

Luna's Magic Reference

Suzune Nisiyama

July 14, 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
1.1	Vimrc	2
2	Data Structure	2
2.1	KD tree	2
2.2	Splay	2
2.3	Link-cut tree	2
3	Formula	2
3.1	Zellers congruence	2
3.2	Lattice points below segment	2
3.3	Adaptive Simpson's method	3
4	Number theory	3
4.1	Fast power module	3
4.2	Euclidean algorithm	3
4.3	Discrete Fourier transform	3
4.4	Number theoretic transform	3
4.5	Chinese remainder theorem	3
4.6	Linear Recurrence	3
4.7	Baby step giant step algorithm	3
4.8	Miller Rabin primality test	4
4.9	Pollard's Rho algorithm	4
5	Geometry	4
5.1	Point	4
5.2	Line	4
5.3	Circle	4
5.4	Centers of a triangle	5
5.5	Fermat point	5
5.6	Convex hull	5
5.7	Half plane intersection	5
5.8	Minimum circle	5
5.9	Intersection of a polygon and a circle	5
5.10	Union of circles	5
6	Graph	6
6.1	Hopcroft-Karp algorithm	6
6.2	Kuhn-Munkres algorithm	6
6.3	Blossom algorithm	6
6.4	Weighted blossom algorithm	6
6.5	Maximum flow	7
6.6	Minimum cost flow	7
6.7	Stoer Wagner algorithm	8
6.8	DN maximum clique	8
7	Appendix	9
7.1	Calculus table	9

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] <= dmax.data[i]) continue;
24             ret += std::min (1ll * (dmin.data[i] - rhs.data[i]
25                 ) * (dmin.data[i] - rhs.data[i]),
26                 1ll * (dmax.data[i] - rhs.data[i]) * (dmax.
27                     data[i] - rhs.data[i]));
28             // ret += std::max (0, rhs.data[i] - dmax.data[i])
29             // + std::max (0, dmin.data[i] - rhs.data[i]);
30         } return ret; }
31     long long max_dist (const point &rhs, int k) {
32         long long ret = 0;
33         for (int i = 0; i < k; ++i) {
34             int tmp = std::max (std::abs (dmin.data[i] - rhs.
35                 data[i]), std::abs (dmax.data[i] - rhs.data[i]
36                 ));
37             ret += 1ll * tmp * tmp; }
38         // ret += std::max (std::abs (rhs.data[i] - dmax.
39             data[i]) + std::abs (rhs.data[i] - dmin.data[i]));
40     } return ret; } } tree[MAXN * 4];
41 struct result {
42     long long dist; point d; result() {}
43     result (const long long &dist, const point &d) :
44         dist (dist), d (d) {}
45     bool operator > (const result &rhs) const { return
46         dist > rhs.dist || (dist == rhs.dist && d.id >
47             rhs.d.id); }
48     bool operator < (const result &rhs) const { return
49         dist < rhs.dist || (dist == rhs.dist && d.id <
50             rhs.d.id); } };
51 long long sqrdist (const point &a, const point &b) {
52     long long ret = 0;
53     for (int i = 0; i < k; ++i) ret += 1ll * (a.data[i]
54         - b.data[i]) * (a.data[i] - b.data[i]);
55     // for (int i = 0; i < k; ++i) ret += std::abs (a.
56         data[i] - b.data[i]);
57     return ret; }
58 int alloc() { tree[size].l = tree[size].r = -1;
59     return size++; }
60 void build (const int &depth, int &rt, const int &l,
61     const int &r) {
62     if (l > r) return;
63     register int middle = (l + r) >> 1;
64     std::nth_element (p + l, p + middle, p + r + 1, [=]
65         (const point &a, const point &b) { return a.
66             data[depth] < b.data[depth]; });
67     tree[rt] = alloc(); kd_node (p[middle]);
68     if (l == r) return;
69     build ((depth + 1) % k, tree[rt].l, l, middle - 1);
70     build ((depth + 1) % k, tree[rt].r, middle + 1, r);
71     if (!tree[rt].l) tree[rt].merge (tree[tree[rt].l], k
72         );
73     if (!tree[rt].r) tree[rt].merge (tree[tree[rt].r], k
74         );
75     std::priority_queue<result, std::vector<result>, std
76         ::less<result>> heap_l;
77     std::priority_queue<result, std::vector<result>, std
78         ::greater<result>> heap_r;
79     void min_kth (const int &depth, const int &rt, const
80         int &m, const point &d) {
81         result tmp = result (sqrdist (tree[rt].p, d), tree[
82             rt].p);
83         if ((int)heap_l.size() < m) heap_l.push (tmp);
84         else if (tmp < heap_l.top()) {
85             heap_l.pop();
86             heap_l.push (tmp); }
87     }
88     void max_kth (const int &depth, const int &rt, const
89         int &m, const point &d) {
90         result tmp = result (sqrdist (tree[rt].p, d), tree[
91             rt].p);
92         if ((int)heap_r.size() < m) heap_r.push (tmp);
93         else if (tmp > heap_r.top()) {
94             heap_r.pop();
95             heap_r.push (tmp); }
96     }
97     int x = tree[rt].l, y = tree[rt].r;
98     if (!x && !y && sqrdist (d, tree[x].p) < sqrdist (d,
99         tree[y].p)) std::swap (x, y);
100     if (!x && ((int)heap_l.size() < m || tree[x].
101         min_dist (d, k) < heap_l.top().dist))
102         min_kth ((depth + 1) % k, x, m, d);
103     if (!y && ((int)heap_r.size() < m || tree[y].
104         min_dist (d, k) < heap_r.top().dist))
105         max_kth ((depth + 1) % k, y, m, d); }
106 void init (int n, int k) { this->k = k; size = 0;
107     int rt = 0; build (0, rt, 0, n - 1); }
108 result min_kth (const point &d, const int &m) {
109     heap_l = decltype (heap_l) (); min_kth (0, 0, m,
110         d); return heap_l.top (); }
111 result max_kth (const point &d, const int &m) {
112     heap_r = decltype (heap_r) (); max_kth (0, 0, m,
113         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 (!x && ((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     max_kth ((depth + 1) % k, y, m, d); }
70 void min_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_l.size() < m) heap_l.push (tmp);
75     else if (tmp < heap_l.top()) {
76         heap_l.pop();
77         heap_l.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 (!x && ((int)heap_l.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_r.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);
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);

```

```

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 */
3 struct simpson {
4     double area (double (*f) (double), double l, double r
5     ) {
6         double m = 1 + (r - l) / 2;
7         return (f (l) + 4 * f (m) + f (r)) * (r - l) / 6; }
8     double solve (double (*f) (double), double l, double
9     r, double eps, double a) {
10        double m = 1 + (r - l) / 2;
11        double left = area (f, l, m), right = area (f, m, r)
12        ;
13        if (fabs (left + right - a) <= 15 * eps) return left
14        + right + (left + right - a) / 15.0;
15        return solve (f, l, m, eps / 2, left) + solve (f, m,
16        r, eps / 2, right); }
17    double solve (double (*f) (double), double l, double
18    r, double eps) {
19        return solve (f, l, r, eps, area (f, l, r)); } };

```

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 = ans * mul % mod;
5         mul = int (1LL * mul * mul % mod); n >>= 1; }
6     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; 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 < (j >> 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 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) {

```

```

18        int &pA = a[j + k], &pB = a[j + k + (i >> 1)];
19        int A = pA, B = int (1LL * pB * exp[k] % mod);
20        pA = (A + B) % mod;
21        pB = (A - B + mod) % mod; } } }
22    if (f == 1) {
23        int rev = fpm (n, mod - 2, mod);
24        for (int i = 0; i < n; ++i) a[i] = int (1LL * a[i]
25        * rev % mod); } }
26    int crt (int *a, int mod) {
27        static int inv[3][3];
28        for (int i = 0; i < 3; ++i) for (int j = 0; j < 3;
29        ++j)
30            inv[i][j] = (int) inverse (MOD[i], MOD[j]);
31        static int x[3];
32        for (int i = 0; i < 3; ++i) { x[i] = a[i];
33        for (int j = 0; j < i; ++j) {
34            int t = (x[i] - x[j] + MOD[i]) % MOD[i];
35            if (t < 0) t += MOD[i];
36            x[i] = int (1LL * t * inv[j][i] % MOD[i]); } }
37        int sum = 1, ret = x[0] % mod;
38        for (int i = 1; i < 3; ++i) {
39            sum = int (1LL * sum * MOD[i - 1] % mod);
40            ret += int (1LL * x[i] * sum % mod);
41            if (ret >= mod) ret -= mod; }
42        return ret; } };

```

4.5 Chinese remainder theorem

```

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

```

4.6 Linear Recurrence

```

1 /* Linear recurrence : finds the n-th element of a
2 linear recurrence.
3 Usage : vector <int> - first n terms, vector <int> -
4 transition function, calc (k) : the kth term mod
5 MOD.
6 Example : In : {2, 1}, {2, 1} :
7 a1 = 2, a2 = 1, an = 2an-1 + an-2, Out : calc (3) = 5,
8 calc (10007) = 959155122 (MOD 1E9+7) */
9 struct linear_rec {
10     const int LOG = 30, MOD = 1E9 + 7; int n;
11     std::vector <int> first, trans;
12     std::vector <std::vector <int>> bin;
13     std::vector <int> add (std::vector <int> &a, std:::
14     vector <int> &b) {
15         std::vector <int> result (n * 2 + 1, 0);
16         for (int i = 0; i <= n; ++i) for (int j = 0; j <= n;
17         ++j)
18             if ((result[i + j] += 1LL * a[i] * b[j] % MOD) >=
19             MOD) result[i + j] -= MOD;
20         for (int i = 2 * n; i > n; --i) {
21             for (int j = 0; j < n; ++j)
22                 if ((result[i - 1 - j] += 1LL * result[i] * trans[
23                 j] % MOD) >= MOD) result[i - 1 - j] -= MOD;
24             result[i] = 0; }
25         result.erase (result.begin() + n + 1, result.end());
26         return result; }
27     linear_rec (const std::vector <int> &first, const std
28     :vector <int> &trans) : first (first), trans (
29     trans) {
30         n = first.size (); std::vector <int> a (n + 1, 0); a
31         [1] = 1; bin.push_back (a);
32         for (int i = 1; i < LOG; ++i) bin.push_back (add (bin
33         [i - 1], bin[i - 1])); }
34     int solve (int k) {
35         std::vector <int> a (n + 1, 0); a[0] = 1;
36         for (int i = 0; i < LOG; ++i) if (k >> i & 1) a =
37         add (a, bin[i]);
38         int ret = 0;
39         for (int i = 0; i < n; ++i) if ((ret += (long long)
40         a[i + 1] * first[i] % MOD) >= MOD) ret -= MOD;
41         return ret; } };

```

4.7 Baby step giant step algorithm

```

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

```

4.8 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 &prime, const long long &
        base) {
5         long long number = prime - 1;
6         for (; ~number & 1; number >>= 1);
7         long long result = 1ll * (base, number, prime);
8         for (; number != prime - 1 && result != 1 && result
            != prime - 1; number <= 1)
9             result = mul_mod (result, result, prime);
10        return result == prime - 1 || (number & 1) == 1; }
11 bool solve (const long long &number) {
12     if (number < 2) return false;
13     if (number < 4) return true;
14     if (~number & 1) return false;
15     for (int i = 0; i < 12 && BASE[i] < number; ++i) if
        (!check (number, BASE[i])) return false;
16     return true; } };

```

4.9 Pollard's Rho algorithm

```

1 /* Pollard Rho : factorizes an integer. */
2 struct pollard_rho {
3     miller_rabin is_prime;
4     const long long threshold = 13E9;
5     long long factorize (const long long &number, const
        long long &seed) {
6         long long x = rand () % (number - 1) + 1, y = x;
7         for (int head = 1, tail = 2; ; ) {
8             x = mul_mod (x, x, number);
9             x = (x + seed) % number;
10            if (x == y) return number;
11            long long answer = gcd (abs (x - y), number);
12            if (answer > 1 && answer < number) return answer;
13            if (++head == tail) { y = x; tail <= 1; } } }
14 void search (const long long &number, std::vector <
        long long> &divisor) {
15     if (number > 1) {
16         if (is_prime.solve (number)) divisor.push_back (
            number);
17         else {
18             long long factor = number;
19             for (; factor >= number; factor = factorize (
                number, rand () % (number - 1) + 1));
20             search (number / factor, divisor); search (factor,
                divisor); } } }
21 std::vector <long long> solve (const long long &
        number) {
22     std::vector <long long> ans;
23     if (number > threshold) search (number, ans);
24     else {
25         long long rem = number;
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
        : sgn (y) < 0; }
6     point unit () const { double l = sqrt (x * x + y * y)
        ; return point (x / l, y / l); }
7     //counter-clockwise
8     point rot90 () const { return point (-y, x); }
9     //clockwise
10    point _rot90 () const { return point (y, -x); }
11    point rot (cd t) const {
12        double c = cos (t), s = sin (t);
13        return point (x * c - y * s, x * s + y * c); } };
14 bool operator == (cp a, cp b) { return cmp (a.x, b.x)
    == 0 && cmp (a.y, b.y) == 0; }
15 bool operator != (cp a, cp b) { return cmp (a.x, b.x)
    != 0 || cmp (a.y, b.y) != 0; }
16 bool operator < (cp a, cp b) { return cmp (a.x, b.x)
    == 0 ? cmp (a.y, b.y) < 0 : cmp (a.x, b.x) < 0; }
17 point operator - (cp a) { return point (-a.x, -a.y); }
18 point operator + (cp a, cp b) { return point (a.x + b.
    x, a.y + b.y); }
19 point operator - (cp a, cp b) { return point (a.x - b.
    x, a.y - b.y); }
20 point operator * (cp a, cd b) { return point (a.x * b,
    a.y * b); }
21 point operator / (cp a, cd b) { return point (a.x / b,
    a.y / b); }

```

```

22 double dot (cp a, cp b) { return a.x * b.x + a.y * b.y
    ; }
23 double det (cp a, cp b) { return a.x * b.y - a.y * b.x
    ; }
24 double dis2 (cp a, cp b = point ()) { return sqr (a.x
    - b.x) + sqr (a.y - b.y); }
25 double dis (cp a, cp b = point ()) { return sqrt (dis2
    (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
        (s), t (t) {} };
5 bool point_on_segment (cp a, cl b) { return sgn (det (
    a - b.s, b.t - b.s)) == 0 && sgn (dot (b.s - a, b.
    t - a)) <= 0; }
6 bool two_side (cp a, cp b, cl c) { return sgn (det (a
    - c.s, c.t - c.s)) * sgn (det (b - c.s, c.t - c.s))
    < 0; }
7 bool intersect_judgment (cl a, cl b) {
8     if (point_on_segment (b.s, a) || point_on_segment (b.
        t, a)) return true;
9     if (point_on_segment (a.s, b) || point_on_segment (a.
        t, b)) return true;
10    return two_side (a.s, a.t, b) && two_side (b.s, b.t,
        a); }
11 point line_intersect (cl a, cl b) {
12     double s1 = det (a.t - a.s, b.s - a.s), s2 = det (a.t
        - a.s, b.t - a.s);
13     return (b.s * s2 - b.t * s1) / (s2 - s1); }
14 double point_to_line (cp a, cl b) { return fabs (det (
    b.t - b.s, a - b.s)) / dis (b.s, b.t); }
15 point project_to_line (cp a, cl b) { return b.s + (b.t
    - b.s) * (dot (a - b.s, b.t - b.s) / dis2 (b.t, b.
    s)); }
16 double point_to_segment (cp a, cl b) {
17     if (sgn (dot (b.s - a, b.t - b.s)) * dot (b.t - a, b.
        t - b.s)) <= 0) return fabs (det (b.t - b.s, a - b.
        s)) / dis (b.s, b.t);
18     return std::min (dis (a, b.s), dis (a, b.t)); }
19 bool in_polygon (cp p, const std::vector <point> &po)
    {
20     int n = (int) po.size (), counter = 0;
21     for (int i = 0; i < n; ++i) {
22         point a = po[i], b = po[(i + 1) % n];
23         //Modify the next line if necessary.
24         if (point_on_segment (p, line (a, b))) return true;
25         int x = sgn (det (p - a, b - a)), y = sgn (a.y - p.y)
            , z = sgn (b.y - p.y);
26         if (x > 0 && y <= 0 && z > 0) counter++;
27         if (x < 0 && z <= 0 && y > 0) counter--; }
28     return counter != 0; }
29 double polygon_area (const std::vector <point> &a) {
30     double ans = 0.0;
31     for (int i = 0; i < (int) a.size (); ++i) ans += det
        (a[i], a[(i + 1) % a.size ()]) / 2.0;
32     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) :
        c (c), r (r) {} };
5 bool operator == (cc a, cc b) { return a.c == b.c &&
    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)
    , b.r) <= 0; }
8 circle make_circle (cp a, cp b) { return circle ((a +
    b) / 2, dis (a, b) / 2); }
9 circle make_circle (cp a, cp b, cp c) { point p =
    circumcenter (a, b, c); return circle (p, dis (p,
    a)); }
10 //In the order of the line vector.
11 std::vector <point> line_circle_intersect (cl a, cc b)
    {
12     if (cmp (point_to_line (b.c, a), b.r) > 0) return std
        ::vector <point> ();
13     double x = sqrt (sqr (b.r) - sqr (point_to_line (b.c,
        a)));
14     return std::vector <point> ({project_to_line (b.c, a)
        + (a.s - a.t).unit () * x, project_to_line (b.c,
        a) - (a.s - a.t).unit () * x}); }
15 double circle_intersect_area (cc a, cc b) {
16     double d = dis (a.c, b.c);
17     if (sgn (d - (a.r + b.r)) >= 0) return 0;
18     if (sgn (d - abs (a.r - b.r)) <= 0) {
19         double r = std::min (a.r, b.r); return r * r * PI; }
20     double x = (d * d + a.r * a.r - b.r * b.r) / (2 * d),
        t1 = acos (min (1., max (-1., x / a.r))), t2 =
        acos (min (1., max (-1., (d - x) / b.r)));
21     return a.r * a.r * t1 + b.r * b.r * t2 - d * a.r *
        sin (t1); }
22 //Counter-clockwise with respect of vector OaOb.
23 std::vector <point> circle_intersect (cc a, cc b) {
24     if (a.c == b.c || cmp (dis (a.c, b.c), a.r + b.r) > 0
        || cmp (dis (a.c, b.c), std::abs (a.r - b.r)) <
        0) return std::vector <point> ();
25     point r = (b.c - a.c).unit ();
26     double d = dis (a.c, b.c);
27     double x = ((sqr (a.r) - sqr (b.r)) / d + d) / 2;
28     double h = sqrt (sqr (a.r) - sqr (x));
29     if (sgn (h) == 0) return std::vector <point> ({a.c +
        r * x});
30     return std::vector <point> ({a.c + r * x - r.rot90 ()
        * h, a.c + r * x + r.rot90 () * h}); }
31 //Counter-clockwise with respect of point a.

```



```

32 std::vector<point> tangent (cp a, cc b) { circle p =
    make_circle (a, b.c); return circle_intersect (p,
    b); }
33 //Counter-clockwise with respect of point Oa.
34 std::vector<line> extangent (cc a, cc b) {
35     std::vector<line> ret;
36     if (cmp (dis (a.c, b.c), std::abs (a.r - b.r)) <= 0)
37         return ret;
38     if (sgn (a.r - b.r) == 0) {
39         point dir = b.c - a.c; dir = (dir * a.r / dis (dir))
40             .rot90 ();
39         ret.push_back (line (a.c - dir, b.c - dir));
40         ret.push_back (line (a.c + dir, b.c + dir));
41     } else {
42         point p = (b.c * a.r - a.c * b.r) / (a.r - b.r);
43         std::vector pp = tangent (p, a), qq = tangent (p, b);
44         if (pp.size () == 2 && qq.size () == 2) {
45             if (cmp (a.r, b.r) < 0) std::swap (pp[0], pp[1]),
46                 std::swap (qq[0], qq[1]);
46             ret.push_back (line (pp[0], qq[0]));
47             ret.push_back (line (pp[1], qq[1])); } }
48     return ret; }
49 //Counter-clockwise with respect of point Oa.
50 std::vector<line> intangent (cc cl, cc c2) {
51     point p = (b.c * a.r + a.c * b.r) / (a.r + b.r);
52     std::vector pp = tangent (p, a), qq = tangent (p, b);
53     if (pp.size () == 2 && qq.size () == 2) {
54         ret.push_back (line (pp[0], qq[0]));
55         ret.push_back (line (pp[1], qq[1])); }
56     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);
6     return a + point (det (s, point (p.y, q.y)), det (
7         point (p.x, q.x), s)) / det (p, q); }
7 point orthocenter (cp a, cp b, cp c) { return a + b +
    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;
4     double ab = dis (a, b), bc = dis (b, c), ca = dis (c,
5         a);
5     double cosa = dot (b - a, c - a) / ab / ca;
6     double cosb = dot (a - b, c - b) / ab / bc;
7     double cosc = dot (b - c, a - c) / ca / bc;
8     double sq3 = PI / 3.0; point mid;
9     if (sgn (cosa + 0.5) < 0) mid = a;
10    else if (sgn (cosb + 0.5) < 0) mid = b;
11    else if (sgn (cosc + 0.5) < 0) mid = c;
12    else if (sgn (det (b - a, c - a) < 0); mid =
        line_intersect (line (a, b + (c - b).rot (sq3)),
        line (b, c + (a - c).rot (sq3)));
13    else mid = line_intersect (line (a, c + (b - c).rot (
        sq3)), line (c, b + (a - b).rot (sq3)));
14    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
3     - a, c - a)) >= 0; }
3 std::vector<point> convex_hull (std::vector<point> a
4     ) {
4     int cnt = 0; std::sort (a.begin (), a.end ());
5     std::vector<point> ret (a.size (), point ());
6     for (int i = 0; i < (int) a.size (); ++i) {
7         while (cnt > 1 && turn_left (ret[cnt - 2], a[i], ret
8             [cnt - 1])) --cnt;
8         ret[cnt++] = a[i];
9         int fixed = cnt;
10        for (int i = (int) a.size () - 1; i >= 0; --i) {
11            while (cnt > fixed && turn_left (ret[cnt - 2], a[i],
12                ret[cnt - 1])) --cnt;
12            ret[cnt++] = a[i]; }
13        return std::vector (ret.begin (), ret.begin () + cnt
14            - 1); }

```

5.7 Half plane intersection

```

1 /* Online half plane intersection : complexity O(n)
2    each operation. */
3 std::vector<point> cut (const std::vector<point> &c,
4     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 (
9             line_intersect (p, line (c[i], c[j]))); }
9     return ret; }
10 // Offline half plane intersection : complexity
11    O(n log n). */
11 bool turn_left (cl l, cp p) { return turn_left (l.s, l
12     .t, p); }

```

```

12 int cmp (cp a, cp b) { return a.dim () != b.dim () ? (
13     a.dim () < b.dim () ? -1 : 1) : -sgn (det (a, b)); }
13 std::vector<point> half_plane_intersect (std::vector<
14     line> h) {
14     typedef std::pair<point, line> polar;
15     std::vector<polar> g; g.resize (h.size ());
16     for (int i = 0; i < (int) h.size (); ++i) g[i] = std
17         ::make_pair (h[i].t - h[i].s, h[i]);
17     sort (g.begin (), g.end (), [&] (const polar &a,
18         const polar &b) {
18         if (cmp (a.first, b.first) == 0) return sgn (det (a.
19             second.t - a.second.s, b.second.t - a.second.s))
20             < 0;
19         else return cmp (a.first, b.first) < 0; });
20     h.resize (std::unique (g.begin (), g.end (), [&] (
21         const polar &a, const polar &b) { return cmp (a.
22             first, b.first) == 0 }) - g.begin ());
21     for (int i = 0; i < (int) h.size (); ++i) h[i] = g[i]
22         .second;
22     int fore = 0, rear = -1; std::vector<line> ret (h.
23         size (), line ());
23     for (int i = 0; i < (int) h.size (); ++i) {
24         while (fore < rear && !turn_left (h[i],
25             line_intersect (ret[rear - 1], ret[rear]))) --
26             rear;
25         while (fore < rear && !turn_left (h[i],
26             line_intersect (ret[fore], ret[fore + 1]))) ++
27             fore;
26         ret.push_back (++rear) = h[i]; }
27     while (rear - fore > 1 && !turn_left (ret[fore],
28         line_intersect (ret[rear - 1], ret[rear]))) --
29         rear;
28     while (rear - fore > 1 && !turn_left (ret[rear],
29         line_intersect (ret[fore], ret[fore + 1]))) ++
30         fore;
29     if (rear - fore < 2) return std::vector<point> ();
30     std::vector<point> ans; ans.resize (rear + 1);
31     for (int i = 0; i < rear + 1; ++i) ans[i] =
32         line_intersect (ret[i], ret[(i + 1) % (rear + 1)
33             ]);
32     return ans; }

```

5.8 Minimum circle

```

1 circle minimum_circle (std::vector<point> p) {
2     circle ret; std::random_shuffle (p.begin (), p.end ()
3         );
3     for (int i = 0; i < (int) p.size (); ++i) if (!
4         in_circle (p[i], ret)) {
4         ret = circle (p[i], 0); for (int j = 0; j < i; ++j)
5             if (!in_circle (p[j], ret)) {
5             ret = make_circle (p[j], p[i]); for (int k = 0; k <
6                 j; ++k)
6                 if (!in_circle (p[k], ret)) ret = make_circle (p[i]
7                     ], p[j], p[k]); } }
7     return ret; }

```

5.9 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 *
4             r);
4         return r * r * acos (c) / 2.0; }
5     double area (cp a, cp b, const double &r) {
6         double dA = dot (a, a), dB = dot (b, b), dC =
7             point_to_segment (point (a, b),
8             line (a, b));
7         if (sgn (dA - r * r) <= 0 && sgn (dB - r * r) <= 0)
8             return det (a, b) / 2.0;
8         point tA = a.unit () * r, tB = b.unit () * r;
9         if (sgn (dC - r) > 0) return sector_area (tA, tB, r)
10            ;
10        std::vector<point> ret = line_circle_intersect (
11            line (a, b), circle (point (a, b), r));
11        if (sgn (dA - r * r) > 0 && sgn (dB - r * r) > 0)
12            return sector_area (tA, ret[0], r) + det (ret[0],
13                ret[1]) / 2.0 + sector_area (ret[1], tB, r);
13        if (sgn (dA - r * r) > 0) return det (ret[0], b) /
14            2.0 + sector_area (tA, ret[0], r);
14        else return det (a, ret[1]) / 2.0 + sector_area (ret
15            [1], tB, r); }
15 double solve (const std::vector<point> &p, cc c) {
16     double ret = 0.0;
17     for (int i = 0; i < (int) p.size (); ++i) {
18         int s = sgn (det (p[i] - c.c, p[(i + 1) % p.size ()]
19             ) - c.c);
19         if (s > 0) ret += area (p[i] - c.c, p[(i + 1) % p.
20             size ()] - c.c, c.r);
20         else ret -= area (p[(i + 1) % p.size ()] - c.c, p[i]
21             ] - c.c, c.r); }
21     return std::abs (ret); } }

```

5.10 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 =
6             0) : p(p), ang(ang), delta(delta) {}
6         bool operator < (const event &a) { return ang < a.
7             ang; }
7     };
8     void addevent (cc a, cc b, std::vector<event> &evt,
9         int &cnt) {
9         double d2 = dis2 (a.c, b.c), d_ratio = ((a.r - b.r)
10             * (a.r + b.r) / d2 + 1) / 2;
10        p_ratio = sqrt (std::max (0., -(d2 - sqr (a.r - b.r)
11            ) * (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
12            * d_ratio + p * p_ratio, q1 = a.c + d * d_ratio
13            - p * p_ratio;

```



```

22 int get_pr (int b, int xr){
23     int pr = std::find (flower[b].begin (), flower[b].
24         end (), xr) - flower[b].begin ();
25     if (pr % 2 == 1) { std::reverse (flower[b].begin ()
26         + 1, flower[b].end ()); return (int) flower[b].
27         size () - pr; }
28     else return pr; }
29 void set_match (int u, int v){
30     match[u] = g[u][v].v; if (u > n) {
31         edge e = g[u][v]; int xr = flower_from[u][e.u], pr
32         = get_pr (u, xr);
33         for (int i = 0; i < pr; ++i) set_match (flower[u][i
34             ], flower[u][i + 1]);
35         set_match (xr, v); std::rotate (flower[u].begin (),
36             flower[u].begin () + pr, flower[u].end ()); }
37 void augment (int u, int v) {
38     for (; ) {
39         int xnv = st[match[u]]; set_match (u, v);
40         if (!xnv) return; set_match (xnv, st[pa[xnv]]);
41         u = st[pa[xnv]], v = xnv; } }
42 int get_lca (int u, int v){
43     static int t = 0;
44     for (++t; u || v; std::swap (u, v)) {
45         if (u == 0) continue; if (vis[u] == t) return u;
46         vis[u] = t; u = st[match[u]]; if (u) u = st[pa[u]];
47     }
48     return 0; }
49 void add_blossom (int u, int lca, int v) {
50     int b = n + 1; while (b <= n_x && st[b]) ++b;
51     if (b > n_x) ++n_x;
52     lab[b] = 0, S[b] = 0;
53     match[b] = match[lca]; flower[b].clear ();
54     flower[b].push_back (lca);
55     for (int x = u, y, x != lca; x = st[pa[y]]) {
56         flower[b].push_back (x); flower[b].push_back (y =
57             st[match[x]]); q.push (y); }
58     std::reverse (flower[b].begin () + 1, flower[b].end
59         ());
60     for (int x = v, y, x != lca; x = st[pa[y]]) {
61         flower[b].push_back (x); flower[b].push_back (y =
62             st[match[x]]); q.push (y); }
63     set_st (b, b);
64     for (int x = 1; x <= n_x; ++x) g[b][x].w = g[x][b].w
65         = 0;
66     for (int x = 1; x <= n; ++x) flower_from[b][x] = 0;
67     for (size_t i = 0; i < flower[b].size (); ++i) {
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     {
74         int xs = flower[b][i]; S[xs] = -1, set_slack(xs); }
75     st[b] = 0; }
76 bool on_found_edge (const edge &e) {
77     int u = st[e.u], v = st[e.v];
78     if (S[v] == -1) {
79         pa[v] = e.u, S[v] = 1; int nu = st[match[v]];
80         slack[v] = slack[nu] = 0; S[nu] = 0, q.push(nu);
81     } else if (S[v] == 0) {
82         int lca = get_lca(u, v);
83         if (!lca) return augment(u, v), augment(v, u), true
84         ;
85         else add_blossom(u, lca, v); }
86     return false; }
87 bool matching () {
88     memset (S + 1, -1, sizeof (int) * n_x);
89     memset (slack + 1, 0, sizeof (int) * n_x);
90     q = std::queue<int> ();
91     for (int x = 1; x <= n_x; ++x) if (st[x] == x && !
92         match[x]) pa[x] = 0, S[x] = 0, q.push (x);
93     if (q.empty ()) return false;
94     for (; ) {
95         while (q.size ()) {
96             int u = q.front (); q.pop ();
97             if (S[st[u]] == 1) continue;
98             for (int v = 1; v <= n; ++v) if (g[u][v].w > 0 &&
99                 st[v] != st[v]) {
100                 if (e_delta (g[u][v]) == 0) {
101                     if (on_found_edge (g[u][v])) return true;
102                     else update_slack (u, st[v]); } }
103             int d = INF;
104             for (int b = n + 1; b <= n_x; ++b) if (st[b] == b &&
105                 S[b] == 1) d = std::min (d, lab[b] / 2);
106             for (int x = 1; x <= n_x; ++x) if (st[x] == x &&
107                 slack[x] == -1) d = std::min (d, e_delta (g[
108                     slack[x]][x]));
109             else if (S[x] == 0) d = std::min (d, e_delta (g[
110                 slack[x]][x]) / 2); }
111             for (int u = 1; u <= n; ++u) {
112                 if (S[st[u]] == 0) {
113                     if (lab[u] <= d) return 0;
114                     lab[u] -= d;
115                 } else if (S[st[u]] == 1) lab[u] += d; }
116             for (int b = n + 1; b <= n_x; ++b)
117             if (st[b] == b) {
118                 if (S[st[b]] == 0) lab[b] += d * 2;
119                 else if (S[st[b]] == 1) lab[b] -= d * 2; }
120             q = std::queue<int> ();
121             for (int x = 1; x <= n_x; ++x)

```

```

113     if (st[x] == x && slack[x] && st[slack[x]] != x &&
114         e_delta (g[slack[x]][x]) == 0)
115         if (on_found_edge (g[slack[x]][x])) return true;
116     for (int b = n + 1; b <= n_x; ++b) if (st[b] == b
117         && 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].
124         clear();
125     int w_max = 0;
126     for (int u = 1; u <= n; ++u) for (int v = 1; v <= n;
127         ++v) {
128         flower_from[u][v] = (u == v ? u : 0); w_max = std::
129             max (w_max, g[u][v].w); }
130     for (int u = 1; u <= n; ++u) lab[u] = w_max;
131     while (matching ()) ++n_matches;
132     for (int u = 1; u <= n; ++u) if (match[u] && match[u]
133         < u) tot_weight += g[u][match[u]].w;
134     return std::make_pair (tot_weight, n_matches); }
135 void init () { for (int u = 1; u <= n; ++u) for (int
136     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[MAXN], next[MAXN], flow[
6             MAXN];
7         void clear (int n) { size = 0; std::fill (begin,
8             begin + n, -1); }
9         flow_edge_list (int n = MAXN) { clear (n); }
10         void add_edge (int u, int v, int f) {
11             dest[size] = v; next[size] = begin[u]; flow[size] =
12                 f; begin[u] = size++;
13             dest[size] = u; next[size] = begin[v]; flow[size] =
14                 0; begin[v] = size++; } }
15     int pre[MAXN], d[MAXN], gap[MAXN], cur[MAXN];
16     int solve (flow_edge_list &e, int n, int s, int t) {
17         for (int i = 0; i < n; ++i) { pre[i] = d[i] = gap[i]
18             = 0; cur[i] = e.begin[i]; }
19         gap[0] = n; int u = pre[s] = s, v, maxflow = 0;
20         while (d[s] < n) {
21             v = n; for (int i = cur[u]; ~i; i = e.next[i])
22             if (e.flow[i] && d[u] == d[e.dest[i]] + 1) {
23                 v = e.dest[i]; cur[u] = i; break; }
24             if (v < n) {
25                 pre[v] = u; u = v;
26                 if (v == t) {
27                     int dflow = INF, p = t; u = s;
28                     while (p != s) { p = pre[p]; dflow = std::min (
29                         dflow, e.flow[cur[p]]); }
30                     maxflow += dflow; p = t;
31                     while (p != s) { p = pre[p]; e.flow[cur[p]] -=
32                         dflow; e.flow[cur[p] ^ 1] += dflow; } }
33             } else {
34                 int mindist = n + 1;
35                 for (int i = e.begin[u]; ~i; i = e.next[i])
36                 if (e.flow[i] && mindist > d[e.dest[i]]) {
37                     mindist = d[e.dest[i]]; cur[u] = i; }
38                 if (!--gap[d[u]]) return maxflow;
39                 gap[d[u] = mindist + 1]++; u = pre[u]; } }
40             return maxflow; } }
41 /* Dense graph maximum flow : dinic. */
42 template<int MAXN = 1000, int MAXM = 100000>
43 struct dinic {
44     struct flow_edge_list {
45         int size, begin[MAXN], dest[MAXN], next[MAXN], flow[
46             MAXN];
47         void clear (int n) { size = 0; std::fill (begin,
48             begin + n, -1); }
49         flow_edge_list (int n = MAXN) { clear (n); }
50         void add_edge (int u, int v, int f) {
51             dest[size] = v; next[size] = begin[u]; flow[size] =
52                 f; begin[u] = size++;
53             dest[size] = u; next[size] = begin[v]; flow[size] =
54                 0; begin[v] = size++; } }
55     int n, s, t, d[MAXN], w[MAXN], q[MAXN];
56     int bfs (flow_edge_list &e) {
57         std::fill (d, d + n, -1);
58         int l, r; q[l = r = 0] = s, d[s] = 0;
59         for (; l <= r; l++)
60         for (int k = e.begin[q[l]]; ~k; k = e.next[k])
61         if (!d[e.dest[k]] && e.flow[k] > 0) d[e.dest[k]]
62             = d[q[l]] + 1, q[++r] = e.dest[k];
63         return ~d[t] ? 1 : 0; }
64     int dfs (flow_edge_list &e, int u, int ext) {
65         if (u == t) return ext; int k = w[u], ret = 0;
66         for (; ~k; k = e.next[k], w[u] = k) {
67             if (ext == 0) break;
68             if (d[e.dest[k]] == d[u] + 1 && e.flow[k] > 0) {
69                 int flow = dfs (e, e.dest[k], std::min (e.flow[k],
70                     ext));
71                 if (flow > 0) {
72                     e.flow[k] -= flow, e.flow[k ^ 1] += flow;
73                     ret += flow, ext -= flow; } }
74             if (!k) d[u] = -1; return ret; } }
75     int solve (flow_edge_list &e, int n_, int s_, int t_)
76     {
77         int ans = 0; n = n_; s = s_; dinic::t = t_;
78         while (bfs (e)) {
79             for (int i = 0; i < n; ++i) w[i] = e.begin[i];
80             ans += dfs (e, s, INF); }
81         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[
6             MAXM], flow[MAXM];
7         void clear (int n) { size = 0; std::fill (begin,
8             begin + n, -1); }
9         cost_flow_edge_list (int n = MAXN) { clear (n); }
10        void add_edge (int u, int v, int c, int f) {
11            dest[size] = v; next[size] = begin[u]; cost[size] =
12            c; flow[size] = f; begin[u] = size++;
13            dest[size] = u; next[size] = begin[v]; cost[size] =
14            -c; flow[size] = 0; begin[v] = size++; }
15        int n, s, t, prev[MAXN], dist[MAXN], occur[MAXN];
16        bool augment (cost_flow_edge_list &e) {
17            std::vector <int> queue;
18            std::fill (dist, dist + n, INF); std::fill (occur,
19                occur + n, 0);
20            dist[s] = 0; occur[s] = true; queue.push_back (s);
21            for (int head = 0; head < (int)queue.size(); ++head) {
22                int x = queue[head];
23                for (int i = e.begin[x]; ~i; i = e.next[i]) {
24                    int y = e.dest[i];
25                    if (e.flow[i] && dist[y] > dist[x] + e.cost[i]) {
26                        dist[y] = dist[x] + e.cost[i]; prev[y] = i;
27                        if (!occur[y]) {
28                            occur[y] = true; queue.push_back (y); } }
29                occur[x] = false;
30                return dist[t] < INF; }
31            std::pair <int, int> solve (cost_flow_edge_list &e,
32                int n_, int s_, int t_) {
33                n = n_; s = s_; t = t_; std::pair <int, int> ans =
34                std::make_pair (0, 0);
35                while (augment (e)) {
36                    int num = INF;
37                    for (int i = t; i != s; i = e.dest[prev[i] ^ 1]) {
38                        num = std::min (num, e.flow[prev[i]]); }
39                    ans.first += num;
40                    for (int i = t; i != s; i = e.dest[prev[i] ^ 1]) {
41                        e.flow[prev[i]] -= num; e.flow[prev[i] ^ 1] += num
42                        ans.second += num * e.cost[prev[i]]; } }
43                return ans; }
44        }
45        /* Dense graph minimum cost flow : zkw. */
46        template <int MAXN = 1000, int MAXM = 100000>
47        struct zkw_flow {
48            struct cost_flow_edge_list {
49                int size, begin[MAXN], dest[MAXM], next[MAXM], cost[
50                    MAXM], flow[MAXM];
51                void clear (int n) { size = 0; std::fill (begin,
52                    begin + n, -1); }
53                cost_flow_edge_list (int n = MAXN) { clear (n); }
54                void add_edge (int u, int v, int c, int f) {
55                    dest[size] = v; next[size] = begin[u]; cost[size] =
56                    c; flow[size] = f; begin[u] = size++;
57                    dest[size] = u; next[size] = begin[v]; cost[size] =
58                    -c; flow[size] = 0; begin[v] = size++; }
59                int n, s, t, tf, tc, dis[MAXN], slack[MAXN], visit[
60                    MAXN];
61                int modlable() {
62                    int delta = INF;
63                    for (int i = 0; i < n; i++) {
64                        if (!visit[i] && slack[i] < delta) delta = slack[i];
65                        slack[i] = INF; }
66                    if (delta == INF) return 1;
67                    for (int i = 0; i < n; i++) if (visit[i]) dis[i] +=
68                        delta;
69                    return 0; }
70                int dfs (cost_flow_edge_list &e, int x, int flow) {
71                    if (x == t) { tf += flow; tc += flow * (dis[s] - dis
72                        [t]); return flow; }
73                    visit[x] = 1; int left = flow;
74                    for (int i = e.begin[x]; ~i; i = e.next[i])
75                    if (e.flow[i] > 0 && !visit[e.dest[i]]) {
76                        int y = e.dest[i];
77                        if (dis[y] + e.cost[i] == dis[x]) {
78                            int delta = dfs (e, y, std::min (left, e.flow[i])
79                                );
80                            e.flow[i] -= delta; e.flow[i ^ 1] += delta; left
81                                -= delta;
82                            if (!left) { visit[x] = false; return flow; }
83                            else {
84                                slack[y] = std::min (slack[y], dis[y] + e.cost[i]
85                                    - dis[x]); }
86                            return flow - left; }
87                    }
88                std::pair <int, int> solve (cost_flow_edge_list &e,
89                    int n_, int s_, int t_) {
90                    n = n_; s = s_; t = t_; tf = tc = 0;
91                    std::fill (dis + 1, dis + t + 1, 0);
92                    do { do {
93                        std::fill (visit + 1, visit + t + 1, 0);
94                        } while (dfs (e, s, INF)); } while (!modlable ());
95                    return std::make_pair (tf, tc);
96                }
97            };
98        };

```

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));

```

```

13        int i, j, k, mincut, maxc;
14        for (i = 1; i <= n; i++) {
15            k = -1; maxc = -1;
16            for (j = 1; j <= n; j++)
17                if (!bin[j] && !vis[j] && dist[j] > maxc) {
18                    k = j; maxc = dist[j]; }
19            if (k == -1) return mincut;
20            s = t; t = k; mincut = maxc; vis[k] = true;
21            for (j = 1; j <= n; j++) if (!bin[j] && !vis[j])
22                dist[j] += edge[k][j];
23            return mincut; }
24        int solve () {
25            int mincut, i, j, s, t, ans;
26            for (mincut = INF, i = 1; i < n; i++) {
27                ans = contract (s, t); bin[t] = true;
28                if (mincut > ans) mincut = ans;
29                if (mincut == 0) return 0;
30                for (j = 1; j <= n; j++) if (!bin[j])
31                    edge[s][j] = (edge[j][s] += edge[j][t]); }
32            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 (v[i].d = 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(); ++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 () > 0) {
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 () > 0) {
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 (); ++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                endl;

```

7 Appendix

7.1 Calculus table

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

7.1.1 $ax + b$ ($a \neq 0$)

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

7.1.2 $\sqrt{ax+b}$

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

7.1.3 $x^2 \pm a^2$

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

7.1.4 $ax^2 + b$ ($a > 0$)

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

7.1.5 $ax^2 + bx + c$ ($a > 0$)

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

7.1.6 $\sqrt{x^2+a^2}$ ($a > 0$)

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

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

7.1.7 $\sqrt{x^2-a^2}$ ($a > 0$)

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

7.1.8 $\sqrt{a^2-x^2}$ ($a > 0$)

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

7.1.9 $\sqrt{\pm ax^2 + bx + c}$ ($a > 0$)

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

7.1.10 $\sqrt{\pm \frac{x-a}{x-b}}$ & $\sqrt{(x-a)(x-b)}$

$$\begin{aligned}
 1. \int \sqrt{\frac{x-a}{x-b}} dx &= (x-b) \sqrt{\frac{x-a}{x-b}} + (b-a) \ln(\sqrt{|x-a|} + \sqrt{|x-b|}) + C \\
 2. \int \sqrt{\frac{x-a}{b-x}} dx &= (x-b) \sqrt{\frac{x-a}{b-x}} + (b-a) \arcsin \sqrt{\frac{x-a}{b-x}} + C \\
 3. \int \frac{dx}{\sqrt{(x-a)(b-x)}} &= 2 \arcsin \sqrt{\frac{x-a}{b-x}} + C \quad (a < b) \\
 4. \int \sqrt{(x-a)(b-x)} dx &= \frac{2x-a-b}{4} \sqrt{(x-a)(b-x)} + \frac{(b-a)^2}{4} \arcsin \sqrt{\frac{x-a}{b-x}} + C \quad (a < b)
 \end{aligned}$$

7.1.11 Triangular function

$$\begin{aligned}
 1. \int \tan x dx &= -\ln |\cos x| + C \\
 2. \int \cot x dx &= \ln |\sin x| + C \\
 3. \int \sec x dx &= \ln \left| \tan \left(\frac{\pi}{4} + \frac{x}{2} \right) \right| + C = \ln |\sec x + \tan x| + C \\
 4. \int \csc x dx &= \ln \left| \tan \frac{x}{2} \right| + C = \ln |\csc x - \cot x| + C \\
 5. \int \sec^2 x dx &= \tan x + C \\
 6. \int \csc^2 x dx &= -\cot x + C \\
 7. \int \sec x \tan x dx &= \sec x + C \\
 8. \int \csc x \cot x dx &= -\csc x + C \\
 9. \int \sin^2 x dx &= \frac{x}{2} - \frac{1}{4} \sin 2x + C \\
 10. \int \cos^2 x dx &= \frac{x}{2} + \frac{1}{4} \sin 2x + C
 \end{aligned}$$

11. $\int \sin^n x dx = -\frac{1}{n} \sin^{n-1} x \cos x + \frac{n-1}{n} \int \sin^{n-2} x dx$
12. $\int \cos^n x dx = \frac{1}{n} \cos^{n-1} x \sin x + \frac{n-1}{n} \int \cos^{n-2} x dx$
13. $\frac{dx}{\sin^n x} = -\frac{1}{n-1} \frac{\cos x}{\sin^{n-1} x} + \frac{n-2}{n-1} \int \frac{dx}{\sin^{n-2} x}$
14. $\frac{dx}{\cos^n x} = \frac{1}{n-1} \frac{\sin x}{\cos^{n-1} x} + \frac{n-2}{n-1} \int \frac{dx}{\cos^{n-2} x}$
- 15.

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

16. $\int \sin ax \cos bx dx = -\frac{1}{2(a+b)} \cos(a+b)x - \frac{1}{2(a-b)} \cos(a-b)x + C$
17. $\int \sin ax \sin bx dx = -\frac{1}{2(a+b)} \sin(a+b)x + \frac{1}{2(a-b)} \sin(a-b)x + C$
18. $\int \cos ax \cos bx dx = \frac{1}{2(a+b)} \sin(a+b)x + \frac{1}{2(a-b)} \sin(a-b)x + C$
19. $\int \frac{dx}{a+b \sin x} = \begin{cases} \frac{2}{\sqrt{a^2-b^2}} \arctan \frac{a \tan \frac{x}{2} + b}{\sqrt{a^2-b^2}} + C & (a^2 > b^2) \\ \frac{1}{\sqrt{b^2-a^2}} \ln \left| \frac{a \tan \frac{x}{2} + b - \sqrt{b^2-a^2}}{a \tan \frac{x}{2} + b + \sqrt{b^2-a^2}} \right| + C & (a^2 < b^2) \end{cases}$
20. $\int \frac{dx}{a+b \cos x} = \begin{cases} \frac{2}{a+b} \sqrt{\frac{a+b}{a-b}} \arctan \left(\sqrt{\frac{a-b}{a+b}} \tan \frac{x}{2} \right) + C & (a^2 > b^2) \\ \frac{1}{a+b} \sqrt{\frac{a+b}{a-b}} \ln \left| \frac{\tan \frac{x}{2} + \sqrt{\frac{a+b}{b-a}}}{\tan \frac{x}{2} - \sqrt{\frac{a+b}{b-a}}} \right| + C & (a^2 < b^2) \end{cases}$
21. $\int \frac{dx}{a^2 \cos^2 x + b^2 \sin^2 x} = \frac{1}{ab} \arctan \left(\frac{b}{a} \tan x \right) + C$
22. $\int \frac{dx}{a^2 \cos^2 x - b^2 \sin^2 x} = \frac{1}{2ab} \ln \left| \frac{b \tan x + a}{b \tan x - a} \right| + C$
23. $\int x \sin ax dx = \frac{1}{a^2} \sin ax - \frac{1}{a} x \cos ax + C$
24. $\int x^2 \sin ax dx = -\frac{1}{a} x^2 \cos ax + \frac{2}{a^2} x \sin ax + \frac{2}{a^3} \cos ax + C$
25. $\int x \cos ax dx = \frac{1}{a^2} \cos ax + \frac{1}{a} x \sin ax + C$
26. $\int x^2 \cos ax dx = \frac{1}{a} x^2 \sin ax + \frac{2}{a^2} x \cos ax - \frac{2}{a^3} \sin ax + C$

7.1.12 Inverse triangular function ($a > 0$)

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

7.1.13 Exponential function

1. $\int a^x dx = \frac{1}{\ln a} a^x + C$
2. $\int e^{ax} dx = \frac{1}{a} e^{ax} + C$
3. $\int x e^{ax} dx = \frac{1}{a^2} (ax - 1) e^{ax} + C$
4. $\int x^n e^{ax} dx = \frac{1}{a} x^n e^{ax} - \frac{n}{a} \int x^{n-1} e^{ax} dx$
5. $\int x a^x dx = \frac{x}{\ln a} a^x - \frac{1}{(\ln a)^2} a^x + C$
6. $\int x^n a^x dx = \frac{1}{\ln a} x^n a^x - \frac{n}{\ln a} \int x^{n-1} a^x dx$
7. $\int e^{ax} \sin bx dx = \frac{1}{a^2 + b^2} e^{ax} (a \sin bx - b \cos bx) + C$
8. $\int e^{ax} \cos bx dx = \frac{1}{a^2 + b^2} e^{ax} (b \sin bx + a \cos bx) + C$
9. $\int e^{ax} \sin^n bx dx = \frac{1}{a^2 + b^2 n^2} e^{ax} \sin^{n-1} bx (a \sin bx - n b \cos bx) + \frac{n(n-1)b^2}{a^2 + b^2 n^2} \int e^{ax} \sin^{n-2} bx dx$
10. $\int e^{ax} \cos^n bx dx = \frac{1}{a^2 + b^2 n^2} e^{ax} \cos^{n-1} bx (a \cos bx + n b \sin bx) + \frac{n(n-1)b^2}{a^2 + b^2 n^2} \int e^{ax} \cos^{n-2} bx dx$

7.1.14 Logarithmic function

1. $\int \ln x dx = x \ln x - x + C$
2. $\int \frac{dx}{x \ln x} = \ln |\ln x| + C$
3. $\int x^n \ln x dx = \frac{1}{n+1} x^{n+1} (\ln x - \frac{1}{n+1}) + C$
4. $\int (\ln x)^n dx = x (\ln x)^n - n \int (\ln x)^{n-1} dx$
5. $\int x^m (\ln x)^n dx = \frac{1}{m+1} x^{m+1} (\ln x)^n - \frac{n}{m+1} \int x^m (\ln x)^{n-1} dx$