Contents

1 Some useful stuff **1**
1.1 Fast I/O . . . 1
1.2 Java template . . . 2
1.3 Pragmas . . . 2

2 Data structures **2**
2.1 Fenwick tree . . . 2
2.2 Hash table . . . 2
2.3 Ordered set and bitset . . . 3

3 Geometry **3**
3.1 Common tangents of two circles . . . 3
3.2 Convex hull 3D in $O(n^2)$ . . . 3
3.3 Dynamic convex hull trick . . . 4
3.4 Halfplanes intersection . . . 4
3.5 Minimal covering disk . . . 5
3.6 Polygon tangent . . . 5
3.7 Rotate 3D . . . 5
3.8 Rotation matrix 2D . . . 6
3.9 Sphere distance . . . 6
3.10 Draw svg pictures . . . 6

4 Graphs **6**
4.1 2-Chinese algorithm . . . 6
4.2 Dominator tree . . . 7
4.3 General matching . . . 8
4.4 Gomory-Hu tree . . . 9
4.5 Hungarian algorithm . . . 9
4.6 Link-Cut Tree . . . 10
4.7 Push-Relabel . . . 11
4.8 Smith algorithm (Game on cyclic graph) . . . 12
4.9 Stoer-Vagner algorithm (Global min-cut) . . . 13

5 Matroids **13**
5.1 Matroids intersection . . . 13

6 Numeric **14**
6.1 Berlekamp-Massey Algorithm . . . 14
6.2 Burnside's lemma . . . 15
6.3 Chinese remainder theorem . . . 15
6.4 AND/OR/XOR convolution . . . 15
6.5 Counting size of the maximum general matching . . . 15
6.6 Counting number of spanning trees . . . 16
6.7 Some formulas . . . 16
6.8 Miller–Rabin primality test . . . 16
6.9 Taking by modullo (Inline assembler) . . . 16
6.10 First solution of $(p + step \cdot x) \bmod mod < l$ . . . 16
6.11 Multiplication by modulo in `long double` . . . 16
6.12 Numerical integration . . . 16
6.13 Pollard's rho algorithm . . . 17
6.14 Polynom division and inversion . . . 17
6.15 Polynom roots . . . 18
6.16 Simplex method . . . 18
6.17 Some integer sequences . . . 20

7 Strings **20**
7.1 Duval algorithm (Lyndon factorization) . . . 20
7.2 Palindromic tree . . . 20
7.3 Manacher's algorithm . . . 21
7.4 Suffix array + LCP . . . 21
7.5 Suffix automaton . . . 22
7.6 Suffix tree . . . 22

1 Some useful stuff

1.1 Fast I/O

```
#include <algorithm>
#include <cstdio>

/** Interface */

inline int readInt();
inline int readUInt();
inline bool isEof();

/** Read */

static const int buf_size = 100000;
static char buf[buf_size];
static int buf_len = 0, pos = 0;

inline bool isEof() {
  if (pos == buf_len) {
    pos = 0, buf_len = fread(buf, 1, buf_size,
      stdin);
    if (pos == buf_len)
      return 1;
  }
  return 0;
}

inline int getChar() { return isEof() ? -1 :
  buf[pos++]; }

inline int readChar() {
  int c = getChar();
  while (c != -1 && c <= 32)
    c = getChar();
  return c;
}

inline int readUInt() {
  int c = readChar(), x = 0;
  while ('0' <= c && c <= '9')
    x = x * 10 + c - '0', c = getChar();
  return x;
}

inline int readInt() {
  int s = 1, c = readChar();
  int x = 0;
  if (c == '-')
    s = -1, c = getChar();
  while ('0' <= c && c <= '9')
    x = x * 10 + c - '0', c = getChar();
  return s == 1 ? x : -x;
}

// 10M int [0..1e9)
// cin 3.02
// scanf 1.2
// cin sync_with_stdio(false) 0.71
// fastRead getchar 0.53
// fastRead fread 0.15
```

1.2 Java template

```
import java.util.*;
import java.io.*;

public class Template {
  FastScanner in;
  PrintWriter out;

  public void solve() throws IOException {
    int n = in.nextInt();
    out.println(n);
  }

  public void run() {
    try {
      in = new FastScanner();
      out = new PrintWriter(System.out);

      solve();

      out.close();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }

  class FastScanner {
    BufferedReader br;
    StringTokenizer st;

    FastScanner() {
      br = new BufferedReader(new
        InputStreamReader(System.in));
    }

    String next() {
      while (st == null || !st.hasMoreTokens()) {
        try {
          st = new
            StringTokenizer(br.readLine());
        } catch (IOException e) {
          e.printStackTrace();
        }
      }
      return st.nextToken();
    }

    int nextInt() {
      return Integer.parseInt(next());
    }
  }

  public static void main(String[] arg) {
    new Template().run();
  }
}
```

1.3 Pragmas

```
// have no idea what sse flags are really cool;
  list of some of them
// -- very good with bitsets
```

```cpp
#pragma GCC optimize("O3")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx")
```

2 Data structures

2.1 Fenwick tree

```cpp
struct FT {
  vector<ll> s;
  FT(int n) : s(n) {}
  void update(int pos, ll dif) { // a[pos] += dif
    for (; pos < sz(s); pos |= pos + 1) s[pos] += dif;
  }
  ll query(int pos) { // sum of values in [0, pos)
    ll res = 0;
    for (; pos > 0; pos &= pos - 1) res += s[pos-1];
    return res;
  }
  int lower_bound(ll sum) {// min pos st sum of [0, pos] >= sum
    // Returns n if no sum is >= sum, or -1 if empty sum is.
    if (sum <= 0) return -1;
    int pos = 0;
    for (int pw = 1 << 25; pw; pw >>= 1) {
      if (pos + pw <= sz(s) && s[pos + pw-1] < sum)
        pos += pw, sum -= s[pos-1];
    }
    return pos;
  }
};
```

2.2 Hash table

```cpp
template <const int max_size, class HashType, class Data,
          const Data default_value>
struct hashTable {
  HashType hash[max_size];
  Data f[max_size];
  int size;

  int position(HashType H) const {
    int i = H % max_size;
    while (hash[i] && hash[i] != H)
      if (++i == max_size)
        i = 0;
    return i;
  }

  Data &operator[](HashType H) {
    assert(H != 0);
    int i = position(H);
    if (!hash[i]) {
      hash[i] = H;
      f[i] = default_value;
      size++;
    }
    return f[i];
  }
};

hashTable<13, int, int, 0> h;
```

2.3 Ordered set and bitset

```cpp
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;

template <typename T> using ordered_set = tree<T, null_type, less<T>, rb_tree_tag, tree_order_statistics_node_update>;
template <typename K, typename V> using ordered_map = tree<K, V, less<K>, rb_tree_tag, tree_order_statistics_node_update>;

// HOW TO USE ::
// -- order_of_key(10) returns the number of elements in set/map strictly less than 10
// -- *find_by_order(10) returns 10-th smallest element in set/map (0-based)

bitset<N> a;
for (int i = a._Find_first(); i != a.size(); i = a._Find_next(i)) {
  cout << i << endl;
}
```

3 Geometry

3.1 Common tangents of two circles

```cpp
vector<Line> commonTangents(pt A, dbl rA, pt B, dbl rB) {
  vector<Line> res;
  pt C = B - A;
  dbl z = C.len2();
  for (int i = -1; i <= 1; i += 2) {
    for (int j = -1; j <= 1; j += 2) {
      dbl r = rB * j - rA * i;
      dbl d = z - r * r;
      if (ls(d, 0))
        continue;
      d = sqrt(max(0.01, d));
      pt magic = pt(r, d) / z;
      pt v(magic % C, magic * C);
      dbl CC = (rA * i - v % A) / v.len2();
      pt O = v * -CC;
      res.pb(Line(O, O + v.rotate()));
    }
  }
  return res;
}

// HOW TO USE ::
// --      *D*---------------*F*
// --     *...*-            -*...*
```

```
// --    *.....* -       - *.....*
// --   *.......*  -   -  *.......*
// --   *...A...*   --    *...B...*
// --   *.......*  -   -  *.......*
// --    *.....* -       - *.....*
// --     *...*-          -*...*
// --      *C*---------------*E*
// --  res = {CE, CF, DE, DF}
```

3.2 Convex hull 3D in $O(n^2)$

```
struct Plane {
  pt O, v;
  vector<int> id;
};

vector<Plane> convexHull3(vector<pt> p) {
  vector<Plane> res;
  int n = p.size();
  for (int i = 0; i < n; i++)
    p[i].id = i;
  for (int i = 0; i < 4; i++) {
    vector<pt> tmp;
    for (int j = 0; j < 4; j++)
      if (i != j)
        tmp.pb(p[j]);
    res.pb({tmp[0],
            (tmp[1] - tmp[0]) * (tmp[2] - tmp[0]),
            {tmp[0].id, tmp[1].id, tmp[2].id}});
    if ((p[i] - res.back().O) % res.back().v > 0) {
      res.back().v = res.back().v * -1;
      swap(res.back().id[0], res.back().id[1]);
    }
  }
  vector<vector<int>> use(n, vector<int>(n, 0));
  int tmr = 0;
  for (int i = 4; i < n; i++) {
    int cur = 0;
    tmr++;
    vector<pair<int, int>> curEdge;
    for (int j = 0; j < sz(res); j++) {
      if ((p[i] - res[j].O) % res[j].v > 0) {
        for (int t = 0; t < 3; t++) {
          int v = res[j].id[t];
          int u = res[j].id[(t + 1) % 3];
          use[v][u] = tmr;
          curEdge.pb({v, u});
        }
      } else {
        res[cur++] = res[j];
      }
    }
    res.resize(cur);
    for (auto x : curEdge) {
      if (use[x.S][x.F] == tmr)
        continue;
      res.pb({p[i], (p[x.F] - p[i]) * (p[x.S] - p[i]), {x.F, x.S, i}});
    }
  }
  return res;
}

// plane in 3d
// (A, v) * (B, u) -> (O, n)

pt n = v * u;
pt m = v * n;
double t = (B - A) % u / (u % m);
pt O = A - m * t;
```

3.3 Dynamic convex hull trick

```
const ll is_query = -(1LL << 62);

struct Line {
  ll m, b;
  mutable function<const Line *()> succ;

  bool operator<(const Line &rhs) const {
    if (rhs.b != is_query)
      return m < rhs.m;
    const Line *s = succ();
    if (!s)
      return 0;
    ll x = rhs.m;
    return b - s->b < (s->m - m) * x;
  }
};

struct HullDynamic : public multiset<Line> {
  bool bad(iterator y) {
    auto z = next(y);
    if (y == begin()) {
      if (z == end())
        return 0;
      return y->m == z->m && y->b <= z->b;
    }
    auto x = prev(y);
    if (z == end())
      return y->m == x->m && y->b <= x->b;
    return (x->b - y->b) * (z->m - y->m) >= (y->b - z->b) * (y->m - x->m);
  }

  void insert_line(ll m, ll b) {
    auto y = insert({m, b});
    y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
    if (bad(y)) {
      erase(y);
      return;
    }
    while (next(y) != end() && bad(next(y)))
      erase(next(y));
    while (y != begin() && bad(prev(y)))
      erase(prev(y));
  }

  ll eval(ll x) {
    auto l = *lower_bound((Line){x, is_query});
    return l.m * x + l.b;
```

```
    }
};
```

3.4 Halfplanes intersection

```
int getPart(pt v) {
    return ls(v.y, 0) || (eq(0, v.y) && ls(v.x, 0));
}

int cmpV(pt a, pt b) {
    int partA = getPart(a);
    int partB = getPart(b);
    if (partA < partB) return 1;
    if (partA > partB) return -1;
    if (eq(0, a * b)) return 0;
    if (0 < a * b) return -1;
    return 1;
}

double planeInt(vector<Line> l) {
    sort(all(l), [](Line a, Line b) {
            int r = cmpV(a.v, b.v);
            if (r != 0) return r < 0;
            return a.O % a.v.rotate() > b.O % a.v.rotate();
        });

    l.resize(unique(all(l), [](Line A, Line B) { return cmpV(A.v, B.v) == 0; }) - l.begin());
    for (int i = 0; i < sz(l); i++)
        l[i].id = i;

    // if an infinite answer is possible
    int flagUp = 0;
    int flagDown = 0;
    for (int i = 0; i < sz(l); i++) {
        int part = getPart(l[i].v);
        if (part == 1) flagUp = 1;
        if (part == 0) flagDown = 1;
    }
    if (!flagUp || !flagDown) return -1;

    for (int i = 0; i < sz(l); i++) {
        pt v = l[i].v;
        pt u = l[(i + 1) % sz(l)].v;
        if (eq(0, v * u) && ls(v % u, 0)) {
            pt dir = l[i].v.rotate();
            if (le(l[(i + 1) % sz(l)].O % dir, l[i].O % dir)) return 0;
            return -1;
        }
        if (ls(v * u, 0))
            return -1;
    }
    // main part
    vector<Line> st;
    for (int tt = 0; tt < 2; tt++) {
        for (auto L: l) {
            for (; sz(st) >= 2 && le(st[sz(st) - 2].v * (st.back() * L - st[sz(st) - 2].O), 0); st.pop_back());
            st.pb(L);
            if (sz(st) >= 2 && le(st[sz(st) - 2].v * st.back().v, 0)) return 0; // useless line
        }
    }
    vector<int> use(sz(l), -1);
    int left = -1, right = -1;
    for (int i = 0; i < sz(st); i++) {
        if (use[st[i].id] == -1) {
            use[st[i].id] = i;
        }
        else {
            left = use[st[i].id];
            right = i;
            break;
        }
    }
    vector<Line> tmp;
    for (int i = left; i < right; i++)
        tmp.pb(st[i]);
    vector<pt> res;
    for (int i = 0; i < (int)tmp.size(); i++)
        res.pb(tmp[i] * tmp[(i + 1) % tmp.size()]);
    double area = 0;
    for (int i = 0; i < (int)res.size(); i++)
        area += res[i] * res[(i + 1) % res.size()];
    return area / 2;
}
```

3.5 Minimal covering disk

```
pair<pt, dbl> minDisc(vector<pt> p) {
    int n = p.size();
    pt O = pt(0, 0);
    dbl R = 0;
    random_shuffle(all(p));
    for (int i = 0; i < n; i++) {
        if (ls(R, (O - p[i]).len())) {
            O = p[i];
            R = 0;
            for (int j = 0; j < i; j++) {
                if (ls(R, (O - p[j]).len())) {
                    O = (p[i] + p[j]) / 2;
                    R = (p[i] - p[j]).len() / 2;
                    for (int k = 0; k < j; k++) {
                        if (ls(R, (O - p[k]).len())) {
                            Line l1((p[i] + p[j]) / 2,
                                    (p[i] + p[j]) / 2 + (p[i] - p[j]).rotate());
                            Line l2((p[k] + p[j]) / 2,
                                    (p[k] + p[j]) / 2 + (p[k] - p[j]).rotate());
                            O = l1 * l2;
                            R = (p[i] - O).len();
                        }
                    }
                }
            }
        }
```

```
    }
  }
  return {O, R};
}
```

3.6 Polygon tangent

```
pt tangent(vector<pt>& p, pt O, int cof) {
  int step = 1;
  for (; step < (int)p.size(); step *= 2);
  int pos = 0;
  int n = p.size();
  for (; step > 0; step /= 2) {
    int best = pos;
    for (int dx = -1; dx <= 1; dx += 2) {
      int id = ((pos + step * dx) % n + n) % n;
      if ((p[id] - O) * (p[best] - O) * cof > 0)
        best = id;
    }
    pos = best;
  }
  return p[pos];
}
```

3.7 Rotate 3D

```
// Rotate 3d point along axis on angle
/*
 * 2D
 * x' = x cos a - y sin a
 * y' = x sin a + y cos a
*/
struct quater {
  double w, x, y, z; // w + xi + yj + zk
  quater(double tw, const pt3 &v) : w(tw),
    x(v.x), y(v.y), z(v.z) { }
  quater(double tw, double tx, double ty, double
    tz) : w(tw), x(tx), y(ty), z(tz) { }
  pt3 vector() const {
    return {x, y, z};
  }
  quater conjugate() const {
    return {w, -x, -y, -z};
  }
  quater operator*(const quater &q2) {
    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};
  }
};

pt3 rotate(pt3 axis, pt3 p, double angle) {
  quater q = quater(cos(angle / 2), axis *
    sin(angle / 2));
  return (q * quater(0, p) *
    q.conjugate()).vector();
}
```

3.8 Rotation matrix 2D

Rotation of point (x, y) through an angle α in counterclockwise direction in 2D.

$$\begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x' \\ y' \end{pmatrix}$$

3.9 Sphere distance

```
double sphericalDistance(double f1, double t1,
    double f2, double t2, double radius) {
  double dx = sin(t2)*cos(f2) - sin(t1)*cos(f1);
  double dy = sin(t2)*sin(f2) - sin(t1)*sin(f1);
  double dz = cos(t2) - cos(t1);
  double d = sqrt(dx*dx + dy*dy + dz*dz);
  return radius*2*asin(d/2);
}
```

3.10 Draw svg pictures

```
struct SVG {
  FILE *out;
  double sc = 50;
  void open() {
    out = fopen("image.svg", "w");
    fprintf(out, "<svg
      xmlns='http://www.w3.org/2000/svg'
      viewBox='-1000 -1000 2000 2000'>\n");
  }
  void line(point a, point b) {
    a = a * sc, b = b * sc;
    fprintf(out, "<line x1='%f' y1='%f' x2='%f'
      y2='%f' stroke='black'/>\n", a.x, -a.y,
      b.x, -b.y);
  }
  void circle(point a, double r = -1, string col
    = "red") {
    r = sc * (r == -1 ? 0.3 : r);
    a = a * sc;
    fprintf(out, "<circle cx='%f' cy='%f' r='%f'
      fill='%s'/>\n", a.x, -a.y, r,
      col.c_str());
  }
  void text(point a, string s) {
    a = a * sc;
    fprintf(out, "<text x='%f' y='%f'
      font-size='100px'>%s</text>\n", a.x,
      -a.y, s.c_str());
  }
  void close() {
    fprintf(out, "</svg>\n");
    fclose(out);
    out = 0;
  }
  ~SVG() {
    if (out) {
      close();
    }
  }
} svg;
```

4 Graphs

4.1 2-Chinese algorithm

```
namespace twoc {
struct Heap {
  static Heap *null;
  ll x, xadd;
  int ver, h;
#ifdef ANS
  int ei;
#endif
  Heap *l, *r;
  Heap(ll xx, int vv) : x(xx), xadd(0), ver(vv),
    ↪ h(1), l(null), r(null) {}
  Heap(const char *) : x(0), xadd(0), ver(0),
    ↪ h(0), l(this), r(this) {}
  void add(ll a) {
    x += a;
    xadd += a;
  }
  void push() {
    if (l != null)
      l->add(xadd);
    if (r != null)
      r->add(xadd);
    xadd = 0;
  }
};
Heap *Heap::null = new Heap("wqeqw");
Heap *merge(Heap *l, Heap *r) {
  if (l == Heap::null)
    return r;
  if (r == Heap::null)
    return l;
  l->push();
  r->push();
  if (l->x > r->x)
    swap(l, r);
  l->r = merge(l->r, r);
  if (l->l->h < l->r->h)
    swap(l->l, l->r);
  l->h = l->r->h + 1;
  return l;
}
Heap *pop(Heap *h) {
  h->push();
  return merge(h->l, h->r);
}
const int N = 666666;
struct DSU {
  int p[N];
  void init(int nn) { iota(p, p + nn, 0); }
  int get(int x) { return p[x] == x ? x : p[x] =
    ↪ get(p[x]); }
  void merge(int x, int y) { p[get(y)] = get(x);
    ↪ }
} dsu;
Heap *eb[N];
int n;
#ifdef ANS
struct Edge {
  int x, y;
  ll c;
};
vector<Edge> edges;
int answer[N];
#endif
void init(int nn) {
  n = nn;
  dsu.init(n);
  fill(eb, eb + n, Heap::null);
  edges.clear();
}
void addEdge(int x, int y, ll c) {
  Heap *h = new Heap(c, x);
#ifdef ANS
  h->ei = sz(edges);
  edges.push_back({x, y, c});
#endif
  eb[y] = merge(eb[y], h);
}
ll solve(int root = 0) {
  ll ans = 0;
  static int done[N], pv[N];
  memset(done, 0, sizeof(int) * n);
  done[root] = 1;
  int tt = 1;
#ifdef ANS
  int cnum = 0;
  static vector<ipair> eout[N];
  for (int i = 0; i < n; ++i)
    eout[i].clear();
#endif
  for (int i = 0; i < n; ++i) {
    int v = dsu.get(i);
    if (done[v])
      continue;
    ++tt;
    while (true) {
      done[v] = tt;
      int nv = -1;
      while (eb[v] != Heap::null) {
        nv = dsu.get(eb[v]->ver);
        if (nv == v) {
          eb[v] = pop(eb[v]);
          continue;
        }
        break;
      }
      if (nv == -1)
        return LINF;
      ans += eb[v]->x;
      eb[v]->add(-eb[v]->x);
#ifdef ANS
      int ei = eb[v]->ei;
      eout[edges[ei].x].push_back({++cnum, ei});
#endif
      if (!done[nv]) {
        pv[v] = nv;
        v = nv;
        continue;
      }
      if (done[nv] != tt)
        break;
```

```
      int v1 = nv;
      while (v1 != v) {
        eb[v] = merge(eb[v], eb[v1]);
        dsu.merge(v, v1);
        v1 = dsu.get(pv[v1]);
      }
    }
  }
#ifdef ANS
  memset(answer, -1, sizeof(int) * n);
  answer[root] = 0;
  set<ipair> es(all(eout[root]));
  while (!es.empty()) {
    auto it = es.begin();
    int ei = it->second;
    es.erase(it);
    int nv = edges[ei].y;
    if (answer[nv] != -1)
      continue;
    answer[nv] = ei;
    es.insert(all(eout[nv]));
  }
  answer[root] = -1;
#endif
  return ans;
}
/* Usage: twoc::init(vertex_count);
 *        twoc::addEdge(v1, v2, cost);
 *        twoc::solve(root); - returns cost or
 ↪  LINF
 * twoc::answer contains index of ingoing edge
 ↪  for each vertex
 */
} // namespace twoc
```

4.2 Dominator tree

```
namespace domtree {
const int K = 18;
const int N = 1 << K;

int n, root;
vector<int> e[N], g[N];
int sdom[N], dom[N];
int p[N][K], h[N], pr[N];
int in[N], out[N], tmr, rev[N];

void init(int _n, int _root) {
  n = _n;
  root = _root;
  tmr = 0;
  for (int i = 0; i < n; i++) {
    e[i].clear();
    g[i].clear();
    in[i] = -1;
  }
}

void addEdge(int u, int v) {
  e[u].push_back(v);
  g[v].push_back(u);
}

void dfs(int v) {
  in[v] = tmr++;
  for (int to : e[v]) {
    if (in[to] != -1)
      continue;
    pr[to] = v;
    dfs(to);
  }
  out[v] = tmr - 1;
}

int lca(int u, int v) {
  if (h[u] < h[v])
    swap(u, v);
  for (int i = 0; i < K; i++)
    if ((h[u] - h[v]) & (1 << i))
      u = p[u][i];
  if (u == v)
    return u;
  for (int i = K - 1; i >= 0; i--) {
    if (p[u][i] != p[v][i]) {
      u = p[u][i];
      v = p[v][i];
    }
  }
  return p[u][0];
}

void solve(int _n, int _root, vector<pair<int,
 ↪  int>> _edges) {
  init(_n, _root);
  for (auto ed : _edges)
    addEdge(ed.first, ed.second);

  dfs(root);
  for (int i = 0; i < n; i++)
    if (in[i] != -1)
      rev[in[i]] = i;
  segtree tr(tmr); // a[i]:=min(a[i],x) and
   ↪  return a[i]
  for (int i = tmr - 1; i >= 0; i--) {
    int v = rev[i];
    int cur = i;
    for (int to : g[v]) {
      if (in[to] == -1)
        continue;
      if (in[to] < in[v])
        cur = min(cur, in[to]);
      else
        cur = min(cur, tr.get(in[to]));
    }
    sdom[v] = rev[cur];
    tr.upd(in[v], out[v], in[sdom[v]]);
  }
  for (int i = 0; i < tmr; i++) {
    int v = rev[i];
    if (i == 0) {
      dom[v] = v;
      h[v] = 0;
    } else {
      dom[v] = lca(sdom[v], pr[v]);
```

```
      h[v] = h[dom[v]] + 1;
    }
    p[v][0] = dom[v];
    for (int j = 1; j < K; j++)
      p[v][j] = p[p[v][j - 1]][j - 1];
  }
  for (int i = 0; i < n; i++)
    if (in[i] == -1)
      dom[i] = -1;
}
} // namespace domtree
```

4.3 General matching

```
// COPYPASTED FROM E-MAXX
namespace general_matching {
const int MAXN = 256;
int n;
vector<int> g[MAXN];
int match[MAXN], p[MAXN], base[MAXN], q[MAXN];
bool used[MAXN], blossom[MAXN];

int lca(int a, int b) {
  bool used[MAXN] = {0};
  for (;;) {
    a = base[a];
    used[a] = true;
    if (match[a] == -1)
      break;
    a = p[match[a]];
  }
  for (;;) {
    b = base[b];
    if (used[b])
      return b;
    b = p[match[b]];
  }
}

void mark_path(int v, int b, int children) {
  while (base[v] != b) {
    blossom[base[v]] = blossom[base[match[v]]] =
    ↪  true;
    p[v] = children;
    children = match[v];
    v = p[match[v]];
  }
}

int find_path(int root) {
  memset(used, 0, sizeof used);
  memset(p, -1, sizeof p);
  for (int i = 0; i < n; ++i)
    base[i] = i;

  used[root] = true;
  int qh = 0, qt = 0;
  q[qt++] = root;
  while (qh < qt) {
    int v = q[qh++];
    for (size_t i = 0; i < g[v].size(); ++i) {
      int to = g[v][i];
      if (base[v] == base[to] || match[v] == to)
        continue;
      if (to == root || (match[to] != -1 &&
      ↪  p[match[to]] != -1)) {
        int curbase = lca(v, to);
        memset(blossom, 0, sizeof blossom);
        mark_path(v, curbase, to);
        mark_path(to, curbase, v);
        for (int i = 0; i < n; ++i)
          if (blossom[base[i]]) {
            base[i] = curbase;
            if (!used[i]) {
              used[i] = true;
              q[qt++] = i;
            }
          }
      } else if (p[to] == -1) {
        p[to] = v;
        if (match[to] == -1)
          return to;
        to = match[to];
        used[to] = true;
        q[qt++] = to;
      }
    }
  }
  return -1;
}

vector<pair<int, int>> solve(int _n,
↪  vector<pair<int, int>> edges) {
  n = _n;
  for (int i = 0; i < n; i++)
    g[i].clear();
  for (auto o : edges) {
    g[o.first].push_back(o.second);
    g[o.second].push_back(o.first);
  }
  memset(match, -1, sizeof match);
  for (int i = 0; i < n; ++i) {
    if (match[i] == -1) {
      int v = find_path(i);
      while (v != -1) {
        int pv = p[v], ppv = match[pv];
        match[v] = pv, match[pv] = v;
        v = ppv;
      }
    }
  }
  vector<pair<int, int>> ans;
  for (int i = 0; i < n; i++) {
    if (match[i] > i) {
      ans.push_back(make_pair(i, match[i]));
    }
  }
  return ans;
}
} // namespace general_matching
```

4.4 Gomory-Hu tree

```
// graph has n nodes
// reset() clears all flows in graph
// dinic(s, t) pushes max flow from s to t
// dist[v] is distance from s to v in residual
↪ network

vector<vector<long long>> prec;
void buildTree() {
	vector<int> p(n, 0);
	prec = vector<vector<long long>>(n, vector<long
	↪ long>(n, inff));
	for (int i = 1; i < n; i++) {
		reset();
		long long f = dinic(i, p[i]);
		for (int j = 0; j < n; j++) {
			if (j != i && dist[j] < inff && p[j] ==
			↪ p[i]) {
				p[j] = i;
			}
		}
		prec[p[i]][i] = prec[i][p[i]] = f;
		for (int j = 0; j < i; j++) {
			prec[i][j] = prec[j][i] =
			↪ min(prec[j][p[i]], f);
		}
		{
			int j = p[i];
			if (dist[p[j]] < inff) {
				p[i] = p[j];
				p[j] = i;
			}
		}
	}
}

long long fastFlow(int S, int T) {
	return prec[S][T];
}
```

4.5 Hungarian algorithm

```
namespace hungary {
const int N = 210;

int a[N][N];
int ans[N];

int calc(int n, int m) {
	++n, ++m;
	vector<int> u(n), v(m), p(m), prev(m);
	for (int i = 1; i < n; ++i) {
		p[0] = i;
		int x = 0;
		vector<int> mn(m, INF);
		vector<int> was(m, 0);
		while (p[x]) {
			was[x] = 1;
			int ii = p[x], dd = INF, y = 0;
			for (int j = 1; j < m; ++j)
				if (!was[j]) {
					int cur = a[ii][j] - u[ii] - v[j];
					if (cur < mn[j])
						mn[j] = cur, prev[j] = x;
					if (mn[j] < dd)
						dd = mn[j], y = j;
				}
			for (int j = 0; j < m; ++j) {
				if (was[j])
					u[p[j]] += dd, v[j] -= dd;
				else
					mn[j] -= dd;
			}
			x = y;
		}
		while (x) {
			int y = prev[x];
			p[x] = p[y];
			x = y;
		}
	}
	for (int j = 1; j < m; ++j) {
		ans[p[j]] = j;
	}
	return -v[0];
}
// How to use:
// * Set values to a[1..n][1..m] (n <= m)
// * Run calc(n, m) to find minimum
// * Optimal edges are (i, ans[i]) for i = 1..n
// * Everything works on negative numbers
//
// !!! I don't understand this code, it's
↪ copypasted from e-maxx
} // namespace hungary
```

4.6 Link-Cut Tree

```
#include <cassert>
#include <cstdio>
#include <iostream>

using namespace std;

// BEGIN ALGO

const int MAXN = 110000;

typedef struct _node {
	_node *l, *r, *p, *pp;
	int size;
	bool rev;
	_node();
	explicit _node(nullptr_t) {
		l = r = p = pp = this;
		size = rev = 0;
	}
	void push() {
		if (rev) {
			l->rev ^= 1;
			r->rev ^= 1;
			rev = 0;
			swap(l, r);
		}
```

```
  }
  void update();
} * node;
node None = new _node(nullptr);
node v2n[MAXN];
_node::_node() {
  l = r = p = pp = None;
  size = 1;
  rev = false;
}
void _node::update() {
  size = (this != None) + l->size + r->size;
  l->p = r->p = this;
}
void rotate(node v) {
  assert(v != None && v->p != None);
  assert(!v->rev);
  assert(!v->p->rev);
  node u = v->p;
  if (v == u->l)
    u->l = v->r, v->r = u;
  else
    u->r = v->l, v->l = u;
  swap(u->p, v->p);
  swap(v->pp, u->pp);
  if (v->p != None) {
    assert(v->p->l == u || v->p->r == u);
    if (v->p->r == u)
      v->p->r = v;
    else
      v->p->l = v;
  }
  u->update();
  v->update();
}
void bigRotate(node v) {
  assert(v->p != None);
  v->p->p->push();
  v->p->push();
  v->push();
  if (v->p->p != None) {
    if ((v->p->l == v) ^ (v->p->p->r == v->p))
      rotate(v->p);
    else
      rotate(v);
  }
  rotate(v);
}
inline void Splay(node v) {
  while (v->p != None)
    bigRotate(v);
}
inline void splitAfter(node v) {
  v->push();
  Splay(v);
  v->r->p = None;
  v->r->pp = v;
  v->r = None;
  v->update();
}
void expose(int x) {
  node v = v2n[x];
  splitAfter(v);
  while (v->pp != None) {
    assert(v->p == None);
    splitAfter(v->pp);
    assert(v->pp->r == None);
    assert(v->pp->p == None);
    assert(!v->pp->rev);
    v->pp->r = v;
    v->pp->update();
    v = v->pp;
    v->r->pp = None;
  }
  assert(v->p == None);
  Splay(v2n[x]);
}
inline void makeRoot(int x) {
  expose(x);
  assert(v2n[x]->p == None);
  assert(v2n[x]->pp == None);
  assert(v2n[x]->r == None);
  v2n[x]->rev ^= 1;
}
inline void link(int x, int y) {
  makeRoot(x);
  v2n[x]->pp = v2n[y];
}
inline void cut(int x, int y) {
  expose(x);
  Splay(v2n[y]);
  if (v2n[y]->pp != v2n[x]) {
    swap(x, y);
    expose(x);
    Splay(v2n[y]);
    assert(v2n[y]->pp == v2n[x]);
  }
  v2n[y]->pp = None;
}
inline int get(int x, int y) {
  if (x == y)
    return 0;
  makeRoot(x);
  expose(y);
  expose(x);
  Splay(v2n[y]);
  if (v2n[y]->pp != v2n[x])
    return -1;
  return v2n[y]->size;
}

// END ALGO

_node mem[MAXN];

int main() {
  freopen("linkcut.in", "r", stdin);
  freopen("linkcut.out", "w", stdout);

  int n, m;
  scanf("%d %d", &n, &m);

  for (int i = 0; i < n; i++)
    v2n[i] = &mem[i];
```

```cpp
    for (int i = 0; i < m; i++) {
        int a, b;
        if (scanf(" link %d %d", &a, &b) == 2)
            link(a - 1, b - 1);
        else if (scanf(" cut %d %d", &a, &b) == 2)
            cut(a - 1, b - 1);
        else if (scanf(" get %d %d", &a, &b) == 2)
            printf("%d\n", get(a - 1, b - 1));
        else
            assert(false);
    }
    return 0;
}
```

4.7 Push-Relabel

```cpp
struct edge_t {
    int to;
    int next;
    int64_t flow;
    int64_t capacity;
};

int main() {
    int n = input<int>();
    int m = input<int>();
    int S = 0;
    int T = n - 1;

    vector<edge_t> edges;
    vector<int> head(n, -1);

    auto add_edge = [&](int v, int u, int cap, int rcap) {
        edges.push_back(edge_t {u, head[v], 0, cap});
        head[v] = SZ(edges) - 1;
        edges.push_back(edge_t {v, head[u], 0, rcap});
        head[u] = SZ(edges) - 1;
    };

    for (int i = 0; i < m; ++i) {
        int v, u, cap;
        cin >> v >> u >> cap;
        --v, --u;
        add_edge(v, u, cap, 0);
    }

    vector<int> d(n);
    vector<int64_t> exc(n);
    d[S] = n;

    auto push_edge = [&](int e, int64_t W) {
        int to = edges[e].to;
        int from = edges[e ^ 1].to;
        edges[e].flow += W;
        edges[e ^ 1].flow -= W;
        exc[from] -= W;
        exc[to] += W;
    };

    auto global_relabel = [&]() {
        for (int v = 0; v < n; ++v)
            if (v != S and v != T)
                d[v] = -1;

        for (int fixed: {T, S}) {
            queue<int> q;
            q.push(fixed);
            while (not q.empty()) {
                int v = q.front();
                q.pop();
                for (int e = head[v]; e != -1; e = edges[e].next) {
                    if (edges[e].to != S and edges[e^1].flow != edges[e^1].capacity and d[edges[e].to] == -1) {
                        d[edges[e].to] = d[v] + 1;
                        q.push(edges[e].to);
                    }
                }
            }
        }

        for (int v = 0; v < n; ++v)
            if (d[v] == -1)
                d[v] = 2 * n - 1;
    };

    for (int e = head[S]; e != -1; e = edges[e].next) {
        push_edge(e, edges[e].capacity);
    }

    vector<char> in_queue(n, false);
    queue<int> que;

    for (int v = 0; v < n; ++v)
        if (v != S and v != T and exc[v] > 0) {
            in_queue[v] = 1;
            que.push(v);
        }

    int processed = 0;
    while (not que.empty()) {
        if (++processed >= 3 * n) {
            processed -= 3 * n;
            global_relabel();
        }

        int v = que.front();
        que.pop();
        in_queue[v] = false;

        if (exc[v] == 0)
            continue;

        int new_d = TYPEMAX(int);
        for (int e = head[v]; e != -1; e = edges[e].next) {
            if (edges[e].flow == edges[e].capacity)
                continue;
```

```
            if (exc[v] == 0)
                break;

            if (d[v] != d[edges[e].to] + 1) {
                new_d = min(new_d, 1 + d[edges[e].to]);
                continue;
            }

            int delta = min(edges[e].capacity -
            ↪  edges[e].flow, exc[v]);
            push_edge(e, delta);

            if (edges[e].flow < edges[e].capacity)
                new_d = min(new_d, 1 + d[edges[e].to]);

            if (exc[edges[e].to] > 0 and edges[e].to !=
            ↪  S and edges[e].to != T and not
            ↪  in_queue[edges[e].to]) {
                que.push(edges[e].to);
                in_queue[edges[e].to] = 1;
            }
        }

        if (exc[v]) {
            que.push(v);
            in_queue[v] = true;
            d[v] = new_d;
        }
    }

    cout << exc[T] << "\n";
    for (int i = 0; i < SZ(edges); i += 2)
        cout << edges[i].flow << "\n";

    return 0;
}
```

4.8 Smith algorithm (Game on cyclic graph)

```
const int N = 1e5 + 10;

struct graph {
    int n;

    vi v[N];
    vi vrev[N];

    void read() {
        int m;
        scanf("%d%d", &n, &m);
        forn(i, m) {
            int x, y;
            scanf("%d%d", &x, &y);
            --x, --y;
            v[x].pb(y);
            vrev[y].pb(x);
        }
    }

    int deg[N], cnt[N], used[N], f[N];
    int q[N], st, en;

    set<int> s[N];

    void calc() {
        for (int x = 0; x < n; ++x)
            f[x] = -1, cnt[x] = 0;
        int val = 0;
        while (1) {
            st = en = 0;
            for (int x = 0; x < n; ++x) {
                deg[x] = 0;
                used[x] = 0;
                for (int y : v[x])
                    if (f[y] == -1)
                        deg[x]++;
            }
            for (int x = 0; x < n; ++x)
                if (!deg[x] && f[x] == -1 && cnt[x] ==
                ↪  val) {
                    q[en++] = x;
                    f[x] = val;
                }
            if (!en)
                break;
            while (st < en) {
                int x = q[st];
                st++;
                for (int y : vrev[x]) {
                    if (used[y] == 0 && f[y] == -1) {
                        used[y] = 1;
                        cnt[y]++;
                        for (int z : vrev[y]) {
                            deg[z]--;
                            if (f[z] == -1 && deg[z] == 0 &&
                            ↪  cnt[z] == val) {
                                f[z] = val;
                                q[en++] = z;
                            }
                        }
                    }
                }
            }
            val++;
        }
        for (int x = 0; x < n; ++x)
            eprintf("%d%c", f[x], " \n"[x + 1 == n]);
        for (int x = 0; x < n; ++x)
            if (f[x] == -1) {
                for (int y : v[x])
                    if (f[y] != -1)
                        s[x].insert(f[y]);
            }
    }
} g1, g2;

string get(int x, int y) {
    int f1 = g1.f[x], f2 = g2.f[y];
    if (f1 == -1 && f2 == -1)
        return "draw";
    if (f1 == -1) {
        if (g1.s[x].count(f2))
            return "first";
```

```
        return "draw";
    }
    if (f2 == -1) {
        if (g2.s[y].count(f1))
            return "first";
        return "draw";
    }
    if (f1 ^ f2)
        return "first";
    return "second";
}
```

4.9 Stoer-Vagner algorithm (Global min-cut)

```
const int MAXN = 500;
int n, g[MAXN][MAXN];
int best_cost = 1000000000;
vector<int> best_cut;

void mincut() {
    vector<int> v[MAXN];
    for (int i = 0; i < n; ++i)
        v[i].assign(1, i);
    int w[MAXN];
    bool exist[MAXN], in_a[MAXN];
    memset(exist, true, sizeof exist);
    for (int ph = 0; ph < n - 1; ++ph) {
        memset(in_a, false, sizeof in_a);
        memset(w, 0, sizeof w);
        for (int it = 0, prev; it < n - ph; ++it) {
            int sel = -1;
            for (int i = 0; i < n; ++i)
                if (exist[i] && !in_a[i] && (sel == -1 ||
                    w[i] > w[sel]))
                    sel = i;
            if (it == n - ph - 1) {
                if (w[sel] < best_cost)
                    best_cost = w[sel], best_cut = v[sel];
                v[prev].insert(v[prev].end(),
                    v[sel].begin(), v[sel].end());
                for (int i = 0; i < n; ++i)
                    g[prev][i] = g[i][prev] += g[sel][i];
                exist[sel] = false;
            } else {
                in_a[sel] = true;
                for (int i = 0; i < n; ++i)
                    w[i] += g[sel][i];
                prev = sel;
            }
        }
    }
}
```

5 Matroids

5.1 Matroids intersection

```
// check(ctaken, 1) -- first matroid
// check(ctaken, 2) -- second matroid
vector<char> taken(m);
while (1) {
    vector<vector<int>> e(m);
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < m; j++) {
            if (taken[i] && !taken[j]) {
                auto ctaken = taken;
                ctaken[i] = 0;
                ctaken[j] = 1;
                if (check(ctaken, 2)) {
                    e[i].push_back(j);
                }
            }
            if (!taken[i] && taken[j]) {
                auto ctaken = taken;
                ctaken[i] = 1;
                ctaken[j] = 0;
                if (check(ctaken, 1)) {
                    e[i].push_back(j);
                }
            }
        }
    }
    vector<int> type(m);
    // 0 -- cant, 1 -- can in \2, 2 -- can in \1
    for (int i = 0; i < m; i++) {
        if (!taken[i]) {
            auto ctaken = taken;
            ctaken[i] = 1;
            if (check(ctaken, 2))
                type[i] |= 1;
        }
        if (!taken[i]) {
            auto ctaken = taken;
            ctaken[i] = 1;
            if (check(ctaken, 1))
                type[i] |= 2;
        }
    }
    vector<int> w(m);
    for (int i = 0; i < m; i++) {
        w[i] = taken[i] ? ed[i].c : -ed[i].c;
    }
    vector<pair<int, int>> d(m, {INF, 0});
    for (int i = 0; i < m; i++) {
        if (type[i] & 1)
            d[i] = {w[i], 0};
    }
    vector<int> pr(m, -1);
    while (1) {
        vector<pair<int, int>> nd = d;
        for (int i = 0; i < m; i++) {
            if (d[i].first == INF)
                continue;
            for (int to : e[i]) {
                if (nd[to] > make_pair(d[i].first +
                    w[to], d[i].second + 1)) {
                    nd[to] = make_pair(d[i].first + w[to],
                        d[i].second + 1);
                    pr[to] = i;
                }
            }
        }
        if (d == nd)
```

```
            break;
        d = nd;
    }
    int v = -1;
    for (int i = 0; i < m; i++) {
        if ((d[i].first < INF && (type[i] & 2)) &&
          ↪ (v == -1 || d[i] < d[v]))
            v = i;
    }
    if (v == -1)
        break;
    while (v != -1) {
        sum += w[v];
        taken[v] ^= 1;
        v = pr[v];
    }
    ans[--cnt] = sum;
}
```

6 Numeric

6.1 Berlekamp-Massey Algorithm

```
vector<int> berlekamp(vector<int> s) {
    int l = 0;
    vector<int> la(1, 1);
    vector<int> b(1, 1);
    for (int r = 1; r <= (int)s.size(); r++) {
        int delta = 0;
        for (int j = 0; j <= l; j++) {
            delta = (delta + 1LL * s[r - 1 - j] *
              ↪ la[j]) % MOD;
        }
        b.insert(b.begin(), 0);
        if (delta != 0) {
            vector<int> t(max(la.size(), b.size()));
            for (int i = 0; i < (int)t.size(); i++) {
                if (i < (int)la.size())
                    t[i] = (t[i] + la[i]) % MOD;
                if (i < (int)b.size())
                    t[i] = (t[i] - 1LL * delta * b[i] % MOD
                      ↪ + MOD) % MOD;
            }
            if (2 * l <= r - 1) {
                b = la;
                int od = inv(delta);
                for (int &x : b)
                    x = 1LL * x * od % MOD;
                l = r - l;
            }
            la = t;
        }
    }
    assert((int)la.size() == l + 1);
    assert(l * 2 + 30 < (int)s.size());
    reverse(la.begin(), la.end());
    return la;
}

vector<int> mul(vector<int> a, vector<int> b) {
    vector<int> c(a.size() + b.size() - 1);
    for (int i = 0; i < (int)a.size(); i++) {
        for (int j = 0; j < (int)b.size(); j++) {
            c[i + j] = (c[i + j] + 1LL * a[i] * b[j]) %
              ↪ MOD;
        }
    }
    vector<int> res(c.size());
    for (int i = 0; i < (int)res.size(); i++)
        res[i] = c[i] % MOD;
    return res;
}

vector<int> mod(vector<int> a, vector<int> b) {
    if (a.size() < b.size())
        a.resize(b.size() - 1);

    int o = inv(b.back());
    for (int i = (int)a.size() - 1; i >=
      ↪ (int)b.size() - 1; i--) {
        if (a[i] == 0)
            continue;
        int coef = 1LL * o * (MOD - a[i]) % MOD;
        for (int j = 0; j < (int)b.size(); j++) {
            a[i - (int)b.size() + 1 + j] =
                    (a[i - (int)b.size() + 1 + j] + 1LL *
                      ↪ coef * b[j]) % MOD;
        }
    }
    while (a.size() >= b.size()) {
        assert(a.back() == 0);
        a.pop_back();
    }
    return a;
}

vector<int> bin(int n, vector<int> p) {
    vector<int> res(1, 1);
    vector<int> a(2);
    a[1] = 1;
    while (n) {
        if (n & 1)
            res = mod(mul(res, a), p);
        a = mod(mul(a, a), p);
        n >>= 1;
    }
    return res;
}

int f(vector<int> t, int m) {
    vector<int> v = berlekamp(t);
    vector<int> o = bin(m - 1, v);
    int res = 0;
    for (int i = 0; i < (int)o.size(); i++)
        res = (res + 1LL * o[i] * t[i]) % MOD;
    return res;
}
```

6.2 Burnside's lemma

$$|X/G| = \frac{1}{|G|}\sum_{g\in G}|St(g)|$$

$St(g)$ denote the set of elements in X that are fixed by g, i.e. $St(g) = \{x \in X | gx = x\}$.

6.3 Chinese remainder theorem

```
int CRT(int a1, int m1, int a2, int m2) {
| return (a1 - a2 % m1 + m1) * (ll)rev(m2, m1) %
  ↪ m1 * m2 + a2;
}
```

6.4 AND/OR/XOR convolution

```
// Transform to a basis with fast convolutions of
↪ the form c[z] = \sum_{z=x⊕y} a[x] · b[y],
// where ⊕ is one of AND, OR, XOR.
// The size of a must be a power of two.

void FST(vector<int> &a, bool inv) {
| int n = szof(a);
| for (int step = 1; step < n; step *= 2) {
| | for (int i = 0; i < n; i += 2 * step) {
| | | for (j = i; j < i + step; ++j) {
| | | | int &u = a[j], &v = a[j + step];
| | | | tie(u, v) =
| | | | | inv ? pii(v - u, u) : pii(v, u + v); //
  ↪ AND
| | | | | inv ? pii(v, u - v) : pii(u + v, u); //
  ↪ OR
| | | | | pii(u + v, u - v); // XOR
| | | }
| | }
| }
| if (inv)
| | for (int &x : a)
| | | x /= sz(a); // XOR only
}

vector<int> conv(vector<int> a, vector<int> b) {
| FST(a, 0);
| FST(b, 0);

| for (int i = 0; i < szof(a); ++i) {
| | a[i] *= b[i];
| }

| FST(a, 1);
| return a;
}
```

6.5 Counting size of the maximum general matching

In order to find a size of the maximum matching:

1. Build Tutte matrix. (x_{ij} are random numbers)

$$A_{ij} = \begin{cases} x_{ij} & \text{if edge } (i,j) \text{ exists and } i < j \\ -x_{ij} & \text{if edge } (i,j) \text{ exists and } i > j \\ 0 & \textit{otherwise} \end{cases}$$

2. The size of the maximum matching equals to the size of the maximum independent set divided by 2.

3. $(A^{-1})_{ji} \neq 0$ iff edge (i,j) belongs to some complete matching.

6.6 Counting number of spanning trees

In order to count number of spanning trees:

1. Build the Laplacian matrix. That is difference between the degree matrix and the adjacency matrix.

2. Delete any row and any column of this matrix.

3. Calculate it's determinant.

6.7 Some formulas

- $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$
- $\sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4}$
- $\sum_{i=1}^{n} i^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$
- $\sum_{k=0}^{n} k\binom{n}{k} = n2^{n-1}$
- $\sum_{k=0}^{n} \binom{n}{k}^2 = \binom{2n}{n}$

6.8 Miller–Rabin primality test

```
// assume p > 1
bool isprime(ll p) {
| const int a[] = {2, 3, 5, 7, 11, 13, 17, 19,
  ↪ 23, 0};
| ll d = p - 1;
| int cnt = 0;
| while (!(d & 1)) {
| | d >>= 1;
| | cnt++;
| }
| for (int i = 0; a[i]; i++) {
| | if (p == a[i]) {
| | | return true;
| | }
| | if (!(p % a[i])) {
| | | return false;
| | }
| }
| for (int i = 0; a[i]; i++) {
| | ll cur = mpow(a[i], d, p); // a[i] ^ d (mod
  ↪ p)
| | if (cur == 1) {
| | | continue;
| | }
| | bool good = false;
| | for (int j = 0; j < cnt; j++) {
| | | if (cur == p - 1) {
| | | | good = true;
| | | | break;
| | | }
| | | cur = mult(cur, cur);
| | }
| | if (!good) {
| | | return false;
| | }
| }
| return true;
}
```

6.9 Taking by modullo (Inline assembler)

```
inline void fasterLLDivMod(ull x, uint y, uint
    &out_d, uint &out_m) {
  uint xh = (uint)(x >> 32), xl = (uint)x, d, m;
#ifdef __GNUC__
  asm(
    "divl %4; \n\t"
    : "=a" (d), "=d" (m)
    : "d" (xh), "a" (xl), "r" (y)
  );
#else
  __asm {
    mov edx, dword ptr[xh];
    mov eax, dword ptr[xl];
    div dword ptr[y];
    mov dword ptr[d], eax;
    mov dword ptr[m], edx;
  };
#endif
  out_d = d; out_m = m;
}
```

6.10 First solution of $(p+step \cdot x)$ mod $mod < l$

```
// returns value of (p + step * x), i.e. number
    of steps x = (ans - p) / step (mod mod)
int smart_calc(int mod, int step, int l, int p) {
  if (p < l) {
    return p;
  }
  int d = (mod - p + step - 1) / step;
  int np = (p + d * step) % mod;
  if (np < l) {
    return np;
  }
  int res = smart_calc(step, mod % step, l, l +
    step - 1 - np);
  return l - 1 - res;
}
```

6.11 Multiplication by modulo in `long double`

```
ll mul(ll a, ll b, ll m) { // works for MOD 8e18
  ll k = (ll)((long double)a * b / m);
  ll r = a * b - m * k;
  if (r < 0)
    r += m;
  if (r >= m)
    r -= m;
  return r;
}
```

6.12 Numerical integration

```
function<dbl(dbl, dbl, function<dbl(dbl)>)> f =
    [&](dbl L, dbl R, function<dbl(dbl)> g) {
  const int ITERS = 1000000;
  dbl ans = 0;
  dbl step = (R - L) * 1.0 / ITERS;
  for (int it = 0; it < ITERS; it++) {
    double xl = L + step * it;
    double xr = L + step * (it + 1);
    dbl x1 = (xl + xr) / 2;
    dbl x0 = x1 - (x1 - xl) * sqrt(3.0 / 5);
    dbl x2 = x1 + (x1 - xl) * sqrt(3.0 / 5);
    ans += (5 * g(x0) + 8 * g(x1) + 5 * g(x2)) /
      18 * step;
  }
  return ans;
};
```

6.13 Pollard's rho algorithm

```
namespace pollard {
using math::p;

vector<pair<ll, int>> getFactors(ll N) {
  vector<ll> primes;

  const int MX = 1e5;
  const ll MX2 = MX * (ll)MX;

  assert(MX <= math::maxP && math::pc > 0);

  function<void(ll)> go = [&go, &primes](ll n) {
    for (ll x : primes)
      while (n % x == 0)
        n /= x;
    if (n == 1)
      return;
    if (n > MX2) {
      auto F = [&](ll x) {
        ll k = ((long double)x * x) / n;
        ll r = (x * x - k * n + 3) % n;
        return r < 0 ? r + n : r;
      };
      ll x = mt19937_64() % n, y = x;
      const int C = 3 * pow(n, 0.25);

      ll val = 1;
      forn(it, C) {
        x = F(x), y = F(F(y));
        if (x == y)
          continue;
        ll delta = abs(x - y);
        ll k = ((long double)val * delta) / n;
        val = (val * delta - k * n) % n;
        if (val < 0)
          val += n;
        if (val == 0) {
          ll g = __gcd(delta, n);
          go(g), go(n / g);
          return;
        }
        if ((it & 255) == 0) {
          ll g = __gcd(val, n);
          if (g != 1) {
            go(g), go(n / g);
            return;
          }
        }
      }
```

```
    }
    primes.pb(n);
  };

  ll n = N;

  for (int i = 0; i < math::pc && p[i] < MX; ++i)
    if (n % p[i] == 0) {
      primes.pb(p[i]);
      while (n % p[i] == 0)
        n /= p[i];
    }

  go(n);

  sort(primes.begin(), primes.end());

  vector<pair<ll, int>> res;
  for (ll x : primes) {
    int cnt = 0;
    while (N % x == 0) {
      cnt++;
      N /= x;
    }
    res.push_back({x, cnt});
  }
  return res;
}
} // namespace pollard
```

6.14 Polynom division and inversion

```
poly inv(poly A, int n) // returns A^-1 mod x^n
{
  assert(sz(A) && A[0] != 0);
  A.cut(n);

  auto cutPoly = [](poly &from, int l, int r) {
    poly R;
    R.v.resize(r - l);
    for (int i = l; i < r; ++i) {
      if (i < sz(from))
        R[i - l] = from[i];
    }
    return R;
  };

  function<int(int, int)> rev = [&rev](int x, int
  ↪ m) -> int {
    if (x == 1)
      return 1;
    return (1 - rev(m % x, x) * (ll)m) / x + m;
  };

  poly R({rev(A[0], mod)});
  for (int k = 1; k < n; k <<= 1) {
    poly A0 = cutPoly(A, 0, k);
    poly A1 = cutPoly(A, k, 2 * k);
    poly H = A0 * R;
    H = cutPoly(H, k, 2 * k);
    poly R1 = (((A1 * R).cut(k) + H) * (poly({0})
    ↪ - R)).cut(k);
    R.v.resize(2 * k);
    forn(i, k) R[i + k] = R1[i];
  }
  return R.cut(n).norm();
}

pair<poly, poly> divide(poly A, poly B) {
  if (sz(A) < sz(B))
    return {poly({0}), A};

  auto rev = [](poly f) {
    reverse(all(f.v));
    return f;
  };

  poly q =
      rev((inv(rev(B), sz(A) - sz(B) + 1) *
      ↪ rev(A)).cut(sz(A) - sz(B) + 1));
  poly r = A - B * q;

  return {q, r};
}
```

6.15 Polynom roots

```
const double EPS = 1e-9;
double cal(const vector<double> &coef, double x)
↪ {
  double e = 1, s = 0;
  for (double i : coef) s += i * e, e *= x;
  return s;
}

int dblcmp(double x) {
  if (x < -EPS) return -1;
  if (x > EPS) return 1;
  return 0;
}

double find(const vector<double> &coef, double l,
↪ double r) {
  int sl = dblcmp(cal(coef, l)), sr =
  ↪ dblcmp(cal(coef, r));
  if (sl == 0) return l;
  if (sr == 0) return r;
  for (int tt = 0; tt < 100 && r - l > EPS; ++tt)
  ↪ {
    double mid = (l + r) / 2;
    int smid = dblcmp(cal(coef, mid));
    if (smid == 0) return mid;
    if (sl * smid < 0) r = mid;
    else l = mid;
  }
  return (l + r) / 2;
}

vector<double> rec(const vector<double> &coef,
↪ int n) {
  vector<double> ret; //
  ↪ c[0]+c[1]*x+c[2]*x^2+...+c[n]*x^n, c[n]==1
  if (n == 1) {
    ret.push_back(-coef[0]);
```

```
    return ret;
  }
  vector<double> dcoef(n);
  for (int i = 0; i < n; ++i) dcoef[i] = coef[i +
  ↪  1] * (i + 1) / n;
  double b = 2; // fujiwara bound
  for (int i = 0; i <= n; ++i) b = max(b, 2 *
  ↪  pow(fabs(coef[i]), 1.0 / (n - i)));
  vector<double> droot = rec(dcoef, n - 1);
  droot.insert(droot.begin(), -b);
  droot.push_back(b);
  for (int i = 0; i + 1 < droot.size(); ++i) {
    int sl = dblcmp(cal(coef, droot[i])), sr =
    ↪  dblcmp(cal(coef, droot[i + 1]));
    if (sl * sr > 0) continue;
    ret.push_back(find(coef, droot[i], droot[i +
    ↪  1]));
  }
  return ret;
}

vector<double> solve(vector<double> coef) {
  int n = coef.size() - 1;
  while (coef.back() == 0) coef.pop_back(), --n;
  for (int i = 0; i <= n; ++i) coef[i] /=
  ↪  coef[n];
  return rec(coef, n);
}
```

6.16 Simplex method

```
struct simplex_t {
  vector<vector<double>> mat;
  int EQ, VARS, p_row;

  vector<int> column;

  void row_subtract(int what, int from, double x)
  ↪  {
    for (int i = 0; i <= VARS; ++i)
      mat[from][i] -= mat[what][i] * x;
  }

  void row_scale(int what, double x) {
    for (int i = 0; i <= VARS; ++i)
      mat[what][i] *= x;
  }

  void pivot(int var, int eq) {
    row_scale(eq, 1. / mat[eq][var]);

    for (int p = 0; p <= EQ; ++p)
      if (p != eq)
        row_subtract(eq, p, mat[p][var]);

    column[eq] = var;
  }

  void iterate() {
    while (true) {
      int j = 0;
      for (; j != VARS and mat[EQ][j] < eps; ++j)
      ↪  {}

      if (j == VARS)
        break;

      double lim = 1e100;
      int arg_min = -1;

      for (int p = 0; p != EQ; ++p) {
        if (mat[p][j] < eps)
          continue;

        double newlim = mat[p][VARS] / mat[p][j];
        if (newlim < lim)
          lim = newlim, arg_min = p;
      }

      if (arg_min == -1)
        throw "unbounded";

      pivot(j, arg_min);
    }
  }

  simplex_t(const vector<vector<double>>& mat_):
  ↪  mat(mat_) {
    for (int i = 0; i < SZ(mat); ++i) // fictuous
    ↪  variable
      mat[i].insert(mat[i].begin() + SZ(mat[i]) -
      ↪  1, double(0));

    EQ = SZ(mat), VARS = SZ(mat[0]) - 1;
    column.resize(EQ, -1);
    p_row = 0;

    for (int i = 0; i < VARS; ++i) {
      int p;
      for (p = p_row; p < EQ and abs(mat[p][i]) <
      ↪  eps; ++p) {}

      if (p == EQ)
        continue;

      swap(mat[p], mat[p_row]);
      column[p_row] = i;
      row_scale(p_row, 1. / mat[p_row][i]);

      for (p = 0; p != EQ; ++p)
        if (p != p_row)
          row_subtract(p_row, p, mat[p][i]);

      p_row += 1;
    }

    for (int p = p_row; p < EQ; ++p)
      if (abs(mat[p][VARS]) > eps)
        throw "unsolvable (bad equalities)";

    if (p_row) {
      int minr = 0;
      for (int i = 0; i < p_row; ++i)
```

```
                if (mat[i][VARS] < mat[minr][VARS])
                    minr = i;

            if (mat[minr][VARS] < -eps) {
                mat.push_back(vector<double>(VARS + 1));

                mat[EQ][VARS - 1] = -1;
                for (int i = 0; i != p_row; ++i)
                    mat[i][VARS - 1] = -1;

                pivot(VARS - 1, minr);
                iterate();

                if (abs(mat[EQ][VARS]) > eps)
                    throw "unsolvable";

                for (int c = 0; c != EQ; ++c)
                    if (column[c] == VARS - 1) {
                        int p = 0;
                        while (p != VARS - 1 and abs(mat[c][p]) < eps)
                            ++p;

                        assert(p != VARS - 1);
                        pivot(p, c);
                        break;
                    }

                for (int p = 0; p != EQ; ++p)
                    mat[p][VARS - 1] = 0;

                mat.pop_back();
            }
        }
    }

    double solve(vector<double> coeff, vector<double>& pans) {
        auto mat_orig = mat;
        auto col_orig = column;

        coeff.resize(VARS + 1);
        mat.push_back(coeff);

        for (int i = 0; i != p_row; ++i)
            row_subtract(i, EQ, mat[EQ][column[i]]);

        iterate();

        auto ans = -mat[EQ][VARS];
        if (not pans.empty()) {
            for (int i = 0; i < EQ; ++i) {
                assert(column[i] < VARS);
                pans[column[i]] = mat[i][VARS];
            }
        }

        mat = std::move(mat_orig);
        column = std::move(col_orig);
        return ans;
    }

    double solve_min(vector<double> coeff, vector<double>& pans) {
        for (double& elem: coeff)
            elem = -elem;

        return -solve(coeff, pans);
    }
};
```

6.17 Some integer sequences

Bell numbers:			
n	B_n	n	B_n
0	1	10	115 975
1	1	11	678 570
2	2	12	4 213 597
3	5	13	27 644 437
4	15	14	190 899 322
5	52	15	1 382 958 545
6	203	16	10 480 142 147
7	877	17	82 864 869 804
8	4 140	18	682 076 806 159
9	21 147	19	5 832 742 205 057

Numbers with many divisors:		
$x \leq$	x	$d(x)$
20	12	6
50	48	10
100	60	12
1000	840	32
10 000	9 240	64
100 000	83 160	128
10^6	720 720	240
10^7	8 648 640	448
10^8	91 891 800	768
10^9	931 170 240	1 344
10^{11}	97 772 875 200	4 032
10^{12}	963 761 198 400	6 720
10^{15}	866 421 317 361 600	26 880
10^{18}	897 612 484 786 617 600	103 680

Partitions of n into unordered summands					
n	$a(n)$	n	$a(n)$	n	$a(n)$
0	1	20	627	40	37 338
1	1	21	792	41	44 583
2	2	22	1 002	42	53 174
3	3	23	1 255	43	63 261
4	5	24	1 575	44	75 175
5	7	25	1 958	45	89 134
6	11	26	2 436	46	105 558
7	15	27	3 010	47	124 754
8	22	28	3 718	48	147 273
9	30	29	4 565	49	173 525
10	42	30	5 604	50	204 226
11	56	31	6 842	51	239 943
12	77	32	8 349	52	281 589
13	101	33	10 143	53	329 931
14	135	34	12 310	54	386 155
15	176	35	14 883	55	451 276
16	231	36	17 977	56	526 823
17	297	37	21 637	57	614 154
18	385	38	26 015	58	715 220
19	490	39	31 185	59	831 820
100	190 569 292				

7 Strings

7.1 Duval algorithm (Lyndon factorization)

```
void duval(string s) {
| int n = (int)s.length();
| int i = 0;
| while (i < n) {
| | int j = i + 1, k = i;
| | while (j < n && s[k] <= s[j]) {
| | | if (s[k] < s[j])
| | | | k = i;
| | | else
| | | | ++k;
| | | ++j;
| | }
| | while (i <= k) {
| | | cout << s.substr(i, j - k) << ' ';
| | | i += j - k;
| | }
| }
}
```

7.2 Palindromic tree

```
namespace eertree {
const int INF = 1e9;
const int N = 5e6 + 10;
char _s[N];
char *s = _s + 1;
int to[N][2];
int suf[N], len[N];
int sz, last;

const int odd = 1, even = 2, blank = 3;

void go(int &u, int pos) {
| while (u != blank && s[pos - len[u] - 1] !=
  ↪ s[pos]) {
| | u = suf[u];
| }
}

int add(int pos) {
| go(last, pos);
| int u = suf[last];
| go(u, pos);
| int c = s[pos] - 'a';
| int res = 0;
| if (!to[last][c]) {
| | res = 1;
| | to[last][c] = sz;
| | len[sz] = len[last] + 2;
| | suf[sz] = to[u][c];
| | sz++;
| }
| last = to[last][c];
| return res;
}

void init() {
| to[blank][0] = to[blank][1] = even;
| len[blank] = suf[blank] = INF;
| len[even] = 0, suf[even] = odd;
| len[odd] = -1, suf[odd] = blank;
| last = even;
| sz = 4;
}
} // namespace eertree
```

7.3 Manacher's algorithm

```
// returns vector ret of length (|s| * 2 - 1),
//   ret[i * 2] -- maximal length of palindrome
↪  with center in i-th symbol
//   ret[i * 2 + 1] -- maximal length of
↪  palindrome with center between i-th and (i +
↪  1)-th symbols
vector<int> find_palindromes(string const& s) {
| string tmp;
| for (char c : s) {
| | tmp += c;
| | tmp += '!';
| }
| tmp.pop_back();

| int c = 0, r = 1;
| vector<int> rad(szof(tmp));
| rad[0] = 1;
| for (int i = 1; i < szof(tmp); ++i) {
| | if (i < c + r) {
| | | rad[i] = min(c + r - i, rad[2 * c - i]);
| | }
| | while (i - rad[i] >= 0 && i + rad[i] <
    ↪ szof(tmp) && tmp[i - rad[i]] == tmp[i +
    ↪ rad[i]]) {
| | | ++rad[i];
| | }
| | if (i + rad[i] > c + r) {
| | | c = i;
| | | r = rad[i];
```

```
        }
    }

    for (int i = 0; i < szof(tmp); ++i) {
        if (i % 2 == 0) {
            rad[i] = (rad[i] + 1) / 2 * 2 - 1;
        } else {
            rad[i] = rad[i] / 2 * 2;
        }
    }

    return rad;
}
```

7.4 Suffix array + LCP

```
vector<int> build_suffarr(string s) {
    int n = szof(s);
    auto norm = [&](int num) {
        if (num >= n) {
            return num - n;
        }
        return num;
    };
    vector<int> classes(s.begin(), s.end()),
    ↪ n_classes(n);
    vector<int> order(n), n_order(n);
    iota(order.begin(), order.end(), 0);
    vector<int> cnt(max(szof(s), 128));
    for (int num : classes) {
        cnt[num + 1]++;
    }
    for (int i = 1; i < szof(cnt); ++i) {
        cnt[i] += cnt[i - 1];
    }

    for (int i = 0; i < n; i = i == 0 ? 1 : i * 2)
    ↪ {
        for (int pos : order) {
            int pp = norm(pos - i + n);
            n_order[cnt[classes[pp]]++] = pp;
        }
        int q = -1;
        pii prev = {-1, -1};
        for (int j = 0; j < n; ++j) {
            pii cur = {classes[n_order[j]],
            ↪ classes[norm(n_order[j] + i)]};
            if (cur != prev) {
                prev = cur;
                ++q;
                cnt[q] = j;
            }
            n_classes[n_order[j]] = q;
        }
        swap(n_classes, classes);
        swap(n_order, order);
    }
    return order;
}

void solve() {
    string s;
    cin >> s;
    s += "$";
    auto suffarr = build_suffarr(s);

    vector<int> where(szof(s));
    for (int i = 0; i < szof(s); ++i) {
        where[suffarr[i]] = i;
    }

    vector<int> lcp(szof(s));
    int cnt = 0;
    for (int i = 0; i < szof(s); ++i) {
        if (where[i] == szof(s) - 1) {
            cnt = 0;
            continue;
        }
        cnt = max(cnt - 1, 0);
        int next = suffarr[where[i] + 1];
        while (i + cnt < szof(s) && next + cnt <
        ↪ szof(s) && s[i + cnt] == s[next + cnt]) {
            ++cnt;
        }
        lcp[where[i]] = cnt;
    }
}
```

7.5 Suffix automaton

```
struct state {
    state() { std::fill(next, next + 26, -1); }

    int len = 0, link = -1;
    bool term = false;

    int next[26];
};

vector<state> st;
int last;

void sa_init() {
    last = 0;
    st.clear();
    st.resize(1);
}

void sa_extend(char c) {
    int cur = st.size();
    st.resize(st.size() + 1);

    st[cur].len = st[last].len + 1;
    int p;
    for (p = last; p != -1 && st[p].next[c - 'a']
    ↪ == -1; p = st[p].link)
        st[p].next[c - 'a'] = cur;
    if (p == -1)
        st[cur].link = 0;
    else {
        int q = st[p].next[c - 'a'];
        if (st[p].len + 1 == st[q].len)
            st[cur].link = q;
        else {
```

```
			int clone = st.size();
			st.resize(st.size() + 1);
			st[clone].len = st[p].len + 1;
			std::copy(st[q].next, st[q].next + 26,
			  ↪ st[clone].next);
			st[clone].link = st[q].link;
			for (; p != -1 && st[p].next[c - 'a'] == q;
			  ↪ p = st[p].link)
				st[p].next[c - 'a'] = clone;
			st[q].link = st[cur].link = clone;
		}
	}
	last = cur;
}

for (int v = last; v != -1; v = st[v].link) //
↪ set termination flag.
	st[v].term = 1;
```

7.6 Suffix tree

```
#include <bits/stdc++.h>

using namespace std;

#define forn(i, n) for (int i = 0; i < (int)(n);
↪ i++)

const int N = 1e5, VN = 2 * N;

char s[N + 1];
map<char, int> t[VN];
int l[VN], r[VN], p[VN]; // edge p[v] -> v
↪ matches to [l[v], r[v]) of string
int cc, n, suf[VN], vn = 2, v = 1, pos; // going
↪ by edge from p[v] to v, now standing in pos

void go(int v) {
	int no = cc++;
	for (auto p : t[v]) {
		v = p.second;
		printf("%d %d %d\n", no, l[v], min(n, r[v]));
		go(v);
	}
}

int main() {
	assert(freopen("suftree.in", "r", stdin));
	assert(freopen("suftree.out", "w", stdout));

	gets(s);
	forn(i, 127) t[0][i] = 1; // 0 = fictitious, 1
	  ↪ = root
	l[1] = -1;
	for (n = 0; s[n]; n++) {
		char c = s[n];
		auto new_leaf = [&](int v) {
			p[vn] = v, l[vn] = n, r[vn] = N, t[v][c] =
			  ↪ vn++;
		};
	go:;
		if (r[v] <= pos) {
			if (!t[v].count(c)) {
				new_leaf(v), v = suf[v], pos = r[v];
				goto go;
			}
			v = t[v][c], pos = l[v] + 1;
		} else if (c == s[pos]) {
			pos++;
		} else {
			int x = vn++;
			l[x] = l[v], r[x] = pos, l[v] = pos;
			p[x] = p[v], p[v] = x;
			t[p[x]][s[l[x]]] = x, t[x][s[pos]] = v;
			new_leaf(x);
			v = suf[p[x]], pos = l[x];
			while (pos < r[x])
				v = t[v][s[pos]], pos += r[v] - l[v];
			suf[x] = (pos == r[x] ? v : vn);
			pos = r[v] - (pos - r[x]);
			goto go;
		}
	}
	printf("%d\n", vn - 1);
	go(1);
}
```

Table of Integrals*

Basic Forms

$$\int x^n dx = \frac{1}{n+1}x^{n+1} \tag{1}$$

$$\int \frac{1}{x}dx = \ln|x| \tag{2}$$

$$\int udv = uv - \int vdu \tag{3}$$

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

Integrals of Rational Functions

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

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

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

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

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

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

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

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

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

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

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

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

Integrals with Roots

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

$$\int \frac{1}{\sqrt{x\pm a}}dx = 2\sqrt{x\pm a} \tag{18}$$

$$\int \frac{1}{\sqrt{a-x}}dx = -2\sqrt{a-x} \tag{19}$$

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

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

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

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

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

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

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

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

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

$$\int \sqrt{x^2\pm a^2}dx = \frac{1}{2}x\sqrt{x^2\pm a^2} \pm \frac{1}{2}a^2\ln\left|x + \sqrt{x^2\pm a^2}\right| \tag{29}$$

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

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

$$\int \frac{1}{\sqrt{x^2\pm a^2}}dx = \ln\left|x + \sqrt{x^2\pm a^2}\right| \tag{32}$$

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

$$\int \frac{x}{\sqrt{x^2\pm a^2}}dx = \sqrt{x^2\pm a^2} \tag{34}$$

$$\int \frac{x}{\sqrt{a^2-x^2}}dx = -\sqrt{a^2-x^2} \tag{35}$$

$$\int \frac{x^2}{\sqrt{x^2\pm a^2}}dx = \frac{1}{2}x\sqrt{x^2\pm a^2} \mp \frac{1}{2}a^2\ln\left|x + \sqrt{x^2\pm a^2}\right| \tag{36}$$

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

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

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

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

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

Integrals with Logarithms

$$\int \ln ax dx = x\ln ax - x \tag{42}$$

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

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

$$\int \ln(x^2+a^2)\,\mathrm{d}x = x\ln(x^2+a^2) + 2a\tan^{-1}\frac{x}{a} - 2x \tag{45}$$

$$\int \ln(x^2-a^2)\,\mathrm{d}x = x\ln(x^2-a^2) + a\ln\frac{x+a}{x-a} - 2x \tag{46}$$

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

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

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

Integrals with Exponentials

$$\int e^{ax}dx = \frac{1}{a}e^{ax} \tag{50}$$

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

$$\int xe^x dx = (x-1)e^x \tag{52}$$

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

$$\int x^2e^x dx = \left(x^2-2x+2\right)e^x \tag{54}$$

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

$$\int x^3e^x dx = \left(x^3-3x^2+6x-6\right)e^x \tag{56}$$

$$\int x^ne^{ax}\,\mathrm{d}x = \frac{x^ne^{ax}}{a} - \frac{n}{a}\int x^{n-1}e^{ax}\,\mathrm{d}x \tag{57}$$

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

$$\int e^{ax^2}\,\mathrm{d}x = -\frac{i\sqrt{\pi}}{2\sqrt{a}}\operatorname{erf}\left(ix\sqrt{a}\right) \tag{59}$$

$$\int e^{-ax^2}\,\mathrm{d}x = \frac{\sqrt{\pi}}{2\sqrt{a}}\operatorname{erf}\left(x\sqrt{a}\right) \tag{60}$$

$$\int xe^{-ax^2}\,\mathrm{d}x = -\frac{1}{2a}e^{-ax^2} \tag{61}$$

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

Integrals with Trigonometric Functions

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Products of Trigonometric Functions and Monomials

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

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

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

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

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

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

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

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

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

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

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

Products of Trigonometric Functions and Exponentials

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

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

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

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

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

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

Integrals of Hyperbolic Functions

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

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

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

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

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

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

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

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

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

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

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

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