Contents

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	Check
030	All Problems Read. Write Tags.	
060	Ace Decided. Choose Coder.	
120	Decide & Order Solveable Problems	
150	Status Check	
180	Status Check	
210	Status Check	
240	Status Check	
270	Status Check	

1 2D Geometry

1.1 Primitives

```cpp
typedef complex<double> point;
struct circle {
  point c; double r;
  circle(point c, double r):c(c),r(r){}
  circle(){}
};
double cross(const point &a, const point &b) {
  return imag(conj(a)*b);
}
double dot(const point &a, const point &b) {
  return real(conj(a)*b);
}
```

1.2 Intersections

```cpp
// Line - Line
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Line - Segment
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Segment - Segment
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Circle - Line
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Circle - Segment
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Circle - Circle
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Line - Point
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Segment - Point
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

1.3 Circle Generation

```cpp
// From 3 Points
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// From 1 Line 2 Points
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// From 2 Lines 1 Point
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// From 3 Lines
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

1.4 Heron Triangle Area

```cpp
// Formula
// Formula
// Formula
```

1.5 Polygon Centroid

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

1.6 Point In Polygon

```cpp
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

1.7 Convex Hull

```cpp
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

1.8 Line Segment Set Intersection

```cpp
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

1.9 Voronoi Diagrams

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

2 3D Geometry

2.1 Primitives

2.2 Convex Hull

2.3 Great Circle Distance

3 Combinatorics

3.1 Basics

```
// catalan numbers
long long C(int n) {
  return (C(n-1)*2*n*(2*n-1))/(n*(n+1));
  return NCR(2*n, n) - NCR(2*n, n+1);
  return NCR(2*n, n)/(n+1);
}
// derangements
long long D(int n) {
  return n*D(n-1) + pow(-1, n);
  return (n-1)*(D(n-1) + D(n-2));
}
// iterate over all subsets with < m elements
for (int i = 0; i < (1<<n); i=Integer.bitCount(i) < m ? i
    +1 : (i|(i-1))+1)
```

```
// iterate over all the subsets
for (int i=0; i < (1<<n); i++)
  // iterate over all the subsets of the i-th subset
  for(int i2 = i; i2 > 0; i2 = (i2-1) & i)
```

4 Data Structures

4.1 Palindromic Tree

4.2 Treap

4.3 Sparse Array

4.4 Skip Lists

5 Game Theory

5.1 Nim Game

5.2 Grundy Numbers

6 Graph Theory

6.1 Articulation Points & Bridges

6.2 SCC

6.3 2-SAT

6.4 Edmonds-Karp Max Flow

6.5 Dinic's Max Flow

6.6 Min-Cist Max Flow

6.7 Euler Cycles

6.8 Maximum Matching

6.9 HL Decomposition

7 Linear Programming

7.1 Simplex

8 Number Theory

8.1 Extended GCD

```
long long gcd( long long a, long long b )
{ return( b == 0 ? a : gcd( b, a % b ) ); }
//USED BY: egcd, msolve, inverse, ldioph
template< class Int > struct Triple {
  Int d, x, y;
  Triple(Int q, Int w, Int e):d(q), x(w), y(e){}
};
//USED BY: msolve, inverse, ldioph
template< class Int > Triple< Int > egcd( Int a, Int b ) {
  if( !b ) return Triple< Int >( a, Int( 1 ), Int( 0 ) );
  Triple< Int > q = egcd( b, a % b );
  return Triple< Int >( q.d, q.y, q.x - a / b * q.y );
}
```

8.2 Modular Inverse

```
//solves ax = 1 (mod n).
template< class Int > Int inverse( Int a, Int n ) {
    Triple< Int > t = egcd( a, n );
    if( t.d > Int( 1 ) ) return Int( 0 );
    Int r = t.x % n;
    return( r < Int( 0 ) ? r + n : r );
}
```

8.3 Modular Linear Equation

8.4 Linear Diophantine Equation

8.5 Modular Powers

8.6 Sieve of Eratosthenes

8.7 Primality Testing & Factoring

8.8 Euler Phi

8.9 Chinese Remainder

8.10 Discerete Logarithm

8.11 Gaussian Elimination

8.12 Fast Fourier-Transform

```
double* GaussianElimination(int N, double **mat) {
  int i, j, k, L; double t;
  for (i = 0; i < N - 1; i++) {
    L = i;
    for (j = i + 1; j < N; j++)
      if (fabs(mat[j][i]) > fabs(mat[L][i]))
        L = j;
    for (k = i; k <= N; k++)
    swap(mat[i][k], mat[L][k]);
    for (j = i + 1; j < N; j++)
      for (k = N; k >= i; k--)
        mat[j][k] -= (mat[i][k] * mat[j][i]) / mat[i][i];
  }
  double *res = new double[N];
  for (j = N - 1; j >= 0; j--) {
    for (t = 0.0, k = j + 1; k < N; k++)
    t += mat[j][k] * res[k];
    res[j] = (mat[j][N] - t) / mat[j][j];
  }
  return res;
}
```

8.13 Tortoise & Hare

```
 // mu = start of cycle, lambda = cycle length
 ii floyd(int x0) {
  int tortoise = f(x0), hare = f(f(x0));
  while(tortoise != hare)
    tortoise = f(tortoise), hare = f(f(hare));
  int mu = 0; hare = x0;
  while(tortoise != hare)
    tortoise = f(tortoise), hare = f(hare), mu++;
  int lambda = 1; hare = f(tortoise);
  while(tortoise != hare)
    hare = f(hare), lambda++;
  return ii(mu, lambda);
 }
```

9 Search

9.1 Binary Search

9.2 Ternary Search

```
long double min() {
    long double lo = -1e6, hi = 1e6, res = 3e6;
    while(fabs(lo-hi) > EPS) {
        long double left = (hi-lo)/3 + lo, right = (2*(hi-lo))/3 + lo;
        long double resL = F(left), resR = F(right);
        if(resL < resR)
            hi = right;
        else
            lo = left;
        res = min(res, min(resL, resR));
    }
    return res;
}
```

10 Strings

10.1 Aho Corasick

10.2 Hashing

10.3 KMP

10.4 Suffix Array