

**Contents**

<b>1</b>	<b>2D Geometry</b>	<b>2</b>	<b>9</b>	<b>Search</b>	<b>2</b>
1.1	Primitives . . . . .	2	9.1	Binary Search . . . . .	2
1.2	Circle Intersection . . . . .	2	9.2	Ternary Search . . . . .	2
1.3	Line-Circle Intersection . . . . .	2	<b>10</b>	<b>Strings</b>	<b>3</b>
1.4	Line Intersection . . . . .	2	10.1	Aho Corasick . . . . .	3
1.5	Segment Intersection . . . . .	2	10.2	Hashing . . . . .	3
1.6	Parabola-Line Intersection . . . . .	2	10.3	KMP . . . . .	3
1.7	Circle Generation . . . . .	2	10.4	Suffix Array . . . . .	3
1.8	Heron Triangle Area . . . . .	2			
1.9	Polygon Centroid . . . . .	2			
1.10	Point In Polygon . . . . .	2			
1.11	Convex Hull . . . . .	2			
1.12	Full Line Segment Intersection . . . . .	2			
1.13	Voronoi . . . . .	2			
<b>2</b>	<b>3D Geometry</b>	<b>2</b>			
2.1	Primitives . . . . .	2			
2.2	Convex Hull . . . . .	2			
2.3	Great Circle Distance . . . . .	2			
<b>3</b>	<b>Combinatorics</b>	<b>2</b>			
3.1	Basics . . . . .	2			
<b>4</b>	<b>Data Structures</b>	<b>2</b>			
4.1	Palindromic Tree . . . . .	2			
4.2	Treap . . . . .	2			
4.3	Sparse Array . . . . .	2			
4.4	Skip Lists . . . . .	2			
<b>5</b>	<b>Game Theory</b>	<b>2</b>			
5.1	Nim Game . . . . .	2			
5.2	Grundy Numbers . . . . .	2			
<b>6</b>	<b>Graph Theory</b>	<b>2</b>			
6.1	Articulation Points & Bridges . . . . .	2			
6.2	SCC . . . . .	2			
6.3	2-SAT . . . . .	2			
6.4	Edmonds-Karp Max Flow . . . . .	2			
6.5	Dinic's Max Flow . . . . .	2			
6.6	Min-Cost Max Flow . . . . .	2			
6.7	Euler Cycles . . . . .	2			
6.8	Maximum Matching . . . . .	2			
6.9	HL Decomposition . . . . .	2			
<b>7</b>	<b>Linear Programming</b>	<b>2</b>			
7.1	Simplex . . . . .	2			
<b>8</b>	<b>Number Theory</b>	<b>2</b>			
8.1	Extended GCD . . . . .	2			
8.2	Sieve of Eratosthenes . . . . .	2			
8.3	Chinese Remainder . . . . .	2			
8.4	Modular Inverse . . . . .	2			
8.5	Discrete Logarithm . . . . .	2			
8.6	Gaussian Elimination . . . . .	2			
8.7	Fast Fourier-Transform . . . . .	2			
8.8	Tortoise & Hare . . . . .	2			

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

```

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 Circle Intersection

### 1.3 Line-Circle Intersection

### 1.4 Line Intersection

### 1.5 Segment Intersection

### 1.6 Parabola-Line Intersection

### 1.7 Circle Generation

### 1.8 Heron Triangle Area

### 1.9 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 }

```

### 1.10 Point In Polygon

### 1.11 Convex Hull

### 1.12 Full Line Segment Intersection

### 1.13 Voronoi

## 2 3D Geometry

### 2.1 Primitives

### 2.2 Convex Hull

### 2.3 Great Circle Distance

## 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
8 // derangements
9 long long D(int n) {
10    return n*D(n-1) + pow(-1, n);
11    return (n-1)*(D(n-1) + D(n-2));
12 }
13
14 // iterate over all the subsets with no more than m
15 // elements
16 for (int i = 0; i < (1<n); i=Integer.bitCount(i) < m ? i
17     +1 : (i|(i-1))+1)
18
19 // iterate over all the subsets
20 for (int i=0; i < (1<n); i++)
21     // iterate over all the subsets of the i-th subset
22     for(int i2 = i; i2 > 0; i2 = (i2-1) & i)
23         // generate the subset induced by i2

```

## 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-Cost 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

### 8.2 Sieve of Eratosthenes

### 8.3 Chinese Remainder

### 8.4 Modular Inverse

### 8.5 Discrete Logarithm

### 8.6 Gaussian Elimination

### 8.7 Fast Fourier-Transform

```

1 double* GaussianElimination(int N, double **mat) {
2     int i, j, k, l; double t;
3
4     for (i = 0; i < N - 1; i++) {
5         l = i;
6         for (j = i + 1; j < N; j++)
7             if (fabs(mat[j][i]) > fabs(mat[l][i]))
8                 l = j;
9         // partial pivot
10        for (k = i; k <= N; k++)
11            swap(mat[i][k], mat[l][k]);
12        for (j = i + 1; j < N; j++)
13            for (k = N; k >= i; k--)
14                mat[j][k] -= (mat[i][k] * mat[j][i]) / mat[i][i];
15    }
16
17    double *res = new double[N];
18    for (j = N - 1; j >= 0; j--) {
19        for (t = 0.0, k = j + 1; k < N; k++)
20            t += mat[j][k] * res[k];
21        res[j] = (mat[j][N] - t) / mat[j][j]; // the answer is
22        here
23    }
24    return res;
25 }

```

### 8.8 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 Search

### 9.1 Binary Search

### 9.2 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)

```

```
7         hi = right;
8     else
9         lo = left;
10    res = min(res, min(resL, resR));
11    }
12    return res;
13 }
```

## 10 Strings

### 10.1 Aho Corasick

### 10.2 Hashing

### 10.3 KMP

### 10.4 Suffix Array