



Contents

1 2D Geometry	2
1.1 Primitives	2
1.2 Intersections	2
1.3 Circle Generation	2
1.4 Heron Triangle Area	2
1.5 Polygon Centroid	2
1.6 Point In Polygon	2
1.7 Convex Hull	2
1.8 Line Segment Set Intersection	2
1.9 Voronoi Diagrams	2
2 3D Geometry	2
2.1 Primitives	2
2.2 Convex Hull	2
2.3 Great Circle Distance	2
3 Combinatorics	2
3.1 Basics	2
4 Data Structures	2
4.1 Palindromic Tree	2
4.2 Treap	2
4.3 Sparse Array	2
4.4 Skip Lists	2
5 Game Theory	2
5.1 Nim Game	2
5.2 Grundy Numbers	2
6 Graph Theory	2
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
7 Linear Programming	2
7.1 Simplex	2

8 Number Theory	2
8.1 Extended GCD	2
8.2 Modular Inverse	2
8.3 Modular Linear Equation	2
8.4 Linear Diophantine Equation	2
8.5 Modular Powers	2
8.6 Sieve of Eratosthenes	2
8.7 Primality Testing & Factoring	2
8.8 Euler Phi	2
8.9 Chinese Remainder	2
8.10 Discrete Logarithm	2
8.11 Gaussian Elimination	2
8.12 Fast Fourier-Transform	2
8.13 Tortoise & Hare	2

9 Search	3
9.1 Binary Search	3
9.2 Ternary Search	3

10 Strings	3
10.1 Aho Corasick	3
10.2 Hashing	3
10.3 KMP	3
10.4 Suffix Array	3

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 Solvable 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 Intersections

```

1 // Line - Line
2 // Line - Segment
3 // Segment - Segment
4 // Circle - Line
5 // Circle - Segment
6 // Circle - Circle
7 // Line - Point
8 // Segment - Point

```

1.3 Circle Generation

```

1 // From 3 Points
2 // From 1 Line 2 Points
3 // From 2 Lines 1 Point
4 // From 3 Lines

```

1.4 Heron Triangle Area

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

1.7 Convex Hull

1.8 Line Segment Set Intersection

1.9 Voronoi Diagrams

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
// elements
15 for (int i = 0; i < (1<<n); i=Integer.bitCount(i) <= m ? i
+1 : (i|(i-1))+1)
16
17 // iterate over all the subsets
18 for (int i=0; i < (1<<n); i++)
19     // iterate over all the subsets of the i-th subset
20     for(int i2 = i; i2 > 0; i2 = (i2-1) & i)
21         // 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 Modular Inverse

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 Discrete Logarithm

8.11 Gaussian Elimination

8.12 Fast Fourier-Transform

```

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 }

```

8.13 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)  
8             hi = right;  
9         else  
10            lo = left;  
11        res = min(res, min(resL, resR));  
12    }  
13    return res;  
}
```

10 Strings

10.1 Aho Corasick

10.2 Hashing

10.3 KMP

10.4 Suffix Array