



## Contents

<b>1 2D Geometry</b>	<b>2</b>	<b>7 Graph Theory</b>	<b>5</b>
1.1 Primitives . . . . .	2	7.1 Primitives . . . . .	5
1.2 Intersections . . . . .	2	7.2 Articulation Points & Bridges . . . . .	6
1.3 Circle Generation . . . . .	2	7.3 SCC . . . . .	6
1.4 Heron Triangle Area . . . . .	2	7.4 2-SAT . . . . .	6
1.5 Polygon Centroid . . . . .	2	7.5 Edmonds-Karp Max Flow . . . . .	6
1.6 Point In Polygon . . . . .	2	7.6 Dinic's Max Flow . . . . .	6
1.7 Convex Hull . . . . .	2	7.7 Min-Cost Max Flow . . . . .	6
1.8 Line Segment Set Intersection . . . . .	2	7.8 Euler Cycles . . . . .	7
1.9 Voronoi Diagrams . . . . .	3	7.9 Maximum Matching . . . . .	7
<b>2 3D Geometry</b>	<b>3</b>	7.10 HL Decomposition . . . . .	7
2.1 Primitives . . . . .	3	7.11 Modelling Inequalities . . . . .	7
2.2 Convex Hull . . . . .	3	7.12 Max Flow Tricks . . . . .	7
2.3 Great Circle Distance . . . . .	3	7.13 Bellman Ford . . . . .	7
<b>3 Combinatorics</b>	<b>3</b>	7.14 Stable Marriage . . . . .	7
3.1 Basics . . . . .	3	7.15 Maximum Assignment . . . . .	7
3.2 Permutation (Un)Ranking . . . . .	3	<b>8 Linear Programming</b>	<b>7</b>
3.3 Combination (Un)Ranking . . . . .	3	8.1 Simplex . . . . .	7
<b>4 Data Structures</b>	<b>4</b>	<b>9 Number Theory</b>	<b>8</b>
4.1 Palindromic Tree . . . . .	4	9.1 Extended GCD . . . . .	8
4.2 Treap . . . . .	4	9.2 Modular Inverse . . . . .	8
4.3 Sparse Array . . . . .	4	9.3 Modular Linear Equation . . . . .	8
4.4 Skip Lists . . . . .	4	9.4 Linear Diophantine Equation . . . . .	8
4.5 BIT + Search . . . . .	4	9.5 Modular Powers . . . . .	8
4.6 Segment Tree + Lazy Propagation . . . . .	4	9.6 Sieve of Eratosthenes . . . . .	8
4.7 Weighted Union Disjoint Sets . . . . .	5	9.7 Primality Testing & Factoring . . . . .	8
<b>5 Game Theory</b>	<b>5</b>	9.8 Euler Phi . . . . .	8
5.1 Nim Game . . . . .	5	9.9 Chinese Remainder . . . . .	8
5.2 Grundy Numbers . . . . .	5	9.10 Discrete Logarithm . . . . .	8
5.3 General Josephus Problem . . . . .	5	9.11 Tortoise & Hare . . . . .	8
<b>6 General Mathematics</b>	<b>5</b>	9.12 Pollard Rho . . . . .	9
6.1 Inclusion-Exclusion Patterns . . . . .	5	<b>10 Search</b>	<b>9</b>
6.2 Determinant . . . . .	5	10.1 Binary Search . . . . .	9
6.3 Gaussian Elimination . . . . .	5	10.2 Ternary Search . . . . .	9
6.4 Fast Fourier-Transform . . . . .	5	<b>11 Strings</b>	<b>9</b>
6.5 Misc. Formulas . . . . .	5	11.1 Aho Corasick . . . . .	9
		11.2 Hashing . . . . .	9
		11.3 Z-Algorithm . . . . .	9
		11.4 KMP + Periods . . . . .	9
		11.5 Manacher . . . . .	9
		11.6 Suffix Array . . . . .	9
		<b>12 Misc</b>	<b>9</b>
		<b>13 Last Page</b>	<b>10</b>

## 1 2D Geometry

### 1.1 Primitives

```

1 typedef complex<double> point;
2 struct circle {
3     point c; double r;
4     circle(point c, double r):c(c),r(r){}
5     circle(){}
6 };
7 double cross(const point &a, const point &b) {
8     return imag(conj(a)*b);
9 }
10 double dot(const point &a, const point &b) {
11     return real(conj(a)*b);
12 }

```

### 1.2 Intersections

```

1 // Line - Line
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Line - Segment
8 // Algorithm
9 // Algorithm
10 // Algorithm
11 // Algorithm
12 // Algorithm
13 // Segment - Segment
14 // Algorithm
15 // Algorithm
16 // Algorithm
17 // Algorithm
18 // Algorithm
19 // Circle - Line
20 // Algorithm
21 // Algorithm
22 // Algorithm
23 // Algorithm
24 // Algorithm
25 // Circle - Segment
26 // Algorithm
27 // Algorithm
28 // Algorithm
29 // Algorithm
30 // Algorithm
31 // Circle - Circle
32 // Algorithm
33 // Algorithm
34 // Algorithm
35 // Algorithm
36 // Algorithm
37 // Line - Point
38 // Algorithm
39 // Algorithm
40 // Algorithm
41 // Algorithm
42 // Algorithm
43 // Segment - Point
44 // Algorithm
45 // Algorithm
46 // Algorithm
47 // Algorithm
48 // Algorithm

```

### 1.3 Circle Generation

```

1 // From 3 Points
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // From 1 Line 2 Points
8 // Algorithm
9 // Algorithm
10 // Algorithm
11 // Algorithm
12 // Algorithm
13 // From 2 Lines 1 Point
14 // Algorithm
15 // Algorithm

```

```

16 // Algorithm
17 // Algorithm
18 // Algorithm
19 // From 3 Lines
20 // Algorithm
21 // Algorithm
22 // Algorithm
23 // Algorithm
24 // Algorithm

```

### 1.4 Heron Triangle Area

```

1 // Formula
2 // Formula
3 // Formula

```

### 1.5 Polygon Centroid

```

1 for(int i = 1; i < n-1; i++) {
2     pt ai = pts[i] - pts[i-1],
3     ib = pts[i+1] - pts[i];
4     area += (conj(ai)*ib).imag();
5 }

```

### 1.6 Point In Polygon

```

1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm
11 // Algorithm
12 // Algorithm
13 // Algorithm
14 // Algorithm
15 // Algorithm
16 // Algorithm
17 // Algorithm
18 // Algorithm
19 // Algorithm
20 // Algorithm

```

### 1.7 Convex Hull

```

1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm
11 // Algorithm
12 // Algorithm
13 // Algorithm
14 // Algorithm
15 // Algorithm
16 // Algorithm
17 // Algorithm
18 // Algorithm
19 // Algorithm
20 // Algorithm

```

### 1.8 Line Segment Set Intersection

```

1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm
11 // Algorithm
12 // Algorithm

```

```

13 // Algorithm
14 // Algorithm
15 // Algorithm
16 // Algorithm
17 // Algorithm
18 // Algorithm
19 // Algorithm
20 // Algorithm
21 // Algorithm
22 // Algorithm
23 // Algorithm
24 // Algorithm
25 // Algorithm
26 // Algorithm
27 // Algorithm
28 // Algorithm
29 // Algorithm
30 // Algorithm
31 // Algorithm
32 // Algorithm
33 // Algorithm
34 // Algorithm
35 // Algorithm
36 // Algorithm
37 // Algorithm
38 // Algorithm
39 // Algorithm
40 // Algorithm
41 // Algorithm
42 // Algorithm
43 // Algorithm
44 // Algorithm
45 // Algorithm
46 // Algorithm
47 // Algorithm
48 // Algorithm
49 // Algorithm

```

## 1.9 Voronoi Diagrams

```

1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm
11 // Algorithm
12 // Algorithm
13 // Algorithm
14 // Algorithm
15 // Algorithm
16 // Algorithm
17 // Algorithm
18 // Algorithm
19 // Algorithm
20 // Algorithm

```

## 2 3D Geometry

### 2.1 Primitives

```

1 // Code
2 // Code
3 // Code
4 // Code
5 // Code
6 // Code
7 // Code
8 // Code
9 // Code
10 // Code
11 // Code
12 // Code
13 // Code
14 // Code
15 // Code

```

### 2.2 Convex Hull

```

1 // Algorithm
2 // Algorithm

```

```

3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm
11 // Algorithm
12 // Algorithm
13 // Algorithm
14 // Algorithm
15 // Algorithm
16 // Algorithm
17 // Algorithm
18 // Algorithm
19 // Algorithm
20 // Algorithm

```

## 2.3 Great Circle Distance

```

1 // Code
2 // Code
3 // Code
4 // Code
5 // Code
6 // Code
7 // Code
8 // Code
9 // Code
10 // Code

```

## 3 Combinatorics

### 3.1 Basics

```

1 // catalan numbers
2 long long C(int n) {
3     return (C(n-1)*2*n*(2*n-1))/(n*(n+1));
4     return NCR(2*n, n) - NCR(2*n, n+1);
5     return NCR(2*n, n)/(n+1);
6 }
7 // derangements
8 long long D(int n) {
9     return n*D(n-1) + pow(-1, n);
10    return (n-1)*(D(n-1) + D(n-2));
11 }
12 // iterate over all subsets with < m elements
13 for (int i = 0; i < (1<<n); i=Integer.bitCount(i) < m ? i
14     +1 : (i|(i-1))+1)
15 // iterate over all the subsets
16 for (int i=0; i < (1<<n); i++)
17     // iterate over all the subsets of the i-th subset
18     for(int i2 = i; i2 > 0; i2 = (i2-1) & i)

```

### 3.2 Permutation (Un)Ranking

```

1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm
11 // Algorithm
12 // Algorithm
13 // Algorithm
14 // Algorithm
15 // Algorithm
16 // Algorithm
17 // Algorithm
18 // Algorithm
19 // Algorithm
20 // Algorithm

```

### 3.3 Combination (Un)Ranking

```

1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm

```

```

6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm
11 // Algorithm
12 // Algorithm
13 // Algorithm
14 // Algorithm
15 // Algorithm
16 // Algorithm
17 // Algorithm
18 // Algorithm
19 // Algorithm
20 // Algorithm

```

## 4 Data Structures

### 4.1 Palindromic Tree

```

1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm
11 // Algorithm
12 // Algorithm
13 // Algorithm
14 // Algorithm
15 // Algorithm
16 // Algorithm
17 // Algorithm
18 // Algorithm
19 // Algorithm
20 // Algorithm

```

### 4.2 Treap

```

1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm
11 // Algorithm
12 // Algorithm
13 // Algorithm
14 // Algorithm
15 // Algorithm
16 // Algorithm
17 // Algorithm
18 // Algorithm
19 // Algorithm
20 // Algorithm
21 // Algorithm
22 // Algorithm
23 // Algorithm
24 // Algorithm
25 // Algorithm
26 // Algorithm
27 // Algorithm
28 // Algorithm
29 // Algorithm
30 // Algorithm

```

### 4.3 Sparse Array

```

1 // Code
2 // Code
3 // Code
4 // Code
5 // Code
6 // Code
7 // Code
8 // Code
9 // Code

```

```

10 // Code
11 // Code
12 // Code
13 // Code
14 // Code
15 // Code

```

### 4.4 Skip Lists

```

1 // Code
2 // Code
3 // Code
4 // Code
5 // Code
6 // Code
7 // Code
8 // Code
9 // Code
10 // Code
11 // Code
12 // Code
13 // Code
14 // Code
15 // Code
16 // Code
17 // Code
18 // Code
19 // Code
20 // Code
21 // Code
22 // Code

```

### 4.5 BIT + Search

```

1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm
11 // Algorithm
12 // Algorithm
13 // Algorithm
14 // Algorithm
15 // Algorithm
16 // Algorithm
17 // Algorithm
18 // Algorithm
19 // Algorithm
20 // Algorithm

```

### 4.6 Segment Tree + Lazy Propagation

```

1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm
11 // Algorithm
12 // Algorithm
13 // Algorithm
14 // Algorithm
15 // Algorithm
16 // Algorithm
17 // Algorithm
18 // Algorithm
19 // Algorithm
20 // Algorithm
21 // Algorithm
22 // Algorithm
23 // Algorithm
24 // Algorithm
25 // Algorithm
26 // Algorithm
27 // Algorithm
28 // Algorithm

```

```
29 // Algorithm
30 // Algorithm
```

#### 4.7 Weighted Union Disjoint Sets

```
1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm
11 // Algorithm
12 // Algorithm
13 // Algorithm
14 // Algorithm
15 // Algorithm
```

### 5 Game Theory

#### 5.1 Nim Game

```
1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm
```

#### 5.2 Grundy Numbers

```
1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm
11 // Algorithm
12 // Algorithm
13 // Algorithm
14 // Algorithm
15 // Algorithm
16 // Algorithm
17 // Algorithm
18 // Algorithm
19 // Algorithm
20 // Algorithm
```

#### 5.3 General Josephus Problem

### 6 General Mathematics

#### 6.1 Inclusion-Exclusion Patterns

```
1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm
```

#### 6.2 Determinant

```
1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
```

```
9 // Algorithm
10 // Algorithm
11 // Algorithm
12 // Algorithm
13 // Algorithm
14 // Algorithm
15 // Algorithm
```

#### 6.3 Gaussian Elimination

```
1 double* GaussianElimination(int N, double **mat) {
2     int i, j, k, L; double t;
3     for (i = 0; i < N - 1; i++) {
4         L = i;
5         for (j = i + 1; j < N; j++)
6             if (fabs(mat[j][i]) > fabs(mat[L][i]))
7                 L = j;
8         for (k = i; k <= N; k++)
9             swap(mat[i][k], mat[L][k]);
10        for (j = i + 1; j < N; j++)
11            for (k = N; k >= i; k--)
12                mat[j][k] -= (mat[i][k] * mat[j][i]) / mat[i][i];
13    }
14    double *res = new double[N];
15    for (j = N - 1; j >= 0; j--) {
16        for (t = 0.0, k = j + 1; k < N; k++)
17            t += mat[j][k] * res[k];
18        res[j] = (mat[j][N] - t) / mat[j][j];
19    }
20    return res;
21 }
```

#### 6.4 Fast Fourier-Transform

```
1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm
```

#### 6.5 Misc. Formulas

```
1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm
11 // Algorithm
12 // Algorithm
13 // Algorithm
14 // Algorithm
15 // Algorithm
16 // Algorithm
17 // Algorithm
18 // Algorithm
19 // Algorithm
20 // Algorithm
21 // Algorithm
22 // Algorithm
23 // Algorithm
24 // Algorithm
25 // Algorithm
26 // Algorithm
27 // Algorithm
28 // Algorithm
29 // Algorithm
30 // Algorithm
```

### 7 Graph Theory

#### 7.1 Primitives

```
1 // Algorithm
2 // Algorithm
3 // Algorithm
```

```

4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm
11 // Algorithm
12 // Algorithm
13 // Algorithm
14 // Algorithm
15 // Algorithm

```

## 7.2 Articulation Points & Bridges

```

1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm
11 // Algorithm
12 // Algorithm
13 // Algorithm
14 // Algorithm
15 // Algorithm
16 // Algorithm
17 // Algorithm
18 // Algorithm
19 // Algorithm
20 // Algorithm

```

## 7.3 SCC

```

1 #define MAXN 100
2
3 int N;
4 vector< vector<int> > adj;
5
6 int num[MAXN], low[MAXN];
7 bool vis[MAXN];
8 vector<int> S;
9 vector< vector<int> > SCCs;
10 int dfsNumber;
11
12 void tarjanSCC(int u) {
13     low[u] = num[u] = dfsNumber++;
14     S.push_back(u);
15     vis[u] = 1;
16     for(int i=0; i<adj[u].size(); i++) {
17         int v = adj[u][i];
18         if(num[v] == -1)
19             tarjanSCC(v);
20         if(vis[v] == 1)
21             low[u] = min(low[u], low[v]);
22     }
23     if(low[u] == num[u]) {
24         vector<int> SCC;
25         while(1) {
26             int v = S.back();
27             S.pop_back();
28             vis[v] = 0;
29             SCC.push_back(v);
30             if(u == v)
31                 break;
32         }
33         SCCs.push_back(SCC);
34     }
35 }
36
37 void findSCC() {
38     dfsNumber = 0;
39     memset(vis, 0, sizeof(vis));
40     memset(num, -1, sizeof(num));
41     memset(low, 0, sizeof(low));
42     S.clear();
43     SCCs.clear();
44     for(int i=0; i<N; i++)
45         if(num[i] == -1)
46             tarjanSCC(i);
47 }

```

## 7.4 2-SAT

```

1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm
11 // Algorithm
12 // Algorithm
13 // Algorithm
14 // Algorithm
15 // Algorithm
16 // Algorithm
17 // Algorithm
18 // Algorithm
19 // Algorithm
20 // Algorithm

```

## 7.5 Edmonds-Karp Max Flow

```

1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm
11 // Algorithm
12 // Algorithm
13 // Algorithm
14 // Algorithm
15 // Algorithm
16 // Algorithm
17 // Algorithm
18 // Algorithm
19 // Algorithm
20 // Algorithm

```

## 7.6 Dinic's Max Flow

```

1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm
11 // Algorithm
12 // Algorithm
13 // Algorithm
14 // Algorithm
15 // Algorithm
16 // Algorithm
17 // Algorithm
18 // Algorithm
19 // Algorithm
20 // Algorithm

```

## 7.7 Min-Cost Max Flow

```

1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm
11 // Algorithm
12 // Algorithm
13 // Algorithm
14 // Algorithm

```

```

15 // Algorithm
16 // Algorithm
17 // Algorithm
18 // Algorithm
19 // Algorithm
20 // Algorithm

```

## 7.8 Euler Cycles

```

1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm
11 // Algorithm
12 // Algorithm
13 // Algorithm
14 // Algorithm
15 // Algorithm
16 // Algorithm
17 // Algorithm
18 // Algorithm
19 // Algorithm
20 // Algorithm

```

## 7.9 Maximum Matching

```

1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm
11 // Algorithm
12 // Algorithm
13 // Algorithm
14 // Algorithm
15 // Algorithm
16 // Algorithm
17 // Algorithm
18 // Algorithm
19 // Algorithm
20 // Algorithm

```

## 7.10 HL Decomposition

```

1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm
11 // Algorithm
12 // Algorithm
13 // Algorithm
14 // Algorithm
15 // Algorithm
16 // Algorithm
17 // Algorithm
18 // Algorithm
19 // Algorithm
20 // Algorithm

```

## 7.11 Modelling Inequalities

```

1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm

```

```

7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm

```

## 7.12 Max Flow Tricks

```

1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm
11 // Algorithm
12 // Algorithm
13 // Algorithm
14 // Algorithm
15 // Algorithm

```

## 7.13 Bellman Ford

```

1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm

```

## 7.14 Stable Marriage

```

1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm
11 // Algorithm
12 // Algorithm
13 // Algorithm
14 // Algorithm
15 // Algorithm

```

## 7.15 Maximum Assignment

```

1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm
11 // Algorithm
12 // Algorithm
13 // Algorithm
14 // Algorithm
15 // Algorithm

```

# 8 Linear Programming

## 8.1 Simplex

```

1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm

```



```

10 // Algorithm
11 // Algorithm
12 // Algorithm
13 // Algorithm
14 // Algorithm
15 // Algorithm
16 // Algorithm
17 // Algorithm
18 // Algorithm
19 // Algorithm
20 // Algorithm
21 // Algorithm
22 // Algorithm
23 // Algorithm
24 // Algorithm
25 // Algorithm
26 // Algorithm
27 // Algorithm
28 // Algorithm
29 // Algorithm
30 // Algorithm
31 // Algorithm

```

```

5 int sqrtN = (int) sqrt( (double) N );
6 for(int i = 2; i <= sqrtN; i++) if(prime[i]){
7     for( int j = i * i; j < N; j += i )
8         prime[j] = false;
9 }
10 } /*(fast, memory efficient version)
11 * gP(n) is non-zero iff n is prime.
12 * Requires N / 16 bytes of memory.
13 * WARNING! Only works for odd numbers.*/
14 #define N 51000000
15 unsigned int prime[N / 64];
16 #define gP(n) (prime[n>>6] & (1<<((n>>1)&31)))
17 #define rP(n) (prime[n>>6] & ~(1<<((n>>1)&31)))
18 void sieve() {
19     memset( prime, -1, sizeof( prime ) );
20     unsigned int i, i2, j,
21     sqrtN = (unsigned int) sqrt((double)N)+1;
22     for(i = 3; i < sqrtN; i += 2 ) if(gP(i)) {
23         i2 = i + i;
24         for(j = i*i; j < N; j+=i2) rP(j);
25     }
26 }

```

## 9 Number Theory

### 9.1 Extended GCD

```

1 typedef pair<int, int> ii;
2 long long gcd( long long a, long long b )
3 { return( b == 0 ? a : gcd( b, a % b ) ); }
4 //USED BY: egcd, msolve, inverse, ldioph
5 template< class Int > struct Triple {
6     Int d, x, y;
7     Triple( Int q, Int w, Int e ):d(q), x(w), y(e){}
8 }; //USED BY: msolve, inverse, ldioph
9 template< class Int > Triple< Int > egcd( Int a, Int b ) {
10     if( !b ) return Triple< Int > ( a, Int( 1 ), Int( 0 ) );
11     Triple< Int > q = egcd( b, a % b );
12     return Triple< Int > ( q.d, q.y, q.x - a / b * q.y );
13 }

```

### 9.2 Modular Inverse

```

1 //solves ax = 1 (mod n), O( log(an) )
2 template< class Int > Int inverse( Int a, Int n ){
3     Triple< Int > t = egcd( a, n );
4     if( t.d > Int( 1 ) ) return Int( 0 );
5     Int r = t.x % n;
6     return( r < Int( 0 ) ? r + n : r );
7 }

```

### 9.3 Modular Linear Equation

```

1 // solves ax = b (mod n), O( log(an) + gcd( a, n ) )
2 template<class Int> vector<Int> msolve( Int a, Int b, Int n ){
3     if( n < 0 ) n = -n;
4     Triple< Int > t = egcd( a, n );
5     vector< Int > r;
6     if( b % t.d ) return r;
7     Int x = ( b / t.d * t.x ) % n;
8     if( x < Int( 0 ) ) x += n;
9     for( Int i = 0; i < t.d; i++ )
10         r.push_back( ( x + i * n / t.d ) % n );
11     return r;
12 }

```

### 9.4 Linear Diophantine Equation

```

1 /* Solves ax + by = c. If .d == 0 -> no Solutions.
2 Otherwise:
3     x = t.x + k * b / t.d,
4     y = t.y - k * a / t.d; */
5 template<class Int> Triple<Int> ldioph( Int a, Int b, Int c ){
6     Triple< Int > t = egcd( a, b );
7     if( c % t.d ) return Triple< Int > ( 0, 0, 0 );
8     t.x *= c / t.d; t.y *= c / t.d;
9     return t;
10 }

```

### 9.5 Sieve of Eratosthenes

```

1 /*(simple, slow version) O( N.log(N) )*/
2 void sieve( bool prime[], int N ){
3     memset( prime, -1, N * sizeof( prime[0] ) );
4     prime[0] = prime[1] = false;

```

### 9.6 Primality Testing & Factoring

```

1 vector<ii> factor(long long N) {
2     vector<ii> res;
3     for(int i = 0, j = 0; primes[i]*primes[i] <= N; i++, j = 0) {
4         while(N % primes[i] == 0)
5             j++, N /= primes[i];
6         if(j) res.push_back(ii(primes[i], j));
7     }
8     return res;
9 }

```

### 9.7 Euler Phi

```

1 /* num of +ve ints < than n relatively prime to n. */
2 int phi(int n){
3     vector< ii > p = factor(n);
4     for( int i = 0; i < (int) p.size(); i++ )
5         n /= p[i].first, n *= p[i].first - 1;
6     return n;
7 }

```

### 9.8 Continued Fractions of Rationals

```

1 /*O( log n ) 1 + 1/x*/
2 void contFract( int m, int n, vector<int> &ans ){
3     while( n )
4         ans.push_back( m / n ),
5         m %= n, m ^= n ^= m ^= n; // swap(m, n)
6 }

```

### 9.9 Chinese Remainder

```

1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm

```

### 9.10 Discrete Logarithm

```

1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm

```

### 9.11 Tortoise & Hare



```

1 // mu = start of cycle, lambda = cycle length
2 ii floyd(int x0) {
3     int tortoise = f(x0), hare = f(f(x0));
4     while(tortoise != hare)
5         tortoise = f(tortoise), hare = f(f(hare));
6     int mu = 0; hare = x0;
7     while(tortoise != hare)
8         tortoise = f(tortoise), hare = f(hare), mu++;
9     int lambda = 1; hare = f(tortoise);
10    while(tortoise != hare)
11        hare = f(hare), lambda++;
12    return ii(mu, lambda);
13 }

```

## 9.12 Pollard Rho

```

1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm

```

## 10 Search

### 10.1 Binary Search

```

1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm
11 // Algorithm
12 // Algorithm
13 // Algorithm
14 // Algorithm
15 // Algorithm

```

### 10.2 Ternary Search

```

1 long double min() {
2     long double lo = -1e6, hi = 1e6, res = 3e6;
3     while(fabs(lo-hi) > EPS) {
4         long double left = (hi-lo)/3 + lo, right = (2*(hi-
5             lo))/3 + lo;
6         long double resL = F(left), resR = F(right);
7         if(resL < resR)
8             hi = right;
9         else
10            lo = left;
11        res = min(res, min(resL, resR));
12    }
13    return res;
14 }

```

## 11 Strings

### 11.1 Aho Corasick

```

1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm
11 // Algorithm
12 // Algorithm
13 // Algorithm
14 // Algorithm
15 // Algorithm

```

### 11.2 Hashing

```

1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm
11 // Algorithm
12 // Algorithm
13 // Algorithm
14 // Algorithm
15 // Algorithm

```

### 11.3 Z-Algorithm

```

1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm
11 // Algorithm
12 // Algorithm
13 // Algorithm
14 // Algorithm
15 // Algorithm

```

### 11.4 KMP + Periods

```

1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm
11 // Algorithm
12 // Algorithm
13 // Algorithm
14 // Algorithm
15 // Algorithm

```

### 11.5 Manacher

```

1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm
7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm
11 // Algorithm
12 // Algorithm
13 // Algorithm
14 // Algorithm
15 // Algorithm
16 // Algorithm
17 // Algorithm
18 // Algorithm
19 // Algorithm
20 // Algorithm
21 // Algorithm

```

### 11.6 Suffix Array

```

1 // Algorithm
2 // Algorithm
3 // Algorithm
4 // Algorithm
5 // Algorithm
6 // Algorithm

```

```

7 // Algorithm
8 // Algorithm
9 // Algorithm
10 // Algorithm
11 // Algorithm
12 // Algorithm
13 // Algorithm
14 // Algorithm
15 // Algorithm
16 // Algorithm
17 // Algorithm
18 // Algorithm
19 // Algorithm
20 // Algorithm
21 // Algorithm
22 // Algorithm
23 // Algorithm
24 // Algorithm
25 // Algorithm
26 // Algorithm
27 // Algorithm
28 // Algorithm
29 // Algorithm
30 // Algorithm

```

## 12 Misc

## 13 Last Page

Cut this paper out. Use it wisely!

Problem	Tags
01 A	
02 B	
03 C	
04 D	
05 E	
06 F	
07 G	
08 H	
09 I	
10 J	
11 K	
12 L	
13 M	

Time	Meeting Description	Chk
030	All Problems Read. Write Tags.	
060	Ace Decided. Choose Coder.	
090	Decide & Order Solveable Problems	
120	Status Check	
150	Status Check	
180	Status Check	
210	Status Check	
240	Blind Hour. One Problem.	
270	Status Check	
300	Contest Ends	

## Solving A Problem

Read the statement carefully.

Break the problem down into pieces.

Plan the solution to each piece.

Think of corner cases to solution.

Calculate Complexity.

Simplify the solution.

Write steps of solution on paper.

Estimate Coding time.

## During The Contest

Stay **calm** and **focused** or you and your team mates won't make it

If you don't understand it, it doesn't mean it's hard.

Tell your team mate.

READ THE STATEMENT AGAIN. TELL YOUR TEAMMATE!

Write **significant** tags! Think how all the topics might come in use!

Run over the index page. Maybe one of the topic titles will inspire a solution!

## Common Bugs

Add eps to double before getting floor or round