# 中山大學程式競賽程式庫(Library)　楊昌彪　　2021/9/29

***僅供中山大學學生參加程式設計競賽使用，請勿外流

## C Library

#include 兩個都可以

看到可能想用的 function 可是不知道參數，就打開終端機　用 man 查詢　例：　man printf

| stdio.h | cstdio | | |
|---|---|---|---|
| remove | Remove file | fputc | Write character to stream |
| rename | Rename file | fputs | Write string to stream |
| tmpfile | Open a temporary file | getc | Get character from stream |
| tmpnam | Generate temporary filename | getchar | Get character from stdin |
| fclose | Close file | gets | Get string from stdin |
| fflush | Flush stream | putc | Write character to stream |
| fopen | Open file | putchar | Write character to stdout |
| freopen | Reopen stream with different file or mode | puts | Write string to stdout |
| setbuf | Set stream buffer | ungetc | Unget character from stream |
| setvbuf | Change stream buffering | fread | Read block of data from stream |
| fprintf | Write formatted output to stream | fwrite | Write block of data to stream |
| fscanf | Read formatted data from stream | fgetpos | Get current position in stream |
| printf | Print formatted data to stdout | fseek | Reposition stream position indicator |
| scanf | Read formatted data from stdin | fsetpos | Set position indicator of stream |
| sprintf | Write formatted data to string | ftell | Get current position in stream |
| sscanf | Read formatted data from string | rewind | Set position indicator to the beginning |
| vfprintf | Write formatted variable argument list to stream | clearerr | Clear error indicators |
| vprintf | Print formatted variable argument list to stdout | feof | Check End-of-File indicator |
| vsprintf | Print formatted variable argument list to string | ferror | Check error indicator |
| fgetc | Get character from stream | perror | Print error message |
| fgets | Get string from stream | | |

| ctype.h | cctype | | |
|---|---|---|---|
| isalnum | Check if character is alphanumeric | ispunct | Check if character is a punctuation character |
| isalpha | Check if character is alphabetic | isspace | Check if character is a white-space |
| iscntrl | Check if character is a control character | isupper | Check if character is uppercase letter |
| isdigit | Check if character is decimal digit | isxdigit | Check if character is hexadecimal digit |
| isgraph | Check if character has graphical representation | tolower | Convert uppercase letter to lowercase |
| islower | Check if character is lowercase letter | toupper | Convert lowercase letter to uppercase |
| isprint | Check if character is printable | | |

| stdlib.h | cstdlib | | |
|---|---|---|---|
| atof | Convert string to double | exit | Terminate calling process |
| atoi | Convert string to integer | getenv | Get environment string |
| atol | Convert string to long integer | system | Execute system command |
| strtod | Convert string to double | bsearch | Binary search in array |
| strtol | Convert string to long integer | qsort | Sort elements of array |
| strtoul | Convert string to unsigned long integer | abs | Absolute value |
| rand | Generate random number | div | Integral division |
| srand | Initialize random number generator | labs | Absolute value |
| calloc | Allocate space for array in memory | ldiv | Integral division |
| free | Deallocate space in memory | mblen | Get length of multibyte character |
| malloc | Allocate memory block | mbtowc | Convert multibyte character to wide character |
| realloc | Reallocate memory block | wctomb | Convert wide character to multibyte character |
| abort | Abort current process | mbstowcs | Convert multibyte string to wide-character string |
| atexit | Set function to be executed on exit | wcstombs | Convert wide-character string to multibyte string |

| string.h | cstring | | |
|---|---|---|---|
| memcpy | Copy block of memory | memchr | Locate character in block of memory |
| memmove | Move block of memory | strchr | Locate first occurrence of character in string |
| strcpy | Copy string | strcspn | Get span until character in string |
| strncpy | Copy characters from string | strpbrk | Locate character in string |
| strcat | Concatenate strings | strrchr | Locate last occurrence of character in string |
| strncat | Append characters from string | strspn | Get span of character set in string |
| memcmp | Compare two blocks of memory | strstr | Locate substring |
| strcmp | Compare two strings | strtok | Split string into tokens |
| strcoll | Compare two strings using locale | memset | Fill block of memory |
| strncmp | Compare characters of two strings | strerror | Get pointer to error message string |
| strxfrm | Transform string using locale | strlen | Get string length |

| time.h | ctime | | |
|---|---|---|---|
| clock | Clock program | ctime | Convert time_t value to string |
| difftime | Return difference between two times | gmtime | Convert time_t to tm as UTC time |
| mktime | Convert tm structure to time_t | localtime | Convert time_t to tm as local time |
| time | Get current time | strftime | Format time to string |
| asctime | Convert tm structure to string | | |

| math.h | cmath | | |
|---|---|---|---|
| cos | Compute cosine | frexp | Get significand and exponent |
| sin | Compute sine | ldexp | Generate number from significand and exponent |
| tan | Compute tangent | log | Compute natural logarithm |
| acos | Compute arc cosine | log10 | Compute common logarithm |
| asin | Compute arc sine | modf | Break into fractional and integral parts |
| atan | Compute arc tangent | pow | Raise to power |
| atan2 | Compute arc tangent with two parameters | sqrt | Compute square root |
| cosh | Compute hyperbolic cosine | ceil | Round up value |
| sinh | Compute hyperbolic sine | fabs | Compute absolute value |
| tanh | Compute hyperbolic tangent | floor | Round down value |
| exp | Compute exponential function | fmod | Compute remainder of division |

## C++ Library(只列出可能使用的)

| algorithm | |
|---|---|
| for_each | Apply function to range |
| find | Find value in range |
| count | Count appearances of value in range |
| equal | Test whether the elements in two ranges are equal |
| search | Find subsequence in range |
| copy | Copy range of elements |
| swap | Exchange values of two objects |
| replace | Replace value in range |
| remove | Remove value from range |
| binary_search | Test if value exists in sorted array |
| reverse | Reverse range |
| sort | Sort elements in range |
| stable_sort | Sort elements preserving order of equivalents |
| min_element | Return smallest element in range |
| max_element | Return largest element in range |
| next_permutation | Transform range to next permutation |
| prev_permutation | Transform range to previous permutation |

| vector | |
|---|---|
| begin | Return iterator to beginning |
| end | Return iterator to end |
| size | Return size |
| resize | Change size |
| empty | Test whether vector is empty |
| front | Access first element |
| back | Access last element |
| push_back | Add element at the end |
| pop_back | Delete last element |
| insert | Insert elements |
| erase | Erase elements |
| swap | Swap content |

| | |
|---|---|
| clear | Clear content |
| **Deque** (Same as vector) | |
| push_front | Insert element at beginning |
| pop_front | Delete first element |
| stack/priority_queue | |

| **queue** | |
|---|---|
| empty | Test whether container is empty |
| size | Return size |
| front | Access next element |
| back | Access last element |
| push | Insert element |
| back | Delete next element |

| **string** (NOT string.h) | |
|---|---|
| begin | Return iterator to beginning |
| end | Return iterator to end |
| length | Return length of string |
| resize | Resize string |
| clear | Clear string |
| empty | Test if string is empty |
| at | Get character in string |

| Stream manipulators (iostream/iomanip) | |
|---|---|
| dec | Use decimal base |
| hex | Use hexadecimal base |
| oct | Use octal base |
| fixed | Use fixed-point notation |
| scientific | Use scientific notation |

| | |
|---|---|
| empty | Test whether container is empty |
| size | Return size |
| top | Access next element |
| Push | Add element |
| pop | Remove element |

| **map** | |
|---|---|
| begin | Return iterator to beginning |
| end | Return iterator to end |
| empty | Test whether container is empty |
| size | Return container size |
| insert | Insert elements |
| erase | Erase elements |
| swap | Swap content |
| clear | Clear content |
| find | Get iterator to element |
| count | Count elements with a specific key |

| **string** (not string.h) | |
|---|---|
| append | Append to string |
| erase | Erase characters from string |
| replace | Replace part of string |
| c_str | Get C string equivalent |
| find | Find content in string |
| substr | Generate substring |
| compare | Compare strings |
| String::npos | npos indicates the end of the string |

| | |
|---|---|
| left | Adjust output to the left |
| right | Adjust output to the right |
| endl | Insert newline and flush |
| ends | Insert null character |
| setfill | Set fill character |
| setprecision | Set decimal precision |
| setw | Set field width |

## 檔案讀寫

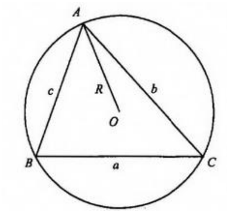
### 無腦版 （將螢幕鍵盤的 IO 改為從檔案 IO）

```
freopen("file","r",stdin);
freopen("file","w",stdout);
```

### 不定量輸入

```
string tmp;
getline(cin,tmp);
istringstream cin2(tmp) ;
while(cin2 >> data){ … }
```

### 三角學



- $\cos^2\theta+\sin^2\theta=1$
- $\sin(\alpha+\beta)=\sin\alpha \cos\beta+ \cos\alpha \sin\beta$
- $\sin(\alpha-\beta)=\sin\alpha \cos\beta- \cos\alpha \sin\beta$
- $\cos(\alpha+\beta)=\cos\alpha \cos\beta- \sin\alpha \sin\beta$
- $\cos(\alpha-\beta)=\cos\alpha \cos\beta + \sin\alpha \sin\beta$
- $\sin2\alpha=2\sin\alpha \cos\alpha$
- $\cos2\alpha = \cos^2\alpha - \sin^2\alpha$
- $a^2+b^2=c^2 - 2bc\cos A$
- $a/\sin A= b/\sin B =c/\sin C =2R$
- 三角形三邊長為 a,b,c，面積=sqrt(p(p-a)(p-b)(p-c))=(abc)/(4R), where p=(a+b+c)/2
- 點(x', y')至直線 Ax+By+C=0 之距離 =|Ax'+By'+C|/sqrt($A^2+B^2$)

### 求和公式

$$\sum_{i=1}^{n} i^2 = \frac{1}{6}n(n+1)(2n+1)$$

$$\sum_{i=1}^{n} i^3 = \frac{1}{4}n^2(n+1)^2 = \left(\sum_{i=1}^{n} i\right)^2$$

$$\sum_{i=1}^{n} i^4 = \frac{1}{30}n(n+1)(2n+1)(3n^2+3n-1)$$

$$\sum_{i=1}^{n} i^5 = \frac{1}{12}n^2(n+1)^2(2n^2+2n-1)$$

$$\sum_{i=1}^{n} \frac{1}{i(i+1)(i+2)} = \frac{1}{4} - \frac{1}{2(n+1)(n+2)}$$

$$\sum_{i=1}^{n} \frac{1}{i(i+1)(i+2)(i+3)} = \frac{1}{18} - \frac{1}{3(n+1)(n+2)(n+3)}$$

### Math Functions

//歐拉函數: 從 1~n-1 與 n 互質的數的個數

```
int eular(int n)
{
        int ret=1, i;
        for (i=2; i*i<=n; i++)
                if (n%i==0) {
                        n/=i,ret*=i-1;
                        while (n%i==0) { n/=i; ret*=i; }
                }
        if (n>1) ret*=n-1;
        return ret;
}
void getphi()
{
        memset(phi, 0, sizeof (phi));    //phi 記錄歐拉函數
        phi[1] = 1;
        for (int i = 2; i < N; ++i) {
                if (!phi[i]) {
                        for (int j = i; j < N; j += i) {
                                if (!phi[j]) phi[j] = j;
                                phi[j] = phi[j] / i * (i - 1);
                        }
                }
        }
}
```

//卡塔蘭數 (1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, ⋯)

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

h(n)=h(n-1)*(4*n-2)/(n+1)

卡塔蘭數可能問題如下：1.括號方法數 2.stack 序列數 3.多邊形劃分三角形方法數 4.n 個頂點二分樹的組成數

//原根

g^i mod p ≠g^j mod p (p 為素數,i≠j,1≤i,j≤p-1)，則 g 為 p 的原根。

//第一類 Stirling

n 個人分成 k 組，每組內再按特定順序圍圈的分組方法的數目。

給定 s(n,0)=0，s(1,1)=1 ，遞迴關係 s(n+1,k)=s(n,k-1)+ns(n,k)

//第二類 Stirling

n 個人分成 k 組的分組方法的數目。

給定 S(n,n)=S(n,1)=1 ，遞迴關係 S(n,k)=S(n-1,k-1)+kS(n-1,k)

//Bell number (1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, …)

n 個元素可被分成非空子集合的劃分個數。

1
1 2
2 3 5
5 7 10 15
15 20 27 37 52
…

```
int bell[MAXN], t[2][MAXN];
void bellNum()
{
        int i, j;
        t[0][0]=bell[0]=bell[1]=1;
        for(i=1; i<MAXN; ++i) {
                t[i&1][0]=t[(i-1)&1][i-1];
                for(j=1; j<=i; ++j) t[i&1][j]=t[i&1][j-1]+t[(i-1)&1][j-1];
                bell[i+1]=t[i&1][i];
        }
}
```

Euler equation: V-E+F=2
Pick theorem: A=i+b/2-1

```
//Pell's equation  佩爾方程
  x^2-ny^2=1，n 若為完全平方數，解為(±1,0)
void Pell(int n, int &x, int &y)    //x,y 为最小正整数解
{
        y=1;
        while(1) {
                x=sqrt(n*y*y+1);
                if(x*x-n*y*y==1)break;
                y++;
        }
}
```

兩圓相交求交點

兩圓方程:

$$(x-x_1)^2+(y-y_1)^2=r_1^2$$
$$(x-x_2)^2+(y-y_2)^2=r_2^2$$

兩圓交點:

$$x = \frac{x_2+x_1}{2}+\frac{(x_2-x_1)(r_1^2-r_2^2)}{2d^2} \pm \frac{y_2-y_1}{2d^2}\sqrt{((r_1+r_2)^2-d^2)(d^2-(r_1-r_2)^2)}$$

$$y = \frac{y_2+y_1}{2}+\frac{(y_2-y_1)(r_1^2-r_2^2)}{2d^2} \mp \frac{x_2-x_1}{2d^2}\sqrt{((r_1+r_2)^2-d^2)(d^2-(r_1-r_2)^2)}$$

$$d=\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$$

## Map 範例(計算每個字出現的次數)

```
map<string,int> stringCounts;
string str;
while (cin >> str) stringCounts[str]++;

map<string,int>::iterator iter;
for( iter = stringCounts.begin(); iter != stringCounts.end();
iter++ ) {
    cout << iter->first << "=" << iter->second << endl;
}
/* Input: here are some words and here are some more
words
Output:
and=1
are=2
here=2
more=1
some=2
words=2
*/
```

## Set 範例(重複出現者，僅計算一次)

```
set<string> s;
s.insert("xyz");
s.insert("def");
s.insert("abc");
s.insert("bbb");
s.insert("bbb");

set<string>::iterator iter;
for( iter = s.begin(); iter != s.end(); iter++ ) {
    cout << *iter << ", ";
}
cout << endl << s.count("aaa") << endl;
cout << s.count("bbb") << endl;
//Outputs
abc, bbb, def, xyz,
0
1
```

## Algorithm 內的 sort 範例

```
bool myfunction (int i,int j) { return (i<j); }
struct myclass {
    bool operator() (int i,int j) { return (i<j);}
} myobject;
int main () {
  int myints[] = {32,71,12,45,26,80,53,33};
  vector<int> myvector (myints, myints+8);
                  // 32 71 12 45 26 80 53 33
  vector<int>::iterator it;
    // using default comparison (operator <):
  sort (myvector.begin(), myvector.begin()+4);
      //(12 32 45 71)26 80 53 33
  // using function as comp
  sort(myvector.begin()+4, myvector.end(), myfunction);
    // 12 32 45 71(26 33 53 80)
    // using object as comp
  sort (myvector.begin(), myvector.end(), myobject);
        //(12 26 32 33 45 53 71 80)
return 0;
}
```

## Algorithm 內的 binary_search 範例

```
bool myfunction (int i,int j) { return (i<j); }
int main () {
    int myints[] = {1,2,3,4,5,4,3,2,1};
    vector<int> v(myints,myints+9);      // 1 2 3 4 5 4 3 2 1
    // using default comparison:
    sort (v.begin(), v.end());
    cout << "looking for a 3... ";
    if (binary_search (v.begin(), v.end(), 3))
        cout << "found!\n"; else cout << "not found.\n";
    // using myfunction as comp:
    sort (v.begin(), v.end(), myfunction);
    cout << "looking for a 6... ";
```

```
   if (binary_search (v.begin(), v.end(), 6, myfunction))
      cout << "found!\n"; else cout << "not found.\n";
   return 0;
}
```

## Algorithm 內的 search 範例

```
bool mypredicate (int i, int j) {return (i==j);}
int main () {
   vector<int> myvector;
   vector<int>::iterator it;
   for (int i=1; i<10; i++) myvector.push_back(i*10);
//myvector:10 20 30 40 50 60 70 80 90
   // using default comparison:
   int match1[] = {40,50,60,70};
   it = search (myvector.begin(), myvector.end(), match1,
match1+4);
   if (it!=myvector.end())
      cout << "match1 found at position " << int(it-
         myvector.begin()) << endl;
   else
      cout << "match1 not found" << endl;
   // using predicate comparison:
   int match2[] = {20,30,50};
   it = search (myvector.begin(), myvector.end(), match2,
      match2+3, mypredicate);
   if (it!=myvector.end())
      cout << "match2 found at position " << int(it-
        myvector.begin()) << endl;
   else
      cout << "match2 not found" << endl;
   return 0;
}
```

## Algorithm 內的 next_permutation 範例

```
int myints[] = {1,2,3};
sort (myints,myints+3);
do {
     cout << myints[0] << " " << myints[1] << " " << myints[2]
<< endl;
} while ( next_permutation (myints,myints+3) );
//OUTPUT:
1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1
```

## Algorithm 內 lower_bound/upper_bound 範例

```
int main () {
   int myints[] = {10,20,30,30,20,10,10,20};
   vector<int> v(myints,myints+8);
        // 10 20 30 30 20 10 10 20
   vector<int>::iterator low,up;

   sort (v.begin(), v.end());           // 10 10 10 20 20 20 30 30
   low=lower_bound (v.begin(), v.end(), 20);
   up= upper_bound (v.begin(), v.end(), 20);
   cout << "lower_bound at position " << int(low- v.begin())
<< endl;//3
   cout << "upper_bound at position " << int(up - v.begin())
<< endl;//3
   return 0;
}
```

## String 內 find 與 substr 範例

```
//substr
string str="We think in generalities, but we live in details.";
string str2, str3;
size_t pos;
str2 = str.substr (12,12); // "generalities"
pos = str.find("live");        // position of "live" in str
str3 = str.substr (pos);       // get from "live" to the end
//find
string str ("There are two needles in this haystack with
needles.");
string str2 ("needle");
size_t found;
found=str.find(str2);
if (found!=string::npos)
   cout << "first 'needle' found at: " << int(found) << endl;
                 //14
found=str.find("needles are small",found+1,6);
if (found!=string::npos)
   cout << "second 'needle' found at: " << int(found) << endl;
                 //44
found=str.find("haystack");
if (found!=string::npos)
   cout << "'haystack' also found at: " << int(found) << endl;
                 //30
found=str.find('.');
if (found!=string::npos)
   cout << "Period found at: " << int(found) << endl;
                             //51
// let's replace the first needle:
str.replace(str.find(str2),str2.length(),"preposition");
cout << str << endl;
 //There are two prepositions in this haystack with needles.
```

## Segment Tree

## Construction O(n)      Range min O(log n)

```
int cc [1 << 22] ,m, n ; // memset cc f i r s t
void update ( int ii , int s , int t , int ss , int tt , bool insert ) {
     if ( ss>tt ) return ; int mid ( ( s+t ) / 2 ) ;
     if ( s==ss && t==tt ) { if ( insert ) cc [ ii ]=t-s +1; else cc
[ ii ]=0 ; return ; }
     if ( cc [ ii ]==0) if ( ! insert ) return ; else cc [ ii*2]=cc
[ ii*2+1]=0;
     else if ( cc [ ii ]==t-s+1) if ( insert ) return ;
     else { cc [ ii*2]=mid-s +1; cc [ ii*2+1]=t-mid ; }
     update ( ii*2 , s ,mid , ss , min (mid , tt ) , insert ) ;
     update ( ii*2+1 ,mid+1, t , max (mid+1, ss ) , tt , insert ) ;
     cc [ ii ]=cc [ ii*2]+cc [ ii*2+1] ;
}
int query ( int ii , int s , int t , int ss , int tt ) {
     if ( ss>tt ) return 0 ; int mid ( ( s+t ) / 2 ) ;
     if ( s==ss && t==tt ) return cc [ ii ] ;
     if ( cc [ ii ]==0) cc [ ii* 2 ] = cc [ ii*2+1] = 0;
```

```
    if ( cc [ ii ]==t-s+1) { cc [ ii*2]=mid-s +1; cc [ ii*2+1]=t-
mid ; }
    return query ( ii*2 , s ,mid , ss , min (mid , tt ) )+query
( ii*2+1 ,mid+1, t , max (mid+1, ss ) , tt ) ;
}
```

## Union-Find in Set

```
int rank [maxn ] , pnt [maxn ] ;
void makeset ( int x )
{ rank[pnt[x]=x] = 0 ; }
int find( int x )
{
    int px=x , i ;
    while ( px!=pnt[px]) px=pnt[px] ;
    while ( x!=px ) { i=pnt[x] ; pnt[x]=px ; x=i ; } ;
    return px ;
}
void merge ( int x , int y ) // or just pnt[find(X)]= find(y)
{
    if ( rank[x=finf(x)] > rank[y=find(y)]) pnt[y]=x ;
    else { pnt[x]=y ; rank[y]+=( rank[x]==rank[y] ) ; } ;
}
```

## Union-Find Set

```
//2014/10/13 提供者：翁丞世、林敬哲、林必祥
int s[N];
/* find the set with path compression - O(logN) */
int findSet( int p )
{
     if( s[p] < 0 ) return p;
     return s[p] = setFind(s[p]);
}

/* merge two sets - O(1) */
void unionSet( int p, int q )
{
     p = setFind(p);
     q = setFind(q);
     if( p != q ) s[p] = q;
}
```

## Select kth smallest element

```
int select( int *a , int b , int e , int k )
{
    if ( b==e ) return a[b] ;
    int x = a[ b+rand()%( e-b+1 )] , i = b , j = e ;
    i--; j++;
    while ( i<j ) {
        while ( a[++i ] < x ) ; while (a[--j] > x) ;
        if (i<j ) std:: swap(a[i],a[j]) ;
    }
    if (j==e ) j--; i = j-b+1;
    if ( k <= i ) return select( a , b , j , k ) ;
    else return select( a , j +1 , e , k-i ) ;
}
```

## KMP String Matching O(m+n)

```
// |X|=m: pattern,   |Y|=n: text, search substring X in Y
//先幫 X 建立 prefix function (kmpNext)，適用於只搜尋一
次的情形
void preKmp(char *x, int m, int kmpNext[]) {
    int i, j;
    i = 0;
    j = kmpNext[0] = -1;
    while (i < m) {
        while (j > -1 && x[i] != x[j])
            j = kmpNext[j];
        i++;
        j++;
        if (x[i] == x[j])
            kmpNext[i] = kmpNext[j];
        else
            kmpNext[i] = j;
    }
}

void KMP(char *x, int m, char *y, int n) {
    int i, j, kmpNext[XSIZE];
    /* Preprocessing */
    preKmp(x, m, kmpNext);
    /* Searching */
    i = j = 0;
    while (j < n) {
        while (i > -1 && x[i] != y[j])
            i = kmpNext[i];
        i++;
        j++;
        if (i >= m) {
            OUTPUT(j - i);
            i = kmpNext[i];
        }
    }
}
```

## KMP String Matching O(m+n)

```
//2014/10/13 提供者：翁丞世、林敬哲、林必祥
/* use kmp algorithm to get the fail array */
#define N 100005
int fail[N];
void kmp( char *s )
{
   static int i, k, l;
   l = strlen( s );
   fail[0] = -1;     fail[1] = 0;
   k = 0;
   for( i = 2; i <= l; ++i ) {
      while( k >= 0 && s[i-1] != s[k] ) k = fail[k];
      fail[i] = ++k;
   }
}
int main( void )
{
   int i, wl,tl , j;
   scanf( "%s%s", w, t );
   kmp(w);
   wl = strlen(w);     tl = strlen(t);
   for( i = j = 0; i < tl; ) {
      if( j == -1 || t[i] == w[j] ) {
            ++j; ++i;
        } else {    j = fail[j]; }
        if( j == wl ) break;
    }
    return 0;
}
```

## Suffix Array

```
//2014/10/13  提供者：翁丞世、林敬哲、林必祥
/* use doubling algorithm(DA) to get the suffix array that
contains the sorted suffixes of the strings      */
#define N 200010
char s[N], ss[N];
int sa[N], h[N], rk[N];
int n, mid;
const bool cmp( const int &a, const int &b )
{
    return ( s[a]<s[b] );
}
void suffix( void )
{
    static int i, h, j, k;
    static int rk2[N], head[N], next[N];
    for( i = 0; i < n; ++i ) sa[i] = i;
    sort( sa, sa+n, cmp );
    rk[sa[0]] = 0;
    for( i = 1; i < n; ++i ) {
        if( s[sa[i]] == s[sa[i-1]] ) rk[sa[i]] = rk[sa[i-1]];
        else rk[sa[i]] = i;
    }
    for( h = 1; h < n; h <<= 1 ) {
        for( i = 0; i < n; ++i ) head[i] = next[i] = -1;
        for( i = n-1; i >= 0; --i ) if( sa[i] ) {
            j = sa[i]-h; if( j < 0 ) j += n;
            next[j] = head[rk[j]]; head[rk[j]] = j;
        }
        j = n-h;
        next[j] = head[rk[j]]; head[rk[j]] = j;
        for( i = k = 0; i < n; ++i ) if( head[i] >= 0 )
            for( j = head[i]; j >= 0; j = next[j] ) sa[k++] = j;
        rk2[sa[0]] = 0;
        for( i = 1; i < n; ++i ) {
            if( sa[i]+h < n && sa[i-1]+h < n
&& rk[sa[i]] == rk[sa[i-1]] && rk[sa[i]+h] == rk[sa[i-1]+h] )
rk2[sa[i]] = rk2[sa[i-1]];
            else rk2[sa[i]] = i;
        }
        memcpy( rk, rk2, sizeof(int)*n );
    }
}

void geth( void )
{
    static int i, j, k;
    h[0] = 0;
    for( i = k = 0; i < n; ++i ) if( rk[i] ) {
        j = sa[ rk[i]-1 ];
        while( s[i+k] == s[j+k] ) ++k;
        h[ rk[i] ] = k;
        if( k > 0 ) --k;
```

```
    }
}
```

## GCD&LCM(最大公約數、最小公倍數)

```
//2014/10/13  提供者：翁丞世、林敬哲、林必祥
/* return the greatest common divisor of a and b */
const int gcd( int a, int b )
{
    static int t;
    while( b ) {
        t = a%b; a = b; b = t;    }
    return a;
}

/* return the least common multiple of a and b */
const int lcm( int a, int b )
{
    return (a*b)/gcd(a,b);
}
```

## Extended Euclidean(輾轉相除法)

```
//2014/10/13  提供者：翁丞世、林敬哲、林必祥
/* return the greatest common divisor of a and b with
a*dx+b*dy = gcd(a,b)                                   */
const int extendedEuclidean( const int a, const int b, int &dx,
int &dy )
{
    if( !b ) {
        dx = 1;
        dy = 0;
        return a;
    } else {
        int tmp = extendedEuclidean( b, a%b, dx, dy );
        int tx = dx;
        dx = dy;
        dy = tx-dy*(a/b);
        return tmp;
    }
}
```

## Chinese Remainder Theorem

```
//2014/10/13  提供者：翁丞世、林敬哲、林必祥
/* find the x satisfied x = r[i] (mod w[i]) gcd(w[i],w[j]) = 1, i !=
j       */
int chinese( int n, int r[], int w[] )
{
    int i, x, m, v, a, b;
    m = 1;
    x = 0;
    for( i = 0; i < n; ++i ) m *= w[i];
    for( i = 0; i < n; ++i ) {
        v = m/w[i];
        extendedEuclidean( v, w[i], a, b );
        x = (x+a*w[i]*b[i])%m;
    }
    return (m+(x%m))%m;
}
```

## C 大數運算

```
//加法
void   add(int a[100], int b[100], int c[100]){
for (int i=0; i<100; i++)      // 對應的位數相加
    c[i] = a[i] + b[i];
for (int i=0; i<100-1; i++) {//  一口氣進位
    c[i+1] += c[i] / 10;       //  進位
    c[i] %= 10;                     //  進位後餘下的數
}
}

//減法
void   sub(int a[100], int b[100], int c[100]){
for (int i=0; i<100; i++)
    c[i] = a[i] - b[i];
for (int i=0; i<100-1; i++) //  一口氣借位和補位
    if (c[i] < 0){
        c[i+1]--;                 //  借位
        c[i] += 10;              //  補位
    }
}

//兩個大數乘法
void   mul(int a[100], int b[100], int c[100]){
for (int i=0; i<100; i++)
    c[i] = 0;
for (int i=0; i<100; i++)
    for (int j=0; j<100; j++)
        if (i+j < 100)
            c[i+j] += a[i] * b[j];
for (int i=0; i<100-1; i++) {//  一口氣進位
    c[i+1] += c[i] / 10;
    c[i] %= 10;
}
}

//一個大數與一個一般數之乘法
void mul(int a[100],int b, int c[100]){
for(int i=0;i<100;i++)
    c[i]=a[i]*b;
for(int i=0;i<100;i++){
    c[i+1]+=c[i]/10;
    c[i]%=10;
}
}

//兩個大數除法
void div(int a[100], int b[100], int c[100]){
int t[100];
for (int i=100-1; i>=0; i--)
    for (int k=9; k>0; k--) {//  嘗試商數
        mul(b+i, k, t);
        if (largerthan(a+i, t)){
            sub(a+i, t, c+i);
            break;
        }
    }
}
```

```
//大數除以一般數
void div(int a[100], int b, int c[100]){
int r = 0;
for (int i=100-1; i>=0; i--){
    r = r * 10 + a[i];
    c[i] = r / b;
    r %= b;
}
}
```

## Java  大數  印出 n!

```
import java.util.Scanner;
import java.math.BigInteger;
public class p623 {
static BigInteger factorial(int n) {     //  回傳 n! (大數!!)
    BigInteger result = BigInteger.valueOf(1);
                        //  把 int 灌進 BigInteger 的方法
    for (int i = 2; i <= n; i++) {
        result = result.multiply(BigInteger.valueOf(i));
                // .multiply(BigInteger)  乘上一個大數
    }


    return result;
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    int n;
    while(scanner.hasNextInt()) {      //  讀到沒有 int 為止
        n = scanner.nextInt();              //  讀入 int    n
      System.out.printf("%d! = %s\n", n, factorial(n).toString());
    }
}
} // end of public class p623
```

## BigInteger 內的函式  (Java)

| | |
|---|---|
| .add() | 加 |
| .subtract() | 減 |
| .multiply() | 乘 |
| .divide() | 除 |
| .mod() | % |
| .remainder() | 跟 mod()一樣 |
| .toString() | 回傳 value 的字串 |
| .intValue() | 回傳 value  的 int (當然  要他很小才有用) |

## C++運算式(expression)計算

```
double Eval2(istream& iss) {
    double Eval0(istream& iss);
    double res=0;
    if (iss.peek() == '(' && iss.get()) {
        res = Eval0(iss);
        iss.peek() == ')' && iss.get();
    }
    else { iss >> res; }
    return res;
}
double Eval1(istream& iss) {
    double res = Eval2(iss);
    while (iss.peek() == '*' || iss.peek() == '/')
        (iss.get() == '*')? (res*=Eval2(iss)): (res/=Eval2(iss));
    return res;
}
double Eval0(istream& iss) {
    double res = Eval1(iss);
    while (iss.peek() == '+' || iss.peek() == '-')
        res +=(iss.get() == '+')? Eval1(iss): -Eval1(iss);
    return res;
}
int main() {
    cout << Eval0(cin) << endl;
}
```

## 篩法　建質數表

```
/* usage: create_prime_table(array, array, value, const); */
void create_prime_table(int table[], int prime[], int &cnt, int arr_Max) {
    memset(table, 0, sizeof(int) * arr_Max);
    memset(prime, 0, sizeof(int) * arr_Max);
    for(int i = 2; i < arr_Max; i++)
        if(!table[i]) {
            table[i] = i;
            prime[cnt++] = i;
            for(int j = 2; i*(double)j < arr_Max; j++)
                table[i*j] = i;
        }
}
int main() {
    int *prime = new int[Max];
    int *table = new int[Max];
    int cnt = 0;

    create_prime_table(table, prime, cnt, Max);
    cout << "total:" << cnt << endl;
    for(int i = 0; i < cnt; i++)
        cout << prime[i] << "\t";
    return 0;
}
```

## LCS 長度 外加 traceback　　$O(n^2)$

```
// 為了實作方便，從陣列的第 1 格開始存入序列。
int s1[7+1] = {0, 2, 5, 7, 9, 3, 1, 2};
int s2[5+1] = {0, 3, 5, 3, 2, 8};
int array[7+1][5+1];                // DP 的表格
int prev[7+1][5+1];                 // 記錄這一格的最大值是從哪一格求得的

void LCS() {  //需要別種 type 就自己改將 array[x][0] 和 array[0][x] 都設為 0 ;
    for (int i = 1; i <= s1_length; i++)
        for (int j = 1; j <= s2_length; j++)
            if (s1[i] == s2[j])
                array[i][j] = array[i-1][j-1] + 1; //prev[i][j] = 左上方;
            else
                if (array[i-1][j] < array[i][j-1])
                     array[i][j] = array[i][j-1];     //prev[i][j] = 左方;
                     else
                        array[i][j] = array[i-1][j]; //prev[i][j] = 上方;
    cout << "LCS 的長度是" << array[s1_length][s2_length];
    cout << "LCS 為 ";    print_LCS(s1_length, s2_length);
}
void print_LCS(int i, int j){
    // 第一個或第二個 sequence 為空的的時候就可停止了
    if (i!=0 || j!=0) return;
    if (prev[i][j] == 左上方) {
        print_LCS(i-1, j-1);
        cout << s1[i];   // 印出 LCS 的元素
    }
    else if (prev[i][j] == 上方)
        print_LCS(i-1, j);
    else if (prev[i][j] == 左方)
        print_LCS(i, j-1);
}
```

## LCS - Hunt-Szymanski (LCS 轉 LIS )　O(nlogn)

```
int LCS(vector<int>& s1, vector<int>& s2) {
    //   if (s1.size() == 0 || s2.size() == 0) return 0;
    /* Counting Sort */
    vector<int> p[128]; // 假設字元範圍為 0 ~ 127
    for (int i = 0; i < s2.size(); ++i)
        p[s2[i]].push_back(i);
    /* LIS: modified version */
    vector<int> v;
    v.push_back(-1);       // 先放入一個數字，免得 v.back() 出錯
    for (int i = 0; i < s1.size(); ++i)
        for (int j = p[s1[i]].size() - 1; j >= 0; --j) {
            int n = p[s1[i]][j];
            if (n > v.back())
                v.push_back(n);
            else
                *lower_bound(v.begin(), v.end(), n) = n;
    }
    return v.size() - 1;
}
```

## LIS(Longest Increasing Subsequence) + track back O(nlogn)

```
int print_LIS(vector<int>& s, vector<int>& pos, int Max) {  // Max: LIS 長度
    vector<int>::reverse_iterator p, q;
    vector<int> lis;
    for(p = pos.rbegin(), q = s.rbegin(); *p != Max; p++, q++);
    lis.push_back(*q);  Max--;      // 最大的那個比較複雜一點
    for( ; Max > 0 && p != pos.rend() ;p++, q++)
        if(*p == Max) {
                // 從最後面找第一個符合長度的塞進 vector
            lis.push_back(*q);          Max--;
        }
    for(p = lis.rbegin(); p != lis.rend(); p++)       // 反著印回來
         cout << *p << " ";
    cout << endl;
}
int LIS(vector<int>& s, vector<int>& pos) {        // s 為原來的 sequence
   if (s.size() == 0) return 0;        // 不得不判斷的 special case
   vector<int> v;
   vector<int>::iterator p;
   v.push_back(s[0]);    // 先放入一個數字，免得 v.back() 出錯
   pos.push_back(1);
   for (int i = 1; i < s.size(); ++i)        {
      int n = s[i];
      if (n > v.back()) {
         v.push_back(n);
         pos.push_back(v.size());
      } else {
         p = lower_bound(v.begin(), v.end(), n);
         *p = n;
         pos.push_back(p - v.begin() + 1);
      }
   }
print_LIS(s, pos, v.size());      // trackback
   return v.size();
}
```

## Longest Nondescending Subsequence

```
int LNDSS( int a [], int n )
{
   int i, j, k,*b=new int[n+1], ans=0;
   b[ans]=?0x3f3f3f3f;
   for(i=0; i<n; i++){ //lower bound for Asending Subsequence
      j=std::upper_bound (b, b+ans+1, a[i])?b;
      if (j>ans) b[++ans]=a[i];
       else if(a[i]<b[j]) b[j]=a[i];
   }
   delete b; return ans;
}
```

## Convex Hull 尋找凸多邊形

```
// P 為平面上散佈的點。設定為 10 點。
// CH 為凸包上的頂點。設定為逆時針方向排列。可以視作一個 stack。
struct Point {int x, y;} P[10], CH[10*2];

// 向量 OA 外積向量 OB。大於零表示從 OA 到 OB 為逆時針旋轉。
double cross(Point& o, Point& a, Point& b) {
   return (a.x - o.x) * (b.y - o.y) - (a.y - o.y) * (b.x - o.x);
}
// 小於。依座標大小排序，先排 x 再排 y。
bool compare(Point& a, Point& b) {
   return (a.x < b.x) || (a.x == b.x && a.y < b.y);
}

void Andrew_monotone_chain() {
   // 將所有點依照座標大小排序
   sort(P, P+10, compare);
   int m = 0;    // m 為凸包頂點數目
   // 包下半部
   for (int i=0; i<10; ++i) {
     while (m >= 2 && cross(CH[m-2], CH[m-1], P[i]) <= 0) m--;
     CH[m++] = P[i];
   }
   // 包上半部，不用再包入方才包過的終點，但會再包一次起點
   for (int i=10-2, t=m+1; i>=0; --i) {
      while (m >= t && cross(CH[m-2], CH[m-1], P[i]) <= 0) m--;
      CH[m++] = P[i];
   }
   m--;     // 最後一點是重複出現兩次的起點，故要減一。
}
```

## 判斷點是否在多邊形內

```
typedef struct {double x,y;} point;
typedef vector<point> polygon;
typedef enum {EP_ON_EDGE,EP_HORIZON,EP_UPPER,EP_LOWER,EP_YES,EP_NO} ep_result;

template <typename T> inline bool between(T x,T a,T b){
   return (a<b)?(a<= x&&x<=b):(b<=x&&x<=a);}

bool point_on_edge(const point &p,const point &e0,const point &e1) {
   if (p.x*(e0.y-e1.y)+e0.x*(e1.y-p.y)+e1.x*(p.y-e0.y)==0)
      if (between(p.x,e0.x,e1.x)&&between(p.y,e0.y,e1.y))
         return true;
   return false;
}

ep_result edge_on_right_of_point(const point &p,const point &e0,const point &e1) {
   if (point_on_edge(p,e0,e1)) return EP_ON_EDGE;
   if (p.y==e0.y&&p.y==e1.y) return EP_HORIZON;
   if (p.y==e0.y&&p.x<e0.x) return (p.y<e1.y)?EP_UPPER:EP_LOWER;
   if (p.y==e1.y&&p.x<e1.x) return (p.y<e0.y)?EP_UPPER:EP_LOWER;
   double scale=(p.y-e0.y)/(e1.y-e0.y);
   if (between(scale,0.0,1.0)&&p.x<e0.x+scale*(e1.x-e0.x)) return EP_YES;
   return EP_NO;
}

bool point_in_polygon(const point &p,const polygon &g) {
   int cou=0;
   ep_result cond=EP_NO;
   const point *g_prev=&g.back();
   for (int i=0;i<g.size();i++) {
      ep_result result=edge_on_right_of_point(p,*g_prev,g[i]);
      switch (result) {
         case EP_ON_EDGE: return true;
         case EP_UPPER: case EP_LOWER:
          if (cond==EP_UPPER||cond==EP_LOWER) {
             if ((cond==EP_UPPER&&result==EP_LOWER)
                ||(cond==EP_LOWER&&result==EP_UPPER))
```

```
                              cou++;
                  cond=EP_NO;
              } else
              cond=result;
              break;
          case EP_HORIZON: case EP_NO: break;
          case EP_YES: cou++; break;
        }
        g_prev=&g[i];
    }
    return (cou%2==1);
}
```

## Breadth-First Search

BFS(Breadth-First Search)

```
void BFS(Node* root){
    stack<Node*> s;
    s.push(root);
    while (!s.empty()){
        Node* p = s.front(); s.pop();
        cout << p->data;        // 這行往下挪，結果仍相同。
        if (p->left)     s.push(p->left);
        if (p->right) s.push(p->right);
    }
}
```

## Depth-First Search 與 Topological Sort

```
bool adj[9][9];          // adjacency matrix
int visit[9];            // 記錄 DFS 遍歷過的點
int order[9], n;         // 儲存一個合理的排列順序
bool cycle;                 // 記錄 DFS 的過程中是否偵測到環

void DFS(int s){
    // back edge，有環。
    if (visit[s] == 1) cycle = true;
    // forward edge、cross edge。
    if (visit[s] == 2) return;
    visit[s] = 1;
    for (int t=0; t<9; ++t)
          if (adj[s][t])
              DFS(t);
    visit[s] = 2;
    order[n--] = s;         // 記錄合理的排列順序
}

void topological_sort(){
    // 初始化
    for (int i=0; i<9; i++) visit[i] = 0;
    cycle = false;
    n = 9-1;
    // 進行 DFS
    for (int s=0; s<9; ++s)
            if (!v[s])
                DFS(s);
    // 輸出結果
    if (cycle)
        cout << "圖上有環";
    else
            // 印出一個合理的排列順序
```

```
        for (int i=0; i<9; ++i)
            cout << order[i];
}
```

## 找出最短路徑樹 Dijkstra　O($n^2$)

```
int w[9][9];  //  一張有權重的圖
int d[9];                       //  紀錄起點到各個點的最短路徑
長度
int parent[9];                  //  紀錄各個點在最短路徑樹上的
父親是誰
bool visit[9]; //  紀錄各個點是不是已在最短路徑樹之中

void dijkstra(int source) {
    for (int i=0; i<9; i++)
        visit[i] = false;       // initialize
    for (int i=0; i<9; i++)
        d[i] = 1e9;             // 1e9 -> 1 * 10^9
    d[source] = 0;
    parent[source] = source;
    for (int k=0; k<9; k++)    {
        int a = -1, b = -1, min = 1e9;
        for (int i=0; i<9; i++)
            if (!visit[i] && d[i] < min) {
                a = i;    //  記錄這一條邊
                min = d[i];
            }
            if (a == -1) break; //起點有連通的最短路徑都已找完
//          if (min == 1e9) break;    //  不連通即是最短路徑長度
無限長
            visit[a] = true;
            for (b=0; b<9; b++)         //  把起點到 b 點的最短路
徑當作捷徑
                if (!visit[b] && d[a] + w[a][b] < d[b]) {
                    d[b] = d[a] + w[a][b];
                    parent[b] = a;
                }
        }
}
```

## 從最短路徑樹上找出最短路徑

```
//  若要找出某一點的最短路徑，利用 parent 陣列即可。
void find_path(int x)     //  印出由起點到 x 點的最短路徑
{
    if (x != parent[x]) //  先把之前的路徑都印出來
        find_path(parent[x]);
    cout << x << endl;    //  再把現在的位置印出來
}
```

## 所有兩點之間的最短路徑 Floyd-Warshall

```
int w[9][9];
int d[9][9];
int next[9][9]; //  由 i 點到 j 點的路徑，第二點為
next[i][j]。

void Floyd_Warshall(){
for (int i=0; i<9; i++)
    for (int j=0; j<9; j++){
        d[i][j] = w[i][j];
        next[i][j] = j; // 一開始沒有中繼點，所以第二點就是
終點。
    }
for (int i=0; i<9; i++)
    d[i][i] = 0;
for (int k=0; k<9; k++)
    for (int i=0; i<9; i++)
        for (int j=0; j<9; j++)
            if (d[i][k] + d[k][j] < d[i][j]){
                d[i][j] = d[i][k] + d[k][j];
                    //由 i 點到 j 點的路徑的第二點，
                        // 正是由 i 點到 k 點的路徑的第二點。
                next[i][j] = next[i][k];
            }
}
// 印出由 i 點到 j 點的最短路徑，遞迴版
void find_path(int i, int j){
    cout << i;              // 先把起點印出來
    if (i != j)           // 當還有中繼點的時候
        find_path(next[i][j], j);      // 再把第二點以後的路徑都
印出來
}
// 印出由 a 點到 b 點的最短路徑，迴圈版
void find_path(int a, int b){
    for (int i=a; i!=b; i=p[i][b])
        cout << i;
    cout << b;
}
```

## Bellman Ford + Queue (shortest path for negative edge) O($n^3$)

```
const int maxn = maxm = 1000005
const int inf = 1000000000
int nbs[maxn], next[maxm], value[maxn], open[maxn],
open1[maxn] ;
int ev[maxm], ew[maxm], mk[maxn],n,m,num, cur, tail ;
void BellmanFord ( int src)
{
     int i , j , k , l , t , u , v , p=0;
     for ( i =1; i<=n ; i ++) { value[i]= inf ; mk[i]=0 ; }
     value[src]= tail=0; open[0]= src;
     while(++p , tail >=0){
          for(i =0; i<=tail ; i++) open1 [i]=open[i] ;
          for( cur=0, t=tail , tail =-1; cur<=t ; cur++)
               for (u=open1[cur], i=nbs [ u ] ; i ; i=next[i]) {
                    v=ev[i] ;
                    if ( value[u]+ew[i]<value[v ) {
                         value[v]=value[u]+ew[i] ;
                         if (mk[v]!=p ) {
                              open[++tail]=v ; mk[v]=p ; }
                    }
               }
     }
}
```

## Prim’s minimum spanning tree O（$n^2$）

```
/* usedp=>how many points already used
p->array of structures, consisting x,y,& used/not used
this problem is to get the MST of graph with n vertices
which weight of an edge is the distance between 2 points */

usedp=p[0].used=1; /* select arbitrary point as starting point
*/
while (usedp<n) {
    small=-1.0;
    for (i=0;i<n;i++) if (p[i].used)
        for (j=0;j<n;j++) if (!p[j].used) {
            length=sqrt(pow(p[i].x-p[j].x,2) + pow(p[i].y-p[j].y,2));
            if (small==-1.0 || length<small) {
                small=length;
                smallp=j;
            }
        }
        minLength+=small;
        p[smallp].used=1;
        usedp++ ;
}
```

## Kruskal’s minimum spanning tree

```
struct Edge{
        int a, b, c;        // 起點，終點，權重。
        bool operator<(const Edge& e){// 用於比大小的函式
            return c < e.c;
        }
};
// edges[]存放著圖上所有邊，E 為邊的總數，V 為點的總
數。
void Kruskal(Edge edges[], int E, int V){
    DisjointSets sets;
    // Quick Sort
    // 將圖上所有邊依照權重大小，由小到大排序。
    sort(edges, edges+E);
    int i, j;
    for (i = 0, j = 0; i < V-1 && j < E; ++i){      // 找出 V-1 條邊
        // 擷取出最短的、不會造成環的邊
        while (sets.find(edges[j].a, edges[j].b)) j++;
        // 連結選到的邊
        sets.union(edges[j].a, edges[j].b);
        // 印出選到的 edge
        cout << "起點：" << edges[j].a << "終點：" << dges[j].b
            << "權重：" << edges[j].c;
        j++;        // 別忘記累計索引值。也可以寫入迴圈。
    }
    if (i != V-1) cout << "MST 不存在!";
}
```

## Tarjan’s Strongly connected components 收縮所有的環

```
int adj[9][9];                       // adjacency matrix
int dis[9], low[9], t = 0;    // 遍歷順序、追溯到的最高祖先
（的遍歷順序）
int stack[9], top = 0;            // 堆疊
bool instack[9];            // 紀錄 DFS forest 目前還有哪些點
int contract[9];                    // 每個點收縮到的點

void DFS(int i){
    dis[i] = low[i] = ++t;
    stack[top++] = i;
    instack[i] = true;
    for (int j=0; j<9; ++j)
        if (adj[i][j]){
            if (!dis[j])
                DFS(j);
            if (instack[j])
                low[i] = min(low[i], low[j]);
        }
    // 形成 SCC，從目前的 DFS forest 移除它。
    // i 點也是 SCC 裡面，發現時間最早的點。
    if (dis[i] == low[i]){
        int j;
        do{
            j = stack[--top];
            instack[j] = false;
            contract[j] = i;
        } while (j != i);
    }
}
void tarjan(){
    memset(dis, 0, sizeof(dis));
    t = 0;
    for (int i=0; i<9; ++i)
        if (!dis[i])
            DFS(i);
}
```

## 最大二分匹配 Maximum Cardinality Bipartite Matching (無權重圖)

```
int nx, ny;             // X 的點數目、Y 的點數目
int mx[100], my[100];   // X 各點的配對對象、Y 各點的配對對象
bool vy[100];           // 紀錄 Graph Traversal 拜訪過的點
bool adj[100][100];     // 精簡過的 adjacency matrix

// 以 DFS 建立一棵交錯樹
bool DFS(int x) {
    for (int y=0; y<ny; ++y)
        if (adj[x][y] && !vy[y]) {
            vy[y] = true;
            // 找到擴充路徑
            if (my[y] == -1 || DFS(my[y])) {
                mx[x] = y; my[y] = x;
                return true;
            }
        }
    return false;
}

int bipartite_matching() {
    // 全部的點初始化為未匹配點。
    memset(mx, -1, sizeof(mx));
    memset(my, -1, sizeof(my));
    // 依序把 X 中的每一個點作為擴充路徑的端點，
    // 並嘗試尋找擴充路徑。
    int c = 0;
    for (int x=0; x<nx; ++x) {
    //  if (mx[x] == -1)      // x 為未匹配點，這行可精簡。
      // 開始 Graph Traversal
        memset(vy, false, sizeof(vy));
        if (DFS(i)) c++;
    }
    return c;
}
```

## 匈牙利演算法: 找出一個最大權最大二分匹配（精簡過的 adjacency matrix）

```
int N;                   // X 的點數目，也等於 Y 的點數目
int mx[50], my[50]; // X 各點的配對對象、Y 各點的配對對象
int q[50], qf, qb;   // 交錯樹，X 的部分
int py[50];           // 交錯樹，Y 的部分
int lx[50], ly[50]; // vertex labeling
int adj[50][50];      // 精簡過的 adjacency matrix
bool match(int r){
    while (true){
        memset(py, -1, sizeof(py));
        for (qf=0, qb=1, q[0]=r; qf<qb; )
            for (int x=q[qf++], y=0; y<N; ++y)
                if (lx[x] + ly[y] == adj[x][y] && py[y] == -1){
                    q[qb++] = my[y]; py[y] = x;
                    if (my[y] == -1){
                        for (int ty = 0; ty != -1; y = ty)
                            ty = mx[x = py[y]], my[y] = x, mx[x] = y;
                        return true;
                    }
                }
        int d = 1e9;
        for (int i=0; i<qb; ++i)        // 在交錯樹上的 X
            for (int y=0; y<N; ++y) if (py[y] == -1)       // 不在交錯樹上的 Y
                if (adj[q[i]][y] != 1e9)
                    d = min(d, lx[q[i]] + ly[y] - adj[q[i]][y]);
        if (d == 1e9) break;   // 未新增等邊，無擴充路徑。
        for (int i=0; i<qb; ++i) lx[q[i]] -= d;
        for (int y=0; y<N; ++y) if (py[y] != -1) ly[y] += d;
    }
    return false;
}

int Hungarian(){
    memset(mx, -1, sizeof(mx));
    memset(my, -1, sizeof(my));
    memset(lx, 0, sizeof(lx));
    memset(ly, 0, sizeof(ly));
    for (int x=0; x<N; ++x)
        for (int y=0; y<N; ++y)
            lx[x] = max(lx[x], adj[x][y]);
    for (int x=0; x<N; ++x)
        if (!match(x))
            mx[x] = -1; // 此點為未匹配點
    int cost = 0;
    for (int x=0; x<N; ++x)
        if (mx[x] != -1)
        cost += adj[x][mx[x]];
    return cost;
}
```

## Ford-Fulkerson Algorithm: 給定一張圖，並給定源點、匯點，找出其中一個最大流

```
#define in(i) 2*(i-1)
#define out(i) 2*(i-1)+1
#define INT_MAX 2147483647
#define min(a,b) (a>b)?b:a
int **value;  //權重圖
int **x_to_y;               //用來跑 Ford_Fulkerson,初始化為 0
int *pre;                   //用來記錄上個點
int Ford_Fulkerson(int node,int start,int end){    //
    int ans=0;
    while(true){
        for(int i=0;i<node;i++)pre[i]=-1;
        vector<int>dir;
        dir.push_back(start);
        while(dir.size()!=0&&pre[end]==-1){
            int temp=dir.at(0);
            dir.erase(dir.begin());
            for(int i=0;i<node;i++){
if(i!=start&&pre[i]==-1&&value[temp][i]-x_to_y[temp][i]>0){
                    pre[i]=temp;
                    dir.push_back(i);
                }
            }
        }
        if(pre[end]==-1)break;
        int temp=INT_MAX;
        int v=end;
        for(int i=pre[v];i!=-1;i=pre[v]){
            temp=min(temp,value[i][v]-x_to_y[i][v]);
            v=i;
        }
        v=end;
        for(int i=pre[v];i!=-1;i=pre[v]){
            x_to_y[i][v]+=temp;
            x_to_y[v][i]=-x_to_y[i][v];
            v=i;
        }
        ans+=temp;
    }
    return ans;
}
```

## Travelling Salesman Problem

```
int n , x [maxn] , y [maxn] , id[maxn] ;
double g [maxn] [ maxn] ;
double dis( int x1 , int y1 , int x2 , int y2 )
{ return sqrt( ( x1-x2 )*( x1-x2)+(y1-y2)*( y1-y2 ) ) ; }
double solve( )
{
int i , j , k , l , loop ;
double cur , ans=1e30 ;
for ( i =0; i<n ; i++)
      for ( j =0; j<n ; j++)
            g[i][j]=dis(x[i] ,y[i], x[j] , y[j] ) ;
for ( k=0;k<n ; k++){
      for ( l =0; l <50; l++){
            for ( i =0; i<n ; i++)
              id[i]= i ;
            std :: swap ( id [ 0 ] , id [ k ] ) ;
            std :: random_shuffle ( id+1, id+n ) ;
            loop=1;
            while ( loop ){
                loop=0;
                for ( i =1; i<n ; i++)
                   for ( j=i +1; j<n-1; j++)
                        if(g[id[i-1]][id[i]]+g[id[j]][id[j+1]]
                  >g[id[i-1]][id[j]]+g[id[i]][id[j+1]]+1e-8 ) {
                              loop=1;
                              std :: reverse( id+i , id+j +1);
                        }
            } ;
            for ( cur=0, i =0; i<n-1; i++)
                cur+=g[id[i]][id[i+1]] ;
            if ( cur<ans ) ans=cur ;
      }
}
return ans ;
}
```

## Traveling Salesperson Problem (branch and bound)

```
#include <math.h>
#include <stdio.h>
#define SIZE 10
int min(int map[SIZE][SIZE],int n) {
int minisum=0;
int i,j,mini;
for (i=0;i<n;i++) {
    for (mini=30000,j=0;j<n;j++)
        if ((map[i][j]>=0)&&(mini>map[i][j]))
            mini=map[i][j];
            for (j=0;j<n;j++) //需再往左，不知何故，無法排版
            map[i][j]=map[i][j]-mini;
        minisum=minisum+mini;
}
for (j=0;j<n;j++) {
    for (mini=30000,i=0;i<n;i++)
        if ((map[i][j]>=0)&&(mini>map[i][j]))
            mini=map[i][j];
    for (i=0;i<n;i++)
            map[i][j]=map[i][j]-mini;
    minisum=minisum+mini;
}
return minisum;
}

void split(int map[SIZE][SIZE],int tmp[SIZE][2],
      int tmpout[SIZE*SIZE][2],int sum,int n,int *lowerbound) {
int with[SIZE][2],without[SIZE*SIZE][2];
int i,j,m,mini,tmpmini,minii,minij,flag;
for (i=0;i<n;i++) {
    with[i][0]=tmp[i][0];
    with[i][1]=tmp[i][1];
```

```
}
for (i=0;i<n*n;i++) {
   without[i][0]=tmpout[i][0];
   without[i][1]=tmpout[i][1];
}
for (i=0,mini=30000;i<n;i++)
   for (j=0;j<n;j++,flag=1) {
      for (m=0;(m<n)&&(flag==1);m++)
         if ((with[m][0]==i)||(with[m][1]==j)||((with[m][0]==j)
              &&(with[m][1]==i)))
            flag=0;
      for (m=0;(m<n*n)&&(flag==1);m++)
         if ((without[m][0]==i)&&(without[m][1]==j))
            flag=0;
      if ((flag==1)&&(map[i][j]>=0)&&(mini>map[i][j])) {
         mini=map[i][j];
         minii=i;
         minij=j;
      }
   }
if (mini==30000)
   return;
for (j=0,tmpmini=30000;j<n;j++)
if ((j!=minij)&&(tmpmini>map[minii][j])&&(map[minii][j]>=0))
    tmpmini=map[minii][j];
 if (sum + mini < *lowerbound) {
    if (with[n-2][0]>=0) {
       *lowerbound=sum+mini;
       return;
    }
    else {
       for (i=0;(i<n)&&(with[i][0]>=0);i++) {}
       with[i][0]=minii;
       with[i][1]=minij;
       split(map,with,tmpout,sum+mini,n,lowerbound);
```

```
                    }
     }
     if ((sum+tmpmini<*lowerbound)&&(without[n*n-
2][0]==-1)) {
        for (i=0;(i<n*n)&&(without[i][0]>=0);i++) {}
        without[i][0]=minii;
        without[i][1]=minij;
        split(map,tmp,without,sum,n,lowerbound);
    }
   return;
}
```

## 產生所有 permutation (recursion)

```
#include <iostream>
void Permutations (char *a, const int k, const int m)
//Generate all the permutations of a[k], ..., a[m]
{   if (k == m) {   //Output permutation
        for (int i = 0; i <= m; i++) cout << a[i] << " ";
         cout << endl;
    }
    else { //a[k], ..., a[m] has more than one permutation
         for (int i = k; i <= m; i++) {
              swap(a[k], a[i]); // exchange
              Permutations(a, k+1, m);
             swap(a[k], a[i]);
          }
    } // end of else
}
int main()
{    char b[10]＝{'a','b','c','d','e','f','g'};
    Permutations(b,0,2);
    cout << endl
}
```

## 二維平面最大點 Max. point on 2-D plane

```
#include <math.h>
#include <stdio.h>
#define SIZE 100
/* p[SIZE][2]: Input Point
     ans[SIZE][2]: Max Point
     n: # of input points
     return: # of max. point
*/
int findmaxpoint(int p[SIZE][2],int ans[SIZE][2],int n)
{
int tmp[SIZE][2],index,i,j,temp;
int min=-30000;
for (i=0;i<n;i++) {
   tmp[i][0]=p[i][0];
   tmp[i][1]=p[i][1];
}
for (i=0;i<n;i++)
   for (j=0;j<n-i-1;j++)
         if (tmp[j][0]>tmp[j+1][0]) {
            temp=tmp[j][0];
            tmp[j][0]=tmp[j+1][0];
         tmp[j+1][0]=temp;
         temp=tmp[j][1];
         tmp[j][1]=tmp[j+1][1];
         tmp[j+1][1]=temp;
         }
for (i=n-1,index=0;i>=0;i--)
   if (tmp[i][1]>=min) {
         ans[index][0]=tmp[i][0];
         ans[index][1]=tmp[i][1];
         min=tmp[i][1];
         index++;
   }
return index;
}
```

## 兩直線夾角

```
#include <math.h>
typedef struct {
     double x, y;
} Point;
double angle(Point p1, Point p2, Point p3, Point p4)
/*   Subject : 求兩直線夾角
   p1, p2    :  直線上兩點
   p3, p4    :  直線上兩點
   return value : 直線夾角中較小者 (單位為"弳")
*/
{
double v1[2], v2[2];
double dot, dia, dib;

v1[0] = p1.x - p2.x;
v1[1] = p1.y - p2.y;
v2[0] = p3.x - p4.x;
v2[1] = p3.y - p4.y;

dot = v1[0] * v2[0] + v1[1] * v2[1];
dia = v1[0] * v1[0] + v1[1] * v1[1];
dib = v2[0] * v2[0] + v2[1] * v2[1];
return acos(abs(dot) / sqrt(dia * dib));
}
```

## 直線與直線的交點

```
typedef struct {
     double x, y;
} Point;

Point linex(Point p1, Point p2, Point p3, Point p4, int
*flag)
/*   Input    :
p1, p2 :  直線上之兩點
p3, p4 :  另一直線上之兩點
flag     :  存放兩直線的關係
Output   :
傳回值 :  兩直線交點
flag     : = 0 (有交點)
              = 1 (兩線重合, 傳回值未定義)
              = 2 (兩線平行, 傳回值未定義)
*/
{
Point ret;
double v1[3], v2[3];
double xy, xc, yc;

v1[0] = p2.y - p1.y;
v1[1] = p1.x - p2.x;
v1[2] = -(v1[0] * p1.x + v1[1] * p1.y);

v2[0] = p4.y - p3.y;
v2[1] = p3.x - p4.x;
v2[2] = -(v2[0] * p3.x + v2[1] * p3.y);

xy = v1[0] * v2[1] - v1[1] * v2[0];
xc = v1[2] * v2[0] - v1[0] * v2[2];
yc = v1[1] * v2[2] - v1[2] * v2[1];
if (xy == 0) {
   /* *flag = (xc == 0) ? 1 : 2; */
   /* xeon.040823:  應該要檢查  xc  和  yc */
   *flag = (xc == 0 && yc == 0) ? 1 : 2;
   return ret;
}
*flag = 0;
ret.x = yc / xy;
ret.y = xc / xy;
return ret;
}
```

## 線段與線段的交點

```
typedef struct
{
double x, y;
} Point;
Point linexx(Point p1, Point p2, Point p3, Point p4,
int *flag)
/*   Input    :
p1, p2 :  直線之端點
p3, p4 :  另一線段之端點
flag     :  存放兩線段的關係
Output   :
傳回值 :  兩直線交點
flag     : = 0 (有交點)
        = 1 (兩直線重合, 傳回值未定義, 且線段不
一定有交集)
  = 2 (兩線平行, 傳回值未定義)
  = 3 (無交點)*/
{
Point ret;
double v1[3], v2[3];
double xy, xc, yc;

v1[0] = p2.y - p1.y;
v1[1] = p1.x - p2.x;
v1[2] = -(v1[0] * p1.x + v1[1] * p1.y);

v2[0] = p4.y - p3.y;
v2[1] = p3.x - p4.x;
v2[2] = -(v2[0] * p3.x + v2[1] * p3.y);

xy = v1[0] * v2[1] - v1[1] * v2[0];
xc = v1[2] * v2[0] - v1[0] * v2[2];
yc = v1[1] * v2[2] - v1[2] * v2[1];

if (xy == 0){
   *flag = (xc == 0) ? 1 : 2;
   return ret;
}

*flag = 0;
ret.x = yc / xy;
ret.y = xc / xy;

if ((ret.x - p1.x) * (p2.x - ret.x) < 0)
   *flag = 3;
if ((ret.y - p1.y) * (p2.y - ret.y) < 0)
   *flag = 3;
if ((ret.x - p3.x) * (p4.x - ret.x) < 0)
   *flag = 3;
if ((ret.y - p3.y) * (p4.y - ret.y) < 0)
   *flag = 3;
return ret;
}
```

## 三角形外心

```
#include <stdio.h>
typedef struct {
            double x, y;
} Point;

Point circum(Point c1, Point c2, Point c3)
{
Point ret;
double a1, b1, a2, b2;

a1 = c1.x - c3.x;
b1 = c1.y - c3.y;

a2 = c2.x - c3.x;
b2 = c2.y - c3.y;

ret.x = (a1*a1 * b2 - a2*a2 * b1 + b1*b1 * b2 - b1 *
b2*b2) / (2 * (b2 * a1 - b1 * a2)) + c3.x;
ret.y = (b1*b1 * a2 - b2*b2 * a1 + a1*a1 * a2 - a1 *
a2*a2) / (2 * (a2 * b1 - a1 * b2)) + c3.y;
return ret;
}
```

## 兩圓內公切線方程式

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

#define INPUTFILE "c_intan.in"     /*  輸入檔檔名
*/

typedef struct POINT{
            double x,y;
}POINT;
/* center :  圓心的座標  (center.x,center.y)
     r          :  圓的半徑
     p           :  圓外一點的座標  (p.x,p.y)
     m1,m2    :  傳回的切線斜率
*/
#define ERROR 0                                        /*
點在圓內  */
#define ONLY_ONE_PERPENDICULAR_LINE 1      /*
點在圓上,切線垂直 x 軸  */
#define AMONG_ONE_IS_PERPENDICULAR 2       /*
兩條切線,其中一條垂直 x 軸  */
#define TWO_GENERAL_LINES 3                        /*
兩條不垂直 X 軸的切線  */
#define ONLY_ONE_GENERAL_LINE 4                 /*
點在圓上,切線不垂直 X 軸  */
}


int point2circle_slope(POINT center, double r,POINT
p, double* m1, double*
m2) {
```

```
double a,b,c,t;
double D=0;
bool OnCircle=false;

a = p.x - center.x;
a *= a;
b = p.y - center.y;
b *= b;
c = sqrt(a + b);
if (c < r)
   return ERROR;
else if (c == r)
   OnCircle = true;
b = (p.y - center.y) * (center.x - p.x);
a = center.x - p.x;
a *= a;
a -= r*r;
c = p.y - center.y;
c *= c;
c -= r*r;

if (a == 0 && !OnCircle){
   *m1 = (-c) / (2 * b);
   return AMONG_ONE_IS_PERPENDICULAR;
}
else{
   D = 4 * (b*b - a*c);
   if (D == 0){
      if (p.y - center.y == 0)
         return ONLY_ONE_PERPENDICULAR_LINE;
      else{
         *m1 = (-2*(p.y - center.y) * (center.x -p.x)) /
(2*a);
         return ONLY_ONE_GENERAL_LINE;
      }
   }
   else{
      D = sqrt(D);
      t = -2*(p.y - center.y) * (center.x - p.x);
      *m1 = (t + D) / (2*a);
      *m2 = (t - D) / (2*a);
      return TWO_GENERAL_LINES;
   }
}
}
/* c1 :圓 1 的座標
  r1 :圓 1 的半徑
  c2 :圓 2 的座標
  r2 :圓 2 的半徑
  *m1 :回傳外公切線斜率(若公切線只有一條,則
斜率存放在*m1)
  *m2 :回傳外公切線斜率
  *p   :兩外公切線的交點
*/
int inner_tangent(POINT c1, double r1, POINT c2,
double r2, double* m1,
double* m2, POINT* p)
{
if (r1 + r2 > sqrt(pow(c1.x - c2.x,2) + pow(c1.y -
c2.y,2)))
   return ERROR;
p->x = (r2*c1.x + r1*c2.x) / (r1 + r2);
p->y = (r2*c1.y + r1*c2.y) / (r1 + r2);
return point2circle_slope(c1,r1,*p,m1,m2);
}
```

## 求兩圓外公切線方程式

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

#define INPUTFILE "c_outtan.in"                  /*
輸入檔檔名*/
typedef struct POINT{
            double x,y;
}POINT;

/* center :  圓心的座標  (center.x,center.y)
     r          :  圓的半徑
     p          :  圓外一點的座標  (p.x,p.y)
     m1,m2    :  傳回的切線斜率
*/

#define ERROR 0                                   /*
點在圓內  */
#define ONLY_ONE_PERPENDICULAR_LINE 1      /*
點在圓上,切線垂直 x 軸  */
#define AMONG_ONE_IS_PERPENDICULAR 2      /*
兩條切線,其中一條垂直 x 軸  */
#define TWO_GENERAL_LINES 3                  /*
兩條不垂直 X 軸的切線  */
#define ONLY_ONE_GENERAL_LINE 4              /*
點在圓上,切線不垂直 X 軸  */

int point2circle_slope(POINT center, double r,POINT
p, double* m1, double*
m2)
{
double a,b,c,t;
double D=0;
bool OnCircle=false;

a = p.x - center.x;
a *= a;
b = p.y - center.y;
b *= b;
c = sqrt(a + b);
if (c < r)
   return ERROR;
else if (c == r)
   OnCircle = true;
b = (p.y - center.y) * (center.x - p.x);
a = center.x - p.x;
a *= a;
a -= r*r;
c = p.y - center.y;
c *= c;
c -= r*r;

if (a == 0 && !OnCircle){
   *m1 = (-c) / (2 * b);
   return AMONG_ONE_IS_PERPENDICULAR;
}
else{
   D = 4 * (b*b - a*c);
   if (D == 0){
      if (p.y - center.y == 0)
         return ONLY_ONE_PERPENDICULAR_LINE;
      else{
         *m1 = (-2*(p.y - center.y) * (center.x -
               p.x)) / (2*a);
      return ONLY_ONE_GENERAL_LINE;
      }
   }
   else{
      D = sqrt(D);
      t = -2*(p.y - center.y) * (center.x - p.x);
      *m1 = (t + D) / (2*a);
      *m2 = (t - D) / (2*a);
      return TWO_GENERAL_LINES;
   }
}
}

/*
  c1 :圓 1 的座標
  r1 :圓 1 的半徑
  c2 :圓 2 的座標
  r2 :圓 2 的半徑
  *m1 :回傳外公切線斜率(若公切線只有一條,則
斜率存放在*m1)
  *m2 :回傳外公切線斜率
  *p   :兩外公切線的交點
*/
#define SAME 5                        /*  兩圓半
徑一樣,兩外公切線不垂直 X 軸  */
#define TWO_ORTHOGONAL_LINES 6      /*  兩圓半
徑一樣,兩外公切線垂直 X 軸  */

int outer_tangent(POINT c1, double r1, POINT c2,
double r2, double* m1,
double* m2, POINT* p)
{
POINT t;
double tr;

if (fabs(r1 - r2) >= sqrt(pow(c1.x - c2.x,2)
      + pow(c1.y -c2.y,2)))
   return ERROR;
if (r1 == r2){
   if (c1.x == c2.x){
      p->y = c1.y;
      p->x = c1.x + r1;
      return TWO_ORTHOGONAL_LINES;
   }
   else{
      *m1 = (c1.y - c2.y) / (c1.x - c2.x);
      p->x = c1.x + sqrt((pow(r1,2) + pow(*m1,2)) /
(pow(*m1,2) + 1));
      if (*m1 == 0)
         p->y = c1.y+r1;
      else
         p->y = -(p->x - c1.x) / *m1 + c1.y;
      return SAME;
   }
}
else if (r2 > r1){
   t.x = c1.x; t.y = c1.y; tr = r1;
   c1.x = c2.x; c1.y = c2.y; r1 = r2;
   c2.x = t.x; c2.y = t.y; r2 = tr;
}

p->x = (r1*c2.x - r2*c1.x) / (r1 - r2);
p->y = (r1*c2.y - r2*c2.y) / (r1 - r2);
return point2circle_slope(c1,r1,*p,m1,m2);
}
```

## 圓外一點與圓的切線方程式之斜率

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

#define INPUTFILE "cir2pnt.in"     /*  輸入檔檔名*/

typedef struct POINT{
            double x,y;
}POINT;

/* center :  圓心的座標  (center.x,center.y)
     r          :  圓的半徑
     p          :  圓外一點的座標  (p.x,p.y)
     m1,m2    :  傳回的切線斜率
*/

#define ERROR 0                                   /*
點在圓內  */
#define ONLY_ONE_PERPENDICULAR_LINE 1      /*
點在圓上,切線垂直 x 軸  */
#define AMONG_ONE_IS_PERPENDICULAR 2      /*
```

```
兩條切線,其中一條垂直 x 軸  */
#define TWO_GENERAL_LINES 3                    /*
兩條不垂直 X 軸的切線  */
#define ONLY_ONE_GENERAL_LINE 4                /*
點在圓上,切線不垂直 X 軸  */

int point2circle_slope(POINT center, double r,POINT
p, double* m1, double* m2)
{
double a,b,c,t;
double D=0;
bool OnCircle=false;

a = p.x - center.x;
a *= a;
b = p.y - center.y;
b *= b;
c = sqrt(a + b);
if (c < r)
    return ERROR;
else if (c == r)
    OnCircle = true;
b = (p.y - center.y) * (center.x - p.x);
a = center.x - p.x;
a *= a;
a -= r*r;
c = p.y - center.y;
c *= c;
c -= r*r;

if (a == 0 && !OnCircle){
    *m1 = (-c) / (2 * b);
    return AMONG_ONE_IS_PERPENDICULAR;
}
else{
    D = 4 * (b*b - a*c);
    if (D == 0){
        if (p.y - center.y == 0)
            return ONLY_ONE_PERPENDICULAR_LINE;
        else{
            *m1 = (-2*(p.y - center.y) * (center.x -
p.x)) / (2*a);
            return ONLY_ONE_GENERAL_LINE;
        }
    }
    else{
        D = sqrt(D);
        t = -2*(p.y - center.y) * (center.x - p.x);
        *m1 = (t + D) / (2*a);
        *m2 = (t - D) / (2*a);
        return TWO_GENERAL_LINES;
    }
}
}
```

## 兩圓交點

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

struct circle {
    double x, y, r;
};
struct answer {
    double x, y;
};

int cxc(struct circle cir1, struct circle cir2, struct
answer *ans1, struct answer *ans2)
{
double a, b, R, D, rem;
double LA, A, B, C, Hmm;
a = cir2.x - cir1.x;
b = cir2.y - cir1.y;
R = cir1.r + cir2.r;
D = hypot(a, b);
if (D > R) {
    return 0;           //  兩圓無交點
}
else if (D == R) {
    ans1->x = ans2->x = cir1.x + a * cir1.r / R;
    ans1->y = ans2->y = cir1.y + b * cir1.r / R;
    return 1;           //  兩圓交於一點
}
else {
    if (cir2.r > cir1.r) {
        rem = cir2.x;
        cir2.x = cir1.x;
        cir1.x = rem;
        rem = cir2.y;
        cir2.y = cir1.y;
        cir1.y = rem;
        rem = cir2.r;
        cir2.r = cir1.r;
        cir1.r = rem;
    }
    if (D < cir1.r) {
        if (cir2.r < cir1.r - D)
            return 0;
        else if (cir2.r == cir1.r - D) {
            ans1->x = ans2->x = cir1.x + a * cir1.r / R;
            ans1->y = ans2->y = cir1.y + b * cir1.r / R;
            return 1;
        }
    }
    if (a == 0) {
        ans1->y = ans2->y = (cir1.r*cir1.r - cir2.r*cir2.r
+ b*b) / (2*b);
        ans1->x = sqrt(cir1.r*cir1.r - (ans1->y)*(ans1-
>y));
        ans2->x = -(sqrt(cir1.r*cir1.r - (ans1->y)*(ans1-
>y)));
        return 2;           //  兩圓相交兩點
    }
    else if (b == 0) {
        ans1->x = ans2->x = (cir1.r*cir1.r - cir2.r*cir2.r
+ a*a) / (2*a);
        ans1->y = sqrt(cir1.r*cir1.r - (ans1->x)*(ans1-
>x));
        ans2->y = -(sqrt(cir1.r*cir1.r - (ans1->x)*(ans1-
>x)));
        return 2;           //  兩圓相交兩點
    }
    else {
        LA = (a*a + b*b + cir1.r*cir1.r - cir2.r*cir2.r) /
(2*b);
        B = LA * (a/b);
        A = 1 + (a*a) / (b*b);
        C = LA * LA - cir1.r*cir1.r;
        Hmm = sqrt(B*B - A*C);
        ans1->x = (-b + Hmm) / a + cir1.x;
        ans1->y = sqrt(cir1.r*cir1.r - ((-b + Hmm) / a) *
((-b + Hmm) / a));
        ans2->x = (-b - Hmm) / b + cir1.x;
        ans2->y = sqrt(cir1.r*cir1.r - ((-b - Hmm) / a) *
((-b - Hmm) / a));
        return 2;           //  兩圓相交兩點
    }
}
}
```

## All-pair Shortest Path of a Directed Graph,without getting the path

```
#include<stdio.h>
#define SIZE 50          //maximum number of nodes
#define INF 9999         //used a large number
denote infinity
void all_pair_shortest(int record[][SIZE],int
dis[][SIZE],int n)
/* record[][]: input matrix to represent a
graph,unchanged
      dis[][]: output data, length of the shortest path
of each pair
      n: # of nodes (vertices) in the graph
      Note:    A very large number in a[][] represents
that no edge connects    the pair of nodes      (ie
INF)
*/
{
int i,j,k;

for(i=0;i<n;i++)        /*copy data from record to dis*/
    for(j=0;j<n;j++)
        dis[i][j]=record[i][j];

for(k=0;k<n;k++)
    for(i=0;i<n;i++)
for(j=0;j<n;j++){
    if(dis[i][k]==INF || dis[k][j]==INF)
        continue;
    if(dis[i][j]>dis[i][k]+dis[k][j])
        dis[i][j]=dis[i][k]+dis[k][j];
}
}
```

## All-pair Shortest Path of a Directed Graph,with getting the path

```
#include<stdio.h>
#define SIZE 50          //maximum number of nodes
#define INF 9999         //used a large number
denote infinity
int grob=0;

void printpath(int path[][SIZE],int pse[],int start,int
end)
{
if(path[start][end]==start){
    pse[grob++]=end;
    return;
}
printpath(path,pse,start,path[start][end]);
printpath(path,pse,path[start][end],end);
}

void all_pair_shortest(int record[][SIZE],int
dis[][SIZE],
                int path[][SIZE],int n)
/* record[][]: input matrix to represent a
graph,unchanged
      dis[][]: output data, length of the shortest path
of each pair
      path[][]: the path of shortest path
      n: # of nodes (vertices) in the graph
      Note:    A very large number in a[][] represents
that no edge connects    the pair of nodes      (ie
INF)
*/
{
int i,j,k;
for(i=0;i<n;i++)             /*copy data from record to
dis*/
    for(j=0;j<n;j++){         /*also initial the path*/
        dis[i][j]=record[i][j];
        path[i][j]=i;
    }
for(k=0;k<n;k++)
    for(i=0;i<n;i++)
        for(j=0;j<n;j++){
            if(dis[i][k]==INF || dis[k][j]==INF)
            continue;
        if(dis[i][j]>dis[i][k]+dis[k][j]){
            dis[i][j]=dis[i][k]+dis[k][j];
            path[i][j]=k;
        }
    }
}
```

## Breadth-First Search of a directed graph

```
#include <stdio.h>
#define SIZEN 50

void BFS(int a[SIZEN][SIZEN],int b[SIZEN],int n)
/*    a[][]: input matrix to represent a graph,not
changed 0/1 matrix
        b[]: output data
        i: # of nodes(vertices) in the graph     */
{
int c[SIZEN]; /* node 0/1 */
int d[SIZEN]; /* queue */
int i,j; /* index */
int queue_h=0; /* queue head */
int queue_t=1; /* queue tail */
int node_i=0; /* node index */

for (i=0;i<n;i++)
    c[i]=0;
for (i=0;i<n;i++)
    d[i]=0;
b[0]=1; c[0]=1; d[0]=1;
while (queue_h!=queue_t) {
    for (i=0;i<n;i++)
        if ((!c[i])&&(a[d[queue_h]-1][i])) {
            c[i]=1;
            d[queue_t]=i+1;
            queue_t++;
            node_i++;
            b[node_i]=i+1;
        }
        d[queue_h]=0;
        queue_h++;
    }
}
```

## Trnansitive Closure of a Directed Graph (logn iterations)

```
#include <stdio.h>
#define    SIZE 50
/* Set the stack size to be greater than the default
(4k bytes). */
/* This declaration must be done in the global data
area.    */
extern unsigned _stklen = 30000U;

void matrix_add(int array[][SIZE],int
record[][SIZE],int n)
{
      int i,j,k,temp[SIZE][SIZE];

        for(i=0;i<n;i++)
            for(j=0;j<n;j++)
    temp[i][j]=record[i][j];

        for(i=0;i<n;i++)
            for(j=0;j<n;j++)
    for(k=0;k<n;k++)
    {
        if(temp[i][j]==1)
        break;

        temp[i][j] |= array[i][k]&array[k][j];
    }
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            array[i][j]=temp[i][j];
}

void transitive(int record[][SIZE],int array[][SIZE],int
n)
  /* record[][]: input matrix to represent a
graph,unchanged
    0/1 matrix
        array[][]: output data,transitive closure of
record[][]
                0/1 matrix
        n: # of nodes (vertices) in the graph
  */
{
    int i,j,k;

        for(i=0;i<n;i++)        /* copy data from record
to array*/
            for(j=0;j<n;j++)
           array[i][j]=record[i][j];

        for(k=1;k<n;k*=2)       /* calculate log n time
*/
        matrix_add(array,record,n);

}
```

## Closet pair of n points on the 2-D plane (divide-and-conquer)

```
#include <iostream>
#include <string>
#include <cmath>
#include <vector>
#include <algorithm>
#include <iomanip>
#include <fstream>
using namespace std;

/* NOTE:
            1. When minimum distance is greater than
10000 ,then output "INFINITY".
        2. Output minimum distance must to be 4
digits followed by the integer.
*/
struct point {
        double x, y;
        point() { x = y = 0.0; }
};

double dist( const point& p1, const point& p2 ) {
        return (p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y)
* (p1.y - p2.y);
}

bool compare_x ( const point& p1,const point& p2 )
{
        return p1.x < p2.x;
}

bool compare_y ( const point& p1,const point& p2 )
{
        return p1.y < p2.y;
}

int min( int a, int b ) {
        if ( a < b )    return a;
        else        return b;
}

double closest_pair( vector< point >& p, point&
close1, point& close2 );

double rec_cl_pair( vector< point >&p, int i, int j,
point& close1, point& close2 );

int main() {
        vector< point > p;
        point ptemp;
            int pointNum;
        fstream in,out;

            in.open("cl_pair.txt",ios::in);
        out.open("cl_pair_dist.txt",ios::out);

            while(in>>pointNum && pointNum!=0){
                for(int i=0;i<pointNum;i++)
             {     in >> ptemp.x >> ptemp.y;
                    p.push_back( ptemp );
                }

        point close1, close2;
        double delta=closest_pair( p, close1, close2 ) ;
            if(delta>=10000 || pointNum==1)
                out << "INFINITY" << '\n';
            else
out<<setiosflags(ios::fixed)<<setprecision(4)<<delta
<<endl;
        p.erase(p.begin(),p.end());
            }
        return 0;
}

double closest_pair( vector< point >& p, point&
close1, point& close2 ) {
    point* start = &p[ 0 ];
    point* end = start + p.size();

    partial_sort( start, end, end, compare_x );
    return rec_cl_pair( p, 0, p.size() - 1, close1,
close2 );
}

double rec_cl_pair( vector< point >&p, int i, int j,
point& close1, point& close2)
{
if ( j - i < 3 ) {//if there are only three points
    sort( &p[ i ], &p[ i ] + ( j - i + 1 ), compare_y );
    if ( j - i == 1 ) {
        close1 = p[ i ];
        close2 = p[ j ];
    }
    else {
        int p1 = i, p2 = i + 1;
        if ( dist( p[ i ], p[ i + 2 ] )
            < dist( p[ p1 ], p[ p2 ] ) )
            p2 = i + 1;
        if ( dist( p[ i + 1 ], p[ i + 2 ] )
            < dist( p[ p1 ], p[ p2 ] ) ) {
            p1 = i + 1;
            p2 = i + 2;
        }
        close1 = p[ p1 ];
        close2 = p[ p2 ];
    }
    return sqrt( dist( close1, close2 ) );
}//recusive divide into two parts(left-right)
int k = ( i + j ) / 2;
int l = p[ k ].x;
point cl1L, cl2L, cl1R, cl2R;
double deltaL, deltaR, deltasq, delta;
deltaL = rec_cl_pair( p, i, k, cl1L, cl2L );
deltaR = rec_cl_pair( p, k + 1, j, cl1R, cl2R );
if ( deltaL < deltaR ) {
    deltasq = deltaL * deltaL;
    close1 = cl1L;
    close2 = cl2L;
}
else {
    deltasq = deltaR * deltaR;
    close1 = cl1R;
    close2 = cl2R;
}
delta = sqrt( deltasq );
vector< point > v( j - i + 1 );
merge( &p[ i ], &p[ k ] + 1, &p[ k ] + 1, &p[ j ] + 1,
        v.begin(), compare_y );

int t;
for ( t = i; t <= j; t++ )
    p[ t ] = v[ t - i ];
t = -1;
for ( k = i; k <= j; k++ )
    if ( p[ k ].x > l - delta && p[ k ].x < l + delta )
        v[ ++t ] = p[ k ];
/*finally check if there exists the minimum distance
between the central line
    within current minimum distance(compare nearly
6 points each point)*/
int s;
float dtemp;
for ( k = 0; k < t; k++ )
    for ( s = k + 1; s <= min( t, k + 7 ); s++ ) {
        dtemp = dist( v[ k ], v[ s ] );
        if ( dtemp < deltasq ) {
            deltasq = dtemp;
            close1 = v[ k ];
            close2 = v[ s ];
        }
    }
return sqrt( deltasq );
}
```

## Convert binary code to Gray code with 0/1 bit string

```
#include <stdio.h>
#define SIZE 50

int bi_gray(char b[],int g[])
/*    b[]: given binary code in string
      g[]:corresponding graycode in array
      return (n): # of bits of the gray codes
*/
{
int i,n;
int temp[SIZE];
for(n=0;b[n]!='\0';n++)
    temp[n]=(int)(b[n]-'0');    //convert char to int

g[0]=temp[0];
for(i=1;i<n;i++)
    g[i]=(temp[i-1]^temp[i]);

return(n);
}
```

## Convert Gray code to binary code with 0/1 bit string

```
#include <stdio.h>
#define SIZE 50

int gray_bi(char g[],int b[])
/*    g[]: given graycode in string
      b[]:corresponding binary code in array
      return (n):# of bits of the binary codes
*/
{
    int i,n;
    int temp[SIZE];

        for(n=0;g[n]!='\0';n++)
            temp[n]=(int)(g[n]-'0');
        b[0]=temp[0];
        for(i=1;i<n;i++)
            b[i]=(b[i-1]^temp[i]);
        return(n);
}
```

## Huffman code of radix k (k-ary tree), k>=2

```
#include <stdlib.h>
#define N          100                  // maximun # of
elements
#define SIZEN 100                  // maximun length of
code
void huffman(const int num[], const int n, const int
radix,
                    char code[N][SIZEN])
/*
    Author    : Jong-rong Shyy
    Subject : Huffman code
    Input      :
        num       : # of every element
        n          : elements of num
        radix : radix
        code     : result of encoded (string format)
    Output    :
        return value : none
*/
{
    typedef struct NODE
    {
        int freq;
        char sym, count;
        struct NODE *father;
    } Node;

    Node node[N * 2], *tmp;
    int newn, a, b, i, j, k, sw;
    int max[N][2], m;
    char cd[SIZEN];

    for (i = 0; i < n; i++)
    {
        node[i].freq = num[i];
        node[i].father = NULL;
    }
    for (newn = n; ; newn++)
        if ((newn - radix) % (radix - 1) == 0)
            break;
    for ( ; i < newn; i++)
    {
        node[i].freq = 0;
        node[i].father = NULL;
    }

    for (i = newn; ; i++)
    {
        m = 0;
        for (j = 0; j < i; j++)
            if (node[j].father == NULL)
            {
                max[m][0] = node[j].freq;
                max[m++][1] = j;
            }
        if (m < radix)
            break;

        for (a = m - 1; a >= 0; a--)
        {
            sw = 0;
            for (b = 0; b < a; b++)
                if (max[b][0] > max[b + 1][0])
                {
                    sw = 1;
                    k = max[b][0];
                    max[b][0] = max[b + 1][0];
                    max[b + 1][0] = k;
                    k = max[b][1];
                    max[b][1] = max[b + 1][1];
                    max[b + 1][1] = k;
                }
            if (sw == 0)
                break;
        }

        node[i].freq = 0;
        node[i].father = NULL;
        node[i].count = '0';

        for (j = 0; j < radix; j++)
        {
            node[max[j][1]].father = &node[i];
            node[max[j][1]].sym = node[i].count++;
            node[i].freq += node[max[j][1]].freq;
        }
    }

    for (i = 0; i < n; i++)
    {
        m = 0;
        for (tmp = &node[i]; ; tmp = tmp->father,
m++)
        {
            if (tmp->father == NULL)
                break;
            cd[m] = tmp->sym;
            if (cd[m] > '9')
                cd[m] += 10 - 'A';
        }
        for (j = 0, k = m - 1; j < m; j++, k--)
            code[i][j] = cd[k];
        code[i][j] = '\0';
    }
}
```

## Get combination of a given sequence # in C(n, k)

```
/*      count from 0, e,g, in C(4,3), all combinations
are, abc, abd, acd, bcd. Thus, the sequence # of acd
is 2 */
/* PS : need initiation (call combine1(0, 0, NULL,
NULL, 0)*/
#include <stdio.h>
typedef char Type;              // data type of elements
void combine1(int n, int k, Type src[32], Type
ret[32], int seq)
/*      Input       :
            n            : # of total elements
            k            : # of picked elements
            src         : elements, i.e. seq = 0;
            ret         : the seq-th permutation of src
            seq         : the sequence #
        Output     :
            return value : none
*/
{
    static int c[32][32];
    int i, j;
    int r, p;
    int tn, tk;

    if (src == NULL && ret == NULL) {
            n=32;
            for(i=1;i<n;i++) {
                    c[i][1]=i;
                    c[i][0]=1;
            }
            for(i=0;i<n+1;i++)
                    c[i][i]=1;
            for(j=2;j<n+1;j++)
```

```
                for(i=j+1;i<n+1;i++)
                        c[i][j]=c[i-1][j]+c[i-1][j-1];
        return;
    }
    tn=n-1, tk=k-1;
    r = seq+1;
    p=0;
    for (i=0; i < k; i++)    {
        while(c[tn][tk]<r) {
                r -= c[tn][tk];
                tn--;
                p++;
        }
        ret[i] = src[p];
        tn--;
        tk--;
        p++;
    }
}
```

## Get sequence # of a combination

```
/* get the sequence # of a combination in C(n, k)
combinations */
/*      count from 0, e,g, in C(4,3), all combinations
are, abc, abd, acd, bcd. Thus, the sequence # of acd
is 2 */
/* PS : need initiation (call combine2(0, 0, NULL,
NULL, 0)*/
#include <stdio.h>
typedef char Type;              // data type of elements
int combine2(int n, int k, Type src[], Type des[])
/*      Input       :
            n           : # of total elements
            k           : # of picked elements
            src         : permutation of sequence # 0
            des         : the problem
        Output      :
            return value : sequence #
*/
{
    static int c[32][32];
    int i, j;
    int sum, p;
    int tn, tk;

    if (src == NULL && des == NULL) {
        for(i=1;i<32;i++) {
                c[i][1]=i;
                c[i][0]=1;
        }
        for(i=0;i<32;i++)
                c[i][i]=1;
        for(j=2;j<32;j++)
                for(i=j+1;i<32;i++)
                        c[i][j]=c[i-1][j]+c[i-1][j-1];
        return 0;
    }
    tn=n-1, tk=k-1;
    p=0;
    sum=0;
    for (i=0; i < k; i++)    {
        while(src[p]<des[i]) {
                sum += c[tn][tk];
                tn--;
                p++;
        }
        tn--;
        tk--;
        p++;
    }
    return sum;
}
```

## Permutation of a given sequence # in P(n, k) permutations

```
/*      count from 0, e,g, in P(4, 3), the sequence # of
acb is 2*/
/* PS : need initiation (call permute1(0, 0, NULL,
NULL, 0)*/
#include <stdio.h>
typedef char Type;              // data type of elements
void permute1(int n, int k, Type src[], Type ret[], int
seq)
/*      Input       :
            n           : # of total elements
            k           : # of picked elements
            src         : elements, i.e. seq = 0;
            ret         : the seq-th permutation of src
            seq         : the sequence #
        Output      :
            return value : none
*/
{
    static int c[32][32], nf[13];
    int tmp[32];
    int a, b, s;
    int q, r, mu;

    if (src == NULL && ret == NULL)
    {
        nf[0] = 1;
        for (a = 1; a < 13; a++)
            nf[a] = a * nf[a - 1];
        for (a = 0; a < 32; a++)
            c[a][0] = 1;
        for (a = 1; a < 32; a++)
            for (b = 1; b <= a; b++)
                c[a][b] = c[a - 1][b] + c[a - 1][b - 1];
        return;
    }
    for (a = 0; a < n; a++)
        tmp[a] = 1;
    r = seq;
    for (a = 0; a < k; a++) {
        mu = c[n - a - 1][k - a - 1] * nf[k - a - 1];
        q = r / mu;
        r = r % mu;
        for (b = 0, s = -1; b < n; b++)
            if (tmp[b] == 1)
                if (++s == q) {
                    tmp[b] = 0;
                    break;
                }
        ret[a] = src[b];
    }
}
```

## Get sequence # of a permutation in P(n, k) permutations

```
/* count from 0, e,g, in P(4, 3), the sequence # of
acb is 2*/
/* PS : need initiation (call permute2(0, 0, NULL,
NULL)*/
#include <stdio.h>
typedef char Type;
int permute2(int n, int k, Type src[], Type des[])
/*      Input       :
            n           : # of total elements
            k           : # of picked elements
            src         : permutation of sequence # 0
            des         : the problem
        Output      :
            return value : sequence #
*/
{
    static int c[32][32], nf[13];
    int i, j, s, sum;
    int num[32];
    if (src == NULL && des == NULL)    {
        nf[0] = 1;
        for (i = 1; i < 13; i++)
            nf[i] = i * nf[i - 1];
        for (i = 0; i < 32; i++)
            c[i][0] = 1;
        for (i = 1; i < 32; i++)
            for (j = 1; j <= i; j++)
                c[i][j] = c[i - 1][j] + c[i - 1][j - 1];
        return 0;
    }
    for (i = 0; i < n; i++)
        num[i] = 1;
    sum = 0;
    for (i = 0; i < k; i++)    {
        s = -1;
        for (j = 0; j < n; j++)
            if (num[j] == 1)    {
                s++;
                if (des[i] == src[j]) {
                    num[j] = 0;
                    break;
                }
            }
        sum += s * c[n - i - 1][k - i - 1] * nf[k - i - 1];
    }
    return sum;
}
```

## N queen problem

```
//2014/10/13  提供者：翁丞世、林敬哲、林必祥
#include <stdio.h>
main() {
   int n, i, odd;
   for (; scanf("%d", &n) == 1;)
      if (n < 4) printf("Impossible\n");
      else
         if ((n / 2) % 3 != 1) {
            printf("2");
            for (i = 4; i <= n; i += 2)
               printf(" %d", i);
            for (i = 1; i <= n; i += 2)
               printf(" %d", i);
            printf("\n");
         }
      else {
         if (n & 1) n--, odd = 1; else odd = 0;
         printf("%d", n / 2);
         for (i=n/2+1; i!=n/2-1; i=(i+2)% n)
            printf(" %d", i + 1);
         for (i = (i+n-2)%n; i!=n/2-1; i=(i+n-2)%n)
            printf(" %d", n - i);
         printf(" %d", n - i);
         if (odd) printf(" %d", n + 1);
         printf("\n");
      }
   return 0;
}
```