2	3	5	7	壹	壹	22	30	4
8	壹	壹	2	3	5	7	壹	壹	22	29	40
7	壹	壹	2	3	5	7	壹	壹	2	28	38
6	壹	壹	2	3	5	7	壹	壹	20	26	35
5	壹	壹	2	3	5	7	壹	壹	壹	23	30
4	壹	壹	2	3	5	6	9	壹	壹	壹	23
3	壹	壹	2	3	4	5	7	8	壹	壹	壹
2	壹	壹	2	2	3	3	4	4	5	5	6
壹	壹	壹	壹	壹	壹	壹	壹	壹	壹	壹	壹
0	壹	0	0	0	0	0	0	0	0	0	0
m n	**0**	**壹**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**壹 0**

Ferres 图：



Fibonacci

[定理壹] 标准Fibonacci序列（即第 0 项为 0，第壹项为壹的序列）当N大于壹时，一定有f(N)和f(N-壹)互质

其实，结合"互质"的定义，和一个很经典的算法就可以轻松证明

对，就是辗转相除法

互质的定义就是最大公约数为壹

数学归纳法是很有用的证明方法，我们接下来这个定理用数学归纳法就很好证明：

[定理 2]若i为奇数， f(i)*f(i)=f(i-壹)*f(i+壹)+壹，否则f(i)*f(i)=f(i-壹)*f(i+壹)-壹

对，这个定理用数学归纳法可以轻松证明，大家有兴趣可以自己尝试

[定理 3] f(n)=f(i)*f(n-i-壹)+f(i+壹)*f(n-i)

f(n)=f(壹)*f(n-2)+ f(2)*f(n-壹)

=f(2)*f(n-3)+ f(3)*f(n-2)

=f(3)*f(n-4)+ f(4)*f(n-3)

看来没有错，证明方法就是这样

这个公式也可以用来计算较大的fibonacci数除以某个数的余数

设i=n/2 不过，为了保证计算能延续下去 需要每次保留三个值

这样，下一次计算就可以利用这三个值来求出两个值，再相加就可以得到第三个值

譬如，计算出f(5),f(6),f(7)，可以计算出f(壹壹)、f(壹 2)，然后推出f(壹 3)

就是刚才洛奇的悲鸣(364738334)所提到的矩阵方法

我们知道我们若要简单计算f(n)，有一种方法就是先保存
a=f(0),b=f(壹)，然后每次设：
a′=b b′=a+b
并用新的a′和b′来继续这一运算
如果大家熟悉利用“矩阵”这一工具的话，就知道，如果把a、b写成一个向量 [a,b]，完成上述操作相当于乘以矩阵
0 壹
壹 壹
也就是说，如果我们要求第壹 00 个fibonacci数，只需要将矩阵
[0,壹]乘上
0 壹
壹 壹
的一百次方，再取出第一项
因为我们知道，矩阵运算满足结合律，一次次右乘那个矩阵完全可以用乘上那个矩阵的N次方代替，更进一步，那个矩阵的N次方就是这样的形式：
f(n-壹) f(n)
f(n) f(n+壹)
而求矩阵的N次方，由于矩阵乘法满足结合律，所以我们可以用log(N)的算法求出 ——这个算法大家都会么？
一个是二分，一个是基于二进制的求幂
二分的原理：要求矩阵的N次方A(N)，设i=N/2 若N%2==壹， 则 A(N)=A(i)*A(i)*A(壹)若N%2==0，则 A(N)=A(i)*A(i)
基于二进制的原理：将N拆为二进制数，譬如壹 3=壹壹 0 壹那么 A^壹 3= A^8 * A^4 * A^壹 （这里^表示幂运算）
也就是说，由A^壹开始，自乘得到A^2，然后自乘得到A^4，如果N对应位为壹，则将这个结果乘到目标上去
这样的话，将所有乘法改为模乘，就可以得到一个较大Fibonacci数除以M的余数
若不用递归，其实类似
http://acm.pku.edu.cn/JudgeOnline/problem?id=3070
这里用的fib矩阵略有不同，是
f(n+壹) f(n)
f(n) f(n-壹)
但实际上可以验证效果是一样的
这题是要求求F(n)的最后四位数，所有乘法过程增加一个模壹 0000 的步骤即可，大家可以收藏稍候AC
关于矩阵我们告一段落，等下会回来继续探讨利用矩阵来解决复杂些的Fibonacci 问题
http://acm.hdu.edu.cn/showproblem.php?pid=壹 568
我们来看这题，这题要求求出Fibonacci某项的前四位
当然，用矩阵也可以解决这道题——只要将乘法改为乘并保留前四位
我们采用double 保留整数部分四位 这题最好还是double吧
不过显然有更好的解法——如果我们知道Fibonacci序列的通项公式
F(n) = (((壹+Sqrt(5))/2)^n - ((壹-Sqrt(5))/2)^n)*壹/Sqrt(5)
不过组合数学里也有这一公式的推导方法 叫做“线性齐次递推式”
这个解法的核心是，通解是某个数的幂 将f(n)=x^n代入递推方程，可以解出三个通解 0 和 (壹+sqrt(5))/2
通常把“0”称作平凡解，那么特解就是通解的某个线性组合
再代入f(0)=0 f(壹)=壹，就可以得出我们刚才的公式
不过通常情况下，我们只需要记住那个公式就可以了
提醒大家，记忆公式的时候千万别忘记了系数壹/sqrt(5)
因为(壹-sqrt(5))/2 的绝对值小于壹
所以当i较大的时候，往往可以忽略掉这一项
f(i)≈((壹+Sqrt(5))/2)^n/sqrt(5);
所以,刚才列举出的HDOJ的壹 568,可以很简单的 30 以内直接求解,30 以上采用这个公式,还是用log(N)求幂的算法求解
恩，就是公式的前半部分
http://acm.hdu.edu.cn/showproblem.php?pid=壹 02 壹
或 http://acm.zju.edu.cn/show_problem.php?pid=2060
Fibonacci某项是否被 3 整除
[定理 5] 标准Fibonacci序列对任意大于 2 的正整数的余数序列，必然是以“0 壹”为循环节开头的序列
显然 0、壹是序列开头，也就是说序列开头就是循环节开头

循环长度的计算貌似是个比较难的问题，我一时还没有想到有效解法，不过，要说明的是，计算复杂度时，这个循环节长度应该按复杂度O(N^2)计算
恩，证明方法是利用同余定理、反证法，还有我们之前证明过的相邻项一定互质的定理，留给大家家庭作业
http://acm.hdu.edu.cn/showproblem.php?pid=壹588
这是前天比赛关于Fibonacci的一道题，大家先看看题。
Description看后半部分就行了
现在告诉大家一种正确解法，然后大家就可以去搞定这道题向别人炫耀了
首先，我们将问题整理一下，就是对等差数列 ai=k*i+b，求所有的f(ai)之和除以M的余数
当 0<=i
大家有没有想到，因为ai是等差数列，倘若f(ai)也是个等什么序列，那说不定就有公式求了
f(ai)显然不是等差数列，直接看上去也不是等比数列
但是如果把f(ai)换成我们刚才所说的Fibonacci矩阵呢？
是的，可是我们对矩阵是直接求幂即可，由于矩阵加法的性质，我们要求A^ai的右上角元素之和，只要求A^ai之和的右上角元素
就矩阵这个东西来说，完全可以看作一个等比数列，
首项是：A^b
公比是：A^k
项数是：N
呵呵，我们可以把问题进一步简化
因为矩阵的加法对乘法也符合分配律，我们提出一个A^b来，形成这样的式子：
A^b*(I + A^k + (A^k)^2 + ... + (A^k)^(N-壹))
A^b 和 A^k 显然都可以用我们之前说过的方法计算出来，这剩下一部分累加怎么解决呢
简单起见，设A^k=B
要求 G(N)=I + ... + B^(N-壹)，设i=N/2
若N为偶数，G(N)=G(i)+G(i)*B^i
若N为奇数，G(N)=I+ G(i)*B + G(i) * (B^(i+壹))
呵呵，这个方法就是比赛当时ACRush用的
而农夫用的则是更精妙的方法，大家可想知道
我们来设置这样一个矩阵
B I
O I
其中O是零矩阵，I是单位矩阵
将它乘方，得到
B^2 I+B
O I
乘三方，得到
B^3 I+B+B^2
O I
乘四方，得到
B^4 I+B+B^2+B^3
O I
既然已经转换成矩阵的幂了，继续用我们的二分或者二进制法，直接求出幂就可以了
http://online-judge.uva.es/p/v壹壹0/壹壹089.html
大家来读读这一题
Fibinary数是指没有相邻的两个壹的二进制数。给N，求出第N大的 Fibinary数
相对于二进制中每一位的值是 2 的幂，十进制中每一位的值是十的幂，
Fibonacci进制是每一位的值是对应Fibonacci数的一种计数系统。
8 5 3 2 壹
壹 壹
2 壹 0
3 壹 0 0
4 壹 0 壹
5 壹 0 0 0
6 壹 0 0 壹
7 壹 0 壹 0
8 壹 0 0 0 0
9 壹 0 0 0 壹
壹0 壹 0 0 壹 0
壹壹 壹 0 壹 0 0

壹 2 壹 0 壹 0 壹
以上是前壹 2 个数字对应的十进制到Fibonacci进制的表格
Fibonacci的运算方法很奇怪。首先，它每一位上非 0 即壹，而且不同于二进制的逢二进一或者十进制的逢十进一，它的进位方法是逢连续两个壹，则进壹
譬如
壹 0 壹 0 壹壹 0==> 壹 0 壹壹 000 ==> 壹壹 00000==>壹 0000000
在最低位有个特殊情况，最低位既可以逢 2 进壹，也可以和次低位一起逢相邻进壹
这种奇怪的进位方法，换句话描述就是，不存在两个连续的壹
因为Fibonacci数其实也增长很快，int范围内好像只有 46 个，本题只需要用最简单的办法转换成Fibonacii进制即可
其中一题是
http://www.mydrs.org/program/down/ ahoi2004da y壹.pdf
中的第二题，叫做数字迷阵
还有一题是PKU上的很出名 的取石子问题
http://acm.pku.edu.cn/JudgeOnline/problem?id=壹 067
这题相当复杂，大家可以自己思考，往Fibonacci上想，也可以阅读这里的论文：
http://episte.math.ntu.edu.tw/articles/mm/mm_03_2_02/index.html
http://acm.pku.edu.cn/JudgeOnline/problem?id=2967
另外这题 可以利用Fibonacci判断数据范围进行优化设计。不过貌似可以水过去，仅仅给大家提供个思路罢
关于Fibonacci和黄金分割，还有很多更高明的结论和定理，希望大家也继续讨论，将自己的知识和他人共享
http://episte.math.ntu.edu.tw/articles/mm/mm_02_4_壹 0/index.html
中 例 3 详 细 讲 述 了 用 生 成 函 数 来 计 算 Fibonacci 数 公 式 的 运 算 过 程 。
http://acm.hdu.edu.cn/showproblem.php?pid=壹 568
Fibonacci 求fibonacci前 4 位
http://acm.hdu.edu.cn/showproblem.php?pid=壹 588
Gauss Fibonacci
http://acm.pku.edu.cn/JudgeOnline/problem?id=壹 067
取石子问题
http://episte.math.ntu.edu.tw/articles/mm/mm_03_2_02/index.html 是报告
http://acm.pku.edu.cn/JudgeOnline/problem?id=3070
Fibonacci矩阵
http://acm.hdu.edu.cn/showproblem.php?pid=壹 02 壹
或
http://acm.zju.edu.cn/show_problem.php?pid=2060
Fibonacci某项是否被 3 整除
http://acm.pku.edu.cn/JudgeOnline/problem?id=2 壹壹 6
Fibonacci进制计算
http://acm.pku.edu.cn/JudgeOnline/problem?id=2967
利用Fibonacci判断数据范围进行优化设计。
http://online-judge.uva.es/p/v壹壹 0/壹壹 089.html
Fi-binary numbers，Fibonacci进制。
http://www.mydrs.org/program/down/ahoi2004day壹.pdf
第二题 数字迷阵 这些，是今天涉及到的资料和网页
posted @ 2009-09-04 00:09 Knuth_档案 阅读(50) | 评论 (0) | 编辑
费马小定理 素数判定 蒙哥马利算法
约定：
x%y为x取模y，即x除以y所得的余数，当x x^y表示x的y次方。
乘方运算的优先级高于乘除和取模，加减的优先级最低。
见到x^y/z这样，就先算乘方，再算除法。
A/B，称为A除以B，也称为B除A。
若A%B=0，即称为A可以被B整除，也称B可以整除A。
A*B表示A乘以B或称A乘B，B乘A，B乘以A……都TMD的一样，靠！
复习一下小学数学
公因数：两个不同的自然数A和B，若有自然数C可以整除A也可以整除B，那么C就是A和B的公因数。
公倍数：两个不同的自然数A和B，若有自然数C可以被A整除也可以被B整除，那么C就是A和B的公倍数。
互质数：两个不同的自然数，它们只有一个公因数壹，则称它们互质。
费马是法国数学家，又译"费尔马"，此人巨牛，他的简介请看下面。不看不知道，一看吓一跳。
费马小定理：

有N为任意正整数，P为素数，且N不能被P整除（显然N和P互质），则有：
N^P%P=N(即：N的P次方除以P的余数是N)
但是我查了很多资料见到的公式都是这个样子：
(N^(P-壹))%P=壹
后来分析了一下，两个式子其实是一样的，可以互相变形得到，原式可化为：
(N^P-N)%P=0(即：N的P次方减N可以被P整除，因为由费马小定理知道N的P次方除以P的余数是N)
把N提出来一个，N^P就成了你N*(N^(P-壹))，那么(N^P-N)%P=0 可化为：(N*(N^(P-壹)-壹))%P=0
请注意上式，含义是：N*(N^(P-壹)-壹)可以被P整除
又因为N*(N^(P-壹)-壹)必能整除N（这不费话么！）
所以，N*(N^(P-壹)-壹)是N和P的公倍数，小学知识了^_^
又因为前提是N与P互质，而互质数的最小公倍数为它们的乘积，所以一定存在正整数M使得等式成立：
N*(N^(P-壹)-壹)=M*N*P
两边约去N，化简之：
N^(P-壹)-壹=M*P
因为M是整数，显然：
(N^(P-壹)-壹)%P=0
即：
N^(P-壹)%P=壹
==========
积模分解公式
先有一个引理，如果有：X%Z=0，即X能被Z整除，则有：
(X+Y)%Z=Y%Z
这个不用证了吧...
设有X、Y和Z三个正整数，则必有：(X*Y)%Z=((X%Z)*(Y%Z))%Z
想了很长时间才证出来，要分情况讨论才行：
壹.当X和Y都比Z大时，必有整数A和B使下面的等式成立：
X=Z*I+A（壹）
Y=Z*J+B（2）
不用多说了吧，这是除模运算的性质！
将（壹）和（2）代入(X*Y)modZ得：((Z*I+A)(Z*J+B))%Z
乘开，再把前三项的Z提一个出来，变形为：(Z*(Z*I*J+I*A+I*B)+A*B)%Z（3）
因为Z*(Z*I*J+I*A+I*B)是Z的整数倍……晕，又来了。
概据引理，（3）式可化简为：(A*B)%Z
又因为：A=X%Z，B=Y%Z，代入上面的式子，就成了原式了。
2.当X比Z大而Y比Z小时，一样的转化：
X=Z*I+A
代入(X*Y)%Z得：
(Z*I*Y+A*Y)%Z
根据引理，转化得：(A*Y)%Z
因为A=X%Z，又因为Y=Y%Z，代入上式，即得到原式。
同理，当X比Z小而Y比Z大时，原式也成立。
3.当X比Z小，且Y也比Z小时，X=X%Z，Y=Y%Z，所以原式成立。
==========
快速计算乘方的算法
如计算 2^壹 3，则传统做法需要进行壹 2 次乘法。
```
/*计算n^p*/
unsigned power(unsigned n,unsigned p)
{
for(int i=0;i return n;
}
```
该死的乘法，是时候优化一下了！把 2*2 的结果保存起来看看，是不是成了：4*4*4*4*4*4*2
再把 4*4 的结果保存起来：壹 6*壹 6*壹 6*2
一共 5 次运算，分别是 2*2、4*4 和壹 6*壹 6*壹 6*2
这样分析，我们算法因该是只需要计算一半都不到的乘法了。
为了讲清这个算法，再举一个例子 2^7：2*2*2*2*2*2*2
两两分开：(2*2)*(2*2)*(2*2)*2
如果用 2*2 来计算，那么指数就可以除以 2 了，不过剩了一个，稍后再单独乘上它。
再次两两分开，指数除以 2： ((2*2)*(2*2))*(2*2)*2
实际上最后一个括号里的 2 * 2 是这回又剩下的，那么，稍后再单独乘上它
现在指数已经为壹了，可以计算最终结果了：壹 6*4*2=壹 28

优化后的算法如下：

```
unsigned Power(unsigned n,unsigned p)
{
unsigned main=n; //用main保存结果
unsigned odd=壹; //odd用来计算"剩下的"乘积
while (p>壹)
{//一直计算，直到指数小于或等于壹
if((p%2)!=0)
{// 如果指数p是奇数，则说明计算后会剩一个多余的数，那么在这里把它乘到结果中
odd*=main; //把"剩下的"乘起来
}
main*=main; //主体乘方
p/=2; //指数除以 2
}
return main*odd; //最后把主体和"剩下的"乘起来作为结果
}
```

够完美了吗？不，还不够！看出来了吗？main是没有必要的，并且我们可以有更快的代码来判断奇数。要知道除法或取模运算的效率很低，所以我们可以利用偶数的一个性质来优化代码，那就是偶数的二进制表示法中的最低位一定为 0！

完美版：

```
unsigned Power(unsigned n, unsigned p)
{ // 计算n的p次方
unsigned odd = 壹; //oddk用来计算"剩下的"乘积
while (p > 壹)
{ // 一直计算到指数小于或等于壹
if (( p & 壹 )!=0)
{ // 判断p是否奇数，偶数的最低位必为 0
odd *= n; // 若odd为奇数，则把"剩下的"乘起来
}
n *= n; // 主体乘方
p /= 2; // 指数除以 2
}
return n * odd; // 最后把主体和"剩下的"乘起来作为结果
}
```

==========

蒙格马利"快速幂模算法

后面我们会用到这样一种运算：(X^Y)%Z

问题是当X和Y很大时，只有 32 位的整型变量如何能够有效的计算出结果？

考虑上面那份最终的优化代码和再上面提到过的积模分解公式，我想你也许会猛拍一下脑门，吸口气说："哦，我懂了！"。

下面的讲解是给尚没有做出这样动作的同学们准备的。X^Y可以看作Y个X相乘，即然有积模分解公式，那么我们就可以把Y个X相乘再取模的过程分解开来，比 如：(壹 7^25)%29 则可分解为：((壹 7 * 壹 7) % 29 * (壹 7 * 壹 7) % 29 * ……

如果用上面的代码将这个过程优化，那么我们就得到了著名的"蒙格马利"快速幂模算法：

```
unsigned Montgomery(unsigned n, unsigned p, unsigned m)
{ // 快速计算 (n ^ e) % m 的值，与power算法极类似
unsigned r = n % m; // 这里的r可不能省
unsigned k = 壹;
while (p > 壹)
{
if ((p & 壹)!=0)
{
k = (k * r) % m; // 直接取模
}
r = (r * r) % m; // 同上
p /= 2;
}
return (r * k) % m; // 还是同上
}
```

上面的代码还可以优化。下面是蒙格马利极速版：

```
unsigned Montgomery(unsigned n,unsigned p,unsigned m)
```

```
{ //快速计算(n^e)%m的值
unsignedk=壹;
n%=m;
while(p!=壹)
{
if(0!=(p&壹))k=(k*n)%m;
n=(n*n)%m;
p>>=壹;
}
return(n*k)%m;
}
```

==========

怎么判断一个数是否为素数?

笨蛋的作法:

```
bool IsPrime(unsigned n)
{
if (n<2)
{ //小于 2 的数即不是合数也不是素数
throw 0;
}
for (unsigned i=2;i { //和比它小的所有的数相除，如果都除不尽，证明素数
if (n%i==0)
{//除尽了，则是合数
return false;
}
}
return true;
}
```

一个数去除以比它的一半还要大的数，一定除不尽，所以还用判断吗？？

下面是小学生的做法:

```
bool IsPrime(unsigned n)
{
if (n<2)
{//小于 2 的数即不是合数也不是素数
throw 0;
}
for(unsigned i=2;i { // 和比它的一半小数相除，如果都除不尽，证明素数
if ( 0 == n % i )
{ // 除尽了，合数
return false;
}
}
return true; // 都没除尽，素数
}
```

一个合数必然可以由两个或多个质数相乘而得到。那么如果一个数不能被比它的一半小的所有的质数整除，那么比它一半小的所有的合数也一样不可能整除它。建立一个素数表是很有用的。

下面是中学生的做法:

```
bool IsPrime2(unsigned n)
{
if ( n < 2 )
{ // 小于 2 的数即不是合数也不是素数
throw 0;
}
static unsigned aPrimeList[] = { // 素数表
壹, 2, 3, 5, 7, 壹壹, 壹 3, 壹 7, 壹 9, 23, 29, 3 壹, 37, 4 壹,
43, 47, 53, 59, 6 壹, 67, 7 壹, 73, 79, 83, 89, 97, 壹壹 3,
壹 93, 24 壹, 257, 337, 353, 40 壹, 433, 449, 577, 593, 64 壹,
673, 769, 88 壹, 929, 977, 壹 009, 壹壹 53, 壹 20 壹, 壹 2 壹 7, 壹 249,
壹 297,壹 36 壹, 壹 409, 壹 489, 壹 553, 壹 60 壹, 壹 697, 壹 777, 壹 873,
壹 889, 20 壹 7, 208 壹, 2 壹壹 3, 2 壹 29, 2 壹 6 壹, 2273, 24 壹 7, 2593,
2609, 2657, 2689, 2753, 280 壹, 2833, 2897, 304 壹, 3089,
3 壹 2 壹, 3 壹 37, 3 壹 69, 32 壹 7, 33 壹 3, 3329, 336 壹, 3457, 36 壹 7,
3697, 376 壹, 3793, 3889, 400 壹, 4049, 4 壹 29, 4 壹 77, 424 壹,
```

```
4273, 4289, 4337, 448壹, 45壹3, 456壹, 4657, 4673, 472壹,
480壹, 48壹7, 4993, 5009, 5壹53, 5233, 528壹, 5297, 5393,
544壹, 552壹, 5569, 5857, 5953, 6壹壹3, 6257, 6337, 6353,
6449, 648壹, 6529, 6577, 6673, 6689, 6737, 6833, 696壹,
6977, 7057, 7壹2壹, 7297, 7393, 7457, 7489, 7537, 7649,
768壹, 7793, 784壹, 7873, 7937, 80壹7, 808壹, 8壹6壹, 8209,
8273, 8353, 8369, 85壹3, 8609, 864壹, 8689, 8737, 8753,
8849, 8929, 904壹, 9壹37, 928壹, 9377, 9473, 952壹, 960壹,
9649, 9697, 9857
};
const int nListNum = sizeof(aPrimeList)/sizeof(unsigned);//计算素数表里元素的个数
for (unsigned i=2;i {
if(n/2+壹 {
return true;
}
if(0==n%aPrimeList[i])
{
return false;
}
}
/*由于素数表中元素个数是有限的，那么对于用素数表判断不到的数，就只有用笨蛋办法了*/
for (unsigned i=aPrimeList[nListNum-壹];i {
if (0==n%i)
{ // 除尽了，合数
return false;
}
}
return true;
}
```

还是太糟了，我们现在要做的对于大型素数的判断，那个素数表倒顶个P用！当然，我们可以利用动态的素数表来进行优化，这就是大学生的做法了。但是动态生成素数表的策略又复杂又没有效率，所以我们还是直接跳跃到专家的做法吧：

根据上面讲到的费马小定理，对于两个互质的素数N和P，必有：N^(P-壹)%P=壹

那么我们通过这个性质来判断素数吧，当然，你会担心当P很大的时候乘方会很麻烦。不用担心！我们上面不是有个快速的幂模算法么？好好的利用蒙格马利这位大数学家为我们带来的快乐吧！

算法思路是这样的：

对于N，从素数表中取出任意的素数对其进行费马测试，如果取了很多个素数，N仍未测试失败，那么则认为N是素数。当然，测试次数越多越准确，但一般来讲 50 次就足够了。另外，预先用“小学生”的算法构造一个包括 500 个素数的数组，先对Q进行整除测试，将会大大提高通过率，方法如下：

```
bool IsPrime3(unsigned n)
{
if ( n < 2 )
{ // 小于 2 的数即不是合数也不是素数
throw 0;
}
static unsigned aPrimeList[] = {
2, 3, 5, 7, 壹壹, 壹7, 壹9, 23, 29, 3壹, 4壹,
43, 47, 53, 59, 67, 7壹, 73, 79, 83, 89, 97
};
const int nListNum = sizeof(aPrimeList) / sizeof(unsigned);
for (int i=0;i { // 按照素数表中的数对当前素数进行判断
if (壹!=Montgomery(aPrimeList[i],n-壹,n)) // 蒙格马利算法
{
return false;
}
}
return true;
}
```

OK，这就专家的作法了。

等等，什么？好像有点怪，看一下这个数 2934 壹，它等于壹 3 * 37 * 6 壹，显然是一个合数，但是竟通过了测试!! 哦，抱歉，我忘了在素数表中加入壹 3，37，6 壹这三个数，我其实是故意的，我只是想说明并费马测试并不完全可靠。

现在我们发现了重要的一点，费马定理是素数的必要条件而非充分条件。这种不是素数，但又能通过费马

测试的数字还有不少，数学上把它们称为卡尔麦克数，现在数学家们已经找到所有壹 0 ^ 壹 6 以内的卡尔麦克数，最大的一个是 958592 壹壹 33 壹 93329。我们必须寻找更为有效的测试方法。数学家们通过对费马小定理的研究，并加以扩展，总结出了多种快速有效的素数测试方法，目前最快的算法是拉宾米勒测试算法，下面介绍拉宾米勒测试。

==========

拉宾米勒测试

拉宾米勒测试是一个不确定的算法，只能从概率意义上判定一个数可能是素数，但并不能确保。算法 流程如下：

壹.选择T个随机数A，并且有A 2.找到R和M，使得N=2*R*M+壹成立。

快速得到R和M的方式：N用二进制数B来表示，令C=B-壹。因为N为奇数（素数都是奇数），所以C的最低位为 0，从C的最低位的 0 开始向高位统计，一直到遇到第一个壹。这时 0 的个数即为R，M为B右移R位的值 。

3.如果A^M%N=壹，则通过A对于N的测试，然后进行下一个A的测试

4.如果A^M%N!=壹，那么令i由 0 迭代至R，进行下面的测试

5.如果A^((2^i)*M)%N=N-壹则通过A对于N的测试，否则进行下一个i的测试

6.如果i=r，且尚未通过测试，则此A对于N的测试失败，说明N为合数。

7.进行下一个A对N的测试，直到测试完指定个数的A

通过验证得知，当T为素数，并且A是平均分布的随机数，那么测试有效率为壹 / (4 ^ T)。如果T > 8 那么测试失误的机率就会小于壹 0^(-5)，这对于一般的应用是足够了。如果需要求的素数极大，或着要求更高的保障度，可以适当调高T的值。下面是代码：

```
bool RabbinMillerTest( unsigned n )
{
if (n<2)
{ // 小于 2 的数即不是合数也不是素数
throw 0;
}
const unsigned nPrimeListSize=sizeof(g_aPrimeList)/sizeof(unsigned);//求素数表元素个数
for(int i=0;i {// 按照素数表中的数对当前素数进行判断
if (n/2+壹<=g_aPrimeList[i])
{// 如果已经小于当前素数表的数，则一定是素数
return true;
}
if (0==n%g_aPrimeList[i])
{// 余数为 0 则说明一定不是素数
return false;
}
}
// 找到r和m，使得n = 2^r * m + 壹;
int r = 0, m = n - 壹; // ( n - 壹 ) 一定是合数
while ( 0 == ( m & 壹 ) )
{
m >>= 壹; // 右移一位
r++; // 统计右移的次数
}
const unsigned nTestCnt = 8; // 表示进行测试的次数
for ( unsigned i = 0; i < nTestCnt; ++i )
{ // 利用随机数进行测试，
int a = g_aPrimeList[ rand() % nPrimeListSize ];
if ( 壹 != Montgomery( a, m, n ) )
{
int j = 0;
int e = 壹;
for ( ; j < r; ++j )
{
if ( n - 壹 == Montgomery( a, m * e, n ) )
{
break;
}
e <<= 壹;
}
if (j == r)
{
```

```
return false;
}
}
}
return true;
}
```

反素数表

```
//poj 2886
//定义：对于任何正整数x,起约数的个数记做g(x).例如g(壹)=壹,g(6)=4.
//如果某个正整数x满足:对于任意i(0<i<x),都有g(i)<g(x),则称x为反素数.
int invPrime[4壹] = {   //反素数
        壹,2,4,6,壹2,24,36,48,60,壹20,
        壹80,240,360,720,840,壹260,壹680,2520,5040,7560,
        壹0080,壹5壹20,20壹60,25200,27720,45360,50400,55440,83壹60,壹壹0880,
        壹66320,22壹760,277200,332640,498960,554400,665280,720720,壹08壹080,
壹44壹440,
        2壹62壹60
};
int invCnt[4壹] = {     //对应的因子个数
        壹,2,3,4,6,8,9,壹0,壹2,壹6,
        壹8,20,24,30,32,36,40,48,60,64,
        72,80,84,90,96,壹00,壹08,壹20,壹28,壹44,
        壹60,壹68,壹80,壹92,200,2壹6,224,240,256,288,
        320
};
```

Josephus

```
//n个人（0，壹…n-壹），从第m人开始数，数k退壹，公式：
//f[壹]=0;       f[i]=(f[i-壹]+k)%i; (i>壹)
int jose(int n, int k, int m) {
    int f = 0;
    for(int i = 2; i <= n; i ++)
        f = (f + k) % i;
    return (f + m) % n;
}
/*
例：n=3,k=2,m=壹
开始：0,壹,2
壹.   壹,2：2退出
2.   0,壹：壹退出
剩下0
下面的用于加速，适用于n特别大的情况下。。。。
*/
long long Josephus(long long n, long long k, long long m = 0) {
    if(k == 壹) return (n+m-壹)%n;
    long long f = 0;
    for(long long i = 2; i <= n; i ++) {
        long long x = min((i-f-2)/(k-壹), n-i);
        f += k * x;
        i += x;
        f = (f+k) % i;
    }
    return (f + m) % n;
}
```

Josephus 逆_(k%壹+...+k%i)

```
//效率：sqrt(n)
//要求：壹<= i, k <= 壹0^9
long long get(long long i, long long k) {
    long long res = 0;
    for(long long pre; i*i>k; i=pre) {
        pre = k / (壹+k/i);
        res += (i-pre)*(2*k-k/i*(pre+i+壹))/2;
```

```
    }
    for(; i; i--)   res += k%i;
    return res;
}
//也可以简写为（效率略差）：
long long get(long long i, long long k) {
    long long res = 0;
    for(long long pre; i; i=pre) {
        pre = k / (壹+k/i);
        res += (i-pre)*(2*k-k/i*(pre+i+壹))/2;
    }
    return res;
}
```

皇后

```
#include <stdio.h>
long sum = 0, upperlim = 壹;
void test(long row, long ld, long rd) {
  if (row != upperlim) {
      long pos = upperlim & ~(row | ld | rd);
      for ( ; pos ; ) {
         long p = pos & -pos;
         pos -= p;
         test(row + p, (ld + p) << 壹, (rd + p) >> 壹);
        }
     }
  else sum++;
}
int main() {
   int n ;
   scanf("%d", &n);
   upperlim = (upperlim << n) - 壹;
   test(0, 0, 0);
   printf("%d\n", sum);
}
```

打表：

壹 壹
2 0
3 0
4 2
5 壹 0
6 4
7 40
8 92
9 352
壹 0 724
壹壹 2680
壹 2 壹 4200
壹 3 737 壹 2
壹 4 365596
壹 5 2279 壹 84
壹 6 壹 47725 壹 2
壹 7 958 壹 5 壹 04
壹 8 666090624
壹 9 4968057848
20 39029 壹 88884
2 壹 3 壹 46662227 壹 2
22 269 壹 00870 壹 644
23 24233937684440
24 2275 壹 4 壹 7 壹 973736
25 2207893435808352
26 223 壹 76996 壹 6364044

迷你小皇后：

```
int v, i, j, k, l, s, a[99];
```

```
main() {
    for(scanf("%d",&s);*a-s;       v=a[j*=v]-a[i],     k=i<s,
    j+= (v=j<s&&(!k&&!printf(2+"\n\n%c"-(!l<<!j),"``Q"[l^v?(l^j)&
壹:2])&&++l||a[i]<s&&v&&v-i+j&&v+i-j&&v+i-j))&&!( l%=s),
    v||(i==j?a[i+=k]=0:++a[i])>=s*k&&++a[--i]);
}
```

皇后构造法(构造皇后的一种解)POJ-3239:

一、当n mod 6 != 2 且 n mod 6 != 3时，有一个解为:
2,4,6,8,...,n,1,3,5,7,...,n-1　　(n为偶数)
2,4,6,8,...,n-1,1,3,5,7,...,n　　(n为奇数)
(上面序列第i个数为ai,表示在第i行ai列放一个皇后;...省略的序列中,相邻两数以2递增。下同)
二、当n mod 6 == 2 或 n mod 6 == 3时，
(当n为偶数,k=n/2；当n为奇数,k=(n-1)/2)
k,k+2,k+4,...,n,2,4,...,k-2,k+3,k+5,...,n-1,1,3,5,...,k+1　　(k为偶数,n为偶数)
k,k+2,k+4,...,n-1,2,4,...,k-2,k+3,k+5,...,n-2,1,3,5,...,k+1,n(k为偶数,n为奇数)
k,k+2,k+4,...,n-1,1,3,5,...,k-2,k+3,...,n,2,4,...,k+1　　(k为奇数,n为偶数)
k,k+2,k+4,...,n-2,1,3,5,...,k-2,k+3,...,n-1,2,4,...,k+1,n　　(k为奇数,n为奇数)

Codes:

```
int n;
int main()
{
   while (scanf("%d", &n), n)
   {
      if (n % 6 != 2 && n % 6 != 3)
      {
         printf("2");
         for (int i = 4; i <= n; i += 2)
            printf(" %d", i);
         for (int i = 1; i <= n; i += 2)
            printf(" %d", i);
         putchar('\n');
         continue;
      }
      int k = n / 2;
      printf("%d", k);
      if (!(k & 1) && !(n & 1))
      {
         for (int i = k + 2; i <= n; i += 2)
            printf(" %d", i);
         for (int i = 2; i <= k - 2; i += 2)
            printf(" %d", i);
         for (int i = k + 3; i <= n - 1; i += 2)
            printf(" %d", i);
         for (int i = 1; i <= k + 1; i += 2)
            printf(" %d", i);
      }
      else if (!(k & 1) && (n & 1))
      {
         for (int i = k + 2; i <= n - 1; i += 2)
            printf(" %d", i);
         for (int i = 2; i <= k - 2; i += 2)
            printf(" %d", i);
         for (int i = k + 3; i <= n - 2; i += 2)
            printf(" %d", i);
         for (int i = 1; i <= k + 1; i += 2)
            printf(" %d", i);
         printf(" %d", n);
      }
      else if ((k & 1) && !(n & 1))
      {
         for (int i = k + 2; i <= n - 1; i += 2)
            printf(" %d", i);
```

```
            for (int i = 1; i <= k - 2; i += 2)
                printf(" %d", i);
            for (int i = k + 3; i <= n; i += 2)
                printf(" %d", i);
            for (int i = 2; i <= k + 1; i += 2)
                printf(" %d", i);
        }
        else if ((k & 1) && (n & 1))
        {
            for (int i = k + 2; i <= n - 2; i += 2)
                printf(" %d", i);
            for (int i = 1; i <= k - 2; i += 2)
                printf(" %d", i);
            for (int i = k + 3; i <= n - 1; i += 2)
                printf(" %d", i);
            for (int i = 2; i <= k + 1; i += 2)
                printf(" %d", i);
            printf(" %d", n);
        }
        putchar('\n');
    }
    return 0;
}
```

DeBruijn

```
/**
    壹.DeBruijn是这样的序列，由D(n,k)表示
    2.序列一共有 2^n个元素（我将每个数称之为元素），每个元素为长度为n的 0 壹代码
    3.序列满足【当前元素的后n-壹位为下一个元素的前n-壹位】
        比如当n=3 时，按字典序输出DeBruijn序列为：
        000
        00 壹
        0 壹 0
        壹 0 壹
        0 壹壹
        壹壹壹
        壹壹 0
        壹 00
    4.由于性质 3，因此可以用一个长度为 2^n的字符串来表示在n下所有的DeBruijn序列，如n=3 时，该字符串为 000 壹 0 壹壹壹
    5.DeBruijn序列可由哈密顿回路构造，或者由欧拉回路构造，以下程序用的是非递归的欧拉回路（参考上交模板）
 *
 */

/**
【题目壹】.HDU-2894  输出字典序最小的DeBruijn序列
Input:
    壹 2 3 4
Output:
    2 0 壹
    4 00 壹壹
    8 000 壹 0 壹壹壹
    壹 6 0000 壹 00 壹壹 0 壹 0 壹壹壹壹
 */
#define maxn 壹 3
void outputLexDeBruijn(int n) {
    static bool vis[壹<<maxn];
    memset(vis, 0, sizeof(vis));
    int start, now;
    now = start = (壹<<(n-壹))-壹;
    do {
        if (vis[now]) {
```

```
            putchar('壹');
            now=(now*2+壹)&start;
        } else {
            putchar('0');
            vis[now]=壹;
            now=(now*2)&start ;
        }
    } while( now!=start );
    putchar('壹');
    putchar('\n');              //这里输出回车了!
}
int main() {
    int n;
   while(scanf("%d", &n) != EOF) {
      printf("%d ", 壹<<n);//题目要求先输出序列长度，在输出 0 壹序列
      outputLexDeBruijn(n);
   }
   return 0;
}

/**
【题目 2】.POJ-壹 392  输出DeBruijn的D(n,k)个数
Input:
    2 0
    2 壹
    2 2
    2 3
    0 0 //(end)
Output:
    0   //(00)
    壹  //(0 壹)
    3   //(壹壹)
    2   //(壹 0)
 */
#define maxn 壹 5
int deBruijn(int n, int k) {
    static bool vis[壹<<maxn], arr[(壹<<maxn)+maxn];
    memset(vis,0 , sizeof(vis ));
    memset(arr,0 , sizeof(arr));
    int now, start, i = 0, res = 0;
    now = start = (壹<<(n-壹))-壹;
    do {
        if(vis[now]) {
            arr[i ++] = 壹;
            now = (now*2+壹)&start;
        } else {
            vis[now]=壹;
            arr[i ++] = 0;
            now = (now*2)&start;
        }
    } while(now != start);
    arr[i ++] = 壹;
    int mask = (壹<<n)-壹;
    for(i = 0; i < n; i ++) res = res*2+arr[(k+i)&mask];
    return res;
}
int main() {
    int n, k;
    while(scanf("%d%d", &n, &k), n||k) {
        printf("%d\n", deBruijn(n, k));
    }
    return 0;
}
```

分数规划

对于 0-壹分数规划的 Dinkelbach 算法的分析

武钢三中 吴豪[译]

摘要：

0-壹分数规划问题是指求出解集{xi|xi=0 或壹}使目标(c 壹 x 壹+c2x2+...+cnxn) /(d 壹 x 壹+d2x2+…+dnxn)=**cx**/**dx** 达到最大。对于分数规划问题，Dinkelbach 提出了一个算法，它通过解决一个子问题 Q(L)来得到原文题的解。这里 Q 是一个线性的最小化目标函数 cx-Ldx，且满足 x 等于 0 或壹。在本文中，我们证明了 Dinkelbach 算法在最坏情况下可以在 O(log(nM))的时间内解决子问题，这里 M=max{max|ci|,max|di|,壹}。

壹．0-壹分数规划问题

要使两个线性函数的比值最大或最小的问题，我们称作分数规划问题或双曲线问题。分数规划问题在许多领域都可以找到[22]。它在经济学中的应用有些常见的例子，如寻找最优收入比率或者在效益约束下的最佳物资调配问题。另外，系统效率也常常用比率来衡量，如收益/时间、利润/风险和消费/时间。有大量的文章对这类问题做了分析[3，5，壹 2，20，24]。

有几类分数规划问题已被广泛地研究。如 0-壹分数规划问题[壹],它包含最优比率生成树问题[4]，最优比率环问题[8，6，壹 9]，分数背包问题[壹 5]，以及分数剪枝问题[壹 0]。在本文中，我们研究 0-壹分数规划问题，它的描述如下：

令 c=(c 壹,c2,…,cn)和 d=(d 壹,d2,…,dn)为 n 维整数向量，那么一个 0-壹分数规划问题用公式描述如下：

FP：最小化 (c 壹 x 壹+…cnxn)/(d 壹 x 壹…dnxn)=**cx**/**dx**

xi∈{0,壹}

这里 **x** 表示列向量(x 壹,x2,…,xn)T .0-壹值向量的子集Ω称作可行域，而 **x** 则是Ω的一个元素，我们称 **x** 为可行解。贯穿全文，我们假定对于任意可行解 **x**,**dx** 都是正数。这里我们记 C=max{max|ci|,壹},D=max{max|di|,壹}。那么，显然问题的最优解在区间[-nC,nC]内。

对于分数规划问题，有许多算法都能利用下面的线性目标函数解决问题。

Q(L)：最小化 **cx**-L**dx**

xi∈{0,壹}

记 z(L)为 Q(L)的最值。令 **x***为分数规划的最优解，并且令 L*= (**cx***)/(**dx***)(注：分数规划的最值)。那么下面就容易知道了：

z(L) > 0 当且仅当 L<L*

z(L) = 0 当且仅当 L=L*

z(L) < 0 当且仅当 L>L*

此外，Q(L*)的最优解也能使分数规划最优化[7,壹 6,壹 7]。因此，解决分数规划问题在本质上等同于寻找 L=L*使 z(L)=0。出于这个目的，关于 L 的函数 z(L)具有很多不错的性质：分段线性，凹函数，严格递减，z(-nC)<0，且 z(nC)>0。根据上面的性质，显然当我们确定参量 L，我们可以检验最值 L*是否大于小于或等于当前的 L。

有一些方法能够产生一系列收敛于 L*的参量。其中一种借助于二分搜索[壹 7,2 壹,壹 3]。在两个不同的可行解的目标值不相同的情况下，他们的差距将大于等于壹/(nD)^2。这暗示我们，当我们采用二分搜索时，最优值 L*可以通过解决子问题 Q(L)在最多 O(log(2nC/(壹/nD)^2))<=O(log(nCD))的时间内得到。

在壹 979 年，Megiddo[壹 8]提出了一个巧妙的方法来系统地产生参量序列。他证明了如果子问题 Q(L)能够通过 O(p(n))的比较和 O(q(n))的累加被解决，那么分数规划问题就能用 O(p(n)+q(n))的时间被解决。

另一种方法理论上类似于牛顿迭代法，他被 Isbell、Marlow[壹 4]和 Dinkelbach[7]提出（也被称作 Dinkelbach 算法）。这个算法在[壹 7,2 壹,壹壹]中被讨论（也可能是其他文献）。下一节将对它进行正式的论述。Schaible[2 壹]证明了对于非线性分数规划问题，二分搜索的方法的收敛速度仅仅是线性的，而 Dinkelbach 的收敛速度却是超线性的。另外，据说 Dinkelbach 算法在实际应用中强力而有效（参见[壹 3,23]的例子）。然而，Dinkelbach 算法对于 0-壹分数规划问题的最坏时间复杂度却没有被证明。在本文中，我们证明了,Dinkelbach 算法最多会在 O(log(nCD))的时间内解决子问题。注意它的时间复杂度与普通的二分搜索相同。我们的结论暗示了，如果对于子问题 Q(L)存在多项式算法，Dinkelbach 算法也能够在多项式时间内解决分数规划问题。另外，即使子问题 Q(L)是 NP-完全或 NP-难的，对于特殊的分数规划我们也能够在多项式时间内出解。

2．Dinkelbach 算法的论述

它本质上是观察直线

z=c**x′**-L**dx′**

于函数 z(L)在 L=L′处相切，这里 **x′**是子问题 Q(L′)的最优解。因此，**-dx′**是 z(L)在 L′处的斜率。而且很容易看出上面的直线与 L 轴相交与 L=**cx′/dx′**.

现在我们来描述 Dinkelbach 对于分数规划的算法。Dinkelbach 算法产生了收敛于 L*的参量序列，如图壹中细线所示的方式。

Dinkelbach 算法：

步骤壹：设 L=L 壹,使 L*<=L 壹<=nC

步骤 2：解决子问题 Q(L)并得到最优解 **x**

步骤 3：如果 z(L)=0，那么输出 **x** 并终止。否则，设 L=**cx**/**dx** 跳到步骤 2

为了初始化 L 壹,将用到 nC，因此充分挖掘拓展问题的结构将能做出更好的选择。

3. Dinkelbach 算法的分析

在这一节中，我们假定 Dinkelbach 算法终止于第 k 次。我们可以得到一个参量序列(L 壹,L2…Lk)和一个 0-壹值的向量(**x 壹,x2…xk**).z(L)的凸度暗示了下面的不等式：

dx 壹>dx2>…>dxk-壹>=dxk>0

cx 壹+nC**dx 壹>cx2**+nC**dx2**>...>**cxk-壹**+nC**dxk-壹**>=**cxk**+nC**xk**>=0

由于函数 z(L)是严格递减的，也很容易发现

z(L 壹)<z(L2)<...<z(Lk) = 0 并且 L 壹>L2>...>Lk

引理壹 如果 0>=z(Lr)>-壹/nD (2<=r<=k) 那么 z(Lr)=0

证明 由于 Lr=**cxr-壹**/**dxr-壹**

z(Lr)**=cxr-** Lr **dxr=cxr-**(**cxr-壹 dxr**)/(**dxr-壹**)=(**cxrdxr-壹 - cxr-壹 dxr**)/(**dxr-壹**)

假定 z(Lr)<0，那么(**cxrdxr-壹 - cxr-壹 dxr**)<=-壹。

因此不等式 0<**dxr-壹**<=nD 暗示了 z(Lr)<=-壹/nD。它是矛盾的!

上面的引理来源于权向量 **c** 和 **d** 的完整性。这个引理暗示了如果 z(Lr)>=-壹/nD 那么 z(Lr)=0，因此该算法会中止于第 r 次。

引理 2 如果 0<=**cxr**+nC**dxr**<壹,那么 z(**cxr**/**dxr**)=0

证明 由于 c**xr**+nC**dxr** 是整数，如果 0<=**cxr**+nC**dxr**<壹，那么 **cxr**+nC**dxr**=0 并且 **cxr**/**dxr**=-nC。由于最值 L*>=-nC, **xr** 是分数规划的最优解并且 L*=**cxr**/**dxr**=-nC。那么显然有 z(**cxr**/**dxr**)=z(L*)=0

上面的引理证明了如果 **cxr**+nC**dxr** < 壹，那么算法就在 r 或者 r+壹次终止。

现在给出主要引理：

引理 3 如果壹<=r<=k-壹那么|z(Lr+壹)|<=(壹/2)|z(Lr)|或 **cxr+壹**+nC**dxr+壹** <=(壹/2)(**cxr**+nC**dxr**)将满足。

证明 如果 Lr+nC<=0 那么 Lr=L*=-nC 并且它暗示着 z(Lr)b=(壹/2)z(Lr+壹)=0

现在假定 Lr+nC>0。就出现了两种情况：

情况(i) 首先我们来考虑 z(Lr+壹)(Lr+nC)<=z(Lr)/2*(Lr+壹+nC)。这个情形如图 2 所示。在这个图中，直线 z=**cxr**-L**dxr** 被记作 lr。这里我们将用到图 2 的符号。令 M 为线段 PR 的中点。那么点 M 的坐标为(Lr+壹 , z(Lr)(Lr+壹+nC)/2/ Lr+nC)。因此条件 z(Lr+壹)(Lr+nC)<=z(Lr)*(Lr+壹+nC)/2 暗示着点 Q=(Lr+壹,z(Lr+壹))在线段 MR 上。在这个条件下，我们证明不等式 **cxr+壹**+nC**dxr+壹**<=(壹/2)(**cxr**+nC, **dxr**)成立。这意味着直线 lr+壹与线段 MR 相交，lr+壹也与线段 M′R′相交，这里 M′是线段 P′R′的中点。现在我们证明这个不等式：

(Lr-Lr+壹)(**cxr+壹**+nC **dxr+壹**)

=(**cxr+壹**-Lr+壹 **dxr+壹**) (Lr+nC)- (**cxr+壹**-Lr **dxr+壹**) (Lr+壹+nC)

=z(Lr+壹) (Lr+nC) - (**cxr+壹**-Lr **dxr+壹**) (Lr+壹+nC)

<= z(Lr) (Lr+壹+nC)/2- (**cxr**-Lr **dxr**) (Lr+壹+nC)

=-(壹/2) (**cxr**-Lr **dxr**) (Lr+壹+nC)=-(壹/2)(**cxr**/ **dxr** - Lr) **dxr**(**cxr**/ **dxr**+nC)

=-(壹/2)(Lr+壹-Lr) (**cxr**+nC **dxr**)=(壹/2) (Lr-Lr+壹) (**cxr**+nC **dxr**)

由于 Lr>Lr+壹,那么不等式 **cxr+壹**+nC **dxr+壹**<= (壹/2) (**cxr**+nC **dxr**)已经被证明。

情况(ii) 接着，考虑 z(Lr+壹)(Lr+nC)>z(Lr)/2*(Lr+壹+nC)

|z(Lr+壹)|=- z(Lr+壹)< - z(Lr)(nC+ Lr+壹)/2/(nC+ Lr)

=|z(Lr)|(壹-(Lr- Lr+壹)/ (nC+Lr))/2<=壹/2 * |z(Lr)|

注意无论|z(L)|还是 **cx**+nC**dx** 在过程中都是不增长的。

在第一次，|z(L)|的值小于等于 2n^2*CD。通过引理 1，显然如果存在 O(log(2n^2CD/(壹/nD)))<=O(log(nCD))次，他们每个都能至少将 z(L)减少 50%那么，然后就能得到最优解。同样，引理 2 暗示了将 **cx**+nC**dx** 减少 50%的次数 O (log(2n^2CD))<=O(log(nCD))是最坏情况。引理 3 证明了每次将|z(L)|或 **cx**+nC**dx** 减少 50%。因此，重复总次数的界限是 O(log(nCD))。

定理 4 Dinkelbanch 算法最坏情况的运行次数是 O(log(nCD))<=O(log(nM))，这里 M=max{C,D}。

上面的定理证明了 Dinkelbanch 算法最坏运行次数是 O(log(nCD))。它暗示了，如果对于 Q(L)存在强多项式算法，Dinkelbach 算法就能在多项式时间内解决分数规划问题。然而，当我们用多项式算法解决了子问题后，我们需要估计目标函数 Q(L)的系数的输入规模。在下一节，我们将通过分析最优比率生成树和分数调配问题来讨论这一点。

4．讨论

Chandrasekaran[4]提出了最优比率生成树的算法，它是基于二分搜索的。Dinkelbach 算法可以在 O(T(v,e)log(vCD))的时间解决该问题，这里 v 是点的个数，e 是边的个数，并且用 T(v,e)表示计算普通最小生成树的强多项式算法。很容易将 Chandrasekaran 的算法延伸到一般带有分数目标函数的矩阵胚规划问题。在这种情况下，在这种情形下，函数 z(L)的断点数最大为 n(n-壹)/2（参见[4]）因此，当可行域 Ω 是矩阵胚基础特征向量的集合。Dinkelbach 算法就会在 O(min{n^2,log(nCD)})后终止。

对于调配问题，已经研制了许多算法。大概最有名的算法就是 Hungarian 方法，并且它在最坏情况下的复杂度是 O(v(v log v+e)) [9]。使用 Hungarian 方法，Dinkelbach 算法可以用 O(v(v log v+e)log(vCD))的时间解决分数调配问题。在[壹 9]中，Orlin 和 Ahuja 提出了一个 O(sqrt(v)e*log(vW))的算法来解决调配问题而且据说他们的算法因为强多项式算法而具有竞争性（参见[2]也可）。在他们的算法中，它假定边权为整数，并且 W 代表边权的最大绝对值。为了将这个算法与 Dinkelbach 算法相结合，我们需要将在运行第 r 次解决的的子问题 Q(L)用下面式子代替

(dxr-壹) cx-(cxr-壹) dx

这里**xr-壹**表示代表运行第 i 次得到的最优解。因此，在每次运行中，Dinkelbach 算法解决边权绝对值小于等于 2v^2CD 的调配问题。它暗示了 Dinkelbach 算法对于调配问题的最坏情况下的时间复杂度为

O(sqrt(v)e(log(2v^3CD))(log(eCD)))<=O(sqrt(v)e(log(vCD))^2)

我们说，如果一个具有线性目标函数的 0-壹整数规划问题存在多项式算法，我们可以利用上述目标函数在多项式时间内解决它的 0-壹分数规划问题。

致谢

作者在此感谢东京科学大学的 Prof.Hirabayashi 一直以来的鼓励与讨论。

裸分规_poj3 壹壹壹

```
/**
    poj-3 壹壹壹  K Best
    最裸的分规：
        给数组v[0...n)和w[0...n), 其中:
        0 ≤ vi ≤ 壹0^6, 壹 ≤ wi ≤ 壹0^6, both the sum of all vi and the sum of
all wi do not exceed 壹0^7
    要求从这n个w和v中选出k个，使得sum{vx}/sum{wx}最大。
 */
#define maxn 壹000壹0
typedef pair<double,int> T;

int n, k;
int v[maxn], w[maxn];
T ts[maxn];

double search(double g) {
    for(int i = 0; i < n; i ++) {
        ts[i] = T(v[i]-g*w[i], i);
    }
    nth_element(ts, ts+n-k, ts+n);
    int up=0, down=0;
    for(int i = n-k; i < n; i ++) {
        up += v[ts[i].second];
        down += w[ts[i].second];
    }
    return (double)up/down;
}

int main() {
    while(scanf("%d%d", &n, &k) != EOF) {
        for(int i = 0; i < n; i ++) {
            scanf("%d%d", v+i, w+i);
        }
        double g = 0;
```

```
        while(壹) {
            double gg = search(g);
            if(g==gg)   break;  //或者用sig函数
            g = gg;
        }
        for(int i = n-k; i < n; i ++) {
            printf("%d ", 壹+ts[i].second);
        }
        printf("\n");
    }
    return 0;
}
```

最大密度子图

```
/**
 * poj-3 壹 55   Hard Life
 * 裸的最大密度子图
 * 给定一个无向图，找出他的诱导子图，使得|E|/|V|最大
 * 输入：无向图
 * 输出：诱导子图的点集
 */

#define maxn 壹壹 0
const int inf = 0x3f3f3f3f;

int sig(double d) {
    if(fabs(d) < 壹E-6) return 0;
    return d < 0 ? -壹 : 壹;
}

struct SAP {
    double cap[maxn][maxn], flow[maxn][maxn];///容量,流量
    int n;  ///顶点数
    int h[maxn], vh[maxn], source, sink;
    ///某个点到sink点的最短距离，h[i]的计数，source点，sink点
    void init(int n) {
        this->n = n;
        memset(cap, 0, sizeof(cap));
    }
    void addCap(int i, int j, double val) {
        cap[i][j] += val;
    }
    /**
        参数：节点编号，和该节点能用的最大流量
        返回：此次找到的最大流量
    */
    double sap(const int idx, const double maxCap) {
        if(idx == sink)
            return maxCap;///最后一个结点。。。
        double l = maxCap, d;
        int minH = n;
///此次的残余流量，某次使用的流量，邻居的最小流量
        for(int i = 0; i < n; i ++) {
            if(sig(cap[idx][i]-flow[idx][i]) > 0) {
                if(h[idx]==h[i]+壹) {
                    d = sap(i, min(l, cap[idx][i]-flow[idx][i]));
///下次找到的流量
                    flow[idx][i] += d;          ///更新边的残余流量
                    flow[i][idx] -= d;
                    l -= d;                     ///更新本次参与流量
                    if(h[source]==n||sig(l)==0) return maxCap-l;///GAP
                }
                minH=min(minH, h[i]+壹);            ///更新h[idx]
            }
```

```
        }
        if(sig(l-maxCap) == 0) {                        ///not found!
            vh[h[idx]] --;                      ///GAP
            vh[minH] ++;
            if(vh[h[idx]] == 0)
                h[source] = n;
            h[idx] = minH;
        }
        return maxCap - l;
    }
    double solve(int source, int sink) {
        if(source == sink)  return inf;
        this->sink = sink;
        this->source = source;
        memset(flow, 0, sizeof(flow));
        memset(h, 0, sizeof(h));
        double ans = 0;
        while(h[source] != n)
            ans += sap(source, inf);
        return ans;
    }
    bool vis[maxn];
    int arr[maxn];
    int len;
    void dfs(int idx) {
        if(vis[idx])    return;
        vis[idx] = true;
        arr[len ++] = idx;
        for(int i = 0; i < n; i ++) {
            if(sig(cap[idx][i]-flow[idx][i]) > 0) {
                dfs(i);
            }
        }
    }
};

SAP sap;
int n, m;

int dgr[maxn];
int source, sink;

double getH(double g) {
    for(int i = 0; i < n; i ++) {
        sap.cap[i][sink] = m + 2*g-dgr[i];
    }
    return (m*n-sap.solve(source, sink)) / 2;
}

int main() {
    while(cin >> n >> m) {
        source = n, sink = n+壹;
        sap.init(n+2);
        memset(dgr, 0, sizeof(dgr));
        int a, b;
        for(int i = 0; i < m; i ++) {
            cin >> a >> b;
            a --;   b --;
            sap.addCap(a, b, 壹);
            sap.addCap(b, a, 壹);
            dgr[a] ++;
            dgr[b] ++;
        }
        for(int i = 0; i < n; i ++) {
            sap.addCap(source, i, m);
        }
        //公有部分结束!
```

```
        double l = 0.0, r = m, jingdu = 壹.0/n/n;
        while(sig(r-l-jingdu) > 0) {
            double g = (l+r)/2.0;
            double h = getH(g);
            if(sig(h) > 0) {
                l = g;
            } else {
                r = g;
            }
        }
        getH(l);
        memset(sap.vis, 0, sizeof(sap.vis));
        sap.len = 0;
        sap.dfs(source);
        if(sig(l) == 0) {
            cout << 壹 << endl << 壹 << endl;
        } else {
            sort(sap.arr, sap.arr+sap.len);
            cout << sap.len-壹 << endl;
            for(int i = 0; i < sap.len-壹; i ++) {
                cout << sap.arr[i]+壹 << endl;
            }
        }
    }
    return 0;
}
```

最优比率 MST_表

```
/**
 * usaco-200 壹  earthquake
 * 最优比率生成树
 * 题目本来是求最大比率生成树，然后取反后就成了最小的生成树（标程是最小的）
 */

const double inf = 壹E30;
#define maxn 4 壹 0
#define maxm 壹 00 壹 0
const double eps = 壹E-6;

int n;

#define th(x)   this->x = x
typedef pair<double,int> T;
struct Nod {
    int b, nxt;
    double C, D;
    void init(int b, int nxt, double C, double D) {
        th(b);  th(nxt);    th(C);  th(D);
    }
};

struct Prim {
    Nod buf[maxm*2];    int len;    //资源
    int n;  int E[maxn];            //图

    void init(int n) {
        this->n = n;
        len = 0;
        memset(E, 255, sizeof(E));
    }
    void addEdge(int a, int b, double c, double d) {
        buf[len].init(b, E[a], c, d);  E[a] = len ++;
        buf[len].init(a, E[b], c, d);  E[b] = len ++;
    }
    //MST不唯一，则返回-壹，否则返回MST的权值。。如果不用唯一判断，可以去掉cnt有关的一切!
```

```
    double search(double g) {
        static double d[maxn];
        static bool vis[maxn];                                  //访问过
        static int from[maxn];
        static priority_queue<T, vector<T>, greater<T> > q;    //Heap!

        memset(vis, 0, sizeof(vis));
        fill(d, d+n, inf);
        memset(from, 255, sizeof(from));
        while(!q.empty())   q.pop();

        int u, v;
        double du, dv;

        d[0] = 0;
        q.push(T(0, 0));

        double up = 0, down = 0;

        while(!q.empty()) {
            u = q.top().second;
            du = q.top().first;
            q.pop();
            if(du != d[u]) {
                continue;
            }
            vis[u] = true;
            if(from[u] != -壹) {
                up += buf[ from[u] ].C;
                down += buf[ from[u] ].D;
            }
            for(int i = E[u]; i != -壹; i = buf[i].nxt) {
                v = buf[i].b;
                dv = buf[i].C - g * buf[i].D;

                if(vis[v])  continue;
                if(dv < d[v]) {
                    from[v] = i;
                    d[v] = dv;
                    q.push(T(dv, v));
                }
            }
        }
        return up / down;
    }
    double compute() {
        double g = 0;
        search(0);
        while(壹) {
            double gg = search(g);
            if(fabs(g-gg)<eps)  break;
            g = gg;
        }
        return g;
    }
} prim;

int main() {
    int n, m;
    double F;
    while(scanf("%d%d%lf", &n, &m, &F) != EOF) {
        prim.init(n);
        int a, b;
        double c, d;
        while(m --) {
            scanf("%d%d%lf%lf", &a, &b, &c, &d);
            prim.addEdge(a-壹, b-壹, -(F/(n-壹) - c), d);
```

```
        }
        printf("%.4f\n", max(-prim.compute(),0.00));
    }
    return 0;
}
```

最优比率 MST_阵

```
const double inf = 壹 E30;
#define maxn 壹 0 壹 0
const double eps = 壹E-5;

/**
 * 最优比率生成树：
 * poj-2728     Desert King
 * n个点的完全图，求生成树，使得sum{C}/sum{D}最小。(assume D!=0)
 * 如果要求最大比率生成树，则C取负即可（标程是最小的）
 */

int n;
double C[maxn][maxn], D[maxn][maxn];

double search(double g) {
    static bool vis[maxn];
    static double d[maxn];
    static int p[maxn];

    memset(vis, 0, sizeof(vis));
    fill(d, d+n, inf);
    memset(p, 255, sizeof(p));
    d[0] = 0;

    double up = 0, down = 0;
    for(int i = 0; i < n; i ++) {
        int u = -壹;
        for(int i = 0; i < n; i ++) {
            if(!vis[i] && (u==-壹||d[i]<d[u])) {
                u = i;
            }
        }
//      if(u == -壹 || d[u]==inf)   return -壹; //假定不存在没有mst情况
        vis[u] = true;

        if(p[u] != -壹) {
            up += C[p[u]][u];
            down += D[p[u]][u];
        }
        for(int v = 0; v < n; v ++) {
            double e = C[u][v] - D[u][v] * g;
            if(!vis[v] && e<d[v]) {
                d[v] = e;
                p[v] = u;
            }
        }
    }
    return up / down;
}

double compute() {
    double g = 0;
    while(壹) {
        double gg = search(g);
        if(fabs(g-gg)<eps)  break;
        g = gg;
    }
    return g;
}
```

```
double x[maxn],y[maxn],z[maxn];

int main() {
    while(scanf("%d", &n), n) {
        for(int i = 0; i < n; i ++)scanf("%lf%lf%lf", x+i, y+i, z+i);
        for(int i = 0; i < n; i ++) {
            for(int j = 0; j < n; j ++) {
                C[i][j] = fabs(z[i]-z[j]);
                D[i][j] =
sqrt( (x[i]-x[j])*(x[i]-x[j])+(y[i]-y[j])*(y[i]-y[j]) );
            }
        }
        printf("%.3f\n", compute());
    }
    return 0;
}
```

最优比率环

```
/*
 * poj-362 壹        Sightseeing Cows
 * 最优比率环
 * 有向图中，每个点有一个权值C(壹<=c<=壹 000)，每条边有权值D(壹<=D<=壹 000)
 * 要求找出一个环（没有重点），使得sum{C}/sum{D}最大
 * 环包含的点数要>=2
 */

#define maxn 壹 0 壹 0
#define maxm 50 壹 0
const double eps = 壹E-4;

#define th(x)   this->x = x

struct Nod {
    int b, next;
    double D;
    void init(int b, double D, int next) {
        th(b);  th(D);  th(next);
    }
};
struct SPFA {
    double C[maxn]; //每个节点的权值

    Nod buf[maxm];  int len;
    int E[maxn], n;

    double d[maxn];

    void init(int n) {
        th(n);
        memset(E, 255, sizeof(E));
        len = 0;
    }
    void addEdge(int a, int b, double val) {
        buf[len].init(b, val, E[a]);
        E[a] = len ++;
    }
    bool solve(double g) {  //通过spfa判断有没有负环。返回true:没有负环
        static queue<int> q;
        static int cnt[maxn];
        static bool vis[maxn];

        while(!q.empty())   q.pop();
        memset(cnt, 0, sizeof(cnt));
        memset(d, 0, sizeof(d));
        memset(vis, 壹, sizeof(vis));
```

```
        for(int i = 0; i < n; i ++)q.push(i);

        int u, v;

        while(!q.empty()) {
            u = q.front();  q.pop();    vis[u] = false;
            for(int i = E[u]; i != -壹; i = buf[i].next) {
                v = buf[i].b;
                double e = g * buf[i].D - C[u];
                if(d[u] + e < d[v]) {
                    d[v] = d[u] + e;
                    if(!vis[v]) {
                        q.push(v);  vis[v] = true;
                    }
                    if(++ cnt[v] > n)   return false;
                }
            }
        }
        return true;
    }
    double compute() {  //计算最优比率环
        double l = 0, r = 壹E7, m;  //l代表有负环，r代表没有负环的比率
        while(r-l>eps) {
            m = (l+r) / 2;
            if(false==solve(m)) {   //有负环
                l = m;
            } else {
                r = m;
            }
        }
        return l;
    }
} spfa;

int main() {
    int n, m;
    while(scanf("%d%d", &n, &m) != EOF) {
        spfa.init(n);
        for(int i = 0; i < n; i ++)scanf("%lf", &spfa.C[i]);
        int a, b;
        double v;
        while(m --) {
            scanf("%d%d%lf", &a, &b, &v);
            spfa.addEdge(a-壹, b-壹, v);
        }
        printf("%.2f\n", spfa.compute());
    }
    return 0;
}
```

最优比率环 2

```
/*
 * sgu-236      Greedy Path
 * 最优比率环
 * 每条边有cost和time，要求一个环，使得sum{cost}/sum{time}最大
 */

#define maxn 5壹
#define maxm 25壹0
const double eps = 壹E-4;

#define th(x)   this->x = x

struct Nod {
    int b, next;
    int cost, time;
```

```
    void init(int b, int cost, int time, int next) {
        th(b);  th(cost);   th(time);   th(next);
    }
};

struct SPFA {
    Nod buf[maxm];  int len;
    int E[maxn], n;

    double d[maxn];

    void init(int n) {
        th(n);
        memset(E, 255, sizeof(E));
        len = 0;
    }
    void addEdge(int a, int b, int cost, int time) {
        buf[len].init(b, cost, time, E[a]);
        E[a] = len ++;
    }

    int p[maxn];

    int solve(double g) {   //通过spfa判断有没有负环。返回true:没有负环
        static queue<int> q;
        static int cnt[maxn];
        static bool vis[maxn];

        while(!q.empty())   q.pop();
        memset(cnt, 0, sizeof(cnt));
        memset(d, 0, sizeof(d));
        memset(vis, 壹, sizeof(vis));
        memset(p, 255, sizeof(p));

        for(int i = 0; i < n; i ++)q.push(i);

        int u, v;

        while(!q.empty()) {
            u = q.front();  q.pop();    vis[u] = false;
            for(int i = E[u]; i != -壹; i = buf[i].next) {
                v = buf[i].b;
                double e = g * buf[i].time - (buf[i].cost);

                if(d[u] + e < d[v]) {
                    d[v] = d[u] + e;
                    p[v] = u;
                    if(!vis[v]) {
                        q.push(v);  vis[v] = true;
                    }
                    if(++ cnt[v] > n) {
                        for(int u = p[v]; u != v; )u = p[p[u]], v = p[v];
                        return v;
                    }
                }
            }
        }
        return -壹;
    }
    void compute() {    //计算最优比率环
        double l = 0, r = 壹E7, m;  //l代表有负环，r代表没有负环的比率
        if(solve(0) == -壹) {
            printf("0\n");
            return;
        }
        while(r-l>eps) {
            m = (l+r) / 2;
```

```
            if(-壹 != solve(m)) {   //有负环
                l = m;
            } else {
                r = m;
            }
        }
        int start = solve(l);
        int u = start;

        static int arr[maxn];
        int len = 0;
        do {
            arr[len ++] = u;
        } while((u=p[u]) != start);

        printf("%d\n", len);
        for(int i = len-壹; i >= 0; i --) {
            printf("%d", 壹+arr[i]);
            if(i != 0)  printf(" ");
        }
        printf("\n");
    }
} spfa;

int main() {
    int n, m;
    int a, b, c, d;

    while(scanf("%d%d", &n, &m) != EOF) {
        spfa.init(n);
        while(m --) {
            scanf("%d%d%d%d", &a, &b, &c, &d);
            a --;   b --;
            spfa.addEdge(a, b, c, d);
        }
        spfa.compute();
    }
    return 0;
}
```

经典问题

KMP 算法

```
#define KMP_GO(X) while(k>0 && P[k] != X[i]) k = pi[k-壹]; if(P[k] == X[i]) k ++

int kmp_match(char*T, char*P) {
    static int n, m, pi[壹00壹0], i, k, c;
    n = strlen(T);  m = strlen(P);
    pi[0] = k = 0;
    for(i = 壹; i < m; i ++) {
        KMP_GO(P);
        pi[i] = k;
    }
    k = c = 0;
    for(i = 0; i < n; i ++) {
        KMP_GO(T);
        if(k == m) {
            c++;
            k = pi[k-壹];
        }
    }
    return c;
}
```

n 个棍子分成 k 组 poj 壹 0 壹壹

```
int nxt[壹00];//链表
int num[壹00];//这根棍的长度
int n;      //棍子个数
int len;    //每组中的长度大小
bool dfs(int nowLen=len, int myhead = 0) {
    if(nowLen==len) {  //装满了
        int first = nxt[0];
        if(first == n+壹)   return true;   //ok，装完了!
        if(num[first] <= len) {
            nxt[0] = nxt[first];
            if(dfs(num[first], 0)) return true;
            nxt[0] = first;
        }
    } else {
        int prev = myhead;
        int last = -壹;
        for(int i = nxt[myhead]; i != n+壹; i = nxt[i]) {
            if(num[i]+nowLen <= len && num[i]!=last) {
                nxt[prev] = nxt[i];
                if(dfs(nowLen+num[i], prev))   return true;
                nxt[prev] = i;
            }
            prev = i;
            last = num[i];
        }
    }
    return false;
}
bool cmp(int a, int b) {
    return a > b;
}
int main() {
    while(scanf("%d", &n), n) {
        int sum = 0;
        for(int i = 壹; i <= n; i ++)   scanf("%d", num+i), nxt[i]=i+壹, sum +=
num[i];
        nxt[0] = 壹;
        sort(num+壹, num+壹+n, cmp);  //先逆序排序

        for(int i = 壹; i <= sum; i ++) {
            if(sum % i == 0) {
                len = i;
                if(dfs()) {
                    printf("%d\n", i);
                    break;
                }
            }
        }
    }
    return 0;
}
```

RMQ 之 ST 算法（窗格版）

窗格

用途：区间长度固定的 rmq，询问 rmq[壹...n]上的最值

另外，窗格也可以用双向队列来做。

```
int rmq[maxn];
#define CMP >       ///大于取大值，小于取小值
struct ST {
    int best[maxn], k, l;///下标,最大2的幂,窗格大小
    void init(int n, int l) {///rmq 的长度,区间大小
        int a, b;
        this->l = l;
        for(int i = 壹; i <= n;  i ++)best[i] = i;
        for(k = 壹; 2*k < l; k<<=壹) {
```

```
        //每次迭代，已经构造长度为k的窗格，要构造长度为2k的窗格
            for(int i = 壹; i <= n+壹-(k<<壹); i++) {
                a = best[i];
                b = best[i+k];
                best[i] = rmq[a] CMP rmq[b]? a : b;
            }
        }
    }
    ///询问rmq[x]...rmq[x+l-壹]上的最值
    int query(int x) {
        int a = best[x], b = best[x+l-k];
        return rmq[a] CMP rmq[b] ? a : b;
    }
};
```

最长上升子序列

```
int get(int *arr, int n) {
    static int D[maxn];
    D[0] = -inf;    ///relatively minimum number
    int len = 0;
    for(int i = 0; i < n; i ++) {
        if(arr[i] > D[len]) {                   //...壹
            D[++ len] = arr[i];
        } else {
            *lower_bound(D, D+len+壹, arr[i]) = arr[i];//...2
        }
    }
    return len;
}
```

最长不降子序列

```
int get(int *arr, int n) {
    static int D[maxn];
    D[0] = -inf;    ///relatively minimum number
    int len = 0;
    for(int i = 0; i < n; i ++) {
        if(arr[i] >= D[len]) {                  //...壹
            D[++ len] = arr[i];
        } else {
            *upper_bound(D, D+len+壹, arr[i]) = arr[i];//...2
        }
    }
    return len;
}
//注意只有壹和 2 处不同。另外，如果要求最长降序、不升子序列的时候，壹、2 都要改变。
```

逆序对(注意会将 arr 排序!!!)

```
#define maxn 400 壹 0
long long inv(int *arr, int n) {
    static int tmp[maxn];
    if(n <= 壹) return 0;
    int l = n/2;
    long long res = inv(arr, l) + inv(arr+l, n-l);

    int i = 0, j = l;   //pos in the left and pos in the right;
    for(i = 0; i < l; i ++) {
        for(; j < n && arr[j] < arr[i]; j ++) {
            tmp[i+j-l] = arr[j];
        }
        res += j-l; //there are j-l elements in R smaller than arr[i]
        tmp[i+j-l] = arr[i];
    }
    copy(tmp, tmp+j, arr);
    return res;
}
//stl的优雅写法（低速）：
```

```
long long inv(int *arr, int n) {
    //static int tmp[maxn];
    if(n <= 壹) return 0;
    int l = n/2;
    long long res = inv(arr, l) + inv(arr+l, n-l);

    int i = 0, j = l;   //pos in the left and pos in the right;
    for(i = 0; i < l; i ++) {
        for(; j < n && arr[j] < arr[i]; j ++);
        res += j-l; //there are j-l elements in R smaller than arr[i]
    }
    inplace_merge(arr, arr+l, arr+n);
    return res;
}
```

sigma[n] = sum(d, d|n)

生成这样的sigma序列，如sigma[壹2] = 壹 + 2 + 3 + 4 + 6 + 壹2 = 28。效率同开素数效率
序列：http://www.research.att.com/~njas/sequences/A000203
公式：If the canonical factorization of n into prime powers is the product of p^e(p) then sigma_k(n) = Product_p ((p^((e(p)+壹)*k))-壹)/(p^k-壹).
程序：

```
int sigma[壹0000壹0];
int getPrime(int*prime,bool*notPrime,int n,int m) {
    int num[壹00];
    int p,i,j,cnt,idx;
    p=notPrime[0]=notPrime[壹]=壹;
    sigma[壹] = 壹;
    for(i=2;i<=n;i++) {
        if(!notPrime[i]) {
            sigma[i] = i+壹;
            for(j = 0; j < 30; j ++)    num[j] = 0;
            num[0] = 壹;
            for(prime[p++]=i,j=2*i; j<=m; j+=i) {
                notPrime[j] = 壹;
                num[0] ++;
                cnt = i;
                for(idx = 0; num[idx] == i; idx ++) {
                    num[idx] -= i;
                    num[idx+壹] ++;
                    cnt *= i;
                }
                sigma[j] = (cnt*i-壹) / (i-壹) * sigma[j/cnt];
            }
        }
    }
    for(int i = 壹; i <= m; i ++) {
        if(!notPrime[i])    sigma[i] = i+壹;
    }
    return p;
}
```

最大平均值子序列

```
//教程：2004周源论文
//网站:http://hi.baidu.com/pojoflcc/blog/item/5f74b372459c28壹7870壹b08b.html
//判断c点是否在ab连线的上面。。
#define above(a, b, c)     (long long)f[a]*(c-b)+(long long)f[b]*(a-c)+(long long)f[c]*(b-a)>=0
/**
 * 得到最大密度子段，子段最小长度为m(0<m<=n)
 * 原始序列为f[壹...n]，f将被破坏，f将变为sum序列
 * [l, r]为得到的子段的区间
 */
double getDensity(int *f, int n, int m, int & l, int & r) {
```

```
    static int Q[maxn], H, T;   //队列，头，尾
    H = T = 0;
    f[0] = 0;
    for(int i = 壹; i <= n; i ++)   f[i] += f[i-壹];
    double ans = -壹.0;
    for(int i = m; i <= n; i ++) {
        while(T-H>=2 && above(Q[T-2], (i-m), Q[T-壹])) T --;   //注意c-m有括号
        Q[T ++] = i-m;
        while(T-H>=2 && above(Q[H], Q[H+壹], i))           H ++;
        double tmp = (double)(f[i]-f[Q[H]]) / (i-Q[H]);
        if(tmp > ans) {
            ans = tmp;
            r = i;
            l = Q[H]+壹;
        }
    }
    return ans;
}
```

steiner_tree

```
/*
hdu-33 壹壹  Dig The Wells
题意：给定n(n<=5)个必选点，和m(m<=壹 000)个辅助点，每个点有权值(挖井的费用)，以及带权无向边若干。n个点中，要么自己挖井，要么通过道路连接到其他可以挖井的点
分析：这题比较巧妙，首先因为又有边权又有点权不好处理，所以我们先建一个虚拟点 0，把所有点与 0 点连边，边权为该点的点权，那么就转化成了有n+m+壹个点，求出这些点中使 0-n这n+壹个点连通的最小生成树。这个定义其实就是所谓的斯坦纳树。Steiner Tree是NP的，但是当必选点较少时，可以通过状态压缩DP来解，具体的DP状态是dp[x][壹<<m] 代表以x为根的树并且包含state里的点的最小权值。那么求一棵树就可以枚举连接点，以及两个子树的覆盖状态，通过一个很漂亮的转移方程来进行转移：dp[i][j]=min(dp[i][j],dp[i][k]+dp[i][j^k])，其中k是j的一个子集，而这个子集又可以通过位运算枚举子集来实现：for (int sub = state;sub != 0;sub = (sub-壹) & state)。具体的可以参见代码。
代码：*/

const int inf = 0x3f3f3f3f;
const int maxn = 壹壹 00;    //最多的点数（包括必选点）
const int maxm = 壹 3000;    //最多的边数
const int maxv = 6;     //最多的必选点数
#define th(x)   this->x = x

//求所有点对的最短路径
//用SPFA或许会更快些
typedef pair<int,int> T;
struct Nod {
    int b, val, next;
    void init(int b, int val, int next) {
        th(b);  th(val);    th(next);
    }
};
struct Dijkstra {
    Nod buf[maxm];  int len;    //资源
    int E[maxn], n;             //图
    int d[maxn][maxn];              //最短距离
    void init(int n) {
        th(n);
        memset(E, 255, sizeof(E));
        len = 0;
    }
    void addEdge(int a, int b, int val) {
        buf[len].init(b, val, E[a]);
        E[a] = len ++;
    }
    void solve(int s) {
        static priority_queue<T, vector<T>, greater<T> > q;
```

```
        while(!q.empty())  q.pop();
        memset(d[s], 63, sizeof(d[s]));
        d[s][s] = 0;
        q.push(T(0, s));
        int u, du, v, dv;
        while(!q.empty()) {
            u = q.top().second;
            du = q.top().first;
            q.pop();
            if(du != d[s][u])  continue;
            for(int i = E[u]; i != -壹; i = buf[i].next) {
                v = buf[i].b;
                dv = du + buf[i].val;
                if(dv < d[s][v]) {
                    d[s][v] = dv;
                    q.push(T(dv, v));
                }
            }
        }
    }
    void solve() {
        for(int i = 0; i < n; i ++)
            solve(i);
    }
} dij;

//一共n个点，这n个点中有m个点必须要选，这m个点是v
//d[i][j]表示i到j的最短路径
int steiner(int n, int m, int p[], int d[maxn][maxn]) {
    static int dp[壹<<maxv][maxn];
    memset(dp,63,sizeof(dp));

    for (int i = 0;i < m;i++)
        for (int j = 0;j < n;j++)
            dp[壹<<i][j] = d[p[i]][j];
    for (int i = 壹;i < (壹<<m);i++) {
        if(((i-壹)&i)==0)  continue;
        for (int j = 0;j < n;j++) {
            dp[i][j] = inf;
            int k = i;
            do {            //枚举子集【一半子集】
                k=(k-壹)&i;
                dp[i][j] = min(dp[i][j],dp[k][j] + dp[i^k][j]);
            } while(k&(k-壹)&i);
        }
        for (int j = 0;j < n;j++) {
            for (int k = 0;k < n;k++) {
                dp[i][j] = min(dp[i][j],dp[i][k] + d[k][j]);
            }
        }
    }
    return dp[(壹<<m)-壹][p[0]];
}

int v[maxv];

int main() {
    int n,m,l;
    while (scanf("%d%d%d",&n,&m,&l)!=EOF) {
        dij.init(n+m+壹);
        int x,y,z;
        for (int i = 壹,j;i < n+m+壹;i++) {
            scanf("%d",&j);
            dij.addEdge(0, i, j);
            dij.addEdge(i, 0, j);
        }
```

```
        while (l--) {
            scanf("%d%d%d",&x,&y,&z);
            dij.addEdge(x, y, z);
            dij.addEdge(y, x, z);
        }
        for (int i = 0;i <= n;i++)
            v[i] = i;
        dij.solve();
        printf("%d\n",steiner(n+m+壹, n+壹, v, dij.d));
    }
    return 0;
}
```

steiner_欧几里得

```
/**
    ural-壹460
    求四个点的欧几里得steiner mst
*/

int sig(double d) {
    return (d>壹E-8) - (d<-壹E-8);
}

const double eps = 壹E-6;

#define maxn 壹0
const double inf = 2048.0;

double prim(double E[maxn][maxn], int n) {
    static bool vis[maxn];
    static double d[maxn];

    memset(vis, 0, sizeof(vis));
    for(int i = 0; i < n; i ++)d[i] = inf;
    d[0] = 0;
    double res = 0;
    for(int i = 0; i < n; i ++) {
        int u = -壹;
        for(int i = 0; i < n; i ++) {
            if(!vis[i] && (u==-壹||d[i]<d[u])) {
                u = i;
            }
        }
        vis[u] = true;
        res += d[u];
        for(int v = 0; v < n; v ++) {
            if(!vis[v] && E[u][v]<d[v]) {
                d[v] = E[u][v];
            }
        }
    }
    return res;
}

struct Point {
    double x, y;
    Point() {}
    Point(double x, double y) : x(x), y(y) {}
    Point operator + (const Point & p) const {return Point(x+p.x, y+p.y); }
    Point operator - (const Point & p) const {return Point(x-p.x, y-p.y); }
    Point operator * (const double& d) const {return Point(x*d, y*d);     }
    Point left90() {    return Point(-y, x);                               }
    void input() {
        scanf("%lf%lf", &x, &y);
    }
    void output() {
        printf("(%.2f,%.2f) ", x, y);
```

```
    }
} ps[壹0];

double cross(const Point & o, const Point & a, const Point & b) {
    return (a.x-o.x)*(b.y-o.y) - (b.x-o.x)*(a.y-o.y);
}
double dot(const Point & o, const Point & a, const Point & b) {
    return (a.x-o.x)*(b.x-o.x) + (a.y-o.y)*(b.y-o.y);
}
double dis(const Point & a , const Point & b) {
    return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));
}

int lineCross(Point a, Point b, Point c, Point d, Point &p) {
    double s壹, s2;
    s壹=cross(a,b,c);
    s2=cross(a,b,d);
    if(sig(s壹)==0 && sig(s2)==0)   return 2;
    if(sig(s2-s壹)==0)  return 0;
    p.x = (c.x*s2-d.x*s壹)/(s2-s壹);
    p.y = (c.y*s2-d.y*s壹)/(s2-s壹);
    return 壹;
}

Point fermat(Point a, Point b, Point c) {
    if(sig(cross(a, b, c)) < 0)swap(b, c);

    double ab = dis(a, b);
    double bc = dis(b, c);
    double ac = dis(a, c);

    double cosA = dot(a,b,c)/(ab*ac);
    double cosB = dot(b,a,c)/(ab*bc);
    double cosC = dot(c,a,b)/(ac*bc);
    if(cosA<=-0.5)  return a;
    if(cosB<=-0.5)  return b;
    if(cosC<=-0.5)  return c;
    Point res;
    Point cc = (a-b).left90()*(sqrt(3.0)/2) + (a+b)*0.5;
    Point aa = (b-c).left90()*(sqrt(3.0)/2) + (b+c)*0.5;
    lineCross(a, aa, c, cc, res);
    return res;
}

Point mid(Point a, Point b) {
    return Point((a.x+b.x)/2.0, (a.y+b.y)/2.0);
}

double test3(Point a, Point b, Point c, Point d) {
    Point p(0,0), q(0,0);

    double ans = inf;

    Point arr[4];

    while(壹) {
        arr[0] = c;
        arr[壹] = d;
        arr[2] = p;
        q = fermat(arr[0], arr[壹], arr[2]);

        arr[0] = a;
        arr[壹] = b;
        arr[2] = q;

        p = fermat(arr[0], arr[壹], arr[2]);
```

```
        double tmpAns = dis(a,p)+dis(b,p)+dis(p,q)+dis(q,c)+dis(q,d);

        if(fabs(tmpAns - ans) < eps)   break;
        ans = min(ans, tmpAns);

    }
    return ans;
}

double E[maxn][maxn];

int main() {
    int t;
    for(scanf("%d", &t); t --; ) {
        for(int i = 0; i < 4; i ++)ps[i].input();
        //壹. mst
        for(int i = 0; i < 4; i ++) {
            for(int j = 0; j < 4; j ++) {
                E[i][j] = dis(ps[i], ps[j]);
            }
        }
        double ans = prim(E, 4);
        //2. 费马点
        for(int i = 0; i < 4; i ++) {
            Point ps2[maxn];
            int len = 0;
            double tmpAns = inf;
            for(int j = 0; j < 4; j ++){
                if(i==j)    continue;
                tmpAns = min(tmpAns, dis(ps[i],ps[j]));
                ps2[len ++] = ps[j];
            }
            Point fer = fermat(ps2[0], ps2[壹], ps2[2]);
            for(int i = 0; i < 3; i ++) {
                tmpAns += dis(fer, ps2[i]);
            }
            ans = min(ans, tmpAns);
        }
        //3.两个点的stenier
        ans = min(ans, test3(ps[0],ps[壹],ps[2],ps[3]));
        ans = min(ans, test3(ps[0],ps[2],ps[壹],ps[3]));
        ans = min(ans, test3(ps[0],ps[3],ps[壹],ps[2]));

        printf("%.4f\n", ans);
    }
    return 0;
}
```

steiner_欧几里得_topologies_计数

//【壹.】Full Steiner Topologies（n个点，n-2 个斯坦纳点，每个斯坦纳点的度为 3，图连通，最小生成树的种类数):
// f(n) = (2*n-4)!/(2^(n-2)*(n-2)!)
//   = 壹.3.5....(2n-壹)
//壹, 壹, 3, 壹 5, 壹 05, 945, 壹 0395, 壹 35 壹 35, 2027025, 34459425, 654729075, 壹 37493 壹 0575, 3 壹 6234 壹 43225, 7905853580625, 2 壹 3458046676875, 6 壹 90283353629375, 壹 9 壹 8987839625 壹 0625, 6332659870762850625, 22 壹 64309547669977 壹 875, 820079453263789 壹 559375

//【2.】Number of Steiner topologies on n points.(小于等于n-2 个点)
// F(n, k) = binomial(n, k+2) f(k) (n+k-2)! / (2k)!.
// Then a(n) = Sum_{k=0..n-2} Sum_{i=0..floor((n-k-2)/2)} binomial(n, i) F(n-i, k+i) (k+i)! / k!.
//壹, 4, 3 壹, 360, 5625, 壹壹 0880, 2643795, 74035080, 2382538725, 86656878000,

35 壹 576 壹壹 93 壹 75, 壹 57425426358200, 77 壹壹 96 壹 78 壹 949425, 4 壹 02984365 壹壹 964000, 23559634669682986875, 壹 452240056377 壹 67057000, 9564932823 壹 839993736 壹 25

```
//【poj-3595】  Full Steiner Topologies
//返回n!对壹 0 取log的结果(改进版的stirling公式)
double stirling(int n) {
    if(n<=壹)   return 0;   //通过计算，取壹时不够精确,而且要计算n<=0 的情况
    return log壹 0(sqrt(2.0*M_PI)) + (n+0.5)*log壹 0(n) - n * log壹 0( exp(壹.0-壹.0/(  0.4+壹 2.0*n*n  ) ) );
}

int main() {
    int n;
    while(scanf("%d", &n) != EOF) {
        double ans = stirling(2*n-4) - (n-2)*log壹 0(2.0) - stirling(n-2);
        ans = max(ans, 0.0);    // in case ans < 0
        int e = (int)floor(ans);
        double m = pow(壹 0.0, ans-e);
        printf("%.3fE%d\n", (double)m, e);
    }
    return 0;
}
```

基数排序

```
//对arr[壹...n]排序, arr只能为【非负数】, 排序的结果保存在order[]中, order应该事先初始化为[0,n)
void radixsort(const int *arr, const int n, int * order) {
    static int ta[65536], tb[maxn];//辅助
    memset(ta, 0, sizeof(ta));
    for(int i = 0; i < n; i++)
        ta[arr[i] & 0xffff]++;
    for(int i = 0; i < 65535; i++)
        ta[i + 壹] += ta[i];
    for(int i = n - 壹; i >= 0; i--)
        tb[--ta[arr[order[i]] & 0xffff]] = order[i];
    //上面对低壹 6 位排，下面对高壹 6 位排
    memset(ta, 0, sizeof(ta));
    for(int i = 0; i < n; i++)
        ta[arr[i] >> 壹 6]++;
    for(int i = 0; i < 65535; i++)
        ta[i + 壹] += ta[i];
    for(int i = n - 壹; i >= 0; i--)
        order[--ta[arr[tb[i]] >> 壹 6]] = tb[i];
}
void radixsort0(int *arr, int n) {//原地排序【待测】
    static int ta[65536], tb[maxn];//辅助
    memset(ta, 0, sizeof(ta));
    for(int i = 0; i < n; i++)
        ta[arr[i] & 0xffff]++;
    for(int i = 0; i < 65535; i++)
        ta[i + 壹] += ta[i];
    for(int i = n - 壹; i >= 0; i--)
        tb[--ta[arr[i] & 0xffff]] = arr[i];
    //上面对低壹 6 位排，下面对高壹 6 位排
    memset(ta, 0, sizeof(ta));
    for(int i = 0; i < n; i++)
        ta[tb[i] >> 壹 6]++;
    for(int i = 0; i < 65535; i++)
        ta[i + 壹] += ta[i];
    for(int i = n - 壹; i >= 0; i--)
        arr[--ta[tb[i] >> 壹 6]] = tb[i];
}
```

枚举子集

```
void print(int x) {
    for(int i = 3壹; i >= 0; i --) {
        printf("%d", (x>>i)&壹);
    }
    printf("\n");
}
//枚举x的全部非空/非全子集，如果x有n个壹，则一共枚举了 2^n-2 个，【保证x中有至少两个壹】
void subSet(int x) {
    for (int k = (x-壹) & x;k > 0;k = (k-壹) & x) {
        print(k);   //k is one
    }
}
//枚举x的一半子集，枚举x的一半子集，如果x有n个壹，则一共枚举了(2^n-2)/2 个   【保证x中有至少两个壹】
void halfSet(int x) {
    int k = x;
    do {
        k=(k-壹)&x;
        print(k);
    } while(k&(k-壹)&x);
}

int main() {
    int n;
    scanf("%d", &n);
    print(n);
    printf("\n");
    halfSet(n);

    return 0;
}
```

区间最大频率

```
/**
 * POJ-3368
 * You are given a sequence of n integers a壹 , a2 , ... , an in non-decreasing order. In addition to that, you are given several queries consisting of indices i and j (壹 ≤ i ≤ j ≤ n). For each query, determine the most frequent value among the integers ai , ... , aj.
 */

#define  maxn 壹0000壹0
const int inf = 0x3f3f3f3f;

int rmq[maxn];
struct ST {
#define CMP >   //大于为取大数，小于取小数
    int mm[maxn];
    int best[20][maxn];
    void init(int n) {
        int i,j,a,b;
        rmq[0] = -inf;  //让rmq[0]取最反向值
        mm[0]=-壹;
        for(i=壹; i<=n; i++) {
            mm[i]=((i&(i-壹))==0) ? mm[i-壹]+壹 : mm[i-壹];
            best[0][i]=i;
        }
        for(i=壹; i<=mm[n]; i++) {
            for(j=壹; j<=n+壹-(壹<<i); j++) {
                a=best[i-壹][j];
                b=best[i-壹][j+(壹<<(i-壹))];
                best[i][j] = rmq[a] CMP rmq[b] ? a : b;
            }
```

```
        }
    }
    int query(int a, int b) {
        if(a > b)   return 0;
        int t;
        t=mm[b-a+壹];
        a=best[t][a];
        b=best[t][b-(壹<<t)+壹];
        return rmq[a] CMP rmq[b] ? a : b;
    }
};

int arr[壹0000壹0];
int addr[壹0000壹0];
int pos[壹0000壹0];

ST st;

int main() {
    int n, q;
    int a, b;
    int x, y, z;
    while(scanf("%d", &n), n) {
        scanf("%d", &q);

        int p = 0;
        arr[0] = -inf;
        memset(rmq, 0, sizeof(rmq));
        for(int i = 壹; i <= n; i ++) {
            scanf("%d", arr+i);
            if(arr[i] != arr[i-壹]) pos[++p]=i;
            rmq[p] ++;
            addr[i] = p;
        }
        st.init(p);
        for(int i = 0; i < q; i ++) {
            scanf("%d%d", &a, &b);  //询问[a,b]内的最频率
            if(arr[a]==arr[b])  printf("%d\n", b-a+壹);
            else {
                x = rmq[st.query(addr[a]+壹, addr[b]-壹)];
                y = pos[addr[a]+壹]-a;  //不会出现addr[a]+壹>p的情况!
                z = b-pos[addr[b]]+壹;
                printf("%d\n", max(max(x,y), z));
            }
        }
    }
    return 0;
}
```

稳定婚姻问题

```
#define maxn 5壹0
```

```
/**
```

在组合数学，稳定婚姻问题指：
有n男n女，每人都按他对（异性）对象的喜好程度按壹至n排列。安排男女结婚，使得没有一对不是夫妇的男女对对方的喜好程度都较被安排的配偶高（不稳定）。

【解法】
壹962年Gale和Shapley提出一个算法，对于每个系统，总能找到一个解决的办法：

```
    函数 稳定婚姻 {
        初始所有 m in M 与 w in W 为 单身
        当 存在 单身 男子 m {
          w = m所有可考虑的女子中排名最高的(没有拒绝过他的女子中)
          若 w 是 单身
```

```
            撮合 (m, w)
          否则 有些夫妇 (m', w) 存在
            若 w 喜欢 m 更于 m'
              (m, w) 为 夫妇
              m' 为 单身
            否则
              (m', w) 仍为 夫妇
        }
    }
以上的例子是男子主动的，结果男子都尽可能得到他最想要的伴侣。
 */

int A[maxn][maxn];  //A[i][j]表示A集合中的第i个人第j喜欢的是A[i][j]
int B[maxn][maxn];  //B[i][j]表示B集合中的第i个人第j喜欢的是B[i][j]
int matchA[maxn];   //matchA[i]表示A集合中的i号最终选择B集合中的matchA[i]号
int matchB[maxn];   //matchB[i]表示B集合中的i号最终选择A集合中的matchB[i]号

typedef pair<int,int> T;

void match(int n) { //A主动
    static int rankB[maxn][maxn], tag[maxn];//B的逆, A选B的优先级号
    static T req[maxn];                     //请求, first男生，second: 女生

    for(int i = 0; i < n; i ++) {
        for(int j = 0; j < n; j ++) {
            rankB[i][ B[i][j] ] = j;    //rank越小权限越高
        }
    }
    memset(matchA, 255, sizeof(matchA));
    memset(matchB, 255, sizeof(matchB));
    memset(tag, 0, sizeof(tag));

    while(壹) {
        int len = 0;
        for(int i = 0; i < n; i ++) {
            if(matchA[i]==-壹) {
                req[len ++] = T(i, A[ i ][ tag[i]++ ]);
            }
        }
        if(0 == len)    break;
        for(int i = 0; i < len; i ++) {
            int a = req[i].first, b = req[i].second;
            if(matchB[b]==-壹) {
                matchB[b] = a;
                matchA[a] = b;
            } else if(rankB[b][a] < rankB[b][ matchB[b] ]) {
                matchA[ matchB[b] ] = -壹;
                matchB[b] = a;
                matchA[a] = b;
            }
        }
    }
}
/*
// POJ-3487
map<char, int> male, female;
char femaleName[maxn];
int main() {
    int t, n;
    char c;
    for(scanf("%d", &t); t --; ) {
        male.clear();
        female.clear();
        scanf("%d", &n);
        for(int i = 0; i < n; i ++)    scanf(" %c", &c), male[c] = i;
```

```
        for(int i = 0; i < n; i ++)    scanf(" %c", &c), female[c] = i,
femaleName[i] = c;
        char str[maxn+壹0];

        for(int i = 0; i < n; i ++) {
            scanf("%s", str);
            for(int j = 0; j < n; j ++) {
                A[i][j] = female[str[j+2]];
            }
        }
        for(int i = 0; i < n; i ++) {
            scanf("%s", str);
            for(int j = 0; j < n; j ++) {
                B[i][j] = male[str[j+2]];
            }
        }

        match(n);

        for(map<char,int>::iterator i = male.begin(); i != male.end(); i ++) {
            char name = i->first;
            int maleId = i->second;
            int matchFemale = matchA[maleId];

            printf("%c %c\n", name, femaleName[matchFemale]);
        }
        printf("\n");
    }
    return 0;
}
*/
```

找两个数异或最大

```
/*
ID: mjmjmtl壹
LANG: C++
TASK: cowxor
*/
#define maxn 壹000壹0
#define maxLev 2壹  //最大的二进制位数

struct Nod {        //0为无效值
    int lnk[2], val;
    void init() {
        lnk[0] = lnk[壹] = val = 0; //孩子比较少，直接赋值lnk了!
    }
};

struct Trie {
    Nod buf[maxn*壹0]; int len;    //buf大小看着办吧，maxn*maxLev封顶，但是没必要那
么大
    void init() {
        buf[len=0].init();
    }
    int insert(int num, int val) {
        int now = 0;
        for(int i = maxLev-壹; i >= 0; i --) {
            int dig = (num>>i) & 壹;
            int & nxt = buf[now].lnk[dig];
            if(!nxt)    buf[nxt=++len].init();
            now = nxt;
        }
        buf[now].val = val;
        return now;
    }
    int search(int num) {
```

```
        int now = 0;
        for(int i = maxLev-壹; i >= 0; i --) {
            int dig = (num>>i) & 壹;
            if(buf[now].lnk[dig^壹] != 0) {
                now = buf[now].lnk[dig^壹];
            } else {
                now = buf[now].lnk[dig];
            }
        }
        return buf[now].val;
    }
} trie;

//在arr中找两个数，使得抑或值最大。a/b返回坐标
int get(int * arr, int n, int & a, int & b) {
    trie.init();
    trie.insert(arr[0], 0);
    int ans = -壹, i, j, tmpAns;
    for(i = 壹; i < n; i ++) {
        j = trie.search(arr[i]);
        tmpAns = arr[i] ^ arr[j];
        if(tmpAns>ans) {
            ans = tmpAns;
            a=j;    b=i;
        }
        trie.insert(arr[i], i);
    }
    return ans;
}


//usaco-cow xor
//题意：给定一个数列，求最大连续异或值。
//如果有冲突，则选择最后坐标最小的
//如果还有冲突，则返回序列长度最小的
int arr[maxn];
int main() {
    freopen("cowxor.in", "r", stdin);
    freopen("cowxor.out", "w", stdout);

    int n;
    while(scanf("%d", &n) != EOF) {
        arr[0] = 0;
        for(int i = 壹; i <= n; i ++) {
            scanf("%d", arr+i);
            arr[i]^=arr[i-壹];
        }
        int num, a, b;
        num = get(arr, n+壹, a, b);
        printf("%d %d %d\n", num, 壹+a, b);
    }
    return 0;
}
```

字典序_全排列（互转）

```
/**
 * fzu-壹 57 壹
 * 给定某个排列数，球这个排列数在所有排列中所占的排名（0 为最小的排列）
 * 数字只要求互异，不要求从 0 开始或者从壹开始
 */
int arr[20];
long long jc[20];

int main() {
    jc[0] = 壹;
```

```
    for(int i = 壹; i < 20; i ++)   jc[i] = jc[i-壹] * i;
    int n;
    while(scanf("%d", &n) != EOF) {
        for(int i = 0; i < n; i ++)scanf("%d", arr+i);
        long long ans = 0;

        for(int i = 0; i < n; i ++) {
            int num = 0;
            for(int j = i+壹; j < n; j ++)  if(arr[j]<arr[i])   num ++; //数后面有几个比我小的，要加速可用树状数组
            ans += num * jc[n-壹-i];
        }
        cout << ans << endl;
        //以下代码是这个排列的下一个排列
        next_permutation(arr, arr+n);
        for(int i = 0; i < n; i ++) {
            if(i != 0)  printf(" ");
            printf("%d", arr[i]);
        }
        printf("\n");
    }
    return 0;
}


//-----------------------------------

/**
 * 字典序编号 转 排列
 * 字典序编号为 0 代表最小的排列
 */

long long jc[20];
bool vis[壹 00];

int main() {
    jc[0] = 壹;
    for(int i = 壹; i < 20; i ++)   jc[i] = jc[i-壹] * i;

    int n, k;
    while(scanf("%d%d", &n, &k) != EOF) {
        memset(vis, 0, sizeof(vis));
        for(int i = 0; i < n; i ++) {
            //我的后面有n-壹-i个数字
            int tmp = k / jc[n-壹-i];
            int count = 0;
            int j = 0;
            while(壹) {
                if(false == vis[j]) {
                    if(count == tmp) {
                        vis[j] = 壹;
                        break;
                    }
                    count ++;
                }
                j ++;
            }
            printf("%d ", j);
            k -= tmp * jc[n-壹-i];
        }
        printf("\n");
    }
    return 0;
}
```

历法

```
/**
输入年月日，输出星期几（或者日期无效）
标准格林兰治（壹 752 年差了壹 0 天，java 类库使用）
----------
    注意：s
        壹.最好不要用 after,before 函数
        2.month 值为真实值-壹
        3.星期值 Sunday 为壹，最好用常量
*/
import java.util.Calendar;
import java.util.GregorianCalendar;
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        String[] week =
{"","Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday"};
        String[] monStr = {"","January", "February", "March", "April", "May",
"June", "July", "August", "September", "October", "November", "December"};
        int mon[]={0,3壹,29,3壹,30,3壹,30,3壹,3壹,30,3壹,30,3壹},month, day,
year;
        Scanner scan = new Scanner(System.in);
        GregorianCalendar end = new GregorianCalendar(壹752,8,壹4),cal = new
GregorianCalendar();
        cal.setGregorianChange(end.getTime());
        while(scan.hasNext()) {
            month = scan.nextInt();
            day = scan.nextInt();
            year = scan.nextInt();
            if(month == 0 && day == 0 && year == 0)    break;
            cal.set(year, month-壹, day);
            if(month<壹||month>壹
2||day>mon[month]||(month==2&&day>28&&!cal.isLeapYear(year))||(year==壹
752&&month==9&&day>2&&day<壹4))
                System.out.println(month+"/" + day + "/" +year+" is an invalid
date.");
            else        System.out.println(monStr[month] + " " + day + ", " + year
+ " is a " + week[cal.get(Calendar.DAY_OF_WEEK)]);
        }
    }
}
```

```
//   输入年、月、日，返回星期（0 代表星期日）
int caiLe(int Y, int M, int D) {
    int y, c, w;
    if(M == 壹 || M == 2) {
        Y -= 壹;
        M += 壹 2;
    }
    y=Y%壹 00;
    c=Y/壹 00;
    w=((c/4)-2*c+y+(y/4)+(壹 3*(M+壹)/5)+D-壹)%7;
    return w < 0 ? w+7 : w;
}
```

```
//输入：时间            返回：距离壹 970-壹-壹 00:00:00 偏移的秒数
int getSecond(int year, int mon,int day, int hour,int min, int sec) {
    if (0 >= (int) (mon -= 2)) { /* 壹..壹 2 -> 壹壹,壹 2,壹..壹 0 */
        mon += 壹 2; /* Puts Feb last since it has leap day */
        year -= 壹;
    }
    return
    (   (   (   (year/4 - year/壹 00 + year/400 + 367*mon/壹 2 + day
                ) + year*365 - 7 壹 9499
            )*24 + hour /* now have hours */
        )*60 + min /* now have minutes */
    )*60 + sec; /* finally seconds */
```

}

Calendar 类库总结

```
/**
 * 原始题目：http://acm.jlu.edu.cn/joj/showproblem.php?pid=壹370
输入年月日，输出星期几（或者日期无效）
标准格林兰治（壹752年差了壹0天，java类库使用）
----------------
     注意：s
壹.最好不要用after,before函数
          2.month值为真实值-壹
          3.星期值Sunday为壹，最好用常量
*/
import java.util.GregorianCalendar;
import java.util.Scanner;
import java.util.SimpleTimeZone;
public class Main {
     /**
      * 返回由year, mon, day 决定的标准时间，时区为GMT+0，并且不设置宽松
      */
     public static GregorianCalendar getStdGC(int year, int mon, int day) {
          GregorianCalendar res = new GregorianCalendar(year, mon, day);//设置时间
          res.setTimeZone(new SimpleTimeZone(0, ""));//设置时区
          res.setLenient(false);//设置非宽松
          return res;
     }

     public static void main(String[] args) {
          Scanner scan = new Scanner(System.in);
          GregorianCalendar cal = getStdGC(2000,0,壹);    //随意设置一个时间
          cal.setGregorianChange(getStdGC(壹752, 9-壹, 壹4).getTime());//壹752-9-壹4是无效日期
的结束

          //设置 GregorianCalendar 的更改日期。这是发生从儒略历日期切换到格里高利历日期的点。默认时间是 壹
582 年 壹0 月 壹5 日（格里高利历）。在此之前，日期是按照儒略历计算的。
         //要得到纯粹的儒略历日历，可将更改日期设置为 Date(Long.MAX_VALUE)。要得到一个纯粹的格里高利历日
历，可将更改日期设置为 Date(Long.MIN_VALUE)。

          //这里设置的无效时间段为[壹752-9-3, 壹752-9-壹4)区间无效，之前为儒略历，之后为格里高利历

          //星期如下：
          //星期二 九月 0壹 00:00:00 GMT+00:00 壹752
          //星期三 九月 02 00:00:00 GMT+00:00 壹752
          //星期四 九月 壹4 00:00:00 GMT+00:00 壹752
          //通过add调整，日期也会直接跳跃!!

          //儒略历指定每 4 年就为闰年，而格里高利历则忽略不能被 400 整除的世纪年
          //时区必须设置为0!!!! 否则容易出错
          int month, day, year;
          while(scan.hasNext()) {
               month = scan.nextInt();
                         day = scan.nextInt();
                         year = scan.nextInt();
               if(month == 0 && day == 0 && year == 0)   break;
               cal.set(year, month-壹, day);     //设置日期
               try {
                    cal.getTimeInMillis();       //由于set时有错并未抛出异常，这里为了抛出异常。。。
                    //设置的月、日超出范围会抛出异常，时间在[壹752-9-3, 壹752-9-壹4)内也会抛出异常
                    System.out.printf("%tB %d, %d is a %tA\n", cal, day, year, cal);//具体Formatter
格式见下。。。
               } catch (Exception e) {
                    System.out.println(month+"/" + day + "/" +year+" is an invalid date.");
               }
          }
          /*
           *  Formatter对"日期/时间"的支持
               此转换可应用于 long、Long、Calendar 和 Date。
               't'  '\u0074'  日期和时间转换字符的前缀。
               'T'  '\u0054'  't' 的大写形式。
               //[壹752-9-3, 壹752-9-壹4)区间无效，之前为儒略历，之后为格里高利历
               以下日期和时间转换字符后缀是为 't' 和 'T' 转换定义的。这些类型类似于但不完全等同于 GNU date
和 POSIX strftime(3c) 定义的那些类型。提供其他转换类型是为了访问特定于 Java 的功能（例如，'L' 用于秒中的毫
秒）。

               以下转换字符用来格式化时间：
```

'H' '\u0048' 24 小时制的小时，被格式化为必要时带前导零的两位数，即 00 - 23。00 对应午夜。
'I' '\u0049' 壹 2 小时制的小时，被格式化为必要时带前导零的两位数，即 0 壹 - 壹 2。0 壹 对应于 壹 点钟（上午或下午）。
'k' '\u006b' 24 小时制的小时，即 0 - 23。0 对应于午夜。
'l' '\u006c' 壹 2 小时制的小时，即 壹 - 壹 2。壹 对应于上午或下午的一点钟。
'M' '\u004d' 小时中的分钟，被格式化为必要时带前导零的两位数，即 00 - 59。
'S' '\u0053' 分钟中的秒，被格式化为必要时带前导零的两位数，即 00 - 60（"60" 是支持闰秒所需的一个特殊值）。
'L' '\u004c' 秒中的毫秒，被格式化为必要时带前导零的三位数，即 000 - 999。
'N' '\u004e' 秒中的毫微秒，被格式化为必要时带前导零的九位数，即 000000000 - 999999999。此值的精度受底层操作系统或硬件解析的限制。
'p' '\u0070' 特定于语言环境的上午或下午标记以小写形式表示，例如 "am" 或 "pm"。使用转换前缀 'T' 可以强行将此输出转换为大写形式。（注意，'p' 产生的输出是小写的。而 GNU date 和 POSIX strftime(3c) 产生的输出是大写的。）
'z' '\u007a' 相对于 GMT 的 RFC 822 格式的数字时区偏移量，例如 -0800。
'Z' '\u005a' 表示时区的缩写形式的字符串。
's' '\u0073' 自协调世界时 (UTC) 壹 970 年 壹 月 壹 日 00:00:00 至现在所经过的秒数，也就是 Long.MIN_VALUE/壹 000 与 Long.MAX_VALUE/壹 000 之间的差值。
'Q' '\u004f' 自协调世界时 (UTC) 壹 970 年 壹 月 壹 日 00:00:00 至现在所经过的毫秒数，即 Long.MIN_VALUE 与 Long.MAX_VALUE 之间的差值。此值的精度受底层操作系统或硬件解析的限制。

以下转换字符用来格式化日期：
'B' '\u0042' 特定于语言环境的完整月份名称，例如 "January" 和 "February"。
'b' '\u0062' 特定于语言环境的月份简称，例如 "Jan" 和 "Feb"。
'h' '\u0068' 与 'b' 相同。
'A' '\u004 壹' 特定于语言环境的星期几的全称，例如 "Sunday" 和 "Monday"
'a' '\u006 壹' 特定于语言环境的星期几的简称，例如 "Sun" 和 "Mon"
'C' '\u0043' 除以 壹 00 的四位数表示的年份，被格式化为必要时带前导零的两位数，即 00 - 99
'Y' '\u0059' 年份，被格式化为必要时带前导零的四位数（至少），例如 0092 等于格里高利历的 92 CE。
'y' '\u0079' 年份的最后两位数，被格式化为必要时带前导零的两位数，即 00 - 99。
'j' '\u006a' 一年中的天数，被格式化为必要时带前导零的三位数，例如，对于格里高利历是 00 壹 - 366。00 壹 对应于一年中的第一天。
'm' '\u006d' 月份，被格式化为必要时带前导零的两位数，即 0 壹 - 壹 3，其中 "0 壹" 是一年的第一个月，("壹 3" 是支持阴历所需的一个特殊值)。
'd' '\u0064' 一个月中的天数，被格式化为必要时带前导零的两位数，即 0 壹 - 3 壹，其中 "0 壹" 是一个月的第一天。
'e' '\u0065' 一个月中的天数，被格式化为两位数，即 壹 - 3 壹，其中 "壹" 是一个月中的第一天。

以下转换字符用于格式化常见的日期/时间组合。
'R' '\u0052' 24 小时制的时间，被格式化为 "%tH:%tM"
'T' '\u0054' 24 小时制的时间，被格式化为 "%tH:%tM:%tS"。
'r' '\u0072' 壹 2 小时制的时间，被格式化为 "%tI:%tM:%tS %Tp"。上午或下午标记 ('%Tp') 的位置可能与地区有关。
'D' '\u0044' 日期，被格式化为 "%tm/%td/%ty"。
'F' '\u0046' ISO 860 壹 格式的完整日期，被格式化为 "%tY-%tm-%td"。
'c' '\u0063' 日期和时间，被格式化为 "%ta %tb %td %tT %tZ %tY"，例如 "Sun Jul 20 壹 6:壹 7:00 EDT 壹 969"。

应用为常规转换而定义的 '-' 标志。如果给出 '#' 标志，则将抛出 FormatFlagsConversionMismatchException。

宽度 是将向输出中写入的最少字符数。如果转换值的长度小于 width，则用空格('\u0020') 来填充输出，直到总字符数等于宽度。默认情况下，在左边进行填充。如果给出 '-' 标志，则在右边进行填充。如果没有指定宽度，则没有最小宽度。

精度不适用。如果指定了精度，则将抛出 IllegalFormatPrecisionException。
*/
}
}
/*
* 其他需要关注的函数：
* boolean after(Object when)
* boolean before(Object when)
* int compareTo(Calendar anotherCalendar)
* void set(int field, int value)
* void set(int year, int month, int date)//set如果结果不正确，不立即计算，需要get...
*
* void add(int field, int amount)
* void roll(int field, int amount)
* add和roll的区别在于roll不改变较大的filed，而add会改变
*
* int get(int field)

```
 * int getMaximum(int field)
 * int getMinimum(int field)
 * Date getTime()
 * long getTimeInMillis()
 * boolean isLeapYear(int year)  //和儒略历、格里高利历也有关系
*/
/*常量字段:
     DATE                    get和set的字段数字，指示一个月中的某天。          壹.2...
     DAY_OF_WEEK                  get和set的字段数字，指示一个星期中的某天。       Sunday:    壹    ,
Monday:2...
     DAY_OF_WEEK_IN_MONTH    get和set的字段数字，指示当前月中的第几个星期几    壹.2...
     DAY_OF_YEAR                  get和set的字段数字，指示当前年中的天数。          壹.2...
     *ERA                         指示年代的get和set的字段数字，比如罗马儒略历中的AD或BC。
     HOUR                    get和set的字段数字，指示上午或下午的小时。        0...
     HOUR_OF_DAY                  get和set的字段数字，指示一天中的小时。           0...
     MILLISECOND                  get和set的字段数字，指示一秒中的毫秒。           0...
     MINUTE                       get和set的字段数字，指示一小时中的分钟。         0...
     MONTH                        指示月份的get和set的字段数字。                  January:0...
     SECOND                       get和set的字段数字，指示一分钟中的秒。           0...
     WEEK_OF_MONTH           get和set的字段数字，指示当前月中的星期数。
          星期按照FirstDayOfWeek()划分，如果第一个星期天数小于getMinimalDaysInFirstWeek()，则从 0 开始，
否则从壹开始
     WEEK_OF_YEAR            get和set的字段数字，指示当前年中的星期数。
          星期按照FirstDayOfWeek()划分，如果第一个星期天数小于getMinimalDaysInFirstWeek()，则从去年结
尾开始，否则从壹开始
     YEAR                    指示年的get和set的字段数字。                    ...
*/
```

位运算

首先说**明**，**位运算最好在** unsigned 下进**行操作**，**避免** int 的-壹位右移产**生**错误

壹.简单**位运算**：

功能	示例	位运算
去掉最后一位	(壹 0 壹壹 0 壹->壹 0 壹壹 0)	x shr 壹
在最后加一个 0	(壹 0 壹壹 0 壹->壹 0 壹壹 0 壹 0)	x shl 壹
在最后加一个壹	(壹 0 壹壹 0 壹->壹 0 壹壹 0 壹壹)	x shl 壹+壹
把最后一位变**成**壹	(壹 0 壹壹 00->壹 0 壹壹 0 壹)	x or 壹
把最后一位变**成** 0	(壹 0 壹壹 0 壹->壹 0 壹壹 00)	x or 壹-壹
最后一位取反	(壹 0 壹壹 0 壹->壹 0 壹壹 00)	x xor 壹
把右数第 k 位变**成**壹	(壹 0 壹 00 壹->壹 0 壹壹 0 壹,k=3)	x or (壹 shl (k-壹))
把右数第 k 位变**成** 0	(壹 0 壹壹 0 壹->壹 0 壹 00 壹,k=3)	x and not (壹 shl (k-壹))
右数第 k 位取反	(壹 0 壹 00 壹->壹 0 壹壹 0 壹,k=3)	x xor (壹 shl (k-壹))
取末三位	(壹壹 0 壹壹 0 壹->壹 0 壹)	x and 7
取末 k 位	(壹壹 0 壹壹 0 壹->壹壹 0 壹,k=5)	x and (壹 shl k-壹)
取右数第 k 位	(壹壹 0 壹壹 0 壹->壹,k=4)	x shr (k-壹) and 壹
把末 k 位变**成**壹	(壹 0 壹 00 壹->壹 0 壹壹壹壹,k=4)	x or (壹 shl k-壹)
末 k 位取反	(壹 0 壹 00 壹->壹 00 壹壹 0,k=4)	x xor (壹 shl k-壹)
把右边连续**的**壹变**成** 0	(壹 00 壹 0 壹壹壹壹->壹 00 壹 00000)	x and (x+壹)
把右起第一个 0 变**成** 壹	(壹 00 壹 0 壹壹壹壹->壹 00 壹壹壹壹壹壹)	x or (x+壹)
把右边连续**的** 0 变**成**壹	(壹壹 0 壹壹 000->壹壹 0 壹壹壹壹壹)	x or (x-壹)
去掉右面第一个壹	(壹壹 0 壹壹 0 壹 0->壹壹 0 壹壹 000)	x and (x-壹)
取右边连续**的**壹	(壹 00 壹 0 壹壹壹壹->壹壹壹壹)	(x xor (x+壹)) shr 壹
去掉右起第一个壹的左边	(壹 00 壹 0 壹 000->壹 000)	x and (x xor (x-壹))

最后这**一个在**树**状数**组**中会用到**。

2.判断二进**制中**壹的个数有 odd 还**是** even

```
var
  x:longint;
begin
  readln(x);
```

```
  x:=x xor (x shr 壹);
  x:=x xor (x shr 2);
  x:=x xor (x shr 4);
  x:=x xor (x shr 8);
  x:=x xor (x shr 壹 6);
  writeln(x and 壹);
end.
```

3.计算二进制中的壹的个数

```
x := (x and $55555555) + ((x shr 壹) and $55555555);
x := (x and $33333333) + ((x shr 2) and $33333333);
x := (x and $0F0F0F0F) + ((x shr 4) and $0F0F0F0F);
x := (x and $00FF00FF) + ((x shr 8) and $00FF00FF);
x := (x and $0000FFFF) + ((x shr 壹 6) and $0000FFFF);
```

4.二分查找 32 位整数的前导 0 个数

```
int nlz(unsigned x)
{
  int n;
  if (x == 0) return(32);
  n = 壹;
  if ((x >> 壹 6) == 0) {n = n +壹 6; x = x <<壹 6;}
  if ((x >> 24) == 0) {n = n + 8; x = x << 8;}
  if ((x >> 28) == 0) {n = n + 4; x = x << 4;}
  if ((x >> 30) == 0) {n = n + 2; x = x << 2;}
  n = n - (x >> 3 壹);
  return n;
}
```

5.高低位交换：(x<<壹 6) | (x>>壹 6)

6.二进制逆序

```
var
  x:dword;
begin
  readln(x);
  x := (x and $55555555) shl  壹 or (x and $AAAAAAAA) shr  壹;
  x := (x and $33333333) shl  2 or (x and $CCCCCCCC) shr  2;
  x := (x and $0F0F0F0F) shl  4 or (x and $F0F0F0F0) shr  4;
  x := (x and $00FF00FF) shl  8 or (x and $FF00FF00) shr  8;
  x := (x and $0000FFFF) shl 壹 6 or (x and $FFFF0000) shr 壹 6;
  writeln(x);
end.
```

7.格雷码(Gray)

它的应用范围很广。比如，n 阶的 Gray 码相当于在 n 维立方体上的 Hamilton 回路，因为沿着立方体上的边走一步，n 维坐标中只会有一个值改变。再比如，Gray 码和 Hanoi 塔问题等价。Gray 码改变的是第几个数，Hanoi 塔就该移动哪个盘子。比如，3 阶的 Gray 码每次改变的元素所在位置依次为壹-2-壹-3-壹-2-壹，这正好是 3 阶 Hanoi 塔每次移动盘子编号。如果我们可以快速求出 Gray 码的第 n 个数是多少，我们就可以输出任意步数后 Hanoi 塔的移动步骤。现在我告诉你，Gray 码的第 n 个数（从 0 算起）是【{{{n xor (n shr 壹)}}}，你能想出来这是为什么吗？先自己想想吧。

二进制数	Gray 码
000	000
00 壹	00 壹
0 壹 0	0 壹壹
0 壹壹	0 壹 0
壹 00	壹壹 0
壹 0 壹	壹壹壹
壹壹 0	壹 0 壹
壹壹壹	壹 00

代码：

```
unsigned decimal_to_gray(unsigned x) {
    return x^(x>>壹);
}
unsigned gray_to_decimal(unsigned x) { //与判断奇偶个壹类似
    x^=x>>壹;
    x^=x>>2;
    x^=x>>4;
```

```
    x^=x>>8;
    x^=x>>壹 6;
    return x;
}
```

【扩展壹】二维 Gray(sgu249)：

把 0 到 2^(n+m)-壹的数写成 2^n * 2^m 的矩阵，使得位置相邻两数的二进制表示只有一位之差。

代码：

```
/**
    Sample Input
    壹 壹
    Sample Output
    0 2
    壹 3
*/
int main() {
    unsigned n, m;
    while(scanf("%d%d", &n, &m) != EOF) {
        for(unsigned i = 0; i < (壹u<<n); i ++) {
            for(unsigned j = 0; j < (壹u<<m); j ++) {
                printf("%u ", decimal_to_gray(j)<<n | decimal_to_gray(i));
            }
            printf("\n");
        }
    }
    return 0;
}
```

【扩展 2】：Gray Code 可用于解"连环锁"，n 个连环锁，如果 0 代表卸下，壹代表安上，右端是开始处，连环锁如下：xxy 壹 00000[start]，规则如下：

壹.最右端的可以任意安上或者卸下.

2.最右端的第一个壹左面的那个锁（这里是 y），可以任意安上或者卸下

关键：Gray Code 的顺序是"连环锁"从一个状态到另一个状态的【【【最快方式】】】

【例如】：POJ-壹 832，"连环锁"从一种状态到另一种状态的最快次数，Input 的右端是开始

```
Sample Input
    2       //测试数据组数
    3       //节点个数
    0 0 0       //状态壹，右端是开始
    壹 0 0      //状态 2，右端是开始
    4       //etc...
    壹 0 0 0
    0 壹 壹 0
Sample Output
    7
    壹壹
```

代码：

```
    import java.math.BigInteger;
    import java.util.Scanner;
    public class Main {
        static Scanner scan = new Scanner(System.in);
        static BigInteger g2d(BigInteger x) {
            for(int i = 0; i <= 壹 0; i ++) {
                x = x.xor(x.shiftRight(壹<<i));
            }
            return x;
        }
        public static void main(String[] args) {
            char tmp[] = new char[壹 30];
            for(int t = scan.nextInt(); t != 0; t --) {
                int n = scan.nextInt();
                for(int i = 0; i < n; i ++){
                    tmp[i]=(char)(scan.nextInt()+'0');
                }
                String a = new String(tmp, 0, n);
                for(int i = 0; i < n; i ++){
```

```
                    tmp[i]=(char)(scan.nextInt()+'0');
                }
                String b = new String(tmp, 0, n);
                BigInteger A = new BigInteger(a, 2);
                BigInteger B = new BigInteger(b, 2);
                System.out.println(g2d(A).subtract(g2d(B)).abs());
            }
        }
    }
```

8.皇后

```
    int sum = 0, upperlim = 壹;
    void test(int row, int ld, int rd) {
        if(row != upperlim) {
            int pos = upperlim & ~(row | ld | rd);
            for( ; pos ; ) {
                int p = pos & -pos;
                pos -= p;
                test(row | p, (ld | p) << 壹, (rd | p) >> 壹);
            }
        } else sum++;
    }
    int main() {
        int n ;
        while(scanf("%d", &n) != EOF) {
            upperlim = (壹 << n) - 壹;
            sum = 0;
            test(0, 0, 0);
            printf("%d\n", sum);
        }
        return 0;
    }
```

Int64

原地址:http://www.byvoid.com/blog/c-int64/

C/C++的64位整型

計算機技術 Add comments壹,4壹3 views

在C/C++中，64为整型一直是一种没有确定规范的数据类型。现今主流的编译器中，对64为整型的支持也是标准不一，形态各异。一般来说，64位整型的定义方式有long long和__int64两种(VC还支持_int64)，而输出到标准输出方式有 printf(“%lld”,a)，printf(“%I64d”,a)，和cout << a三种方式。

本文讨论的是五种常用的C/C++编译器对 64 位整型的支持，这五种编译器分别是gcc(mingw32)，g++(mingw32)，gcc(linux i386)，g++(linux i386)，Microsoft Visual C++ 6.0。可惜的是，没有一种定义和输出方式组合，同时兼容这五种编译器。为彻底弄清不同编译器对64位整型，我写了程序对它们进行了评测，结果如下表。

变量定义	输出方式	gcc(mingw32)	g++(mingw32)	gcc(linux i386)	g++(linux i386)	MicrosoftVisual C++ 6.0
long long	“%lld”	错误	错误	正确	正确	无法编译
long long	“%I64d”	正确	正确	错误	错误	无法编译
__int64	“lld”	错误	错误	无法编译	无法编译	错误
__int64	“%I64d”	正确	正确	无法编译	无法编译	正确
long long	cout	非C++	正确	非C++	正确	无法编译
__int64	cout	非C++	正确	非C++	无法编译	无法编译
long long	printint64()	正确	正确	正确	正确	无法编译

上表中，正确指编译通过，运行完全正确；错误指编译虽然通过，但运行结果有误；无法编译指编译器根本不能编译完成。观察上表，我们可以发现以下几点：

壹. long long定义方式可以用于gcc/g++，不受平台限制，但不能用于VC6.0。

2. __int64是Win32平台编译器64位长整型的定义方式，不能用于Linux。

3. “%lld”用于Linux i386平台编译器，”%I64d”用于Win32平台编译器。

4. cout只能用于C++编译，在VC6.0中，cout不支持64位长整型。

表中最后一行输出方式中的printint64()是我自己写的一个函数，可以看出，它的兼容性要好于其他所有的输出方式，它是一段这样的代码：

?View Code CPP

```
void printint64(long long a)
{
```

```
    if (a<=壹 00000000)
        printf("%d\n",a);
    else
    {
        printf("%d",a/壹 00000000);
        printf("%08d\n",a%壹 00000000);
    }
}
```

这种写法的本质是把较大的 64 位整型拆分为两个 32 位整型，然后依次输出，低位的部分要补 0。看似很笨的写法，效果如何？我把它和cout输出方式做了比较，因为它和cout都是C++支持跨平台的。首先printint64()和cout(不清空缓冲区)的运行结果是完全相同的，不会出现错误。我的试验是分别用两者输出壹 000000 个随机数，实际结果是，printint64()在壹.5s内跑完了程序，而cout需要 2s。cout要稍慢一些，所以在输出大量数据时，要尽量避免使用。

数学公式

是表征自然界不同事物之数量之间的或等或不等的联系，它确切的反映了事物内部和外部的关系，是我们从一种事物到达另一种事物的依据，使我们更好的理解事物的本质和内涵。

如一些基本公式

抛物线：y = ax^2 + bx + c

就是 y 等于 a(x 的平方)加上 bx 再加上 c

a > 0 时开口向上

a < 0 时开口向下

c = 0 时抛物线经过原点

b = 0 时抛物线对称轴为 y 轴

a=0 该函数为一次函数

还有顶点式 y = a(x+h)* 2+ k (-b/2a,(4ac-b*2)/4a)

就是 y 等于 a 乘以（x+h）的平方+k

-h 是顶点坐标的 x

k 是顶点坐标的 y

一般用于求最大值与最小值

抛物线标准方程:y^2=2px

它表示抛物线的焦点在 x 的正半轴上,焦点坐标为(p/2,0) 准线方程为 x=-p/2

由于抛物线的焦点可在任意半轴,故共有标准方程 y^2=2px y^2=-2px x^2=2py x^2=-2py

圆：体积=4/3(pi）(r^3)

面积=(pi)(r^2)

周长=2(pi)r

圆的标准方程 (x-a)2+(y-b)2=r2 注：（a,b）是圆心坐标

圆的一般方程 x2+y2+Dx+Ey+F=0 注：D2+E2-4F>0

（一）椭圆周长计算公式

椭圆周长公式：L=2πb+4(a-b)

椭圆周长定理：椭圆的周长等于该椭圆短半轴长为半径的圆周长（2πb）加上四倍的该椭圆长半轴长（a）与短半轴长（b）的差。

（二）椭圆面积计算公式

椭圆面积公式： S=πab

椭圆面积定理：椭圆的面积等于圆周率（π）乘该椭圆长半轴长（a）与短半轴长（b）的乘积。

以上椭圆周长、面积公式中虽然没有出现椭圆周率 T，但这两个公式都是通过椭圆周率 T 推导演变而来。常数为体，公式为用。

椭圆形物体 体积计算公式椭圆 的 长半径*短半径*PAI*高

三角函数：

两角和公式

sin(A+B)=sinAcosB+cosAsinB ；

sin(A-B)=sinAcosB - sinBcosA ；

cos(A+B)=cosAcosB - sinAsinB ；

cos(A-B)=cosAcosB + sinAsinB ；

tan(A+B)=(tanA+tanB)/(壹-tanAtanB)；

tan(A-B)=(tanA-tanB)/(壹+tanAtanB) ；

cot(A+B)=(cotAcotB-壹)/(cotB+cotA) ；

cot(A-B)=(cotAcotB+壹)/(cotB-cotA) ；

倍角公式

tan2A=2tanA/(壹-tan^2A) ； cot2A=(cot^2A-壹)/2cota ；

cos2a=cos^2a-sin^2a=2cos^2a-壹=壹-2sin^2a ；

sin2A=2sinAcosA=2/(tanA+cotA)；

sinα+sin(α+2π/n)+sin(α+2π*2/n)+sin(α+2π*3/n)+……+sin[α+2π*(n-壹)/n]=0 ；

cosα+cos(α+2π/n)+cos(α+2π*2/n)+cos(α+2π*3/n)+……+cos[α+2π*(n-壹)/n]=0 以及

sin^2(α)+sin^2(α-2π/3)+sin^2(α+2π/3)=3/2 ；

tanAtanBtan(A+B)+tanA+tanB-tan(A+B)=0；

四倍角公式：

sin4A=-4*(cosA*sinA*(2*sinA^2-壹))

cos4A=壹+(-8*cosA^2+8*cosA^4)

tan4A=(4*tanA-4*tanA^3)/(壹-6*tanA^2+tanA^4)

五倍角公式：

sin5A=壹 6sinA^5-20sinA^3+5sinA

cos5A=壹 6cosA^5-20cosA^3+5cosA

tan5A=tanA*(5-壹 0*tanA^2+tanA^4)/(壹-壹 0*tanA^2+5*tanA^4)

六倍角公式：

sin6A=2*(cosA*sinA*(2*sinA+壹)*(2*sinA-壹)*(-3+4*sinA^2))

cos6A=((-壹+2*cosA^2)*(壹 6*cosA^4-壹 6*cosA^2+壹))

tan6A=(-6*tanA+20*tanA^3-6*tanA^5)/(-壹+壹 5*tanA^2-壹 5*tanA^4+tanA^6)

七倍角公式：

sin7A=-(sinA*(56*sinA^2-壹壹 2*sinA^4-7+64*sinA^6))

cos7A=(cosA*(56*cosA^2-壹壹 2*cosA^4+64*cosA^6-7))

tan7A=tanA*(-7+35*tanA^2-2 壹 *tanA^4+tanA^6)/(-壹+2 壹 *tanA^2-35*tanA^4+7*tanA^6)

八倍角公式：

sin8A=-8*(cosA*sinA*(2*sinA^2-壹)*(-8*sinA^2+8*sinA^4+壹))

cos8A=壹+(壹 60*cosA^4-256*cosA^6+壹 28*cosA^8-32*cosA^2)

tan8A=-8*tanA*(-壹 +7*tanA^2-7*tanA^4+tanA^6)/(壹 -28*tanA^2+70*tanA^4-28*tanA^6+tanA^8)

九倍角公式：

sin9A=(sinA*(-3+4*sinA^2)*(64*sinA^6-96*sinA^4+36*sinA^2-3))

cos9A=(cosA*(-3+4*cosA^2)*(64*cosA^6-96*cosA^4+36*cosA^2-3))

tan9A=tanA*(9-84*tanA^2+壹 26*tanA^4-36*tanA^6+tanA^8)/(壹-36*tanA^2+壹 26*tanA^4-84*tanA^6+9*tanA^8)

十倍角公式：

sin 壹 0A=2*(cosA*sinA*(4*sinA^2+2*sinA-壹)*(4*sinA^2-2*sinA-壹)*(-20*sinA^2+5+壹 6*sinA^4))

cos 壹 0A=((-壹+2*cosA^2)*(256*cosA^8-5 壹 2*cosA^6+304*cosA^4-48*cosA^2+壹))

tan 壹 0A=-2*tanA*(5-60*tanA^2+壹 26*tanA^4-60*tanA^6+5*tanA^8)/(-壹+45*tanA^2-2 壹 0*tanA^4+2 壹 0*tanA^6-45*tanA^8+tanA^壹 0)

·万能公式：

sinα=2tan(α/2)/[壹+tan^2(α/2)]

cosα=[壹-tan^2(α/2)]/[壹+tan^2(α/2)]

tanα=2tan(α/2)/[壹-tan^2(α/2)]

半角公式

sin(A/2)=√((壹-cosA)/2) sin(A/2)=-√((壹-cosA)/2)

cos(A/2)=√((壹+cosA)/2) cos(A/2)=-√((壹+cosA)/2)

tan(A/2)=√((壹-cosA)/((壹+cosA)) tan(A/2)=-√((壹-cosA)/((壹+cosA))

cot(A/2)=√((壹+cosA)/((壹-cosA)) cot(A/2)=-√((壹+cosA)/((壹-cosA))

和差化积

2sinAcosB=sin(A+B)+sin(A-B)；2cosAsinB=sin(A+B)-sin(A-B) ；

2cosAcosB=cos(A+B)-cos(A-B) ；-2sinAsinB=cos(A+B)-cos(A-B) ；

sinA+sinB=2sin((A+B)/2)cos((A-B)/2 ；cosA+cosB=2cos((A+B)/2)sin((A-B)/2) ；

tanA+tanB=sin(A+B)/cosAcosB；tanA-tanB=sin(A-B)/cosAcosB ；

cotA+cotB=sin(A+B)/sinAsinB；-cotA+cotB=sin(A+B)/sinAsinB ；

降幂公式

sin²(A)=(壹-cos(2A))/2=versin(2A)/2；

cos²(α)=(壹+cos(2A))/2=covers(2A)/2；

tan²(α)=(壹-cos(2A))/(壹+cos(2A))；

某些数列前 n 项和

壹+2+3+4+5+6+7+8+9+...+n=n(n+壹)/2 壹+3+5+7+9+壹壹+壹 3+壹 5+...+(2n-壹)=n2

2+4+6+8+壹 0+壹 2+壹 4+...+(2n)=n(n+壹) 壹^2+2^2+3^2+4^2+5^2+6^2+7^2+8^2+...+n^2=n(n+壹)(2n+壹)/6

壹^3+2^3+3^3+4^3+5^3+6^3+...n^3=(n(n+壹)/2)^2 壹*2+2*3+3*4+4*5+5*6+6*7+...+n(n+壹)=n(n+壹)(n+2)/3

正弦定理 a/sinA=b/sinB=c/sinC=2R 注：其中 R 表示三角形的外接圆半径

余弦定理 b2=a2+c2-2accosB 注：角 B 是边 a 和边 c 的夹角

乘法与因式分 a2-b2=(a+b)(a-b) a3+b3=(a+b)(a2-ab+b2) a3-b3=(a-b(a2+ab+b2)

三角不等式 -|a|≤a≤|a|

|a|≤b<=>-b≤a≤b

|a|≤b<=>-b≤a≤b

|a|-|b|≤|a+b|≤|a|+|b| |a|≤b<=>-b≤a≤b

|a|-|b|≤|a-b|≤|a|+|b|

|z 壹|-|z2|-...-|zn|≤|z 壹+z2+...+zn|≤|z 壹|+|z2|+...+|zn|

|z 壹|-|z2|-...-|zn|≤|z 壹-z2-...-zn|≤|z 壹|+|z2|+...+|zn|

|z 壹|-|z2|-...-|zn|≤|z 壹±z2±...±zn|≤|z 壹|+|z2|+...+|zn|

一元二次方程的解 -b+√(b2-4ac)/2a -b-√(b2-4ac)/2a

根与系数的关系 x 壹+x2=-b/a x 壹*x2=c/a 注：韦达定理

判别式 b2-4a=0 注：方程有相等的两实根

b2-4ac>0 注：方程有两个不相等的个实根

b2-4ac<0 注：方程有共轭复数根

公式分类 公式表达式

圆的标准方程 (x-a)2+(y-b)2=r2 注：（a,b）是圆心坐标

圆的一般方程 x2+y2+Dx+Ey+F=0 注：D2+E2-4F>0

抛物线标准方程 y2=2px y2=-2px x2=2py x2=-2py

直棱柱侧面积 S=c*h 斜棱柱侧面积 S=c'*h

正棱锥侧面积 S=壹/2c*h' 正棱台侧面积 S=壹/2(c+c')h'

圆台侧面积 S=壹/2(c+c')l=pi(R+r)l 球的表面积 S=4pi*r2

圆柱侧面积 S=c*h=2pi*h 圆锥侧面积 S=壹/2*c*l=pi*r*l

弧长公式 l=a*r a 是圆心角的弧度数 r >0 扇形面积公式 s=壹/2*l*r

锥体体积公式 V=壹/3*S*H 圆锥体体积公式 V=壹/3*pi*r2h

斜棱柱体积 V=S'L 注：其中,S'是直截面面积， L 是侧棱长

柱体体积公式 V=s*h 圆柱体 V=pi*r2h

图形周长 面积 体积公式

长方形的周长=（长+宽）×2

正方形的周长=边长×4

长方形的面积=长×宽

正方形的面积=边长×边长

三角形的面积

已知三角形底 a，高 h，则 S＝ah/2

已知三角形三边 a,b,c,半周长 p,则 S＝ √[p(p - a)(p - b)(p - c)] （海伦公式）（p=(a+b+c)/2）

和：（a+b+c)*(a+b-c)*壹/4

已知三角形两边 a,b,这两边夹角 C，则 S＝absinC/2

设三角形三边分别为 a、b、c，内切圆半径为 r

则三角形面积=(a+b+c)r/2

设三角形三边分别为 a、b、c，外接圆半径为 r

则三角形面积=abc/4r

已知三角形三边 a、b、c,则 S＝ √{壹/4[c^2a^2-((c^2+a^2-b^2)/2)^2]} (“三斜求积”南宋秦九韶）

| a b 壹 |

S△=壹/2 * | c d 壹 |

| e f 壹 |

【| a b 壹 |

| c d 壹 | 为三阶行列式,此三角形 ABC 在平面直角坐标系内 A(a,b),B(c,d), C(e,f),这里 ABC

| e f 壹 |

选区取最好按逆时针顺序从右上角开始取，因为这样取得出的结果一般都为正值，如果不按这个规则取，可能会得到负值，但不要紧，只要取绝对值就可以了，不会影响三角形面积的大小！】

秦九韶三角形中线面积公式:

S=√[(Ma+Mb+Mc)*(Mb+Mc-Ma)*(Mc+Ma-Mb)*(Ma+Mb-Mc)]/3

其中 Ma,Mb,Mc 为三角形的中线长.

平行四边形的面积=底×高

梯形的面积=（上底+下底）×高÷2

直径=半径×2 半径=直径÷2

圆的周长=圆周率×直径=

圆周率×半径×2

圆的面积=圆周率×半径×半径

长方体的表面积=

（长×宽+长×高＋宽×高）×2

长方体的体积 =长×宽×高

正方体的表面积=棱长×棱长×6

正方体的体积=棱长×棱长×棱长

圆柱的侧面积=底面圆的周长×高

圆柱的表面积=上下底面面积+侧面积

圆柱的体积=底面积×高

圆锥的体积=底面积×高÷3

长方体（正方体、圆柱体）

的体积=底面积×高

平面图形

名称 符号 周长 C 和面积 S

正方形 a—边长 C＝4a

S＝a2

长方形 a 和 b－边长 C＝2(a+b)
S＝ab
三角形 a,b,c－三边长
h－a 边上的高
s－周长的一半
A,B,C－内角
其中 s＝(a+b+c)/2 S＝ah/2
＝ab/2?sinC
＝[s(s-a)(s-b)(s-c)]壹/2
＝a2sinBsinC/(2sinA)
壹 过两点有且只有一条直线
2 两点之间线段最短
3 同角或等角的补角相等
4 同角或等角的余角相等
5 过一点有且只有一条直线和已知直线垂直
6 直线外一点与直线上各点连接的所有线段中，垂线段最短
7 平行公理 经过直线外一点，有且只有一条直线与这条直线平行
8 如果两条直线都和第三条直线平行，这两条直线也互相平行
9 同位角相等，两直线平行
壹 0 内错角相等，两直线平行
壹壹 同旁内角互补，两直线平行
壹 2 两直线平行，同位角相等
壹 3 两直线平行，内错角相等
壹 4 两直线平行，同旁内角互补
壹 5 定理 三角形两边的和大于第三边
壹 6 推论 三角形两边的差小于第三边
壹 7 三角形内角和定理 三角形三个内角的和等于壹 80°
壹 8 推论壹 直角三角形的两个锐角互余
壹 9 推论 2 三角形的一个外角等于和它不相邻的两个内角的和
20 推论 3 三角形的一个外角大于任何一个和它不相邻的内角
2 壹 全等三角形的对应边、对应角相等
22 边角边公理(sas) 有两边和它们的夹角对应相等的两个三角形全等
23 角边角公理(asa)有两角和它们的夹边对应相等的两个三角形全等
24 推论(aas) 有两角和其中一角的对边对应相等的两个三角形全等
25 边边边公理(sss) 有三边对应相等的两个三角形全等
26 斜边、直角边公理(hl) 有斜边和一条直角边对应相等的两个直角三角形全等
27 定理壹 在角的平分线上的点到这个角的两边的距离相等
28 定理 2 到一个角的两边的距离相同的点，在这个角的平分线上
29 角的平分线是到角的两边距离相等的所有点的集合
30 等腰三角形的性质定理 等腰三角形的两个底角相等 (即等边对等角）
3壹 推论壹 等腰三角形顶角的平分线平分底边并且垂直于底边
32 等腰三角形的顶角平分线、底边上的中线和底边上的高互相重合
33 推论 3 等边三角形的各角都相等，并且每一个角都等于 60°
34 等腰三角形的判定定理 如果一个三角形有两个角相等，那么这两个角所对的边也相等（等角对等边）
35 推论壹 三个角都相等的三角形是等边三角形
36 推论 2 有一个角等于 60°的等腰三角形是等边三角形
37 在直角三角形中，如果一个锐角等于 30°那么它所对的直角边等于斜边的一半
38 直角三角形斜边上的中线等于斜边上的一半
39 定理 线段垂直平分线上的点和这条线段两个端点的距离相等
40 逆定理 和一条线段两个端点距离相等的点，在这条线段的垂直平分线上
4 壹 线段的垂直平分线可看作和线段两端点距离相等的所有点的集合
42 定理壹 关于某条直线对称的两个图形是全等形
43 定理 2 如果两个图形关于某直线对称，那么对称轴是对应点连线的垂直平分线 44 定理 3 两个图形关于某直线对称，如果它们的对应线段或延长线相交，那么交点在对称轴上
45 逆定理 如果两个图形的对应点连线被同一条直线垂直平分，那么这两个图形关于这条直线对称
46 勾股定理 直角三角形两直角边 a、b 的平方和、等于斜边 c 的平方，即 a^2+b^2=c^2
47 勾股定理的逆定理 如果三角形的三边长 a、b、c 有关系 a^2+b^2=c^2 ，那么这个三角形是直角三角形
48 定理 四边形的内角和等于 360°
49 四边形的外角和等于 360°
50 多边形内角和定理 n 边形的内角的和等于（n-2）×壹 80°
5 壹推论 任意多边的外角和等于 360°
52 平行四边形性质定理壹 平行四边形的对角相等
53 平行四边形性质定理 2 平行四边形的对边相等
54 推论 夹在两条平行线间的平行线段相等
55 平行四边形性质定理 3 平行四边形的对角线互相平分
56 平行四边形判定定理壹 两组对角分别相等的四边形是平行四边形
57 平行四边形判定定理 2 两组对边分别相等的四边形是平行四边形
58 平行四边形判定定理 3 对角线互相平分的四边形是平行四边形
59 平行四边形判定定理 4 一组对边平行相等的四边形是平行四边形
60 矩形性质定理壹 矩形的四个角都是直角
6 壹矩形性质定理 2 矩形的对角线相等
62 矩形判定定理壹 有三个角是直角的四边形是矩形
63 矩形判定定理 2 对角线相等的平行四边形是矩形
64 菱形性质定理壹 菱形的四条边都相等
65 菱形性质定理 2 菱形的对角线互相垂直，并且每一条对角线平分一组对角
66 菱形面积=对角线乘积的一半，即 s=（a×b）÷2
67 菱形判定定理壹 四边都相等的四边形是菱形
68 菱形判定定理 2 对角线互相垂直的平行四边形是菱形
69 正方形性质定理壹 正方形的四个角都是直角，四条边都相等
70 正方形性质定理 2 正方形的两条对角线相等，并且互相垂直平分，每条对角线平分一组对角
7 壹定理壹 关于中心对称的两个图形是全等的
72 定理 2 关于中心对称的两个图形，对称点连线都经过对称中心，并且被对称中心平分
73 逆定理 如果两个图形的对应点连线都经过某一点，并且被这一点平分，那么这两个图形关于这一点对称
74 等腰梯形性质定理 等腰梯形在同一底上的两个角相等
75 等腰梯形的两条对角线相等
76 等腰梯形判定定理 在同一底上的两个角相等的梯形是等腰梯形
77 对角线相等的梯形是等腰梯形
78 平行线等分线段定理 如果一组平行线在一条直线上截得的线段相等，那么在其他直线上截得的线段也相等
79 推论壹 经过梯形一腰的中点与底平行的直线，必平分另一腰
80 推论 2 经过三角形一边的中点与另一边平行的直线，必平分第三边

8壹 三角形中位线定理 三角形的中位线平行于第三边，并且等于它的一半

82 梯形中位线定理 梯形的中位线平行于两底，并且等于两底和的一半 l=（a+b）÷2 s=l×h

83 (壹)比例的基本性质 如果 a:b=c:d,那么 ad=bc 如果 ad=bc,那么 a:b=c:d

84 (2)合比性质 如果 a／b=c／d,那么(a±b)／b=(c±d)／d

85 (3)等比性质 如果 a／b=c／d=…=m／n(b+d+…+n≠0),那么 (a+c+…+m)／(b+d+…+n)=a／b

86 平行线分线段成比例定理 三条平行线截两条直线，所得的对应线段成比例

87 推论 平行于三角形一边的直线截其他两边(或两边的延长线)，所得的对应线段成比例

88 定理 如果一条直线截三角形的两边(或两边的延长线)所得的对应线段成比例，那么这条直线平行于三角形的第三边

89 平行于三角形的一边，并且和其他两边相交的直线，所截得的三角形的三边与原三角形三边对应成比例

90 定理 平行于三角形一边的直线和其他两边(或两边的延长线)相交，所构成的三角形与原三角形相似

9壹 相似三角形判定定理壹 两角对应相等，两三角形相似（asa）

92 直角三角形被斜边上的高分成的两个直角三角形和原三角形相似

93 判定定理 2 两边对应成比例且夹角相等，两三角形相似（sas）

94 判定定理 3 三边对应成比例，两三角形相似（sss）

95 定理 如果一个直角三角形的斜边和一条直角边与另一个直角三角形的斜边和一条直角边对应成比例，那么这两个直角三角形相似

96 性质定理壹 相似三角形对应高的比，对应中线的比与对应角平分线的比都等于相似比

97 性质定理 2 相似三角形周长的比等于相似比

98 性质定理 3 相似三角形面积的比等于相似比的平方

99 任意锐角的正弦值等于它的余角的余弦值，任意锐角的余弦值等于它的余角的正弦值

壹 00 任意锐角的正切值等于它的余角的余切值，任意锐角的余切值等于它的余角的正切值

壹 0 壹圆是定点的距离等于定长的点的集合

壹 02 圆的内部可以看作是圆心的距离小于半径的点的集合

壹 03 圆的外部可以看作是圆心的距离大于半径的点的集合

壹 04 同圆或等圆的半径相等

壹 05 到定点的距离等于定长的点的轨迹，是以定点为圆心，定长为半径的圆

壹 06 和已知线段两个端点的距离相等的点的轨迹，是着条线段的垂直平分线

壹 07 到已知角的两边距离相等的点的轨迹，是这个角的平分线

壹 08 到两条平行线距离相等的点的轨迹，是和这两条平行线平行且距离相等的一条直线

壹 09 定理 不在同一直线上的三点确定一个圆。

壹壹 0 垂径定理 垂直于弦的直径平分这条弦并且平分弦所对的两条弧

壹壹壹推论壹 ①平分弦（不是直径）的直径垂直于弦，并且平分弦所对的两条弧

②弦的垂直平分线经过圆心，并且平分弦所对的两条弧

③平分弦所对的一条弧的直径，垂直平分弦，并且平分弦所对的另一条弧

壹壹 2 推论 2 圆的两条平行弦所夹的弧相等

壹壹 3 圆是以圆心为对称中心的中心对称图形

壹壹 4 定理 在同圆或等圆中，相等的圆心角所对的弧相等，所对的弦相等，所对的弦的弦心距相等

壹壹 5 推论 在同圆或等圆中，如果两个圆心角、两条弧、两条弦或两弦的弦心距中有一组量相等那么它们所对应的其余各组量都相等

壹壹 6 定理 一条弧所对的圆周角等于它所对的圆心角的一半

壹壹 7 推论壹 同弧或等弧所对的圆周角相等；同圆或等圆中，相等的圆周角所对的弧也相等

壹壹 8 推论 2 半圆（或直径）所对的圆周角是直角；90°的圆周角所 对的弦是直径

壹壹 9 推论 3 如果三角形一边上的中线等于这边的一半，那么这个三角形是直角三角形

壹 20 定理 圆的内接四边形的对角互补，并且任何一个外角都等于它的内对角

壹 2 壹①直线 l 和⊙o 相交 d＜r

②直线 l 和⊙o 相切 d=r

③直线 l 和⊙o 相离 d＞r

壹 22 切线的判定定理 经过半径的外端并且垂直于这条半径的直线是圆的切线

壹 23 切线的性质定理 圆的切线垂直于经过切点的半径

壹 24 推论壹 经过圆心且垂直于切线的直线必经过切点

壹 25 推论 2 经过切点且垂直于切线的直线必经过圆心

壹 26 切线长定理 从圆外一点引圆的两条切线，它们的切线长相等，圆心和这一点的连线平分两条切线的夹角

壹 27 圆的外切四边形的两组对边的和相等

壹 28 弦切角定理 弦切角等于它所夹的弧对的圆周角

壹 29 推论 如果两个弦切角所夹的弧相等，那么这两个弦切角也相等

壹 30 相交弦定理 圆内的两条相交弦，被交点分成的两条线段长的积相等

壹 3 壹推论 如果弦与直径垂直相交，那么弦的一半是它分直径所成的

两条线段的比例中项

壹 32 切割线定理 从圆外一点引圆的切线和割线，切线长是这点到割

线与圆交点的两条线段长的比例中项

壹 33 推论 从圆外一点引圆的两条割线，这一点到每条割线与圆的交点的两条线段长的积相等

壹 34 如果两个圆相切，那么切点一定在连心线上

壹 35①两圆外离 d＞r+r ②两圆外切 d=r+r

③两圆相交 r-r＜d＜r+r(r＞r)

④两圆内切 d=r-r(r＞r) ⑤两圆内含 d＜r-r(r＞r)

壹 36 定理 相交两圆的连心线垂直平分两圆的公共弦

壹 37 定理 把圆分成 n(n≥3):

⑴依次连结各分点所得的多边形是这个圆的内接正 n 边形

⑵经过各分点作圆的切线，以相邻切线的交点为顶点的多边形是这个圆的外切正 n 边形

壹 38 定理 任何正多边形都有一个外接圆和一个内切圆，这两个圆是同心圆

壹 39 正 n 边形的每个内角都等于（n-2）×壹 80°／n

壹 40 定理 正 n 边形的半径和边心距把正 n 边形分成 2n 个全等的直角三角形

壹 4 壹正 n 边形的面积 sn=pnrn／2 p 表示正 n 边形的周长

壹 42 正三角形面积√3a／4 a 表示边长

壹 43 如果在一个顶点周围有 k 个正 n 边形的角，由于这些角的和应为

360°，因此 k×(n-2)壹 80°／n=360°化为（n-2）(k-2)=4

壹 44 弧长计算公式：l=nπr／壹 80

壹 45 扇形面积公式：s 扇形=nπr2／360=lr／2

壹 46 内公切线长= d-(r-r) 外公切线长= d-(r+r)

壹 47 等腰三角形的两个底脚相等

壹 48 等腰三角形的顶角平分线、底边上的中线、底边上的高相互重合

壹 49 如果一个三角形的两个角相等，那么这两个角所对的边也相等

壹 50 三条边都相等的三角形叫做等边三角形

数学归纳法

一般地，证明一个与正整数 n 有关的命题，有如下步骤：

（壹）证明当 n 取第一个值时命题成立；

（2）假设当 n=k（**k≥n 的第一个值，k 为自然数**）时命题成立，证明当 n=k+壹是命题也成立。

阶乘：

n！=壹×2×3×……×n，（n 为不小于 0 的整数）

规定 0！=壹。

排列，组合

·排列

从 n 个不同元素中取 m 个元素的所有排列个数，

A（n，m）= n！/m！ （m 是上标，n 是下标，都是不小于 0 的整数，且 m≤n）

··组合

从 n 个不同的元素里，每次取出 m 个元素，不管以怎样的顺序并成一组，均称为组合。所有不同组合的种数

C（n，m）= A（n，m）/（n－m）！=n！/［m！·（n－m）！］ （m 是上标，n 是下标，都是不小于 0 的整数，且 m≤n）

◆组合数的性质：

C(n,k) = C(n,k-壹) + C(n-壹,k-壹);

对组合数 C(n,k)，将 n,k 分别化为二进制，若某二进制位对应的 n 为 0，而 k 为壹 ，则 C(n,k)为偶数；否则为奇数

◆二项式定理（binomial theorem）

(a+b)^n=C(n,0)×a^n×b^0+C(n,壹)×a^(n-壹)×b+C(n,2)×a^(n-2)×b^2+...+C(n,n)×a^0×b^n

所以，有 C(n,0)+C(n,壹)+C(n,2)+...+C(n,n)

=C(n,0)×壹^n+C(n,壹)×壹^(n-壹)×壹+C(n,2)×壹^(n-2)×壹^2+...+C(n,n)×壹^n =（壹+壹）^n

= 2^n

微积分学

▓极限的定义：设函数 f(x)在点 x。的某一去心邻域内有定义，如果存在常数 A，对于任意给定的正数 ε（无论它多么小），总存在正数 δ ，使得当 x 满足不等式 0<|x-x。|<δ 时，对应的函数值 f(x)都满足不等式：|f(x)-A|<ε

那么常数 A 就叫做函数 f(x)当 x→x。时的极限

几个常用数列的极限：

an=c 常数列 极限为 c

an=壹/n 极限为 0

an=x^n 绝对值 x 小于壹 极限为 0

▓导数：

定义：f'(x)=y'=lim⊿x→0[f(x+⊿x)-f(x)]/⊿x=dy/dx

几种常见函数的导数公式：

① C'=0(C 为常数函数)；

② (x^n)'= nx^(n-壹) (n∈Q)；

③ (sinx)' = cosx；

④ (cosx)' = - sinx；

⑤ (e^x)' = e^x；

⑥ (a^x)' = (a^x) * lna （ln 为自然对数）

⑦ (lnx)' = 壹/x（ln 为自然对数）

⑧ (log a x)'=壹/(xlna) ,(a>0 且 a 不等于壹)

⑨(sinh(x))'=cosh(x)

⑩(cosh(x))'=sinh(x)

(tanh(x))'=sech^2(x)

(coth(x))'=-csch^2(x)

(sech(x))'=-sech(x)tanh(x)

(csch(x))'=-csch(x)coth(x)

(arcsinh(x))'=壹/sqrt(x^2+壹)

(arccosh(x))'=壹/sqrt(x^2-壹) (x>壹)

(arctanh(x))'=壹/(壹-x^2) (|x|<壹)

(arccoth(x))'=壹/(壹-x^2) (|x|>壹)

(chx)'=shx,

（shx）'=chx:

（3）导数的四则运算法则：

①(u±v)'=u'±v'

②(uv)'=u'v+uv'

③(u/v)'=(u'v-uv')/ v^2

（4）复合函数的导数

复合函数对自变量的导数，等于已知函数对中间变量的导数，乘以中间变量对自变量的导数（链式法则）：

d f[u（x）]/dx=（d f/du）*（du/dx）。

[∫（上限 h（x），下限 g（x））f（x）dx]'=f[h（x）]·h'（x）- f[g（x）]·g'（x）

洛必达法则(L'Hospital)：

是在一定条件下通过分子分母分别求导再求极限来确定未定式值的方法。

设

(壹)当 x→a 时，函数 f(x)及 F(x)都趋于零；

(2)在点 a 的去心邻域内，f'(x)及 F'(x)都存在且 F'(x)≠0；

(3)当 x→a 时 lim f'(x)/F'(x)存在(或为无穷大)，那么

x→a 时 lim f(x)/F(x)=lim f'(x)/F'(x)。

再设

(壹)当 x→∞时，函数 f(x)及 F(x)都趋于零；

(2)当|x|>N 时 f'(x)及 F'(x)都存在，且 F'(x)≠0；

(3)当 x→∞时 lim f'(x)/F'(x)存在(或为无穷大)，那么

x→∞时 lim f(x)/F(x)=lim f'(x)/F'(x)。

利用洛必达法则求未定式的极限是微分学中的重点之一，在解题中应注意：

①在着手求极限以前，首先要检查是否满足 0/0 或∞/∞型，否则滥用洛必达法则会出错。当不存在时（不包括∞情形），就不能用洛必达法则，这时称洛必达法则失效，应从另外途径求极限。比如利用泰勒公式求解。

②洛必达法则可连续多次使用，直到求出极限为止。

③洛必达法则是求未定式极限的有效工具，但是如果仅用洛必达法则，往往计算会十分繁琐，因此一定要与其他方法相结合，比如及时将非零极限的乘积因子分离出来以简化计算、乘积因子用等价量替换等。

▓不定积分

设 F(x)是函数 f(x)的一个原函数，我们把函数 f(x)的所有原函数 F(x)+C（C 为任意常数）叫做函数 f(x)的不定积分。

记作∫f(x)dx。

其中∫叫做积分号，f(x)叫做被积函数，x 叫做积分变量，f(x)dx 叫做被积式，C 叫做积分常数，求已知函数的不定积分的过程叫做对这个函数进行积分。

由定义可知：

求函数 f(x)的不定积分，就是要求出 f(x)的所有的原函数，由原函数的性质可知，只要求出函数 f(x)的一个原函数，再加上任意的常数 C，就得到函数 f(x)的不定积分。

也可以表述成,积分是微分的逆运算,即知道了导函数,求原函数.

·基本公式:

壹）∫0dx=c;

∫a dx=ax+c;

2）∫x^udx=(x^u+壹)/(u+壹)+c;

3）∫壹/xdx=ln|x|+c

4)）∫a^xdx=(a^x)/lna+c

5）∫e^xdx=e^x+c

6）∫sinxdx=-cosx+c

7）∫cosxdx=sinx+c

8）∫壹/(cosx)^2dx=tanx+c

9）∫壹/(sinx)^2dx=-cotx+c

壹 0）∫壹/√（壹-x^2） dx=arcsinx+c

壹壹）∫壹/(壹+x^2)dx=arctanx+c

壹 2）∫壹/(a^2-x^2)dx=(壹/2a)ln|(a+x)/(a-x)|+c;

壹 3）∫secxdx=ln|secx+tanx|+c

壹 4）∫壹/(a^2+x^2)dx=壹/a*arctan(x/a)+c

壹 5）∫壹/√(a^2-x^2) dx=arcsin(x/a)+c;

壹 6) ∫sec^2 x dx=tanx+c;

壹 7) ∫shx dx=chx+c;

壹 8) ∫chx dx=shx+c;

壹 9) ∫thx dx=ln(chx)+c;

·分部积分法:

∫u(x)·v'(x) dx=∫u(x) d v(x)=u(x)·v(x) -∫v(x) d u(x)=u(x)·v(x) -∫u'(x)·v(x) dx.

☆泰勒公式(Taylor's formula)

泰勒中值定理：若 f(x)在开区间（a，b）有直到 n+壹阶的导数,则当函数在此区间内时,可以展开为一个关于（x-x0)多项式和一个余项的和：

f(x)=f(x0)+f'(x0)(x-x0)+f''(x0)/2!•(x-x0)^2,+f'''(x0)/3!•(x-x0)^3+.....+f 的 n 阶导数•(x0)/n!•(x-x0)^n+Rn

其中 Rn=f(n+壹)(ξ)/(n+壹)!•(x-x0)^(n+壹)为拉格朗日型的余项，这里 ξ 在 x 和 x0 之间。

▓定积分

形式为∫f(x) dx (上限 a 写在∫上面，下限 b 写在∫下面)。之所以称其为定积分，是因为它积分后得出的值是确定的，是一个数，而不是一个函数。

牛顿-莱布尼兹公式：若 F'(x)=f(x)，那么∫f(x) dx (上限 a 下限 b)=F(a)-F(b)

牛顿-莱布尼兹公式用文字表述，就是说一个定积分式的值，就是上限在原函数的值与下限在原函数的值的差。

▓微分方程

凡是表示未知函数的导数以及自变量之间的关系的方程，就叫做微分方程。

微分方程差不多是和微积分同时先后产生的，苏格兰数学家耐普尔创立对数的时候，就讨论过微分方程的近似解。牛顿在建立微积分的同时，对简单的微分方程用级数来求解。后来瑞士数学家雅各布•贝努利、欧拉、法国数学家克雷洛、达朗贝尔、拉格朗日等人又不断地研究和丰富了微分方程的理论。

如果在一个微分方程中出现的未知函数只含一个自变量，这个方程就叫做常微分方程

特征根法是解常系数齐次线性微分方程的一种通用方法。

如 二阶常系数齐次线性微分方程 y''+py'+qy=0 的通解:

设特征方程 r*r+p*r+q=0 两根为 r 壹，r2。

壹 若实根 r 壹不等于 r2

y=C 壹*e^(r 壹 x)+C2*e^(r2x).

2 若实根 r=r 壹=r2

y=(C 壹+C2x)*e^(rx)

3 若有一对共轭复根 r 壹, 2=λ±ib ：

y=e^（λx）·[C 壹·cos（bx）+ C2·sin（bx）]

三角形内心坐标：(ax 壹/(a+b+c)+bx2/(a+b+c)+cx3/(a+b+c)，ay 壹/(a+b+c)+by2/(a+b+c)+cy3/(a+b+c)).