1. Code Templates

1.1. Optimization of cin and cout

```
#define optimizar_io ios_base::sync_with_stdio(0);cin.tie(0);
```

1.2. Java Template

```
import java.util.*;
import java.math.*;
import java.io.*;
public class Main {
    public static void main(String[] args) throws IOException {
          Scanner cin = new Scanner(System.in);
          int a, b;
          b = cin.nextInt(); a = cin.nextInt();
         System.out.printf("%d", a+b);
          cin.close();
    }
}
```

2. Number Theory

2.1. Formulas

Number Catalan:

```
C[n] => FOR(k=0,n-1) C[k] * C[n-1-k]
C[n] => Comb(2*n,n) / (n + 1)
C[n] => C[n-1]*(4*n-2)/(n+1)
```

Euler's formula:

```
A + C = V + 2
```

Desarranjo:

```
d(1) = 0, d(2) = 1
d(n) = (n-1)*(d(n-1) + d(n-2))
```

2.2. Inverso modular de N!

```
ifact[n+1] = fact[n+1]^(mod-2)
ifact[n] = (ifact[n+a]*(i+1))%mod;
```

2.3. Modular Multiplication of big numbers

```
inline ll mulmod(ll a, ll b, ll m) {
    ll x = 0, y = a % m;
    while (b > 0) {
          if (b % 2 == 1)    x = (x + y) % m;
          y = (y * 2) % m;
          b /= 2;
    }
    return x;
}
```

2.4. Rabin-Miller

```
//using: mulmod( ), powmod()
bool suspect(ll a, int s, ll d, ll n) {
   ll x = powMod(a, d, n);
   if (x == 1)  return true;
   for (int r = 0; r < s; ++r) {
         if (x == n - 1)    return true;
         x = mulmod(x, x, n);
   }
   return false;
}
// {2,7,61,0}                 is for n < 4759123141 (= 2^32)
// {2,3,5,7,11,13,17,19,23,0} is for n < 10^16 (at least)
unsigned test[] = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 0 };
bool miller_rabin(ll n) {
   if (n <= 1 || (n > 2 && n % 2 == 0)) return false;
   ll d = n - 1; int s = 0;
   while (d % 2 == 0) ++s, d /= 2;
   for (int i = 0; test[i] < n && test[i] != 0; i++)
         if (!suspect(test[i], s, d, n))
               return false;
   return true;
}
```

2.5. Pollard-Rho

```
// Randomized Factorization Algorithm O(sqrt(s(n))) expected
// where s(n) is the smallest prime divisor of n
// use in conjuntion with miller_rabin test for primality
//using: Rabin-Miller(), mulmod()
#define func(x)(mulmod(x, x+B, n)+ A )
ll pollard_rho(ll n) {
  if( n == 1 ) return 1;
  if( miller_rabin( n ) )
     return n;
  ll d = n;
  while( d == n ){
    ll A = 1 + rand()%(n-1), B = 1 + rand()%(n-1);
    ll x = 2, y = 2;
    d = -1;
    while( d == 1 || d == -1 ){
        x = func(x), y = func(func(y));
        d = __gcd( x-y, n );
    }
  }
```

```
20.   return abs(d);
21. }
```

2.6. Extended GCD(ax+by = d)

```
1.  //devuelve x,y tal que ax+by = gcd(a,b)
2.  int64 extended_euclid(int64 a, int64 b, int64& x, int64& y) {
3.    int64 g = a;
4.    x = 1; y = 0;
5.    if ( b != 0 ) {
6.      g = extended_euclid( b, a % b, y, x );
7.      y -= ( a / b ) * x;
8.    }
9.    return g;
10. }
```

2.7. Inverso Modular

```
1.  //using: Extended GCD
2.  int inverso_mod(int n ,int m){
3.      int s, t, d;
4.     d = extended_euclid( n, m, s, t );
5.     return ((s % m)+m)% m;
6.  }
```

2.8. Teorema del Resto Chino

```
1.  //using: Inverso Modular
2.  int resto_chino (int x[], int m[], int k){
3.      int i, tmp, MOD = 1, RES = 0;
4.      for (i=0; i <k ; i++) MOD *= m[i];
5.      for (i =0; i <k ; i++){
6.          tmp = MOD/m[i];
7.          tmp *= inverso_mod(tmp, m[i]);
8.          RES += (tmp*x[i]) % MOD;
9.      }
10.     return RES % MOD;
11. }
```

2.9. Modular Equations (ax(n)=b(n))

```
1.  //using: Extended GCD
2.  int modular_equations( int a, int b, int n ){
3.   int s, t, x0;
4.   int d = extended_euclid( a, n, s, t );
5.   if( b % d == 0 ){
6.      int tmp = (b/d)*s;
7.      x0 = ((tmp%n)+n)%n; // x0 = s*b
8.      return x0; // x = x0+(n/d)*k, 0 < k < d
9.   }
10.  return -1;
11. }
```

2.10. Find a primitive root of a prime number

```
1.  // O( log^6(p)*sqrt(p) ), using: powmod()
2.  int generator (int p){
3.      vector<int> fact;
4.      int phi = p-1, n = phi;
5.      for (int i=2; i*i<=n; ++i)
6.          if (n % i == 0){
7.              fact.push_back (i);
8.              while (n % i == 0)
9.                  n /= i;
10.         }
11.     if (n > 1) fact.push_back (n);
12.     for (int res=2; res<=p; ++res){
13.         bool ok = true;
14.         for (size_t i=0; i<fact.size() && ok; ++i)
15.             ok &= powmod (res, phi / fact[i], p) != 1;
16.         if (ok) return res;
17.     }
18.     return -1;
19. }
```

2.11. Algoritmo Shanka-Tonelli(x^2 = a(mod p))

```
1.  //using: powmod()
2.  long long solve_quadratic( long long a, int p ){
3.      if( a == 0 ) return 0;
4.      if( p == 2 ) return a;
5.      if( powMod(a,(p-1)/2, p) != 1 ) return -1;
6.      int phi = p-1, n = 0, k = 0, q = 0;
7.      while( phi%2 == 0 ) phi/=2, n ++;
8.      k = phi;
9.      for( int j = 2; j < p; j ++ )
10.         if( powMod( j, (p-1)/2, p ) == p-1 ){
11.           q = j; break;
12.         }
13.     long long t = powMod( a, (k+1)/2, p );
14.     long long r = powMod( a, k, p );
15.     while( r != 1 ){
16.         int i = 0, v = 1;
17.         while( powMod( r, v, p ) != 1 ) v *= 2, i ++;
18.         long long e = powMod( 2, n-i-1, p );
19.         long long u = powMod( q, k*e, p );
20.         t = (t*u)%p;
21.         r = (r*u*u)%p;
```

```
    }
    return t;
}
```

2.12. Shanks' Algorithm(a^x = b(m))

```
// O(sqrt(m)), return x such that a^x = b mod m
int solve ( int a, int b, int m ){
    int n = (int)sqrt( m + .0 )+1, an = 1;
    for ( int i = 0; i < n; i++ )
        an = (an * a)%m;
    map<int, int>vals;
    for ( int i = 1, cur = an; i <= n; ++ i ){
        if ( ! vals. count ( cur ) )
            vals [ cur ] = i ;
        cur = (cur * an)%m;
    }
    for ( int i = 0, cur = b; i <= n; ++ i ){
        if ( vals. count ( cur ) ){
            int ans = vals [ cur ] * n - i ;
            if ( ans < m )return ans;
        }
        cur = (cur * a)%m;
    }
    return -1;
}
```

2.13. FFT sin complex

```
///----FFT-O(n*log(n))-----
const int MOD = 167772161;
// so the algorithm works until n = 2 ^17 = 131072
const int g = 3; // primitive root
//const int MOD = 1073872897 = 2^30 + 2^17 + 1, g = 7
//another good choice is MOD=167772161=2^27+2^25+1, g = 3
//a bigger choice would be MOD=3221225473=2^31+2^30+1, g = 5
// but it requires unsigned long long for multiplications
typedef long long i64;
// n must be a power of two
// sign = 1, scale = 1 for DFT
// sign = -1, scale=(1/n)(MOD) or (MOD-(MOD-1)/n) for inverse
//using: powmod()
void ifft(int n, i64 a[],int sign, i64 scale) {
  int k;
  for (k = 0; (1 << k) < n; k++);
  for (int i = 0; i < n; i++) {
      int q = 0;
      for (int j = 0; j < k; j++) {
          q <<= 1;
          if (i & 1 << j)     q++;
      }
      if (i < q) swap(a[i], a[q]);
  }
  i64 x = powmod(g, (MOD - 1) / n, MOD);
  for (int q = 2; q <= n; q <<= 1) {
      int q2 = q / 2;
      i64 wn = powmod(x, n + sign * n / q, MOD);
      i64 w = 1;
      for (int i = 0; i < q2; i++) {
          for (int j = i; j < n; j += q) {
              int v = w * a[j + q2] % MOD;
              a[j + q2] = (a[j] - v + MOD) % MOD;
              a[j] = (a[j] + v) % MOD;
          }
          w = i64(w) * wn % MOD;
      }
  }
  for (int i = 0; i < n; i++) a[i] = a[i] * scale % MOD;
}
```

2.14. FFT con complex

```
#define PI 2*acos(0)
typedef complex<double> base;
void fft (vector<base> & a, bool invert) {
  int n = (int) a.size();
  for (int i=1, j=0; i<n; ++i) {
      int bit = n >> 1;
      for (; j>=bit; bit>>=1)
          j -= bit;
      j += bit;
      if (i < j) swap (a[i], a[j]);
  }
  for (int len=2; len<=n; len<<=1) {
      double ang = 2*PI/len * (invert ? -1 : 1);
      base wlen (cos(ang), sin(ang));
      for (int i=0; i<n; i+=len) {
          base w (1);
          for (int j=0; j<len/2; ++j) {
              base u = a[i+j],  v = a[i+j+len/2] * w;
              a[i+j] = u + v;
              a[i+j+len/2] = u - v;
              w *= wlen;
  }   }   }
```

```
23.    if (invert){
24.          for (int i=0; i<n; ++i)
25.                a[i] /= n;}
26. }
27. void multiply(vector<int>&a, vector<int>&b,vector<int> &res){
28. vector<base> fa (a.begin(), a.end()),  fb(b.begin(), b.end());
29.    size_t n = 1;
30.    while (n < max (a.size(), b.size()))  n <<= 1;
31.    n <<= 1;
32.    fa.resize (n),  fb.resize (n);
33.    fft (fa, false),  fft (fb, false);
34.    for (size_t i=0; i<n; ++i)
35.          fa[i] *= fb[i];
36.    fft (fa, true);
37.    res.resize (n);
38.    for (size_t i=0; i<n; ++i)
39.          res[i] = int (fa[i].real() + 0.5);
40. }
```

2.15. Phi

```
1.  // computes the number of coprimes of p^k, being p prime
2.  //int phi(int p,int k){return pow(p,k)-pow(p,k-1);}//phi(p^k)
3.  int phi(int p,int pk){return pk-(pk/p);}//phi(p^k)where pk=p^k
4.  // computes the number of coprimes of n
5.  // phi(n) = (p_1-1)*p_1^(k_1-1)*(p_2-1)*p_2^(k_2-1)
6.  int phi(int n){
7.      int coprimes = (n != 1); // phi(1) = 0
8.      for (int i = 2; i*i <= n; i++)
9.          if (n%i == 0){
10.             int pk = 1;
11.             while (n%i == 0)
12.                 n /= i, pk *= i;
13.             coprimes *= phi(i, pk);
14.         }
15.     if (n > 1) coprimes *= phi(n, n); // n is prime
16.     return coprimes;
17. }
```

2.16. Brent's Algorithm (Cycle detection)

```
1.  //O(u+l) Fμ(x0) = Fμ+l(x0)
2.  par cycle_detaction( ){
3.     int p = 1, l = 1, t = x0, h = f(x0), u;
4.     while (t != h){
5.           if (p == l) t = h, p*= 2, l = 0;
6.           h = f(h), ++l;
7.     }
8.     u = 0, t = h = x0;
9.     for (i = l; i != 0; --i) h = f(h);
10.    while (t != h) t = f(t), h = f(h), ++u;
11.    return par( u, l );
12. }
```

2.17. Floyd's Cycle-Finding algorithm

```
1.  par find_cycle() {
2.      int t = f(x0), h = f(t), mu = 0, lam = 1;
3.      while (t != h) t = f(t), h = f(f(h));
4.      h = x0;
5.      while (t != h) t = f(t), h = f(h), mu++;
6.      h = f(t);
7.      while (t != h) h = f(h), lam++;
8.      return par(mu, lam);
9.  }
```

2.18. Matrix Exponentiation

```
1.  // O( n^3*log(n) )
2.  typedef vector <int> vect;
3.  typedef vector < vect > matrix;
4.  matrix identity (int n) {
5.     matrix A(n, vect(n));
6.     for (int i = 0; i <n; i++) A[i][i] = 1;
7.     return A;
8.   }
9.  matrix mul(const matrix &A, const matrix &B) {
10.   matrix C(A.size(), vect(B[0].size()));
11.   for (int i = 0; i < C.size(); i++)
12.     for (int j = 0; j < C[i].size(); j++)
13.       for (int k = 0; k < A[i].size(); k++)
14.         C[i][j] += A[i][k] * B[k][j];
15.   return C;
16. }
17. matrix pow(const matrix &A, int e) {
18.   return ( e == 0 ) ? identity(A.size()) :
19.          ( e%2 == 0 )? pow(mul(A,A),e/2) : mul(A,pow(A,e-1));
20. }
```

2.19. Fast Square Testing

```
1.  long long M;
2.  void init_is_square(){
3.      rep(i,0,64) M |= 1ULL << (63-(i * i)%64);
4.  }
```

```
5.  inline bool is_square(long long x) {
6.      if ((M << x) >= 0) return false;
7.      int c =__builtin_ctz(x);
8.      if (c & 1) return false;
9.      x >>= c;
10.     if ((x&7) - 1) return false;
11.     long long r = sqrt(x);
12.     return r*r == x;
13. }
```

3. String

3.1. Hashing

```
1.  unsigned long long calc_hash( int ptr, int in, int f ){
2.     return Dp[f] - Dp[in-1]*pot33[f-in+1];
3.  }
4.  void buil_hash(  ){
5.     for( int i = 1; i <= ta; i ++ )
6.           Dp[i] = Dp[i-1]*33LL + ( A[i] - 'a' );
7.  }
```

3.2. Aho Corasick

```
1.  const int alph = 26;
2.  struct tree {
3.      int parent, slink;
4.      bool band;
5.      int hij[30];
6.      tree( int p ){
7.          parent = p, slink = 0, band = false;
8.          fill( hij, hij + 30, -1 );
9.      }
10. };
11. vector<tree> trie;
12. void addWord( string s1 ){
13.     int root = 0;
14.     for( int i = 0; i < (int)s1.length(); i ++ ){
15.         if( trie[root].hij[s1[i] - 'a'] == -1 ){
16.             trie[root].hij[s1[i] - 'a'] = trie.size();
17.             trie.push_back( tree( root ) );
18.         }
19.         root = trie[root].hij[s1[i] - 'a'];
20.     }
21.     trie[root].band = true;
22. }
23. queue<int> Q;
24. void buildSuffixLinks(  ){
25.     int nod, nextC;
26.     Q.push( 0 );
27.     Q.push( 0 );
28.  while( !Q.empty() ){
29.     nod = Q.front(), Q.pop();
30.     nextC = Q.front(), Q.pop();
31.     for( int i = 0; i <= alph; i ++ )
32.         if( trie[nod].hij[i] != -1 ){
33.             Q.push( trie[nod].hij[i] );
34.             Q.push( i );
35.         }
36.     if( nod == 0 || trie[0].hij[nextC] == nod )
37.         continue;
38.     int& link = trie[nod].slink;
39.     link = trie[trie[nod].parent].slink;
40.     while( link != 0 && trie[link].hij[nextC] == -1 )
41.         link = trie[link].slink;
42.     link = trie[link].hij[nextC];
43.     if( link == -1 )
44.         link ++;
45.     if( trie[link].band )
46.         trie[nod].band = true;
47.  }
48. }
49. int go( int nod, char c ){
50.     if( nod == 0 )
51.         return trie[0].hij[c - 'a'];
52.     if( trie[nod].hij[c - 'a'] != -1 )
53.         return trie[nod].hij[c - 'a'];
54.     int link = trie[nod].slink;
55.     while( link != 0 && trie[link].hij[c-'a'] == -1 )
56.         link = trie[link].slink;
57.     return trie[link].hij[c-'a'];
58. }
59. int automata[10005][30], N, M;
60. void Aho_Corasick( ){
61.     string tmp;
62.     cin >> N >> M;
63.     trie.clear();
64.     trie = vector<tree> ( 1, tree(0) );
65.     for( int i = 1; i <= M; i ++ ){
66.         cin>>tmp;
67.         addWord( tmp );
68.     }
69.     buildSuffixLinks( );
```

```
70.     for( int j = 0; j < (int)trie.size(); j ++ )
71.         for( int h = 'a'; h <= 'z'; h ++ )
72.             automata[j][h-'a'] = go( j, h );
73. }
```

3.3. Manacher

```
1.  int rad[ 2 * MAXLEN ], n;
2.  char s[MAXLEN];
3.  void manacher( ){ /// i%2!=0 par, i%2==0 par
4.    int i, j, k;
5.    for ( i = 0, j = 0; i < 2 * n - 1; i += k ) {
6.      while ( i - j >= 0 && i + j + 1 < 2 * n &&
7.              s[ ( i - j ) / 2 ] == s[ ( i + j + 1 ) / 2 ] )
8.                j++;
9.     rad[i] = j;
10.     for ( k = 1; k <= rad[i] && rad[i-k] != rad[i]-k; k++ )
11.       rad[ i + k ] = min( rad[ i - k ], rad[i] - k );
12.    j = max( j - k, 0 );
13.   }
14. }
```

3.4. Suffix Array O(nlog(n))

```
1.  #define MN 200005
2.  int N,in[305],prox[MN],sa[MN],k;
3.  int cant[MN],pos[MN],lcp[MN],may,s1;
4.  char A[MN];
5.  bool b1[MN], b2[MN];
6.  void LCP(  ){
7.     for( int p = 0, i = 0; i < N; i ++ )
8.           if( pos[i] != N - 1 ){
9.                 for( int j = sa[pos[i]+1]; i + p <= N &&
10.                  j + p <= N && A[i+p] == A[j+p]; p ++ );
11.                      lcp[pos[i]] = p;
12.                if( p ) p --;
13.          }
14. }
15. inline void upper( int x ){
16.    int p = pos[x];
17.    pos[x] = p + cant[p];
18.    cant[p] ++;
19.    b2[pos[x]] = true;
20. }
21. void Suffix_Array( ){
22.    fill( in, in + 300, -1 );
23.    for( int i = 0; i < N; i ++ )
24.          prox[i] = in[(int)A[i]], in[(int)A[i]] = i;
25.    for( int i = 'a'; i <= 'z'; i ++ )
26.          for( int j = in[i]; j != -1; j = prox[j] ){
27.                sa[k] = j;
28.                if( j == in[i] ) b1[k] = true;
29.                k ++;
30.          }
31.    int p;
32.    for( int H = 1; H < N; H *= 2 ){
33.          fill( b2, b2 + N + 1, false );
34.          for( int i = 0; i < N; i = k ){
35.                for( k = i+1; k < N && !b1[k]; k ++ );
36.                      cant[i] = 0;
37.                for( int j = i; j < k; j ++ )
38.                      pos[sa[j]] = i;
39.          }
40.          upper( N - H );
41.          for( int i = 0; i < N; i = k ){
42.                for( k = i+1; k < N && !b1[k]; k ++ );
43.                for( int j = i; j < k; j ++ )
44.                      if( sa[j] - H >= 0 )
45.                            upper( sa[j] - H );
46.                for( int j = i; j < k; j ++ )
47.                      if( sa[j]-H >= 0 && b2[pos[sa[j]-H]] ){
48.                            for( p = pos[sa[j] - H] + 1; p < N
49.                              && !b1[p] && b2[p]; p ++ )
50.                                  b2[p] = false;
51.                      }
52.          }
53.          for( int i = 0; i < N; i ++ ){
54.                sa[pos[i]] = i;
55.                b1[i] = ( b1[i] || b2[i] );
56.          }
57.    }
58.    LCP(  );
59. }
```

3.5. Z-Algorithm

```
1.  void Z_algorithm( ){
2.    int L = 0, R = 0, k;
3.    for (int i = 1; i < n; i++){
4.     if( i <= R && z[i-L] < R-i+1 )
5.      z[i] = z[i-L];
6.     else{
7.      L = i, R = max( R, i );
```

```
8.      while( R < n && s[R-L] == s[R] )
9.       R ++;
10.     z[i] = R - L;
11.     R --;
12.    }
13.   }
14. }
```

3.6. Decomposition of lyndon

```
1.  //decomposition of lyndon s= w1w2w3..wk, w1 >= w2 >=...>= wk
2.  void lyndon( string s ){
3.      int n = (int)s.length(), i = 0;
4.      while( i < n ){
5.          int j = i+1, k = i;
6.          while( j < n && s[k] <= s[j] ){
7.              if( s[k] < s[j] ) k = i;
8.              else k ++;
9.              j ++;
10.         }
11.         while( i <= k ){
12.             cout << s.substr( i, j-k )<<endl;
13.             i += j-k;
14.         }
15.     }
16. }
```

3.7. LCS

```
1.  int lcs ( ){
2.      for ( int i = 1; i <= t1; i ++ )
3.         for ( int j = 1; j <= t2; j ++ )
4.             if ( cad1[i] == cad2[j] )
5.                Dp[i][j] = Dp[i-1][j-1]+1, P[i][j] = 'D';
6.             else
7.                if ( Dp[i-1][j] > Dp[i][j-1] )
8.                   Dp[i][j] = Dp[i-1][j], P[i][j] = 'I';
9.                else
10.                 Dp[i][j] = Dp[i][j-1], P[i][j] = 'S';
11.     return Dp[t1][t2];
12. }
```

3.8. Edit Distant

```
1.  int Edit_Dist( ){
2.   for(int i = 0; i <= max(la, lb); i ++)
3.    C[0][i] = i, C[i][0] = i;
4.
5.   for(int i = 1; i <= la; i ++)
6.    for(int j = 1; j <= lb; j ++)
7.     if(A[i] == B[j] && C[i - 1][j - 1] != 1 << 30)
8.      C[i][j] = C[i - 1][j - 1];
9.     else{
10.     C[i][j] = 1 << 30;
11.     C[i][j] = min(C[i][j], C[i - 1][j] + 1);
12.     C[i][j] = min(C[i][j], C[i][j - 1] + 1);
13.     C[i][j] = min(C[i][j], C[i - 1][j - 1] + 1);
14.    }
15.    return C[la][lb];
16. }
```

3.9. KMP

```
1.  void pre_kmp( ){
2.   for(int j = 0, i = 2; i <= tp; i ++){
3.    while( j && P[j + 1] != P[i]) j = fall[j];
4.    if(P[j + 1] == P[i]) j ++;
5.    fall[i] = j;
6.   }
7.  }
8.  void kmp(  ){
9.   for(int j = 0, i = 1; i <= tt; i ++){
10.   while( j && P[j + 1] != T[i]) j = fall[j];
11.   if(P[j + 1] == T[i]) j ++;
12.   if(j == tp) printf("%d\n", i - tp + 1);
13.  }
14. }
```

3.10. Lex-Rot

```
1.  int lexRot(string str){
2.      int n = str.size(), ini=0, fim=1, rot=0;
3.      str += str;
4.      while(fim < n && rot+ini+1 < n)
5.          if (str[ini+rot] == str[ini+fim]) ini++;
6.          else if(str[ini+rot]<str[ini+fim])fim+=ini+1, ini = 0;
7.          else rot = max(rot+ini+1, fim), fim = rot+1, ini = 0;
8.      return rot;
9.  }
```

4. Graphs

4.1. Struct edges

```
1.  int pos, Index[10005];///index = -1
2.  struct edges{
3.   int nod, newn, cap, cost, next;
```

```
4.    bool band;
5.    edges(int a = 0, int b = 0, int c = 0, int d = 0, int e = 0){
6.     nod = a, newn = b, cap = c, cost = d, next = e;
7.    }
8.    int nextn ( int a ){
9.     if( nod == a )
10.     return newn;
11.    return nod;
12.   }
13. }G[100005];
14. ///nod, newn, cap, cost
15. void insertar( int a, int b, int c, int d = 0 ){
16.   G[pos] = edges( a, b, c, d, Index[a] );
17.   Index[a] = pos ++;
18.   G[pos] = edges( b, a, 0, -d, Index[b] );
19.   Index[b] = pos ++;
20. }
```

4.2. Bellman Ford

```
1.  double Bellman_Ford(  ){
2.    double newc;
3.    int nod, newn;
4.    fill( dist + 1, dist + 1 + N, maxint );
5.    fill( parent + 1, parent + 1 + N, -1 );
6.    dist[In] = D, parent[In] = 1 << 30;
7.    for( int i = 1; i < N; i ++ )
8.     for( int j = 1; j <= M; j ++ ){
9.        nod = G[j].nod, newn = G[j].newn;
10.       newc = dist[nod] * G[j].cost;
11.       if( dist[newn] > newc ){
12.        dist[newn] = newc;
13.        parent[newn] = nod;
14.       }
15.      }
16.   if( parent[Fin] == -1 )
17.    return 0;
18.   for( int j = 1; j <= M; j ++ ){
19.       nod = G[j].nod, newn = G[j].newn;
20.       newc = dist[nod] * G[j].cost;
21.       if( dist[newn] > newc )
22.          return 0;//se encontro un ciclo negativo
23.      }
24.    return dist[Fin];
25. }
```

4.3. LCA

```
1.  void LCA( ){
2.     lv[1] = 1, Q.push( 1 );
3.     int logg, nod, newn, t;
4.     mark[1] = true;
5.     while( !Q.empty() ){
6.           nod = Q.front();
7.           Q.pop();
8.           t = V[nod].size();
9.           for(int i = 0; i < t; i ++){
10.                newn = V[nod][i];
11.                if( mark[newn] ) continue;
12.                Q.push( newn );
13.                lv[newn] = lv[nod] + 1;
14.                Dp[newn][0] = nod;
15.                logg = log2( lv[newn] );
16.                for(int j = 1; j <= logg; j ++)
17.                      if( Dp[newn][j - 1] )
18.                           Dp[newn][j]=Dp[Dp[newn][j-1]][j-1];
19.          }
20.    }
21. }
22. int ancestro( int a, int b ){
23.    if( lv[a] < lv[b] ) swap( a, b );
24.    int logg = log2( lv[a] );
25.    for( int i = logg; i >= 0; i -- )
26.          if(lv[a] - ( 1 << i) >= lv[b] && Dp[a][i])
27.                a = Dp[a][i];
28.    if( a == b ) return a;
29.    for(int i = logg; i >= 0; i --)
30.          if(Dp[a][i] != Dp[b][i] && Dp[a][i])
31.                a = Dp[a][i], b = Dp[b][i];
32.    return Dp[a][0];
33. }
```

4.4. Bridges y Punto de Articulacion

```
1.  void bridges_PtoArt ( int nod ){
2.       int newn, num;
3.       vector<int>::iterator it;
4.       Td[nod] = low[nod] = ++ k;
5.       for(it = V[nod].begin(); it != V[nod].end(); it ++){
6.           num = *it;
7.           newn = G[num].nextn( nod );
8.           if( G[num].band ) continue;
9.           G[num].band = true;
```

```
10.         if( Td[newn] ){
11.             low[nod] = min( low[nod], Td[newn] );
12.             continue;
13.         }
14.         bridges_PtoArt( newn );
15.         low[nod] = min( low[nod], low[newn] );
16.
17.         if(Td[nod] < low[newn])
18.             puente.push(par( nod, newn ));
19.
20.         if( (Td[nod] == 1 && Td[newn] > 2 ) ||
21.             ( Td[nod] != 1 && Td[nod] <= low[newn] ) )
22.             Punto_art[nod] = true;
23.     }
24. }
```

4.5. Tarjan SCC

```
1.  void Tarjan_SCC( int nod ){
2.      int newn;
3.      vector<int>::iterator it;
4.      Td[nod] = low[nod] = ++ k;
5.      P.push( nod );
6.      for(it = V[nod].begin(); it != V[nod].end(); it ++){
7.            newn = *it;
8.
9.            if( Td[newn] ){
10.                 if( !mark[newn] )
11.                       low[nod] = min( low[nod], Td[newn] );
12.                 continue;
13.           }
14.
15.           Tarjan_SCC( newn );
16.           low[nod] = min( low[nod], low[newn] );
17.     }
18.     if( low[nod] == Td[nod] ){
19.           sol ++;
20.           while( !mark[nod] )
21.           {
22.                 printf("%d ", (int)P.top());
23.                 mark[(int)P.top()] = true;
24.                 P.pop();
25.           }
26.     }
27. }
```

4.6. Tarjan BCC

```
1.  void BCC( int nod ){
2.      Td[nod] = Low[nod] = ++ k;
3.      int newn, id;
4.      vector<int>::iterator it;
5.      for( it = V[nod].begin(); it != V[nod].end(); it ++ ){
6.          id = *it;
7.          newn = G[id].nextn( nod );
8.          if( !mark[id] ){
9.              P.push( id );
10.             mark[id] = true;
11.         }
12.         if( Td[newn] ){
13.             Low[nod] = min( Low[nod], Td[newn] );
14.             continue;
15.         }
16.         BCC( newn );
17.         Low[nod] = min( Low[newn], Low[nod] );
18.         if( Td[nod] <= Low[newn] ){
19.             num ++;
20.             while( !CB[id] ){
21.                 CB[P.top()] = num;
22.                 P.pop();
23.             }
24.         }
25.     }
26. }
```

4.7. Vertex cover bipartite

```
1.  //  Running time: O(VE)
2.  #define MAXV 5000
3.  int X, Y, E;
4.  int matched[MAXV];
5.  bool mark[MAXV];
6.  bool T[MAXV];
7.  vector<int> ady[MAXV];
8.  typedef pair<int, bool> par;
9.  queue<par> Q;
10. bool augment( int nod ){
11.   if ( nod == -1 ) return true;
12.   int size = ady[nod].size();
13.   for ( int i = 0; i < size; i++ ){
14.         int newn = ady[nod][i];
15.         if ( mark[newn] ) continue ;
16.         mark[newn] = true;
```

```
17.          if ( augment( matched[newn] ) ) {
18.                matched[nod] = newn;
19.                matched[newn] = nod;
20.                return true;
21.          }
22.    }
23.    return false;
24. }
25. void Vertex_Cover_Bipartite(  ){ /// X->Y
26.      /* Find maximum matching */
27.      memset( matched, -1, sizeof( matched ) );
28.      memset( T, false, sizeof( T ) );
29.    int cardinality = 0;
30.    for ( int i = 0; i < X; i++ ){
31.          memset( mark, 0, sizeof( mark ) );
32.          if ( augment( i ) ) cardinality++;
33.    }
34.      /* Find minimum vertex cover */
35.    for ( int i = 0; i < X; i++ )
36.          if ( matched[i] == -1 ){
37.                T[i] = true;
38.                Q.push( par( i, true ) );
39.          }
40.      int nod, newn; bool band;
41.    while ( !Q.empty() ){
42.          nod = Q.front().first;
43.          band = Q.front().second; Q.pop();
44.          int size = ady[nod].size();
45.          for ( int i = 0; i < size; i++ ){
46.                newn = ady[nod][i];
47.                if ( T[newn] ) continue ;
48.                if ( ( band && newn != matched[nod] ) ||
49.                  ( !band && newn == matched[nod] )){
50.                      T[newn] = true;
51.                      Q.push( par( newn, !band ) );
52.                }
53.          }
54.    }
55.    printf("Minimum Vertex Cover:%d\n", cardinality );
56.    //for ( int i = X; i < X + Y; i++ ) if( T[i] )
57.    // vline %d %d %d -> V[i-X+1].x, V[i-X+1].a, V[i-X+1].b
58.    //for ( int i = 0; i < X; i++ ) if ( !T[i] )
59. //hline %d %d %d\n" -> H[i+1].x, H[i+1].a, H[i+1].b
60. }
```

4.8. Edmons-Karp

```
1.  void Edmon_Karp( ){
2.     int nod, newn, flow[10005], P[10005];
3.     bool band;
4.     for( ; ; ){
5.           fill( flow, flow + 2 + 2*N, 0 );
6.           fill( P, P + 2 + 2*N, -1 );
7.           P[0] = 0, flow[0] = 1, band = false;
8.           while( !Q.empty() ) Q.pop();
9.           Q.push( 0 );
10.          while( !band && !Q.empty() ){
11.                nod = Q.front();  Q.pop();
12.                for(int i=Index[nod];i != -1; i = G[i].next ){
13.                      newn = G[i].newn;
14.                      if( P[newn] != -1 || !G[i].cap  )
15.                            continue;
16.                      flow[newn] = min( G[i].cap, flow[nod] );
17.                      P[newn] = i, Q.push( newn );
18.                      if( newn == fin ){
19.                            band = true;
20.                            break;
21.                      }
22.                }
23.          }
24.          if( !flow[fin] ) break;
25.          sol += flow[fin];
26.          for( int i = fin; i != 0; i = G[P[i]].nod ){
27.                G[P[i]].cap -= flow[fin];
28.                G[P[i]^1].cap += flow[fin];
29.          }
30.    }
31. }
```

4.9. Dinic O(NM)

```
1.  int lv[2005], Id[2005];
2.  bool Bfs( int limt ){
3.     while( !Q.empty() ) Q.pop();
4.     fill( lv, lv + 2001, 0);
5.     lv[0] = 1;
6.     Q.push( 0 );
7.
8.     int nod, newn;
9.     while( !Q.empty() ) {
10.          nod = Q.front();
11.          Q.pop();
```

```
            for( int i = Index[nod]; i != -1; i = G[i].next ){
                  newn = G[i].newn;
                  if( lv[newn]!=0 || G[i].cap<limt )continue;
                  lv[newn] = lv[nod] + 1;
                  Q.push( newn );
                  if( newn == fin ) return true;
            }
      }
      return false;
}
bool Dfs( int nod, int limt ){
      if( nod == fin ) return true;
      int newn;
      for( ; Id[nod] != -1; Id[nod] = G[Id[nod]].next ){
            newn = G[Id[nod]].newn;
            if(lv[nod]+1==lv[newn] && G[Id[nod]].cap>=limt
               && Dfs( newn, limt ) ){
                  G[Id[nod]].cap -= limt;
                  G[Id[nod]^1].cap += limt;
                  return true;
            }
      }
      return false;
}
int Dinic( ){
      int flow = 0;
      for( int limt = 4; limt > 0;  ){
            if( !Bfs( limt ) ){
                  limt >>= 1;
                  continue;
            }

            for( int i = 0; i <= fin; i ++ )
                  Id[i] = Index[i];

            while( limt > 0 && Dfs( 0, limt ) )
                  flow += limt;
      }
      return flow;
}
```

4.10. StoerWagner

```
//maximo flujo seleccionando la mejor fuente y mejor sumidero
int G[MAXN][MAXN], w[MAXN], N;
bool A[MAXN], merged[MAXN];
int StoerWagner(int n){
    int best = 1e8;
    for(int i=1;i<n;++i) merged[i] = 0;
    merged[0] = 1;
    for(int phase=1;phase<n;++phase){
        A[0] = 1;
        for(int i=1;i<n;++i){
            if(merged[i]) continue;
            A[i] = 0;
            w[i] = G[0][i];
        }
        int prev = 0,next;
        for(int i=n-1-phase;i>=0;--i){
            // hallar siguiente vertice que no esta en A
            next = -1;
            for(int j=1;j<n;++j)
                if(!A[j] && (next==-1 || w[j]>w[next]))
                    next = j;
            A[next] = true;
            if(i>0){
                prev = next;
                // actualiza los pesos
                for(int j=1;j<n;++j) if(!A[j])
                      w[j] += G[next][j];
            }
        }
        if(best>w[next]) best = w[next];
        // mezcla s y t
        for(int i=0;i<n;++i){
            G[i][prev] += G[next][i];
            G[prev][i] += G[next][i];
        }
        merged[next] = true;
    }
    return best;
}
```

4.11. Max_Flow_Min_Cost

```
priority_queue<par, vector<par>, greater<par> >Qp;
par Max_Flow_Min_Cost( ){
   int FlowF = 0, CostF = 0, F[1005], parent[1005];
    int  nod, newn, newc, flow, dist[1005], cost;
   for( ; ; ){
         fill( F + 1, F + 1 + Fin, 0 );
         fill( dist + 1, dist + 1 + Fin, 1 << 30 );
```

```
8.          F[In] = 1 << 30, dist[In] = 0;
9.          Qp.push( par( 0, In ) );
10.         while( !Qp.empty() ){
11.               nod = Qp.top().second, cost = Qp.top().first;
12.               Qp.pop();
13.               flow = F[nod];
14.               for(int i=Index[nod];i != -1; i = G[i].next ){
15.                     newn = G[i].newn;
16.                     newc=cost+G[i].cost+Phi[nod]-Phi[newn];
17.                     if( G[i].cap > 0 && dist[newn] > newc ){
18.                           dist[newn] = newc;
19.                           F[newn] = min( flow, G[i].cap );
20.                           parent[newn] = i;
21.                           Qp.push( par( newc, newn ) );
22.                     }
23.               }
24.         }
25.         if( F[Fin] <= 0 ) break;
26.         CostF += (( dist[Fin] + Phi[Fin] ) * F[Fin] );
27.         FlowF += F[Fin];
28.         for( int i = 1; i <= N; i ++ )
29.               if( F[i] ) Phi[i] += dist[i];
30.         nod = Fin;
31.         while( nod != In ){
32.               G[parent[nod]].cap -= F[Fin];
33.               G[parent[nod]^1].cap += F[Fin];
34.               nod = G[parent[nod]].nod;
35.         }
36.   }
37.   return par( CostF, FlowF );
38. }
```

4.12. Hungarian O(N^3)

```
1.  #define MAXN 300
2.  int N,A[MAXN+1][MAXN+1],p,q, oo = 1 <<30;
3.  int fx[MAXN+1],fy[MAXN+1],x[MAXN+1],y[MAXN+1];
4.  int hungarian(){
5.      memset(fx,0,sizeof(fx));
6.      memset(fy,0,sizeof(fy));
7.      memset(x,-1,sizeof(x));
8.      memset(y,-1,sizeof(y));
9.      for(int i = 0; i < N; ++i)
10.         for(int j = 0; j < N; ++j) fx[i] = max(fx[i],A[i][j]);
11.     for(int i = 0; i < N; ){
12.         vector<int> t(N,-1), s(N+1,i);
13.         for(p = q = 0; p <= q && x[i]<0; ++p)
14.             for(int k = s[p], j = 0; j < N && x[i]<0; ++j)
15.                 if (fx[k]+fy[j]==A[k][j] && t[j]<0)
16.                 {
17.                     s[++q]=y[j];
18.                     t[j]=k;
19.                     if(s[q]<0)
20.                         for(p=j; p>=0; j=p)
21.                             y[j]=k=t[j], p=x[k], x[k]=j;
22.                 }
23.         if (x[i]<0){
24.             int d = oo;
25.             for(int k = 0; k < q+1; ++k)
26.                for(int j = 0; j < N; ++j)
27.                  if(t[j]<0)d=min(d,fx[s[k]]+fy[j]-A[s[k]][j]);
28.             for(int j = 0; j < N; ++j) fy[j]+=(t[j]<0?0:d);
29.             for(int k = 0; k < q+1; ++k) fx[s[k]]-=d;
30.         }
31.         else ++i;
32.     }
33.     int ret = 0;
34.     for(int i = 0; i < N; ++i) ret += A[i][x[i]];
35.     return ret;
36. }
```

4.13. Kuhn Bipartite Matching O(NM)

```
1.  bool khun( int nodo ){
2.      if( mark[nodo] )
3.          return false;
4.      mark[nodo] = 1;
5.      int tam = V[nodo].size();
6.      for( int i = 0; i < tam; i++ ){
7.          int ady = V[nodo][i];
8.          if( ( match[ady] == -1 || khun(match[ady])) ){
9.              match[ady] = nodo;
10.             return true;
11.         }
12.     }
13.     return false;
14. }
15. void PreMatching() {
16.   for( int i = 1; i <= N; i++ ){
17.         for( int j = 0; j < (int)V[i].size(); j++ ){
18.               int ady  = V[i][j];
19.               if( match[ady] != -1 )
```

```
                    continue;
                match[ady] = i;
                used[i] = true;
                break;
            }
        }
    }
    /// a -> N+b  N|W
    int Bipartite_matchin( ){
      memset(match,-1,sizeof(int)*(N+W+1));
      PreMatching();
      int sol = 0;
      for( int i = 1; i <= N; i++ ){
         fill(mark,mark+N+1,false);
         if( used[i] ){
                sol++;
                continue;
         }
         if( khun(i) ) sol++;
      }
      return sol;
    }
```

4.14. Hopcroft-Karp Bipartite Matching O(M*sqrt(N))

```
const int MAXV = 1001;
const int MAXV1 = 2*MAXV;
int N,M;
vector<int> ady[MAXV];
int D[MAXV1],Mx[MAXV], My[MAXV];
bool BFS(){
    int u, v, i, e;
    queue<int> cola;
    bool f = 0;
    for (i = 0; i < N+M; i++) D[i] = 0;
    for (i = 0; i < N; i++)
        if (Mx[i] == -1) cola.push(i);
    while (!cola.empty()){
        u = cola.front(); cola.pop();
        for (e = ady[u].size()-1; e >= 0; e--) {
            v = ady[u][e];
            if (D[v + N]) continue;
            D[v + N] = D[u] + 1;
            if (My[v] != -1){
                D[My[v]] = D[v + N] + 1;
                cola.push(My[v]);
            }
            else f = 1;
        }
    }
    return f;
}
int DFS(int u){
    for (int v, e = ady[u].size()-1; e >=0; e--){
        v = ady[u][e];
        if (D[v+N] != D[u]+1) continue;
        D[v+N] = 0;
        if (My[v] == -1 || DFS(My[v])){
            Mx[u] = v;  My[v] = u;  return 1;
        }
    }
    return 0;
}
int Hopcroft_Karp(){
    int i, flow = 0;
    for (i = max(N,M); i >=0; i--) Mx[i] = My[i] = -1;
    while (BFS())
        for (i = 0; i < N; i++)
            if (Mx[i] == -1 && DFS(i))
                ++flow;
     return flow;
}
```

4.15. Havy light decomposition

```
//Havy light decomposition
/// cant- la cantidad de nodos
/// pos- la pos. donde aparece
/// nn- el nod en el cual aparece
/// pd- el link con el padre full superior
/// G-Dp
/// L-lazy
vector<int> G[MN], V[MN];
vector<bool> L[MN];
int cant[MN], pos[MN], nn[MN], pd[MN];
void Dfs( int nod, int pad ){
    int t = V[nod].size(), newn;
    if( t == 1 && nod != 1 ){
      pos[nod] = 0;
      nn[nod] = nod;
      cant[nod] = 1;
      pd[nod] = pad;
```

```
      return;
    }
    int mej = nod;
    for( int i = 0; i < t; i ++ ){
        newn = V[nod][i];
        if( newn == pad )
            continue;

        Dfs( newn, nod );
        if( cant[mej] < cant[nn[newn]] )
            mej = nn[newn];
    }
    pos[nod] = cant[mej];
    cant[mej] ++;
    nn[nod] = mej;
    pd[mej] = pad;
}
typedef pair<int, int> par;
typedef pair<int, par> tri;
typedef vector<tri> vt;
typedef vector<par> vp;
/// me da el recorrido desde a hasta b en vector<tri>
/// f posicion s.f in, s.f fin
vt rec( int a, int b ){
  vp A1, B1;
  A1.clear(), B1.clear();
  for( int i = a; i != -1; i = pd[nn[i]] )
       A1.push_back( par( nn[i], pos[i] ) );
  for( int i = b; i != -1; i = pd[nn[i]] )
       B1.push_back( par( nn[i], pos[i] ) );

  vt C1;
  C1.clear();
  reverse( A1.begin(), A1.end() );
  reverse( B1.begin(), B1.end() );
  int t = 0;
  while(t < (int)A1.size() && t <
       (int)B1.size() && A1[t] == B1[t] ) t ++;
  if( t >= (int)A1.size() || t >= (int)B1.size() ||
    ( t < (int)B1.size() && t < (int)A1.size()
     && A1[t].first != B1[t].first ) )   t --;
  if( (t <(int) A1.size() && t < (int)B1.size())
     && A1[t].first == B1[t].first ){
       C1.push_back( tri( A1[t].first,
              par( min( A1[t].second, B1[t].second ),
                   max( A1[t].second, B1[t].second ) ) ) );
       t ++;
  }
  for( int i = t; i <(int) A1.size(); i ++ )
       C1.push_back( tri( A1[i].first, par( A1[i].second,
                          cant[A1[i].first] - 1 ) ) );
  for( int i = t; i < (int)B1.size(); i ++ )
       C1.push_back( tri( B1[i].first, par( B1[i].second,
                          cant[B1[i].first] - 1 ) ) );
  return C1;
}
void havy_light( ){
  Dfs( 1, -1 ); // root
  for( int i = 1; i <= N; i ++ )/// rellenar con 4*cant
       if( cant[i] ){
            G[i] = vector<int> ( cant[i]*4, 0 );
            L[i] = vector<bool> ( cant[i]*4, false );
       }
}
```

4.16. Estable Marriage

```
typedef vector<int> vi;
typedef vector<vi> vvi;
#define rep(i,a,b) for ( __typeof(a) i=(a); i<(b); ++i)
vi stable_marriage(int n, int **m, int **w){
    queue<int> q;
    vi at(n, 0), eng(n, -1), res(n, -1); vvi inv(n, vi(n));
    rep(i,0,n) rep(j,0,n) inv[i][w[i][j]] = j;
    rep(i,0,n) q.push(i);
    while (!q.empty()) {
        int curm = q.front(); q.pop();
        for (int &i = at[curm]; i < n; i++) {
            int curw = m[curm][i];
            if (eng[curw] == -1) { }
            else if (inv[curw][curm] < inv[curw][eng[curw]])
                q.push(eng[curw]);
            else continue;
            res[eng[curw] = curm] = curw, ++i; break;
        }
    }
    return res;
}
```

4.17. Edmons.

```
struct MaxMatching {
```

```
2.    static const int MaxV = 1001;
3.    int V, E;
4.    int match[MaxV];
5.    int head, tail, Q[MaxV];
6.    int start, finish;
7.    int newbase;
8.    int father[MaxV], base[MaxV];
9.    bool graph[MaxV][MaxV];
10.   int queue[MaxV];
11.   bool inpath[MaxV];
12.   bool inblossom[MaxV];
13.   bool inqueue[MaxV];
14.   void initialize(int __nodes){
15.         V = __nodes;
16.         memset(graph,false,sizeof(graph));
17.   }
18.   void addEdge(int u, int v){
19.         graph[u][v] = true;
20.         graph[v][u] = true;
21.   }
22.   void push(int u){
23.         Q[tail++] = u;
24.         inqueue[u] = true;
25.   }
26.   int pop(){ return Q[head++]; }
27.   int findCommonAncestor(int u, int v){
28.         memset(inpath,0,sizeof(inpath));
29.         while(true){
30.               u = base[u];
31.               inpath[u] = true;
32.               if(u == start) break;
33.               u = father[match[u]];
34.         }
35.         while(true){
36.               v = base[v];
37.               if(inpath[v]) break;
38.               v = father[match[v]];
39.         }
40.         return v;
41.   }
42.   void resetTrace(int u){
43.         while(base[u] != newbase){
44.               int v = match[u];
45.               inblossom[base[u]]= true;
46.               inblossom[base[v]]= true;
47.               u = father[v];
48.               if(base[u] != newbase) father[u] = v;
49.         }
50.   }
51.   void  blossomContract(int u, int v){
52.         newbase = findCommonAncestor(u,v);
53.         memset(inblossom,false,sizeof(inblossom));
54.         resetTrace(u);
55.         resetTrace(v);
56.         if(base[u] != newbase) father[u]= v;
57.         if(base[v] != newbase) father[v]= u;
58.         for(int i = 1; i <= V; ++i)
59.               if(inblossom[base[i]]){
60.                     base[i] = newbase;
61.                     if(!inqueue[i]) push(i);
62.               }
63.   }
64.   void find_augmenting_path()     {
65.         memset(inqueue,false,sizeof(inqueue));
66.         memset(father,0,sizeof(father));
67.         for(int i = 1; i <= V; ++i) base[i] = i;
68.         head = 0;
69.         tail = 0;
70.         push(start);
71.         finish = 0;
72.         while(head < tail){
73.               int u = pop();
74.               for(int v = 1; v <= V; ++v)
75.                   if(graph[u][v]&&(base[u] != base[v])
76.                                 && (match[u] != v)){
77.                         if((v == start) || ((match[v] > 0)
78.                           && (father[match[v]] > 0))){
79.                               blossomContract(u,v);
80.                               continue;
81.                         }
82.                         if(father[v] == 0){
83.                            father[v] = u;
84.                            if(match[v] > 0)push(match[v]);
85.                            else{
86.                                     finish = v;
87.                                     return;
88.                               }
89.                         }
90.                   }
91.         }
```

```
92.    }
93.    void augment_path(){
94.          int u = finish;
95.          while(u > 0){
96.                int v = father[u];
97.                int w = match[v];
98.                match[v] = u;
99.                match[u] = v;
100.                     u = w;
101.               }
102.         }
103.         int edmonds(){
104.               memset(match,0,sizeof(match));
105.               for(int i = 1; i <= V; ++i)
106.                     if(!match[i]){
107.                           start = i;
108.                           find_augmenting_path();
109.                           if(finish > 0) augment_path();
110.                    }
111.              int ans = 0;
112.              for(int i = 1; i <= V; ++i)
113.                    if(match[i]) ++ans;
114.              return ans / 2;
115.        }
116.   } edmond;
```

4.18. Centroid descomposition

```
1.  #define MAXN 100005
2.  bool mark[MAXN];
3.  int cant[MAXN], timer;
4.  void Dfs( int nod, int pad ){
5.      cant[nod] = 1;
6.      for( auto i:V[nod] )
7.          if( i.first != pad && !mark[i.first] ){
8.              Dfs( i.first, nod );
9.              cant[nod] += cant[i.first];
10.         }
11. }
12. int centroid( int nod, int pad, int nn ){
13.     for( auto i:V[nod] )
14.       if(i.first!=pad && !mark[i.first] && cant[i.first]>nn/2)
15.             return centroid( i.first, nod, nn );
16.     return nod;
17. }
18. long long solve( int nod, int pad ){
19.     Dfs( nod, pad );
20.     int centr = centroid( nod, pad, cant[nod] );
21.     long long sol = 0;
22.     mark[centr] = true;
23.     timer ++;
24.     for( auto i : V[centr] )
25.         if( i.first != pad && !mark[i.first] ){
26.             sol += query( i.first, centr, i.second );
27.             updater( i.first, centr, i.second );
28.         }
29.     for( auto i : V[centr] )
30.         if( i.first != pad && !mark[i.first] )
31.             sol += solve( i.first, centr );
32.     return sol;
33. }
```

5. Data Structures

5.1. Suma de intervalos con BIT

```
1.  void updater( int x, int v ){
2.   int tmp = x-1;
3.   for( ; x <= N; x += (x&-x) ){
4.      Dp[1][x] += v, Dp[2][x] += v*tmp;
5.   }
6.  }
7.  int sum( int p, int x ){
8.   int s = 0;
9.   for( ; x >= 1; x -= (x&-x) )
10.     s += Dp[p][x];
11.  return s;
12. }
13. int sumsum( int a ){
14.   return sum( 1, a )*a - sum( 2, a );
15. }
16. void updater_interv( int a, int b, int v ){
17.  updater( a, v ), updater( b+1, -v );
18. }
```

5.2. AVL

```
1.  template <class T>
2.  struct avl_tree {
3.    struct node {
4.      T key;
5.      int size, height;
6.      node *child[2];
7.      node(const T &key) : key(key), size(1), height(1) {
8.        child[0] = child[1] = 0; }
```

```
  } *root;
  typedef node *pointer;
  avl_tree() { root = NULL; }

  pointer find(const T &key) { return find(root, key); }
  node *find(node *t, const T &key) {
    if (t == NULL) return NULL;
    if (key == t->key) return t;
    else if (key < t->key) return find(t->child[0], key);
    else                   return find(t->child[1], key);
  }
  void insert(const T &key){root=insert(root,new node(key));}
  node *insert(node *t, node *x) {
    if (t == NULL) return x;
    if (x->key < t->key) t->child[0] = insert(t->child[0], x);
    else t->child[1] = insert(t->child[1], x);
    t->size += 1;
    return balance(t);
  }
  void erase(const T &key) { root = erase(root, key); }
  node *erase(node *t, const T &x) {
    if (t == NULL) return NULL;
    if (x == t->key) {
      return move_down(t->child[0], t->child[1]);
    } else {
      if (x < t->key) t->child[0] = erase(t->child[0], x);
      else            t->child[1] = erase(t->child[1], x);
      t->size -= 1;
      return balance(t);
    }
  }
  node *move_down(node *t, node *rhs) {
    if (t == NULL) return rhs;
    t->child[1] = move_down(t->child[1], rhs);
    return balance(t);
  }
#define sz(t) (t ? t->size : 0)
#define ht(t) (t ? t->height : 0)
  node *rotate(node *t, int l, int r) {
    node *s = t->child[r];
    t->child[r] = s->child[l];
    s->child[l] = balance(t);
    if (t) t->size = sz(t->child[0]) + sz(t->child[1]) + 1;
    if (s) s->size = sz(s->child[0]) + sz(s->child[1]) + 1;
    return balance(s);
  }
  node *balance(node *t) {
   for (int i = 0; i < 2; ++i) {
    if (ht(t->child[!i]) - ht(t->child[i]) < -1) {
     if(ht(t->child[i]->child[!i])-ht(t->child[i]->child[i])
>0)
        t->child[i] = rotate(t->child[i], i, !i);
      return rotate(t, !i, i);
     }
    }
    if(t)t->height = max(ht(t->child[0]), ht(t->child[1]))+1;
    if (t) t->size = sz(t->child[0]) + sz(t->child[1]) + 1;
    return t;
  }
  pointer rank(int k) const { return rank(root, k); }
  pointer rank(node *t, int k) const {
    if (!t) return NULL;
    int m = sz(t->child[0]);
    if (k  < m) return rank(t->child[0], k);
    if (k == m) return t;
    if (k  > m) return rank(t->child[1], k - m - 1);
  }

  void clear( node *x ){
   if( !x ) return;
   if( x->child[0] )
     clear( x->child[0] );
   if( x->child[1] )
     clear( x->child[1] );
   delete x;
  }

  int solve( const T v ){
   node *p = root;
   int sol = 0;

   while( p ){
        if (v < p->key)
         p = p->child[0];
        else
         sol+=(( !p->child[0] )?0:p->child[0]->size)+1,
p=p->child[1];
   }
   return sol;
  }
```

```
97. };
```

5.3. Misof Tree->the nth largest element

```
1.  // Misof Tree. A simple tree data structure for inserting,
2.  erasing, and querying the nth largest element.
3.  #define BITS 15
4.  struct misof_tree {
5.      int cnt[BITS][1<<BITS];
6.      misof_tree() { memset(cnt, 0, sizeof(cnt)); }
7.      void insert(int x) {
8.           for (int i = 0; i < BITS; cnt[i++][x]++, x >>= 1); }
9.      void erase(int x) {
10.          for (int i = 0; i < BITS; cnt[i++][x]--, x >>= 1); }
11.     int nth(int n) {
12.         int res = 0;
13.         for (int i = BITS-1; i >= 0; i--)
14.             if (cnt[i][res <<= 1] <= n)n-=cnt[i][res], res|=1;
15.         return res;
16.     }
17. };
```

5.4. Convex Hull Trick

```
1.  struct convex_hull_trick {
2.      vector< pair<double,double> > h;
3.      double intersect(int i) {
4.      return (h[i+1].second-h[i].second)/
5.             (h[i].first-h[i+1].first);
6.      }
7.      void add(double m, double b) {
8.          h.push_back(make_pair(m,b));
9.          while (h.size() >= 3) {
10.             int n = h.size();
11.             if (intersect(n-3) < intersect(n-2)) break;
12.             swap(h[n-2], h[n-1]);
13.             h.pop_back();
14.     }   }
15.     double get_min(double x) {
16.         int lo = 0, hi = h.size() - 2, res = -1;
17.         while (lo <= hi) {
18.             int mid = lo + (hi - lo) / 2;
19.             if (intersect(mid) <= x) res = mid, lo = mid + 1;
20.             else hi = mid - 1;
21.         }
22.         return h[res+1].first*x + h[res+1].second;
23.     }};
```

5.5. Monotonic Queue

```
1.  typedef long long i64;
2.  struct monotonic_queue {
3.      deque< pair<int, i64> > D;
4.      void add( int p, i64 v ){
5.          while( !D.empty() && D.front().second <= v )
6.              D.pop_front();
7.          D.push_front({p, v});
8.      }
9.      void borrar( int x ){
10.         while( !D.empty() && D.back().first <= x )
11.             D.pop_back();
12.     }
13.     i64 maximo( ){ return D.back().second; }
14. };
```

5.6. Splay Tree

```
1.  struct splay_tree{
2.      const int inf = 1e9;
3.     struct nodo {
4.           int size, cant[30];
5.           nodo *l, *r, *p;
6.           bool inv;
7.           int laz, let;
8.           nodo(nodo *f=0, nodo *i = 0, nodo *d = 0){
9.               l=i, p=f,r=d,size=1,let=0,laz = -1, inv = false;
10.                for(int i=0; i<30; i++) cant[i]=0;
11.          }
12.   } *root;
13.     splay_tree(){ root = NULL; }
14.     inline void zig(nodo *x) {
15.          nodo *y = x->p, *z = y->p;
16.          y->l = x->r;
17.          if( x->r )
18.           x->r->p = y;
19.          x->p = z;
20.          if( z )
21.            if (z->l == y)z->l = x; else z->r = x;
22.          y->p = x, x->r = y;
23.          updata(y);
24.   }
25.   inline void zag(nodo *x) {
26.          nodo *y = x->p, *z = y->p;
27.          y->r = x->l;
```

```
        if( x->l )
         x->l->p = y;
        x->p = z;
        if( z )
          if (z->l == y)z->l = x; else z->r = x;
        y->p = x, x->l = y;
        updata(y);
  }
  inline void splay(nodo *x) {
        for (; x->p ;) {
              nodo *y = x->p, *z = y->p;
              if (!z) {
                    if (y->l == x) zig(x); else zag(x);
              } else {
                    if (z->l == y){
                          if (y->l == x)zig(y), zig(x);
                          else zag(x), zig(x);
                    }
                    else if (y->r == x) zag(y), zag(x);
                    else zig(x), zag(x);
              }
        }
        root = x, updata(root);
  }
  void find(int x) {
        if(!root)return;
        nodo *p = root;
        for( ; ; ) {
              lazy( p );
              int izq = (p->l)?p->l->size:0;
              if (x == izq + 1) break;
              if (x > izq + 1){
                    x -= izq + 1;
                    if ( p->r ) p = p->r; else break;
              }
              else
                if ( p->l ) p = p->l; else break;
        }
        splay(p);
  }
  inline void insertpos( int a, int b ){
    nodo *nn = new nodo( 0, 0, 0 );
     nn->let = b;
    find( a );
    if( !root ){ root = nn, updata(root); return; }
      nodo *p = root;
      root = root->r;
      if( root ) root->p = 0;
      p->r = nn, nn->p = p;
      find( -inf );
      nn->r = root;
      if( root )
       root->p = nn;
      root = p;
      updata(nn), updata(root);
      int ui = 0;
  }
  inline void insert(int a) {
        nodo *p = root, *f=0;
        while(p){ f=p; p = p->r; }
        p = new nodo(f, 0, 0);
        p->let = a;
        if( f )
         f->r = p;
        splay(p);
  }
  inline splay_tree split(int x){
        if(!root)return splay_tree();
        splay_tree L = splay_tree();
        find(x);
        if( root->l )
         root->l->p=0;
              L.root = root->l, root->l=0;
              updata(root);
              return L;
        }
        inline void join(splay_tree L){
            if( !L.root ) return;

              if(!root) root = L.root;
              else{
                    find(-inf);
                    root->l = L.root, root->l->p = root;
                    updata(root);
              }
              L.root = NULL;
        }
        void print(nodo *r){
              if(r == NULL)return;
              lazy(r);
```

```
118.              print(r->l);
119.              printf("%c ", r->let);
120.              print(r->r);
121.        }
122.        void erase(int x) {
123.              find(x);
124.              if(!root)return;
125.
126.              if (!root->l) {
127.                    nodo *tmp = root;
128.                    root = root->r;
129.                    if(root)
130.                     root->p = 0;
131.                    delete tmp;
132.              } else {
133.                    nodo *t = root->r, *tmp = root;
134.                    root = root->l;
135.                    if(root)root->p = 0;
136.                    find(x);
137.                    if(root)root->r = t;
138.                    if( t ) t->p = root;
139.                    updata(root);
140.                    delete tmp;
141.              }
142.        }
143.        void clear( nodo*x ){
144.            if( x ) return;
145.        clear( x->l );
146.        clear( x->r );
147.        delete x;
148.        }
149.        inline void updata(nodo *x) {
150.              x->size=((x->l)?x->l->size:0)+
151.                      ((x->r)?x->r->size:0)+1;
152.              for(int i = 0; i < 30; i ++)
153.                    x->cant[i] = ((x->l)?x->l->cant[i]:0) +
154.                                 ((x->r)?x->r->cant[i]:0) +
155.                                 (x->let == i );
156.        }
157.        inline void lazy(nodo *p){
158.              if(!p)return;
159.              if(p->inv){
160.                    swap(p->r, p->l);
161.                    if( p->r ) p->r->inv = !p->r->inv;
162.                    if( p->l ) p->l->inv = !p->l->inv;
163.                    p->inv=0;
164.              }
165.              if(p->laz!=-1){
166.                    updlazy(p->l, p->laz);
167.                    updlazy(p->r, p->laz);
168.                    p->laz = -1;
169.              }
170.        }
171.        inline void updlazy(nodo *p, int laz){
172.            if( !p ) return;
173.              p->laz = laz;
174.              for(int i=0; i<30; i++)
175.                if(i==p->laz)     p->cant[i] = p->size;
176.                else p->cant[i] = 0;
177.              p->let = laz;
178.        }
179.        void solve(char opt, int a, int b, int c = 0 ){
180.              splay_tree t1 = split( a );
181.              splay_tree t = split( b - a + 2 );
182.
183.              if(opt=='S') t.updlazy(t.root, c);
184.          else if( opt == 'R' )t.root->inv=( !t.root->inv );
185.              else printf("%d\n", t.root->cant[c]);
186.
187.              join(t);
188.              join(t1);
189.        }
190.  }ST;
```

5.7. RMQ

```
1.  void build_rmq( ){
2.      for(int i = 0; i < N; i ++) M[i][0] = i;
3.      for(int j = 1; ( 1 << j ) < N; j ++ )
4.          for(int i = 0; i+(1<<(j-1)) < N; i ++ )
5.              if(arr[M[i][j-1]] <= arr[ M[i+(1<<(j-1))][j-1]])
6.                  M[i][j] = M[i][j-1];
7.              else M[i][j] = M[i+(1<<(j-1))][j-1];
8.  }
9.  int query_rmq( int x, int y ){
10.     int lg = log2( y - x + 1 );
11.     if( arr[ M[x][lg] ] <= arr[M[y-(1<<lg)+1][lg] ])
12.         return M[x][lg];
13.     else return M[y-(1<<lg)+1][lg];
14. }
```

6. Dynamic Programming

6.1. Conquer and Divide Optimizations

```
void compute(int k, int L, int R, int optL, int optR){
  if (L > R) return;
  int m = (L + R) / 2, opt = -1;
  dp[m][1] = oo;
  for (int i = optL; i <= min(m, optR); i++){
        i64 t = dp[i - 1][0] + w(i, m);
        if (dp[m][1] > t)
              dp[m][1] = t, opt = i;
  }
  compute(k, L, m - 1, optL, opt);
  compute(k, m + 1, R, opt, optR);
}
```

6.2. LIS-LDS

```
void write ( int ID ){
    if( !ID ) return;
    write ( Last[ID] );
    printf ("%d ", List[ID]);
}
void LIS_LDS( ){
    for ( int i = 1; i <= N; i ++ ){
        if ( Sol[m] <= List[i] ){
            Sol[++ m] = List[i];
            Id[m] = i;
            Last[i] = Id[m - 1];
        }
        else{
            up = upper_bound(Sol + 1, Sol + m+1, List[i])-Sol;
            Sol[up] = List[i];
            Id[up] = i;
            Last[i] = Id[up - 1];
        }
    }
    printf ("%d\n", m);
    write ( Id[m] );
}
```

6.3. Towers of Hanoi

```
void move( int n, char from, char to, char aux ) {
  if ( n == 1 )
      printf( "Move disk from %c to %c\n", from, to );
  else {
    move( n - 1, from, aux, to );
    printf( "Move disk from %c to %c\n", from, to );
    move( n - 1, aux, to, from );
  }
}
```

7. Geometry

7.1. Base element

```
const double EPS = 1e-8;
const double inf = 1e12;
typedef complex<double> P;
typedef vector<P> polygon;
namespace std {
  bool operator < (const P& a, const P& b) {
    return real(a)!=real(b)?real(a)<real(b):imag(a)<imag(b);
  }
}
double cross(const P& a, const P& b) {
  return imag(conj(a)*b);
}
double dot(const P& a, const P& b) {
  return real(conj(a)*b);
}
struct L : public vector<P> {
  L(const P &a, const P &b) {
    push_back(a); push_back(b);
  }
};
struct C {
  P p; double r;
  C(const P &p, double r) : p(p), r(r) { }
};
P crosspoint(const L &l, const L &m) {
  double A = cross(l[1] - l[0], m[1] - m[0]);
  double B = cross(l[1] - l[0], l[1] - m[0]);
  if (abs(A) < EPS && abs(B) < EPS) return m[0]; // same line
  if (abs(A) < EPS) assert(false); // NOT SATISFIED!!!
  return m[0] + B / A * (m[1] - m[0]);
}
```

7.2. The traveling direction of the point

```
int ccw(P a, P b, P c) {
  b -= a; c -= a;
  if (cross(b, c) > 0)   return +1;       // counter clockwise
  if (cross(b, c) < 0)   return -1;       // clockwise
  if (dot(b, c) < 0)     return +2;       // c--a--b on line
```

```
    if (norm(b) < norm(c)) return -2;       // a--b--c on line
    return 0;
  }
```

7.3. Intersection

```
  bool intersectLL(const L &l, const L &m) {
    return abs(cross(l[1]-l[0],m[1]-m[0]))>EPS ||//non-parallel
           abs(cross(l[1]-l[0], m[0]-l[0]))<EPS;// same line
  }
  bool intersectLS(const L &l, const L &s) {
    return cross(l[1]-l[0], s[0]-l[0])*// s[0] is left of l
           cross(l[1]-l[0], s[1]-l[0])<EPS;//s[1] is right of l
  }
  bool intersectLP(const L &l, const P &p) {
    return abs(cross(l[1]-p, l[0]-p)) < EPS;
  }
  bool intersectSS(const L &s, const L &t) {
    return ccw(s[0],s[1],t[0])*ccw(s[0],s[1],t[1]) <= 0 &&
           ccw(t[0],t[1],s[0])*ccw(t[0],t[1],s[1]) <= 0;
  }
  bool intersectSP(const L &s, const P &p) {
    return abs(s[0]-p)+abs(s[1]-p)-abs(s[1]-s[0]) < EPS;
    // triangle inequality
  }
```

7.4. Distance

```
  P projection(const L &l, const P &p) {
    double t = dot(p-l[0], l[0]-l[1]) / norm(l[0]-l[1]);
    return l[0] + t*(l[0]-l[1]);
  }
  P reflection(const L &l, const P &p) {
    return p + P(2,0)*(projection(l, p) - p);
  }
  double distanceLP(const L &l, const P &p) {
    return abs(p - projection(l, p));
  }
  double distanceLL(const L &l, const L &m) {
    return intersectLL(l, m) ? 0 : distanceLP(l, m[0]);
  }
  double distanceLS(const L &l, const L &s) {
    if (intersectLS(l, s)) return 0;
    return min(distanceLP(l, s[0]), distanceLP(l, s[1]));
  }
  double distanceSP(const L &s, const P &p) {
    const P r = projection(s, p);
    if (intersectSP(s, r)) return abs(r - p);
    return min(abs(s[0] - p), abs(s[1] - p));
  }
  double distanceSS(const L &s, const L &t) {
    if (intersectSS(s, t)) return 0;
    return min(min(distanceSP(s, t[0]), distanceSP(s, t[1])),
               min(distanceSP(t, s[0]), distanceSP(t, s[1])));
  }
```

7.5. End point

```
  #define d(G, k) (dot(G[k], l[1] - l[0]))
  P extreme(const polygon &G, const L &l) {
    int k = 0;
    for (int i = 1; i < (int)G.size(); ++i)
      if (d(G, i) > d(G, k)) k = i;
    return G[k];
  }
```

7.6. Polygon inclusion decision point

```
  #define curr(G, i) G[i]
  #define next(G, i) G[(i+1)%G.size()]
  enum { OUT, ON, IN };
  int contains(const polygon &G, const P& p) {
    bool in = false;
    for (int i = 0; i < (int)G.size(); ++i) {
      P a = curr(G,i) - p, b = next(G,i) - p;
      if (imag(a) > imag(b)) swap(a, b);
      if (imag(a) <= 0 && 0 < imag(b))
        if (cross(a, b) < 0) in = !in;
      if (cross(a, b) == 0 && dot(a, b) <= 0) return ON;
    }
    return in ? IN : OUT;
  }
```

7.7. Area of a polygon

```
  double area2(const polygon& G) {
    double A = 0;
    for (int i = 0; i < (int)G.size(); ++i)
      A += cross(curr(G, i), next(G, i));
    return A;
  }
```

7.8. Perturbative deformation of a polygon

```
  #define prev(G,i) G[(i-1+G.size())%G.size()]
  polygon shrink_polygon(const polygon &G, double len) {
```

```
    polygon res;
    for (int i = 0; i < (int)G.size(); ++i) {
      P a = prev(G,i), b = curr(G,i), c = next(G,i);
      P u = (b - a) / abs(b - a);
      double th = arg((c - b)/ u) * 0.5;
      res.push_back(b+u * P(-sin(th), cos(th))*len / cos(th) );
    }
    return res;
}
```

7.9. triangulation

```
polygon make_triangle(const P& a, const P& b, const P& c) {
    polygon ret(3);
    ret[0] = a; ret[1] = b; ret[2] = c;
    return ret;
}
bool triangle_contains(const polygon& tri, const P& p) {
    return ccw(tri[0], tri[1], p) >= 0 &&
           ccw(tri[1], tri[2], p) >= 0 &&
           ccw(tri[2], tri[0], p) >= 0;
}
bool ear_Q(int i, int j, int k, const polygon& G) {
    polygon tri = make_triangle(G[i], G[j], G[k]);
    if (ccw(tri[0], tri[1], tri[2]) <= 0) return false;
    for (int m = 0; m < (int)G.size(); ++m)
      if (m != i && m != j && m != k)
        if (triangle_contains(tri, G[m]))
          return false;
    return true;
}
void triangulate(const polygon& G, vector<polygon>& t) {
    const int n = G.size();
    vector<int> l, r;
    for (int i = 0; i < n; ++i) {
      l.push_back( (i-1+n) % n );
      r.push_back( (i+1+n) % n );
    }
    int i = n-1;
    while ((int)t.size() < n-2) {
      i = r[i];
      if (ear_Q(l[i], i, r[i], G)) {
        t.push_back(make_triangle(G[l[i]], G[i], G[r[i]]));
        l[ r[i] ] = l[i];
        r[ l[i] ] = r[i];
      }
    }
}
```

7.10. Convex_hull

```
vector<P> convex_hull(vector<P> ps) {
    int n = ps.size(), k = 0;
    sort(ps.begin(), ps.end());
    vector<P> ch(2*n);
    for (int i = 0; i < n; ch[k++] = ps[i++])
      while (k >= 2 && ccw(ch[k-2], ch[k-1], ps[i]) <= 0) --k;
    for (int i = n-2, t = k+1; i >= 0; ch[k++] = ps[i--])
       while (k >= t && ccw(ch[k-2], ch[k-1], ps[i]) <= 0) --k;
    ch.resize(k-1);
    return ch;
}
```

7.11. Convexity determination

```
bool isconvex(const polygon &G) {
    for (int i = 0; i < (int)G.size(); ++i)
      if(ccw(prev(G,i),curr(G,i),next(G, i)) > 0)return false;
    return true;
}
```

7.12. Cutting of a convex polygon

```
polygon convex_cut(const polygon& G, const L& l) {
    polygon Q;
    for (int i = 0; i < (int)G.size(); ++i) {
      P A = curr(G, i), B = next(G, i);
      if (ccw(l[0], l[1], A) != -1) Q.push_back(A);
      if (ccw(l[0], l[1], A)*ccw(l[0], l[1], B) < 0)
        Q.push_back(crosspoint(L(A, B), l));
    }
    return Q;
}
```

7.13. Diameter of a convex polygon

```
#define diff(G, i) (next(G, i) - curr(G, i))
double convex_diameter(const polygon &pt) {
    const int n = pt.size();
    int is = 0, js = 0;
    for (int i = 1; i < n; ++i) {
      if (imag(pt[i]) > imag(pt[is])) is = i;
      if (imag(pt[i]) < imag(pt[js])) js = i;
    }
    double maxd = norm(pt[is]-pt[js]);
```

```
10.   int i, maxi, j, maxj;
11.   i = maxi = is;
12.   j = maxj = js;
13.   do {
14.     if (cross(diff(pt,i), diff(pt,j)) >= 0) j = (j+1) % n;
15.     else i = (i+1) % n;
16.     if (norm(pt[i]-pt[j]) > maxd) {
17.       maxd = norm(pt[i]-pt[j]);
18.       maxi = i; maxj = j;
19.     }
20.   } while (i != is || j != js);
21.   return maxd; /* farthest pair is (maxi, maxj). */
22. }
```

7.14. End point of a convex polygon

```
1.  P convex_extreme(const polygon &G, const L &l) {
2.    const int n = G.size();
3.    int a = 0, b = n;
4.    if (d(G, 0) >= d(G, n-1) && d(G, 0) >= d(G, 1)) return G[0];
5.    while (a < b) {
6.      int c = (a + b) / 2;
7.      if (d(G,c)>= d(G,c-1) && d(G,c)>=d(G, c+1))return G[c];
8.      if (d(G, a+1) > d(G, a)) {
9.        if (d(G, c+1) <= d(G, c) || d(G, a) > d(G, c)) b = c;
10.       else                                           a = c;
11.     } else {
12.       if (d(G, c+1) > d(G, c) || d(G, a) >= d(G, c)) a = c;
13.       else                                           b = c;
14.     }
15.   }
16.   return G[0];
17. }
```

7.15. Convex polygon inclusion decision point

```
1.  enum { OUT, ON, IN };
2.  int convex_contains(const polygon &G, const P &p) {
3.    const int n = G.size();
4.    P g = (G[0] + G[n/3] + G[2*n/3]) / 3.0; // inner-point
5.    int a = 0, b = n;
6.    while (a+1 < b) { // invariant: c is in fan g-P[a]-P[b]
7.      int c = (a + b) / 2;
8.      if (cross(G[a]-g, G[c]-g) > 0) { // angle < 180 deg
9.        if (cross(G[a]-g,p-g)>0 && cross(G[c]-g,p-g) < 0) b=c;
10.       else a = c;
11.     } else {
12.       if (cross(G[a]-g,p-g)<0 && cross(G[c]-g,p-g) > 0) a=c;
13.       else b = c;
14.     }
15.   }
16.   b %= n;
17.   if (cross(G[a] - p, G[b] - p) < 0) return OUT;
18.   if (cross(G[a] - p, G[b] - p) > 0) return IN;
19.   return ON;
20. }
```

7.16. Incircle

```
1.  bool incircle(P a, P b, P c, P p) {
2.    a -= p; b -= p; c -= p;
3.    return norm(a) * cross(b, c)
4.         + norm(b) * cross(c, a)
5.         + norm(c) * cross(a, b) >= 0;
6.    // < : inside, = cocircular, > outside
7.  }
```

7.17. Closest Pair Points

```
1.  pair<P,P> closestPair(polygon p) {
2.    int n=p.size(), s=0, t=1, m=2, S[n]; S[0] = 0, S[1] = 1;
3.    sort(p.begin(), p.end()); // "p < q" <=> "p.x < q.x"
4.    double d = norm(p[s]-p[t]);
5.    for (int i = 2; i < n; S[m++] = i++)
6.     for(int j = 0; j < m; j ++){
7.      if (norm(p[S[j]]-p[i])<d)d = norm(p[s=S[j]]-p[t=i]);
8.      if (real(p[S[j]]) < real(p[i]) - d) S[j--] = S[--m];
9.    }
10.   return make_pair( p[s], p[t] );
11. }
```

7.18. Intriangle

```
1.  bool intriangle(P a, P b, P c, P p) {
2.    a -= p; b -= p; c -= p;
3.    return cross(a, b) >= 0 &&
4.           cross(b, c) >= 0 &&
5.           cross(c, a) >= 0;
6.  }
```

7.19. Three Point Circle

```
1.  P three_point_circle(const P& a, const P& b, const P& c) {
2.    P x = 1.0/conj(b - a), y = 1.0/conj(c - a);
3.    return (y-x)/(conj(x)*y - x*conj(y)) + a;
4.  }
```

7.20. Circle_circle_intersect

```
pair<P,P>c_c_intersect(const P& c1,const double& r1,
                       const P& c2, const double& r2) {
    P A = conj(c2-c1);
    P B = (r2*r2-r1*r1-(c2-c1)*conj(c2-c1)), C = r1*r1*(c2-c1);
    P D = B*B-4.0*A*C;
    P z1 = (-B+sqrt(D))/(2.0*A)+c1;
    P z2 = (-B-sqrt(D))/(2.0*A)+c1;
    return pair<P, P>(z1, z2);
}
```

8. Solution Ideas

- Dynamic Programming
  - Drop a parameter, recover from others
  - Swap answer and a parameter
  - Parsing CFGs: CYK Algorithm
  - Optimizations
    - Convex hull optimization
      - $dp[i] = \min_{j<i}\{dp[j] + b[j]\times a[i]\}$
      - $b[j] \geq b[j+1]$
      - optionally $a[i] \leq a[i+1]$
      - O(n^2) to O(n)
    - Divide and conquer optimization
      - $dp[i][j] = \min_{k<j}\{dp[i-1][k] + C[k][j]\}$
      - $A[i][j] \leq A[i][j+1]$
      - O(kn^2) to O(knlogn)
      - sufficient:$C[a][c]+C[b][d] \leq C[a][d]+C[b][c]$, $a \leq b \leq c \leq d$ (QI)
    - Knuth optimization
      - $dp[i][j] = \min_{i<k<j}\{dp[i][k]+dp[k][j]+C[i][j]\}$
      - $A[i][j-1] \leq A[i][j] \leq A[i+1][j]$
      - O(n^3) to O(n^2)
      - sufficient:QI and $C[b][c] \leq C[a][d]$, $a \leq b \leq c \leq d$
- Greedy
- Randomized
- Optimizations
  - Use bitset (/64)
  - Switch order of loops (cache locality)
- Process queries offline
  - Mo's algorithm
- Square-root decomposition
- Precomputation
- Efficient simulation
  - Mo's algorithm
  - Sqrt decomposition
  - Store 2^k jump pointers
- Data structure techniques
  - Sqrt buckets
  - Store 2^k jump pointers
  - 2^k merging trick
- Counting
  - Inclusion-exclusion principle
  - Generating functions
- Graphs
  - Can we model the problem as a graph?
  - Can we use any properties of the graph?
  - Strongly connected components
  - Cycles (or odd cycles)
  - Bipartite (no odd cycles)
    - Bipartite matching
    - Hall's marriage theorem
    - Stable Marriage
  - Cut vertex/bridge
  - Biconnected components
  - Degrees of vertices (odd/even)
  - Trees
    - Heavy-light decomposition
    - Centroid decomposition
    - Least common ancestor
    - Centers of the tree
  - Eulerian path/circuit
  - Chinese postman problem
  - Topological sort
  - (Min-Cost) Max Flow
  - Min Cut
    - Maximum Density Subgraph
  - Huffman Coding
  - Min-Cost Arborescence
  - Steiner Tree
  - Kirchoff's matrix tree theorem
  - Prüfer sequences
  - Lovász Toggle
  - Look at the DFS tree (which has no cross-edges)
- Mathematics
  - Is the function multiplicative?
  - Look for a pattern
  - Permutations
    - Consider the cycles of the permutation
  - Functions

        * Sum of piecewise-linear functions is a piecewise-linear function
        * Sum of convex(concave)functions is convex (concave)
    - Modular arithmetic
        * Chinese Remainder Theorem
        * Linear Congruence
    - Sieve
    - System of linear equations
    - Values to big to represent?
        * Compute using the logarithm
        * Divide everything by some large value
    - Linear programming
        * Is the dual problem easier to solve?
- Logic
        - 2-SAT
        - XOR-SAT (Gauss elimination or Bipartite matching)
- Meet in the middle
- Only work with the smaller half (log(n))
- Strings
    - Trie (maybe over something weird, like bits)
    - Suffix array
    - Suffix automaton (+DP?)
    - Aho-Corasick
    - eerTree
    - Work with S + S
- Hashing
- Euler tour, tree to array
- Segment trees
    - Lazy propagation
    - Persistent
    - Implicit
    - Segment tree of X
- Geometry
    - Minkowski sum (of convex sets)
    - Rotating calipers
    - Sweep line (horizontally or vertically?)
    - Sweep angle
    - Convex hull
- Fix a parameter (possibly the answer).
- Are there few distinct values?
- Binary search
- Sliding Window (+ Monotonic Queue)
- Computing a Convolution? Fast Fourier Transform
- Exact Cover (+ Algorithm X)
- Cycle-Finding
- What is the smallest set of values that identify the solution? The cycle structure of the permutation? The powers of primes in the factorization?
- Look at the complement problem
    - Minimize something instead of maximizing
- Immediately enforce necessary conditions. (All values greater than 0? Initialize them all to 1)
- Add large constant to negative numbers to make them positive
- Counting/Bucket sort

9. Debugging Tips

- Stack overflow? Recursive DFS on tree that is actually a long path?
- Rounding negative numbers?
- Double
- Wrong Answer?
    - Quitar el freopen,
    - no mezclar cin con scanf
    - Ver si hay que imprimir fin de linea
    - Leer nuevamente el problema.
    - Ver si es multiple casos, repetir el mismo caso varias veces.
    - long long
    - Posibles Casos:
        * n = 0,n = −1,n = 1,n = 2^31 − 1 or n = −2^31
        * La lista esta vacia o con un solo elemento
        * n is even, n is odd
        * El Grafo esta vacion o contiene un solo vertice
        * El Grafo es un multigrafo (lazo o multiple aristas)
        * El Polygono es convexo o no
    - –Hay condicion inicial para los casos pequeños
    - –Estas utilizando el algoritmo correcto
    - Explique su solucion a algien
    - ¿Usa usted algunas funciones que usted completamente no comprende? ¿Puede que STL funcione?
    - ¿Puede que usted (o alguien más) debiera reescribir la solución?
- Run-Time Error?
    - –Verificar el tamaño de los arreglos
    - –Division por 0