

Содержание

1	final/template/template.cpp	1	49	final/graphs/heavyLight.cpp	17
2	Practice round	1	50	final/graphs/hungary.cpp	17
3	final/stuff/debug.cpp	1	51	final/graphs/minCost.cpp	17
4	final/template/fastIO.cpp	2	52	final/graphs/minCostNegCycle.cpp	18
5	final/template/optimizations.cpp	2	53	final/graphs/retro.cpp	19
6	final/template/useful.cpp	2	54	final/graphs/mincut.cpp	19
7	final/template/Template.java	2	55	final/graphs/twoChineseFast.cpp	19
8	final/template/bitset.cpp	3	56	final/graphs/linkcut.cpp	20
9	final/numeric/fft.cpp	4	57	final/graphs/chordaltree.cpp	21
10	final/numeric/fst.cpp	4	58	final/graphs/minimization.cpp	21
11	final/numeric/fftint.cpp	5	59	final/graphs/matroidIntersection.cpp	21
12	final/numeric/berlekamp.cpp	5	60	final/graphs/compressTree.cpp	22
13	final/numeric/blackbox.cpp	6			
14	final/numeric/crt.cpp	6			
15	final/numeric/extendedgcd.cpp	6			
16	final/numeric/mulMod.cpp				
17	final/numeric/modReverse.cpp				
18	final/numeric/pollard.cpp				
19	final/numeric/poly.cpp				
20	final/numeric/simplex.cpp				
21	final/numeric/sumLine.cpp				
22	final/numeric/integrate.cpp				
23	final/numeric/rootsPolynom.cpp				
24	final/numeric/phiFunction.cpp				
25	final/numeric/partition.cpp				
26	final/geom/commonTangents.cpp				
27	final/geom/halfplaneIntersection.cpp				
28	final/geom/minDisc.cpp				
29	final/geom/convexHull3D-N2.cpp				
30	final/geom/convexDynamic.cpp				
31	final/geom/polygonArcCut.cpp				
32	final/geom/polygonTangent.cpp				
33	final/geom/checkPlaneInt.cpp				
34	final/geom/furthestPoints.cpp				
35	final/geom/chtDynamic.cpp				
36	final/geom/rotate3D.cpp				
37	final/geom/circleInter.cpp				
38	final/geom/sphericalDistance.cpp				
39	final/strings/eertree.cpp				
40	final/strings/manacher.cpp				
41	final/strings/sufAutomaton.cpp				
42	final/strings/sufTree.cpp				
43	final/strings/sufArray.cpp				
44	final/strings/sufArrayLinear.cpp				
45	final/strings/duval.cpp				
46	final/graphs/alphaBeta.cpp				
47	final/graphs/dominatorTree.cpp				
48	final/graphs/generalMatching.cpp				

1 final/template/template.cpp

```

1 // team : SPb ITMO University Komanda
2 #include <bits/stdc++.h>
3 #ifdef SIR
4 #define err(...) fprintf(stderr, __VA_ARGS__)
5 #else
6 #define err(...) 42
7 #endif
8
9 #define db(x) cerr << #x << " = " << x << endl
10 #define db2(x, y) cerr << "(" << #x << ", " << #y << "\n" <<
11 #define db3(x, y, z) cerr << "(" << #x << ", " << #y <<
12 #define dbv(a) cerr << #a << " = "; for (auto xxxx: a) cerr << xxxx << " "; cerr << endl
13
14 using namespace std;
15
16 typedef long long ll;
17
18 void solve() {
19
20 }
21
22 int main() {
23 #ifdef SIR
24     freopen("input.txt", "r", stdin), freopen("output.txt", "w", stdout);
25 #endif
26     ios_base::sync_with_stdio(0);
27     cin.tie(0);
28     solve();
29     return 0;
30 }

```

2 Practice round

- Посабмитить задачи каждому человеку.
- Распечатать решение.
- IDE для джавы.
- Сравнить скорость локального компьютера и сервера.
- Проверить int128.
- Проверить прагмы. Например, на bitset.

3 final/stuff/debug.cpp

```

1 #include <bits/stdc++.h>
2 #define _GLIBCXX_DEBUG
3
4 using namespace std;
5
6 template <class T>
7 struct MyVector : vector<T> {
8     MyVector() : vector<T>() {}
9     MyVector( int n ) : vector<T>(n) {}
10    T &operator [] ( int i ) { return vector<T>::at(i)
11    }; }
12
13 T operator [] ( int i ) const { return vector<T>::at(i); }
14
15 };
16
17 /** Если в вашем коде местами использовать MyVector<int>,
18     использовать MyVector<int>,
19     вы увидите все range check errors- */
20 MyVector<int> b(10), a;
21
22 int main() {
23     MyVector<int> a(50);
24     for (int i = 1; i <= 600; i++) a[i] = i;
25     cout << a[500] << "\n";
26 }

```

4 final/template/fastIO.cpp

```

1 #include <cstdio>
2 #include <algorithm>
3
4 /** Interface */
5
6 inline int readInt();
7 inline int readUInt();
8 inline bool isEof();
9
10 /** Read */
11
12 static const int buf_size = 100000;
13 static char buf[buf_size];
14 static int buf_len = 0, pos = 0;
15
16 inline bool isEof() {
17     if (pos == buf_len) {
18         pos = 0, buf_len = fread(buf, 1, buf_size, stdin);
19     }
20     if (pos == buf_len) return 1;
21     return 0;
22 }
23
24 inline int getChar() { return isEof() ? -1 : buf[pos++]; }
25
26 inline int readChar() {
27     int c = getChar();
28     while (c != -1 && c <= 32) c = getChar();
29     return c;
30 }
31
32 inline int readUInt() {
33     int c = readChar(), x = 0;
34     while ('0' <= c && c <= '9') x = x * 10 + c - '0', c = readChar();
35     return x;
36 }
37
38 inline int readInt() {
39     int s = 1, c = readChar();
40     int x = 0;
41     if (c == '-') s = -1, c = getChar();
42     while ('0' <= c && c <= '9') x = x * 10 + c - '0', c = getChar();
43     return s == 1 ? x : -x;
44 }
45
46 // 10M int [0..1e9]
47 // cin 3.02
48 // scanf 1.2
49 // cin_sync_with_stdio(false) 0.71
50 // fastRead getchar 0.53
51 // fastRead fread 0.15
52

```

5 final/template/optimizations.cpp

```

1 inline void fasterLLDivMod(unsigned long long x, ←
2     unsigned y, unsigned &out_d, unsigned &out_m) {
3     unsigned xh = (unsigned)(x >> 32), xl = (unsigned)
4     x, d, m;
5 #ifdef __GNUC__
6     asm(
7         "divl %4; \n\t"
8         : "=a" (d), "=d" (m)
9         : "d" (xh), "a" (xl), "r" (y)
10        );
11 #else
12     __asm {
13         mov edx, dword ptr[xh];
14         mov eax, dword ptr[xl];
15         div dword ptr[y];
16         mov dword ptr[d], eax;
17         mov dword ptr[m], edx;
18     };
19 #endif
20     out_d = d; out_m = m;
21 }
22
23 // have no idea what sse flags are really cool; list ←
24 // of some of them
25 // -- very good with bitsets
26 #pragma GCC optimize("O3")
27 #pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt, ←
28     abm,mmx")
29

```

6 final/template/useful.cpp

```

1 #include "ext/pb_ds/assoc_container.hpp"
2 #include <bits/extc++.h> /** keep-include */
3 using namespace __gnu_pbds;
4
5 gp_hash_table<ll, int> h({},{},{},{},{}, {1 << 16});
6 template <typename T> using ordered_set = tree<T, ←
7     null_type, less<T>, rb_tree_tag, ←
8     tree_order_statistics_node_update>;
9
10 template <typename K, typename V> using ordered_map ←
11     = tree<K, V, less<K>, rb_tree_tag, ←
12     tree_order_statistics_node_update>;
13
14 // HOW TO USE ::
15 // -- order_of_key(10) returns the number of ←
16 // elements in set/map strictly less than 10
17 // -- *find_by_order(10) returns 10-th smallest ←
18 // element in set/map (0-based)
19
20 bitset<N> a;
21 for (int i = a._Find_first(); i != a.size(); i = a. ←
22     _Find_next(i)) {
23     cout << i << endl;
24 }
25

```

7 final/template/Template.java

```

1 import java.util.*;
2 import java.io.*;
3
4 public class Template {
5     FastScanner in;
6     PrintWriter out;
7
8     public void solve() throws IOException {
9         int n = in.nextInt();
10        out.println(n);
11    }
12
13    public void run() {
14        try {
15            in = new FastScanner();
16            out = new PrintWriter(System.out);
17
18            solve();
19        }
20    }
21

```

```

20     out.close();
21 } catch (IOException e) {
22     e.printStackTrace();
23 }
24 }
25
26 class FastScanner {
27     BufferedReader br;
28     StringTokenizer st;
29
30     FastScanner() {
31         br = new BufferedReader(new InputStreamReader(↵
32         System.in));
33     }
34
35     String next() {
36         while (st == null || !st.hasMoreTokens()) {
37             try {
38                 st = new StringTokenizer(br.readLine());
39             } catch (IOException e) {
40                 e.printStackTrace();
41             }
42         }
43         return st.nextToken();
44     }
45
46     int nextInt() {
47         return Integer.parseInt(next());
48     }
49 }
50
51 public static void main(String[] arg) {
52     new Template().run();
53 }

```

```

47     return res;
48 }
49 };

```

8 final/template/bitset.cpp

```

1
2 const int SZ = 6;
3 const int BASE = pw(SZ);
4 const int MOD = BASE - 1;
5
6 struct Bitset {
7     typedef unsigned long long T;
8     vector<T> data;
9     int n;
10    void resize(int nn) {
11        n = nn;
12        data.resize((n + BASE - 1) / BASE);
13    }
14    void set(int pos, int val) {
15        int id = pos >> SZ;
16        int rem = pos & MOD;
17        data[id] ^= data[id] & pw(rem);
18        data[id] |= val * pw(rem);
19    }
20    int get(int pos) {
21        return (data[pos >> SZ] >> (pos & MOD)) & 1;
22    }
23    // k > 0 -> (*this) << k
24    // k < 0 -> (*this) >> (-k)
25    Bitset shift (int k) {
26        Bitset res;
27        res.resize(n);
28        int s = k / BASE;
29        int rem = k % BASE;
30        if (rem < 0) {
31            rem += BASE;
32            s--;
33        }
34        int p1 = BASE - rem;
35        T mask = (p1 == 64)? -1: pw(p1) - 1;
36        for (int i = max(0, -s); i < sz(data) - max(s, ↵
37            0); i++) {
38            res.data[i + s] |= (data[i] & mask) << rem;
39        }
40        if (rem != 0) {
41            for (int i = max(0, -s - 1); i < sz(data) - ↵
42                max(s + 1, 0); i++) {
43                res.data[i + s + 1] |= (data[i] >> p1) & (pw↵
44                    (rem) - 1);
45            }
46        }
47        int cc = data.size() * BASE - n;
48        res.data.back() <<= cc;
49        res.data.back() >>= cc;

```

9 final/numeric/fft.cpp

```

1 namespace fft
2 {
3     const int maxBase = 21;
4     const int maxN = 1 << maxBase;
5
6     struct num
7     {
8         dbl x, y;
9         num() {}
10        num(dbl xx, dbl yy): x(xx), y(yy) {}
11        num(dbl alp): x(cos(alp)), y(sin(alp)) {}
12    };
13
14    inline num operator + (num a, num b) { return num(a.x + b.x, a.y + b.y); }
15    inline num operator - (num a, num b) { return num(a.x - b.x, a.y - b.y); }
16    inline num operator * (num a, num b) { return num(a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x); }
17
18    inline num conj(num a) { return num(a.x, -a.y); }
19
20    const dbl PI = acos(-1);
21
22    num root[maxN];
23    int rev[maxN];
24    bool rootsPrepared = false;
25
26    void prepRoots()
27    {
28        if (rootsPrepared) return;
29        rootsPrepared = true;
30        root[1] = num(1, 0);
31        for (int k = 1; k < maxBase; ++k)
32        {
33            num x(2 * PI / pw(k + 1));
34            for (int i = pw(k - 1); i < pw(k); ++i)
35            {
36                root[2 * i] = root[i];
37                root[2 * i + 1] = root[i] * x;
38            }
39        }
40
41        int base, N;
42
43        int lastRevN = -1;
44        void prepRev()
45        {
46            if (lastRevN == N) return;
47            lastRevN = N;
48            for (int i = 0; i < N; ++i) rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (base - 1));
49        }
50
51        void fft(num *a, num *f)
52        {
53            for (int i = 0; i < N; i += 2 * k) for (int j = 0; j < 2 * k; ++j)
54            {
55                num z = f[i + j + k] * root[j + k];
56                f[i + j + k] = f[i + j] - z;
57                f[i + j] = f[i + j] + z;
58            }
59        }
60
61        num a[maxN], b[maxN], f[maxN], g[maxN];
62        ll A[maxN], B[maxN], C[maxN];
63
64        void _multMod(int mod)
65        {
66            for (int i = 0; i < N; ++i)
67            {
68                int x = A[i] % mod;
69                a[i] = num(x & (pw(15) - 1), x >> 15);
70            }
71            for (int i = 0; i < N; ++i)
72            {
73                int x = B[i] % mod;
74                b[i] = num(x & (pw(15) - 1), x >> 15);
75            }
76            fft(a, f);
77            fft(b, g);
78
79            for (int i = 0; i < N; ++i)
80            {
81                int j = (N - i) & (N - 1);

```

```

83        num a1 = (f[i] + conj(f[j])) * num(0.5, 0);
84        num a2 = (f[i] - conj(f[j])) * num(0, -0.5);
85        num b1 = (g[i] + conj(g[j])) * num(0.5 / N, 0);
86        num b2 = (g[i] - conj(g[j])) * num(0, -0.5 / N);
87        a[j] = a1 * b1 + a2 * b2 * num(0, 1);
88        b[j] = a1 * b2 + a2 * b1;
89    }
90
91    fft(a, f);
92    fft(b, g);
93
94    for (int i = 0; i < N; ++i)
95    {
96        ll aa = f[i].x + 0.5;
97        ll bb = g[i].x + 0.5;
98        ll cc = f[i].y + 0.5;
99        C[i] = (aa + bb % mod * pw(15) + cc % mod * pw(30)) % mod;
100    }
101
102    void prepAB(int n1, int n2)
103    {
104        base = 1;
105        N = 2;
106        while (N < n1 + n2) base++, N <= 1;
107
108        for (int i = n1; i < N; ++i) A[i] = 0;
109        for (int i = n2; i < N; ++i) B[i] = 0;
110
111        prepRoots();
112        prepRev();
113    }
114
115    void mult(int n1, int n2)
116    {
117        prepAB(n1, n2);
118        for (int i = 0; i < N; ++i) a[i] = num(A[i], B[i]);
119        fft(a, f);
120        for (int i = 0; i < N; ++i)
121        {
122            int j = (N - i) & (N - 1);
123            a[i] = (f[j] * f[j] - conj(f[i] * f[i])) * num(0, -0.25 / N);
124        }
125        fft(a, f);
126        for (int i = 0; i < N; ++i) C[i] = (ll)round(f[i].x);
127    }
128
129    void multMod(int n1, int n2, int mod)
130    {
131        prepAB(n1, n2);
132        _multMod(mod);
133    }
134
135    int D[maxN];
136
137    void multLL(int n1, int n2)
138    {
139        prepAB(n1, n2);
140
141        int mod1 = 1.5e9;
142        int mod2 = mod1 + 1;
143
144        _multMod(mod1);
145
146        for (int i = 0; i < N; ++i) D[i] = C[i];
147
148        _multMod(mod2);
149
150        for (int i = 0; i < N; ++i)
151        {
152            C[i] = D[i] + (C[i] - D[i] + (ll)mod2) * (ll)mod1 % mod2 * mod1;
153        }
154    }
155
156    // HOW TO USE ::
157    // — set correct maxBase
158    // — use mult(n1, n2), multMod(n1, n2, mod) and multLL(n1, n2)
159    // — input : A[], B[]
160    // — output : C[]
161
162    }

```

10 final/numeric/fst.cpp

```

1 Transform to a basis with fast convolutions of the ←
  form
2 *  $\displaystyle c[z] = \sum_{\text{nolimits}} \{z = x \oplus y\} a[x] \cdot b[y]$ ,
3 * where  $\oplus$  is one of AND, OR, XOR. The size ←
  of  $a$  must be a power of two.
4
5 void FST(vi& a, bool inv) {
6     for (int n = sz(a), step = 1; step < n; step *= 2) ←
7     {
8         for (int i = 0; i < n; i += 2 * step) rep(j, i, i + ←
9             step) {
10             int &u = a[j], &v = a[j + step]; tie(u, v) =
11                 inv ? pii(v - u, u) : pii(v, u + v); // AND
12                 inv ? pii(v, u - v) : pii(u + v, u); // OR
13                 pii(u + v, u - v); // XOR
14             }
15         if (inv) trav(x, a) x /= sz(a); // XOR only
16     }
17     vi conv(vi a, vi b) {
18         FST(a, 0); FST(b, 0);
19         rep(i, 0, sz(a)) a[i] *= b[i];
20         FST(a, 1); return a;
    }

```

11 final/numeric/fftint.cpp

```

1 namespace fft {
2     const int MOD = 998244353;
3     const int maxB = 20;
4     const int maxN = 1 << maxB;
5     const int initROOT = 646;
6
7     int root[maxN];
8     int rev[maxN];
9     int N;
10
11     ll inv(ll a, ll m = MOD) {
12         if (a == 0) return 0;
13         return ((1 - inv(m % a, a) * m) / a + m) % m;
14     }
15
16     void _init(int cur_base) {
17         N = 1 << cur_base;
18         for (int i = 0; i < N; i++) rev[i] = (rev[i >> ←
19             1] >> 1) + ((i & 1) << (cur_base - 1));
20
21         int ROOT = initROOT;
22         for (int i = cur_base; i < 20; i++) ROOT = 1ll * ←
23             ROOT * ROOT % MOD;
24
25         int NN = N >> 1;
26         int z = 1;
27         for (int i = 0; i < NN; i++) {
28             root[i + NN] = z;
29             z = z * (1ll)ROOT % MOD;
30         }
31         for (int i = NN - 1; i > 0; --i) root[i] = root ←
32             [2 * i];
33
34         void fft(int *a, int *f) {
35             for (int i = 0; i < N; i++) f[i] = a[rev[i]];
36             for (int k = 1; k < N; k <= 1) {
37                 for (int i = 0; i < N; i += 2 * k) {
38                     for (int j = 0; j < k; j++) {
39                         int z = f[i + j + k] * (1ll)root[j + k] % ←
40                             MOD;
41                         f[i + j + k] = (f[i + j] - z + MOD) % MOD;
42                         f[i + j] = (f[i + j] + z) % MOD;
43                     }
44                 }
45             }
46
47             int A[maxN], B[maxN], C[maxN];
48             int F[maxN], G[maxN];
49
50             void _mult(int eq) {
51                 fft(A, F);
52                 if (eq)
53                     for (int i = 0; i < N; i++)
54                         G[i] = F[i];
55                 else fft(B, G);
            }
        }
    }

```

```

56     int invN = inv(N);
57     for (int i = 0; i < N; i++) A[i] = F[i] * (1ll)G[ ←
58         i] % MOD * invN % MOD;
59     reverse(A + 1, A + N);
60     fft(A, C);
61 }
62
63 void mult(int n1, int n2, int eq = 0) {
64     int n = n1 + n2, cur_base = 0;
65     while ((1 << cur_base) < n) cur_base++;
66     _init(cur_base + 1);
67
68     for (int i = n1; i < N; ++i) A[i] = 0;
69     for (int i = n2; i < N; ++i) B[i] = 0;
70
71     _mult(eq);
72
73     // for (i, n1 + n2) C[i] = 0;
74     // for (i, n1) for (j, n2) C[i + j] = (C[i + j] + ←
75         A[i] * (1ll)B[j]) % mod;
76 }
77
78 vector<int> mult(vector<int> A, vector<int> B) {
79     for (int i = 0; i < A.size(); i++) fft::A[i] = A[ ←
80         i];
81     for (int i = 0; i < A.size(); i++) fft::B[i] = B[ ←
82         i];
83     mult(A.size(), B.size());
84     vector<int> C(A.size() + B.size());
85     for (int i = 0; i < A.size() + B.size(); i++) C[ ←
86         i] = fft::C[i];
87     return C;
88 }

```

12 final/numeric/berlekamp.cpp

```

1 vector<int> berlekamp(vector<int> s) {
2     int l = 0;
3     vector<int> la(1, 1);
4     vector<int> b(1, 1);
5     for (int r = 1; r <= (int)s.size(); r++) {
6         int delta = 0;
7         for (int j = 0; j <= 1; j++) {
8             delta = (delta + 1ll * s[r - 1 - j] * la[j]) % ←
9                 MOD;
10        }
11        b.insert(b.begin(), 0);
12        if (delta != 0) {
13            vector<int> t(max(la.size(), b.size()));
14            for (int i = 0; i < (int)t.size(); i++) {
15                if (i < (int)la.size()) t[i] = (t[i] + la[i] ←
16                    ) % MOD;
17                if (i < (int)b.size()) t[i] = (t[i] - 1ll * ←
18                    delta * b[i] % MOD + MOD) % MOD;
19            }
20            if (2 * l <= r - 1) {
21                b = la;
22                int od = inv(delta);
23                for (int &x : b) x = 1ll * x * od % MOD;
24                l = r - 1;
25            }
26            la = t;
27        }
28        assert((int)la.size() == l + 1);
29        assert(1 * 2 + 30 < (int)s.size());
30        reverse(la.begin(), la.end());
31        return la;
32    }
33
34 vector<int> mul(vector<int> a, vector<int> b) {
35     vector<int> c(a.size() + b.size() - 1);
36     for (int i = 0; i < (int)a.size(); i++) {
37         for (int j = 0; j < (int)b.size(); j++) {
38             c[i + j] = (c[i + j] + 1ll * a[i] * b[j]) % ←
39                 MOD;
40         }
41     }
42     vector<int> res(c.size());
43     for (int i = 0; i < (int)res.size(); i++) res[i] = ←
44         c[i] % MOD;
45     return res;
46 }
47
48 vector<int> mod(vector<int> a, vector<int> b) {
49     if (a.size() < b.size()) a.resize(b.size() - 1);
50 }

```

13 final/numeric/blackbox.cpp

```

46 int o = inv(b.back());
47 for (int i = (int)a.size() - 1; i >= (int)b.size() - 1; i--) {
48     if (a[i] == 0) continue;
49     int coef = 1LL * o * (MOD - a[i]) % MOD;
50     for (int j = 0; j < (int)b.size(); j++) {
51         a[i - (int)b.size() + 1 + j] = (a[i - (int)b.size() + 1 + j] + 1LL * coef * b[j]) % MOD;
52     }
53 }
54 while (a.size() >= b.size()) {
55     assert(a.back() == 0);
56     a.pop_back();
57 }
58 return a;
59 }
60
61 vector<int> bin(int n, vector<int> p) {
62     vector<int> res(1, 1);
63     vector<int> a(2); a[1] = 1;
64     while (n) {
65         if (n & 1) res = mod(mul(res, a), p);
66         a = mod(mul(a, a), p);
67         n >>= 1;
68     }
69     return res;
70 }
71
72 int f(vector<int> t, int m) {
73     vector<int> v = berlekamp(t);
74     vector<int> o = bin(m - 1, v);
75     int res = 0;
76     for (int i = 0; i < (int)o.size(); i++) res = (res + 1LL * o[i] * t[i]) % MOD;
77     return res;
78 }
79

```

14 final/numeric/crt.cpp

```

1 int CRT(int a1, int m1, int a2, int m2) {
2     return (a1 - a2 % m1 + m1) * ((11)rev(m2, m1) % m1 +
3         * m2 + a2;

```

15 final/numeric/extendedgcd.cpp

```

1 int gcd(int a, int b, int &x, int &y) {
2     if (a == 0) {
3         x = 0, y = 1;
4         return b;
5     }
6     int x1, y1;
7     int d = gcd(b % a, a, x1, y1);
8     x = y1 - (b / a) * x1;
9     y = x1;
10    return d;
11 }

```

16 final/numeric/mulMod.cpp

```

1 ll mul(ll a, ll b, ll m) { // works for MOD 8e18
2     ll k = (ll)((long double)a * b / m);
3     ll r = a * b - m * k;
4     if (r < 0) r += m;
5     if (r >= m) r -= m;
6     return r;
7 }

```

17 final/numeric/modReverse.cpp

```

1 int rev(int x, int m) {
2     if (x == 1) return 1;
3     return (1 - rev(m % x, x) * (11)m) / x + m;
4 }

```

18 final/numeric/pollard.cpp

```

1 namespace pollard
2 {
3     using math::p;
4
5     vector<pair<ll, int>> getFactors(ll N) {
6         vector<ll> primes;
7
8         const int MX = 1e5;
9         const ll MX2 = MX * (11)MX;
10
11         assert(MX <= math::maxP && math::pc > 0);
12
13         function<void(ll)> go = [&go, &primes](ll n) {
14             for (ll x : primes) while (n % x == 0) n /= x;
15             if (n == 1) return;
16             if (n > MX2) {
17                 auto F = [&](ll x) {
18                     ll k = ((long double)x * x) / n;
19                     ll r = (x * x - k * n + 3) % n;
20                     return r < 0 ? r + n : r;
21                 };
22                 ll x = mt19937_64()() % n, y = x;
23                 const int C = 3 * pow(n, 0.25);
24
25                 ll val = 1;
26                 for(it, C) {
27                     x = F(x), y = F(F(y));
28                     if (x == y) continue;
29                     ll delta = abs(x - y);
30                     ll k = ((long double)val * delta) / n;
31                     val = (val * delta - k * n) % n;
32                     if (val < 0) val += n;
33                     if (val == 0) {
34                         ll g = __gcd(delta, n);
35                         go(g), go(n / g);
36                         return;
37                     }
38                     if ((it & 255) == 0) {
39                         ll g = __gcd(val, n);

```

```

40         if (g != 1) {
41             go(g), go(n / g);
42             return;
43         }
44     }
45 }
46 }
47 primes.pb(n);
48 };
49
50 ll n = N;
51
52 for (int i = 0; i < math::pc && p[i] < MX; ++i) ←
53 if (n % p[i] == 0) {
54     primes.pb(p[i]);
55     while (n % p[i] == 0) n /= p[i];
56 }
57 go(n);
58 sort(primes.begin(), primes.end());
59
60 vector<pair<ll, int>> res;
61 for (ll x : primes) {
62     int cnt = 0;
63     while (N % x == 0) {
64         cnt++;
65         N /= x;
66     }
67     res.push_back({x, cnt});
68 }
69 return res;
70 }

```

19 final/numeric/poly.cpp

```

1 struct poly
2 {
3     vi v;
4     poly() {}
5     poly(vi vv)
6     {
7         v = vv;
8     }
9     int size()
10    {
11        return (int)v.size();
12    }
13    poly cut(int maxLen)
14    {
15        if (maxLen < sz(v)) v.resize(maxLen);
16        return *this;
17    }
18    poly norm()
19    {
20        while (sz(v) > 1 && v.back() == 0) v.pop_back();
21        return *this;
22    }
23    inline int& operator [] (int i)
24    {
25        return v[i];
26    }
27    void out(string name="")
28    {
29        stringstream ss;
30        if (sz(name)) ss << name << "=";
31        int fst = 1;
32        forn(i, sz(v)) if (v[i])
33        {
34            int x = v[i];
35            int sgn = 1;
36            if (x > mod / 2) x = mod - x, sgn = -1;
37            if (sgn == -1) ss << "-";
38            else if (!fst) ss << "+";
39            fst = 0;
40            if (!i || x != 1)
41            {
42                ss << x;
43                if (i > 0) ss << "*x";
44                if (i > 1) ss << "^" << i;
45            }
46            else
47            {
48                ss << "x";
49                if (i > 1) ss << "^" << i;
50            }
51        }
52        if (fst) ss << "0";

```

```

53     string s;
54     ss >> s;
55     eprintf("%s\n", s.data());
56 }
57 };
58
59 poly operator + (poly A, poly B)
60 {
61     poly C;
62     C.v = vi(max(sz(A), sz(B)));
63     forn(i, sz(C))
64     {
65         if (i < sz(A)) C[i] = (C[i] + A[i]) % mod;
66         if (i < sz(B)) C[i] = (C[i] + B[i]) % mod;
67     }
68     return C.norm();
69 }
70
71 poly operator - (poly A, poly B)
72 {
73     poly C;
74     C.v = vi(max(sz(A), sz(B)));
75     forn(i, sz(C))
76     {
77         if (i < sz(A)) C[i] = (C[i] + A[i]) % mod;
78         if (i < sz(B)) C[i] = (C[i] + mod - B[i]) % mod;
79     }
80     return C.norm();
81 }
82
83 poly operator * (poly A, poly B)
84 {
85     poly C;
86     C.v = vi(sz(A) + sz(B) - 1);
87
88     forn(i, sz(A)) fft::A[i] = A[i];
89     forn(i, sz(B)) fft::B[i] = B[i];
90     fft::multMod(sz(A), sz(B), mod);
91     forn(i, sz(C)) C[i] = fft::C[i];
92     return C.norm();
93 }
94
95 poly inv(poly A, int n) // returns A^{-1} mod x^n
96 {
97     assert(sz(A) && A[0] != 0);
98     A.cut(n);
99
100    auto cutPoly = [](poly &from, int l, int r)
101    {
102        poly R;
103        R.v.resize(r - 1);
104        for (int i = l; i < r; ++i)
105        {
106            if (i < sz(from)) R[i - l] = from[i];
107        }
108        return R;
109    };
110
111    function<int(int, int)> rev = [&rev](int x, int m) ←
112    →int
113    {
114        if (x == 1) return 1;
115        return (1 - rev(m % x, x) * (ll)m) / x + m;
116    };
117
118    poly R({rev(A[0], mod)});
119    for (int k = 1; k < n; k <= 1)
120    {
121        poly A0 = cutPoly(A, 0, k);
122        poly A1 = cutPoly(A, k, 2 * k);
123        poly H = A0 * R;
124        H = cutPoly(H, k, 2 * k);
125        poly R1 = (((A1 * R).cut(k) + H) * (poly({0}) - ←
126        R)).cut(k);
127        R.v.resize(2 * k);
128        forn(i, k) R[i + k] = R1[i];
129    }
130    return R.cut(n).norm();
131 }
132
133 pair<poly, poly> divide(poly A, poly B)
134 {
135     if (sz(A) < sz(B)) return {poly({0}), A};
136
137     auto rev = [](poly f)
138     {
139         reverse(all(f.v));
140         return f;
141     };
142
143     poly q = rev((inv(rev(B), sz(A) - sz(B) + 1) * rev ←
144     (A)).cut(sz(A) - sz(B) + 1));
145     poly r = A - B * q;

```



```

143     return {q, r};
144 }
145

```

20 final/numeric/simplex.cpp

```

1  typedef double T; // long double, Rational, double +<
2  mod<P>...
3  typedef vector<T> vd;
4  typedef vector<vd> vvd;
5
6  const T eps = 1e-8, inf = 1/.0;
7  #define MP make_pair
8  #define ltj(X) if(s == -1 || MP(X[j], N[j]) < MP(X[s<
9  ], N[s])) s=j
10 #define sz(X) ((X).size())
11 #define rep(i,l,r) for (int i = (l); i < (r); i++)
12
13 struct LPSolver {
14     // Description: Solves a general linear
15     // maximization problem: maximize $c^T x$ subject to
16     // $Ax \le b$, $x \ge 0$.
17     // A is a matrix with shape (number of
18     // inequalities, number of variables)
19     // Returns -inf if there is no solution, inf if
20     // there are arbitrarily good solutions, or the
21     // maximum value of $c^T x$ otherwise.
22     // The input vector is set to an optimal $x$ (or
23     // in the unbounded case, an arbitrary solution
24     // fulfilling the constraints).
25
26     int m, n;
27     vector<int> N, B;
28     vvd D;
29
30     LPSolver(const vvd& A, const vd& b, const vd& c) :
31         m(sz(b)), n(sz(c)), N(n+1), B(m), D(m+2, vd(n+<
32         2)) {
33         rep(i,0,m) rep(j,0,n) D[i][j] = A[i][j];
34         rep(i,0,m) { B[i] = n+i; D[i][n] = -1; D[i][n+1]<
35         = b[i]; }
36         rep(j,0,n) { N[j] = j; D[m][j] = -c[j]; }
37         N[n] = -1; D[m+1][n] = 1;
38     }
39
40     void pivot(int r, int s) {
41         T *a = D[r].data(), inv = 1 / a[s];
42         rep(i,0,m+2) if (i != r && abs(D[i][s]) > eps) {
43             T *b = D[i].data(), inv2 = b[s] * inv;
44             rep(j,0,n+2) b[j] -= a[j] * inv2;
45             b[s] = a[s] * inv2;
46         }
47         rep(j,0,n+2) if (j != s) D[r][j] *= inv;
48         rep(i,0,m+2) if (i != r) D[i][s] *= -inv;
49         D[r][s] = inv;
50         swap(B[r], N[s]);
51     }
52
53     bool simplex(int phase) {
54         int x = m + phase - 1;
55         for (int it = 0; it < 100; it++) {
56             int s = -1;
57             rep(j,0,n+1) if (N[j] != -phase) ltj(D[x]);
58             if (D[x][s] >= -eps) return true;
59             int r = -1;
60             rep(i,0,m) {
61                 if (D[i][s] <= eps) continue;
62                 if (r == -1 || MP(D[i][n+1] / D[i][s], B[i])
63                     < MP(D[r][n+1] / D[r][s], B[r])) r = i;
64             }
65             if (r == -1) return false;
66             pivot(r, s);
67         }
68     }
69
70     T solve(vd &x) {
71         int r = 0;
72         rep(i,1,m) if (D[i][n+1] < D[r][n+1]) r = i;
73         if (D[r][n+1] < -eps) {
74             pivot(r, n);
75             if (!simplex(2) || D[m+1][n+1] < -eps) return -inf;
76             rep(i,0,m) if (B[i] == -1) {
77                 int s = 0;
78                 rep(j,1,n+1) ltj(D[i]);
79                 pivot(i, s);
80             }
81         }
82     }
83 }
84

```

```

69     }
70 }
71 bool ok = simplex(1); x = vd(n);
72 rep(i,0,m) if (B[i] < n) x[B[i]] = D[i][n+1];
73 return ok ? D[m][n+1] : inf;
74 }
75

```

21 final/numeric/sumLine.cpp

```

1 // sum(i=0..n-1) (a+b*i) div m
2 ll solve(ll n, ll a, ll b, ll m) {
3     if (b == 0) return n * (a / m);
4     if (a >= m) return n * (a / m) + solve(n, a % m, b<
5     , m);
6     if (b >= m) return n * (n - 1) / 2 * (b / m) + <
7     solve(n, a, b % m, m);
8     return solve((a + b * n) / m, (a + b * n) % m, m, <
9     b);
10 }
11

```

22 final/numeric/integrate.cpp

```

1 function<dbl>(dbl, dbl, function<dbl>(>dbl>))> f = [&](<
2     dbl L, dbl R, function<dbl>(>dbl> g) {
3     const int ITTERS = 1000000;
4     dbl ans = 0;
5     dbl step = (R - L) * 1.0 / ITTERS;
6     for (int it = 0; it < ITTERS; it++) {
7         double x1 = L + step * it;
8         double xr = L + step * (it + 1);
9         dbl x1 = (x1 + xr) / 2;
10        dbl x0 = x1 - (x1 - x1) * sqrt(3.0 / 5);
11        dbl x2 = x1 + (x1 - x1) * sqrt(3.0 / 5);
12        ans += (5 * g(x0) + 8 * g(x1) + 5 * g(x2)) / 18 <
13        * step;
14    }
15    return ans;
16 }
17

```

23 final/numeric/rootsPolynom.cpp

```

1 const double EPS = 1e-9;
2 double cal(const vector<double> &coef, double x) {
3     double e = 1, s = 0;
4     for (double i : coef) s += i * e, e *= x;
5     return s;
6 }
7
8 int dblcmp(double x) {
9     if (x < -EPS) return -1;
10    if (x > EPS) return 1;
11    return 0;
12 }
13
14 double find(const vector<double> &coef, double l, <
15    double r) {
16    int sl = dblcmp(cal(coef, l)), sr = dblcmp(cal(co<
17    ef, r));
18    if (sl == 0) return l;
19    if (sr == 0) return r;
20    for (int tt = 0; tt < 100 && r - l > EPS; ++tt) {
21        double mid = (l + r) / 2;
22        int smid = dblcmp(cal(coef, mid));
23        if (smid == 0) return mid;
24        if (sl * smid < 0) r = mid;
25        else l = mid;
26    }
27    return (l + r) / 2;
28 }
29
30 vector<double> rec(const vector<double> &coef, int n<
31 ) {
32     vector<double> ret; // c[0]+c[1]*x+c[2]*x^2+...+c[<
33     n]*x^n, c[n]==1
34 }
35

```



```

30 if (n == 1) {
31     ret.push_back(-coef[0]);
32     return ret;
33 }
34 vector<double> dcoef(n);
35 for (int i = 0; i < n; ++i) dcoef[i] = coef[i + 1] ←
    * (i + 1) / n;
36 double b = 2; // fujiwara bound
37 for (int i = 0; i <= n; ++i) b = max(b, 2 * pow(←
    fabs(coef[i]), 1.0 / (n - i)));
38 vector<double> droot = rec(dcoef, n - 1);
39 droot.insert(droot.begin(), -b);
40 droot.push_back(b);
41 for (int i = 0; i + 1 < droot.size(); ++i) {
42     int sl = dblcmp(cal(coef, droot[i])), sr = ←
    dblcmp(cal(coef, droot[i + 1]));
43     if (sl * sr > 0) continue;
44     ret.push_back(find(coef, droot[i], droot[i + 1]) ←
    );
45 }
46 return ret;
47 }
48
49 vector<double> solve(vector<double> coef) {
50     int n = coef.size() - 1;
51     while (coef.back() == 0) coef.pop_back(), --n;
52     for (int i = 0; i <= n; ++i) coef[i] /= coef[n];
53     return rec(coef, n);
54 }
    
```

24 final/numeric/phiFunction.cpp

```

1 void totient(){
2     for(int i = 0; i < MAX; i++){
3         phi[i] = i;
4         pr[i] = true;
5     }
6     for(int i = 2; i < MAX; i++){
7         if(pr[i]){
8             for(int j = i; j < MAX; j+=i){
9                 pr[j] = false;
10                phi[j] = phi[j] - (phi[j] / i);
11            }
12            pr[i] = true;
13        }
14    }
    
```

25 final/numeric/partition.cpp

```

1 // number of ways to divide n to integers(unordered) ←
    , O(n^(3/2))
2 int partition(int n) {
3     int dp[n + 1];
4     dp[0] = 1;
5     for (int i = 1; i <= n; i++) {
6         dp[i] = 0;
7         for (int j = 1, r = 1; i - (3 * j * j - j) / 2 ←
            >= 0; ++j, r *= -1) {
8             dp[i] += dp[i - (3 * j * j - j) / 2] * r;
9             if (i - (3 * j * j + j) / 2 >= 0) dp[i] += dp[←
                i - (3 * j * j + j) / 2] * r;
10        }
11    }
12    return dp[n];
13 }
    
```

26 final/geom/commonTangents.cpp

```

1
2
3 vector<Line> commonTangents(pt A, dbl rA, pt B, dbl ←
    rB) {
4     vector<Line> res;
5     pt C = B - A;
6     dbl z = C.len2();
    
```

```

7     for (int i = -1; i <= 1; i += 2) {
8         for (int j = -1; j <= 1; j += 2) {
9             dbl r = rB * j - rA * i;
10            dbl d = z - r * r;
11            if (ls(d, 0)) continue;
12            d = sqrt(max(0.01, d));
13            pt magic = pt(r, d) / z;
14            pt v(magic % C, magic * C);
15            dbl CC = (rA * i - v % A) / v.len2();
16            pt O = v * -CC;
17            res.pb(Line(O, O + v.rotate()));
18        }
19    }
20    return res;
21 }
22
23 // HOW TO USE ::
24 // --- *D*-----*F*
25 // --- *...* - - - *...*
26 // --- *.....* - - *.....*
27 // --- *.....* - - *.....*
28 // --- *...A...* - - *...B...*
29 // --- *.....* - - *.....*
30 // --- *.....* - - *.....*
31 // --- *...* - - *...*
32 // --- *C*-----*E*
33 // --- res = {CE, CF, DE, DF}
    
```

27 final/geom/halfplaneIntersection.cpp

```

1 int getPart(pt v) {
2     return ls(v.y, 0) || (eq(0, v.y) && ls(v.x, 0));
3 }
4
5 int cmpV(pt a, pt b) {
6     int partA = getPart(a);
7     int partB = getPart(b);
8     if (partA < partB) return 1;
9     if (partA > partB) return -1;
10    if (eq(0, a * b)) return 0;
11    if (0 < a * b) return -1;
12    return 1;
13 }
14
15 double planeInt(vector<Line> l) {
16     sort(all(l), [](Line a, Line b) {
17         int r = cmpV(a.v, b.v);
18         if (r != 0) return r < 0;
19         return a.O % a.v.rotate() > b.O % a.v.rotate() ←
20     });
21
22     l.resize(unique(all(l), [](Line A, Line B) { ←
        return cmpV(A.v, B.v) == 0; }) - l.begin());
23     for (int i = 0; i < sz(l); i++)
24         l[i].id = i;
25
26     // if an infinite answer is possible
27     int flagUp = 0;
28     int flagDown = 0;
29     for (int i = 0; i < sz(l); i++) {
30         int part = getPart(l[i].v);
31         if (part == 1) flagUp = 1;
32         if (part == 0) flagDown = 1;
33     }
34     if (!flagUp || !flagDown) return -1;
35
36     for (int i = 0; i < sz(l); i++) {
37         pt v = l[i].v;
38         pt u = l[(i + 1) % sz(l)].v;
39         if (eq(0, v * u) && ls(v % u, 0)) {
40             pt dir = l[i].v.rotate();
41             if (le(l[(i + 1) % sz(l)].O % dir, l[i].O % ←
                dir)) return 0;
42             return -1;
43         }
44         if (ls(v * u, 0))
45             return -1;
46     }
47     // main part
48     vector<Line> st;
49     for (int tt = 0; tt < 2; tt++) {
50         for (auto L: l) {
51             for (; sz(st) >= 2 && le(st[sz(st) - 2].v * (←
                st.back() * L - st[sz(st) - 2].O), 0); st.←
                pop_back());
52                 st.pb(L);
                    
```

```

53     if (sz(st) >= 2 && le(st[sz(st) - 2].v * st.←
54         back().v, 0)) return 0; // useless line
55     }
56     vector<int> use(sz(1), -1);
57     int left = -1, right = -1;
58     for (int i = 0; i < sz(st); i++) {
59         if (use[st[i].id] == -1) {
60             use[st[i].id] = i;
61         }
62         else {
63             left = use[st[i].id];
64             right = i;
65             break;
66         }
67     }
68     vector<Line> tmp;
69     for (int i = left; i < right; i++)
70         tmp.pb(st[i]);
71     vector<pt> res;
72     for (int i = 0; i < (int)tmp.size(); i++)
73         res.pb(tmp[i] * tmp[(i + 1) % tmp.size()]);
74     double area = 0;
75     for (int i = 0; i < (int)res.size(); i++)
76         area += res[i] * res[(i + 1) % res.size()];
77     return area / 2;
78 }

```

28 final/geom/minDisc.cpp

```

1 pair<pt, dbl> minDisc(vector<pt> p) {
2     int n = p.size();
3     pt O = pt(0, 0);
4     dbl R = 0;
5     random_shuffle(all(p));
6     for (int i = 0; i < n; i++) {
7         if (ls(R, (O - p[i]).len())) {
8             O = p[i];
9             R = 0;
10            for (int j = 0; j < i; j++) {
11                if (ls(R, (O - p[j]).len())) {
12                    O = (p[i] + p[j]) / 2;
13                    R = (p[i] - p[j]).len() / 2;
14                    for (int k = 0; k < j; k++) {
15                        if (ls(R, (O - p[k]).len())) {
16                            Line l1((p[i] + p[j]) / 2, (p[i] + p[j]←
17                                )) / 2 + (p[i] - p[j]).rotate());
18                            Line l2((p[k] + p[j]) / 2, (p[k] + p[j]←
19                                )) / 2 + (p[k] - p[j]).rotate());
20                            O = l1 * l2;
21                            R = (p[i] - O).len();
22                        }
23                    }
24                }
25            }
26        }
27        return {O, R};
28    }

```

29 final/geom/convexHull3D-N2.cpp

```

1 struct Plane {
2     pt O, v;
3     vector<int> id;
4 };
5
6 vector<Plane> convexHull3(vector<pt> p) {
7     vector<Plane> res;
8     int n = p.size();
9     for (int i = 0; i < n; i++)
10         p[i].id = i;
11     for (int i = 0; i < 4; i++) {
12         vector<pt> tmp;
13         for (int j = 0; j < 4; j++)
14             if (i != j)
15                 tmp.pb(p[j]);
16     }

```

```

17     res.pb({tmp[0], (tmp[1] - tmp[0]) * (tmp[2] - ←
18         tmp[0]), {tmp[0].id, tmp[1].id, tmp[2].id}});
19     if ((p[i] - res.back().O) % res.back().v > 0) {
20         res.back().v = res.back().v * -1;
21         swap(res.back().id[0], res.back().id[1]);
22     }
23     vector<vector<int>> use(n, vector<int>(n, 0));
24     int tmr = 0;
25     for (int i = 4; i < n; i++) {
26         int cur = 0;
27         tmr++;
28         vector<pair<int, int>> curEdge;
29         for (int j = 0; j < sz(res); j++) {
30             if ((p[i] - res[j].O) % res[j].v > 0) {
31                 for (int t = 0; t < 3; t++) {
32                     int v = res[j].id[t];
33                     int u = res[j].id[(t + 1) % 3];
34                     use[v][u] = tmr;
35                     curEdge.pb({v, u});
36                 }
37             }
38             else {
39                 res[cur++] = res[j];
40             }
41         }
42         res.resize(cur);
43         for (auto x: curEdge) {
44             if (use[x.S][x.F] == tmr) continue;
45             res.pb({p[i], (p[x.F] - p[i]) * (p[x.S] - p[i]←
46                 )}, {x.F, x.S, i});
47         }
48         return res;
49     }
50 }
51 // plane in 3d
52 // (A, v) * (B, u) -> (O, n)
53
54 pt n = v * u;
55 pt m = v * n;
56 double t = (B - A) % u / (u % m);
57 pt O = A - m * t;

```

30 final/geom/convexDynamic.cpp

```

1 struct convex {
2     map<ll, ll> M;
3     bool get(int x, int y) {
4         if (M.size() == 0)
5             return false;
6         if (M.count(x))
7             return M[x] >= y;
8         if (x < M.begin()->first || x > M.rbegin()->←
9             first)
10             return false;
11
12         auto it1 = M.lower_bound(x), it2 = it1;
13         it1--;
14
15         return pt(pt(*it1), pt(x, y)) % pt(pt(*it1), pt←
16             (*it2)) >= 0;
17     }
18     void add(int x, int y) {
19         if (get(x, y)) return;
20
21         pt P(x, y);
22         M[x] = y;
23
24         auto it = M.lower_bound(x), it1 = it;
25         it1--;
26         auto it2 = it1;
27         it2--;
28
29         if (it != M.begin() && it1 != M.begin()) {
30             while (it1 != M.begin() && (pt(pt(*it2), pt(*←
31                 it1)) % pt(pt(*it1), P)) >= 0) {
32                 M.erase(it1);
33                 it1 = it2;
34                 it2--;
35             }
36             it1 = it, it1++;
37             if (it1 == M.end()) return;
38             it2 = it1, it2++;
39
40             if (it1 != M.end() && it2 != M.end()) {

```

```

39     while (it2 != M.end() && (pt(P, pt(*it1)) % pt(
40         (pt(*it1), pt(*it2))) >= 0) {
41         M.erase(it1);
42         it1 = it2;
43         it2++;
44     }
45 }
46 } H, J;
47
48 int solve() {
49     int q;
50     cin >> q;
51     while (q-- > 0) {
52         int t, x, y;
53         cin >> t >> x >> y;
54         if (t == 1) {
55             H.add(x, y);
56             J.add(x, -y);
57         }
58         else {
59             if (H.get(x, y) && J.get(x, -y))
60                 puts("YES");
61             else
62                 puts("NO");
63         }
64     }
65     return 0;
66 }
    
```

31 final/geom/polygonArcCut.cpp

```

1 struct Meta {
2     int type; // 0 - seg, 1 - circle
3     pt O;
4     dbl R;
5 };
6
7 const Meta SEG = {0, pt(0, 0), 0};
8
9 vector<pair<pt, Meta>> cut(vector<pair<pt, Meta>> p,
10     Line l) {
11     vector<pair<pt, Meta>> res;
12     int n = p.size();
13     for (int i = 0; i < n; i++) {
14         pt A = p[i].F;
15         pt B = p[(i + 1) % n].F;
16         if (le(0, l.v * (A - l.O))) {
17             if (eq(0, l.v * (A - l.O)) && p[i].S.type == 1)
18                 res.pb({A, SEG});
19             else
20                 res.pb(p[i]);
21         }
22         if (p[i].S.type == 0) {
23             if (sign(l.v * (A - l.O)) * sign(l.v * (B - l.O))
24                 == -1) {
25                 pt FF = Line(A, B) * l;
26                 res.pb(make_pair(FF, SEG));
27             }
28         }
29         else {
30             pt E, F;
31             if (intCL(p[i].S.O, p[i].S.R, l, E, F)) {
32                 if (onArc(p[i].S.O, A, E, B))
33                     res.pb({E, SEG});
34                 if (onArc(p[i].S.O, A, F, B))
35                     res.pb({F, p[i].S});
36             }
37         }
38     }
39     return res;
    
```

32 final/geom/polygonTangent.cpp

```

1 pt tangent(vector<pt>& p, pt O, int cof) {
2     int step = 1;
3     for (; step < (int)p.size(); step *= 2);
4     int pos = 0;
    
```

```

5     int n = p.size();
6     for (; step > 0; step /= 2) {
7         int best = pos;
8         for (int dx = -1; dx <= 1; dx += 2) {
9             int id = ((pos + step * dx) % n + n) % n;
10             if ((p[id] - O) * (p[best] - O) * cof > 0)
11                 best = id;
12         }
13         pos = best;
14     }
15     return p[pos];
16 }
    
```

33 final/geom/checkPlaneInt.cpp

```

1 bool eq(dbl A, dbl B) { return abs(A - B) < 1e-9; }
2
3 bool ls(dbl A, dbl B) { return A < B && !eq(A, B); }
4
5 bool le(dbl A, dbl B) { return A < B || eq(A, B); }
6 struct pt {
7     double x, y;
8     pt(double x, double y) : x(x), y(y) {}
9     pt() : pt(0, 0) {}
10 double operator%(pt b) const { return x * b.x + y *
11     * b.y; }
12 // Orientation of cross product and rotation DO
13 // matter in some algorithms
14 double operator*(pt b) const { return x * b.y - y *
15     * b.x; }
16 pt rotate() { return {y, -x}; }
17 pt operator-(pt b) const { return {x - b.x, y - b.y}; }
18 pt operator*(double t) const { return {x * t, y * t}; }
19 pt operator+(pt b) const { return {x + b.x, y + b.y}; }
20 };
21
22 // Also this is half-plane struct
23 struct Line {
24     pt O, v;
25
26     // Ax + By + C <= 0
27     Line(double A, double B, double C) {
28         double l = sqrt(A * A + B * B);
29         A /= l, B /= l, C /= l;
30         O = pt(-A * C, -B * C);
31         v = pt(-B, A);
32     }
33 // intersection with l
34 pt operator*(Line l) {
35     pt u = l.v.rotate();
36     dbl t = (l.O - O) % u / (v % u);
37     return O + v * t;
38 }
39 // Half-plane with point O on the border,
40 // everything to the LEFT of direction vector v is
41 // inside
42 Line(pt O, pt v) : O(O), v(v) {}
43 };
44
45 const double EPS = 1e-14;
46 double INF = 1e50;
47
48 // vector<Line> lines{
49 //     Line(pt(0, 0), pt(0, -1)),
50 //     Line(pt(0, 0), pt(-1, 0)),
51 //     Line(pt(1, 1), pt(0, 1)),
52 // };
53 // checkPoint(lines, p) == true
54 // Intersection of lines is rectangle of set o
55 // Time complexity is O(n)
56 bool checkPoint(vector<Line> &l, pt &ret) {
57     random_shuffle(l.begin(), l.end());
58     pt A = l[0].O;
59     for (int i = 1; i < l.size(); i++) {
60         if (l[i].v * (A - l[i].O) < -EPS) {
61             double mn = -INF;
62             double mx = INF;
63             for (int j = 0; j < i; j++) {
64                 if (abs(l[j].v * l[i].v) < EPS) {
65                     if (l[j].v % l[i].v < 0 && (l[j].O - l[i].O)
66                         % l[i].v.rotate() < EPS) {
67                         return false;
68                     }
69                 }
70             }
71         }
72     }
73     return true;
    
```

```

64         pt u = l[j].v.rotate();
65         double proj = (l[j].0 - l[i].0) % u / (l[i]←
        ].v % u);
66         if (l[i].v * l[j].v > 0) {
67             mx = min(mx, proj);
68         } else {
69             mn = max(mn, proj);
70         }
71     }
72 }
73 if (mn <= mx) {
74     A = l[i].0 + l[i].v * mn;
75 } else {
76     return false;
77 }
78 }
79 }
80 ret = A;
81 return true;
82 }

```

34 final/geom/furthestPoints.cpp

```

1 ll furthestPoints(vector<pt> p) {
2     int n = p.size();
3     int cur = 1;
4     ll answer = 0;
5     for (int i = 0; i < n; i++) {
6         for (; (p[(i + 1) % n] - p[i]) * (p[(cur + 1) % ←
        n] - p[cur]) > 0; cur = (cur + 1) % n);
7         answer = max(answer, (p[i] - p[cur]).len2());
8     }
9     return answer;
10 }

```

35 final/geom/chtDynamic.cpp

```

1 const ll is_query = -(1LL << 62);
2
3 struct Line {
4     ll m, b;
5     mutable function<const Line *(> succ;
6
7     bool operator<(const Line &rhs) const {
8         if (rhs.b != is_query) return m < rhs.m;
9         const Line *s = succ();
10        if (!s) return 0;
11        ll x = rhs.m;
12        return b - s->b < (s->m - m) * x;
13    }
14 };
15
16 struct HullDynamic : public multiset<Line> {
17     bool bad(iterator y) {
18         auto z = next(y);
19         if (y == begin()) {
20             if (z == end()) return 0;
21             return y->m == z->m && y->b <= z->b;
22         }
23         auto x = prev(y);
24         if (z == end()) return y->m == x->m && y->b <= x←
        ->b;
25         return (x->b - y->b) * (z->m - y->m) >= (y->b - ←
        z->b) * (y->m - x->m);
26     }
27
28     void insert_line(ll m, ll b) {
29         auto y = insert({m, b});
30         y->succ = [=] { return next(y) == end() ? 0 : &*←
        next(y); };
31         if (bad(y)) {
32             erase(y);
33             return;
34         }
35         while (next(y) != end() && bad(next(y))) erase(←
        next(y));
36         while (y != begin() && bad(prev(y))) erase(prev(←
        y));
37     }
38 }
39

```

```

40 ll eval(ll x) {
41     auto l = *lower_bound((Line) {x, is_query});
42     return l.m * x + l.b;
43 }
44 };

```

36 final/geom/rotate3D.cpp

```

1 // Rotate 3d point along axis on angle
2 /*
3  * 2D
4  * x' = x cos a - y sin a
5  * y' = x sin a + y cos a
6  */
7 struct quater {
8     double w, x, y, z; // w + xi + yj + zk
9     quater(double tw, const pt3 &v) : w(tw), x(v.x), y←
        (v.y), z(v.z) { }
10    quater(double tw, double tx, double ty, double tz)←
        : w(tw), x(tx), y(ty), z(tz) { }
11    pt3 vector() const {
12        return {x, y, z};
13    }
14    quater conjugate() const {
15        return {w, -x, -y, -z};
16    }
17    quater operator*(const quater &q2) {
18        return {w * q2.w - x * q2.x - y * q2.y - z * q2.←
        z, w * q2.x + x * q2.w + y * q2.z - z * q2.y, w←
        * q2.y - x * q2.z + y * q2.w + z * q2.x, w * ←
        q2.z + x * q2.y - y * q2.x + z * q2.w};
19    }
20 };
21
22 pt3 rotate(pt3 axis, pt3 p, double angle) {
23     quater q = quater(cos(angle / 2), axis * sin(angle←
        / 2));
24     return (q * quater(0, p) * q.conjugate()).vector()←
        ;
25 }

```

37 final/geom/circleInter.cpp

```

1 pair<pt, pt> inter(pt O1, double r1, pt O2, double ←
        r2) {
2     line m(O1, O2);
3     double a = (r2 * r2 - r1 * r1 + d * d) / (2 * d);
4     pt H02(-m.b, m.a);
5     H02.to_dist(-a);
6     point H(O2.x + H02.x, O2.y + H02.y);
7     double h = sqrt(r2 * r2 - a * a);
8     pt HP1(m.a, m.b);
9     HP1.to_dist(h);
10    pt P1(H.x + HP1.x, H.y + HP1.y);
11    pt P2(H.x - HP1.x, H.y - HP1.y);
12    return { P1, P2 };
13 }

```

38 final/geom/sphericalDistance.cpp

```

1 double sphericalDistance(double f1, double t1,
2     double f2, double t2, double radius) {
3     double dx = sin(t2)*cos(f2) - sin(t1)*cos(f1);
4     double dy = sin(t2)*sin(f2) - sin(t1)*sin(f1);
5     double dz = cos(t2) - cos(t1);
6     double d = sqrt(dx*dx + dy*dy + dz*dz);
7     return radius*2*asin(d/2);
8 }

```

39 final/strings/eertree.cpp

```

1 namespace eertree {
2     const int INF = 1e9;
3     const int N = 5e6 + 10;
4     char _s[N];
5     char *s = _s + 1;
6     int to[N][2];
7     int suf[N], len[N];
8     int sz, last;
9
10    const int odd = 1, even = 2, blank = 3;
11
12    void go(int &u, int pos) {
13        while (u != blank && s[pos - len[u] - 1] != s[←
14            pos]) {
15            u = suf[u];
16        }
17
18        int add(int pos) {
19            go(last, pos);
20            int u = suf[last];
21            go(u, pos);
22            int c = s[pos] - 'a';
23            int res = 0;
24            if (!to[last][c]) {
25                res = 1;
26                to[last][c] = sz;
27                len[sz] = len[last] + 2;
28                suf[sz] = to[u][c];
29                sz++;
30            }
31            last = to[last][c];
32            return res;
33        }
34
35        void init() {
36            to[blank][0] = to[blank][1] = even;
37            len[blank] = suf[blank] = INF;
38            len[even] = 0, suf[even] = odd;
39            len[odd] = -1, suf[odd] = blank;
40            last = even;
41            sz = 4;
42        }
43    }

```

40 final/strings/manacher.cpp

```

1 vector<int> Pal1(string s) {
2     int n = (int)s.size();
3     vector<int> d1(n);
4     int l = 0, r = -1;
5     for (int i = 0, k; i < n; i++) {
6         if (i > r) k = 1;
7         else k = min(d1[l + r - i], r - i);
8         while (0 <= i - k && i + k < n && s[i - k] == s[←
9             i + k]) k++;
10        d1[i] = k;
11        if (i + k - 1 > r) r = i + k - 1, l = i - k + 1;
12    }
13    return d1;
14
15    vector<int> Pal2(string s) {
16        int n = (int)s.size();
17        vector<int> d2(n);
18        int l = 0, r = -1;
19        for (int i = 0, k; i < n; i++) {
20            if (i > r) k = 0;
21            else k = min(d2[l + r - i + 1], r - i + 1);
22            while (i + k < n && i - k - 1 >= 0 && s[i + k] ←
23                == s[i - k - 1]) k++;
24            d2[i] = k;
25            if (i + k - 1 > r) l = i - k, r = i + k - 1;
26        }
27        return d2;
28    }

```

41 final/strings/sufAutomaton.cpp

```

1 namespace SA {
2     const int MAXN = 1 << 18;
3     const int SIGMA = 26;
4
5     int sz, last;
6     int nxt[MAXN][SIGMA];
7     int link[MAXN], len[MAXN], pos[MAXN];
8
9     void init() {
10        memset(nxt, -1, sizeof(nxt));
11        memset(link, -1, sizeof(link));
12        memset(len, 0, sizeof(len));
13        last = 0;
14        sz = 1;
15    }
16
17    void add(int c) {
18        int cur = sz++;
19        len[cur] = len[last] + 1;
20        pos[cur] = len[cur];
21        int p = last;
22        last = cur;
23        for (; p != -1 && nxt[p][c] == -1; p = link[p]) ←
24            nxt[p][c] = cur;
25        if (p == -1) {
26            link[cur] = 0;
27            return;
28        }
29        int q = nxt[p][c];
30        if (len[p] + 1 == len[q]) {
31            link[cur] = q;
32            return;
33        }
34        int clone = sz++;
35        memcpy(nxt[clone], nxt[q], sizeof(nxt[q]));
36        len[clone] = len[p] + 1;
37        pos[clone] = pos[q];
38        link[clone] = link[q];
39        link[q] = link[cur] = clone;
40        for (; p != -1 && nxt[p][c] == q; p = link[p]) ←
41            nxt[p][c] = clone;
42    }
43
44    int n;
45    string s;
46    int l[MAXN], r[MAXN];
47    int e[MAXN][SIGMA];
48
49    void getSufTree(string _s) {
50        memset(e, -1, sizeof(e));
51        s = _s;
52        n = s.length();
53        reverse(s.begin(), s.end());
54        init();
55        for (int i = 0; i < n; i++) add(s[i] - 'a');
56        reverse(s.begin(), s.end());
57        for (int i = 1; i < sz; i++) {
58            int j = link[i];
59            l[i] = n - pos[i] + len[j];
60            r[i] = n - pos[i] + len[i];
61            e[j][s[l[i]] - 'a'] = i;
62        }
63    }

```

42 final/strings/sufTree.cpp

```

1 const int N = 1e5, VN = 2 * N;
2
3 map<char, int> t[VN];
4 int l[VN], r[VN], p[VN], term[VN]; // ребро p[v] → ←
5 // v это отрезок [l[v], r[v]] исходной строки
6 int cc, suf[VN], vn = 2, v = 1, pos; // ←
7 // идём по ребру p[v] в v, сейчас стоим в pos
8
9 void init() {
10    for (int i = 0; i < 127; i++) t[0][i] = 1; // 0 ←
11    // фиктивная, 1 = корень
12    l[1] = -1;
13
14    void add(char c, int i, const string &s) {

```

```

13 auto new_leaf = [&](int v) {
14     p[vn] = v, l[vn] = i, r[vn] = N, t[v][c] = vn++;
15 };
16 go++;
17 if (r[v] <= pos) {
18     if (!t[v].count(c)) {
19         new_leaf(v), v = suf[v], pos = r[v];
20         goto go;
21     }
22     v = t[v][c], pos = l[v] + 1;
23 } else if (c == s[pos]) {
24     pos++;
25 } else {
26     int x = vn++;
27     l[x] = l[v], r[x] = pos, l[v] = pos;
28     p[x] = p[v], p[v] = x;
29     t[p[x]][s[l[x]]] = x, t[x][s[pos]] = v;
30     new_leaf(x);
31     v = suf[p[x]], pos = l[x];
32     while (pos < r[x])
33         v = t[v][s[pos]], pos += r[v] - l[v];
34     suf[x] = (pos == r[x] ? v : vn);
35     pos = r[v] - (pos - r[x]);
36     goto go;
37 }
38 }
39
40 int main() {
41     init();
42     string s; cin >> s;
43     s += (char)0; // term
44     for (int i = 0; i < (int)s.size(); i++) {
45         add(s[i], i, s);
46     }
47     for (int i = 1; i < vn; i++) r[i] = min(r[i], (int)s.size());
48     for (int i = 1; i < vn; i++) {
49         for (auto c : t[i]) err("%d [%d, %d] %d\n", i, l[c.second], r[c.second], c.second);
50     }
51 }

```

43 final/strings/sufArray.cpp

```

1 int n;
2 char s[N];
3 int p[N], pn[N], c[N], cn[N], cnt[N];
4 int o[N];
5 int lcp[N];
6
7 void build() {
8     for (int i = 0; i < 256; i++) cnt[i] = 0;
9     for (int i = 0; i < n; i++) cnt[(int)s[i]]++;
10    for (int i = 1; i < 256; i++) cnt[i] += cnt[i - 1];
11    for (int i = n - 1; i >= 0; i--) p[--cnt[(int)s[i]]] = i;
12    int cl = 1;
13    c[p[0]] = 0;
14    for (int i = 1; i < n; i++) {
15        cl += s[p[i]] != s[p[i - 1]];
16        c[p[i]] = cl - 1;
17    }
18
19    for (int len = 1; len < n; len <= 1) {
20        for (int i = 0; i < cl; i++) cnt[i] = 0;
21        for (int i = 0; i < n; i++) cnt[c[i]]++;
22        for (int i = 1; i < cl; i++) cnt[i] += cnt[i - 1];
23        for (int i = 0; i < n; i++) pn[i] = (p[i] - len + n) % n;
24        for (int i = n - 1; i >= 0; i--) p[--cnt[c[pn[i]]]] = pn[i];
25        cl = 1;
26        cn[p[0]] = 0;
27        for (int i = 1; i < n; i++) {
28            cl += c[p[i]] != c[p[i - 1]] || c[(p[i] + len) % n] != c[(p[i - 1] + len) % n];
29            cn[p[i]] = cl - 1;
30        }
31        for (int i = 0; i < n; i++) c[i] = cn[i];
32    }
33
34    for (int i = 0; i < n; i++) o[p[i]] = i;
35
36    int z = 0;
37    for (int i = 0; i < n; i++) {

```

```

38        int j = o[i];
39        if (j == n - 1) {
40            z = 0;
41        } else {
42            while (s[i + z] == s[p[j + 1] + z]) z++;
43        }
44        lcp[j] = z;
45        z -= !!z;
46    }
47 }

```

44 final/strings/sufArrayLinear.cpp

```

1 const int dd = (int)2e6 + 3;
2
3 ll cnt2[dd];
4 int AN;
5 int A[3 * dd + 100];
6 int cnt[dd + 1]; // Should be >= 256
7 int SA[dd + 1];
8
9 /* Used by suffix_array. */
10 void radix_pass(int* A, int AN, int* R, int RN, int* D) {
11     memset(cnt, 0, sizeof(int) * (AN + 1));
12     int* C = cnt + 1;
13     for (int i = 0; i < RN; i++) ++C[A[R[i]]];
14     for (int i = -1, v = 0; i <= AN && v < RN; v += C[i]++) swap(v, C[i]);
15     for (int i = 0; i < RN; i++) D[C[A[R[i]]]++] = R[i];
16 }
17
18 /* DC3 in O(N) using 20N bytes of memory. Stores the suffix array of the string
19 * [A,A+AN) into SA where SA[i] (0<=i<=AN) gives the starting position of the
20 * i-th least suffix of A (including the empty suffix).
21 */
22 void suffix_array(int* A, int AN) {
23     // Base case... length 1 string.
24     if (!AN) {
25         SA[0] = 0;
26     } else if (AN == 1) {
27         SA[0] = 1; SA[1] = 0;
28         return;
29     }
30
31     // Sort all strings of length 3 starting at non-multiples of 3 into R.
32     int RN = 0;
33     int* SUBA = A + AN + 2;
34     int* R = SUBA + AN + 2;
35     for (int i = 1; i < AN; i += 3) SUBA[RN++] = i;
36     for (int i = 2; i < AN; i += 3) SUBA[RN++] = i;
37     A[AN + 1] = A[AN] = -1;
38     radix_pass(A + 2, AN - 2, SUBA, RN, R);
39     radix_pass(A + 1, AN - 1, R, RN, SUBA);
40     radix_pass(A + 0, AN - 0, SUBA, RN, R);
41
42     // Compute the relabel array if we need to recursively solve for the
43     // non-multiples.
44     int resfix, resmul, v;
45     if (AN % 3 == 1) {
46         resfix = 1; resmul = RN >> 1;
47     } else {
48         resfix = 2; resmul = RN + 1 >> 1;
49     }
50     for (int i = v = 0; i < RN; i++) {
51         v += i && (A[R[i - 1] + 0] != A[R[i] + 0] ||
52             A[R[i - 1] + 1] != A[R[i] + 1] ||
53             A[R[i - 1] + 2] != A[R[i] + 2]);
54         SUBA[R[i] / 3 + (R[i] % 3 == resfix) * resmul] = v;
55     }
56
57     // Recursively solve if needed to compute relative ranks in the final suffix
58     // array of all non-multiples.
59     if (v + 1 != RN) {
60         suffix_array(SUBA, RN);
61         SA[0] = AN;
62         for (int i = 1; i <= RN; i++) {
63             SA[i] = SA[i] < resmul ? 3 * SA[i] + (resfix == 1 ? 2 : 1) :
64                 3 * (SA[i] - resmul) + resfix;

```

45 final/strings/duval.cpp

```

65 }
66 } else {
67     SA[0] = AN;
68     memcpy(SA + 1, R, sizeof(int) * RN);
69 }
70
71 // Compute the relative ordering of the multiples.
72 int NMN = RN;
73 for(int i = RN = 0; i <= NMN; i++) {
74     if(SA[i] % 3 == 1) {
75         SUBA[RN++] = SA[i] - 1;
76     }
77 }
78 radix_pass(A, AN, SUBA, RN, R);
79
80 // Compute the reverse SA for what we know so far.
81 for(int i = 0; i <= NMN; i++) {
82     SUBA[SA[i]] = i;
83 }
84
85 // Merge the orderings.
86 int ii = RN - 1;
87 int jj = NMN;
88 int pos;
89 for(pos = AN; ii >= 0; pos--) {
90     int i = R[ii];
91     int j = SA[jj];
92     int v = A[i] - A[j];
93     if(!v) {
94         if(j % 3 == 1) {
95             v = SUBA[i + 1] - SUBA[j + 1];
96         } else {
97             v = A[i + 1] - A[j + 1];
98         }
99         if(!v) v = SUBA[i + 2] - SUBA[j + 2];
100     }
101     SA[pos] = v < 0 ? SA[jj--] : R[ii--];
102 }
103 }
104
105 char s[dd + 1];
106
107 /* Copies the string in s into A and reduces the ↵
108    characters as needed. */
109 void prep_string() {
110     int v = AN = 0;
111     memset(cnt, 0, 256 * sizeof(int));
112     for(char* ss = s; *ss; ++ss, ++AN) cnt[*ss]++;
113     for(int i = 0; i < AN; i++) cnt[s[i]]++;
114     for(int i = 0; i < 256; i++) cnt[i] = cnt[i] ? v++ ↵
115         : -1;
116     for(int i = 0; i < AN; i++) A[i] = cnt[s[i]];
117 }
118
119 /* Computes the reverse SA index. REVS_A[i] gives the ↵
120    index of the suffix
121    * starting at i in the SA array. In other words, ↵
122    REVS_A[i] gives the number of
123    * suffixes before the suffix starting at i. This ↵
124    can be useful in itself but
125    * is also used for compute_lcp().
126    */
127 int REVS_A[dd + 1];
128 void compute_reverse_sa() {
129     for(int i = 0; i <= AN; i++) {
130         REVS_A[SA[i]] = i;
131     }
132 }
133
134 /* Computes the longest common prefix between ↵
135    adjacent suffixes. LCP[i] gives
136    * the longest common suffix between the suffix ↵
137    starting at i and the next
138    * smallest suffix. Runs in O(N) time.
139    */
140 int LCP[dd + 1];
141 void compute_lcp() {
142     int len = 0;
143     for(int i = 0; i < AN; i++, len = max(0, len - 1)) ↵
144     {
145         int s = REVS_A[i];
146         int j = SA[s - 1];
147         for(; i + len < AN && j + len < AN && A[i + len] ↵
148             == A[j + len]; len++);
149         LCP[s] = len;
150     }
151 }

```

```

1 void duval(string s) {
2     int n = (int) s.length();
3     int i=0;
4     while (i < n) {
5         int j=i+1, k=i;
6         while (j < n && s[k] <= s[j]) {
7             if (s[k] < s[j]) {
8                 k = i;
9             }
10            ++k;
11            ++j;
12        }
13        while (i <= k) {
14            cout << s.substr (i, j-k) << ' ';
15            i += j - k;
16        }
17    }
18 }

```


46 final/graphs/alphaBeta.cpp

```

1 int alphabeta(state s, int alpha, int beta) {
2     if (s.finished()) return s.score();
3     for (state t : s.next()) {
4         alpha = max(alpha, -alphabeta(t, -beta, -alpha))
5     }
6     if (alpha >= beta) break;
7     return alpha;
8 }

```

```

66     tr.upd(in[v], out[v], in[sdom[v]]);
67 }
68 for (int i = 0; i < tmr; i++) {
69     int v = rev[i];
70     if (i == 0) {
71         dom[v] = v;
72         h[v] = 0;
73     } else {
74         dom[v] = lca(sdom[v], pr[v]);
75         h[v] = h[dom[v]] + 1;
76     }
77     p[v][0] = dom[v];
78     for (int j = 1; j < K; j++) p[v][j] = p[p[v][j-1]][j-1];
79 }
80 for (int i = 0; i < n; i++) if (in[i] == -1) dom[i] = -1;
81 }
82 }

```

47 final/graphs/dominatorTree.cpp

```

1 namespace domtree {
2     const int K = 18;
3     const int N = 1 << K;
4
5     int n, root;
6     vector<int> e[N], g[N];
7     int sdom[N], dom[N];
8     int p[N][K], h[N], pr[N];
9     int in[N], out[N], tmr, rev[N];
10
11 void init(int _n, int _root) {
12     n = _n;
13     root = _root;
14     tmr = 0;
15     for (int i = 0; i < n; i++) {
16         e[i].clear();
17         g[i].clear();
18         in[i] = -1;
19     }
20 }
21
22 void addEdge(int u, int v) {
23     e[u].push_back(v);
24     g[v].push_back(u);
25 }
26
27 void dfs(int v) {
28     in[v] = tmr++;
29     for (int to : e[v]) {
30         if (in[to] != -1) continue;
31         pr[to] = v;
32         dfs(to);
33     }
34     out[v] = tmr - 1;
35 }
36
37 int lca(int u, int v) {
38     if (h[u] < h[v]) swap(u, v);
39     for (int i = 0; i < K; i++) if ((h[u] - h[v]) & (1 << i)) u = p[u][i];
40     if (u == v) return u;
41     for (int i = K - 1; i >= 0; i--) {
42         if (p[u][i] != p[v][i]) {
43             u = p[u][i];
44             v = p[v][i];
45         }
46     }
47     return p[u][0];
48 }
49
50 void solve(int _n, int _root, vector<pair<int, int>> &_edges) {
51     init(_n, _root);
52     for (auto ed : _edges) addEdge(ed.first, ed.second);
53
54     dfs(root);
55     for (int i = 0; i < n; i++) if (in[i] != -1) rev[i] = i;
56     segtree tr(tmr); // a[i] := min(a[i], x) and return a[i]
57     for (int i = tmr - 1; i >= 0; i--) {
58         int v = rev[i];
59         int cur = i;
60         for (int to : g[v]) {
61             if (in[to] == -1) continue;
62             if (in[to] < in[v]) cur = min(cur, in[to]);
63             else cur = min(cur, tr.get(in[to]));
64         }
65         sdom[v] = rev[cur];

```

48 final/graphs/generalMatching.cpp

```

1 //COPYPASTED FROM E-MAXX
2 namespace GeneralMatching {
3     const int MAXN = 256;
4     int n;
5     vector<int> g[MAXN];
6     int match[MAXN], p[MAXN], base[MAXN], q[MAXN];
7     bool used[MAXN], blossom[MAXN];
8
9     int lca(int a, int b) {
10         bool used[MAXN] = { 0 };
11         for (;;) {
12             a = base[a];
13             used[a] = true;
14             if (match[a] == -1) break;
15             a = p[match[a]];
16         }
17         for (;;) {
18             b = base[b];
19             if (used[b]) return b;
20             b = p[match[b]];
21         }
22     }
23
24 void mark_path(int v, int b, int children) {
25     while (base[v] != b) {
26         blossom[base[v]] = blossom[base[match[v]]] = true;
27         p[v] = children;
28         children = match[v];
29         v = p[match[v]];
30     }
31 }
32
33 int find_path(int root) {
34     memset(used, 0, sizeof used);
35     memset(p, -1, sizeof p);
36     for (int i = 0; i < n; ++i)
37         base[i] = i;
38
39     used[root] = true;
40     int qh = 0, qt = 0;
41     q[qt++] = root;
42     while (qh < qt) {
43         int v = q[qh++];
44         for (size_t i = 0; i < g[v].size(); ++i) {
45             int to = g[v][i];
46             if (base[v] == base[to] || match[v] == to) continue;
47             if (to == root || (match[to] != -1 && p[match[to]] != -1)) {
48                 int curbase = lca(v, to);
49                 memset(blossom, 0, sizeof blossom);
50                 mark_path(v, curbase, to);
51                 mark_path(to, curbase, v);
52                 for (int i = 0; i < n; ++i)
53                     if (blossom[base[i]]) {
54                         base[i] = curbase;
55                         if (!used[i]) {
56                             used[i] = true;
57                             q[qt++] = i;
58                         }
59                     }
60             }
61             else if (p[to] == -1) {
62                 p[to] = v;

```

```

63         if (match[to] == -1)
64             return to;
65         to = match[to];
66         used[to] = true;
67         q[qt++] = to;
68     }
69 }
70 }
71 return -1;
72 }
73
74 vector<pair<int, int>> solve(int _n, vector<pair<int,
75 int, int>> edges) {
76     n = _n;
77     for (int i = 0; i < n; i++) g[i].clear();
78     for (auto o : edges) {
79         g[o.first].push_back(o.second);
80         g[o.second].push_back(o.first);
81     }
82     memset(match, -1, sizeof match);
83     for (int i = 0; i < n; ++i) {
84         if (match[i] == -1) {
85             int v = find_path(i);
86             while (v != -1) {
87                 int pv = p[v], ppv = match[pv];
88                 match[v] = pv, match[pv] = v;
89                 v = ppv;
90             }
91         }
92     }
93     vector<pair<int, int>> ans;
94     for (int i = 0; i < n; i++) {
95         if (match[i] > i) {
96             ans.push_back(make_pair(i, match[i]));
97         }
98     }
99     return ans;
100 }

```

49 final/graphs/heavyLight.cpp

```

1 namespace hld {
2     const int N = 1 << 17;
3     int par[N], heavy[N], h[N];
4     int root[N], pos[N];
5     int n;
6     vector<vector<int>> e;
7     segtree tree;
8
9     int dfs(int v) {
10         int sz = 1, mx = 0;
11         for (int to : e[v]) {
12             if (to == par[v]) continue;
13             par[to] = v;
14             h[to] = h[v] + 1;
15             int cur = dfs(to);
16             if (cur > mx) heavy[v] = to, mx = cur;
17             sz += cur;
18         }
19         return sz;
20     }
21
22     template <typename T>
23     void path(int u, int v, T op) {
24         for (; root[u] != root[v]; v = par[root[v]]) {
25             if (h[root[u]] > h[root[v]]) swap(u, v);
26             op(pos[root[v]], pos[v] + 1);
27         }
28         if (h[u] > h[v]) swap(u, v);
29         op(pos[u], pos[v] + 1);
30     }
31
32     void init(vector<vector<int>> _e) {
33         e = _e;
34         n = e.size();
35         tree = segtree(n);
36         memset(heavy, -1, sizeof(heavy[0]) * n);
37         par[0] = -1;
38         h[0] = 0;
39         dfs(0);
40         for (int i = 0, cpos = 0; i < n; i++) {
41             if (par[i] == -1 || heavy[par[i]] != i) {
42                 for (int j = i; j != -1; j = heavy[j]) {
43                     root[j] = i;
44                     pos[j] = cpos++;
45                 }

```

```

46         }
47     }
48 }
49
50 void add(int v, int x) {
51     tree.add(pos[v], x);
52 }
53
54 int get(int u, int v) {
55     int res = 0;
56     path(u, v, [&](int l, int r) {
57         res = max(res, tree.get(l, r));
58     });
59     return res;
60 }
61 }

```

50 final/graphs/hungary.cpp

```

1 namespace hungary
2 {
3     const int N = 210;
4
5     int a[N][N];
6     int ans[N];
7
8     int calc(int n, int m)
9     {
10         ++n, ++m;
11         vi u(n), v(m), p(m), prev(m);
12         for (int i = 1; i < n; ++i)
13         {
14             p[0] = i;
15             int x = 0;
16             vi mn(m, inf);
17             vi was(m, 0);
18             while (p[x])
19             {
20                 was[x] = 1;
21                 int ii = p[x], dd = inf, y = 0;
22                 for (int j = 1; j < m; ++j) if (!was[j])
23                 {
24                     int cur = a[ii][j] - u[ii] - v[j];
25                     if (cur < mn[j]) mn[j] = cur, prev[j] = x;
26                     if (mn[j] < dd) dd = mn[j], y = j;
27                 }
28                 forn(j, m)
29                 {
30                     if (was[j]) u[p[j]] += dd, v[j] -= dd;
31                     else mn[j] -= dd;
32                 }
33                 x = y;
34             }
35             while (x)
36             {
37                 int y = prev[x];
38                 p[x] = p[y];
39                 x = y;
40             }
41         }
42         for (int j = 1; j < m; ++j)
43         {
44             ans[p[j]] = j;
45         }
46         return -v[0];
47     }
48 }
49 // HOW TO USE ::
50 // --- set values to a[1..n][1..m] (n <= m)
51 // --- run calc(n, m) to find MINIMUM
52 // --- to restore permutation use ans[]
53 // --- everything works on negative numbers
54 // !! i don't understand this code, it's ←
55 // copped from e-maxx (and rewritten by enot110←

```

51 final/graphs/minCost.cpp

```

1 ll findflow(int s, int t) {
2     ll cost = 0;

```

```

3  ll flow = 0;
4
5  forn(i, N) G[i] = inf;
6
7  queue<int> q;
8
9  q.push(s);
10 used[s] = true;
11 G[s] = 0;
12
13 while (q.size()) {
14     int v = q.front();
15     used[v] = false;
16     q.pop();
17
18     forn(i, E[v].size()) {
19         auto &e = E[v][i];
20         if (e.f < e.c && G[e.to] > G[v] + e.w) {
21             G[e.to] = G[v] + e.w;
22             if (!used[e.to]) {
23                 q.push(e.to);
24                 used[e.to] = true;
25             }
26         }
27     }
28 }
29
30 while (1) {
31     forn(i, N)
32         d[i] = inf, p[i] = { -1, -1 }, used[i] = 0;
33
34     d[s] = 0;
35     while (1) {
36         int v = -1;
37         forn(i, N) {
38             if (!used[i] && d[i] != inf && (v == -1 || d[
39 i] < d[v]))
40                 v = i;
41         }
42         if (v == -1)
43             break;
44         used[v] = 1;
45
46         forn(i, E[v].size()) {
47             auto &e = E[v][i];
48             if (e.f < e.c && d[e.to] > d[v] + e.w + G[v] ←
49 - G[e.to]) {
50                 p[e.to] = mp(v, i);
51                 d[e.to] = d[v] + e.w + G[v] - G[e.to];
52             }
53         }
54     }
55     if (p[t].first == -1) {
56         break;
57     }
58     int add = inf;
59     for (int i = t; p[i].first != -1; i = p[i].first ←
60 ) {
61         add = min(add, E[p[i].first][p[i].second].c - ←
62 E[p[i].first][p[i].second].f);
63     }
64     for (int i = t; p[i].first != -1; i = p[i].first ←
65 ) {
66         auto &e = E[p[i].first][p[i].second];
67         cost += lll * add * e.w;
68         e.f += add;
69         E[e.to][e.back].f -= add;
70     }
71     flow += add;
72     if (add == 0)
73         break;
74     forn(i, N)
75         G[i] += d[i];
76 }
77 return cost;
78 }

```

52 final/graphs/minCostNegCycle.cpp

```

1 struct Edge {
2     int from, to, cap, flow;
3     double cost;
4 };
5
6

```

```

7 struct Graph {
8     int n;
9     vector<Edge> edges;
10    vector<vector<int>> e;
11
12    Graph(int _n) {
13        n = _n;
14        e.resize(n);
15    }
16
17    void addEdge(int from, int to, int cap, double ←
18 cost) {
19        e[from].push_back(edges.size());
20        edges.push_back({ from, to, cap, 0, cost });
21        e[to].push_back(edges.size());
22        edges.push_back({ to, from, 0, 0, -cost });
23    }
24
25    void maxflow() {
26        while (1) {
27            queue<int> q;
28            vector<int> d(n, INF);
29            vector<int> pr(n, -1);
30            q.push(0);
31            d[0] = 0;
32            while (!q.empty()) {
33                int v = q.front();
34                q.pop();
35                for (int i = 0; i < (int)e[v].size(); i++) {
36                    Edge cur = edges[e[v][i]];
37                    if (d[cur.to] > d[v] + 1 && cur.flow < cur ←
38 .cap) {
39                        d[cur.to] = d[v] + 1;
40                        pr[cur.to] = e[v][i];
41                        q.push(cur.to);
42                    }
43                }
44                if (d[n - 1] == INF) break;
45                int v = n - 1;
46                while (v) {
47                    edges[pr[v]].flow++;
48                    edges[pr[v] ^ 1].flow--;
49                    v = edges[pr[v]].from;
50                }
51            }
52        }
53
54        bool findcycle() {
55            int iters = n;
56            vector<int> changed;
57            for (int i = 0; i < n; i++) changed.push_back(i) ←
58 ;
59
60            vector<vector<double>> d(iters + 1, vector<←
61 double>(n, INF));
62            vector<vector<int>> p(iters + 1, vector<int>(n, ←
63 -1));
64            d[0].assign(n, 0);
65            for (int it = 0; it < iters; it++) {
66                d[it + 1] = d[it];
67                vector<int> nchanged(n, 0);
68                for (int v : changed) {
69                    for (int id : e[v]) {
70                        Edge cur = edges[id];
71                        if (d[it + 1][cur.to] > d[it][v] + cur. ←
72 cost && cur.flow < cur.cap) {
73                            d[it + 1][cur.to] = d[it][v] + cur.cost;
74                            p[it + 1][cur.to] = id;
75                            nchanged[cur.to] = 1;
76                        }
77                    }
78                }
79                changed.clear();
80                for (int i = 0; i < n; i++) if (nchanged[i]) ←
81 changed.push_back(i);
82            }
83            if (changed.empty()) return 0;
84
85            int bestU = 0, bestK = 1;
86            double bestAns = INF;
87            for (int u = 0; u < n; u++) {
88                double curMax = -INF;
89                for (int k = 0; k < iters; k++) {
90                    double curVal = (d[iters][u] - d[k][u]) / (←
91 iters - k);
92                    curMax = max(curMax, curVal);
93                }
94                if (bestAns > curMax) {
95                    bestAns = curMax;
96                    bestU = u;
97                }
98            }
99        }
100    }

```

```

92 int v = bestU;
93 int it = iters;
94 vector<int> was(n, -1);
95 while (was[v] == -1) {
96     was[v] = it;
97     v = edges[p[it][v]].from;
98     it--;
99 }
100 int vv = v;
101 it = was[v];
102 double sum = 0;
103 do {
104     edges[p[it][v]].flow++;
105     sum += edges[p[it][v]].cost;
106     edges[p[it][v] ^ 1].flow--;
107     v = edges[p[it][v]].from;
108     it--;
109 } while (v != vv);
110 return 1;
111 }
112 }
113 };
    
```

53 final/graphs/retro.cpp

```

1 namespace retro
2 {
3     const int N = 4e5 + 10;
4
5     vi v[N];
6     vi vrev[N];
7
8     void add(int x, int y)
9     {
10         v[x].pb(y);
11         vrev[y].pb(x);
12     }
13
14     const int UD = 0;
15     const int WIN = 1;
16     const int LOSE = 2;
17
18     int res[N];
19     int moves[N];
20     int deg[N];
21     int q[N], st, en;
22
23     void calc(int n)
24     {
25         forn(i, n) deg[i] = sz(v[i]);
26         st = en = 0;
27         forn(i, n) if (!deg[i])
28         {
29             q[en++] = i;
30             res[i] = LOSE;
31         }
32         while (st < en)
33         {
34             int x = q[st++];
35             for (int y : vrev[x])
36             {
37                 if (res[y] == UD && (res[x] == LOSE || (--deg[y] == 0 && res[x] == WIN)))
38                 {
39                     res[y] = 3 - res[x];
40                     moves[y] = moves[x] + 1;
41                     q[en++] = y;
42                 }
43             }
44         }
45     }
46 }
    
```

54 final/graphs/mincut.cpp

```

1 const int MAXN = 500;
2 int n, g[MAXN][MAXN];
3 int best_cost = 1000000000;
4 vector<int> best_cut;
5
6 void mincut() {
    
```

```

7     vector<int> v[MAXN];
8     for (int i=0; i<n; ++i)
9         v[i].assign(1, i);
10    int w[MAXN];
11    bool exist[MAXN], in_a[MAXN];
12    memset(exist, true, sizeof exist);
13    for (int ph=0; ph<n-1; ++ph) {
14        memset(in_a, false, sizeof in_a);
15        memset(w, 0, sizeof w);
16        for (int it=0, prev; it<n-ph; ++it) {
17            int sel = -1;
18            for (int i=0; i<n; ++i)
19                if (exist[i] && !in_a[i] && (sel == -1 || w[sel] < w[i]))
20                    sel = i;
21            if (it == n-ph-1) {
22                if (w[sel] < best_cost)
23                    best_cost = w[sel], best_cut = v[sel];
24                v[prev].insert(v[prev].end(), v[sel].begin(), v[sel].end());
25                for (int i=0; i<n; ++i)
26                    g[prev][i] = g[i][prev] += g[sel][i];
27                exist[sel] = false;
28            }
29            else {
30                in_a[sel] = true;
31                for (int i=0; i<n; ++i)
32                    w[i] += g[sel][i];
33                prev = sel;
34            }
35        }
36    }
37 }
    
```

55 final/graphs/twoChineseFast.cpp

```

1 namespace twoc {
2     struct Heap {
3         static Heap* null;
4         ll x, xadd;
5         int ver, h;
6         /* ANS */ int ei;
7         Heap *l, *r;
8         Heap(ll xx, int vv) : x(xx), xadd(0), ver(vv), h(1), l(null), r(null) {}
9         Heap(const char*) : x(0), xadd(0), ver(0), h(0), l(this), r(this) {}
10        void add(ll a) { x += a; xadd += a; }
11        void push() {
12            if (l != null) l->add(xadd);
13            if (r != null) r->add(xadd);
14            xadd = 0;
15        }
16    };
17    Heap *Heap::null = new Heap("wqeqw");
18    Heap* merge(Heap *l, Heap *r) {
19        if (l == Heap::null) return r;
20        if (r == Heap::null) return l;
21        l->push(); r->push();
22        if (l->x > r->x)
23            swap(l, r);
24        l->r = merge(l->r, r);
25        if (l->l->h < l->r->h)
26            swap(l->l, l->r);
27        l->h = l->r->h + 1;
28        return l;
29    }
30    Heap *pop(Heap *h) {
31        h->push();
32        return merge(h->l, h->r);
33    }
34    const int N = 666666;
35    struct DSU {
36        int p[N];
37        void init(int nn) { iota(p, p + nn, 0); }
38        int get(int x) { return p[x] == x ? x : p[x] = get(p[x]); }
39        void merge(int x, int y) { p[get(y)] = get(x); }
40    } dsu;
41    Heap *eb[N];
42    int n;
43    /* ANS */ struct Edge {
44        /* ANS */ int x, y;
45        /* ANS */ ll c;
46        /* ANS */ };
47    /* ANS */ vector<Edge> edges;
48    /* ANS */ int answer[N];
    
```

```

49 void init(int nn) {
50     n = nn;
51     dsu.init(n);
52     fill(eb, eb + n, Heap::null);
53     edges.clear();
54 }
55 void addEdge(int x, int y, ll c) {
56     Heap *h = new Heap(c, x);
57     /* ANS */ h->ei = sz(edges);
58     /* ANS */ edges.push_back({x, y, c});
59     eb[y] = merge(eb[y], h);
60 }
61 ll solve(int root = 0) {
62     ll ans = 0;
63     static int done[N], pv[N];
64     memset(done, 0, sizeof(int) * n);
65     done[root] = 1;
66     int tt = 1;
67     /* ANS */ int cnum = 0;
68     /* ANS */ static vector<ipair> eout[N];
69     /* ANS */ for (int i = 0; i < n; ++i) eout[i].clear();
70     for (int i = 0; i < n; ++i) {
71         int v = dsu.get(i);
72         if (done[v])
73             continue;
74         ++tt;
75         while (true) {
76             done[v] = tt;
77             int nv = -1;
78             while (eb[v] != Heap::null) {
79                 nv = dsu.get(eb[v]->ver);
80                 if (nv == v) {
81                     eb[v] = pop(eb[v]);
82                     continue;
83                 }
84                 break;
85             }
86             if (nv == -1)
87                 return LINF;
88             ans += eb[v]->x;
89             eb[v]->add(-eb[v]->x);
90             /* ANS */ int ei = eb[v]->ei;
91             /* ANS */ eout[edges[ei].x].push_back({++cnum, ei});
92             if (!done[nv]) {
93                 pv[v] = nv;
94                 v = nv;
95                 continue;
96             }
97             if (done[nv] != tt)
98                 break;
99             int v1 = nv;
100             while (v1 != v) {
101                 eb[v] = merge(eb[v], eb[v1]);
102                 dsu.merge(v, v1);
103                 v1 = dsu.get(pv[v1]);
104             }
105         }
106     }
107     /* ANS */ memset(answer, -1, sizeof(int) * n);
108     /* ANS */ answer[root] = 0;
109     /* ANS */ set<ipair> es(all(eout[root]));
110     /* ANS */ while (!es.empty()) {
111         /* ANS */ auto it = es.begin();
112         /* ANS */ int ei = it->second;
113         /* ANS */ es.erase(it);
114         /* ANS */ int nv = edges[ei].y;
115         /* ANS */ if (answer[nv] != -1)
116             continue;
117         /* ANS */ answer[nv] = ei;
118         /* ANS */ es.insert(all(eout[nv]));
119     }
120     /* ANS */ answer[root] = -1;
121     return ans;
122 }
123 /* Usage: twoc::init(vertex_count);
124 * twoc::addEdge(v1, v2, cost);
125 * twoc::solve(root); - returns cost or LINF
126 * twoc::answer contains index of ingoing edge for<
127 * each vertex
128 */

```

56 final/graphs/linkcut.cpp

```

1 #include <iostream>

```

```

2 #include <cstdio>
3 #include <cassert>
4
5 using namespace std;
6
7 // BEGIN ALGO
8
9 const int MAXN = 110000;
10
11 typedef struct _node{
12     _node *l, *r, *p, *pp;
13     int size; bool rev;
14     _node();
15     explicit _node(nullptr_t){
16         l = r = p = pp = this;
17         size = rev = 0;
18     }
19     void push(){
20         if (rev){
21             l->rev ^= 1; r->rev ^= 1;
22             rev = 0; swap(l, r);
23         }
24     }
25     void update();
26 }* node;
27 node None = new _node(nullptr);
28 node v2n[MAXN];
29 _node::_node(){
30     l = r = p = pp = None;
31     size = 1; rev = false;
32 }
33 void _node::update(){
34     size = (this != None) + l->size + r->size;
35     l->p = r->p = this;
36 }
37 void rotate(node v){
38     assert(v != None && v->p != None);
39     assert(!v->rev); assert(!v->p->rev);
40     node u = v->p;
41     if (v == u->l)
42         u->l = v->r, v->r = u;
43     else
44         u->r = v->l, v->l = u;
45     swap(u->p, v->p); swap(v->pp, u->pp);
46     if (v->p != None){
47         assert(v->p->l == u || v->p->r == u);
48         if (v->p->r == u) v->p->r = v;
49         else v->p->l = v;
50     }
51     u->update(); v->update();
52 }
53 void bigRotate(node v){
54     assert(v->p != None);
55     v->p->p->push();
56     v->p->push();
57     v->push();
58     if (v->p->p != None){
59         if ((v->p->l == v) ^ (v->p->p->r == v->p))
60             rotate(v->p);
61         else
62             rotate(v);
63     }
64     rotate(v);
65 }
66 inline void Splay(node v){
67     while (v->p != None) bigRotate(v);
68 }
69 inline void splitAfter(node v){
70     v->push();
71     Splay(v);
72     v->r->p = None;
73     v->r->pp = v;
74     v->r = None;
75     v->update();
76 }
77 void expose(int x){
78     node v = v2n[x];
79     splitAfter(v);
80     while (v->pp != None){
81         assert(v->p == None);
82         splitAfter(v->pp);
83         assert(v->pp->r == None);
84         assert(v->pp->p == None);
85         assert(!v->pp->rev);
86         v->pp->r = v;
87         v->pp->update();
88         v = v->pp;
89         v->r->pp = None;
90     }
91     assert(v->p == None);
92     Splay(v2n[x]);
93 }
94 inline void makeRoot(int x){

```

```

95 expose(x);
96 assert(v2n[x]->p == None);
97 assert(v2n[x]->pp == None);
98 assert(v2n[x]->r == None);
99 v2n[x]->rev ^= 1;
100 }
101 inline void link(int x,int y){
102 makeRoot(x); v2n[x]->pp = v2n[y];
103 }
104 inline void cut(int x,int y){
105 expose(x);
106 Splay(v2n[y]);
107 if (v2n[y]->pp != v2n[x]){
108 swap(x,y);
109 expose(x);
110 Splay(v2n[y]);
111 assert(v2n[y]->pp == v2n[x]);
112 }
113 v2n[y]->pp = None;
114 }
115 inline int get(int x,int y){
116 if (x == y) return 0;
117 makeRoot(x);
118 expose(y); expose(x);
119 Splay(v2n[y]);
120 if (v2n[y]->pp != v2n[x]) return -1;
121 return v2n[y]->size;
122 }
123 // END ALGO
124
125 _node mem[MAXN];
126
127 int main(){
128 freopen("linkcut.in","r",stdin);
129 freopen("linkcut.out","w",stdout);
130
131 int n,m;
132 scanf("%d %d",&n,&m);
133
134 for (int i = 0; i < n; i++)
135 v2n[i] = &mem[i];
136
137 for (int i = 0; i < m; i++){
138 int a,b;
139 if (scanf(" link %d %d",&a,&b) == 2)
140 link(a-1,b-1);
141 else if (scanf(" cut %d %d",&a,&b) == 2)
142 cut(a-1,b-1);
143 else if (scanf(" get %d %d",&a,&b) == 2)
144 printf("%d\n",get(a-1,b-1));
145 else
146 assert(false);
147 }
148 return 0;
149 }

```

57 final/graphs/chordaltree.cpp

```

1 void chordaltree(vector<vector<int>> e) {
2 int n = e.size();
3
4 vector<int> mark(n);
5 set<pair<int, int>> st;
6 for (int i = 0; i < n; i++) st.insert({-mark[i], i});
7
8 vector<int> vct(n);
9 vector<pair<int, int>> ted;
10 vector<vector<int>> who(n);
11 vector<vector<int>> verts(1);
12 vector<int> cliq(n, -1);
13 cliq.push_back(0);
14 vector<int> last(n + 1, n);
15 int prev = n + 1;
16 for (int i = n - 1; i >= 0; i--) {
17 int x = st.begin()->second;
18 st.erase(st.begin());
19 if (mark[x] <= prev) {
20 vector<int> cur = who[x];
21 cur.push_back(x);
22 verts.push_back(cur);
23 ted.push_back({cliq[last[x]], (int)verts.size()
24 (-1)});
25 } else {
26 verts.back().push_back(x);

```

```

27 for (int y : e[x]) {
28 if (cliq[y] != -1) continue;
29 who[y].push_back(x);
30 st.erase({-mark[y], y});
31 mark[y]++;
32 st.insert({-mark[y], y});
33 last[y] = x;
34 }
35 prev = mark[x];
36 vct[i] = x;
37 cliq[x] = (int)verts.size() - 1;
38 }
39
40 int k = verts.size();
41 vector<int> pr(k);
42 vector<vector<int>> g(k);
43 for (auto o : ted) {
44 pr[o.second] = o.first;
45 g[o.first].push_back(o.second);
46 }
47 }

```

58 final/graphs/minimization.cpp

```

1 namespace mimimi /* ^ ^ */ {
2 const int N = 100555;
3 const int S = 3;
4 int e[N][S];
5 int label[N];
6 vector<int> eb[N][S];
7 int ans[N];
8 void solve(int n) {
9 for (int i = 0; i < n; ++i)
10 for (int j = 0; j < S; ++j)
11 eb[i][j].clear();
12 for (int i = 0; i < n; ++i)
13 for (int j = 0; j < S; ++j)
14 eb[e[i][j]][j].push_back(i);
15 vector<unordered_set<int>> classes(*max_element(
16 label, label + n) + 1);
17 for (int i = 0; i < n; ++i)
18 classes[label[i]].insert(i);
19 for (int i = 0; i < sz(classes); ++i)
20 if (classes[i].empty()) {
21 classes[i].swap(classes.back());
22 classes.pop_back();
23 --i;
24 }
25 for (int i = 0; i < sz(classes); ++i)
26 for (int v : classes[i])
27 ans[v] = i;
28 for (int i = 0; i < sz(classes); ++i)
29 for (int c = 0; c < S; ++c) {
30 unordered_map<int, unordered_set<int>>
31 involved;
32 for (int v : classes[i])
33 for (int nv : eb[v][c])
34 involved[ans[nv]].insert(nv);
35 for (auto &pp : involved) {
36 int cl = pp.X;
37 auto &cls = classes[cl];
38 if (sz(pp.Y) == sz(cls))
39 continue;
40 for (int x : pp.Y)
41 cls.erase(x);
42 if (sz(cls) < sz(pp.Y))
43 cls.swap(pp.Y);
44 for (int x : pp.Y)
45 ans[x] = sz(classes);
46 classes.push_back(move(pp.Y));
47 }
48 }
49 /* Usage: initialize edges: e[vertex][character]
50 labels: label[vertex]
51 solve(n)
52 ans[] - classes
53 */

```

59 final/graphs/matroidIntersection.cpp

```

1 struct Graph {
2     vector<vector<int>> G;
3
4     Graph(int n = 0) {
5         G.resize(n);
6     }
7
8     void add_edge(int v, int u) {
9         G[v].push_back(u);
10    }
11
12    vector<int> get_path(vector<int> &s, vector<int> &t) {
13        int n = G.size();
14        vector<int> dist(n, inf), pr(n, -1);
15        queue<int> Q;
16        for (int i : s) {
17            dist[i] = 0;
18            Q.push(i);
19        }
20        while (!Q.empty()) {
21            int v = Q.front();
22            Q.pop();
23            for (int to : G[v]) if (dist[to] > dist[v] + 1) {
24                dist[to] = dist[v] + 1;
25                pr[to] = v;
26                Q.push(to);
27            }
28        }
29        int V = -1;
30        for (int i : t) if (V == -1 || dist[i] < dist[V]) {
31            V = i;
32        }
33        if (V == -1 || dist[V] == inf) return {};
34        vector<int> path;
35        while (V != -1) {
36            path.push_back(V);
37            V = pr[V];
38        }
39        return path;
40    }
41};
42
43void get_ans(vector<int> &used, int m) {
44    Graph G(m);
45    for (int i = 0; i < m; ++i) if (used[i]) {
46        Gauss gauss;
47        vector<int> color(130, 0);
48        for (int j = 0; j < m; ++j) if (used[j] && j != i) {
49            gauss.add(a[j]);
50            color[c[j]] = 1;
51        }
52        for (int j = 0; j < m; ++j) if (!used[j]) {
53            if (gauss.check(a[j])) {
54                G.add_edge(i, j);
55            }
56            if (!color[c[j]]) {
57                G.add_edge(j, i);
58            }
59        }
60    }
61
62    Gauss gauss;
63    vector<int> color(130, 0);
64    for (int i = 0; i < m; ++i) if (used[i]) {
65        gauss.add(a[i]);
66        color[c[i]] = 1;
67    }
68    vector<int> x1, x2;
69    for (int i = 0; i < m; ++i) if (!used[i]) {
70        if (gauss.check(a[i])) {
71            x1.push_back(i);
72        }
73        if (!color[c[i]]) {
74            x2.push_back(i);
75        }
76    }
77    vector<int> path = G.get_path(x1, x2);
78    if (!path.size()) return;
79    for (int i : path) used[i] ^= 1;
80    get_ans(used, m);
81}

```

```

1 vector<pair<int, int>> compressTree(LCA& lca, const vi& subset) {
2     static vector<int> rev; rev.resize(sz(lca.dist));
3     vi li = subset, &T = lca.time;
4     auto cmp = [&](int a, int b) { return T[a] < T[b]; };
5     sort(all(li), cmp);
6     int m = sz(li)-1;
7     rep(i,0,m) {
8         int a = li[i], b = li[i+1];
9         li.push_back(lca.query(a, b));
10    }
11    sort(all(li), cmp);
12    li.erase(unique(all(li)), li.end());
13    rep(i,0,sz(li)) rev[li[i]] = i;
14    vpi ret = {pii(0, li[0])};
15    rep(i,0,sz(li)-1) {
16        int a = li[i], b = li[i+1];
17        ret.emplace_back(rev[lca.query(a, b)], b);
18    }
19    return ret;
20}

```



```
dbl Simpson() return (F(-1) + 4 * F(0) +
F(1)) / 6; dbl Runge2() return (F(-sqrtl(1.0
/ 3)) + F(sqrtl(1.0 / 3))) / 2; dbl Runge3()
return (F(-sqrtl(3.0 / 5)) * 5 + F(0) * 8 +
F(sqrtl(3.0 / 5)) * 5) / 18; Simpson и Runge2 –
точные для полиномов степени <= 3 Runge3 – точен для
полиномов степени <= 5
```

—

Явный Рунге-Кутт четвертого порядка, ошибка $O(h^4)$

```
y' = f(x, y)
yn+1 = yn + (k1 + 2 · k2 + 2 · k3 + k4) · h/6
k1 = f(xn, yn)
k2 = f(xn + h/2, yn + h/2 · k1)
k3 = f(xn + h/2, yn + h/2 · k2)
k4 = f(xn + h, yn + h · k3)
```

Методы Адамса-Башфорта

```
yn+3 = yn+2 + h * (23/12 * f(xn+2, yn+2) - 4/3 *
f(xn+1, yn+1) + 5/12 * f(xn, yn))
yn+4 = yn+3 + h * (55/24 * f(xn+3, yn+3) - 59/24 *
f(xn+2, yn+2) + 37/24 * f(xn+1, yn+1) - 3/8 * f(xn, yn))
yn+5 = yn+4 + h * (1901/720 * f(xn + 4, yn + 4) -
1387/360 * f(xn+3, yn+3) + 109/30 * f(xn+2, yn+2) - 637/360 *
f(xn+1, yn+1) + 251/720 * f(xn, yn))
```

—

Извлечение корня по простому модулю (от Сережи)

$3 \leq p, 1 \leq a < p$, найти $x^2 = a$

1) Если $a^{\frac{p-1}{2}} \neq 1$, return -1 2) Выбрать случайный $1 \leq i < p$ 3) $T(x) = (x + i)^{(p-1)/2} \bmod (x^2 - a) = bx + c$ 4) Если $b \neq 0$ то вернуть $\frac{c}{b}$, иначе к шагу 2)

—

Иногда вместо того чтобы считать первообразный у простого числа, можно написать чекер ответа и перебирать случайный первообразный.

Иногда можно представить ответ в виде многочлена и вместо подсчета самих k-тов посчитать значения и проинтерполировать

— Чтобы посчитать количество остовных деревьев в неориентированном графе G :

создать матрицу $N \times N$ mat, для каждого ребра (a, b) :

```
mat[a][a]++, mat[b][b]++, mat[a][b]-,
mat[b][a]-.
```

Удалить последнюю строку и столбец, взять дискриминант. —

Лемма Бернсайда:

Группа G действует на множество X Тогда число классов эквивалентности = $(\sum |f(g)| \text{ for } g \text{ in } G) / |G|$ где $f(g)$ = число x (из X) : $g(x) = x$

—

Число простых быстрее $O(n)$: $dp(n, k)$ - число чисел от 1 до n в которых все простые $\geq p[k]$

```
dp(n, 1) = n dp(n, j) = dp(n, j + 1) + dp(n /
p[j], j), т. е. dp(n, j + 1) = dp(n, j) - dp(n /
p[j], j) Если p[j], p[k] > sqrt(n) то dp(n, j) + j ==
dp(n, k) + k
```

Делаешь все оптимайзы сверху, но не считаешь глубже $dp(n, k)$, $n < K$ Потом фенвиком+сортировкой подсчитываешь за $(K+Q)\log$ все эти запросы Делаешь во второй раз, но на этот раз берешь прекальканные значения

Если $\sqrt{n} < p[k] < n$ то (число простых до n) = $dp(n, k) + k - 1$

—

$$sum(k = 1..n)k^2 = n(n+1)(2n+1)/6$$

$$sum(k = 1..n)k^3 = n^2(n+1)^2/4$$

Чиселки:

Фибоначчи 45: 1134903170 46: 1836311903
47: 2971215073 91: 4660046610375530309 92:
7540113804746346429 93: 12200160415121876738

Числа с кучей делителей 20: d(12)=6 50:
d(48)=10 100: d(60)=12 1000: d(840)=32 10⁴:
d(9240)=64 10⁵: d(83160)=128 10⁶: d(720720)=240
10⁷: d(8648640)=448 10⁸: d(91891800)=768 10⁹:
d(931170240)=1344 10¹¹: d(97772875200)=4032 10¹²:
d(963761198400)=6720 10¹⁵: d(866421317361600)=26880
10¹⁸: d(897612484786617600)=103680

Bell numbers: $B(p^m + n) = mB(n) + B(n+1)(\bmod p)$
0:1, 1:1, 2:2, 3:5, 4:15, 5:52, 6:203, 7:877, 8:4140,
9:21147, 10:115975, 11:678570, 12:4213597, 13:27644437,
14:190899322, 15:1382958545, 16:10480142147,
17:82864869804, 18:682076806159, 19:5832742205057,
20:51724158235372, 21:474869816156751,
22:4506715738447323, 23:44152005855084346

Catalan numbers: $C_n = \binom{2n}{n} / (n+1) = \binom{2n+1}{n} / (2n+1) =$
 $\binom{2n}{n} - \binom{2n}{n-1}$
0:1, 1:1, 2:2, 3:5, 4:14, 5:42, 6:132, 7:429, 8:1430, 9:4862,
10:16796, 11:58786, 12:208012, 13:742900, 14:2674440,
15:9694845, 16:35357670, 17:129644790, 18:477638700,
19:1767263190, 20:6564120420, 21:24466267020,
22:91482563640, 23:343059613650, 24:1289904147324,
25:4861946401452

Partitions numbers: see partition.cpp

0:1, 1:1, 2:2, 3:3, 4:5, 5:7, 6:11, 7:15, 8:22, 9:30,
10:42, 20:627, 30:5604, 40:37338, 50:204226, 60:966467,
70:4087968, 80:15796476, 90:56634173, 100:190569292

$prod(k = 1.. + inf)(1 - x^k) = sum(q = -inf.. + inf)(-1)^q x^{(3q^2 - q)/2}$

Table of Integrals*

Basic Forms

$$\int x^n dx = \frac{1}{n+1} x^{n+1} \quad (1)$$

$$\int \frac{1}{x} dx = \ln |x| \quad (2)$$

$$\int u dv = uv - \int v du \quad (3)$$

$$\int \frac{1}{ax+b} dx = \frac{1}{a} \ln |ax+b| \quad (4)$$

Integrals of Rational Functions

$$\int \frac{1}{(x+a)^2} dx = -\frac{1}{x+a} \quad (5)$$

$$\int (x+a)^n dx = \frac{(x+a)^{n+1}}{n+1}, n \neq -1 \quad (6)$$

$$\int x(x+a)^n dx = \frac{(x+a)^{n+1}((n+1)x-a)}{(n+1)(n+2)} \quad (7)$$

$$\int \frac{1}{1+x^2} dx = \tan^{-1} x \quad (8)$$

$$\int \frac{1}{a^2+x^2} dx = \frac{1}{a} \tan^{-1} \frac{x}{a} \quad (9)$$

$$\int \frac{x}{a^2+x^2} dx = \frac{1}{2} \ln |a^2+x^2| \quad (10)$$

$$\int \frac{x^2}{a^2+x^2} dx = x - a \tan^{-1} \frac{x}{a} \quad (11)$$

$$\int \frac{x^3}{a^2+x^2} dx = \frac{1}{2} x^2 - \frac{1}{2} a^2 \ln |a^2+x^2| \quad (12)$$

$$\int \frac{1}{ax^2+bx+c} dx = \frac{2}{\sqrt{4ac-b^2}} \tan^{-1} \frac{2ax+b}{\sqrt{4ac-b^2}} \quad (13)$$

$$\int \frac{1}{(x+a)(x+b)} dx = \frac{1}{b-a} \ln \frac{a+x}{b+x}, a \neq b \quad (14)$$

$$\int \frac{x}{(x+a)^2} dx = \frac{a}{a+x} + \ln |a+x| \quad (15)$$

$$\int \frac{x}{ax^2+bx+c} dx = \frac{1}{2a} \ln |ax^2+bx+c| - \frac{b}{a\sqrt{4ac-b^2}} \tan^{-1} \frac{2ax+b}{\sqrt{4ac-b^2}} \quad (16)$$

Integrals with Roots

$$\int \sqrt{x-ax} dx = \frac{2}{3} (x-a)^{3/2} \quad (17)$$

$$\int \frac{1}{\sqrt{x \pm a}} dx = 2\sqrt{x \pm a} \quad (18)$$

$$\int \frac{1}{\sqrt{a-x}} dx = -2\sqrt{a-x} \quad (19)$$

$$\int x\sqrt{x-ax} dx = \frac{2}{3} a(x-a)^{3/2} + \frac{2}{5} (x-a)^{5/2} \quad (20)$$

$$\int \sqrt{ax+bdx} = \left(\frac{2b}{3a} + \frac{2x}{3} \right) \sqrt{ax+b} \quad (21)$$

$$\int (ax+b)^{3/2} dx = \frac{2}{5a} (ax+b)^{5/2} \quad (22)$$

$$\int \frac{x}{\sqrt{x \pm a}} dx = \frac{2}{3} (x \mp 2a) \sqrt{x \pm a} \quad (23)$$

$$\int \sqrt{\frac{x}{a-x}} dx = -\sqrt{x(a-x)} - a \tan^{-1} \frac{\sqrt{x(a-x)}}{x-a} \quad (24)$$

$$\int \sqrt{\frac{x}{a+x}} dx = \sqrt{x(a+x)} - a \ln [\sqrt{x} + \sqrt{x+a}] \quad (25)$$

$$\int x\sqrt{ax+bdx} = \frac{2}{15a^2} (-2b^2 + abx + 3a^2x^2) \sqrt{ax+b} \quad (26)$$

$$\int \sqrt{x(ax+b)} dx = \frac{1}{4a^{3/2}} \left[(2ax+b) \sqrt{ax(ax+b)} - b^2 \ln |a\sqrt{x} + \sqrt{a(ax+b)}| \right] \quad (27)$$

$$\int \sqrt{x^3(ax+b)} dx = \left[\frac{b}{12a} - \frac{b^2}{8a^2x} + \frac{x}{3} \right] \sqrt{x^3(ax+b)} + \frac{b^3}{8a^{5/2}} \ln |a\sqrt{x} + \sqrt{a(ax+b)}| \quad (28)$$

$$\int \sqrt{x^2 \pm a^2} dx = \frac{1}{2} x \sqrt{x^2 \pm a^2} \pm \frac{1}{2} a^2 \ln |x + \sqrt{x^2 \pm a^2}| \quad (29)$$

$$\int \sqrt{a^2 - x^2} dx = \frac{1}{2} x \sqrt{a^2 - x^2} + \frac{1}{2} a^2 \tan^{-1} \frac{x}{\sqrt{a^2 - x^2}} \quad (30)$$

$$\int x\sqrt{x^2 \pm a^2} dx = \frac{1}{3} (x^2 \pm a^2)^{3/2} \quad (31)$$

$$\int \frac{1}{\sqrt{x^2 \pm a^2}} dx = \ln |x + \sqrt{x^2 \pm a^2}| \quad (32)$$

$$\int \frac{1}{\sqrt{a^2 - x^2}} dx = \sin^{-1} \frac{x}{a} \quad (33)$$

$$\int \frac{x}{\sqrt{x^2 \pm a^2}} dx = \sqrt{x^2 \pm a^2} \quad (34)$$

$$\int \frac{x}{\sqrt{a^2 - x^2}} dx = -\sqrt{a^2 - x^2} \quad (35)$$

$$\int \frac{x^2}{\sqrt{x^2 \pm a^2}} dx = \frac{1}{2} x \sqrt{x^2 \pm a^2} \mp \frac{1}{2} a^2 \ln |x + \sqrt{x^2 \pm a^2}| \quad (36)$$

$$\int \sqrt{ax^2+bx+cdx} = \frac{b+2ax}{4a} \sqrt{ax^2+bx+c} + \frac{4ac-b^2}{8a^{3/2}} \ln |2ax+b+2\sqrt{a(ax^2+bx+c)}| \quad (37)$$

$$\int x\sqrt{ax^2+bx+c} = \frac{1}{48a^{5/2}} \left(2\sqrt{a}\sqrt{ax^2+bx+c} \times (-3b^2+2abx+8a(c+ax^2)) + 3(b^3-4abc) \ln |b+2ax+2\sqrt{a}\sqrt{ax^2+bx+c}| \right) \quad (38)$$

$$\int \frac{1}{\sqrt{ax^2+bx+c}} dx = \frac{1}{\sqrt{a}} \ln |2ax+b+2\sqrt{a(ax^2+bx+c)}| \quad (39)$$

$$\int \frac{x}{\sqrt{ax^2+bx+c}} dx = \frac{1}{a} \sqrt{ax^2+bx+c} - \frac{b}{2a^{3/2}} \ln |2ax+b+2\sqrt{a(ax^2+bx+c)}| \quad (40)$$

$$\int \frac{dx}{(a^2+x^2)^{3/2}} = \frac{x}{a^2\sqrt{a^2+x^2}} \quad (41)$$

Integrals with Logarithms

$$\int \ln ax dx = x \ln ax - x \quad (42)$$

$$\int \frac{\ln ax}{x} dx = \frac{1}{2} (\ln ax)^2 \quad (43)$$

$$\int \ln(ax+b) dx = \left(x + \frac{b}{a} \right) \ln(ax+b) - x, a \neq 0 \quad (44)$$

$$\int \ln(x^2+a^2) dx = x \ln(x^2+a^2) + 2a \tan^{-1} \frac{x}{a} - 2x \quad (45)$$

$$\int \ln(x^2-a^2) dx = x \ln(x^2-a^2) + a \ln \frac{x+a}{x-a} - 2x \quad (46)$$

$$\int \ln(ax^2+bx+c) dx = \frac{1}{a} \sqrt{4ac-b^2} \tan^{-1} \frac{2ax+b}{\sqrt{4ac-b^2}} - 2x + \left(\frac{b}{2a} + x \right) \ln(ax^2+bx+c) \quad (47)$$

$$\int x \ln(ax+b) dx = \frac{bx}{2a} - \frac{1}{4} x^2 + \frac{1}{2} \left(x^2 - \frac{b^2}{a^2} \right) \ln(ax+b) \quad (48)$$

$$\int x \ln(a^2-b^2x^2) dx = -\frac{1}{2} x^2 + \frac{1}{2} \left(x^2 - \frac{a^2}{b^2} \right) \ln(a^2-b^2x^2) \quad (49)$$

Integrals with Exponentials

$$\int e^{ax} dx = \frac{1}{a} e^{ax} \quad (50)$$

$$\int \sqrt{x} e^{ax} dx = \frac{1}{a} \sqrt{x} e^{ax} + \frac{i\sqrt{\pi}}{2a^{3/2}} \operatorname{erf}(i\sqrt{ax}), \text{ where } \operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \quad (51)$$

$$\int x e^x dx = (x-1)e^x \quad (52)$$

$$\int x e^{ax} dx = \left(\frac{x}{a} - \frac{1}{a^2} \right) e^{ax} \quad (53)$$

$$\int x^2 e^x dx = (x^2 - 2x + 2) e^x \quad (54)$$

$$\int x^2 e^{ax} dx = \left(\frac{x^2}{a} - \frac{2x}{a^2} + \frac{2}{a^3} \right) e^{ax} \quad (55)$$

$$\int x^3 e^x dx = (x^3 - 3x^2 + 6x - 6) e^x \quad (56)$$

$$\int x^n e^{ax} dx = \frac{x^n e^{ax}}{a} - \frac{n}{a} \int x^{n-1} e^{ax} dx \quad (57)$$

$$\int x^n e^{ax} dx = \frac{(-1)^n}{a^{n+1}} \Gamma[1+n, -ax], \text{ where } \Gamma(a, x) = \int_x^\infty t^{a-1} e^{-t} dt \quad (58)$$

$$\int e^{ax^2} dx = -\frac{i\sqrt{\pi}}{2\sqrt{a}} \operatorname{erf}(i\sqrt{a}x) \quad (59)$$

$$\int e^{-ax^2} dx = \frac{\sqrt{\pi}}{2\sqrt{a}} \operatorname{erf}(x\sqrt{a}) \quad (60)$$

$$\int x e^{-ax^2} dx = -\frac{1}{2a} e^{-ax^2} \quad (61)$$

$$\int x^2 e^{-ax^2} dx = \frac{1}{4} \sqrt{\frac{\pi}{a^3}} \operatorname{erf}(x\sqrt{a}) - \frac{x}{2a} e^{-ax^2} \quad (62)$$

*© 2014. From <http://integral-table.com>, last revised June 14, 2014. This material is provided as is without warranty or representation about the accuracy, correctness or suitability of the material for any purpose, and is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 United States License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/3.0/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

Integrals with Trigonometric Functions

$$\int \sin ax dx = -\frac{1}{a} \cos ax \quad (63)$$

$$\int \sin^2 ax dx = \frac{x}{2} - \frac{\sin 2ax}{4a} \quad (64)$$

$$\int \sin^n ax dx = -\frac{1}{a} \cos ax {}_2F_1 \left[\frac{1}{2}, \frac{1-n}{2}, \frac{3}{2}, \cos^2 ax \right] \quad (65)$$

$$\int \sin^3 ax dx = -\frac{3 \cos ax}{4a} + \frac{\cos 3ax}{12a} \quad (66)$$

$$\int \cos ax dx = \frac{1}{a} \sin ax \quad (67)$$

$$\int \cos^2 ax dx = \frac{x}{2} + \frac{\sin 2ax}{4a} \quad (68)$$

$$\int \cos^p ax dx = -\frac{1}{a(1+p)} \cos^{1+p} ax \times {}_2F_1 \left[\frac{1+p}{2}, \frac{1}{2}, \frac{3+p}{2}, \cos^2 ax \right] \quad (69)$$

$$\int \cos^3 ax dx = \frac{3 \sin ax}{4a} + \frac{\sin 3ax}{12a} \quad (70)$$

$$\int \cos ax \sin bxdx = \frac{\cos[(a-b)x]}{2(a-b)} - \frac{\cos[(a+b)x]}{2(a+b)}, a \neq b \quad (71)$$

$$\int \sin^2 ax \cos bxdx = -\frac{\sin[(2a-b)x]}{4(2a-b)} + \frac{\sin bx}{2b} - \frac{\sin[(2a+b)x]}{4(2a+b)} \quad (72)$$

$$\int \sin^2 x \cos x dx = \frac{1}{3} \sin^3 x \quad (73)$$

$$\int \cos^2 ax \sin bxdx = \frac{\cos[(2a-b)x]}{4(2a-b)} - \frac{\cos bx}{2b} - \frac{\cos[(2a+b)x]}{4(2a+b)} \quad (74)$$

$$\int \cos^2 ax \sin ax dx = -\frac{1}{3a} \cos^3 ax \quad (75)$$

$$\int \sin^2 ax \cos^2 bxdx = \frac{x}{4} - \frac{\sin 2ax}{8a} - \frac{\sin[2(a-b)x]}{16(a-b)} + \frac{\sin 2bx}{8b} - \frac{\sin[2(a+b)x]}{16(a+b)} \quad (76)$$

$$\int \sin^2 ax \cos^2 ax dx = \frac{x}{8} - \frac{\sin 4ax}{32a} \quad (77)$$

$$\int \tan ax dx = -\frac{1}{a} \ln \cos ax \quad (78)$$

$$\int \tan^2 ax dx = -x + \frac{1}{a} \tan ax \quad (79)$$

$$\int \tan^n ax dx = \frac{\tan^{n+1} ax}{a(1+n)} \times {}_2F_1 \left(\frac{n+1}{2}, 1, \frac{n+3}{2}, -\tan^2 ax \right) \quad (80)$$

$$\int \tan^3 ax dx = \frac{1}{a} \ln \cos ax + \frac{1}{2a} \sec^2 ax \quad (81)$$

$$\int \sec x dx = \ln |\sec x + \tan x| = 2 \tanh^{-1} \left(\tan \frac{x}{2} \right) \quad (82)$$

$$\int \sec^2 ax dx = \frac{1}{a} \tan ax \quad (83)$$

$$\int \sec^3 x dx = \frac{1}{2} \sec x \tan x + \frac{1}{2} \ln |\sec x + \tan x| \quad (84)$$

$$\int \sec x \tan x dx = \sec x \quad (85)$$

$$\int \sec^2 x \tan x dx = \frac{1}{2} \sec^2 x \quad (86)$$

$$\int \sec^n x \tan x dx = \frac{1}{n} \sec^n x, n \neq 0 \quad (87)$$

$$\int \csc x dx = \ln \left| \tan \frac{x}{2} \right| = \ln |\csc x - \cot x| + C \quad (88)$$

$$\int \csc^2 ax dx = -\frac{1}{a} \cot ax \quad (89)$$

$$\int \csc^3 x dx = -\frac{1}{2} \cot x \csc x + \frac{1}{2} \ln |\csc x - \cot x| \quad (90)$$

$$\int \csc^n x \cot x dx = -\frac{1}{n} \csc^n x, n \neq 0 \quad (91)$$

$$\int \sec x \csc x dx = \ln |\tan x| \quad (92)$$

Products of Trigonometric Functions and Monomials

$$\int x \cos x dx = \cos x + x \sin x \quad (93)$$

$$\int x \cos ax dx = \frac{1}{a^2} \cos ax + \frac{x}{a} \sin ax \quad (94)$$

$$\int x^2 \cos x dx = 2x \cos x + (x^2 - 2) \sin x \quad (95)$$

$$\int x^2 \cos ax dx = \frac{2x \cos ax}{a^2} + \frac{a^2 x^2 - 2}{a^3} \sin ax \quad (96)$$

$$\int x^n \cos x dx = -\frac{1}{2} (i)^{n+1} [\Gamma(n+1, -ix) + (-1)^n \Gamma(n+1, ix)] \quad (97)$$

$$\int x^n \cos ax dx = \frac{1}{2} (ia)^{1-n} [(-1)^n \Gamma(n+1, -iax) - \Gamma(n+1, iax)] \quad (98)$$

$$\int x \sin x dx = -x \cos x + \sin x \quad (99)$$

$$\int x \sin ax dx = -\frac{x \cos ax}{a} + \frac{\sin ax}{a^2} \quad (100)$$

$$\int x^2 \sin x dx = (2 - x^2) \cos x + 2x \sin x \quad (101)$$

$$\int x^2 \sin ax dx = \frac{2 - a^2 x^2}{a^3} \cos ax + \frac{2x \sin ax}{a^2} \quad (102)$$

$$\int x^n \sin x dx = -\frac{1}{2} (i)^n [\Gamma(n+1, -ix) - (-1)^n \Gamma(n+1, -ix)] \quad (103)$$

Products of Trigonometric Functions and Exponentials

$$\int e^x \sin x dx = \frac{1}{2} e^x (\sin x - \cos x) \quad (104)$$

$$\int e^{bx} \sin ax dx = \frac{1}{a^2 + b^2} e^{bx} (b \sin ax - a \cos ax) \quad (105)$$

$$\int e^x \cos x dx = \frac{1}{2} e^x (\sin x + \cos x) \quad (106)$$

$$\int e^{bx} \cos ax dx = \frac{1}{a^2 + b^2} e^{bx} (a \sin ax + b \cos ax) \quad (107)$$

$$\int x e^x \sin x dx = \frac{1}{2} e^x (\cos x - x \cos x + x \sin x) \quad (108)$$

$$\int x e^x \cos x dx = \frac{1}{2} e^x (x \cos x - \sin x + x \sin x) \quad (109)$$

Integrals of Hyperbolic Functions

$$\int \cosh ax dx = \frac{1}{a} \sinh ax \quad (110)$$

$$\int e^{ax} \cosh bxdx = \begin{cases} \frac{e^{ax}}{a^2 - b^2} [a \cosh bx - b \sinh bx] & a \neq b \\ \frac{e^{2ax}}{4a} + \frac{x}{2} & a = b \end{cases} \quad (111)$$

$$\int \sinh ax dx = \frac{1}{a} \cosh ax \quad (112)$$

$$\int e^{ax} \sinh bxdx = \begin{cases} \frac{e^{ax}}{a^2 - b^2} [-b \cosh bx + a \sinh bx] & a \neq b \\ \frac{e^{2ax}}{4a} - \frac{x}{2} & a = b \end{cases} \quad (113)$$

$$\int e^{ax} \tanh bxdx = \begin{cases} \frac{e^{(a+2b)x}}{(a+2b)} {}_2F_1 \left[1 + \frac{a}{2b}, 1, 2 + \frac{a}{2b}, -e^{2bx} \right] - \frac{1}{a} e^{ax} {}_2F_1 \left[\frac{a}{2b}, 1, 1E, -e^{2bx} \right] & a \neq b \\ \frac{e^{ax} - 2 \tan^{-1}[e^{ax}]}{a} & a = b \end{cases} \quad (114)$$

$$\int \tanh ax dx = \frac{1}{a} \ln \cosh ax \quad (115)$$

$$\int \cos ax \cosh bxdx = \frac{1}{a^2 + b^2} [a \sin ax \cosh bx + b \cos ax \sinh bx] \quad (116)$$

$$\int \cos ax \sinh bxdx = \frac{1}{a^2 + b^2} [b \cos ax \cosh bx + a \sin ax \sinh bx] \quad (117)$$

$$\int \sin ax \cosh bxdx = \frac{1}{a^2 + b^2} [-a \cos ax \cosh bx + b \sin ax \sinh bx] \quad (118)$$

$$\int \sin ax \sinh bxdx = \frac{1}{a^2 + b^2} [b \cosh bx \sin ax - a \cos ax \sinh bx] \quad (119)$$

$$\int \sinh ax \cosh ax dx = \frac{1}{4a} [-2ax + \sinh 2ax] \quad (120)$$

$$\int \sinh ax \cosh bxdx = \frac{1}{b^2 - a^2} [b \cosh bx \sinh ax - a \cosh ax \sinh bx] \quad (121)$$