

Luna's Magic Reference

Suzune Nisiyama

July 18, 2018

MIT License

Copyright (c) 2018 Nisiyama-Suzune

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Contents

1 Environment	2	8.4 Stack hack	12
1.1 Vimrc	2	8.5 Time hack	12
2 Data Structure	2	8.6 Builtin functions	12
2.1 KD tree	2	8.7 Prufer sequence	12
2.2 Splay	2	8.8 Spanning tree counting	12
2.3 Link-cut tree	2	8.9 Mobius inversion	12
3 Formula	2	8.10 2-SAT	12
3.1 Zellers congruence	2	8.11 Interesting numbers	13
3.2 Lattice points below segment	2	9 Appendix	14
3.3 Adaptive Simpson's method	3	9.1 Calculus table	14
3.4 Neural network	3	9.2 Regular expression	15
4 Number theory	3		
4.1 Fast power module	3		
4.2 Euclidean algorithm	3		
4.3 Discrete Fourier transform	3		
4.4 Fast Walsh-Hadamard transform	3		
4.5 Number theoretic transform	3		
4.6 Chinese remainder theorem	4		
4.7 Linear Recurrence	4		
4.8 Berlekamp Massey algorithm	4		
4.9 Baby step giant step algorithm	4		
4.10 Miller Rabin primality test	4		
4.11 Pollard's Rho algorithm	4		
5 Geometry	5		
5.1 Point	5		
5.2 Line	5		
5.3 Circle	5		
5.4 Centers of a triangle	5		
5.5 Fermat point	5		
5.6 Convex hull	6		
5.7 Half plane intersection	6		
5.8 Nearest pair of points	6		
5.9 Minimum circle	6		
5.10 Intersection of a polygon and a circle	6		
5.11 Union of circles	6		
5.12 3D point	6		
5.13 3D line	7		
5.14 3D convex hull	7		
6 Graph	7		
6.1 Hopcroft-Karp algorithm	7		
6.2 Kuhn-Munkres algorithm	7		
6.3 Blossom algorithm	7		
6.4 Weighted blossom algorithm	8		
6.5 Maximum flow	9		
6.6 Minimum cost flow	9		
6.7 Stoer Wagner algorithm	9		
6.8 DN maximum clique	10		
6.9 Dominator tree	10		
6.10 Tarjan	10		
7 String	10		
7.1 Suffix Array	10		
7.2 Suffix Automaton	10		
7.3 Palindromic tree	11		
7.4 Regular expression	11		
8 Tips	11		
8.1 Java	11		
8.2 Random numbers	12		
8.3 Read hack	12		

1 Environment

1.1 Vimrc

```

1 set ru nu ts=4 sts=4 sw=4 si sm hls is ar bs=2 mouse=a
2 syntax on
3 nm <F3> :vsplit %<.in <CR>
4 nm <F4> :!gedit % <CR>
5 au BufEnter *.cpp set cin
6 au BufEnter *.cpp nm <F5> :!time ./%< <CR>|nm <F7> :!
   gdb ./%< <CR>|nm <F8> :!time ./%< <CR>|nm <F9> :!g++ % -o % -g -std=gnu++14 -O2 -DLOCAL &&
   size %< <CR>
7 au BufEnter *.java nm <F5> :!time java %< <CR>|nm <F8>
   :!time java %< <CR>|nm <F9> :!javac % <CR>

```

2 Data Structure

2.1 KD tree

```

1 /* kd_tree : finds the k-th closest point in  $O(kn^{1-\frac{1}{k}})$ .
2 Usage : Stores the data in p[]. Call function init (n,
   k). Call min_kth (d, k). (or max_kth) (k is 1-
   based)
3 Note : Switch to the commented code for Manhattan
   distance.
4 Status : SPOJ-FAILURE Accepted.*/
5 template <int MAXN = 200000, int MAXK = 2>
6 struct kd_tree {
7     int k, size;
8     struct point { int data[MAXN], id; } p[MAXN];
9     struct kd_node {
10         int l, r; point p, dmin, dmax;
11         kd_node() {}
12         kd_node (const point &rhs) : l (-1), r (-1), p (rhs)
13             , dmin (rhs), dmax (rhs) {}
14     void merge (const kd_node &rhs, int k) {
15         for (register int i = 0; i < k; ++i) {
16             dmin.data[i] = std::min (dmin.data[i], rhs.dmin.
17                 data[i]);
18             dmax.data[i] = std::max (dmax.data[i], rhs.dmax.
19                 data[i]); } }
20     long long min_dist (const point &rhs, int k) const {
21         register long long ret = 0;
22         for (register int i = 0; i < k; ++i) {
23             if (dmin.data[i] <= rhs.data[i] && rhs.data[i] <=
24                 dmax.data[i]) continue;
25             ret += std::min (1ll * (dmin.data[i] - rhs.data[i]
26                 ) * (dmin.data[i] - rhs.data[i]),
27                 1ll * (dmax.data[i] - rhs.data[i]) * (dmax.
28                     data[i] - rhs.data[i]));
29             // ret += std::max (0, rhs.data[i] - dmax.data[i])
30             // + std::max (0, dmin.data[i] - rhs.data[i]);
31         } return ret; }
32     long long max_dist (const point &rhs, int k) {
33         long long ret = 0;
34         for (int i = 0; i < k; ++i) {
35             int tmp = std::max (std::abs (dmin.data[i] - rhs.
36                 data[i]), std::abs (dmax.data[i] - rhs.data[i]
37                 ));
38             ret += 1ll * tmp * tmp; }
39             // ret += std::max (std::abs (rhs.data[i] - dmax.
40             // data[i]) + std::abs (rhs.data[i] - dmin.data[i]));
41         } return ret; } } tree[MAXN * 4];
42 struct result {
43     long long dist; point d; result() {}
44     result (const long long &dist, const point &d) :
45         dist (dist), d (d) {}
46     bool operator > (const result &rhs) const { return
47         dist > rhs.dist || (dist == rhs.dist && d.id >
48             rhs.d.id); }
49     bool operator < (const result &rhs) const { return
50         dist < rhs.dist || (dist == rhs.dist && d.id <
51             rhs.d.id); } };
52 long long sqrdist (const point &a, const point &b) {
53     long long ret = 0;
54     for (int i = 0; i < k; ++i) ret += 1ll * (a.data[i]
55         - b.data[i]) * (a.data[i] - b.data[i]);
56     // for (int i = 0; i < k; ++i) ret += std::abs (a.
57     // data[i] - b.data[i]);
58     return ret; }
59 int alloc() { tree[size].l = tree[size].r = -1;
60     return size++; }
61 void build (const int &depth, int &rt, const int &l,
62     const int &r) {
63     if (l > r) return;
64     register int middle = (l + r) >> 1;
65     std::nth_element (p + l, p + middle, p + r + 1, [=]
66         (const point &a, const point &b) { return a.
67             data[depth] < b.data[depth]; });
68     tree[rt] = alloc(); = kd_node (p[middle]);
69     if (l == r) return;
70     build ((depth + 1) % k, tree[rt].l, l, middle - 1);
71     build ((depth + 1) % k, tree[rt].r, middle + 1, r);
72     if (!tree[rt].l) tree[rt].merge (tree[tree[rt].l], k
73         );
74     if (!tree[rt].r) tree[rt].merge (tree[tree[rt].r], k
75         );
76     std::priority_queue<result, std::vector<result>, std
77         ::less<result>> heap_l;
78     std::priority_queue<result, std::vector<result>, std
79         ::greater<result>> heap_r;
80     void min_kth (const int &depth, const int &rt, const
81         int &m, const point &d) {
82         result tmp = result (sqrdist (tree[rt].p, d), tree[
83             rt].p);
84         if ((int)heap_l.size() < m) heap_l.push (tmp);
85         else if (tmp < heap_l.top()) {
86             heap_l.pop();
87             heap_l.push (tmp); }
88     }
89     void max_kth (const int &depth, const int &rt, const
90         int &m, const point &d) {
91         result tmp = result (sqrdist (tree[rt].p, d), tree[
92             rt].p);
93         if ((int)heap_r.size() < m) heap_r.push (tmp);
94         else if (tmp > heap_r.top()) {
95             heap_r.pop();
96             heap_r.push (tmp); }
97     }
98     int x = tree[rt].l, y = tree[rt].r;
99     if ('x' && ~y && sqrdist (d, tree[x].p) < sqrdist (d,
100         tree[y].p)) std::swap (x, y);
101     if (~y && ((int)heap_l.size() < m || tree[x].
102         min_dist (d, k) < heap_l.top().dist))
103         min_kth ((depth + 1) % k, x, m, d);
104     if (~y && ((int)heap_r.size() < m || tree[y].
105         min_dist (d, k) < heap_r.top().dist))
106         min_kth ((depth + 1) % k, y, m, d); }
107 void max_kth (const int &depth, const int &rt, const
108     int &m, const point &d) {
109     result tmp = result (sqrdist (tree[rt].p, d), tree[
110         rt].p);
111     if ((int)heap_r.size() < m) heap_r.push (tmp);
112     else if (tmp > heap_r.top()) {
113         heap_r.pop();
114         heap_r.push (tmp); }
115     int x = tree[rt].l, y = tree[rt].r;
116     if ('x' && ~y && sqrdist (d, tree[x].p) < sqrdist (d,
117         tree[y].p)) std::swap (x, y);
118     if (~y && ((int)heap_r.size() < m || tree[x].
119         max_dist (d, k) >= heap_r.top().dist))
120         max_kth ((depth + 1) % k, x, m, d);
121     if (~y && ((int)heap_l.size() < m || tree[y].
122         max_dist (d, k) >= heap_r.top().dist))
123         max_kth ((depth + 1) % k, y, m, d); }
124 void init (int n, int k) { this->k = k; size = 0;
125     int rt = 0; build (0, rt, 0, n - 1); }
126 result min_kth (const point &d, const int &m) {
127     heap_l = decltype (heap_l) (); min_kth (0, 0, m,
128         d); return heap_l.top (); }
129 result max_kth (const point &d, const int &m) {
130     heap_r = decltype (heap_r) (); max_kth (0, 0, m,
131         d); return heap_r.top (); } }

```

```

61 int x = tree[rt].l, y = tree[rt].r;
62 if ('x' && ~y && sqrdist (d, tree[x].p) > sqrdist (d,
63     tree[y].p)) std::swap (x, y);
64 if (~y && ((int)heap_l.size() < m || tree[x].
65     min_dist (d, k) < heap_l.top().dist))
66     min_kth ((depth + 1) % k, x, m, d);
67 if (~y && ((int)heap_r.size() < m || tree[y].
68     min_dist (d, k) < heap_r.top().dist))
69     min_kth ((depth + 1) % k, y, m, d); }
70 void max_kth (const int &depth, const int &rt, const
71     int &m, const point &d) {
72     result tmp = result (sqrdist (tree[rt].p, d), tree[
73         rt].p);
74     if ((int)heap_r.size() < m) heap_r.push (tmp);
75     else if (tmp > heap_r.top()) {
76         heap_r.pop();
77         heap_r.push (tmp); }
78     int x = tree[rt].l, y = tree[rt].r;
79     if ('x' && ~y && sqrdist (d, tree[x].p) < sqrdist (d,
80         tree[y].p)) std::swap (x, y);
81     if (~y && ((int)heap_r.size() < m || tree[x].
82         max_dist (d, k) >= heap_r.top().dist))
83         max_kth ((depth + 1) % k, x, m, d);
84     if (~y && ((int)heap_l.size() < m || tree[y].
85         max_dist (d, k) >= heap_r.top().dist))
86         max_kth ((depth + 1) % k, y, m, d); }
87 void init (int n, int k) { this->k = k; size = 0;
88     int rt = 0; build (0, rt, 0, n - 1); }
89 result min_kth (const point &d, const int &m) {
90     heap_l = decltype (heap_l) (); min_kth (0, 0, m,
91         d); return heap_l.top (); }
92 result max_kth (const point &d, const int &m) {
93     heap_r = decltype (heap_r) (); max_kth (0, 0, m,
94         d); return heap_r.top (); } }

```

2.2 Splay

```

1 void push_down (int x) {
2     if (~n[x].c[0]) push (n[x].c[0], n[x].t);
3     if (~n[x].c[1]) push (n[x].c[1], n[x].t);
4     n[x].t = tag (); }
5 void update (int x) {
6     n[x].m = gen (x);
7     if (~n[x].c[0]) n[x].m = merge (n[n[x].c[0]].m, n[x].
8         m);
9     if (~n[x].c[1]) n[x].m = merge (n[x].m, n[n[x].c[1]].
10         m); }
11 void rotate (int x, int k) {
12     push_down (x); push_down (n[x].c[k]);
13     int y = n[x].c[k]; n[x].c[k] = n[y].c[k ^ 1]; n[y].c[
14         k ^ 1] = x;
15     if (n[x].f != -1) n[n[x].f].c[n[n[x].f].c[1] == x] =
16         y;
17     n[y].f = n[x].f; n[x].f = y; if (~n[x].c[k]) n[n[x].c
18         [k]].f = x;
19     update (x); update (y); }
20 void splay (int x, int s = -1) {
21     push_down (x);
22     while (n[x].f != s) {
23         if (n[n[x].f].f != s) rotate (n[n[x].f].f, n[n[x].
24             f].f.c[1] == n[x].f);
25         rotate (n[x].f, n[n[x].f].c[1] == x); }
26     update (x);
27     if (s == -1) root = x; }

```

2.3 Link-cut tree

```

1 void access (int x) {
2     int u = x, v = -1;
3     while (u != -1) {
4         splay (u); push_down (u);
5         if (~n[u].c[1]) n[n[u].c[1]].f = -1, n[n[u].c[1]].p
6             = u;
7         n[u].c[1] = v;
8         if (~v) n[v].f = u, n[v].p = -1;
9         update (u); u = n[v = u].p; }
10     splay (x); }

```

3 Formula

3.1 Zellers congruence

```

1 /* Zeller's congruence : converts between a calendar
2    date and its Gregorian calendar day. (y >= 1) (0 =
3    Monday, 1 = Tuesday, ..., 6 = Sunday) */
4 int get_id (int y, int m, int d) {
5     if (m < 3) { --y; m += 12; }
6     return 365 * y + y / 4 - y / 100 + y / 400 + (153 * (
7         m - 3) + 2) / 5 + d - 307; }
8 std::tuple <int, int, int> date (int id) {
9     int x = id + 1789995, n, i, j, y, m, d;
10    n = 4 * x / 146097; x -= (146097 * n + 3) / 4;
11    i = (4000 * (x + 1)) / 1461001; x -= 1461 * i / 4 -
12        31;
13    j = 80 * x / 2447; d = x - 2447 * j / 80;
14    x = j / 11;
15    m = j + 2 - 12 * x; y = 100 * (n - 49) + i + x;
16    return std::make_tuple (y, m, d); }

```

3.2 Lattice points below segment

```

1 /* Euclidean-like algorithm : computes the sum of
2     $\sum_{i=0}^{n-1} \lfloor \frac{a+bi}{m} \rfloor$ . */
3 long long solve (long long n, long long a, long long b,
4     long long m) {
5     if (b == 0) return n * (a / m);
6     if (a >= m) return n * (a / m) + solve (n, a % m, b,
7         m);
8 }

```

```

5 if (b >= m) return (n - 1) * n / 2 * (b / m) + solve
   (n, a, b % m, m);
6 return solve ((a + b * n) / m, (a + b * n) % m, m, b)
   ; }

```

3.3 Adaptive Simpson's method

```

1 /* Adaptive Simpson's method : integrates f in [l, r]. */
2 struct simpson {
3     double area (double (*f) (double), double l, double r
4     ) {
5         double m = 1 + (r - l) / 2;
6         return (f(l) + 4 * f(m) + f(r)) * (r - l) / 6; }
7     double solve (double (*f) (double), double l, double
8     r, double eps, double a) {
9         double m = 1 + (r - l) / 2;
10        double left = area (f, l, m), right = area (f, m, r)
11        ;
12        if (fabs (left + right - a) <= 15 * eps) return left
13        + right + (left + right - a) / 15.0;
14        return solve (f, l, m, eps / 2, left) + solve (f, m,
15        r, eps / 2, right); }
16    double solve (double (*f) (double), double l, double
17    r, double eps) {
18        return solve (f, l, r, eps, area (f, l, r)); } };

```

3.4 Neural network

```

1 /* Neural network : machine learning. */
2 template <int ft = 6, int n = 6, int MAXDATA = 100000>
3 struct network {
4     double wp[n][ft], w[n], avg[ft + 1], sig[ft + 1], val
5     [n];
6     network () {
7         std::mt19937_64 mt (time (0));
8         std::uniform_real_distribution <double> urdp (0, 2 *
9         sqrt (ft));
10        std::uniform_real_distribution <double> urdn (0, 2 *
11        sqrt (n));
12        for (int i = 0; i < n; ++i) for (int j = 0; j < ft;
13        ++j) wp[i][j] = urdp (mt);
14        for (int i = 0; i < n; ++i) w[i] = urdn (mt);
15        for (int i = 0; i < ft + 1; ++i) avg[i] = sig[i] =
16        0; }
17    double compute (double *x) {
18        for (int i = 0; i < n; ++i) {
19            val[i] = 0; for (int j = 0; j < ft; ++j) val[i] +=
20            wp[i][j] * x[j];
21            val[i] = 1 / (1 + exp (-val[i])); }
22        double res = 0; for (int i = 0; i < n; ++i) res +=
23        val[i] * w[i];
24        // return 1 / (1 + exp (-res));
25        return res; }
26    void desc (double *x, double t, double eta) {
27        double o = compute (x), del = (o - t); // * o * (1 -
28        o)
29        for (int i = 0; i < n; ++i) for (int j = 0; j < ft;
30        ++j)
31            wp[i][j] -= eta * del * val[i] * (1 - val[i]) * w[i]
32            * x[j];
33        // for (int i = 0; i < ft; ++i) w[i] -= eta * del * o
34        // * (1 - o) * val[i];
35        for (int i = 0; i < n; ++i) w[i] -= eta * del * val[i]
36        ; }
37    void train (double data[MAXDATA][ft + 1], int dn, int
38    epoch, double eta) {
39        for (int i = 0; i < ft + 1; ++i) for (int j = 0; j <
40        dn; ++j) avg[i] += data[j][i];
41        for (int i = 0; i < ft + 1; ++i) avg[i] /= dn;
42        for (int i = 0; i < ft + 1; ++i) for (int j = 0; j <
43        dn; ++j)
44            sig[i] += (data[j][i] - avg[i]) * (data[j][i] - avg
45            [i]);
46        for (int i = 0; i < ft + 1; ++i) sig[i] = sqrt (sig[i]
47        / dn);
48        for (int i = 0; i < ft + 1; ++i) for (int j = 0; j <
49        dn; ++j)
50            data[j][i] = (data[j][i] - avg[i]) / sig[i];
51        for (int cnt = 0; cnt < epoch; ++cnt) for (int test
52        = 0; test < dn; ++test) {
53            desc (data[test], data[test][ft], eta); }
54        double predict (double *x) {
55            for (int i = 0; i < ft; ++i) x[i] = (x[i] - avg[i])
56            / sig[i];
57            return compute (x) * sig[ft] + avg[ft]; }
58        std::string to_string () {
59            std::ostringstream os; os.precision (20); os << std
60            ::fixed;
61            for (int i = 0; i < n; ++i) for (int j = 0; j < ft;
62            ++j) os << wp[i][j] << " ";
63            for (int i = 0; i < n; ++i) os << w[i] << " ";
64            for (int i = 0; i < ft + 1; ++i) os << avg[i] << " "
65            ;
66            for (int i = 0; i < ft + 1; ++i) os << sig[i] << " "
67            ;
68            return os.str (); }
69        void read (const std::string &str) {
70            std::istringstream is (str);
71            for (int i = 0; i < n; ++i) for (int j = 0; j < ft;
72            ++j) is >> wp[i][j];
73            for (int i = 0; i < n; ++i) is >> w[i];
74            for (int i = 0; i < ft + 1; ++i) is >> avg[i];
75            for (int i = 0; i < ft + 1; ++i) is >> sig[i]; } };

```

4 Number theory

4.1 Fast power module

```

1 /* Fast power module :  $x^n$  */
2 int fpm (int x, int n, int mod) {

```

```

3     int ans = 1, mul = x; while (n) {
4         if (n & 1) ans = int (1ll * ans * mul % mod);
5         mul = int (1ll * mul * mul % mod); n >>= 1; }
6     return ans; }
7 long long mul_mod (long long x, long long y, long long
8     mod) {
9     long long t = (x * y - (long long) ((long double) x /
10     mod * y + 1E-3) * mod) % mod;
11     return t < 0 ? t + mod : t; }
12 long long llfpm (long long x, long long n, long long
13     mod) {
14     long long ans = 1, mul = x; while (n) {
15         if (n & 1) ans = mul_mod (ans, mul, mod);
16         mul = mul_mod (mul, mul, mod); n >>= 1; }
17     return ans; }

```

4.2 Euclidean algorithm

```

1 /* Euclidean algorithm : solves for  $ax + by = \gcd(a, b)$ . */
2 void euclid (const long long &a, const long long &b,
3     long long &x, long long &y) {
4     if (b == 0) x = 1, y = 0;
5     else euclid (b, a % b, y, x), y -= a / b * x; }
6 long long inverse (long long x, long long m) {
7     long long a, b; euclid (x, m, a, b); return (a % m +
8     m) % m; }

```

4.3 Discrete Fourier transform

```

1 /* Discrete Fourier transform : the naffarious you-know
2     -what thing.
3 Usage : call init for the suggested array size, and
4     solve for the transform. (use f!=0 for the inverse)
5 */
6 template <int MAXN = 1000000>
7 struct dft {
8     typedef std::complex <double> complex;
9     complex e[2][MAXN];
10    int init (int n) {
11        int len = 1;
12        for (; len <= 2 * n; len <= 1);
13        for (int i = 0; i < len; ++i) {
14            e[0][i] = complex (cos (2 * PI * i / len), sin (2
15            * PI * i / len));
16            e[1][i] = complex (cos (2 * PI * i / len), -sin (2
17            * PI * i / len)); }
18        return len; }
19    void solve (complex *a, int n, int f) {
20        for (int i = 0, j = 0; i < n; ++i) {
21            if (i > j) std::swap (a[i], a[j]);
22            for (int t = n >> 1; (j ^= t) < t; t >>= 1); }
23        for (int i = 2; i <= n; i <= 1)
24            for (int j = 0; j < n; j += i)
25                for (int k = 0; k < (i >> 1); ++k) {
26                    complex A = a[j + k];
27                    complex B = e[f][n / i * k] * a[j + k + (i >> 1)
28                    ];
29                    a[j + k] = A + B;
30                    a[j + k + (i >> 1)] = A - B; }
31        if (f == 1) {
32            for (int i = 0; i < n; ++i) a[i] = complex (a[i].
33            real () / n, a[i].imag ()); } } };

```

4.4 Fast Walsh-Hadamard transform

```

1 /* Fast Walsh-Hadamard transform : binary operation
2     transform. */
3 void fwt (int *a, int n, int w) {
4     for (int i = 1; i < n; i <= 1)
5         for (int j = 0; j < n; j += i <= 1) {
6             for (int k = 0; k < i; ++k) {
7                 int x = a[j + k], y = a[j + k + i];
8                 if (w) {
9                     /* xor :  $a[j + k] = (x + y) / 2$ ,  $a[j + k + i] = (x - y) / 2$ , and :  $a[j + k] = x - y$ , or :  $a[j + k] = x + y$ ; */
10                    } else {
11                        /* xor :  $a[j + k] = x + y$ ,  $a[j + k + i] = x - y$ , and :  $a[j + k] = x + y$ , or :  $a[j + k] = x - y$ ; */
12                    }
13            }
14        }
15 }

```

4.5 Number theoretic transform

```

1 /* Number theoretic transform : NTT for any module.
2 Usage : Perform NTT on 3 modules and call crt () to
3     merge the result. */
4 template <int MAXN = 1000000>
5 struct ntt {
6     int MOD[3] = {1045430273, 1051721729, 1053818881},
7     PRT[3] = {3, 6, 7};
8     void solve (int *a, int n, int f, int mod, int prt) {
9         for (int i = 0, j = 0; i < n; ++i) {
10            if (i > j) std::swap (a[i], a[j]);
11            for (int t = n >> 1; (j ^= t) < t; t >>= 1); }
12        for (int i = 2; i <= n; i <= 1) {
13            static int exp[MAXN]; exp[0] = 1;
14            exp[1] = fpm (prt, (mod - 1) / i, mod);
15            if (f == 1) exp[1] = fpm (exp[1], mod - 2, mod);
16            for (int k = 2; k < (i >> 1); ++k) {
17                exp[k] = int (1ll * exp[k - 1] * exp[1] % mod); }
18            for (int j = 0; j < n; j += i) {
19                for (int k = 0; k < (i >> 1); ++k) {
20                    int &a = a[j + k], &b = a[j + k + (i >> 1)];
21                    int A = pa, B = int (1ll * pb * exp[k] % mod);
22                    pa = (A + B) % mod;
23                    pb = (A - B + mod) % mod; } } }
24        if (f == 1) {
25            int rev = fpm (n, mod - 2, mod);

```



```

24   for (int i = 0; i < n; ++i) a[i] = int (1LL * a[i]
      * rev % mod); } }
25   int crt (int *a, int mod) {
26   static int inv[3][3];
27   for (int i = 0; i < 3; ++i) for (int j = 0; j < 3;
      ++j)
28   inv[i][j] = (int) inverse (MOD[i], MOD[j]);
29   static int x[3];
30   for (int i = 0; i < 3; ++i) { x[i] = a[i];
31   for (int j = 0; j < i; ++j) {
32   int t = (x[i] - x[j] + MOD[i]) % MOD[i];
33   if (t < 0) t += MOD[i];
34   x[i] = int (1LL * t * inv[j][i] % MOD[i]); } }
35   int sum = 1, ret = x[0] % mod;
36   for (int i = 1; i < 3; ++i) {
37   sum = int (1LL * sum * MOD[i - 1] % mod);
38   ret += int (1LL * x[i] * sum % mod);
39   if (ret >= mod) ret -= mod; }
40   return ret; } };

```

4.6 Chinese remainder theorem

```

1  /* Chinese remainder theorem : finds positive integers
   x = out.first + k * out.second that satisfies x %
   in[i].second = in[i].first. */
2  struct crt {
3  long long fix (const long long &a, const long long &b
   ) { return (a % b + b) % b; }
4  bool solve (const std::vector<std::pair<long long,
   long long>> &in, std::pair<long long, long long>
   &out) {
5  out = std::make_pair (1LL, 1LL);
6  for (int i = 0; i < (int) in.size (); ++i) {
7  long long n, u;
8  euclid (out.second, in[i].second, n, u);
9  long long divisor = std::gcd (out.second, in[i].
   second);
10 if ((in[i].first - out.first) % divisor) return
   false;
11 n *= (in[i].first - out.first) / divisor;
12 n = fix (n, in[i].second);
13 out.first += out.second * n;
14 out.second *= in[i].second / divisor;
15 out.first = fix (out.first, out.second); }
16 return true; } };

```

4.7 Linear Recurrence

```

1  /* Linear recurrence : finds the n-th element of a
   linear recurrence.
2  Usage : vector<int> - first n terms, vector<int> -
   transition function, calc (k) : the kth term mod
   MOD.
3  Example : In : {2, 1}, {2, 1} :
   a1 = 2, a2 = 1, an = 2an-1 + an-2, Out : calc (3) = 5,
   calc (10007) = 959155122 (MOD 1E9+7) */
4  struct linear_rec {
5  const int LOG = 30, MOD = 1E9 + 7; int n;
6  std::vector<int> first, trans;
7  std::vector<std::vector<int>> bin;
8  std::vector<int> add (std::vector<int> &a, std:::
   vector<int> &b) {
9  std::vector<int> result(n * 2 + 1, 0);
10 for (int i = 0; i <= n; ++i) for (int j = 0; j <= n;
   ++j)
11 if ((result[i + j] += 1LL * a[i] * b[j] % MOD) >=
   MOD) result[i + j] -= MOD;
12 for (int i = 2 * n; i > n; --i) {
13 for (int j = 0; j < n; ++j)
14 if ((result[i - 1 - j] += 1LL * result[i] * trans[
   j] % MOD) >= MOD) result[i - 1 - j] -= MOD;
15 result[i] = 0; }
16 result.erase(result.begin() + n + 1, result.end());
17 return result; }
18 linear_rec (const std::vector<int> &first, const std
   ::vector<int> &trans) : first(first), trans(
   trans) {
19 n = first.size(); std::vector<int> a(n + 1, 0); a
   [1] = 1; bin.push_back(a);
20 for (int i = 1; i < LOG; ++i) bin.push_back(add(bin
   [i - 1], bin[i - 1])); }
21 int solve (int k) {
22 std::vector<int> a(n + 1, 0); a[0] = 1;
23 for (int i = 0; i < LOG; ++i) if (k >> i & 1) a =
   add(a, bin[i]);
24 int ret = 0;
25 for (int i = 0; i < n; ++i) if ((ret += (long long)
   a[i + 1] * first[i] % MOD) >= MOD) ret -= MOD;
26 return ret; } };

```

4.8 Berlekamp Massey algorithm

```

1  /* Berlekamp Massey algorithm : Complexity: O(n^2)
   Requirement: const MOD, inverse(int)
2  Input: the first elements of the sequence
3  Output: the recursive equation of the given sequence
4  Example In: {1, 1, 2, 3}
5  Example Out: {1, 1000000006, 1000000006} (MOD = 1e9+7)
   */
6  struct berlekamp-massey {
7  struct Poly { std::vector<int> a; Poly() { a.clear()
   ; }
8  Poly (std::vector<int> &a) : a (a) {}
9  int length () const { return a.size(); }
10 Poly move (int d) { std::vector<int> na (d, 0);
   na.insert (na.end (), a.begin (), a.end ());
11 return Poly (na); }
12 int calc(std::vector<int> &d, int pos) { int ret =
   0;
13 for (int i = 0; i < (int) a.size (); ++i) {

```

```

15 if ((ret += 1LL * d[pos - i] * a[i] % MOD) >= MOD)
   {
16 ret -= MOD; } }
17 return ret; } }
18 Poly operator - (const Poly &b) {
19 std::vector<int> na (std::max (this -> length (),
   b.length ());
20 for (int i = 0; i < (int) na.size (); ++i) {
21 int aa = i < this -> length () ? this -> a[i] : 0;
22 bb = i < b.length () ? b.a[i] : 0;
23 na[i] = (aa + MOD - bb) % MOD; }
24 return Poly (na); } };
25 Poly operator * (const int &c, const Poly &p) {
26 std::vector<int> na (p.length ());
27 for (int i = 0; i < (int) na.size (); ++i) {
28 na[i] = 1LL * c * p.a[i] % MOD; }
29 return na; }
30 std::vector<int> solve(vector<int> a) {
31 int n = a.size (); Poly s, b;
32 s.a.push_back (1), b.a.push_back (1);
33 for (int i = 0, j = -1, ld = 1; i < n; ++i) {
34 int d = s.calc(a, i); if (d) {
35 if ((s.length () - 1) * 2 <= i) {
36 Poly ob = b; b = s;
37 s = s - 1LL * d * inverse (ld) % MOD * ob.move (i
   - j);
38 j = i; ld = d;
39 } else {
40 s = s - 1LL * d * inverse (ld) % MOD * b.move (i
   - j); } } }
41 return s.a; } };

```

4.9 Baby step giant step algorithm

```

1  /* Baby step giant step algorithm : Solves  $a^x = b \pmod c$ 
   in  $O(\sqrt{c})$ . */
2  struct bsgs {
3  int solve (int a, int b, int c) {
4  std::unordered_map<int, int> bs;
5  int m = (int) sqrt ((double) c) + 1, res = 1;
6  for (int i = 0; i < m; ++i) {
7  if (bs.find (res) == bs.end ()) bs[res] = i;
8  res = int (1LL * res * a % c); }
9  int mul = 1, inv = (int) inverse (a, c);
10 for (int i = 0; i < m; ++i) mul = int (1LL * mul *
   inv % c);
11 res = b % c;
12 for (int i = 0; i < m; ++i) {
13 if (bs.find (res) != bs.end ()) return i * m + bs[
   res];
14 res = int (1LL * res * mul % c); }
15 return -1; } };

```

4.10 Miller Rabin primality test

```

1  /* Miller Rabin : tests whether a certain integer is
   prime. */
2  struct miller_rabin {
3  int BASE[12] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29,
   31, 37};
4  bool check (const long long &p, const long long &b) {
5  long long n = p - 1;
6  for (; ~n & 1; n >= 1);
7  long long res = llfpm (b, n, p);
8  for (; n != p - 1 && res != 1 && res != p - 1; n <=
   1)
9  res = mul_mod (res, res, p);
10 return res == p - 1 || (n & 1) == 1; }
11 bool solve (const long long &n) {
12 if (n < 2) return false;
13 if (n < 4) return true;
14 if (~n & 1) return false;
15 for (int i = 0; i < 12 && BASE[i] < n; ++i) if (!
   check (n, BASE[i])) return false;
16 return true; } };

```

4.11 Pollard's Rho algorithm

```

1  /* Pollard Rho : factorizes an integer. */
2  struct pollard_rho {
3  miller_rabin is_prime;
4  const long long thr = 13E9;
5  long long facize (const long long &n, const long long
   &seed) {
6  long long x = rand () % (n - 1) + 1, y = x;
7  for (int head = 1, tail = 2; ; ) {
8  x = mul_mod (x, x, n);
9  x = (x + seed) % n;
10 if (x == y) return n;
11 long long ans = std::gcd (std::abs (x - y), n);
12 if (ans > 1 && ans < n) return ans;
13 if (++head == tail) { y = x; tail <= 1; } } }
14 void search (const long long &n, std::vector<long
   long> &div) {
15 if (n > 1) {
16 if (is_prime.solve (n)) div.push_back (n);
17 else {
18 long long fac = n;
19 for (; fac >= n; fac = facize (n, rand () % (n -
   1) + 1));
20 search (n / fac, div); search (fac, div); } } }
21 std::vector<long long> solve (const long long &n) {
22 std::vector<long long> ans;
23 if (n > thr) search (n, ans);
24 else {
25 long long rem = n;
26 for (long long i = 2; i * i <= rem; ++i)
27 while (! (rem % i)) { ans.push_back (i); rem /= i;
   }
28 if (rem > 1) ans.push_back (rem); }
29 return ans; } };

```

5 Geometry

```
1 #define cd const double &
2 const double EPS = 1E-8, PI = acos (-1);
3 int sgn (cd x) { return x < -EPS ? -1 : x > EPS; }
4 int cmp (cd x, cd y) { return sgn (x - y); }
5 double sqr (cd x) { return x * x; }
```

5.1 Point

```
1 #define cp const point &
2 struct point {
3     double x, y;
4     explicit point (cd x = 0, cd y = 0) : x (x), y (y) {}
5     int dim () const { return sgn (y) == 0 ? sgn (x) < 0
6         : sgn (y) < 0; }
7     point unit () const { double l = sqrt (x * x + y * y)
8         ; return point (x / l, y / l); }
9     //counter-clockwise
10    point rot90 () const { return point (-y, x); }
11    //clockwise
12    point _rot90 () const { return point (y, -x); }
13    point rot (cd t) const {
14        double c = cos (t), s = sin (t);
15        return point (x * c - y * s, x * s + y * c); }
16    bool operator == (cp a, cp b) { return cmp (a.x, b.x)
17        == 0 && cmp (a.y, b.y) == 0; }
18    bool operator != (cp a, cp b) { return cmp (a.x, b.x)
19        != 0 || cmp (a.y, b.y) != 0; }
20    bool operator < (cp a, cp b) { return (cmp (a.x, b.x)
21        == 0) ? cmp (a.y, b.y) < 0 : cmp (a.x, b.x) < 0; }
22    point operator - (cp a) { return point (-a.x, -a.y); }
23    point operator + (cp a, cp b) { return point (a.x + b.x,
24        a.y + b.y); }
25    point operator - (cp a, cp b) { return point (a.x - b.x,
26        a.y - b.y); }
27    point operator * (cp a, cd b) { return point (a.x * b,
28        a.y * b); }
29    point operator / (cp a, cd b) { return point (a.x / b,
30        a.y / b); }
31    double dot (cp a, cp b) { return a.x * b.x + a.y * b.y; }
32    double det (cp a, cp b) { return a.x * b.y - a.y * b.x; }
33    double dis2 (cp a, cp b = point ()) { return sqr (a.x
34        - b.x) + sqr (a.y - b.y); }
35    double dis (cp a, cp b = point ()) { return sqrt (dis2
36        (a, b)); }
```

5.2 Line

```
1 #define cl const line &
2 struct line {
3     point s, t;
4     explicit line (cp s = point (), cp t = point ()) : s
5         (s), t (t) {}
6     bool point_on_segment (cp a, cl b) { return sgn (det (
7         a - b.s, b.t - b.s)) == 0 && sgn (dot (b.s - a, b.
8         t - a)) <= 0; }
9     bool two_side (cp a, cp b, cl c) { return sgn (det (a
10        - c.s, c.t - c.s)) * sgn (det (b - c.s, c.t - c.s))
11        < 0; }
12    bool intersect_judgment (cl a, cl b) {
13        if (point_on_segment (b.s, a) || point_on_segment (b.
14        t, a)) return true;
15        if (point_on_segment (a.s, b) || point_on_segment (a.
16        t, b)) return true;
17        return two_side (a.s, a.t, b) && two_side (b.s, b.t,
18        a); }
19    point line_intersect (cl a, cl b) {
20        double s1 = det (a.t - a.s, b.s - a.s), s2 = det (a.t
21        - a.s, b.t - a.s);
22        return (b.s * s2 - b.t * s1) / (s2 - s1); }
23    double point_to_line (cp a, cl b) { return fabs (det (
24        b.t - b.s, a - b.s)) / dis (b.s, b.t); }
25    point project_to_line (cp a, cl b) { return b.s + (b.t
26        - b.s) * (dot (a - b.s, b.t - b.s) / dis2 (b.t, b.
27        s)); }
28    double point_to_segment (cp a, cl b) {
29        if (sgn (dot (b.s - a, b.t - b.s)) > dot (b.t - a, b.t
30        - b.s)) <= 0 return fabs (det (b.t - b.s, a - b.
31        s)) / dis (b.s, b.t);
32        return std::min (dis (a, b.s), dis (a, b.t)); }
33    bool in_polygon (cp p, const std::vector<point> & po) {
34        int n = (int) po.size (), counter = 0;
35        for (int i = 0; i < n; ++i) {
36            point a = po[i], b = po[(i + 1) % n];
37            //Modify the next line if necessary.
38            if (point_on_segment (p, line (a, b))) return true;
39            int x = sgn (det (p - a, b - a)), y = sgn (a.y - p.y
40            ), z = sgn (b.y - p.y);
41            if (x > 0 && y <= 0 && z > 0) counter++;
42            if (x < 0 && z <= 0 && y > 0) counter--; }
43        return counter != 0; }
44    double polygon_area (const std::vector<point> &a) {
45        double ans = 0.0;
46        for (int i = 0; i < (int) a.size (); ++i) ans += det
47            (a[i], a[(i + 1) % a.size ()]) / 2.0;
48        return ans; }
```

5.3 Circle

```
1 #define cc const circle &
2 struct circle {
3     point c; double r;
4     explicit circle (point c = point (), double r = 0) :
5         c (c), r (r) {}
6     bool operator == (cc a, cc b) { return a.c == b.c &&
7         cmp (a.r, b.r) == 0; }
```

```
6 bool operator != (cc a, cc b) { return !(a == b); }
7 bool in_circle (cp a, cc b) { return cmp (dis (a, b.c)
8     , b.r) <= 0; }
9 circle make_circle (cp a, cp b) { return circle ((a +
10    b) / 2, dis (a, b) / 2); }
11 circle make_circle (cp a, cp b, cp c) { point p =
12    circumcenter (a, b, c); return circle (p, dis (p,
13    a)); }
14 //In the order of the line vector.
15 std::vector<point> line_circle_intersect (cl a, cc b)
16 {
17     if (cmp (point_to_line (b.c, a), b.r) > 0) return std
18         ::vector<point> ();
19     double x = sqrt (sqr (b.r) - sqr (point_to_line (b.c,
20     a)));
21     return std::vector<point> ({project_to_line (b.c, a)
22     + (a.s - a.t).unit () * x, project_to_line (b.c,
23     a) - (a.s - a.t).unit () * x}); }
24 double circle_intersect_area (cc a, cc b) {
25     double d = dis (a.c, b.c);
26     if (sgn (d - (a.r + b.r)) >= 0) return 0;
27     if (sgn (d - abs(a.r - b.r)) <= 0) {
28         double r = std::min (a.r, b.r); return r * r * PI; }
29     double x = (d * d + a.r * a.r - b.r * b.r) / (2 * d),
30         t1 = acos (min (1., max (-1., x / a.r))), t2 =
31         acos (min (1., max (-1., (d - x) / b.r)));
32     return a.r * a.r * t1 + b.r * b.r * t2 - d * a.r *
33         sin (t1); }
34 //Counter-clockwise with respect of vector  $O_a O_b$ .
35 std::vector<point> circle_intersect (cc a, cc b) {
36     if (a.c == b.c || cmp (dis (a.c, b.c), a.r + b.r) > 0
37     || cmp (dis (a.c, b.c), std::abs (a.r - b.r)) <
38     0) return std::vector<point> ();
39     point r = (b.c - a.c).unit ();
40     double d = dis (a.c, b.c);
41     double h = ((sqr (a.r) - sqr (b.r)) / d + d) / 2;
42     double x = sqrt (sqr (a.r) - sqr (h));
43     if (sgn (h) == 0) return std::vector<point> ({a.c +
44     r * x});
45     return std::vector<point> ({a.c + r * x - r.rot90 ()
46     * h, a.c + r * x + r.rot90 () * h}); }
47 //Counter-clockwise with respect of point  $a$ .
48 std::vector<point> tangent (cp a, cc b) { circle p =
49     make_circle (a, b.c); return circle_intersect (p,
50     b); }
51 //Counter-clockwise with respect of point  $O_a$ .
52 std::vector<line> extangent (cc a, cc b) {
53     std::vector<line> ret;
54     if (cmp (dis (a.c, b.c), std::abs (a.r - b.r)) <= 0)
55         return ret;
56     if (sgn (a.r - b.r) == 0) {
57         point dir = b.c - a.c; dir = (dir * a.r / dis (dir))
58         .rot90 ();
59         ret.push_back (line (a.c - dir, b.c - dir));
60         ret.push_back (line (a.c + dir, b.c + dir)); }
61     else {
62         point p = (b.c * a.r - a.c * b.r) / (a.r - b.r);
63         std::vector pp = tangent (p, a), qq = tangent (p, b);
64         if (pp.size () == 2 && qq.size () == 2) {
65             if (cmp (a.r, b.r) < 0) std::swap (pp[0], pp[1]),
66                 std::swap (qq[0], qq[1]);
67             ret.push_back (line (pp[0], qq[0]));
68             ret.push_back (line (pp[1], qq[1])); } }
69     return ret; }
70 //Counter-clockwise with respect of point  $O_a$ .
71 std::vector<line> intangent (cc cl, cc c2) {
72     point p = (b.c * a.r + a.c * b.r) / (a.r + b.r);
73     std::vector pp = tangent (p, a), qq = tangent (p, b);
74     if (pp.size () == 2 && qq.size () == 2) {
75         ret.push_back (line (pp[0], qq[0]));
76         ret.push_back (line (pp[1], qq[1])); }
77     return ret; }
```

5.4 Centers of a triangle

```
1 point incenter (cp a, cp b, cp c) {
2     double p = dis (a, b) + dis (b, c) + dis (c, a);
3     return (a * dis (b, c) + b * dis (c, a) + c * dis (a,
4     b)) / p; }
5 point circumcenter (cp a, cp b, cp c) {
6     point p = b - a, q = c - a, s (dot (p, p) / 2, dot (q
7     , q) / 2);
8     return a + point (det (s, point (p.y, q.y)), det (
9     point (p.x, q.x), s) / det (p, q)); }
10 point orthocenter (cp a, cp b, cp c) { return a + b +
11     c - circumcenter (a, b, c) * 2; }
```

5.5 Fermat point

```
1 /* Fermat point : finds a point  $P$  that minimizes
2     $|PA| + |PB| + |PC|$ . */
3 point fermat_point (cp a, cp b, cp c) {
4     if (a == b) return a; if (b == c) return b; if (c ==
5     a) return c;
6     double ab = dis (a, b), bc = dis (b, c), ca = dis (c,
7     a);
8     double cosa = dot (b - a, c - a) / ab / ca;
9     double cosb = dot (a - b, c - b) / ab / bc;
10    double cosc = dot (b - c, a - c) / ca / bc;
11    double sq3 = PI / 3.0; point mid;
12    if (sgn (cosa + 0.5) < 0) mid = a;
13    else if (sgn (cosb + 0.5) < 0) mid = b;
14    else if (sgn (cosc + 0.5) < 0) mid = c;
15    else if (sgn (det (b - a, c - a)) < 0) mid =
16        line_intersect (line (a, b + (c - b).rot (sq3)),
17        line (b, c + (a - c).rot (sq3)));
18    else mid = line_intersect (line (a, c + (b - c).rot (
19        sq3)), line (c, b + (a - b).rot (sq3)));
20    return mid; }
```


5.6 Convex hull

```

1 //Counter-clockwise, with minimum number of points.
2 bool turn_left (cp a, cp b, cp c) { return sgn (det (b
  - a, c - a)) >= 0; }
3 std::vector<point> convex_hull (std::vector<point> a
  ) {
4   int cnt = 0; std::sort (a.begin (), a.end ());
5   std::vector<point> ret (a.size () <= 1, point ());
6   for (int i = 0; i < (int) a.size (); ++i) {
7     while (cnt > 1 && turn_left (ret[cnt - 2], a[i], ret
      [cnt - 1])) --cnt;
8     ret[cnt++] = a[i];
9   }
10  int fixed = cnt;
11  for (int i = (int) a.size () - 1; i >= 0; --i) {
12    while (cnt > fixed && turn_left (ret[cnt - 2], a[i],
      ret[cnt - 1])) --cnt;
13    ret[cnt++] = a[i];
14  }
15  return std::vector<point> (ret.begin (), ret.begin
    () + cnt - 1);
  }

```

5.7 Half plane intersection

```

1 /* Online half plane intersection : complexity  $O(n)$ 
   each operation. */
2 std::vector<point> cut (const std::vector<point> &c,
  line p) {
3   std::vector<point> ret;
4   if (c.empty ()) return ret;
5   for (int i = 0; i < (int) c.size (); ++i) {
6     int j = (i + 1) % (int) c.size ();
7     if (turn_left (p.s, p.t, c[i])) ret.push_back (c[i])
      ;
8     if (two_side (c[i], c[j], p)) ret.push_back (
      line_intersect (p, line (c[i], c[j])));
9   }
10  return ret;
11 }
12 /* Offline half plane intersection : complexity
    $O(n \log n)$ . */
13 bool turn_left (cl l, cp p) { return turn_left (l.s, l
  .t, p); }
14 int cmp (cp a, cp b) { return a.dim () != b.dim () ? (
  a.dim () < b.dim () ? -1 : 1) : -sgn (det (a, b));
15 }
16 std::vector<point> half_plane_intersect (std::vector
  <line> h) {
17   typedef std::pair<point, line> polar;
18   std::vector<polar> g; g.resize (h.size ());
19   for (int i = 0; i < (int) h.size (); ++i) g[i] = std
    ::make_pair (h[i].t - h[i].s, h[i]);
20   sort (g.begin (), g.end (), [&] (const polar &a,
    const polar &b) {
21     if (cmp (a.first, b.first) == 0) return sgn (det (a.
      second.t - a.second.s, b.second.t - a.second.s))
      < 0;
22     else return cmp (a.first, b.first) < 0; });
23   h.resize (std::unique (g.begin (), g.end (), [&] (
    const polar &a, const polar &b) { return cmp (a.
      first, b.first) == 0; }) - g.begin ());
24   for (int i = 0; i < (int) h.size (); ++i) h[i] = g[i
    ].second;
25   int fore = 0, rear = -1; std::vector<line> ret (h.
    size (), line ());
26   for (int i = 0; i < (int) h.size (); ++i) {
27     while (fore < rear && !turn_left (h[i],
      line_intersect (ret[rear - 1], ret[rear]))) --
      rear;
28     while (fore < rear && !turn_left (h[i],
      line_intersect (ret[fore], ret[fore + 1]))) ++
      fore;
29     ret[++rear] = h[i];
30   }
31   while (rear - fore > 1 && !turn_left (ret[fore],
    line_intersect (ret[rear - 1], ret[rear]))) --
    rear;
32   while (rear - fore > 1 && !turn_left (ret[rear],
    line_intersect (ret[fore], ret[fore + 1]))) ++
    fore;
33   if (rear - fore < 2) return std::vector<point> ();
34   std::vector<point> ans; ans.resize (rear + 1);
35   for (int i = 0; i < rear + 1; ++i) ans[i] =
    line_intersect (ret[i], ret[(i + 1) % (rear + 1)
    ]);
36   return ans;
  }

```

5.8 Nearest pair of points

```

1 /* Nearest pair of points : [l, r), need to sort p
   first. */
2 double solve (std::vector<point> &p, int l, int r) {
3   if (l + 1 >= r) return INF;
4   int m = (l + r) / 2; double mx = p[m].x; std::vector
    <point> v;
5   double ret = std::min (solve(p, l, m), solve(p, m, r)
    );
6   for (int i = l; i < r; ++i)
7     if (sqr (p[i].x - mx) < ret) v.push_back (p[i]);
8   sort (v.begin (), v.end (), [&] (cp a, cp b) { return
    a.y < b.y; });
9   for (int i = 0; i < v.size (); ++i)
10    for (int j = i + 1; j < v.size (); ++j) {
11      if (sqr (v[i].y - v[j].y) > ret) break;
12      ret = min (ret, dis2 (v[i] - v[j]));
13    }
14   return ret;
  }

```

5.9 Minimum circle

```

1 circle minimum_circle (std::vector<point> p) {
2   circle ret; std::random_shuffle (p.begin (), p.end ()
    );
3   for (int i = 0; i < (int) p.size (); ++i) if (!
    in_circle (p[i], ret)) {

```

```

4     ret = circle (p[i], 0); for (int j = 0; j < i; ++j)
      if (!in_circle (p[j], ret)) {
5       ret = make_circle (p[j], p[i]); for (int k = 0; k <
        j; ++k)
6         if (!in_circle (p[k], ret)) ret = make_circle (p[i
          ], p[j], p[k]);
7     }
    return ret;
  }

```

5.10 Intersection of a polygon and a circle

```

1 struct polygon_circle_intersect {
2   double sector_area (cp a, cp b, const double &r) {
3     double c = (2.0 * r * r - dis2 (a, b)) / (2.0 * r *
      r);
4     return r * r * acos (c) / 2.0;
5   }
6   double area (cp a, cp b, const double &r) {
7     double dA = dot (a, a), dB = dot (b, b), dC =
      point_to_segment (point (), line (a, b));
8     if (sgn (dA - r * r) <= 0 && sgn (dB - r * r) <= 0)
9       return det (a, b) / 2.0;
10    point tA = a.unit () * r, tB = b.unit () * r;
11    if (sgn (dC - r) > 0) return sector_area (tA, tB, r)
      ;
12    std::vector<point> ret = line_circle_intersect (
      line (a, b), circle (point (), r));
13    if (sgn (dA - r * r) > 0 && sgn (dB - r * r) > 0)
14      return sector_area (tA, ret[0], r) + det (ret[0],
      ret[1]) / 2.0 + sector_area (ret[1], tB, r);
15    if (sgn (dA - r * r) > 0) return det (ret[0], b) /
      2.0 + sector_area (tA, ret[0], r);
16    else return det (a, ret[1]) / 2.0 + sector_area (ret
      [1], tB, r);
17  }
18  double solve (const std::vector<point> &p, cc c) {
19    double ret = 0.0;
20    for (int i = 0; i < (int) p.size (); ++i) {
21      int s = sgn (det (p[i] - c.c, p[(i + 1) % p.size ()
        ] - c.c));
22      if (s > 0) ret += area (p[i] - c.c, p[(i + 1) % p.
        size ()] - c.c, c.r);
23      else ret -= area (p[(i + 1) % p.size ()] - c.c, p[i
        ] - c.c, c.r);
24    }
25    return std::abs (ret);
  }

```

5.11 Union of circles

```

1 template<int MAXN = 500> struct union_circle {
2   int C; circle c[MAXN]; double area[MAXN];
3   struct event {
4     point p; double ang; int delta;
5     event (cp p = point (), double ang = 0, int delta =
      0) : p(p), ang(ang), delta(delta) {}
6     bool operator < (const event &a) { return ang < a.
      ang; }
7   };
8   void addevent (cc a, cc b, std::vector<event> &evt,
    int &cnt) {
9     double d2 = dis2 (a.c, b.c), d_ratio = ((a.r - b.r)
      * (a.r + b.r) / d2 + 1) / 2;
10    p_ratio = sqrt (std::max (0., -(d2 - sqr(a.r - b.r)
      ) * (d2 - sqr(a.r + b.r)) / (d2 * d2 * 4)));
11    point d = b.c - a.c, p = d.rot (PI / 2), q0 = a.c + d
      * d_ratio + p * p_ratio, q1 = a.c + d * d_ratio
      - p * p_ratio;
12    double ang0 = atan2 ((q0 - a.c).y, (q0 - a.c).x),
      ang1 = atan2 ((q1 - a.c).y, (q1 - a.c).x);
13    evt.emplace_back (q1, ang1, 1); evt.emplace_back (q0,
      ang0, -1); cnt += ang1 > ang0;
14    bool same (cc a, cc b) { return sgn (dis (a.c, b.c))
      == 0 && sgn (a.r - b.r) == 0; }
15    bool overlap (cc a, cc b) { return sgn (a.r - b.r -
      dis (a.c, b.c)) >= 0; }
16    bool intersect (cc a, cc b) { return sgn (dis (a.c, b.
      c) - a.r - b.r) < 0; }
17    void solve () {
18      std::fill (area, area + C + 2, 0);
19      for (int i = 0; i < C; ++i) {
20        int cnt = 1; std::vector<event> evt;
21        for (int j = 0; j < i; ++j) if (same (c[i], c[j]))
22          ++cnt;
23        for (int j = 0; j < C; ++j) if (j != i && !same (c
          [i], c[j]) && overlap (c[j], c[i])) ++cnt;
24        for (int j = 0; j < C; ++j) if (j != i && !overlap
          (c[j], c[i]) && !intersect (c[i], c[j]) &&
          intersect (c[i], c[j]))
25          addevent (c[i], c[j], evt, cnt);
26        if (evt.empty ()) area[cnt] += PI * c[i].r * c[i].r
          ;
27        else {
28          std::sort (evt.begin (), evt.end ());
29          evt.push_back (evt.front ());
30          for (int j = 0; j + 1 < (int) evt.size (); ++j) {
31            cnt += evt[j].delta; area[cnt] += det (evt[j].p,
              evt[j + 1].p) / 2;
32            double ang = evt[j + 1].ang - evt[j].ang; if (ang
              < 0) ang += PI * 2;
33            area[cnt] += ang * c[i].r * c[i].r / 2 - sin(ang)
              * c[i].r * c[i].r / 2;
          }
        }
      }
    }
  }

```

5.12 3D point

```

1 #define cp3 const point3 &
2 struct point3 {
3   double x, y, z;
4   explicit point3 (cd x = 0, cd y = 0, cd z = 0) : x (x
    ), y (y), z (z) {}
5   point3 operator + (cp3 a, cp3 b) { return point3 (a.x
      + b.x, a.y + b.y, a.z + b.z); }
6   point3 operator - (cp3 a, cp3 b) { return point3 (a.x
      - b.x, a.y - b.y, a.z - b.z); }
7   point3 operator * (cp3 a, cd b) { return point3 (a.x *
      b, a.y * b, a.z * b); }
8   point3 operator / (cp3 a, cd b) { return point3 (a.x /
      b, a.y / b, a.z / b); }

```

```

9 double dot (cp3 a, cp3 b) { return a.x * b.x + a.y * b.y + a.z * b.z; }
10 point3 det (cp3 a, cp3 b) { return point3 (a.y * b.z - a.z * b.y, -a.x * b.z + a.z * b.x, a.x * b.y - a.y * b.x); }
11 double dis2 (cp3 a, cp3 b = point3 ()) { return sqr (a.x - b.x) + sqr (a.y - b.y) + sqr (a.z - b.z); }
12 double dis (cp3 a, cp3 b = point3 ()) { return sqrt (dis2 (a, b)); }
13 //right-handed, if x+ -> y+ is right-handed
14 point3 rotate(cp3 p, cp3 axis, double w) {
15     double x = axis.x, y = axis.y, z = axis.z;
16     double s = x * x + y * y + z * z, ss = sqrt(s), cosw = cos(w), sinw = sin(w);
17     double a[4][4]; memset(a, 0, sizeof(a));
18     a[3][3] = 1;
19     a[0][0] = ((y * y + z * z) * cosw + x * x) / s;
20     a[0][1] = x * y * (1 - cosw) / s + z * sinw / ss;
21     a[0][2] = x * z * (1 - cosw) / s - y * sinw / ss;
22     a[1][0] = x * y * (1 - cosw) / s - z * sinw / ss;
23     a[1][1] = ((x * x + z * z) * cosw + y * y) / s;
24     a[1][2] = y * z * (1 - cosw) / s + x * sinw / ss;
25     a[2][0] = x * z * (1 - cosw) / s + y * sinw / ss;
26     a[2][1] = y * z * (1 - cosw) / s - x * sinw / ss;
27     a[2][2] = ((x * x + y * y) * cos(w) + z * z) / s;
28     double ans[4] = {0, 0, 0, 0}, c[4] = {p.x, p.y, p.z, 1};
29     for (int i = 0; i < 4; ++i) for (int j = 0; j < 4; ++j)
30         ans[i] += a[j][i] * c[j];
31     return point3 (ans[0], ans[1], ans[2]);
32 }

```

5.13 3D line

```

1 #define cl3 const line3 &
2 struct line3 {
3     point3 s, t;
4     explicit line3 (cp3 s = point3 (), cp3 t = point3 ()) : s(s), t(t) {}
5     point3 line_plane_intersection (cl3 a, cl3 b) { return a.s + (a.t - a.s) * dot (b.s - a.s, b.t - b.s) / dot (a.t - a.s, b.t - b.s); }
6     line3 plane_intersection (cl3 a, cl3 b) {
7         point3 p = det (a.t - a.s, b.t - b.s), q = det (a.t - a.s, p), s = line_plane_intersection (line3 (a.s, a.s + q), b);
8         return line3 (s, s + p);
9     }
10    point3 project_to_plane (cp3 a, cl3 b) { return a + (b.t - b.s) * dot (b.t - b.s, b.s - a) / dis2 (b.t - b.s); }

```

5.14 3D convex hull

```

1 /* 3D convex hull : initializes n and p / outputs face */
2 template <int MAXN = 500>
3 struct convex_hull3 {
4     double mix (cp3 a, cp3 b, cp3 c) { return dot (det (a, b), c); }
5     double volume (cp3 a, cp3 b, cp3 c, cp3 d) { return mix (b - a, c - a, d - a); }
6     struct tri {
7         int a, b, c;
8         tri() {}
9         tri(int_a, int_b, int_c): a(a), b(b), c(c) {}
10        double area() const { return dis (det (p[b] - p[a], p[c] - p[a])) / 2; }
11        point3 normal() const { return det (p[b] - p[a], p[c] - p[a]).unit (); }
12        double dis (cp3 p0) const { return dot (normal (), p0 - p[a]); }
13    };
14    int n; std::vector <point3> p;
15    std::vector <tri> face; tmp;
16    int mark[MAXN][MAXN], time;
17    void add (int v) {
18        ++time; tmp.clear ();
19        for (int i = 0; i < (int) face.size (); ++i) {
20            int a = face[i].a, b = face[i].b, c = face[i].c;
21            if (sgn (volume (p[v], p[a], p[b], p[c])) > 0)
22                mark[a][b] = mark[b][a] = mark[a][c] = mark[c][a] = mark[b][c] = mark[c][b] = time;
23            else tmp.push_back (face[i]);
24        }
25        face.clear (); face = tmp;
26        for (int i = 0; i < (int) tmp.size (); ++i) {
27            int a = face[i].a, b = face[i].b, c = face[i].c;
28            if (mark[a][b] == time) face.emplace_back (v, b, a);
29            if (mark[b][c] == time) face.emplace_back (v, c, b);
30            if (mark[c][a] == time) face.emplace_back (v, a, c);
31        }
32    }
33    void reorder () {
34        for (int i = 2; i < n; ++i) {
35            point3 tmp = det (p[i] - p[0], p[i] - p[1]);
36            if (sgn (dis (tmp))) {
37                std::swap (p[i], p[2]);
38                for (int j = 3; j < n; ++j)
39                    if (sgn (volume (p[0], p[1], p[2], p[j]))) {
40                        std::swap (p[j], p[3]); return; } } }
41    }
42    void build_convex () {
43        reorder (); face.clear ();
44        face.emplace_back (0, 1, 2);
45        face.emplace_back (0, 2, 1);
46        for (int i = 3; i < n; ++i) add(i); }

```

6 Graph

```

1 template <int MAXN = 100000, int MAXM = 100000>
2 struct edge_list {
3     int size, begin[MAXN], dest[MAXM], next[MAXM];

```

```

4     void clear (int n) { size = 0; std::fill (begin, begin + n, -1); }
5     edge_list (int n = MAXN) { clear (n); }
6     void add_edge (int u, int v) { dest[size] = v; next[size] = begin[u]; begin[u] = size++; }
7     template <int MAXN = 100000, int MAXM = 100000>
8     struct cost_edge_list {
9         int size, begin[MAXN], dest[MAXM], next[MAXM], cost[MAXM];
10        void clear (int n) { size = 0; std::fill (begin, begin + n, -1); }
11        cost_edge_list (int n = MAXN) { clear (n); }
12        void add_edge (int u, int v, int c) { dest[size] = v; next[size] = begin[u]; cost[size] = c; begin[u] = size++; }

```

6.1 Hopcroft-Karp algorithm

```

1 /* Hopcroft-Karp algorithm : unweighted maximum matching for bipartition graphs with complexity  $O(m\sqrt{n})$ . */
2 template <int MAXN = 100000, int MAXM = 100000>
3 struct hopcroft_karp {
4     using edge_list = std::vector <int> [MAXN];
5     int mx[MAXN], my[MAXM], lv[MAXN];
6     bool dfs (edge_list <MAXN, MAXM> &e, int x) {
7         for (int i = e.begin[x]; ~i; i = e.next[i]) {
8             int y = e.dest[i], w = my[y];
9             if (!w || (lv[x] + 1 == lv[w] && dfs (e, w))) {
10                 mx[x] = y; my[y] = x; return true; } }
11         lv[x] = -1; return false; }
12    int solve (edge_list <MAXN, MAXM> &e, int n, int m) {
13        std::fill (mx, mx + n, -1); std::fill (my, my + m, -1);
14        for (int ans = 0; ; ) {
15            std::vector <int> q;
16            for (int i = 0; i < n; ++i)
17                if (mx[i] == -1) {
18                    lv[i] = 0; q.push_back (i);
19                } else lv[i] = -1;
20            for (int head = 0; head < (int) q.size (); ++head) {
21                int x = q[head];
22                for (int i = e.begin[x]; ~i; i = e.next[i]) {
23                    int y = e.dest[i], w = my[y];
24                    if (w && lv[w] < 0) { lv[w] = lv[x] + 1; q.push_back (w); } }
25            }
26            int d = 0; for (int i = 0; i < n; ++i) if (!mx[i] && dfs (e, i)) ++d;
27            if (d == 0) return ans; else ans += d; } }

```

6.2 Kuhn-Munkres algorithm

```

1 /* Kuhn Munkres algorithm : weighted maximum matching for bipartition graphs with complexity  $O(N^3)$ .
2 Note : The graph is 1-based. */
3 template <int MAXN = 500>
4 struct kuhn_munkres {
5     int n, w[MAXN][MAXN], lx[MAXN], ly[MAXN], m[MAXN], way[MAXN], sl[MAXN];
6     bool u[MAXN];
7     void hungary(int x) {
8         m[0] = x; int j0 = 0;
9         std::fill (sl, sl + n + 1, INF); std::fill (u, u + n + 1, false);
10        do {
11            u[j0] = true; int i0 = m[j0], d = INF, j1 = 0;
12            for (int j = 1; j <= n; ++j)
13                if (u[j] == false) {
14                    int cur = -w[i0][j] - lx[i0] - ly[j];
15                    if (cur < sl[j]) { sl[j] = cur; way[j] = j0; }
16                    if (sl[j] < d) { d = sl[j]; j1 = j; } }
17            for (int j = 0; j <= n; ++j) {
18                if (u[j]) { lx[m[j]] += d; ly[j] -= d; }
19                else sl[j] -= d; }
20            j0 = j1; } while (m[j0] != 0);
21        }
22        int j1 = way[j0]; m[j0] = m[j1]; j0 = j1;
23        } while (j0); }
24    int solve () {
25        for (int i = 1; i <= n; ++i) m[i] = lx[i] = ly[i] = way[i] = 0;
26        for (int i = 1; i <= n; ++i) hungary (i);
27        int sum = 0; for (int i = 1; i <= n; ++i) sum += w[m[i]][i];
28        return sum; } }

```

6.3 Blossom algorithm

```

1 /* Blossom algorithm : maximum match for general graph */
2 template <int MAXN = 500, int MAXM = 250000>
3 struct blossom {
4     int match[MAXN], d[MAXN], fa[MAXN], c1[MAXN], c2[MAXN], v[MAXN], q[MAXN];
5     int *qhead, *qtail;
6     struct {
7         int fa[MAXN];
8         void init (int n) { for(int i = 1; i <= n; ++i) fa[i] = i; }
9         int find (int x) { if (fa[x] != x) fa[x] = find (fa[x]); return fa[x]; }
10        void merge (int x, int y) { x = find (x); y = find (y); fa[x] = y; } } ufs;
11    void solve (int x, int y) {
12        if (x == y) return;
13        if (d[y] == 0) {
14            solve (x, fa[fa[y]]); match[fa[y]] = fa[fa[y]];
15            match[fa[fa[y]]] = fa[y];
16        } else if (d[y] == 1) {
17            solve (match[y], c1[y]); solve (x, c2[y]);

```



```

18 match[c1[y]] = c2[y]; match[c2[y]] = c1[y]; } }
19 int lca (int x, int y, int root) {
20 x = ufs.find (x); y = ufs.find (y);
21 while (x != y && v[x] != 1 && v[y] != 0) {
22 v[x] = 0; v[y] = 1;
23 if (x != root) x = ufs.find (fa[x]);
24 if (y != root) y = ufs.find (fa[y]); }
25 if (v[y] == 0) std::swap (x, y);
26 for (int i = x; i != y; i = ufs.find (fa[i])) v[i] =
    -1;
27 v[y] = -1; return x; }
28 void contract (int x, int y, int b) {
29 for (int i = ufs.find (x); i != b; i = ufs.find (fa[
    i])) {
30 ufs.merge (i, b);
31 if (d[i] == 1) { c1[i] = x; c2[i] = y; *qtail++ = i
    ; } } }
32 bool bfs (int root, int n, const edge_list <MAXN,
    MAXN> &e) {
33 ufs.init (n); std::fill (d, d + MAXN, -1); std::fill
    (v, v + MAXN, -1);
34 qhead = qtail = q; d[root] = 0; *qtail++ = root;
35 while (qhead < qtail) {
36 for (int loc = *qhead++; i = e.begin[loc]; ~i; i =
    e.next[i]) {
37 int dest = e.dest[i];
38 if (match[dest] == -2 || ufs.find (loc) == ufs.
    find (dest)) continue;
39 if (d[dest] == -1)
40 if (match[dest] == -1) {
41 solve (root, loc); match[loc] = dest;
42 match[dest] = loc; return 1;
43 } else {
44 fa[dest] = loc; fa[match[dest]] = dest;
45 d[dest] = 1; d[match[dest]] = 0;
46 *qtail++ = match[dest];
47 } else if (d[ufs.find (dest)] == 0) {
48 int b = lca (loc, dest, root);
49 contract (loc, dest, b); contract (dest, loc, b)
    ; } } }
50 return 0; } } }
51 int solve (int n, const edge_list <MAXN, MAXN> &e) {
52 std::fill (fa, fa + n, 0); std::fill (c1, c1 + n, 0)
    ;
53 std::fill (c2, c2 + n, 0); std::fill (match, match +
    n, -1);
54 int re = 0; for (int i = 0; i < n; i++)
55 if (match[i] == -1) if (bfs (i, n, e)) ++re; else
    match[i] = -2;
56 return re; } } }

```

6.4 Weighted blossom algorithm

```

1 /* Weighted blossom algorithm (vfleaking ver.) :
2 maximum matching for general weighted graphs with
3 complexity  $O(n^3)$ .
4 Usage : Set n to the size of the vertices. Run init ().
5 Set g[][].w to the weight of the edge. Run solve
6 ().
7 The first result is the answer, the second one is the
8 number of matching pairs. Obtain the matching with
9 match[].
10 Note : 1-based. */
11 struct weighted_blossom {
12 static const int INF = INT_MAX, MAXN = 400;
13 struct edge { int u, v, w; edge (int u = 0, int v = 0,
    int w = 0): u(u), v(v), w(w) {} };
14 int n, n_x;
15 edge g[MAXN * 2 + 1][MAXN * 2 + 1];
16 int lab[MAXN * 2 + 1], match[MAXN * 2 + 1], slack[
    MAXN * 2 + 1], st[MAXN * 2 + 1], pa[MAXN * 2 +
    1];
17 int flower_from[MAXN * 2 + 1][MAXN + 1], S[MAXN * 2 +
    1], vis[MAXN * 2 + 1];
18 std::vector<int> flower[MAXN * 2 + 1]; std::queue<
    int> q;
19 int e_delta (const edge &e) { return lab[e.u] + lab[e
    .v] - g[e.u][e.v].w * 2; }
20 void update_slack (int u, int x) { if (!slack[x] ||
    e_delta (g[u][x]) < e_delta (g[slack[x]][x]))
    slack[x] = u; }
21 void set_slack (int x) { slack[x] = 0; for (int u =
    1; u <= n; ++u) if (g[u][x].w > 0 && st[u] != x &&
    S[st[u]] == 0)
22 update_slack (u, x); }
23 void q_push (int x) {
24 if (x <= n) q.push (x);
25 else for (size_t i = 0; i < flower[x].size (); i++)
    q.push (flower[x][i]); }
26 void set_st (int x, int b) {
27 st[x] = b; if (x > n) for (size_t i = 0; i < flower[
    x].size (); i++) set_st (flower[x][i], b); }
28 int get_pr (int b, int xr) {
29 end ((), xr) - flower[b].begin ();
30 if (pr % 2 == 1) { std::reverse (flower[b].begin ()
    + 1, flower[b].end ()); return (int) flower[b].
    size () - pr; }
31 else return pr; }
32 void set_match (int u, int v) {
33 match[u] = g[u][v].v; if (u > n) {
34 edge e = g[u][v]; int xr = flower_from[u][e.u], pr
    = get_pr (u, xr);
35 for (int i = 0; i < pr; ++i) set_match (flower[u][i
    ], flower[u][i + 1]);
36 set_match (xr, v); std::rotate (flower[u].begin (),
    flower[u].begin () + pr, flower[u].end ()); }
37 }
38 void augment (int u, int v) {
39 for (; ) {
40 int xnv = st[match[u]]; set_match (u, v);
41 if (!xnv) return; set_match (xnv, st[pa[xnv]]);
42 u = st[pa[xnv]], v = xnv; } }

```

```

36 int get_lca (int u, int v) {
37 static int t = 0;
38 for (++t; u || v; std::swap (u, v)) {
39 if (u == 0) continue; if (vis[u] == t) return u;
40 vis[u] = t; u = st[match[u]]; if (u) u = st[pa[u]];
41 }
42 return 0; }
43 void add_blossom (int u, int lca, int v) {
44 int b = n + 1; while (b <= n_x && st[b]) ++b;
45 if (b > n_x) ++n_x;
46 lab[b] = 0, S[b] = 0;
47 match[b] = match[lca]; flower[b].clear ();
48 flower[b].push_back (lca);
49 for (int x = u, y; x != lca; x = st[pa[y]]) {
50 flower[b].push_back (x); flower[b].push_back (y =
    st[match[x]]); q.push (y); }
51 std::reverse (flower[b].begin () + 1, flower[b].end
    ());
52 for (int x = v, y; x != lca; x = st[pa[y]]) {
53 flower[b].push_back (x); flower[b].push_back (y =
    st[match[x]]); q.push (y); }
54 set_st (b, b);
55 for (int x = 1; x <= n_x; ++x) g[b][x].w = g[x][b].w
    = 0;
56 for (int x = 1; x <= n; ++x) flower_from[b][x] = 0;
57 for (size_t i = 0; i < flower[b].size (); ++i) {
58 int xs = flower[b][i];
59 for (int x = 1; x <= n_x; ++x) if (g[b][x].w == 0
    || e_delta (g[xs][x]) < e_delta (g[b][x]))
60 g[b][x] = g[xs][x], g[x][b] = g[x][xs];
61 for (int x = 1; x <= n; ++x) if (flower_from[xs][x])
    flower_from[b][x] = xs; }
62 set_slack (b); }
63 void expand_blossom (int b) {
64 for (size_t i = 0; i < flower[b].size (); ++i)
65 set_st (flower[b][i], flower[b][i]);
66 int xr = flower_from[b][g[b][pa[b]].u], pr = get_pr
    (b, xr);
67 for (int i = 0; i < pr; i += 2) {
68 int xs = flower[b][i], xns = flower[b][i + 1];
69 pa[xs] = g[xns][xs].u; S[xs] = 1, S[xns] = 0;
70 slack[xs] = 0, set_slack (xns); q_push (xns); }
71 S[xr] = 1, pa[xr] = pa[b];
72 for (size_t i = pr + 1; i < flower[b].size (); ++i)
73 int xs = flower[b][i]; S[xs] = -1, set_slack (xs); }
74 st[b] = 0; }
75 bool on_found_edge (const edge &e) {
76 int u = st[e.u], v = st[e.v];
77 if (S[v] == -1) {
78 pa[v] = e.u, S[v] = 1; int nu = st[match[v]];
79 slack[v] = slack[nu] = 0; S[nu] = 0, q_push (nu);
80 } else if (S[v] == 0) {
81 int lca = get_lca (u, v);
82 if (!lca) return augment (u, v), augment (v, u), true
    ;
83 else add_blossom (u, lca, v); }
84 return false; }
85 bool matching () {
86 memset (S + 1, -1, sizeof (int) * n_x);
87 memset (slack + 1, 0, sizeof (int) * n_x);
88 q = std::queue<int> ();
89 for (int x = 1; x <= n_x; ++x) if (st[x] == x && !
    match[x]) pa[x] = 0, S[x] = 0, q.push (x);
90 if (q.empty ()) return false;
91 for (; ) {
92 while (q.size ()) {
93 int u = q.front (); q.pop ();
94 if (S[st[u]] == 1) continue;
95 for (int v = 1; v <= n; ++v) if (g[u][v].w > 0 &&
    st[u] != st[v]) {
96 if (e_delta (g[u][v]) == 0) {
97 if (on_found_edge (g[u][v])) return true;
98 } else update_slack (u, st[v]); } } }
99 int d = INF;
100 for (int b = n + 1; b <= n_x; ++b) if (st[b] == b &&
    S[b] == 1) d = std::min (d, lab[b] / 2);
101 for (int x = 1; x <= n_x; ++x) if (st[x] == x &&
    slack[x]) {
102 if (S[x] == -1) d = std::min (d, e_delta (g[slack[
    x]][x]));
103 else if (S[x] == 0) d = std::min (d, e_delta (g[
    slack[x]][x]) / 2); }
104 for (int u = 1; u <= n; ++u) {
105 if (S[st[u]] == 0) {
106 if (lab[u] <= d) return 0;
107 lab[u] -= d;
108 } else if (S[st[u]] == 1) lab[u] += d; }
109 for (int b = n + 1; b <= n_x; ++b)
110 if (st[b] == b) {
111 if (S[st[b]] == 0) lab[b] += d * 2;
112 else if (S[st[b]] == 1) lab[b] -= d * 2; }
113 q = std::queue<int> ();
114 for (int x = 1; x <= n_x; ++x)
115 if (st[x] == x && slack[x] && st[slack[x]] != x &&
    e_delta (g[slack[x]][x]) == 0)
116 if (on_found_edge (g[slack[x]][x])) return true;
117 for (int b = n + 1; b <= n_x; ++b) if (st[b] == b
    && S[b] == 1 && lab[b] == 0) expand_blossom (b);
118 }
119 return false; }
120 std::pair<long long, int> solve () {
121 memset (match + 1, 0, sizeof (int) * n); n_x = n;
122 int n_matches = 0; long long tot_weight = 0;
123 for (int u = 0; u <= n; ++u) st[u] = u, flower[u].
    clear ();
124 int w_max = 0;
125 for (int u = 1; u <= n; ++u) for (int v = 1; v <= n;
    ++v) {
126 flower_from[u][v] = (u == v ? u : 0); w_max = std::
    max (w_max, g[u][v].w); }
127 for (int u = 1; u <= n; ++u) lab[u] = w_max;
128 while (matching ()) ++n_matches;
129 for (int u = 1; u <= n; ++u) if (match[u] && match[u]
    < u) tot_weight += g[u][match[u]].w;

```

```

127 return std::make_pair (tot_weight, n_matches); }
128 void init () { for (int u = 1; u <= n; ++u) for (int
    v = 1; v <= n; ++v) g[u][v] = edge (u, v, 0); }

```

6.5 Maximum flow

```

1 /* Sparse graph maximum flow : isap.*/
2 template <int MAXN = 1000, int MAXM = 100000>
3 struct isap {
4     struct flow_edge_list {
5         int size, begin[MAXN], dest[MAXM], next[MAXM], flow[
            MAXM];
6         void clear (int n) { size = 0; std::fill (begin,
            begin + n, -1); }
7         flow_edge_list (int n = MAXN) { clear (n); }
8         void add_edge (int u, int v, int f) {
9             dest[size] = v; next[size] = begin[u]; flow[size] =
                f; begin[u] = size++;
10            dest[size] = u; next[size] = begin[v]; flow[size] =
                0; begin[v] = size++; }
11        int pre[MAXN], d[MAXN], gap[MAXN], cur[MAXN];
12        int solve (flow_edge_list &e, int n, int s, int t) {
13            for (int i = 0; i < n; ++i) { pre[i] = d[i] = gap[i]
                = 0; cur[i] = e.begin[i]; }
14            gap[0] = n; int u = pre[s] = s, v, maxflow = 0;
15            while (d[s] < n) {
16                v = n; for (int i = cur[u]; ~i; i = e.next[i])
17                    if (e.flow[i] && d[u] == d[e.dest[i]] + 1) {
18                        v = e.dest[i]; cur[u] = i; break; }
19                if (v < n) {
20                    pre[v] = u; u = v;
21                    if (v == t) {
22                        int dflow = INF, p = t; u = s;
23                        while (p != s) { p = pre[p]; dflow = std::min (
24                            dflow, e.flow[cur[p]]); }
25                        maxflow += dflow; p = t;
26                        while (p != s) { p = pre[p]; e.flow[cur[p]] -=
27                            dflow; e.flow[cur[p] ^ 1] += dflow; } }
28                    else {
29                        int mindist = n + 1;
30                        for (int i = e.begin[u]; ~i; i = e.next[i])
31                            if (e.flow[i] && mindist > d[e.dest[i]]) {
32                                mindist = d[e.dest[i]]; cur[u] = i; }
33                        if (!--gap[d[u]]) return maxflow;
34                        gap[d[u] = mindist + 1]++; u = pre[u]; } }
35            return maxflow; } };
36 /* Dense graph maximum flow : dinic.*/
37 template <int MAXN = 1000, int MAXM = 100000>
38 struct dinic {
39     struct flow_edge_list {
40         int size, begin[MAXN], dest[MAXM], next[MAXM], flow[
            MAXM];
41         void clear (int n) { size = 0; std::fill (begin,
            begin + n, -1); }
42         flow_edge_list (int n = MAXN) { clear (n); }
43         void add_edge (int u, int v, int f) {
44             dest[size] = v; next[size] = begin[u]; flow[size] =
                f; begin[u] = size++;
45             dest[size] = u; next[size] = begin[v]; flow[size] =
                0; begin[v] = size++; }
46        int n, s, t, d[MAXN], w[MAXN], q[MAXN];
47        int bfs (flow_edge_list &e) {
48            std::fill (d, d + n, -1);
49            int l, r; q[l = r = 0] = s, d[s] = 0;
50            for (; l <= r; l++)
51                for (int k = e.begin[q[l]]; ~k; k = e.next[k])
52                    if (!d[e.dest[k]] && e.flow[k] > 0) d[e.dest[k]]
                    = d[q[l]] + 1, q[++r] = e.dest[k];
53            return ~d[t] ? 1 : 0; }
54        int dfs (flow_edge_list &e, int u, int ext) {
55            if (u == t) return ext; int k = w[u], ret = 0;
56            for (; ~k; k = e.next[k], w[u] = k) {
57                if (ext == 0) break;
58                if (d[e.dest[k]] == d[u] + 1 && e.flow[k] > 0) {
59                    int flow = dfs (e, e.dest[k], std::min (e.flow[k],
60                        ext));
61                    if (flow > 0) {
62                        e.flow[k] -= flow, e.flow[k ^ 1] += flow;
63                        ret += flow, ext -= flow; } }
64                if (!k) d[u] = -1; return ret; } }
65        int solve (flow_edge_list &e, int n_, int s_, int t_) {
66            int ans = 0; n = n_; s = s_; dinic::t = t_;
67            while (bfs (e)) {
68                for (int i = 0; i < n; ++i) w[i] = e.begin[i];
69                ans += dfs (e, s, INF); }
70            return ans; } };

```

6.6 Minimum cost flow

```

1 /* Sparse graph minimum cost flow : EK.*/
2 template <int MAXN = 1000, int MAXM = 100000>
3 struct minimum_cost_flow {
4     struct cost_flow_edge_list {
5         int size, begin[MAXN], dest[MAXM], next[MAXM], cost[
            MAXM], flow[MAXM];
6         void clear (int n) { size = 0; std::fill (begin,
            begin + n, -1); }
7         cost_flow_edge_list (int n = MAXN) { clear (n); }
8         void add_edge (int u, int v, int c, int f) {
9             dest[size] = v; next[size] = begin[u]; cost[size] =
                c; flow[size] = f; begin[u] = size++;
10            dest[size] = u; next[size] = begin[v]; cost[size] =
                -c; flow[size] = 0; begin[v] = size++; }
11        int n, s, t, prev[MAXN], dist[MAXN], occur[MAXN];
12        bool augment (cost_flow_edge_list &e) {
13            std::vector <int> queue;
14            std::fill (dist, dist + n, INF); std::fill (occur,
15                occur + n, 0);
16            dist[s] = 0; occur[s] = true; queue.push_back (s);

```

```

16 for (int head = 0; head < (int)queue.size(); ++head)
17 {
18     int x = queue[head];
19     for (int i = e.begin[x]; ~i; i = e.next[i]) {
20         int y = e.dest[i];
21         if (e.flow[i] && dist[y] > dist[x] + e.cost[i]) {
22             dist[y] = dist[x] + e.cost[i]; prev[y] = i;
23             if (!occur[y]) {
24                 occur[y] = true; queue.push_back (y); } } }
25     occur[x] = false;
26     return dist[t] < INF; }
27 std::pair <int, int> solve (cost_flow_edge_list &e,
28     int n_, int s_, int t_) {
29     n = n_; s = s_; t = t_; std::pair <int, int> ans =
30     std::make_pair (0, 0);
31     while (augment (e)) {
32         int num = INF;
33         for (int i = t; i != s; i = e.dest[prev[i] ^ 1]) {
34             num = std::min (num, e.flow[prev[i]]); }
35         ans.first += num;
36         for (int i = t; i != s; i = e.dest[prev[i] ^ 1]) {
37             e.flow[prev[i]] -= num; e.flow[prev[i] ^ 1] += num;
38             ans.second += num * e.cost[prev[i]]; } }
39     return ans; } };
40 /* Dense graph minimum cost flow : zkw.*/
41 template <int MAXN = 1000, int MAXM = 100000>
42 struct zkw_flow {
43     struct cost_flow_edge_list {
44         int size, begin[MAXN], dest[MAXM], next[MAXM], cost[
            MAXM], flow[MAXM];
45         void clear (int n) { size = 0; std::fill (begin,
            begin + n, -1); }
46         cost_flow_edge_list (int n = MAXN) { clear (n); }
47         void add_edge (int u, int v, int c, int f) {
48             dest[size] = v; next[size] = begin[u]; cost[size] =
                c; flow[size] = f; begin[u] = size++;
49             dest[size] = u; next[size] = begin[v]; cost[size] =
                -c; flow[size] = 0; begin[v] = size++; }
50        int n, s, t, tf, tc, dis[MAXN], slack[MAXN], visit[
            MAXN];
51        int modlable() {
52            int delta = INF;
53            for (int i = 0; i < n; i++) {
54                if (!visit[i] && slack[i] < delta) delta = slack[i];
55            }
56            slack[i] = INF;
57            if (delta == INF) return 1;
58            for (int i = 0; i < n; i++) if (visit[i]) dis[i] +=
59                delta;
60            return 0; }
61        int dfs (cost_flow_edge_list &e, int x, int flow) {
62            if (x == t) { tf += flow; tc += flow * (dis[s] - dis
63                [t]); return flow; }
64            visit[x] = 1; int left = flow;
65            for (int i = e.begin[x]; ~i; i = e.next[i])
66                if (e.flow[i] > 0 && !visit[e.dest[i]]) {
67                    int y = e.dest[i];
68                    if (dis[y] + e.cost[i] == dis[x]) {
69                        int delta = dfs (e, y, std::min (left, e.flow[i])
70                            );
71                        e.flow[i] -= delta; e.flow[i ^ 1] += delta; left
72                            -= delta;
73                        if (!left) { visit[x] = false; return flow; } }
74                    else
75                        slack[y] = std::min (slack[y], dis[x] + e.cost[i]
76                            - dis[y]); }
77            return flow - left; }
78        std::pair <int, int> solve (cost_flow_edge_list &e,
79            int n_, int s_, int t_) {
80            n = n_; s = s_; t = t_; tf = tc = 0;
81            std::fill (dis + 1, dis + t + 1, 0);
82            do { do {
83                std::fill (visit + 1, visit + t + 1, 0);
84                while (dfs (e, s, INF)); } while (!modlable ());
85            } while (tf < tc);
86            return std::make_pair (tf, tc); } };

```

6.7 Stoer Wagner algorithm

```

1 /* Stoer Wagner algorithm : Finds the minimum cut of
2    an undirected graph. (1-based)*/
3 template <int MAXN = 500>
4 struct stoer_wagner {
5     int n, edge[MAXN][MAXN];
6     int dist[MAXN];
7     bool vis[MAXN], bin[MAXN];
8     stoer_wagner () {
9         memset (edge, 0, sizeof (edge));
10        memset (bin, false, sizeof (bin)); }
11    int contract (int &s, int &t) {
12        memset (dist, 0, sizeof (dist));
13        memset (vis, false, sizeof (vis));
14        int i, j, k, mincut, maxc;
15        for (i = 1; i <= n; i++) {
16            k = -1; maxc = -1;
17            for (j = 1; j <= n; j++)
18                if (!bin[j] && !vis[j] && dist[j] > maxc) {
19                    k = j; maxc = dist[j]; }
20            if (k == -1) return mincut;
21            s = t; t = k; mincut = maxc; vis[k] = true;
22            for (j = 1; j <= n; j++) if (!bin[j] && !vis[j])
23                dist[j] += edge[k][j];
24            return mincut; } }
25    int solve () {
26        int mincut, i, j, s, t, ans;
27        for (mincut = INF, i = 1; i < n; i++) {
28            ans = contract (s, t); bin[t] = true;
29            if (mincut > ans) mincut = ans;
30            if (mincut == 0) return 0;
31            for (j = 1; j <= n; j++) if (!bin[j])
32                edge[s][j] = (edge[j][s] += edge[j][t]); }
33        return mincut; } };

```


6.8 DN maximum clique

```

1 /* DN maximum clique : n <= 150 */
2 typedef bool BB[N]; struct Maxclique {
3 const BB *e; int pk, level; const float Tlimit;
4 struct Vertex { int i, d; Vertex (int i) : i(i), d(0)
5 {} };
6 typedef std::vector <Vertex> Vertices; Vertices V;
7 typedef std::vector <int> ColorClass; ColorClass QMAX,
8 Q;
9 std::vector <ColorClass> C;
10 static bool desc_degree (const Vertex &vi, const Vertex
11 &vj) { return vi.d > vj.d; }
12 void init_colors (Vertices &v) {
13 const int max_degree = v[0].d;
14 for (int i = 0; i < (int) v.size(); ++i) v[i].d = std
15 ::min (i, max_degree) + 1;
16 void set_degrees (Vertices &v) {
17 for (int i = 0; i < (int) v.size(); ++i)
18 for (int j = 0; j < (int) v.size(); ++j)
19 v[i].d += e[v[i].i][v[j].i];
20 struct StepCount { int i1, i2; StepCount() : i1 (0), i2
21 (0) {} };
22 std::vector <StepCount> S;
23 bool cut1 (const int pi, const ColorClass &A) {
24 for (int i = 0; i < (int) A.size(); ++i)
25 if (e[pi][A[i]]) return true; return false; }
26 void cut2 (const Vertices &A, Vertices &B) {
27 for (int i = 0; i < (int) A.size() - 1; ++i)
28 if (e[A.back().i][A[i].i]) B.push_back(A[i].i); }
29 void color_sort (Vertices &R) {
30 int j = 0, maxno = 1, min_k = std::max ((int) QMAX.
31 size () - (int) Q.size () + 1, 1);
32 C[1].clear (); C[2].clear ();
33 for (int i = 0; i < (int) R.size(); ++i) {
34 int pi = R[i].i, k = 1; while (cut1(pi, C[k])) ++k;
35 if (k > maxno) maxno = k, C[maxno + 1].clear();
36 C[k].push_back (pi); if (k < min_k) R[j++] .i = pi; }
37 if (j > 0) R[j - 1].d = 0;
38 for (int k = min_k; k <= maxno; ++k)
39 for (int i = 0; i < (int) C[k].size(); ++i)
40 R[j++] .i = C[k][i], R[j++] .d = k; }
41 void expand_dyn (Vertices &R) {
42 S[level].i1 = S[level].i1 + S[level - 1].i1 - S[level
43 ].i2;
44 S[level].i2 = S[level - 1].i1;
45 while ((int) R.size ()) {
46 if ((int) Q.size () + R.back ().d > (int) QMAX.size
47 ()) {
48 Q.push_back (R.back ().i); Vertices Rp; cut2 (R, Rp
49 );
50 if ((int) Rp.size ()) {
51 if ((float) S[level].i1 / ++pk < Tlimit)
52 degree_sort (Rp);
53 color_sort (Rp); ++S[level].i1, ++level;
54 expand_dyn (Rp); --level;
55 } else if ((int) Q.size () > (int) QMAX.size ())
56 QMAX = Q;
57 Q.pop_back (); } else return; R.pop_back (); } }
58 void mcqdyn (int *maxclique, int &sz) {
59 set_degrees (V); std::sort(V.begin (), V.end (),
60 desc_degree); init_colors (V);
61 for (int i = 0; i < (int) V.size () + 1; ++i) S[i].i1
62 = S[i].i2 = 0;
63 expand_dyn (V); sz = (int) QMAX.size ();
64 for (int i = 0; i < (int) QMAX.size (); ++i)
65 maxclique[i] = QMAX[i];
66 void degree_sort (Vertices &R) {
67 set_degrees(R); std::sort(R.begin(), R.end(),
68 desc_degree); }
69 Maxclique (const BB *conn, const int sz, const float
70 tt = .025) : pk (0), level (1), Tlimit (tt) {
71 for (int i = 0; i < sz; ++i) V.push_back (Vertex (i));
72 e = conn, C.resize (sz + 1), S.resize (sz + 1); }
73 BB e[N]; int ans, sol[N]; for (...) e[x][y] = e[y][x]
74 = true;
75 Maxclique mc (e, n); mc.mcqdyn (sol, ans); //0-based.
76 for (int i = 0; i < ans; ++i) std::cout << sol[i] <<
77 std::endl;

```

6.9 Dominator tree

```

1 /* Dominator tree : finds the immediate dominator (
2 idom[]) of each node, idom[x] will be x if x does
3 not have a dominator, and will be -1 if x is not
4 reachable from s. */
5 template <int MAXN = 100000, int MAXM = 100000>
6 struct dominator_tree {
7 using edge_list = std::vector <int> [MAXN];
8 int dfn[MAXN], sdom[MAXN], idom[MAXN], id[MAXN], f[
9 MAXN], fa[MAXN], smin[MAXN], stamp;
10 void predfs (int x, const edge_list <MAXN, MAXM> &
11 succ) {
12 id[dfn[x] = stamp++] = x;
13 for (int i = succ.begin[x]; ~i; i = succ.next[i]) {
14 int y = succ.dest[i];
15 if (dfn[y] < 0) { f[y] = x; predfs (y, succ); } } }
16 int getfa (int x) {
17 if (fa[x] == x) return x;
18 int ret = getfa (fa[x]);
19 if (dfn[sdom[smin[fa[x]]]] < dfn[sdom[smin[x]]])
20 smin[x] = smin[fa[x]];
21 return fa[x] = ret; }
22 void solve (int s, int n, const edge_list <MAXN, MAXM>
23 & succ) {
24 std::fill (dfn, dfn + n, -1); std::fill (idom, idom
25 + n, -1);
26 static edge_list <MAXN, MAXM> pred, tmp; pred.clear
27 (n);
28 for (int i = 0; i < n; ++i) for (int j = succ.begin[
29 i]; ~j; j = succ.next[j])
30 pred.add_edge (succ.dest[j], i);

```

```

21 stamp = 0; tmp.clear (n); predfs (s, succ);
22 for (int i = 0; i < stamp; ++i) fa[id[i]] = smin[id[
23 i]];
24 for (int o = stamp - 1; o >= 0; --o) {
25 int x = id[o];
26 if (o) {
27 sdom[x] = f[x];
28 for (int i = pred.begin[x]; ~i; i = pred.next[i])
29 {
30 int p = pred.dest[i];
31 if (dfn[p] < 0) continue;
32 if (dfn[p] > dfn[x]) { getfa (p); p = sdom[smin[p]
33 ]; }
34 if (dfn[sdom[x]] > dfn[p]) sdom[x] = p; }
35 tmp.add_edge (sdom[x], x); }
36 while (~tmp.begin[x]) {
37 int y = tmp.dest[tmp.begin[x]]; getfa (y);
38 tmp.begin[x] = tmp.next[tmp.begin[x]]; getfa (y);
39 if (x != sdom[smin[y]]) idom[y] = smin[y];
40 else idom[y] = x; }
41 for (int v : succ[x]) if (f[v] == x) fa[v] = x; }
42 idom[s] = s; for (int i = 1; i < stamp; ++i) {
43 int x = id[i]; if (idom[x] != sdom[x]) idom[x] =
44 idom[idom[x]]; } } }

```

6.10 Tarjan

```

1 /* Tarjan : strongly-connected components. */
2 template <int MAXN = 1000000>
3 struct tarjan {
4 int comp[MAXN], size;
5 int dfn[MAXN], ind, low[MAXN], ins[MAXN], stk[MAXN],
6 stks;
7 void dfs (const edge_list <MAXN, MAXM> &e, int i) {
8 dfn[i] = low[i] = ind++;
9 ins[i] = 1; stk[stks++] = i;
10 for (int x = e.begin[i]; ~x; x = e.next[x]) {
11 int j = e.dest[x]; if (!dfn[j]) {
12 dfs (j);
13 if (low[i] > low[j]) low[i] = low[j];
14 if (low[j] >= dfn[i]); //vertex-biconnected
15 if (low[j] > dfn[i]); //edge-biconnected
16 } else if (ins[j] && low[i] > dfn[j])
17 low[i] = dfn[j]; }
18 if (dfn[i] == low[i]) { //strongly-connected
19 for (int j = -1; j != i;
20 j = stk[--stks], ins[j] = 0, comp[j] = size);
21 ++size; } }
22 void solve (const edge_list <MAXN, MAXM> &e, int n) {
23 size = ind = stks = 0;
24 std::fill (dfn, dfn + n, -1);
25 for (int i = 0; i < n; ++i) if (!dfn[i])
26 dfs (e, i); } }

```

7 String

7.1 Suffix Array

```

1 /* Suffix Array : sa[i] - the beginning position of
2 the ith smallest suffix, rk[i] - the rank of the
3 suffix beginning at position i. height[i] - the
4 longest common prefix of sa[i] and sa[i - 1]. */
5 template <int MAXN = 1000000, int MAXC = 26>
6 struct suffix_array {
7 int rk[MAXN], height[MAXN], sa[MAXN];
8 int cmp (int *x, int a, int b, int d) {
9 return x[a] == x[b] && x[a + d] == x[b + d]; }
10 void doubling (int *a, int n) {
11 static int sRank[MAXN], tmpA[MAXN], tmpB[MAXN];
12 int m = MAXC, *x = tmpA, *y = tmpB;
13 for (int i = 0; i < m; ++i) sRank[i] = 0;
14 for (int i = 0; i < n; ++i) ++sRank[x[i] = a[i]];
15 for (int i = 1; i < m; ++i) sRank[i] += sRank[i -
16 1];
17 for (int i = n - 1; i >= 0; --i) sa[--sRank[x[i]]] =
18 i;
19 for (int d = 1, p = 0; p < n; m = p, d <= 1) {
20 p = 0; for (int i = n - d; i < n; ++i) y[p++] = i;
21 for (int i = 0; i < n; ++i) if (sa[i] >= d) y[p++]
22 = sa[i] - d;
23 for (int i = 0; i < m; ++i) sRank[i] = 0;
24 for (int i = 0; i < n; ++i) ++sRank[x[i] = a[i]];
25 for (int i = 1; i < m; ++i) sRank[i] += sRank[i -
26 1];
27 for (int i = n - 1; i >= 0; --i) sa[--sRank[x[i]]] =
28 i;
29 std::swap (x, y); x[sa[0]] = 0; p = 1; y[n] = -1;
30 for (int i = 1; i < n; ++i)
31 x[sa[i]] = cmp (y, sa[i], sa[i - 1], d) ? p - 1 :
32 p++; } }
33 void solve (int *a, int n) {
34 a[n] = -1; doubling (a, n);
35 for (int i = 0; i < n; ++i) rk[sa[i]] = i;
36 int cur = 0;
37 for (int i = 0; i < n; ++i)
38 if (rk[i]) {
39 if (cur) cur--;
40 for (; a[i + cur] == a[sa[rk[i] - 1] + cur]; ++cur)
41 ;
42 height[rk[i]] = cur; } } }

```

7.2 Suffix Automaton

```

1 /* Suffix automaton : head - the first state. tail -
2 the last state. Terminating states can be reached
3 via visiting the ancestors of tail. state::len -
4 the longest length of the string in the state.
5 state::right - 1 - the first location in the
6 string where the state can be reached. state::
7 parent - the parent link. state::dest - the
8 automaton link. */

```



```

2 template <int MAXN = 1000000, int MAXC = 26>
3 struct suffix_automaton {
4     struct state {
5         int len, right; state *parent, *dest[MAXN];
6         state (int len = 0, int right = 0) : len (len),
7             right (right), parent (NULL) {
8             memset (dest, 0, sizeof (dest));
9         }
10    } node_pool[MAXN * 2], *tot_node, *null = new state();
11
12    state *head, *tail;
13    void extend (int token) {
14        state *p = tail;
15        state *np = tail -> dest[token] ? null : new (
16            tot_node++) state (tail -> len + 1, tail -> len
17            + 1);
18        while (p && !p -> dest[token]) p -> dest[token] = np;
19        if (!p) np -> parent = head;
20        else {
21            state *q = p -> dest[token];
22            if (p -> len + 1 == q -> len) {
23                np -> parent = q;
24            } else {
25                state *nq = new (tot_node++) state (*q);
26                nq -> len = p -> len + 1;
27                np -> parent = q -> parent = nq;
28                while (p && p -> dest[token] == q) {
29                    p -> dest[token] = nq, p = p -> parent;
30                }
31            }
32            tail = np == null ? np -> parent : np;
33        }
34    }
35    void init () {
36        tot_node = node_pool;
37        head = tail = new (tot_node++) state ();
38    }
39    suffix_automaton () { init (); }
40 };

```

7.3 Palindromic tree

```

1 /* Palindromic tree : extend () - returns whether the
2    tree has generated a new node. odd, even - the
3    root of two trees. last - the node representing
4    the last char. node::len - the palindromic string
5    length of the node. */
6 template <int MAXN = 1000000, int MAXC = 26>
7 struct palindromic_tree {
8     struct node {
9         node *child[MAXN], *fail; int len;
10        node (int len) : fail (NULL), len (len) {
11            memset (child, NULL, sizeof (child));
12        }
13    } node_pool[MAXN * 2], *tot_node;
14    int size, text[MAXN];
15    node *odd, *even, *last;
16    node *match (node *now) {
17        for (; text[size - now -> len - 1] != text[size];
18            now = now -> fail);
19        return now;
20    }
21    bool extend (int token) {
22        text[++size] = token; node *now = match (last);
23        if (now -> child[token])
24            return last = now -> child[token], false;
25        last = now -> child[token] = new (tot_node++) node (
26            now -> len + 2);
27        if (now == odd) last -> fail = even;
28        else {
29            now = match (now -> fail);
30            last -> fail = now -> child[token];
31        }
32        return true;
33    }
34    void init () {
35        text[size = 0] = -1; tot_node = node_pool;
36        last = even = new (tot_node++) node (0); odd = new (
37            tot_node++) node (-1);
38        even -> fail = odd;
39    }
40    palindromic_tree () { init (); }
41 };

```

7.4 Regular expression

```

1 std::string str = ("The_the_there");
2 std::regex pattern ("(th|Th)[\\w]*", std::
3     regex_constants::optimize | std::regex_constants::
4     ECMAScript);
5 std::cmatch m; //std::cmatch for char *
6 std::regex_match (str, m, pattern);
7
8 auto mbegin = std::sregex_iterator (str.begin (), str.
9     end (), pattern);
10 auto mend = std::sregex_iterator ();
11 std::cout << "Found_" << std::distance (mbegin, mend)
12     << " words:\n";
13 for (std::sregex_iterator i = mbegin; i != mend; ++i)
14 {
15     match = *i;
16     /* The word is match[0], backreferences are match[i]
17        up to match.size ().
18     match.prefix () and match.suffix () give the prefix
19        and the suffix.
20     match.length () gives length and match.position ()
21        gives position of the match. */
22     std::regex_replace (str, pattern, "sh$1");
23     // $n is the backreference, $& is the entire match, $'
24     is the prefix, $' is the suffix, $$ is the $ sign.
25 }

```

8 Tips

8.1 Java

```

1 /* Java reference : References on Java IO, structures,
2    etc. */
3 import java.io.*;
4 import java.lang.*;
5 import java.math.*;
6 import java.util.*;

```

```

6 /* Common usage:
7 Scanner in = new Scanner (System.in);
8 Scanner in = new Scanner (new BufferedInputStream (
9     System.in));
10 in.nextInt () / in.nextBigInteger () / in.
11     nextBigDecimal () / in.nextDouble ()
12 in.nextLine () / in.hasNext ()
13 System.out.print (...);
14 System.out.println (...);
15 System.out.printf (...);
16 BigInteger : BigInteger.valueOf (int) / abs / negate
17     () / max / min / add / subtract / multiply /
18     divide / remainder (BigInteger) / gcd (BigInteger)
19     / modInverse (BigInteger mod) / modPow (
20     BigInteger ex, BigInteger mod) / pow (int ex) /
21     not () / and / or / xor (BigInteger) / shiftLeft /
22     shiftRight (int) / compareTo (BigInteger) /
23     intValue () / longValue () / toString (int radix)
24     / isProbablePrime (int certainty) /
25     nextProbablePrime ()
26 BigDecimal : consists of a BigInteger value and a
27     scale. The scale is the number of digits to the
28     right of the decimal point.
29 divide (BigDecimal) : exact divide.
30 divide (BigDecimal, int scale, RoundingMode
31     roundingMode) : divide with roundingMode, which
32     may be: CEILING / DOWN / FLOOR / HALF_DOWN /
33     HALF_EVEN / HALF_UP / UNNECESSARY / UP.
34 BigDecimal.setScale (int newScale, RoundingMode
35     roundingMode) : returns a BigDecimal with newScale
36     .
37 doubleValue () / toPlainString () : converts to other
38     types.
39 Arrays : Arrays.sort (T [] a); Arrays.sort (T [] a,
40     int fromIndex, int toIndex); Arrays.sort (T [] a,
41     int fromIndex, int toIndex, Comparator <? super T>
42     comparator);
43 LinkedList <E> : addFirst / addLast (E) / getFirst /
44     getLast / removeFirst / removeLast () / clear () /
45     add (int, E) / remove (int) / size () / contains
46     / removeFirstOccurrence / removeLastOccurrence (E)
47 ListIterator <E> listIterator (int index) : returns an
48     iterator :
49     E next / previous () : accesses and iterates.
50     hasNext / hasPrevious () : checks availability.
51     nextIndex / previousIndex () : returns the index of a
52     subsequent call.
53     add / set (E) / remove () : changes element.
54 PriorityQueue <E> (int initcap, Comparator <? super E>
55     comparator) : add (E) / clear () / iterator () /
56     peek () / poll () / size ()
57 TreeMap <K, V> (Comparator <? super K> comparator) :
58     Map.Entry <K, V> ceilingEntry / floorEntry /
59     higherEntry / lowerEntry (K): getKey / getValue ()
60     / setValue (V) : entries.
61     clear () / put (K, V) / get (K) / remove (K) / size
62     ()
63 StringBuilder : StringBuilder (string) / append (int,
64     string, ...) / insert (int offset, ...) charAt (
65     int) / setCharAt (int, char) / delete (int, int) /
66     reverse () / length () / toString ()
67 String : String.format (String, ...) / toLowerCase /
68     toUpperCase () */
69 /* Examples on Comparator :
70 public class Main {
71     public static class Point {
72         public int x; public int y;
73         public Point () {
74             x = 0;
75             y = 0;
76         }
77         public Point (int xx, int yy) {
78             x = xx;
79             y = yy;
80         }
81     }
82     public static class Cmp implements Comparator <Point>
83     {
84         public int compare (Point a, Point b) {
85             if (a.x < b.x) return -1;
86             if (a.x == b.x) {
87                 if (a.y < b.y) return -1;
88                 if (a.y == b.y) return 0;
89                 return 1;
90             }
91         }
92     }
93     public static void main (String [] args) {
94         Cmp c = new Cmp ();
95         TreeMap <Point, Point> t = new TreeMap <Point, Point>
96             (> (c));
97         return;
98     }
99 }
100 */
101 /* or :
102 public static class Point implements Comparable <
103     Point> {
104     public int x; public int y;
105     public Point () {
106         x = 0;
107         y = 0;
108     }
109     public Point (int xx, int yy) {
110         x = xx;
111         y = yy;
112     }
113     public int compareTo (Point p) {
114         if (x < p.x) return -1;
115         if (x == p.x) {
116             if (y < p.y) return -1;
117             if (y == p.y) return 0;
118             return 1;
119         }
120     }
121     public boolean equalTo (Point p) {
122         return (x == p.x && y == p.y);
123     }
124     public int hashCode () {
125         return x + y;
126     }
127 }
128 */
129 /* Faster IO :
130 public class Main {
131     static class InputReader {
132         public BufferedReader reader;
133         public StringTokenizer tokenizer;
134         public InputReader (InputStream stream) {

```

```

80 reader = new BufferedReader (new InputStreamReader
81 (stream), 32768);
82 tokenizer = null; }
83 public String next() {
84 while (tokenizer == null || !tokenizer.
85 hasMoreTokens()) {
86 try {
87 String line = reader.readLine();
88 tokenizer = new StringTokenizer (line);
89 } catch (IOException e) {
90 throw new RuntimeException (e); } }
91 return tokenizer.nextToken(); }
92 public BigInteger nextBigInteger() {
93 return new BigInteger (next (), 10); /* radix */ }
94 public int nextInt() {
95 return Integer.parseInt (next()); }
96 public double nextDouble() {
97 return Double.parseDouble (next()); } }
98 public static void main (String[] args) {
99 InputReader in = new InputReader (System.in);
100 } }

```

8.2 Random numbers

```

1 std::mt19937_64 mt (time (0));
2 std::uniform_int_distribution <int> uid (1, 100);
3 std::uniform_real_distribution <double> urd (1, 100);
4 std::cout << uid (mt) << " " << urd (mt) << "\n";

```

8.3 Read hack

```

1 #define __attribute__ ((optimize ("-O3")))
2 #define __inline__ __attribute__ ((__gnu_inline__,
3 __always_inline__, __artificial__))
4 int next_int () {
5 const int SIZE = 110000; static char buf[SIZE + 1];
6 static int p = SIZE;
7 register int ans = 0, f = 1, sgn = 1;
8 while ((p < SIZE || (p = 0, buf[fread (buf, 1, SIZE,
9 stdin)] = 0, buf[0])) && (isdigit (buf[p]) && (
10 ans = ans * 10 + buf[p] - '0', f = 0, 1) || f &&
11 (buf[p] == '-' && (sgn = 0), 1))) ++p;
12 return sgn ? ans : -ans; }

```

8.4 Stack hack

```

1 //C++
2 #pragma comment (linker, "/STACK:36777216")
3 //G++
4 int __size__ = 256 << 20;
5 char * __p__ = (char*) malloc (__size__ + __size__);
6 __asm__ ("movl %0, %%esp\n" :: "r" (__p__));

```

8.5 Time hack

```

1 clock_t t = clock ();
2 std::cout << 1. * t / CLOCKS_PER_SEC << "\n";

```

8.6 Builtin functions

1. `__builtin_clz`: Returns the number of leading 0-bits in `x`, starting at the most significant bit position. If `x` is 0, the result is undefined.
2. `__builtin_ctz`: Returns the number of trailing 0-bits in `x`, starting at the least significant bit position. If `x` is 0, the result is undefined.
3. `__builtin_clzsb`: Returns the number of leading redundant sign bits in `x`, i.e. the number of bits following the most significant bit that are identical to it. There are no special cases for 0 or other values.
4. `__builtin_popcount`: Returns the number of 1-bits in `x`.
5. `__builtin_parity`: Returns the parity of `x`, i.e. the number of 1-bits in `x` modulo 2.
6. `__builtin_bswap16`, `__builtin_bswap32`, `__builtin_bswap64`: Returns `x` with the order of the bytes (8 bits as a group) reversed.
7. `bitset::Find_first()`, `bitset::Find_next(idx)`: `bitset` built-in functions.

8.7 Prufer sequence

In combinatorial mathematics, the Prufer sequence of a labeled tree is a unique sequence associated with the tree. The sequence for a tree on n vertices has length $n - 2$.

One can generate a labeled tree's Prufer sequence by iteratively removing vertices from the tree until only two vertices remain. Specifically, consider a labeled tree T with vertices $1, 2, \dots, n$. At step i , remove the leaf with the smallest label and set the i th element of the Prufer sequence to be the label of this leaf's neighbour.

One can generate a labeled tree from a sequence in three steps. The tree will have $n + 2$ nodes, numbered from 1 to $n + 2$. For each node set its degree to the number of times it appears in the sequence plus 1. Next, for each number in the sequence $a[i]$, find the first (lowest-numbered) node, j , with degree equal to 1, add the edge $(j, a[i])$ to the tree, and decrement the degrees of j and $a[i]$. At the end of this loop two nodes with degree 1 will remain (call them u, v). Lastly, add the edge (u, v) to the tree.

The Prufer sequence of a labeled tree on n vertices is a unique sequence of length $n - 2$ on the labels 1 to n - this much is clear. Somewhat less obvious is the fact that for a given sequence S of length $n - 2$ on the labels 1 to n , there is a unique labeled tree whose Prufer sequence is S .

8.8 Spanning tree counting

Kirchhoff's Theorem: the number of spanning trees in a graph G is equal to *any* cofactor of the Laplacian matrix of G , which is equal to the difference between the graph's degree matrix (a diagonal matrix with vertex degrees on the diagonals) and its adjacency matrix (a $(0,1)$ -matrix with 1's at places corresponding to entries where the vertices are adjacent and 0's otherwise).

The number of edges with a certain weight in a minimum spanning tree is fixed given a graph. Moreover, the number of its arrangements can be obtained by finding a minimum spanning tree, compressing connected components of other edges in that tree into a point, and then applying Kirchhoff's theorem with only edges of the certain weight in the graph. Therefore, the number of minimum spanning trees in a graph can be solved by multiplying all numbers of arrangements of edges of different weights together.

8.9 Mobius inversion

8.9.1 Mobius inversion formula

$$[x = 1] = \sum_{d|x} \mu(d)$$

$$x = \sum_{d|x} \mu(d)$$

8.9.2 Gcd inversion

$$\sum_{a=1}^n \sum_{b=1}^n \gcd^2(a, b) = \sum_{d=1}^n d^2 \sum_{i=1}^{\lfloor \frac{n}{d} \rfloor} \sum_{j=1}^{\lfloor \frac{n}{d} \rfloor} [\gcd(i, j) = 1]$$

$$= \sum_{d=1}^n d^2 \sum_{i=1}^{\lfloor \frac{n}{d} \rfloor} \sum_{j=1}^{\lfloor \frac{n}{d} \rfloor} \mu(\gcd(i, j))$$

$$= \sum_{d=1}^n d^2 \sum_{t=1}^{\lfloor \frac{n}{d} \rfloor} \mu(t) \sum_{i=1}^{\lfloor \frac{n}{dt} \rfloor} [t|i] \sum_{j=1}^{\lfloor \frac{n}{dt} \rfloor} [t|j]$$

$$= \sum_{d=1}^n d^2 \sum_{t=1}^{\lfloor \frac{n}{d} \rfloor} \mu(t) \left\lfloor \frac{n}{dt} \right\rfloor^2$$

The formula can be computed in $O(n \log n)$ complexity. Moreover, let $l = dt$, then

$$\sum_{d=1}^n d^2 \sum_{t=1}^{\lfloor \frac{n}{d} \rfloor} \mu(t) \left\lfloor \frac{n}{dt} \right\rfloor^2 = \sum_{l=1}^n \left\lfloor \frac{n}{l} \right\rfloor^2 \sum_{d|l} d^2 \mu\left(\frac{l}{d}\right)$$

Let $f(l) = \sum_{d|l} d^2 \mu\left(\frac{l}{d}\right)$. It can be proven that $f(l)$ is multiplicative. Besides, $f(p^k) = p^{2k} - p^{2k-2}$.

Therefore, with linear sieve the formula can be computed in $O(n)$ complexity.

8.10 2-SAT

In terms of the implication graph, two literals belong to the same strongly connected component whenever there exist chains of implications from one literal to the other and vice versa. Therefore, the two literals must have the same value in any satisfying assignment to the given 2-satisfiability instance. In particular, if a variable and its negation both belong to the same strongly connected component, the instance cannot be satisfied, because it is impossible to assign both of these literals the same value. As Aspvall et al. showed, this is a necessary and sufficient condition: a 2-CNF formula is satisfiable if and only if there is no variable that belongs to the same strongly connected component as its negation.

This immediately leads to a linear time algorithm for testing satisfiability of 2-CNF formulae: simply perform a strong connectivity analysis on the implication graph and check that each variable and its negation belong to different components. However, as Aspvall et al. also showed, it also leads to a linear time algorithm for finding a satisfying assignment, when one exists. Their algorithm performs the following steps:

Construct the implication graph of the instance, and find its strongly connected components using any of the known linear-time algorithms for strong connectivity analysis.

Check whether any strongly connected component contains both a variable and its negation. If so, report that the instance is not satisfiable and halt.

Construct the condensation of the implication graph, a smaller graph that has one vertex for each strongly connected component, and an edge from component i to component j whenever the implication graph contains an edge uv such that u belongs to component i and v belongs to component j . The condensation is automatically a directed acyclic graph and, like the implication graph from which it was formed, it is skew-symmetric.

Topologically order the vertices of the condensation. In practice this may be efficiently achieved as a side effect of the previous step, as components are generated by Kosaraju's algorithm in topological order and by Tarjan's algorithm in reverse topological order.

For each component in the reverse topological order, if its variables do not already have truth assignments, set all the literals in the component to be true. This also causes all of the literals in the complementary component to be set to false.

8.11 Interesting numbers

8.11.1 Binomial Coefficients

$$\binom{n}{k} = (-1)^k \binom{k-n-1}{k}, \quad \sum_{k \leq n} \binom{r+k}{k} = \binom{r+n+1}{n}$$

$$\sum_{k=0}^n \binom{k}{m} = \binom{n+1}{m+1}$$

$$\sqrt{1+z} = 1 + \sum_{k=1}^{\infty} \frac{(-1)^{k-1}}{k \times 2^{k-1}} \binom{2k-2}{k-1} z^k$$

$$\sum_{k=0}^r \binom{r-k}{m} \binom{s+k}{n} = \binom{r+s+1}{m+n+1}$$

$$C_{n,m} = \binom{n+m}{m} - \binom{n+m}{m-1}, n \geq m$$

$$\binom{n}{k} \equiv [n \& k = k] \pmod{2}$$

$$\binom{n_1 + \dots + n_p}{m} = \sum_{k_1 + \dots + k_p = m} \binom{n_1}{k_1} \dots \binom{n_p}{k_p}$$

8.11.2 Fibonacci Numbers

$$F(z) = \frac{z}{1-z-z^2}$$

$$f_n = \frac{\phi^n - \hat{\phi}^n}{\sqrt{5}}, \phi = \frac{1+\sqrt{5}}{2}, \hat{\phi} = \frac{1-\sqrt{5}}{2}$$

$$\sum_{k=1}^n f_k = f_{n+2} - 1, \quad \sum_{k=1}^n f_k^2 = f_n f_{n+1}$$

$$\sum_{k=0}^n f_k f_{n-k} = \frac{1}{5}(n-1)f_n + \frac{2}{5}n f_{n-1}$$

$$\frac{f_{2n}}{f_n} = f_{n-1} + f_{n+1}$$

$$f_1 + 2f_2 + 3f_3 + \dots + n f_n = n f_{n+2} - f_{n+3} + 2]$$

$$\gcd(f_m, f_n) = f_{\gcd(m,n)}$$

$$f_n^2 + (-1)^n = f_{n+1} f_{n-1}$$

$$f_{n+k} = f_n f_{k+1} + f_{n-1} f_k$$

$$f_{2n+1} = f_n^2 + f_{n+1}^2$$

$$(-1)^k f_{n-k} = f_n f_{k-1} - f_{n-1} f_k$$

$$\text{Modulo } f_n, f_{mn+r} \equiv \begin{cases} f_r, & m \bmod 4 = 0; \\ (-1)^{r+1} f_{n-r}, & m \bmod 4 = 1; \\ (-1)^n f_r, & m \bmod 4 = 2; \\ (-1)^{r+1+n} f_{n-r}, & m \bmod 4 = 3. \end{cases}$$

8.11.3 Lucas Numbers

$$L_0 = 2, L_1 = 1, L_n = L_{n-1} + L_{n-2} = \left(\frac{1+\sqrt{5}}{2}\right)^n + \left(\frac{1-\sqrt{5}}{2}\right)^n$$

$$L(x) = \frac{2-x}{1-x-x^2}$$

8.11.4 Catlan Numbers

$$c_1 = 1, c_n = \sum_{i=0}^{n-1} c_i c_{n-1-i} = c_{n-1} \frac{4n-2}{n+1} = \frac{\binom{2n}{n}}{n+1}$$

$$= \binom{2n}{n} - \binom{2n}{n-1}, c(x) = \frac{1 - \sqrt{1-4x}}{2x}$$

8.11.5 Stirling Cycle Numbers

Divide n elements into k non-empty cycles.

$$s(n, 0) = 0, s(n, n) = 1, s(n+1, k) = s(n, k-1) - ns(n, k)$$

$$s(n, k) = (-1)^{n-k} \left[\begin{matrix} n \\ k \end{matrix} \right]$$

$$\left[\begin{matrix} n+1 \\ k \end{matrix} \right] = n \left[\begin{matrix} n \\ k \end{matrix} \right] + \left[\begin{matrix} n \\ k-1 \end{matrix} \right], \left[\begin{matrix} n+1 \\ 2 \end{matrix} \right] = n! H_n$$

$$x^{\underline{n}} = x(x-1)\dots(x-n+1) = \sum_k \left[\begin{matrix} n \\ k \end{matrix} \right] (-1)^{n-k} x^k$$

$$x^{\overline{n}} = x(x+1)\dots(x+n-1) = \sum_k \left[\begin{matrix} n \\ k \end{matrix} \right] x^k$$

8.11.6 Stirling Subset Numbers

Divide n elements into k non-empty subsets.

$$\left\{ \begin{matrix} n+1 \\ k \end{matrix} \right\} = k \left\{ \begin{matrix} n \\ k \end{matrix} \right\} + \left\{ \begin{matrix} n \\ k-1 \end{matrix} \right\}$$

$$x^n = \sum_k \left\{ \begin{matrix} n \\ k \end{matrix} \right\} x^{\underline{k}} = \sum_k \left\{ \begin{matrix} n \\ k \end{matrix} \right\} (-1)^{n-k} x^{\overline{k}}$$

$$m! \left\{ \begin{matrix} n \\ m \end{matrix} \right\} = \sum_k \binom{m}{k} k^n (-1)^{m-k}$$

For a fixed k , generating functions :

$$\sum_{n=0}^{\infty} \left\{ \begin{matrix} n \\ k \end{matrix} \right\} x^{n-k} = \prod_{r=1}^k \frac{1}{1-rx}$$

8.11.7 Motzkin Numbers

Draw non-intersecting chords between n points on a circle.

Pick n numbers $k_1, k_2, \dots, k_n \in \{-1, 0, 1\}$ so that $\sum_i^a k_i (1 \leq a \leq n)$ is non-negative and the sum of all numbers is 0.

$$M_{n+1} = M_n + \sum_i^{n-1} M_i M_{n-1-i} = \frac{(2n+3)M_n + 3nM_{n-1}}{n+3}$$

$$M_n = \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} \binom{n}{2k} \text{Catlan}(k)$$

$$M(X) = \frac{1-x-\sqrt{1-2x-3x^2}}{2x^2}$$

8.11.8 Eulerian Numbers

Permutations of the numbers 1 to n in which exactly k elements are greater than the previous element.

$$\langle n \rangle_k = (k+1) \langle n-1 \rangle_k + (n-k) \langle n-1 \rangle_{k-1}$$

$$x^n = \sum_k \langle n \rangle_k \binom{x+k}{n}$$

$$\langle n \rangle_m = \sum_{k=0}^m \binom{n+1}{k} (m+1-k)^n (-1)^k$$

8.11.9 Harmonic Numbers

Sum of the reciprocals of the first n natural numbers.

$$\sum_{k=1}^n H_k = (n+1)H_n - n$$

$$\sum_{k=1}^n k H_k = \frac{n(n+1)}{2} H_n - \frac{n(n-1)}{4}$$

$$\sum_{k=1}^n \binom{k}{m} H_k = \binom{n+1}{m+1} (H_{n+1} - \frac{1}{m+1})$$

8.11.10 Pentagonal Number Theorem

$$\prod_{n=1}^{\infty} (1-x^n) = \sum_{n=-\infty}^{\infty} (-1)^k x^{k(3k-1)/2}$$

$$p(n) = p(n-1) + p(n-2) - p(n-5) - p(n-7) + \dots$$

$$f(n, k) = p(n) - p(n-k) - p(n-2k) + p(n-5k) + p(n-7k) - \dots$$

8.11.11 Bell Numbers

Divide a set that has exactly n elements.

$$B_n = \sum_{k=1}^n \left\{ \begin{matrix} n \\ k \end{matrix} \right\}, \quad B_{n+1} = \sum_{k=0}^n \binom{n}{k} B_k$$

$$B_{p^m+n} \equiv m B_n + B_{n+1} \pmod{p}$$

$$B(x) = \sum_{n=0}^{\infty} \frac{B_n}{n!} x^n = e^{e^x - 1}$$

8.11.12 Bernoulli Numbers

$$B_n = 1 - \sum_{k=0}^{n-1} \binom{n}{k} \frac{B_k}{n-k+1}$$

$$G(x) = \sum_{k=0}^{\infty} \frac{B_k}{k!} x^k = \frac{1}{\sum_{k=0}^{\infty} \frac{x^k}{(k+1)!}}$$

$$\sum_{k=1}^n k^m = \frac{1}{m+1} \sum_{k=0}^m \binom{m+1}{k} B_k n^{m-k+1}$$

8.11.13 Sum of Powers

$$\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}, \quad \sum_{i=1}^n i^3 = \left(\frac{n(n+1)}{2}\right)^2$$

$$\sum_{i=1}^n i^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$

$$\sum_{i=1}^n i^5 = \frac{n^2(n+1)^2(2n^2+2n-1)}{12}$$

8.11.14 Sum of Squares

Denote $r_k(n)$ the ways to form n with k squares. If :

$$n = 2^{a_0} p_1^{2a_1} \dots p_r^{2a_r} q_1 b_1 \dots q_s b_s$$

where $p_i \equiv 3 \pmod{4}$, $q_i \equiv 1 \pmod{4}$, then

$$r_2(n) = \begin{cases} 0 & \text{if any } a_i \text{ is a half-integer} \\ 4 \prod_1^r (b_i + 1) & \text{if all } a_i \text{ are integers} \end{cases}$$

$r_3(n) > 0$ when and only when n is not $4^a(8b+7)$.

8.11.15 Derangement

$$D_1 = 0, D_2 = 1, D_n = n! \left(\frac{1}{0!} - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \dots + \frac{(-1)^n}{n!} \right)$$

$$D_n = (n-1)(D_{n-1} + D_{n-2})$$

8.11.16 Tetrahedron Volume

If U , V , W , u , v , w are lengths of edges of the tetrahedron (first three form a triangle; u opposite to U and so on)

$$V = \frac{\sqrt{4u^2v^2w^2 - \sum_{cyc} u^2(v^2 + w^2 - U^2)^2 + \prod_{cyc} (v^2 + w^2 - U^2)}}{12}$$

9 Appendix

9.1 Calculus table

$$\begin{aligned} \left(\frac{u}{v}\right)' &= \frac{u'v - uv'}{v^2} & (\operatorname{arcsec} x)' &= \frac{1}{x\sqrt{1-x^2}} \\ (a^x)' &= (\ln a)a^x & (\tanh x)' &= \operatorname{sech}^2 x \\ (\tan x)' &= \sec^2 x & (\coth x)' &= -\operatorname{csch}^2 x \\ (\cot x)' &= -\csc^2 x & (\operatorname{sech} x)' &= -\operatorname{sech} x \tanh x \\ (\sec x)' &= \tan x \sec x & (\operatorname{csch} x)' &= -\operatorname{csch} x \coth x \\ (\csc x)' &= -\cot x \csc x & (\operatorname{arcsinh} x)' &= \frac{1}{\sqrt{1+x^2}} \\ (\arcsin x)' &= \frac{1}{\sqrt{1-x^2}} & (\operatorname{arccosh} x)' &= \frac{1}{\sqrt{x^2-1}} \\ (\arccos x)' &= -\frac{1}{\sqrt{1-x^2}} & (\operatorname{arctanh} x)' &= \frac{1}{1-x^2} \\ (\arctan x)' &= \frac{1}{1+x^2} & (\operatorname{arccoth} x)' &= \frac{1}{x^2-1} \\ (\operatorname{arccot} x)' &= -\frac{1}{1+x^2} & (\operatorname{arcsch} x)' &= -\frac{1}{|x|\sqrt{1+x^2}} \\ (\operatorname{arccsc} x)' &= -\frac{1}{x\sqrt{1-x^2}} & (\operatorname{arcsech} x)' &= -\frac{1}{x\sqrt{1-x^2}} \end{aligned}$$

9.1.1 $ax+b$ ($a \neq 0$)

$$\begin{aligned} 1. \int \frac{x}{ax+b} dx &= \frac{1}{a^2} (ax+b - b \ln |ax+b|) + C \\ 2. \int \frac{x^2}{ax+b} dx &= \frac{1}{a^3} \left(\frac{1}{2} (ax+b)^2 - 2b(ax+b) + b^2 \ln |ax+b| \right) + C \\ 3. \int \frac{dx}{x(ax+b)} &= -\frac{1}{b} \ln \left| \frac{ax+b}{x} \right| + C \\ 4. \int \frac{dx}{x^2(ax+b)} &= -\frac{1}{bx} + \frac{a}{b^2} \ln \left| \frac{ax+b}{x} \right| + C \\ 5. \int \frac{x}{(ax+b)^2} dx &= \frac{1}{a^2} \left(\ln |ax+b| + \frac{a}{ax+b} \right) + C \\ 6. \int \frac{x^2}{(ax+b)^2} dx &= \frac{1}{a^3} \left(ax+b - 2b \ln |ax+b| - \frac{b^2}{ax+b} \right) + C \\ 7. \int \frac{dx}{x(ax+b)^2} &= \frac{1}{b(ax+b)} - \frac{1}{b^2} \ln \left| \frac{ax+b}{x} \right| + C \end{aligned}$$

9.1.2 $\sqrt{ax+b}$

$$\begin{aligned} 1. \int \sqrt{ax+b} dx &= \frac{2}{3a} \sqrt{(ax+b)^3} + C \\ 2. \int x \sqrt{ax+b} dx &= \frac{2}{15a^2} (3ax-2b) \sqrt{(ax+b)^3} + C \\ 3. \int x^2 \sqrt{ax+b} dx &= \frac{2}{105a^3} (15a^2x^2 - 12abx + 8b^2) \sqrt{(ax+b)^3} + C \\ 4. \int \frac{x}{\sqrt{ax+b}} dx &= \frac{2}{3a^2} (ax-2b) \sqrt{ax+b} + C \\ 5. \int \frac{x^2}{\sqrt{ax+b}} dx &= \frac{2}{15a^3} (3a^2x^2 - 4abx + 8b^2) \sqrt{ax+b} + C \\ 6. \int \frac{dx}{x\sqrt{ax+b}} &= \begin{cases} \frac{1}{\sqrt{b}} \ln \left| \frac{\sqrt{ax+b}-\sqrt{b}}{\sqrt{ax+b}+\sqrt{b}} \right| + C & (b > 0) \\ \frac{2}{\sqrt{-b}} \arctan \sqrt{\frac{ax+b}{-b}} + C & (b < 0) \end{cases} \\ 7. \int \frac{dx}{x^2\sqrt{ax+b}} &= -\frac{\sqrt{ax+b}}{bx} - \frac{a}{2b} \int \frac{dx}{x\sqrt{ax+b}} \\ 8. \int \frac{\sqrt{ax+b}}{x} dx &= 2\sqrt{ax+b} + b \int \frac{dx}{x\sqrt{ax+b}} \\ 9. \int \frac{\sqrt{ax+b}}{x^2} dx &= -\frac{\sqrt{ax+b}}{x} + \frac{a}{2} \int \frac{dx}{x\sqrt{ax+b}} \end{aligned}$$

9.1.3 $x^2 \pm a^2$

$$\begin{aligned} 1. \int \frac{dx}{x^2+a^2} &= \frac{1}{a} \arctan \frac{x}{a} + C \\ 2. \int \frac{dx}{(x^2+a^2)^n} &= \frac{x}{2(n-1)a^2(x^2+a^2)^{n-1}} + \frac{2n-3}{2(n-1)a^2} \int \frac{dx}{(x^2+a^2)^{n-1}} \\ 3. \int \frac{dx}{x^2-a^2} &= \frac{1}{2a} \ln \left| \frac{x-a}{x+a} \right| + C \end{aligned}$$

9.1.4 ax^2+b ($a > 0$)

$$\begin{aligned} 1. \int \frac{dx}{ax^2+b} &= \begin{cases} \frac{1}{\sqrt{ab}} \arctan \sqrt{\frac{b}{a}} x + C & (b > 0) \\ \frac{1}{2\sqrt{-ab}} \ln \left| \frac{\sqrt{ax^2+b}-\sqrt{-b}}{\sqrt{ax^2+b}+\sqrt{-b}} \right| + C & (b < 0) \end{cases} \\ 2. \int \frac{x}{ax^2+b} dx &= \frac{1}{2a} \ln |ax^2+b| + C \\ 3. \int \frac{x^2}{ax^2+b} dx &= \frac{x}{a} - \frac{b}{a} \int \frac{dx}{ax^2+b} \\ 4. \int \frac{dx}{x(ax^2+b)} &= \frac{1}{2b} \ln \left| \frac{x^2}{ax^2+b} \right| + C \\ 5. \int \frac{dx}{x^2(ax^2+b)} &= -\frac{1}{bx} - \frac{a}{b} \int \frac{dx}{x^2(ax^2+b)} \\ 6. \int \frac{dx}{x^3(ax^2+b)} &= \frac{a}{2b^2} \ln \left| \frac{ax^2+b}{x^2} \right| - \frac{1}{2bx^2} + C \\ 7. \int \frac{dx}{(ax^2+b)^2} &= \frac{x}{2b(ax^2+b)} + \frac{1}{2b} \int \frac{dx}{x^2(ax^2+b)} \end{aligned}$$

9.1.5 ax^2+bx+c ($a > 0$)

$$\begin{aligned} 1. \frac{dx}{ax^2+bx+c} &= \begin{cases} \frac{2}{\sqrt{4ac-b^2}} \arctan \frac{2ax+b}{\sqrt{4ac-b^2}} + C & (b^2 < 4ac) \\ \frac{1}{\sqrt{b^2-4ac}} \ln \left| \frac{2ax+b-\sqrt{b^2-4ac}}{2ax+b+\sqrt{b^2-4ac}} \right| + C & (b^2 > 4ac) \end{cases} \\ 2. \int \frac{x}{ax^2+bx+c} dx &= \frac{1}{2a} \ln |ax^2+bx+c| - \frac{b}{2a} \int \frac{dx}{ax^2+bx+c} \end{aligned}$$

9.1.6 $\sqrt{x^2+a^2}$ ($a > 0$)

$$\begin{aligned} 1. \int \frac{dx}{\sqrt{x^2+a^2}} &= \operatorname{arsh} \frac{x}{a} + C_1 = \ln(x + \sqrt{x^2+a^2}) + C \\ 2. \int \frac{dx}{\sqrt{(x^2+a^2)^3}} &= \frac{x}{a^2\sqrt{x^2+a^2}} + C \\ 3. \int \frac{x}{\sqrt{x^2+a^2}} dx &= \sqrt{x^2+a^2} + C \\ 4. \int \frac{x}{\sqrt{(x^2+a^2)^3}} dx &= -\frac{1}{\sqrt{x^2+a^2}} + C \\ 5. \int \frac{x^2}{\sqrt{x^2+a^2}} dx &= \frac{x}{2} \sqrt{x^2+a^2} - \frac{a^2}{2} \ln(x + \sqrt{x^2+a^2}) + C \\ 6. \int \frac{x^2}{\sqrt{(x^2+a^2)^3}} dx &= -\frac{x}{\sqrt{x^2+a^2}} + \ln(x + \sqrt{x^2+a^2}) + C \\ 7. \int \frac{dx}{x\sqrt{x^2+a^2}} &= \frac{1}{a} \ln \frac{\sqrt{x^2+a^2}-a}{|x|} + C \\ 8. \int \frac{dx}{x^2\sqrt{x^2+a^2}} &= -\frac{\sqrt{x^2+a^2}}{a^2x} + C \\ 9. \int \sqrt{x^2+a^2} dx &= \frac{x}{2} \sqrt{x^2+a^2} + \frac{a^2}{2} \ln(x + \sqrt{x^2+a^2}) + C \\ 10. \int \sqrt{(x^2+a^2)^3} dx &= \frac{x}{8} (2x^2+5a^2) \sqrt{x^2+a^2} + \frac{3}{8} a^4 \ln(x + \sqrt{x^2+a^2}) + C \\ 11. \int x \sqrt{x^2+a^2} dx &= \frac{1}{3} \sqrt{(x^2+a^2)^3} + C \\ 12. \int x^2 \sqrt{x^2+a^2} dx &= \frac{x}{8} (2x^2+a^2) \sqrt{x^2+a^2} - \frac{a^4}{8} \ln(x + \sqrt{x^2+a^2}) + C \\ 13. \int \frac{\sqrt{x^2+a^2}}{x} dx &= \sqrt{x^2+a^2} + a \ln \frac{\sqrt{x^2+a^2}-a}{|x|} + C \\ 14. \int \frac{\sqrt{x^2+a^2}}{x^2} dx &= -\frac{\sqrt{x^2+a^2}}{x} + \ln(x + \sqrt{x^2+a^2}) + C \end{aligned}$$

9.1.7 $\sqrt{x^2-a^2}$ ($a > 0$)

$$\begin{aligned} 1. \int \frac{dx}{\sqrt{x^2-a^2}} &= \frac{x}{|x|} \operatorname{arch} \frac{|x|}{a} + C_1 = \ln |x + \sqrt{x^2-a^2}| + C \\ 2. \int \frac{dx}{\sqrt{(x^2-a^2)^3}} &= -\frac{x}{a^2\sqrt{x^2-a^2}} + C \\ 3. \int \frac{x}{\sqrt{x^2-a^2}} dx &= \sqrt{x^2-a^2} + C \\ 4. \int \frac{x}{\sqrt{(x^2-a^2)^3}} dx &= -\frac{1}{\sqrt{x^2-a^2}} + C \\ 5. \int \frac{x^2}{\sqrt{x^2-a^2}} dx &= \frac{x}{2} \sqrt{x^2-a^2} + \frac{a^2}{2} \ln |x + \sqrt{x^2-a^2}| + C \\ 6. \int \frac{x^2}{\sqrt{(x^2-a^2)^3}} dx &= -\frac{x}{\sqrt{x^2-a^2}} + \ln |x + \sqrt{x^2-a^2}| + C \\ 7. \int \frac{dx}{x\sqrt{x^2-a^2}} &= \frac{1}{a} \operatorname{arccos} \frac{a}{|x|} + C \\ 8. \int \frac{dx}{x^2\sqrt{x^2-a^2}} &= \frac{\sqrt{x^2-a^2}}{a^2x} + C \\ 9. \int \sqrt{x^2-a^2} dx &= \frac{x}{2} \sqrt{x^2-a^2} - \frac{a^2}{2} \ln |x + \sqrt{x^2-a^2}| + C \\ 10. \int \sqrt{(x^2-a^2)^3} dx &= \frac{x}{8} (2x^2-5a^2) \sqrt{x^2-a^2} + \frac{3}{8} a^4 \ln |x + \sqrt{x^2-a^2}| + C \\ 11. \int x \sqrt{x^2-a^2} dx &= \frac{1}{3} \sqrt{(x^2-a^2)^3} + C \\ 12. \int x^2 \sqrt{x^2-a^2} dx &= \frac{x}{8} (2x^2-a^2) \sqrt{x^2-a^2} - \frac{a^4}{8} \ln |x + \sqrt{x^2-a^2}| + C \\ 13. \int \frac{\sqrt{x^2-a^2}}{x} dx &= \sqrt{x^2-a^2} - a \operatorname{arccos} \frac{a}{|x|} + C \\ 14. \int \frac{\sqrt{x^2-a^2}}{x^2} dx &= -\frac{\sqrt{x^2-a^2}}{x} + \ln |x + \sqrt{x^2-a^2}| + C \end{aligned}$$

9.1.8 $\sqrt{a^2-x^2}$ ($a > 0$)

$$\begin{aligned} 1. \int \frac{dx}{\sqrt{a^2-x^2}} &= \arcsin \frac{x}{a} + C \\ 2. \int \frac{dx}{\sqrt{(a^2-x^2)^3}} &= \frac{x}{a^2\sqrt{a^2-x^2}} + C \\ 3. \int \frac{x}{\sqrt{a^2-x^2}} dx &= -\sqrt{a^2-x^2} + C \\ 4. \int \frac{x}{\sqrt{(a^2-x^2)^3}} dx &= \frac{1}{\sqrt{a^2-x^2}} + C \\ 5. \int \frac{x^2}{\sqrt{a^2-x^2}} dx &= -\frac{x}{2} \sqrt{a^2-x^2} + \frac{a^2}{2} \arcsin \frac{x}{a} + C \\ 6. \int \frac{x^2}{\sqrt{(a^2-x^2)^3}} dx &= \frac{x}{\sqrt{a^2-x^2}} - \arcsin \frac{x}{a} + C \\ 7. \int \frac{dx}{x\sqrt{a^2-x^2}} &= \frac{1}{a} \ln \frac{a-\sqrt{a^2-x^2}}{|x|} + C \\ 8. \int \frac{dx}{x^2\sqrt{a^2-x^2}} &= -\frac{\sqrt{a^2-x^2}}{a^2x} + C \\ 9. \int \sqrt{a^2-x^2} dx &= \frac{x}{2} \sqrt{a^2-x^2} + \frac{a^2}{2} \arcsin \frac{x}{a} + C \\ 10. \int \sqrt{(a^2-x^2)^3} dx &= \frac{x}{8} (5a^2-2x^2) \sqrt{a^2-x^2} + \frac{3}{8} a^4 \arcsin \frac{x}{a} + C \\ 11. \int x \sqrt{a^2-x^2} dx &= -\frac{1}{3} \sqrt{(a^2-x^2)^3} + C \\ 12. \int x^2 \sqrt{a^2-x^2} dx &= \frac{x}{8} (2x^2-a^2) \sqrt{a^2-x^2} + \frac{a^4}{8} \arcsin \frac{x}{a} + C \\ 13. \int \frac{\sqrt{a^2-x^2}}{x} dx &= \sqrt{a^2-x^2} + a \ln \frac{a-\sqrt{a^2-x^2}}{|x|} + C \\ 14. \int \frac{\sqrt{a^2-x^2}}{x^2} dx &= -\frac{\sqrt{a^2-x^2}}{x} - \arcsin \frac{x}{a} + C \end{aligned}$$

9.1.9 $\sqrt{\pm ax^2+bx+c}$ ($a > 0$)

$$\begin{aligned} 1. \int \frac{dx}{\sqrt{ax^2+bx+c}} &= \frac{1}{\sqrt{a}} \ln |2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}| + C \\ 2. \int \sqrt{ax^2+bx+c} dx &= \frac{2ax+b}{4a} \sqrt{ax^2+bx+c} + \frac{4ac-b^2}{8\sqrt{a^3}} \ln |2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}| + C \\ 3. \int \frac{x}{\sqrt{ax^2+bx+c}} dx &= \frac{1}{a} \sqrt{ax^2+bx+c} - \frac{b}{2\sqrt{a^3}} \ln |2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}| + C \\ 4. \int \frac{dx}{\sqrt{c+bx-ax^2}} &= -\frac{1}{\sqrt{a}} \arcsin \frac{2ax-b}{\sqrt{b^2+4ac}} + C \\ 5. \int \sqrt{c+bx-ax^2} dx &= \frac{2ax-b}{4a} \sqrt{c+bx-ax^2} + \frac{b^2+4ac}{8\sqrt{a^3}} \arcsin \frac{2ax-b}{\sqrt{b^2+4ac}} + C \\ 6. \int \frac{x}{\sqrt{c+bx-ax^2}} dx &= -\frac{1}{a} \sqrt{c+bx-ax^2} + \frac{b}{2\sqrt{a^3}} \arcsin \frac{2ax-b}{\sqrt{b^2+4ac}} + C \end{aligned}$$

9.1.10 $\sqrt{\pm \frac{x-a}{x-b}}$ & $\sqrt{(x-a)(x-b)}$

- $\int \sqrt{\frac{x-a}{x-b}} dx = (x-b) \sqrt{\frac{x-a}{x-b}} + (b-a) \ln(\sqrt{|x-a|} + \sqrt{|x-b|}) + C$
- $\int \sqrt{\frac{x-a}{b-x}} dx = (x-b) \sqrt{\frac{x-a}{b-x}} + (b-a) \arcsin \sqrt{\frac{x-a}{b-x}} + C$
- $\int \frac{dx}{\sqrt{(x-a)(b-x)}} = 2 \arcsin \sqrt{\frac{x-a}{b-x}} + C \quad (a < b)$
- $\int \sqrt{(x-a)(b-x)} dx = \frac{2x-a-b}{4} \sqrt{(x-a)(b-x)} + \frac{(b-a)^2}{4} \arcsin \sqrt{\frac{x-a}{b-x}} + C \quad (a < b)$

9.1.11 Triangular function

- $\int \tan x dx = -\ln |\cos x| + C$
- $\int \cot x dx = \ln |\sin x| + C$
- $\int \sec x dx = \ln \left| \tan \left(\frac{\pi}{4} + \frac{x}{2} \right) \right| + C = \ln |\sec x + \tan x| + C$
- $\int \csc x dx = \ln \left| \tan \frac{x}{2} \right| + C = \ln |\csc x - \cot x| + C$
- $\int \sec^2 x dx = \tan x + C$
- $\int \csc^2 x dx = -\cot x + C$
- $\int \sec x \tan x dx = \sec x + C$
- $\int \csc x \cot x dx = -\csc x + C$
- $\int \sin^2 x dx = \frac{x}{2} - \frac{1}{4} \sin 2x + C$
- $\int \cos^2 x dx = \frac{x}{2} + \frac{1}{4} \sin 2x + C$
- $\int \sin^n x dx = -\frac{1}{n} \sin^{n-1} x \cos x + \frac{n-1}{n} \int \sin^{n-2} x dx$
- $\int \cos^n x dx = \frac{1}{n} \cos^{n-1} x \sin x + \frac{n-1}{n} \int \cos^{n-2} x dx$
- $\frac{dx}{\sin^n x} = -\frac{1}{n-1} \frac{\cos x}{\sin^{n-1} x} + \frac{n-2}{n-1} \int \frac{dx}{\sin^{n-2} x}$
- $\frac{dx}{\cos^n x} = \frac{1}{n-1} \frac{\sin x}{\cos^{n-1} x} + \frac{n-2}{n-1} \int \frac{dx}{\cos^{n-2} x}$

$$\begin{aligned} & \int \cos^m x \sin^n x dx \\ &= \frac{1}{m+n} \cos^{m-1} x \sin^{n+1} x + \frac{m-1}{m+n} \int \cos^{m-2} x \sin^n x dx \\ &= -\frac{1}{m+n} \cos^{m+1} x \sin^{n-1} x + \frac{n-1}{m+1} \int \cos^m x \sin^{n-2} x dx \end{aligned}$$

- $\int \sin ax \cos bx dx = -\frac{1}{2(a+b)} \cos(a+b)x - \frac{1}{2(a-b)} \cos(a-b)x + C$
- $\int \sin ax \sin bx dx = -\frac{1}{2(a+b)} \sin(a+b)x + \frac{1}{2(a-b)} \sin(a-b)x + C$
- $\int \cos ax \cos bx dx = \frac{1}{2(a+b)} \sin(a+b)x + \frac{1}{2(a-b)} \sin(a-b)x + C$
- $\int \frac{dx}{a+b \sin x} = \begin{cases} \frac{2}{\sqrt{a^2-b^2}} \arctan \frac{a \tan \frac{x}{2} + b}{\sqrt{a^2-b^2}} + C & (a^2 > b^2) \\ \frac{1}{\sqrt{b^2-a^2}} \ln \left| \frac{a \tan \frac{x}{2} + b - \sqrt{b^2-a^2}}{a \tan \frac{x}{2} + b + \sqrt{b^2-a^2}} \right| + C & (a^2 < b^2) \end{cases}$
- $\int \frac{dx}{a+b \cos x} = \begin{cases} \frac{2}{a+b} \sqrt{\frac{a+b}{a-b}} \arctan \left(\sqrt{\frac{a-b}{a+b}} \tan \frac{x}{2} \right) + C & (a^2 > b^2) \\ \frac{1}{a+b} \sqrt{\frac{a+b}{a-b}} \ln \left| \frac{\tan \frac{x}{2} + \sqrt{\frac{a+b}{a-b}}}{\tan \frac{x}{2} - \sqrt{\frac{a+b}{a-b}}} \right| + C & (a^2 < b^2) \end{cases}$
- $\int \frac{dx}{a^2 \cos^2 x + b^2 \sin^2 x} = \frac{1}{ab} \arctan \left(\frac{b}{a} \tan x \right) + C$
- $\int \frac{dx}{a^2 \cos^2 x - b^2 \sin^2 x} = \frac{1}{2ab} \ln \left| \frac{b \tan x + a}{b \tan x - a} \right| + C$
- $\int x \sin ax dx = \frac{1}{a^2} \sin ax - \frac{1}{a} x \cos ax + C$
- $\int x^2 \sin ax dx = -\frac{1}{a^3} \cos ax + \frac{2}{a^2} x \sin ax + \frac{2}{a^3} \cos ax + C$
- $\int x \cos ax dx = \frac{1}{a^2} \cos ax + \frac{1}{a} x \sin ax + C$
- $\int x^2 \cos ax dx = \frac{1}{a^3} \sin ax + \frac{2}{a^2} x \cos ax - \frac{2}{a^3} \sin ax + C$

9.1.12 Inverse triangular function ($a > 0$)

- $\int \arcsin \frac{x}{a} dx = x \arcsin \frac{x}{a} + \sqrt{a^2 - x^2} + C$
- $\int x \arcsin \frac{x}{a} dx = \left(\frac{x^2}{2} - \frac{a^2}{4} \right) \arcsin \frac{x}{a} + \frac{\pi}{4} \sqrt{x^2 - x^2} + C$
- $\int x^2 \arcsin \frac{x}{a} dx = \frac{x^3}{3} \arcsin \frac{x}{a} + \frac{1}{9} (x^2 + 2a^2) \sqrt{a^2 - x^2} + C$
- $\int \arccos \frac{x}{a} dx = x \arccos \frac{x}{a} - \sqrt{a^2 - x^2} + C$
- $\int x \arccos \frac{x}{a} dx = \left(\frac{x^2}{2} - \frac{a^2}{4} \right) \arccos \frac{x}{a} - \frac{\pi}{4} \sqrt{a^2 - x^2} + C$
- $\int x^2 \arccos \frac{x}{a} dx = \frac{x^3}{3} \arccos \frac{x}{a} - \frac{1}{9} (x^2 + 2a^2) \sqrt{a^2 - x^2} + C$
- $\int \arctan \frac{x}{a} dx = x \arctan \frac{x}{a} - \frac{a}{2} \ln(a^2 + x^2) + C$
- $\int x \arctan \frac{x}{a} dx = \frac{1}{2} (a^2 + x^2) \arctan \frac{x}{a} - \frac{a}{2} x + C$
- $\int x^2 \arctan \frac{x}{a} dx = \frac{x^3}{3} \arctan \frac{x}{a} - \frac{a}{6} x^2 + \frac{a^3}{6} \ln(a^2 + x^2) + C$

9.1.13 Exponential function

- $\int a^x dx = \frac{1}{\ln a} a^x + C$
- $\int e^{ax} dx = \frac{1}{a} a^x + C$
- $\int x e^{ax} dx = \frac{1}{a^2} (ax - 1) a^x + C$
- $\int x^n e^{ax} dx = \frac{1}{a} x^n e^{ax} - \frac{n}{a} \int x^{n-1} e^{ax} dx$
- $\int x a^x dx = \frac{x}{\ln a} a^x - \frac{1}{(\ln a)^2} a^x + C$
- $\int x^n a^x dx = \frac{1}{\ln a} x^n a^x - \frac{n}{\ln a} \int x^{n-1} a^x dx$
- $\int e^{ax} \sin bx dx = \frac{1}{a^2 + b^2} e^{ax} (a \sin bx - b \cos bx) + C$
- $\int e^{ax} \cos bx dx = \frac{1}{a^2 + b^2} e^{ax} (b \sin bx + a \cos bx) + C$
- $\int e^{ax} \sin^n bx dx = \frac{1}{a^2 + b^2 n^2} e^{ax} \sin^{n-1} bx (a \sin bx - n b \cos bx) + \frac{n(n-1)b^2}{a^2 + b^2 n^2} \int e^{ax} \sin^{n-2} bx dx$
- $\int e^{ax} \cos^n bx dx = \frac{1}{a^2 + b^2 n^2} e^{ax} \cos^{n-1} bx (a \cos bx + n b \sin bx) + \frac{n(n-1)b^2}{a^2 + b^2 n^2} \int e^{ax} \cos^{n-2} bx dx$

9.1.14 Logarithmic function

- $\int \ln x dx = x \ln x - x + C$
- $\int \frac{dx}{x \ln x} = \ln |\ln x| + C$

- $\int x^n \ln x dx = \frac{1}{n+1} x^{n+1} (\ln x - \frac{1}{n+1}) + C$
- $\int (\ln x)^n dx = x (\ln x)^n - n \int (\ln x)^{n-1} dx$
- $\int x^m (\ln x)^n dx = \frac{1}{m+1} x^{m+1} (\ln x)^n - \frac{n}{m+1} \int x^m (\ln x)^{n-1} dx$

9.2 Regular expression

9.2.1 Special pattern characters

Characters	Description
.	Not newline
\t	Tab (HT)
\n	Newline (LF)
\v	Vertical tab (VT)
\f	Form feed (FF)
\r	Carriage return (CR)
\cletter	Control code
\xhh	ASCII character
\uhhhh	Unicode character
\0	Null
\int	Backreference
\d	Digit
\D	Not digit
\s	Whitespace
\S	Not whitespace
\w	Word (letters, numbers and the underscore)
\W	Not word
\character	Character
[class]	Character class
^[class]	Negated character class

9.2.2 Quantifiers

Characters	Times
*	0 or more
+	1 or more
?	0 or 1
{int}	int
{int,}	int or more
{min,max}	Between min and max

By default, all these quantifiers are greedy (i.e., they take as many characters that meet the condition as possible). This behavior can be overridden to ungreedy (i.e., take as few characters that meet the condition as possible) by adding a question mark (?) after the quantifier.

9.2.3 Groups

Characters	Description
(subpattern)	Group with backreference
(?:subpattern)	Group without backreference

9.2.4 Assertions

Characters	Description
^	Beginning of line
\$	End of line
\b	Word boundary
\B	Not a word boundary
(?=subpattern)	Positive lookahead
(?!subpattern)	Negative lookahead

9.2.5 Alternative

A regular expression can contain multiple alternative patterns simply by separating them with the separator operator (|): The regular expression will match if any of the alternatives match, and as soon as one does.

9.2.6 Character classes

Class	Description
[:alnum:]	Alpha-numerical character
[:alpha:]	Alphabetic character
[:blank:]	Blank character
[:cntrl:]	Control character
[:digit:]	Decimal digit character
[:graph:]	Character with graphical representation
[:lower:]	Lowercase letter
[:print:]	Printable character
[:punct:]	Punctuation mark character
[:space:]	Whitespace character
[:upper:]	Uppercase letter
[:xdigit:]	Hexadecimal digit character
[:d:]	Decimal digit character
[:w:]	Word character
[:s:]	Whitespace character

Please note that the brackets in the class names are additional to those opening and closing the class definition. For example:

[[:alpha:]] is a character class that matches any alphabetic character.
[abc[:digit:]] is a character class that matches a, b, c, or a digit.

[^[:space:]] is a character class that matches any character except a whitespace.