C'est BON! Team Notes

Contents

1 Number theory

1.1 Count primes up to N

```
// To initialize, call init_count_primes() first.
// Function count_primes(N) will compute the number of prime numbers lower
     than
// or equal to N.
//
// Time complexity: Around O(N ^ 0.75)
//
// Constants to configure:
// - MAX is the maximum value of sqrt(N) + 2
bool prime[MAX];
int prec[MAX];
vector<int> P;
llint rec(llint N, int K) {
    if (N <= 1 || K < 0) return 0;
    if (N <= P[K]) return N-1;

    if (N < MAX && llint(P[K])*P[K] > N) return N-1 - prec[N] + prec[P[K]];
    const int LIM = 250;
    static int memo[LIM*LIM][LIM];
    bool ok = N < LIM*LIM;
    if (ok && memo[N][K]) return memo[N][K];
    llint ret = N/P[K] - rec(N/P[K], K-1) + rec(N, K-1);
    if (ok) memo[N][K] = ret;
    return ret;
}
llint count_primes(llint N) {
    if (N < MAX) return prec[N];
    int K = prec[(int)sqrt(N) + 1];
    return N-1 - rec(N, K) + prec[P[K]];
}
void init_count_primes() {
    prime[2] = true;
    for (int i = 3; i < MAX; i += 2) prime[i] = true;
    for (int i = 3; i*i < MAX; i += 2) if (prime[i])
        for (int j = i*i; j < MAX; j += i+i)
            prime[j] = false;
    REP(i, MAX) if (prime[i]) P.push_back(i);
    FOR(i, 1, MAX) prec[i] = prec[i-1] + prime[i];
}
```

1.2 Extended Euclide

```
int bezout(int a, int b) {
    // return x such that ax + by == gcd(a, b)
    int xa = 1, xb = 0;
    while (b) {
        int q = a / b;
        int r = a - q * b, xr = xa - q * xb;
        a = b; xa = xb;
        b = r; xb = xr;
    }
    return xa;
}

pair<int, int> solve(int a, int b, int c) {
    // solve ax + by == c
    int d = __gcd(a, b);
    int x = bezout(a, b);
    int y = (d - a * x) / b;
    c /= d;
    return make_pair(x * c, y * c);
}

int main() {
    int a = 100, b = 128;
    int c = __gcd(a, b);
    int x = bezout(a, b);
    int y = (c - a * x) / b;
    cout << x << ' ' << y << endl;
    pair<int, int> xy = solve(100, 128, 40);
    cout << xy.first << ' ' << xy.second << endl;
    return 0;
}
```

1.3 System of linear equations

```
// extended version, uses diophantine equation solver to solve system of
     congruent equations
pair<int, int> solve(int a, int b, int c) {
    int cc = c;
    // solve ax + by == c
    int d = __gcd(a, b);
    int x = bezout(a / d, b / d);
    int y = (d - a * x) / b;
    c /= d;
    return make_pair(x * c, y * c);
}

int lcm(int a, int b) {
    return a / __gcd(a, b) * b;
}

// use this if input is large, make sure (#define int long long)
int mul(int a, int b, int p) {
    a %= p, b %= p;
    int q = (int) ((long double) a * b / p);
    int r = a * b - q * p;
    while (r < 0) r += p;
    while (r >= p) r -= p;
    return r;
}

int solveSystem(vector<int> a, vector<int> b) {
    // xi mod bi = ai
    int A = a[0], B = b[0];
    // x mod B = A
    for (int i = 1; i < a.size(); ++i) {
        int curB = b[i], curA = a[i];
```

```
        // x = Bi + A = curB * j + curA
        pair<int, int> ij = solve(B, -curB, curA - A);
        if (B * ij.first + A != curB * ij.second + curA) return -1;
        int newB = lcm(B, curB);
        int newA = (mul(B, ij.first, newB) + A) % newB;
        if (newA < 0) newA += newB;
        A = newA; B = newB;
        if (i + 1 == a.size()) return A;
    }
    return -1;
}

int main() {
    vector<int> a = {0, 3, 3};
    vector<int> b = {3, 6, 9};
    cout << solveSystem(a, b) << endl;
    return 0;
}
```

1.4 Pollard Rho

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

using namespace std;

struct PollardRho {
    long long n;
    map<long long, int> ans;
    PollardRho(long long n) : n(n) {}
    long long random(long long u) {
        return abs(rand()) % u;
    }
    long long mul(long long a, long long b, long long p) {
        a %= p; b %= p;
        long long q = (long long)((long double) a * b / p);
        long long r = a * b - q * p;
        while (r < 0) r += p;
        while (r >= p) r -= p;
        return r;
    }

    long long pow(long long u, long long v, long long n) {
        long long res = 1;
        while (v) {
            if (v & 1) res = mul(res, u , n);
            u = mul(u, u, n);
            v >>= 1;
        }
        return res;
    }

    bool rabin(long long n) {
        if (n < 2) return 0;
        if (n == 2) return 1;
        long long s = 0, m = n - 1;
        while (m % 2 == 0) {
            s++;
            m >>= 1;
        }
        // 1 - 0.9 ^ 40
        for (int it = 1; it <= 40; it++) {
            long long u = random(n - 2) + 2;
            long long f = pow(u, m, n);
            if (f == 1 || f == n - 1) continue;
            for (int i = 1; i < s; i++) {
                f = mul(f, f, n);
                if (f == 1) return 0;
                if (f == n - 1) break;
            }
            if (f != n - 1) return 0;
        }
        return 1;
    }

    long long f(long long x, long long n) {
        return (mul(x, x, n) + 1) % n;
    }

    long long findfactor(long long n) {
        long long x = random(n - 1) + 2;
        long long y = x;
        long long p = 1;
        while (p == 1) {
            x = f(x, n);
            y = f(f(y, n), n);
            p = __gcd(abs(x - y), n);
        }
        return p;
    }

    void pollard_rho(long long n) {
        if (n <= 1000000) {
            for (int i = 2; i * i <= n; i++) {
                while (n % i == 0) {
                    ans[i]++;
                    n /= i;
                }
            }
            if (n > 1) ans[n]++;
            return;
        }
        if (rabin(n)) {
            ans[n]++;
            return;
        }
        long long p = 0;
        while (p == 0 || p == n) {
            p = findfactor(n);
        }
        pollard_rho(n / p);
        pollard_rho(p);
    }
};

int main() {
    long long n;
    cin >> n;
    PollardRho f(n);
    f.pollard_rho(f.n);
    for (auto x : f.ans) {
        cout << x.first << " " << x.second << endl;
    }
}
```

1.5 Cubic

```
const double EPS = 1e-6;
struct Result {
    int n; // Number of solutions
    double x[3]; // Solutions
};
Result solve_cubic(double a, double b, double c, double d) {
    long double a1 = b/a, a2 = c/a, a3 = d/a;
    long double q = (a1*a1 - 3*a2)/9.0, sq = -2*sqrt(q);
    long double r = (2*a1*a1*a1 - 9*a1*a2 + 27*a3)/54.0;
    double z = r*r-q*q*q, theta;
    Result s;
    if(z <= EPS) {
        s.n = 3; theta = acos(r/sqrt(q*q*q));
        s.x[0] = sq*cos(theta/3.0) - a1/3.0;
        s.x[1] = sq*cos((theta+2.0*PI)/3.0) - a1/3.0;
        s.x[2] = sq*cos((theta+4.0*PI)/3.0) - a1/3.0;
    }
    else {
        s.n = 1; s.x[0] = pow(sqrt(z)+fabs(r),1/3.0);
        s.x[0] += q/s.x[0]; s.x[0] *= (r < 0) ? 1 : -1;
        s.x[0] -= a1/3.0;
    }
    return s;
}
```

1.6 PythagoreTriple

```
// sinh bo 3 pytago nguyen thuy voi x, y, z <= n
vector< vector<int> > genPrimitivePytTriples(int n) {
    vector< vector<int> > ret;
    for (int r=1; r*r<=n; ++r) for (int s=(r%2==0)?1:2; s<r; s+=2) if (__gcd(
        r,s)==1) {
        vector<int> t;
        t.push_back(r*r+s*s); //z
        t.push_back(2*r*s); // y
        t.push_back(r*r-s*s); // x
        if (t[0]<=n) ret.push_back(t);
    }
    sort(ret.begin(), ret.end());
    return ret;
}
// a^2 + b^2 == c^2
// To generate all primitive triples:
// a = m^2 - n^2, b = 2mn, c = m^2 + n^2 (m > n)
// Primitive triples iff gcd(m, n) == 1 && (m - n) % 2 == 1
```

2 String

2.1 Suffix Array

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

using namespace std;

struct SuffixArray {
    static const int N = 100010;

    int n;
    char *s;
    int sa[N], tmp[N], pos[N];
    int len, cnt[N], lcp[N];

    SuffixArray(char *t) {
        s = t;
        n = strlen(s + 1);
        buildSA();
    }

    bool cmp(int u, int v) {
        if (pos[u] != pos[v]) {
            return pos[u] < pos[v];
        }
        return (u + len <= n && v + len <= n) ? pos[u + len] < pos[v + len] :
            u > v;
    }

    void radix(int delta) {
        memset(cnt, 0, sizeof cnt);
        for (int i = 1; i <= n; i++) {
            cnt[i + delta <= n ? pos[i + delta] : 0]++;
        }
        for (int i = 1; i < N; i++) {
            cnt[i] += cnt[i - 1];
        }
        for (int i = n; i > 0; i--) {
            int id = sa[i];
            tmp[cnt[id + delta <= n ? pos[id + delta] : 0]--] = id;
        }
```

```
        for (int i = 1; i <= n; i++) {
            sa[i] = tmp[i];
        }
    }

    void buildSA() {
        for (int i = 1; i <= n; i++) {
            sa[i] = i;
            pos[i] = s[i];
        }
        len = 1;
        while (1) {
            radix(len);
            radix(0);
            tmp[1] = 1;
            for (int i = 2; i <= n; i++) {
                tmp[i] = tmp[i - 1] + cmp(sa[i - 1], sa[i]);
            }
            for (int i = 1; i <= n; i++) {
                pos[sa[i]] = tmp[i];
            }
            if (tmp[n] == n) {
                break;
            }
            len <<= 1;
        }

        len = 0;
        for (int i = 1; i <= n; i++) {
            if (pos[i] == n) {
                continue;
            }
            int j = sa[pos[i] + 1];
            while (s[i + len] == s[j + len]) {
                len++;
            }
            lcp[pos[i]] = len;
            if (len) {
                len--;
            }
        }
    }
};
```

2.2 Aho Corasick

```
struct AhoCorasick {
    const int N = 30030;

    int fail[N];
    int to[N][26];
    int ending[N];
    int sz;

    AhoCorasick() {
        sz = 1;
    }

    int add(const string &s) {
        int node = 1;
        for (int i = 0; i < s.size(); ++i) {
            if (!to[node][s[i] - 'a']) {
                to[node][s[i] - 'a'] = ++sz;
            }
            node = to[node][s[i] - 'a'];
        }
        ending[node] = true;
        return node;
    }

    void push() {
        queue<int> Q;
        Q.push(1);
        fail[1] = 1;
        while (!Q.empty()) {
            int u = Q.front(); Q.pop();
            for (int i = 0; i < 26; ++i) {
                int &v = to[u][i];
                if (!v) {
                    v = u == 1 ? 1 : to[fail[u]][i];
                } else {
                    fail[v] = u == 1 ? 1 : to[fail[u]][i];
                    ending[v] |= ending[fail[v]];
                    Q.push(v);
                }
            }
        }
    }
};
```

2.3 Z algorithm

```
vector<int> calcZ(const string &s) {
    int L = 0, R = 0;
    int n = s.size();
    vector<int> Z(n);
    Z[0] = n;
    for (int i = 1; i < n; i++) {
        if (i > R)
        {
            L = R = i;
            while (R < n && s[R] == s[R - L]) R++;
            Z[i] = R - L; R--;
        }
        else
        {
            int k = i - L;
            if (Z[k] < R - i + 1) Z[i] = Z[k];
            else
            {
                L = i;
                while (R < n && s[R] == s[R - L]) R++;
                Z[i] = R - L; R--;
            }
        }
    }
    return Z;
}
```

2.4 Manacher

```
struct Manacher {
    int n;
    vector<int> d; //Radius of odd palindromes
    vector<int> e; //Radius of even palindromes
    int build(char* s) {
        n = strlen(s), d.resize(n), e.resize(n);
        int res = 0;
        int l = 0, r = -1;
        for (int i = 0; i < n; ++i) {
            int k = (i > r) ? 1 : min(d[l + r - i], r - i) + 1;
            while (i - k >= 0 && i + k < n && s[i - k] == s[i + k]) k++;
            d[i] = --k;
            res = max(res, k + k + 1);
            if (r < i + k) {
                l = i - k;
                r = i + k;
            }
        }
        l = 0; r = -1;
        for (int i = 0; i < n; ++i) {
            int k = (i > r) ? 1 : min(e[l + r - i + 1], r - i + 1) + 1;
            while (i - k >= 0 && i + k - 1 < n && s[i - k] == s[i + k - 1]) k
                ++;
            e[i] = --k;
            res = max(res, k + k);
            if (r < i + k - 1) {
                l = i - k;
                r = i + k - 1;
            }
        }
        return res;
    }
}
```

2.5 Suffix Automaton

```
//set last = 0 everytime we add new string
struct SuffixAutomaton {
    static const int N = 100000;
    static const int CHARACTER = 26;
    int suf[N * 2], nxt[N * 2][CHARACTER], cnt, last, len[N * 2];

    SuffixAutomaton() {
        memset(suf, -1, sizeof suf);
        memset(nxt, -1, sizeof nxt);
        memset(len, 0, sizeof len);
        last = cnt = 0;
    }

    int getNode(int last, int u) {
        int q = nxt[last][u];
        if (len[last] + 1 == len[q]) {
            return q;
        }
        int clone = ++cnt;
        len[clone] = len[last] + 1;
        for (int i = 0; i < CHARACTER; i++) {
            nxt[clone][i] = nxt[q][i];
        }
        while (last != -1 && nxt[last][u] == q) {
            nxt[last][u] = clone;
            last = suf[last];
        }
        suf[clone] = suf[q];
        return suf[q] = clone;
    }

    void add(int u) {
        if (nxt[last][u] == -1) {
            int newNode = ++cnt;
            len[newNode] = len[last] + 1;
            while (last != -1 && nxt[last][u] == -1) {
                nxt[last][u] = newNode;
                last = suf[last];
            }
            if (last == -1) {
                suf[newNode] = 0;
                last = newNode;
                return;
            }
            suf[newNode] = getNode(last, u);
            last = newNode;
        } else {
            last = getNode(last, u);
        }
    }
};
```

2.6 ALCS

```
define
    a: row, b: col
    c_l,i,j: largest weighted path from (0, i) to (l, j)
```

```
    f_l,j: for all (l, j), 0 <= l <= n_a, 1 <= j <= n_b, f_l,j is the
           smallest value i < j such that C_l,i,j = C_l,i,j-1 + 1
    g_l,j: for all (l, j), 1 <= l <= n_a, 0 <= j <= n_b, g_l,j is the
           smallest value i <= j such that C_l,i,j = C_l-1,i,j

const int N = 2010;
int n, m;
char a[N], b[N];
int f[N][N], g[N][N];
int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    int tc;
    cin >> tc;
    while (tc--) {
        cin >> (a + 1);
        cin >> (b + 1);
        n = strlen(a + 1);
        m = strlen(b + 1);
        for (int i = 1; i <= m; i++) f[0][i] = i;
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= m; j++) {
                if (a[i] == b[j]) {
                    f[i][j] = g[i][j - 1];
                    g[i][j] = f[i - 1][j];
                } else {
                    f[i][j] = max(f[i - 1][j], g[i][j - 1]);
                    g[i][j] = min(f[i - 1][j], g[i][j - 1]);
                }
            }
        }
        int q;
        cin >> q;
        for (int i = 1; i <= q; i++) {
            int l, r, k;
            // substring a(1, r) and b(1, k)
            cin >> l >> r >> k;
            int res = 0;
            for (int j = 1; j <= r; j++) res += (f[k][j] < l);
            cout << res << ' ';
        }
        cout << '\n';
    }
    return 0;
}
```

2.7 Palindromic Tree

```
const int N = 1e5, SIZE = 26;

int s[N], len[N], link[N], to[N][SIZE], depth[N];
int n, last, sz;

void init() {
    s[n++] = -1;
    link[0] = 1;
    len[1] = -1;
    sz = 2;
}

int get_link(int v) {
    while (s[n - len[v] - 2] != s[n - 1]) v = link[v];
    return v;
}

int add_letter(int c) {
    s[n++] = c;
    last = get_link(last);
    if (!to[last][c]) {
        len [sz] = len[last] + 2;
        link[sz] = to[get_link(link[last])][c];
        to[last][c] = sz++;
    }
    last = to[last][c];
    return len[last];
}
```

2.8 Lyndon Factorization

```
// https://cp-algorithms.com/string/lyndon_factorization.html
vector<string> duval(string const& s) {
    int n = s.size();
    int i = 0;
    vector<string> factorization;
    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) {
            factorization.push_back(s.substr(i, j - k));
            i += j - k;
        }
    }
    return factorization;
}
string min_cyclic_string(string s) {
    s += s;
    int n = s.size();
    int i = 0, ans = 0;
    while (i < n / 2) {
        ans = i;
        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)
            i += j - k;
    }
    return s.substr(ans, n / 2);
}
```

3 Combinatorial optimization

4 Geometry

4.1 Geometry

```
#define EPS 1e-6
inline int cmp(double a, double b) { return (a < b - EPS) ? -1 : ((a > b +
    EPS) ? 1 : 0); }
struct Point {
    double x, y;
    Point() { x = y = 0.0; }
    Point(double x, double y) : x(x), y(y) {}

    Point operator + (const Point& a) const { return Point(x+a.x, y+a.y); }
    Point operator - (const Point& a) const { return Point(x-a.x, y-a.y); }
    Point operator * (double k) const { return Point(x*k, y*k); }
    Point operator / (double k) const { return Point(x/k, y/k); }

    double operator * (const Point& a) const { return x*a.x + y*a.y; } // dot
         product
    double operator % (const Point& a) const { return x*a.y - y*a.x; } //
        cross product
    double norm() { return x*x + y*y; }
    double len() { return sqrt(norm()); } // hypot(x, y);
    Point rotate(double alpha) {
        double cosa = cos(alpha), sina = sin(alpha);
        return Point(x * cosa - y * sina, x * sina + y * cosa);
    }
};
double angle(Point a, Point o, Point b) { // min of directed angle AOB & BOA
    a = a - o; b = b - o;
    return acos((a * b) / sqrt(a.norm()) / sqrt(b.norm()));
}
double directed_angle(Point a, Point o, Point b) { // angle AOB, in range [0,
    2*PI)
    double t = -atan2(a.y - o.y, a.x - o.x)
        + atan2(b.y - o.y, b.x - o.x);
    while (t < 0) t += 2*PI;
    return t;
}
// Distance from p to Line ab (closest Point --> c)
double distToLine(Point p, Point a, Point b, Point &c) {
    Point ap = p - a, ab = b - a;
    double u = (ap * ab) / ab.norm();
    c = a + (ab * u);
    return (p-c).len();
}
// Distance from p to segment ab (closest Point --> c)
double distToLineSegment(Point p, Point a, Point b, Point &c) {
    Point ap = p - a, ab = b - a;
    double u = (ap * ab) / ab.norm();
    if (u < 0.0) {
        c = Point(a.x, a.y);
        return (p - a).len();
    }
    if (u > 1.0) {
        c = Point(b.x, b.y);
        return (p - b).len();
    }
    return distToLine(p, a, b, c);
}
// NOTE: WILL NOT WORK WHEN a = b = 0.
struct Line {
    double a, b, c;
    Point A, B; // Added for polygon intersect line. Do not rely on
        assumption that these are valid

    Line(double a, double b, double c) : a(a), b(b), c(c) {}

    Line(Point A, Point B) : A(A), B(B) {
        a = B.y - A.y;
        b = A.x - B.x;
        c = - (a * A.x + b * A.y);
    }
    Line(Point P, double m) {
        a = -m; b = 1;
        c = -((a * P.x) + (b * P.y));
    }
    double f(Point A) {
        return a*A.x + b*A.y + c;
    }
};
bool areParallel(Line l1, Line l2) {
    return cmp(l1.a*l2.b, l1.b*l2.a) == 0;
}
bool areSame(Line l1, Line l2) {
    return areParallel(l1 ,l2) && cmp(l1.c*l2.a, l2.c*l1.a) == 0
                && cmp(l1.c*l2.b, l1.b*l2.c) == 0;
}
bool areIntersect(Line l1, Line l2, Point &p) {
    if (areParallel(l1, l2)) return false;
    double dx = l1.b*l2.c - l2.b*l1.c;
    double dy = l1.c*l2.a - l2.c*l1.a;
    double d  = l1.a*l2.b - l2.a*l1.b;
    p = Point(dx/d, dy/d);
```

```
    return true;
}
void closestPoint(Line l, Point p, Point &ans) {
    if (fabs(l.b) < EPS) {
        ans.x = -(l.c) / l.a; ans.y = p.y;
        return;
    }
    if (fabs(l.a) < EPS) {
        ans.x = p.x; ans.y = -(l.c) / l.b;
        return;
    }
    Line perp(l.b, -l.a, - (l.b*p.x - l.a*p.y));
    areIntersect(l, perp, ans);
}
void reflectionPoint(Line l, Point p, Point &ans) {
    Point b;
    closestPoint(l, p, b);
    ans = p + (b - p) * 2;
}
struct Circle : Point {
    double r;
    Circle(double x = 0, double y = 0, double r = 0) : Point(x, y), r(r) {}
    Circle(Point p, double r) : Point(p), r(r) {}
    bool contains(Point p) { return (*this - p).len() <= r + EPS; }
};

// Find common tangents to 2 circles
// Tested:
// - http://codeforces.com/gym/100803/ - H
// Helper method
void tangents(Point c, double r1, double r2, vector<Line> & ans) {
    double r = r2 - r1;
    double z = sqr(c.x) + sqr(c.y);
    double d = z - sqr(r);
    if (d < -EPS)  return;
    d = sqrt(fabs(d));
    Line l((c.x * r + c.y * d) / z,
            (c.y * r - c.x * d) / z,
            r1);
    ans.push_back(l);
}
// Actual method: returns vector containing all common tangents
vector<Line> tangents(Circle a, Circle b) {
    vector<Line> ans; ans.clear();
    for (int i=-1; i<=1; i+=2)
        for (int j=-1; j<=1; j+=2)
            tangents(b-a, a.r*i, b.r*j, ans);
    for(int i = 0; i < ans.size(); ++i)
        ans[i].c -= ans[i].a * a.x + ans[i].b * a.y;
    vector<Line> ret;
    for(int i = 0; i < (int) ans.size(); ++i) {
        bool ok = true;
        for(int j = 0; j < i; ++j)
            if (areSame(ret[j], ans[i])) {
                ok = false;
                break;
            }
        if (ok) ret.push_back(ans[i]);
    }
    return ret;
}
// Circle & line intersection
vector<Point> intersection(Line l, Circle cir) {
    double r = cir.r, a = l.a, b = l.b, c = l.c + l.a*cir.x + l.b*cir.y;
    vector<Point> res;
    double x0 = -a*c/(a*a+b*b),  y0 = -b*c/(a*a+b*b);
    if (c*c > r*r*(a*a+b*b)+EPS) return res;
    else if (fabs(c*c - r*r*(a*a+b*b)) < EPS) {
        res.push_back(Point(x0, y0) + Point(cir.x, cir.y));
        return res;
    }
    else {
        double d = r*r - c*c/(a*a+b*b);
        double mult = sqrt (d / (a*a+b*b));
        double ax,ay,bx,by;
        ax = x0 + b * mult;
        bx = x0 - b * mult;
        ay = y0 - a * mult;
        by = y0 + a * mult;
        res.push_back(Point(ax, ay) + Point(cir.x, cir.y));
        res.push_back(Point(bx, by) + Point(cir.x, cir.y));
        return res;
    }
}
// helper functions for commonCircleArea
double cir_area_solve(double a, double b, double c) {
    return acos((a*a + b*b - c*c) / 2 / a / b);
}
double cir_area_cut(double a, double r) {
    double s1 = a * r * r / 2;
    double s2 = sin(a) * r * r / 2;
    return s1 - s2;
}
double commonCircleArea(Circle c1, Circle c2) { //return the common area of
     two circle
    if (c1.r < c2.r) swap(c1, c2);
    double d = (c1 - c2).len();
    if (d + c2.r <= c1.r + EPS) return c2.r*c2.r*M_PI;
    if (d >= c1.r + c2.r - EPS) return 0.0;
    double a1 = cir_area_solve(d, c1.r, c2.r);
    double a2 = cir_area_solve(d, c2.r, c1.r);
    return cir_area_cut(a1*2, c1.r) + cir_area_cut(a2*2, c2.r);
}
// Check if 2 circle intersects. Return true if 2 circles touch
bool areIntersect(Circle u, Circle v) {
    if (cmp((u - v).len(), u.r + v.r) > 0) return false;
    if (cmp((u - v).len() + v.r, u.r) < 0) return false;
    if (cmp((u - v).len() + u.r, v.r) < 0) return false;
    return true;
}
// If 2 circle touches, will return 2 (same) points
// If 2 circle are same --> be careful
vector<Point> circleIntersect(Circle u, Circle v) {
    vector<Point> res;
    if (!areIntersect(u, v)) return res;
    double d = (u - v).len();
    double alpha = acos((u.r * u.r + d*d - v.r * v.r) / 2.0 / u.r / d);
    Point p1 = (v - u).rotate(alpha);
    Point p2 = (v - u).rotate(-alpha);
    res.push_back(p1 / p1.len() * u.r + u);
    res.push_back(p2 / p2.len() * u.r + u);
    return res;
}
Point centroid(Polygon p) {
    Point c(0,0);
    double scale = 6.0 * signed_area(p);
    for (int i = 0; i < p.size(); i++){
        int j = (i+1) % p.size();
        c = c + (p[i]+p[j])*(p[i].x*p[j].y - p[j].x*p[i].y);
    }
    return c / scale;
}
// Cut a polygon with a line. Returns one half.
// To return the other half, reverse the direction of Line l (by negating l.a
     , l.b)
// The line must be formed using 2 points
Polygon polygon_cut(const Polygon& P, Line l) {
    Polygon Q;
    for(int i = 0; i < P.size(); ++i) {
        Point A = P[i], B = (i == P.size()-1) ? P[0] : P[i+1];
        if (ccw(l.A, l.B, A) != -1) Q.push_back(A);
        if (ccw(l.A, l.B, A)*ccw(l.A, l.B, B) < 0) {
            Point p; areIntersect(Line(A, B), l, p);
            Q.push_back(p);
        }
    }
    return Q;
}
// Find intersection of 2 convex polygons
// Helper method
bool intersect_1pt(Point a, Point b,
    Point c, Point d, Point &r) {
    double D =  (b - a) % (d - c);
    if (cmp(D, 0) == 0) return false;
    double t =  ((c - a) % (d - c)) / D;
    double s = -((a - c) % (b - a)) / D;
    r = a + (b - a) * t;
    return cmp(t, 0) >= 0 && cmp(t, 1) <= 0 && cmp(s, 0) >= 0 && cmp(s, 1) <=
         0;
}
Polygon convex_intersect(Polygon P, Polygon Q) {
    const int n = P.size(), m = Q.size();
    int a = 0, b = 0, aa = 0, ba = 0;
    enum { Pin, Qin, Unknown } in = Unknown;
    Polygon R;
    do {
        int a1 = (a+n-1) % n, b1 = (b+m-1) % m;
        double C = (P[a] - P[a1]) % (Q[b] - Q[b1]);
        double A = (P[a1] - Q[b]) % (P[a] - Q[b]);
        double B = (Q[b1] - P[a]) % (Q[b] - P[a]);
        Point r;
        if (intersect_1pt(P[a1], P[a], Q[b1], Q[b], r)) {
            if (in == Unknown) aa = ba = 0;
            R.push_back( r );
            in = B > 0 ? Pin : A > 0 ? Qin : in;
        }
        if (C == 0 && B == 0 && A == 0) {
            if (in == Pin) { b = (b + 1) % m; ++ba; }
            else           { a = (a + 1) % m; ++aa; }
        } else if (C >= 0) {
            if (A > 0) { if (in == Pin) R.push_back(P[a]); a = (a+1)%n; ++aa;
                 }
            else       { if (in == Qin) R.push_back(Q[b]); b = (b+1)%m; ++ba;
                 }
        } else {
            if (B > 0) { if (in == Qin) R.push_back(Q[b]); b = (b+1)%m; ++ba;
                 }
            else       { if (in == Pin) R.push_back(P[a]); a = (a+1)%n; ++aa;
                 }
        }
    } while ( (aa < n || ba < m) && aa < 2*n && ba < 2*m );
    if (in == Unknown) {
        if (in_convex(Q, P[0])) return P;
        if (in_convex(P, Q[0])) return Q;
    }
    return R;
}
// Find the diameter of polygon.
// Rotating callipers
double convex_diameter(Polygon pt) {
    const int n = pt.size();
    int is = 0, js = 0;
    for (int i = 1; i < n; ++i) {
        if (pt[i].y > pt[is].y) is = i;
        if (pt[i].y < pt[js].y) js = i;
    }
    double maxd = (pt[is]-pt[js]).norm();
    int i, maxi, j, maxj;
    i = maxi = is;
    j = maxj = js;
    do {
        int jj = j+1; if (jj == n) jj = 0;
        if ((pt[i] - pt[jj]).norm() > (pt[i] - pt[j]).norm()) j = (j+1) % n;
        else i = (i+1) % n;
        if ((pt[i]-pt[j]).norm() > maxd) {
            maxd = (pt[i]-pt[j]).norm();
            maxi = i; maxj = j;
        }
    } while (i != is || j != js);
    return maxd; /* farthest pair is (maxi, maxj). */
}
// Check if we can form triangle with edges x, y, z.
bool isSquare(long long x) { /* */ }
bool isIntegerCoordinates(int x, int y, int z) {
    long long s=(long long)(x+y+z)*(x+y-z)*(x+z-y)*(y+z-x);
    return (s%4==0 && isSquare(s/4));
}

// Pick theorem
// Given non-intersecting polygon.
// S = area
// I = number of integer points strictly Inside
// B = number of points on sides of polygon
// S = I + B/2 - 1
```

```cpp
// Smallest enclosing circle:
// Given N points. Find the smallest circle enclosing these points.
// Amortized complexity: O(N)
struct SmallestEnclosingCircle {
    Circle getCircle(vector<Point> points) {
        assert(!points.empty());
        random_shuffle(points.begin(), points.end());
        Circle c(points[0], 0);
        int n = points.size();
        for (int i = 1; i < n; i++)
            if ((points[i] - c).len() > c.r + EPS) {
                c = Circle(points[i], 0);
                for (int j = 0; j < i; j++)
                    if ((points[j] - c).len() > c.r + EPS) {
                        c = Circle((points[i] + points[j]) / 2, (points[i] -
                            points[j]).len() / 2);
                        for (int k = 0; k < j; k++)
                            if ((points[k] - c).len() > c.r + EPS)
                                c = getCircumcircle(points[i], points[j],
                                    points[k]);
                    }
            }
        return c;
    }
    // NOTE: This code work only when a, b, c are not collinear and no 2
        points are same --> DO NOT
    // copy and use in other cases.
    Circle getCircumcircle(Point a, Point b, Point c) {
        assert(a != b && b != c && a != c);
        assert(ccw(a, b, c));
        double d = 2.0 * (a.x * (b.y - c.y) + b.x * (c.y - a.y) + c.x * (a.y
            - b.y));
        assert(fabs(d) > EPS);
        double x = (a.norm() * (b.y - c.y) + b.norm() * (c.y - a.y) + c.norm
            () * (a.y - b.y)) / d;
        double y = (a.norm() * (c.x - b.x) + b.norm() * (a.x - c.x) + c.norm
            () * (b.x - a.x)) / d;
        Point p(x, y);
        return Circle(p, (p - a).len());
    }
};
bool inside(const Point &u, const vector<Point> &a) {
    for (int i = 0; i < n; i++) {
        if (cmp((a[i] - u) % (a[i == n - 1 ? 0 : i + 1] - u), 0.0) != 0)
            continue;
        if (cmp((a[i] - u) * (a[i == n - 1 ? 0 : i + 1] - u), 0.0) > 0)
            continue;
        return 1;
    }
    int res = 0;
    for (int i = 0; i < n; i++) {
        Point v = a[i], w = a[i == n - 1 ? 0 : i + 1];
        if (cmp(v.x, w.x) == 0) continue;
        if (v.x > w.x) swap(v, w);
        if (u.x < v.x - EPS) continue;
        if (u.x > w.x - EPS) continue;
        res ^= (cmp((u - v) % (w - v), 0) >= 0);
    }
    return res;
}
```

5 Numerical algorithms

5.1 Gauss Elimination

```cpp
const int INF = 1e9;
const double EPS = 1e-9;
int gauss(vector<vector<double> > a, vector<double> &ans) {
    int m = a.size(), n = a[0].size() - 1;
    vector<int> where (n, -1); // corresponding row for each column
    for (int row = 0, col = 0; col < n; ++col) {
        // find the maximum abs value on the current column to reduce
            precision errors
        int maxRow = row;
        for (int i = row + 1; i < m; ++i) {
            if (abs(a[i][col]) > abs(a[maxRow][col]))
            maxRow = i;
        }
        // if cannot find anything rather than zero then forget the current
            column
        if (abs(a[maxRow][col]) < EPS) continue;
        if (maxRow != row) swap(a[maxRow], a[row]);
        where[col] = row;
        for (int i = 0; i < m; ++i) if (i != row) {
            double coef = a[i][col] / a[row][col];
            for (int j = col; j <= n; ++j) {
                a[i][j] -= a[row][j] * coef;
            }
        }
        ++row; // only when found a non-zero element
    }
    ans.assign(m, 0); // default value = 0
    for (int i = 0; i < n; ++i) if (where[i] != -1) {
        ans[i] = a[where[i]][n] / a[where[i]][i];
    }
    // recheck
    for (int i = 0; i < m; ++i) {
        double sum = 0;
        for (int j = 0; j < n; ++j) {
            sum += a[i][j] * ans[j];
        }
        if (abs(sum - a[i][n]) > EPS) return 0; // no solution
    }
    // search for independent variables
    for (int i = 0; i < n; ++i) if (where[i] == -1) return INF; // infinite
        many solution
    return 1; // one solution saved in vector ans
}
```

5.2 Simplex Algorithm

```cpp
/**
 * minimize c^T * x
 * subject to Ax <= b
 * and x >= 0
 * The input matrix a will have the following form
 * 0 c c c c c
 * b A A A A A
 * b A A A A A
 * b A A A A A
 * Result vector will be: val x x x x x
 **/

typedef long double ld;
const ld EPS = 1e-8;
struct LPSolver {
    static vector<ld> simplex(vector<vector<ld>> a) {
        int n = (int) a.size() - 1;
        int m = (int) a[0].size() - 1;
        vector<int> left(n + 1);
        vector<int> up(m + 1);
        iota(left.begin(), left.end(), m);
        iota(up.begin(), up.end(), 0);
        auto pivot = [&](int x, int y) {
            swap(left[x], up[y]);
            ld k = a[x][y];
            a[x][y] = 1;
            vector<int> pos;
            for (int j = 0; j <= m; j++) {
                a[x][j] /= k;
                if (fabs(a[x][j]) > EPS) pos.push_back(j);
            }
            for (int i = 0; i <= n; i++) {
                if (fabs(a[i][y]) < EPS || i == x) continue;
                k = a[i][y];
                a[i][y] = 0;
                for (int j : pos) a[i][j] -= k * a[x][j];
            }
        };
        while (1) {
            int x = -1;
            for (int i = 1; i <= n; i++) {
                if (a[i][0] < -EPS && (x == -1 || a[i][0] < a[x][0])) {
                    x = i;
                }
            }
            if (x == -1) break;
            int y = -1;
            for (int j = 1; j <= m; j++) {
                if (a[x][j] < -EPS && (y == -1 || a[x][j] < a[x][y])) {
                    y = j;
                }
            }
            if (y == -1) return vector<ld>(); // infeasible
            pivot(x, y);
        }
        while (1) {
            int y = -1;
            for (int j = 1; j <= m; j++) {
                if (a[0][j] > EPS && (y == -1 || a[0][j] > a[0][y])) {
                    y = j;
                }
            }
            if (y == -1) break;
            int x = -1;
            for (int i = 1; i <= n; i++) {
                if (a[i][y] > EPS && (x == -1 || a[i][0] / a[i][y] < a[x][0]
                    / a[x][y])) {
                    x = i;
                }
            }
            if (x == -1) return vector<ld>(); // unbounded
            pivot(x, y);
        }
        vector<ld> ans(m + 1);
        for (int i = 1; i <= n; i++) {
            if (left[i] <= m) ans[left[i]] = a[i][0];
        }
        ans[0] = -a[0][0];
        return ans;
    }
};
```

5.3 NTT

```cpp
//Poly Invert: R(2n) = 2R(n) - R(n) ^ 2 * F where R(z) = invert F(z)
//Poly Sqrt: 2 * S(2n) = S(n) + F * S(n) ^ -1
const int MOD = 998244353;
struct NTT {
    int base = 1;
    int maxBase = 0;
    int root = 2;
    vector<int> w = {0, 1};
    vector<int> rev = {0, 1};
    NTT () {
        int u = MOD - 1;
        while (u % 2 == 0) {
            u >>= 1;
            maxBase++;
        }
        while (power(root, 1 << maxBase) != 1 || power(root, 1 << (maxBase -
            1)) == 1) root++;
    }
    void ensure(int curBase) {
        assert(curBase <= maxBase);
        if (curBase <= base) return;
        rev.resize(1 << curBase);
        for (int i = 0; i < (1 << curBase); i++) {
            rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (curBase - 1));
```

```
        }
        w.resize(1 << curBase);
        for (; base < curBase; base++) {
            int wc = power(root, 1 << (maxBase - base - 1));
            for (int i = 1 << (base - 1); i < (1 << base); i++) {
                w[i << 1] = w[i];
                w[i << 1 | 1] = mul(w[i], wc);
            }
        }
    }
    void fft(vector<int> &a) {
        int n = a.size();
        int curBase = 0;
        while ((1 << curBase) < n) curBase++;
        int shift = base - curBase;
        for (int i = 0; i < n; i++) {
            if (i < (rev[i] >> shift)) swap(a[i], a[rev[i] >> shift]);
        }
        for (int k = 1; k < n; k <<= 1) {
            for (int i = 0; i < k; i++) {
                for (int j = i; j < n; j += k * 2) {
                    int foo = a[j];
                    int bar = mul(a[j + k], w[i + k]);
                    a[j] = add(foo, bar);
                    a[j + k] = sub(foo, bar);
                }
            }
        }
    }
    vector<int> mult(vector<int> a, vector<int> b) {
        int nResult = a.size() + b.size() - 1;
        int curBase = 0;
        while ((1 << curBase) < nResult) curBase++;
        ensure(curBase);
        int n = 1 << curBase;
        a.resize(n), b.resize(n);
        fft(a);
        fft(b);
        int invN = inv(n);
        for (int i = 0; i < n; i++) {
            a[i] = mul(mul(a[i], b[i]), invN);
        }
        reverse(a.begin() + 1, a.end());
        fft(a);
        a.resize(nResult);
        return a;
    }
    vector<int> polyInv(vector<int> r, vector<int> f) {
        vector<int> foo = mult(r, f);
        foo.resize(f.size());
        foo[0] = sub(2, foo[0]);
        for (int i = 1; i < foo.size(); i++) {
            foo[i] = sub(0, foo[i]);
        }
        vector<int> res = mult(r, foo);
        res.resize(f.size());
        return res;
    }
    vector<int> polySqrt(vector<int> s, vector<int> invS, vector<int> f) {
        vector<int> res = mult(f, invS);
        res.resize(f.size());
        for (int i = 0; i < s.size(); i++) {
            res[i] = add(res[i], s[i]);
        }
        for (int i = 0; i < res.size(); i++) {
            res[i] = mul(res[i], INV_2);
        }
        return res;
    }
    vector<int> getSqrt(vector<int> c, int sz) {
        vector<int> sqrtC = {1}, invSqrtC = {1}; //change this if c[0] != 1
        for (int k = 1; k < (1 << sz); k <<= 1) {
            vector<int> foo(c.begin(), c.begin() + (k * 2));
            vector<int> bar = sqrtC;
            bar.resize(bar.size() * 2, 0);
            vector<int> tempInv = polyInv(invSqrtC, bar);
            sqrtC = polySqrt(sqrtC, tempInv, foo);
            invSqrtC = polyInv(invSqrtC, sqrtC);
        }
        return sqrtC;
    }
    vector<int> getInv(vector<int> c, int sz) {
        vector<int> res = {INV_2}; // change this if c[0] != 2
        for (int k = 1; k < (1 << sz); k <<= 1) {
            vector<int> foo(c.begin(), c.begin() + (k * 2));
            res = polyInv(res, foo);
        }
        return res;
    }
} ntt;
```

5.4 FFT

```
typedef complex<double> cmplx;
typedef vector<complex<double> > VC;
const double PI = acos(-1);
struct FFT {
    static void fft(VC &u, int sign) {
        int n = u.size();
        double theta = 2. * PI * sign / n;
        for (int m = n; m >= 2; m >>= 1, theta *= 2.) {
            cmplx w(1, 0), wDelta = polar(1., theta);
            for (int i = 0, mh = m >> 1; i < mh; i++) {
                for (int j = i; j < n; j += m) {
                    int k = j + mh;
                    cmplx temp = u[j] - u[k];
                    u[j] += u[k];
                    u[k] = w * temp;
                }
                w *= wDelta;
            }
        }
        for (int i = 1, j = 0; i < n; i++) {
            for (int k = n >> 1; k > (j ^= k); k >>= 1);
            if (j < i) {
                swap(u[i], u[j]);
            }
        }
    }

    static vector<int> mul(const vector<int> &a, const vector<int> &b) {
        int newSz = a.size() + b.size() - 1;
        int fftSz = 1;
        while (fftSz < newSz) {
            fftSz <<= 1;
        }
        VC aa(fftSz, 0.), bb(fftSz, 0.);
        for (int i = 0; i < a.size(); i++) {
            aa[i] = a[i];
        }
        for (int i = 0; i < b.size(); i++) {
            bb[i] = b[i];
        }
        fft(aa, 1);
        fft(bb, 1);
        for (int i = 0; i < fftSz; i++) {
            aa[i] *= bb[i];
        }
        fft(aa, -1);
        vector<int> res(newSz);
        for (int i = 0; i < newSz; i++) {
            res[i] = (int)(aa[i].real() / fftSz + 0.5);
        }
        return res;
    }
};
```

5.5 Bitwise FFT

```
/*
* matrix:
* +1 +1
* +1 -1
*/
void XORFFT(int a[], int n, int p, int invert) {
    for (int i = 1; i < n; i <<= 1) {
        for (int j = 0; j < n; j += i << 1) {
            for (int k = 0; k < i; k++) {
                int u = a[j + k], v = a[i + j + k];
                a[j + k] = u + v;
                if (a[j + k] >= p) a[j + k] -= p;
                a[i + j + k] = u - v;
                if (a[i + j + k] < 0) a[i + j + k] += p;
            }
        }
    }
    if (invert) {
        long long inv = fpow(n, p - 2, p);
        for (int i = 0; i < n; i++) a[i] = a[i] * inv % p;
    }
}
/*
* Matrix:
* +1 +1
* +1 +0
*/
void ORFFT(int a[], int n, int p, int invert) {
    for (int i = 1; i < n; i <<= 1) {
        for (int j = 0; j < n; j += i << 1) {
            for (int k = 0; k < i; k++) {
                int u = a[j + k], v = a[i + j + k];
                if (!invert) {
                    a[j + k] = u + v;
                    a[i + j + k] = u;
                    if (a[j + k] >= p) a[j + k] -= p;
                }
                else {
                    a[j + k] = v;
                    a[i + j + k] = u - v;
                    if (a[i + j + k] < 0) a[i + j + k] += p;
                }
            }
        }
    }
}
/*
* matrix:
* +0 +1
* +1 +1
*/
void ANDFFT(int a[], int n, int p, int invert) {
    for (int i = 1; i < n; i <<= 1) {
        for (int j = 0; j < n; j += i << 1) {
            for (int k = 0; k < i; k++) {
                int u = a[j + k], v = a[i + j + k];
                if (!invert) {
                    a[j + k] = v;
                    a[i + j + k] = u + v;
                    if (a[i + j + k] >= p) a[i + j + k] -= p;
                }
                else {
                    a[j + k] = v - u;
                    if (a[j + k] < 0) a[j + k] += p;
                    a[i + j + k] = u;
                }
            }
        }
    }
}
```

5.6 FFT chemthan

```
#define double long double
namespace FFT {
    const int maxf = 1 << 17;
    struct cp {
        double x, y;
        cp(double x = 0, double y = 0) : x(x), y(y) {}
        cp operator + (const cp& rhs) const {
            return cp(x + rhs.x, y + rhs.y);
        }
        cp operator - (const cp& rhs) const {
            return cp(x - rhs.x, y - rhs.y);
        }
        cp operator * (const cp& rhs) const {
            return cp(x * rhs.x - y * rhs.y, x * rhs.y + y * rhs.x);
        }
        cp operator !() const {
            return cp(x, -y);
        }
    } rts[maxf + 1];
    cp fa[maxf], fb[maxf];
    cp fc[maxf], fd[maxf];

    int bitrev[maxf];
    void fftinit() {
        int k = 0; while ((1 << k) < maxf) k++;
        bitrev[0] = 0;
        for (int i = 1; i < maxf; i++) {
            bitrev[i] = bitrev[i >> 1] >> 1 | ((i & 1) << k - 1);
        }
        double PI = acos((double) -1.0);
        rts[0] = rts[maxf] = cp(1, 0);
        for (int i = 1; i + i <= maxf; i++) {
            rts[i] = cp(cos(i * 2 * PI / maxf), sin(i * 2 * PI / maxf));
        }
        for (int i = maxf / 2 + 1; i < maxf; i++) {
            rts[i] = !rts[maxf - i];
        }
    }
    void dft(cp a[], int n, int sign) {
        static int isinit;
        if (!isinit) {
            isinit = 1;
            fftinit();
        }
        int d = 0; while ((1 << d) * n != maxf) d++;
        for (int i = 0; i < n; i++) {
            if (i < (bitrev[i] >> d)) {
                swap(a[i], a[bitrev[i] >> d]);
            }
        }
        for (int len = 2; len <= n; len <<= 1) {
            int delta = maxf / len * sign;
            for (int i = 0; i < n; i += len) {
                cp *x = a + i,*y = a + i + (len >> 1), *w = sign > 0 ? rts :
                    rts + maxf;
                for (int k = 0; k + k < len; k++) {
                    cp z = *y * *w;
                    *y = *x - z, *x = *x + z;
                    x++, y++, w += delta;
                }
            }
        }
        if (sign < 0) {
            for (int i = 0; i < n; i++) {
                a[i].x /= n;
                a[i].y /= n;
            }
        }
    }
    void multiply(int a[], int b[], int na, int nb, long long c[]) {
        int n = na + nb - 1; while (n != (n & -n)) n += n & -n;
        for (int i = 0; i < n; i++) fa[i] = fb[i] = cp();
        for (int i = 0; i < na; i++) fa[i] = cp(a[i]);
        for (int i = 0; i < nb; i++) fb[i] = cp(b[i]);
        dft(fa, n, 1), dft(fb, n, 1);
        for (int i = 0; i < n; i++) fa[i] = fa[i] * fb[i];
        dft(fa, n, -1);
        for (int i = 0; i < n; i++) c[i] = (long long) floor(fa[i].x + 0.5);
    }
    void multiply(int a[], int b[], int na, int nb, int c[], int mod = (int)
        1e9 + 7) {
        int n = na + nb - 1;
        while (n != (n & -n)) n += n & -n;
        for (int i = 0; i < n; i++) fa[i] = fb[i] = cp();
        static const int magic = 15;
        for (int i = 0; i < na; i++) fa[i] = cp(a[i] >> magic, a[i] & (1 <<
            magic) - 1);
        for (int i = 0; i < nb; i++) fb[i] = cp(b[i] >> magic, b[i] & (1 <<
            magic) - 1);
        dft(fa, n, 1), dft(fb, n, 1);
        for (int i = 0; i < n; i++) {
            int j = (n - i) % n;
            cp x = fa[i] + !fa[j];
            cp y = fb[i] + !fb[j];
            cp z = !fa[j] - fa[i];
            cp t = !fb[j] - fb[i];
            fc[i] = (x * t + y * z) * cp(0, 0.25);
            fd[i] = x * y * cp(0, 0.25) + z * t * cp(-0.25, 0);
        }
        dft(fc, n, -1), dft(fd, n, -1);
        for (int i = 0; i < n; i++) {
            long long u = ((long long) floor(fc[i].x + 0.5)) % mod;
            long long v = ((long long) floor(fd[i].x + 0.5)) % mod;
            long long w = ((long long) floor(fd[i].y + 0.5)) % mod;
            c[i] = ((u << 15) + v + (w << 30)) % mod;
        }
    }
    vector<int> multiply(vector<int> a, vector<int> b, int mod = (int) 1e9 +
        7) {
        static int fa[maxf], fb[maxf], fc[maxf];
        int na = a.size(), nb = b.size();
        for (int i = 0; i < na; i++) fa[i] = a[i];
        for (int i = 0; i < nb; i++) fb[i] = b[i];
        multiply(fa, fb, na, nb, fc, mod);
        int k = na + nb - 1;
        vector<int> res(k);
        for (int i = 0; i < k; i++) res[i] = fc[i];
        return res;
    }
}
```

5.7 Interpolation

```
#include <bits/stdc++.h>
using namespace std;

/*
 * Complexity: O(Nlog(mod), N)
 */
#define IP Interpolation
namespace Interpolation {
    const int mod = (int) 1e9 + 7;
    const int maxn = 1e5 + 5;
    int a[maxn];
    int fac[maxn];
    int ifac[maxn];
    int prf[maxn];
    int suf[maxn];

    int fpow(int n, int k) {
        int r = 1;
        for (; k; k >>= 1) {
            if (k & 1) r = (long long) r * n % mod;
            n = (long long) n * n % mod;
        }
        return r;
    }
    void upd(int u, int v) {
        a[u] = v;
    }
    void build() {
        fac[0] = ifac[0] = 1;
        for (int i = 1; i < maxn; i++) {
            fac[i] = (long long) fac[i - 1] * i % mod;
            ifac[i] = fpow(fac[i], mod - 2);
        }
    }
    //Calculate P(x) of degree k - 1, k values form 1 to k
    //P(i) = a[i]
    int calc(int x, int k) {
        prf[0] = suf[k + 1] = 1;
        for (int i = 1; i <= k; i++) {
            prf[i] = (long long) prf[i - 1] * (x - i + mod) % mod;
        }
        for (int i = k; i >= 1; i--) {
            suf[i] = (long long) suf[i + 1] * (x - i + mod) % mod;
        }
        int res = 0;
        for (int i = 1; i <= k; i++) {
            if (!((k - i) & 1)) {
                res = (res + (long long) prf[i - 1] * suf[i + 1] % mod
                    * ifac[i - 1] % mod * ifac[k - i] % mod * a[i]) % mod
                    ;
            }
            else {
                res = (res - (long long) prf[i - 1] * suf[i + 1] % mod
                    * ifac[i - 1] % mod * ifac[k - i] % mod * a[i] % mod
                    + mod) % mod;
            }
        }
        return res;
    }
}

const int mod = (int) 1e9 + 7;

int main() {
    IP::build();
    for (int i = 1; i < IP::maxn; i++) {
        IP::a[i] = ((long long) 3111 * i * i * i - (long long) 54 * i * i +
            13 * i) % mod;
    }
    assert(IP::calc(1234, 4) == IP::a[1234]);
    cerr << "\nTime elapsed: " << 1000 * clock() / CLOCKS_PER_SEC << "ms\n";
    return 0;
}
```

5.8 Binary vector space

```
int basis[d]; // basis[i] keeps the mask of the vector whose f value is i

int sz; // Current size of the basis

void insertVector(int mask) {
for (int i = 0; i < d; i++) {
    if ((mask & 1 << i) == 0) continue; // continue if i != f(mask)

    if (!basis[i]) { // If there is no basis vector with the i'th bit set,
        then insert this vector into the basis
        basis[i] = mask;
        ++sz;

        return;
    }

    mask ^= basis[i]; // Otherwise subtract the basis vector from this vector
}
```

5.9 DiophanteMod

```
// 1 <= a * k <= r (k > 0) (l <= r) (mod)
long long solve(long long l, long long r, long long a, long long mod) {
    if (a == 0) return INF;
    if (a * 2 > mod) return solve(mod - r, mod - l, mod - a, mod);
    long long firstVal = getCeil(l, a);
    if (a * firstVal <= r) return firstVal;
    if (mod % a == 0) return INF;
    long long kPrime = solve(l % a, r % a, a - mod % a, a);
    if (kPrime == INF) return INF;
    long long res = getCeil(kPrime * mod + l, a);
    return getCeil(kPrime * mod + l, a);
}
```

5.10 Berlekamp-Massey

```
#include <bits/stdc++.h>
using namespace std;
// linear recurrence: a[i] = sum(a[i - j] * p[j]) (sum from 1->m)
// calculate a[k] in O(m^2log(k)) (better than matrix multiplication O(m^3log
    (k)))
// Usage:
// calculate a[0], ..., a[m + m] (2 * m elements is enough) and call calc(
    vector<int>, int)
template<const int maxn, const int mod>
struct linear_solver {
    static const long long sqmod = (long long) mod * mod;
    int n;
    int a[maxn], h[maxn], s[maxn], t[maxn];
    long long t_[maxn];

    inline int fpow(int a, long long b) {
        int res = 1;
        while (b) {
            if (b & 1) res = (long long) res * a % mod;
            a = (long long) a * a % mod;
            b >>= 1;
        }
        return res;
    }
    inline vector<int> BM(vector<int> x) {
        vector<int> ls, cur;
        int lf, ld;
        for (int i = 0; i < (int) x.size(); i++) {
            long long t = 0;
            for (int j = 0; j < (int) cur.size(); j++) {
                t += (long long) x[i - j - 1] * cur[j];
                t -= sqmod <= t ? sqmod : 0;
            }
            t %= mod;
            if ((t - x[i]) % mod == 0) continue;
            if (!cur.size()) {
                cur.resize(i + 1);
                lf = i;
                ld = t - x[i];
                ld += ld < 0 ? mod : 0;
                continue;
            }
            int k = (long long) (t - x[i] + mod) * fpow(ld, mod - 2) % mod;
            vector<int> c(i - lf - 1);
            c.push_back(k);
            for (int j = 0; j < (int) ls.size(); j++) {
                c.push_back((long long) k * (mod - ls[j]) % mod);
            }
            if (c.size() < cur.size()) c.resize(cur.size());
            for (int j = 0; j < (int) cur.size(); j++) {
                c[j] += cur[j];
                c[j] -= mod <= c[j] ? mod : 0;
            }
            if (i - lf + (int) ls.size() >= (int) cur.size()) {
                ls = cur, lf = i;
                ld = t - x[i];
                ld += ld < 0 ? mod : 0;
            }
            cur = c;
        }
        for (int i = 0; i < (int) cur.size(); i++) {
            cur[i] = (cur[i] % mod + mod) % mod;
        }
        return cur;
    }
    inline void mult(int* p, int* q) {
        for (int i = 0; i < n + n; i++) t_[i] = 0;
        for (int i = 0; i < n; i++) if (p[i]) {
            for (int j = 0; j < n; j++) {
                t_[i + j] += (long long) p[i] * q[j];
                t_[i + j] -= sqmod <= t_[i + j] ? sqmod : 0;
            }
        }
        for (int i = n + n - 1; n <= i; i--) if (t_[i]) {
            t_[i] %= mod;
            for (int j = n - 1; ~j; j--) {
                t_[i - j - 1] += t_[i] * h[j];
                t_[i - j - 1] -= sqmod <= t_[i - j - 1] ? sqmod : 0;
            }
        }
        for (int i = 0; i < n; i++) p[i] = t_[i] % mod;
    }
    inline long long calc(long long k) {
        for (int i = n; ~i; i--) {
            s[i] = t[i] = 0;
        }
        s[0] = 1;
        if (n != 1) {
            t[1] = 1;
        }
        else {
            t[0] = h[0];
        }
        while (k) {
            if (k & 1) mult(s, t);
            mult(t, t); k >>= 1;
        }
        long long sum = 0;
        for (int i = 0; i < n; i++) {
            sum = (sum + (long long) s[i] * a[i]) % mod;
        }
        return (sum % mod + mod) % mod;
    }
    inline int calc(vector<int> x, long long k) {
        if (k < (int) x.size()) return x[k];
        vector<int> v = BM(x);
        n = v.size();
        if (!n) return 0;
        for (int i = 0; i < n; i++) h[i] = v[i], a[i] = x[i];
        return calc(k);
    }
};
linear_solver<1 << 18, (int) 1e9 + 7> ls;

const int mod = (int) 1e9 + 7;
const int maxn = 1 << 20;
int a[maxn];
int sa[maxn];

int main() {
    a[1] = 1;
    for (int i = 2; i < maxn; i++) {
        a[i] = ((long long) a[i - 1] * 3 + (long long) a[i - 2] * 5) % mod;
    }
    vector<int> sample;
    for (int i = 0; i < maxn; i++) {
        if (i) sa[i] = sa[i - 1];
        sa[i] = (sa[i] + a[i]) % mod;
        if (i < 2 * 1 + 4) {
            sample.push_back(sa[i]);
        }
    }
    cerr << ls.calc(sample, 50000) << " " << sa[50000] << "\n";
    cerr << "\nTime elapsed: " << 1000 * clock() / CLOCKS_PER_SEC << "ms\n";
    return 0;
}
```

5.11 Linear Sieve

```
vector <int> prime;
bool is_composite[MAXN];
void sieve (int n) {
        fill (is_composite, is_composite + n, false);
        for (int i = 2; i < n; ++i) {
                if (!is_composite[i]) prime.push_back (i);
                for (int j = 0; j < prime.size () && i * prime[j] < n; ++j) {
                        is_composite[i * prime[j]] = true;
                        if (i % prime[j] == 0) break;
                }
        }
}
```

6 Graph algorithms

6.1 Bridges and Articulations

```
void dfs(int u, int p) {
    num[u] = low[u] = ++cnt;
    st.push_back(u);
    for (auto v : adj[u]) {
        if (!num[v]) {
            if (u == 1) nChild++; // root = 1
            dfs(v, u);
            if (low[v] >= num[u]) { // u is articulation point, EXCEPT for 1
                isArticulation[u] = 1;
                numComp++;
                while (!st.empty()) {
                    int x = st.back();
                    st.pop_back();
                    ls[numComp].push_back(x);
                    if (x == v) break;
                }
                ls[numComp].push_back(u);
            }
            if (low[v] > num[u]) { // u - v bridge
                nBridge++;
            }
            low[u] = min(low[u], low[v]);
        } else if (v != p) {
            low[u] = min(low[u], num[v]);
        }
    }
}
if (nChild <= 1) isArticulation[1] = 0;
```

6.2 Bipartite Maximum Matching

```
struct BipartiteGraph {
    vector< vector<int> > a;
    vector<int> match;
    vector<bool> was;
    int m, n;

    BipartiteGraph(int m, int n) {
        // zero-indexed
        this->m = m; this->n = n;
        a.resize(m);
        match.assign(n, -1);
        was.assign(n, false);
```

```
    }

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

    bool dfs(int u) {
        for (int v : a[u]) if (!was[v]) {
            was[v] = true;
            if (match[v] == -1 || dfs(match[v])) {
                match[v] = u;
                return true;
            }
        }
        return false;
    }

    int maximumMatching() {
        vector<int> buffer;
        for (int i = 0; i < m; ++i) buffer.push_back(i);
        bool stop = false;
        int ans = 0;
        do {
            stop = true;
            for (int i = 0; i < n; ++i) was[i] = false;
            for (int i = (int)buffer.size() - 1; i >= 0; --i) {
                int u = buffer[i];
                if (dfs(u)) {
                    ++ans;
                    stop = false;
                    buffer[i] = buffer.back();
                    buffer.pop_back();
                }
            }
        } while (!stop);
        return ans;
    }

    vector<int> konig() {
        // returns minimum vertex cover, run this after maximumMatching()
        vector<bool> matched(m);
        for (int i = 0; i < n; ++i) {
            if (match[i] != -1) matched[match[i]] = true;
        }
        queue<int> Q;
        was.assign(m + n, false);
        for (int i = 0; i < m; ++i) {
            if (!matched[i]) {
                was[i] = true;
                Q.push(i);
            }
        }

        while (!Q.empty()) {
            int u = Q.front(); Q.pop();
            for (int v : a[u]) if (!was[m + v]) {
                was[m + v] = true;
                if (match[v] != -1 && !was[match[v]]) {
                    was[match[v]] = true;
                    Q.push(match[v]);
                }
            }
        }

        vector<int> res;
        for (int i = 0; i < m; ++i) {
            if (!was[i]) res.push_back(i);
        }
        for (int i = m; i < m + n; ++i) {
            if (was[i]) res.push_back(i);
        }

        return res;
    }
};
```

6.3 General Matching

```
/*
* Complexity: O(E*sqrt(V))
* Indexing from 1
*/
struct Blossom {
    static const int MAXV = 1e3 + 5;
    static const int MAXE = 1e6 + 5;
    int n, E, lst[MAXV], next[MAXE], adj[MAXE];
    int nxt[MAXV], mat[MAXV], dad[MAXV], col[MAXV];
    int que[MAXV], qh, qt;
    int vis[MAXV], act[MAXV];
    int tag, total;

    void init(int n) {
        this->n = n;
        for (int i = 0; i <= n; i++) {
            lst[i] = nxt[i] = mat[i] = vis[i] = 0;
        }
        E = 1, tag = total = 0;
    }
    void add(int u,int v) {
        if (!mat[u] && !mat[v]) mat[u] = v, mat[v] = u, total++;
        E++, adj[E] = v, next[E] = lst[u], lst[u] = E;
        E++, adj[E] = u, next[E] = lst[v], lst[v] = E;
    }
    int lca(int u, int v) {
        tag++;
        for(; ; swap(u, v)) {
            if (u) {
                if (vis[u = dad[u]] == tag) {
                    return u;
                }
                vis[u] = tag;
                u = nxt[mat[u]];
            }
        }
    }
    void blossom(int u, int v, int g) {
        while (dad[u] != g) {
            nxt[u] = v;
            if (col[mat[u]] == 2) {
                col[mat[u]] = 1;
                que[++qt] = mat[u];
            }
            if (u == dad[u]) dad[u] = g;
            if (mat[u] == dad[mat[u]]) dad[mat[u]] = g;
            v = mat[u];
            u = nxt[v];
        }
    }
    int augument(int s) {
        for (int i = 1; i <= n; i++) {
            col[i] = 0;
            dad[i] = i;
        }
        qh = 0; que[qt = 1] = s; col[s] = 1;
        for (int u, v, i; qh < qt; ) {
            act[u = que[++qh]] = 1;
            for (i = lst[u];i ; i = next[i]) {
                v = adj[i];
                if (col[v] == 0) {
                    nxt[v] = u;
                    col[v] = 2;
                    if (!mat[v]) {
                        for (; v; v = u) {
                            u = mat[nxt[v]];
                            mat[v] = nxt[v];
                            mat[nxt[v]] = v;
                        }
                        return 1;
                    }
                    col[mat[v]] = 1;
                    que[++qt] = mat[v];
                }
                else if (dad[u] != dad[v] && col[v] == 1) {
                    int g = lca(u, v);
                    blossom(u, v, g);
                    blossom(v, u, g);
                    for (int j = 1; j <= n; j++) {
                        dad[j] = dad[dad[j]];
                    }
                }
            }
        }
        return 0;
    }
    int maxmat() {
        for (int i = 1; i <= n; i++) {
            if (!mat[i]) {
                total += augument(i);
            }
        }
        return total;
    }
}
```

6.4 Dinic Flow

```
/*
    U = max capacity
    Complexity: O(V^2 * E)
    O(min(E^{1/2}, V^{2/3}) * E) if U = 1
    O(V^{1/2} * E)$ for bipartite matching.
*/
template <typename flow_t = int>
struct DinicFlow {
    const flow_t INF = numeric_limits<flow_t>::max() / 2; // 1e9

    int n, s, t;
    vector<vector<int>> adj;
    vector<int> d, cur;
    vector<int> to;
    vector<flow_t> c, f;

    DinicFlow(int n, int s, int t) : n(n), s(s), t(t), adj(n, vector<int>()),
            d(n, -1), cur(n, 0) {}

    int addEdge(int u, int v, flow_t _c) {
        adj[u].push_back(to.size());
        to.push_back(v); f.push_back(0); c.push_back(_c);
        adj[v].push_back(to.size());
        to.push_back(u); f.push_back(0); c.push_back(0);
        return (int)to.size() - 2;
    }

    bool bfs() {
        fill(d.begin(), d.end(), -1);
        d[s] = 0;
        queue<int> q;
        q.push(s);
        while (!q.empty()) {
            int u = q.front(); q.pop();
            for (auto edgeId : adj[u]) {
                int v = to[edgeId];
                if (d[v] == -1 && f[edgeId] < c[edgeId]) {
                    d[v] = d[u] + 1;
                    if (v == t) return 1;
                    q.push(v);
                }
            }
        }
        return d[t] != -1;
    }

    flow_t dfs(int u, flow_t res) {
        if (u == t || !res) return res;
        for (int &i = cur[u]; i < adj[u].size(); i++) {
            int edgeId = adj[u][i];
```

```
                int v = to[edgeId];
                if (d[v] == d[u] + 1 && f[edgeId] < c[edgeId]) {
                    flow_t foo = dfs(v, min(res, c[edgeId] - f[edgeId]));
                    if (foo) {
                        f[edgeId] += foo;
                        f[edgeId ^ 1] -= foo;
                        return foo;
                    }
                }
            }
            return 0;
        }

        flow_t maxFlow() {
            flow_t res = 0;
            while (bfs()) {
                fill(cur.begin(), cur.end(), 0);
                while (flow_t aug = dfs(s, INF)) res += aug;
            }
            return res;
        }
};
```

6.5 Dinic Flow With Scaling

```
/*
    U = max capacity
    Complexity: O(V * E * log(U))
    O(min(E^{1/2}, V^{2/3}) * E) if U = 1
    O(V^{1/2} * E)$ for bipartite matching.
    Tested: https://vn.spoj.com/problems/FFLOW/
    --> CHANGE LIM TO MAX CAPACITY<--
*/
template <typename flow_t = int>
struct DinicFlow {
    const flow_t INF = numeric_limits<flow_t>::max() / 2; // 1e9

    int n, s, t;
    vector<vector<int>> adj;
    vector<int> d, cur;
    vector<int> to;
    vector<flow_t> c, f;

    DinicFlow(int n, int s, int t) : n(n), s(s), t(t), adj(n, vector<int>()),
         d(n, -1), cur(n, 0) {}

    int addEdge(int u, int v, flow_t _c) {
        adj[u].push_back(to.size());
        to.push_back(v); f.push_back(0); c.push_back(_c);
        adj[v].push_back(to.size());
        to.push_back(u); f.push_back(0); c.push_back(0);
        return (int)to.size() - 2;
    }

    bool bfs(flow_t lim) {
        fill(d.begin(), d.end(), -1);
        d[s] = 0;
        queue<int> q;
        q.push(s);
        while (!q.empty()) {
            int u = q.front(); q.pop();
            for (auto edgeId : adj[u]) {
                int v = to[edgeId];
                if (d[v] == -1 && lim <= c[edgeId] - f[edgeId]) {
                    d[v] = d[u] + 1;
                    if (v == t) return 1;
                    q.push(v);
                }
            }
        }
        return d[t] != -1;
    }

    flow_t dfs(int u, flow_t res) {
        if (u == t || !res) return res;
        for (int &i = cur[u]; i < adj[u].size(); i++) {
            int edgeId = adj[u][i];
            int v = to[edgeId];
            if (d[v] == d[u] + 1 && res <= c[edgeId] - f[edgeId]) {
                flow_t foo = dfs(v, res);
                if (foo) {
                    f[edgeId] += foo;
                    f[edgeId ^ 1] -= foo;
                    return foo;
                }
            }
        }
        return 0;
    }

    flow_t maxFlow() {
        flow_t res = 0;
        flow_t lim = (flow_t)1 << int(round(log2(INF))); // change this to
             max capacity
        while (lim >= 1) {
            if (!bfs(lim)) {
                lim >>= 1;
                continue;
            }
            fill(cur.begin(), cur.end(), 0);
            while (flow_t aug = dfs(s, lim)) res += aug;
        }
        return res;
    }
};
```

6.6 Gomory Hu Tree

```
// a weighted tree that represents the minimum s-t cuts for all s-t pairs in
     the graph
vector<pair<pair<int, int>, flow_t> > gomory_hu() {
    vector<pair<pair<int, int>, flow_t> > tree;
    vector<int> p(n);
    for (int u = 1; u < n; u++) {
        tree.push_back(make_pair(make_pair(p[u], u), maxflow(u, p[u])));
        for (int v = u + 1; v < n; ++v) {
            if (lev[v] && p[v] == p[u]) {
                p[v] = u;
            }
        }
    }
    return tree;
}
```

6.7 Min Cost-Max Flow

```
/*
    Complexity: O(V^2 * E^2)
    O(VE) phases, O(VE) for SPFA
    Tested: https://open.kattis.com/problems/mincostmaxflow
*/
template <typename flow_t = int, typename cost_t = int>
struct MinCostMaxFlow {
    const flow_t FLOW_INF = numeric_limits<flow_t>::max() / 2; // 1e9
    const cost_t COST_INF = numeric_limits<cost_t>::max() / 2; // 1e9

    int n, s, t;
    vector<vector<int>> adj;
    vector<int> to;
    vector<flow_t> f, c;
    vector<cost_t> cost;

    vector<cost_t> d;
    vector<bool> inQueue;
    vector<int> prev;

    MinCostMaxFlow(int n, int s, int t) : n(n), s(s), t(t), adj(n, vector<int
          >()), d(n, -1), inQueue(n, 0), prev(n, -1) {}

    int addEdge(int u, int v, flow_t _c, cost_t _cost) {
        adj[u].push_back(to.size());
        to.push_back(v); f.push_back(0); c.push_back(_c); cost.push_back(
             _cost);
        adj[v].push_back(to.size());
        to.push_back(u); f.push_back(0); c.push_back(0); cost.push_back(-
             _cost);
        return (int)to.size() - 2;
    }

    pair<flow_t, cost_t> maxFlow() {
        flow_t res = 0;
        cost_t minCost = 0;
        while (1) {
            fill(prev.begin(), prev.end(), -1);
            fill(d.begin(), d.end(), COST_INF);
            d[s] = 0;
            inQueue[s] = 1;
            queue<int> q;
            q.push(s);
            while (!q.empty()) {
                int u = q.front();
                q.pop();
                inQueue[u] = 0;
                for (int id : adj[u]) {
                    int v = to[id];
                    if (d[v] > d[u] + cost[id] && f[id] < c[id]) {
                        d[v] = d[u] + cost[id];
                        prev[v] = id;
                        if (!inQueue[v]) {
                            inQueue[v] = 1;
                            q.push(v);
                        }
                    }
                }
            }
            if (prev[t] == -1) break;
            int x = t;
            flow_t now = FLOW_INF;
            while (x != s) {
                int id = prev[x];
                now = min(now, c[id] - f[id]);
                x = to[id ^ 1];
            }
            x = t;
            while (x != s) {
                int id = prev[x];
                minCost += cost[id] * now;
                f[id] += now;
                f[id ^ 1] -= now;
                x = to[id ^ 1];
            }
            res += now;
        }
        return {res, minCost};
    }
};
```

6.8 Min Cost Max Flow Potential

```
/*
    Complexity: O(VE * ElogN + VE)
    O(VE) phases, O(ElogN) for Dijkstra, O(VE) for the initial SPFA
    Tested: https://open.kattis.com/problems/mincostmaxflow
            https://codeforces.com/problemset/problem/164/C (92ms vs 936ms)

    --> RUN INIT BEFORE MAXFLOW IF WE HAVE NEG-EDGES <--
```

```cpp
*/
template <typename flow_t = int, typename cost_t = int>
struct MinCostMaxFlow {
    const flow_t FLOW_INF = numeric_limits<flow_t>::max() / 2; // 1e9
    const cost_t COST_INF = numeric_limits<cost_t>::max() / 2; // 1e9

    int n, s, t;
    vector<vector<int>> adj;
    vector<int> to;
    vector<flow_t> f, c;
    vector<cost_t> cost;

    vector<cost_t> pot;
    vector<cost_t> d;
    vector<int> prev;
    vector<bool> used;

    MinCostMaxFlow(int n, int s, int t) : n(n), s(s), t(t), adj(n, vector<int
          >()), d(n, -1), prev(n, -1), pot(n, 0), used(n, 0) {}

    int addEdge(int u, int v, flow_t _c, cost_t _cost) {
        adj[u].push_back(to.size());
        to.push_back(v); f.push_back(0); c.push_back(_c); cost.push_back(
              _cost);
        adj[v].push_back(to.size());
        to.push_back(u); f.push_back(0); c.push_back(0); cost.push_back(-
              _cost);
        return (int)to.size() - 2;
    }

    bool dijkstra() {
        fill(prev.begin(), prev.end(), -1);
        fill(d.begin(), d.end(), COST_INF);
        fill(used.begin(), used.end(), 0);
        d[s] = 0;
        set<pair<cost_t, int>> ss;
        ss.insert({0, s});
        while (!ss.empty()) {
            int u = ss.begin()->second; ss.erase(ss.begin());
            if (used[u]) continue;
            used[u] = 1;
            for (int id : adj[u]) {
                int v = to[id];
                cost_t now = d[u] + pot[u] - pot[v] + cost[id];
                if (!used[v] && d[v] > now && f[id] < c[id]) {
                    d[v] = now;
                    prev[v] = id;
                    ss.insert({d[v], v});
                }
            }
        }
        for (int i = 0; i < n; i++) pot[i] += d[i];
        return prev[t] != -1;
    }

    pair<flow_t, cost_t> maxFlow() {
        flow_t res = 0;
        cost_t minCost = 0;
        while (dijkstra()) {
            int x = t;
            flow_t now = FLOW_INF;
            while (x != s) {
                int id = prev[x];
                now = min(now, c[id] - f[id]);
                x = to[id ^ 1];
            }
            x = t;
            while (x != s) {
                int id = prev[x];
                minCost += cost[id] * now;
                f[id] += now;
                f[id ^ 1] -= now;
                x = to[id ^ 1];
            }
            res += now;
        }
        return {res, minCost};
    }

    void init() {
        fill(pot.begin(), pot.end(), COST_INF);
        pot[s] = 0;
        bool changed = 1;
        while (changed) { // be careful for NEG cycle
            changed = 0;
            for (int i = 0; i < n; i++) if (pot[i] < COST_INF) {
                for (int id : adj[i]) {
                    int v = to[id];
                    if (pot[v] > pot[i] + cost[id] && f[id] < c[id]) {
                        pot[v] = pot[i] + cost[id];
                        changed = 1;
                    }
                }
            }
        }
    }
};
```

6.9 Bounded Feasible Flow

```cpp
struct BoundedFlow {
    int low[N][N], high[N][N];
    int c[N][N];
    int f[N][N];
    int n, s, t;

    void reset() {
        memset(low, 0, sizeof low);
        memset(high, 0, sizeof high);
        memset(c, 0, sizeof c);
        memset(f, 0, sizeof f);
        n = s = t = 0;
    }

    void addEdge(int u, int v, int d, int c) {
        low[u][v] = d; high[u][v] = c;
    }

    int flow;
    int trace[N];

    bool findPath() {
        memset(trace, 0, sizeof trace);
        queue<int> Q;
        Q.push(s);
        while (!Q.empty()) {
            int u = Q.front(); Q.pop();
            for (int v = 1; v <= n; ++v) if (c[u][v] > f[u][v] && !trace[v])
                {
                trace[v] = u;
                if (v == t) return true;
                Q.push(v);
            }
        }
        return false;
    }

    void incFlow() {
        int delta = INF;
        for (int v = t; v != s; v = trace[v])
            delta = min(delta, c[trace[v]][v] - f[trace[v]][v]);
        for (int v = t; v != s; v = trace[v])
            f[trace[v]][v] += delta, f[v][trace[v]] -= delta;
        flow += delta;
    }

    int maxFlow() {
        flow = 0;
        while (findPath()) incFlow();
        return flow;
    }
    bool feasible() {
        c[t][s] = INF;
        s = n + 1; t = n + 2;
        int sum = 0;
        for (int u = 1; u <= n; ++u) for (int v = 1; v <= n; ++v) {
            c[s][v] += low[u][v];
            c[u][t] += low[u][v];
            c[u][v] += high[u][v] - low[u][v];
            sum += low[u][v];
        }
        n += 2;
        return maxFlow() == sum;
    }
};
```

6.10 Hungarian Algorithm

```cpp
struct BipartiteGraph {
    const int INF = 1e9;

    vector<vector<int> > c; // cost matrix
    vector<int> fx, fy; // potentials
    vector<int> matchX, matchY; // corresponding vertex
    vector<int> trace; // last vertex from the left side
    vector<int> d, arg; // distance from the tree && the corresponding node
    queue<int> Q; // queue used for BFS

    int n; // assume that |L| = |R| = n
    int start; // current root of the tree
    int finish; // leaf node of the augmenting path

    BipartiteGraph(int n) {
        this->n = n;
        c = vector<vector<int> >(n + 1, vector<int>(n + 1, INF));
        fx = fy = matchX = matchY = trace = d = arg = vector<int>(n + 1);
    }

    void addEdge(int u, int v, int cost) { c[u][v] = min(c[u][v], cost); }
    int cost(int u, int v) { return c[u][v] - fx[u] - fy[v]; }

    void initBFS(int root) {
        start = root;
        Q = queue<int>(); Q.push(start);
        for (int i = 1; i <= n; ++i) {
            trace[i] = 0;
            d[i] = cost(start, i);
            arg[i] = start;
        }
    }

    int findPath() {
        while (!Q.empty()) {
            int u = Q.front(); Q.pop();
            for (int v = 1; v <= n; ++v) if (trace[v] == 0) {
                int w = cost(u, v);
                if (w == 0) {
                    trace[v] = u;
                    if (matchY[v] == 0) return v;
                    Q.push(matchY[v]);
                }
                if (d[v] > w) d[v] = w, arg[v] = u;
            }
        }
        return 0;
    }

    void enlarge() {
        for (int y = finish, next; y; y = next) {
            int x = trace[y];
            next = matchX[x];
            matchX[x] = y;
            matchY[y] = x;
        }
    }

    void update() {
```

```
        int delta = INF;
        for (int i = 1; i <= n; ++i) if (trace[i] == 0) delta = min(delta, d[
            i]);
        fx[start] += delta;
        for (int i = 1; i <= n; ++i) {
            if (trace[i] != 0) {
                fx[matchY[i]] += delta;
                fy[i] -= delta;
            } else {
                d[i] -= delta;
                if (d[i] == 0) {
                    trace[i] = arg[i];
                    if (matchY[i] == 0)
                        finish = i;
                    else
                        Q.push(matchY[i]);
                }
            }
        }
    }

    void hungarian() {
        for (int i = 1; i <= n; ++i) {
            initBFS(i);
            do {
                finish = findPath();
                if (finish == 0) update();
            } while (finish == 0);
            enlarge();
        }
    }

    void show() {
        int ans = 0;
        for (int i = 1; i <= n; ++i) if (matchX[i]) ans += c[i][matchX[i]];
        cout << ans << endl;
        for (int i = 1; i <= n; ++i) cout << i << ' ' << matchX[i] << endl;
    }
};
```

6.11 Undirected mincut

```
/*
 * Find minimum cut in undirected weighted graph
 * Complexity: O(V^3)
 */
#define SW StoerWagner
#define cap_t int
namespace StoerWagner {
    int n;
    vector<vector<cap_t> > graph;
    vector<int> cut;

    void init(int _n) {
        n = _n;
        graph = vector<vector<cap_t>>(n, vector<cap_t>(n, 0));
    }
    void addEdge(int a, int b, cap_t w) {
        if (a == b) return;
        graph[a][b] += w;
        graph[b][a] += w;
    }
    pair<cap_t, pair<int, int> > stMinCut(vector<int> &active) {
        vector<cap_t> key(n);
        vector<int> v(n);
        int s = -1, t = -1;
        for (int i = 0; i < active.size(); i++) {
            cap_t maxv = -1;
            int cur = -1;
            for (auto j : active) {
                if (v[j] == 0 && maxv < key[j]) {
                    maxv = key[j];
                    cur = j;
                }
            }
            t = s;
            s = cur;
            v[cur] = 1;
            for (auto j : active) key[j] += graph[cur][j];
        }
        return make_pair(key[s], make_pair(s, t));
    }
    cap_t solve() {
        cap_t res = numeric_limits <cap_t>::max();
        vector<vector<int>> grps;
        vector<int> active;
        cut.resize(n);
        for (int i = 0; i < n; i++) grps.emplace_back(1, i);
        for (int i = 0; i < n; i++) active.push_back(i);
        while (active.size() >= 2) {
            auto stcut = stMinCut(active);
            if (stcut.first < res) {
                res = stcut.first;
                fill(cut.begin(), cut.end(), 0);
                for (auto v : grps[stcut.second.first]) cut[v] = 1;
            }
            int s = stcut.second.first, t = stcut.second.second;
            if (grps[s].size() < grps[t].size()) swap(s, t);
            active.erase(find(active.begin(), active.end(), t));
            grps[s].insert(grps[s].end(), grps[t].begin(), grps[t].end());
            for (int i = 0; i < n; i++) {
                graph[i][s] += graph[i][t];
                graph[i][t] = 0;
            }
            for (int i = 0; i < n; i++) {
                graph[s][i] += graph[t][i];
                graph[t][i] = 0;
            }
            graph[s][s] = 0;
        }
        return res;
    }
}
```

6.12 Eulerian Path/Circuit

```
struct EulerianGraph {
    vector< vector< pair<int, int> > > a;
    int num_edges;

    EulerianGraph(int n) {
        a.resize(n + 1);
        num_edges = 0;
    }

    void add_edge(int u, int v, bool undirected = true) {
        a[u].push_back(make_pair(v, num_edges));
        if (undirected) a[v].push_back(make_pair(u, num_edges));
        num_edges++;
    }

    vector<int> get_eulerian_path() {
        vector<int> path, s;
        vector<bool> was(num_edges);

        s.push_back(1);
        // start of eulerian path
        // directed graph: deg_out - deg_in == 1
        // undirected graph: odd degree
        // for eulerian cycle: any vertex is OK

        while (!s.empty()) {
            int u = s.back();
            bool found = false;
            while (!a[u].empty()) {
                int v = a[u].back().first;
                int e = a[u].back().second;
                a[u].pop_back();
                if (was[e]) continue;
                was[e] = true;
                s.push_back(v);
                found = true;
                break;
            }
            if (!found) {
                path.push_back(u);
                s.pop_back();
            }
        }
        reverse(path.begin(), path.end());
        return path;
    }
};
```

6.13 2-SAT

```
inline int pos(int u) { return u << 1; }
inline int neg(int u) { return u << 1 | 1; }
// ZERO-indexed
// color[i] = 1 means we choose i
struct TwoSAT {
    int n;
    int numComp;
    vector<int> adj[V];
    int low[V], num[V], root[V], cntTarjan;
    vector<int> stTarjan;
    int color[V];

    TwoSAT(int n) : n(n * 2) {
        memset(root, -1, sizeof root);
        memset(low, -1, sizeof low);
        memset(num, -1, sizeof num);
        memset(color, -1, sizeof color);
        cntTarjan = 0;
        stTarjan.clear();
    }

    // u | v
    void addEdge(int u, int v) {
        adj[u ^ 1].push_back(v);
        adj[v ^ 1].push_back(u);
    }

    void tarjan(int u) {
        stTarjan.push_back(u);
        num[u] = low[u] = cntTarjan++;
        for (int v : adj[u]) {
            if (root[v] != -1) continue;
            if (low[v] == -1) tarjan(v);
            low[u] = min(low[u], low[v]);
        }
        if (low[u] == num[u]) {
            while (1) {
                int v = stTarjan.back();
                stTarjan.pop_back();
                root[v] = numComp;
                if (u == v) break;
            }
            numComp++;
        }
    }

    bool solve() {
        for (int i = 0; i < n; i++) if (root[i] == -1) tarjan(i);
        for (int i = 0; i < n; i += 2) {
            if (root[i] == root[i ^ 1]) return 0;
            color[i >> 1] = (root[i] < root[i ^ 1]);
        }
        return 1;
    }
};
```

6.14 SPFA

```
struct Graph {
    vector< vector< pair<int, int> > > a;
    vector<int> d;
    int n;

    Graph(int n) {
        this->n = n;
        a.resize(n);
    }

    void add_edge(int u, int v, int c) {
        // x[u] - x[v] <= c
        a[v].push_back(make_pair(u, c));
    }

    bool spfa(int s) {
        // return false if found negative cycle from s
        queue<int> Q;
        vector<bool> inqueue(n);
        d.assign(n, INF);
        d[s] = 0;
        Q.push(s); inqueue[s] = 1;

        vector<int> cnt(n);
        cnt[s] = 1;

        while (!Q.empty()) {
            int u = Q.front(); Q.pop(); inqueue[u] = 0;
            for (auto e : a[u]) {
                int v = e.first;
                int c = e.second;
                if (d[v] > d[u] + c) {
                    d[v] = d[u] + c;
                    cnt[v]++;
                    if (cnt[v] >= n) return false;
                    if (!inqueue[v]) {
                        Q.push(v);
                        inqueue[v] = 1;
                    }
                }
            }
        }

        return true;
    }

    int spfa(int s, int t) {
        assert(spfa(s));
        return d[t];

    }
};
```

7 Data structures

7.1 Treap

```
class Treap {
    struct Node {
        int key;
        uint32_t prior;
        bool rev_lazy;
        int size;
        Node *l, *r;
        Node(int key): key(key), prior(rand()), rev_lazy(false), size(1), l(
              nullptr), r(nullptr) {}
        ~Node() { delete l; delete r; }
    };

    inline int size(Node *x) { return x ? x->size : 0; }

    void push(Node *x) {
        if (x && x->rev_lazy) {
            x->rev_lazy = false;
            swap(x->l, x->r);
            if (x->l) x->l->rev_lazy ^= true;
            if (x->r) x->r->rev_lazy ^= true;
        }
    }

    inline void update(Node *x) {
        if (x) {
            x->size = size(x->l) + size(x->r) + 1;
        }
    }

    void join(Node *&t, Node *l, Node *r) {
        push(l); push(r);
        if (!l || !r)
            t = l ? l : r;
        else if (l->prior < r->prior)
            join(l->r, l->r, r), t = l;
        else
            join(r->l, l, r->l), t = r;
        update(t);
    }

    void splitByKey(Node *v, int x, Node* &l, Node* &r) {
        if (!v) return void(l = r = nullptr);
        push(v);
        if (v->key < x)
            splitByKey(v->r, x, v->r, r), l = v;
        else
            splitByKey(v->l, x, l, v->l), r = v;
        update(v);
    }

    void splitByIndex(Node *v, int x, Node* &l, Node* &r) {
        if (!v) return void(l = r = nullptr);
        push(v);
        int index = size(v->l) + 1;
        if (index < x)
            splitByIndex(v->r, x - index, v->r, r), l = v;
        else
            splitByIndex(v->l, x, l, v->l), r = v;
        update(v);
    }

    void show(Node *x) {
        if (!x) return;
        push(x);
        show(x->l);
        cerr << x->key << ' ';
        show(x->r);
    }

    Node *root;
    Node *l, *m, *r;

    public:
    Treap() { root = NULL; }
    ~Treap() { delete root; }
    int size() { return size(root); }

    int insert(int x) {
        splitByKey(root, x, l, m);
        splitByKey(m, x + 1, m, r);
        int ans = 0;
        if (!m) m = new Node(x), ans = size(l) + 1;
        join(l, l, m);
        join(root, l, r);
        return ans;
    }

    int erase(int x) {
        splitByKey(root, x, l, m);
        splitByKey(m, x + 1, m, r);
        int ans = 0;
        if (m) {
            ans = size(l) + 1;
            delete m;
        }
        join(root, l, r);
        return ans;
    }

    void insertAt(int pos, int x) {
        splitByIndex(root, pos, l, r);
        join(l, l, new Node(x));
        join(root, l, r);
    }

    void eraseAt(int x) {
        splitByIndex(root, x, l, m);
        splitByIndex(m, 2, m, r);
        delete m;
        join(root, l, r);
    }

    void updateAt(int pos, int newValue) {
        eraseAt(pos);
        insertAt(pos, newValue);
    }

    int valueAt(int pos) {
        splitByIndex(root, pos, l, m);
        splitByIndex(m, 2, m, r);
        int res = m->key;
        join(l, l, m);
        join(root, l, r);
        return res;
    }

    void reverse(int from, int to) {
        splitByIndex(root, from, l, m);
        splitByIndex(m, to - from + 2, m, r);
        m->rev_lazy ^= 1;
        join(l, l, m);
        join(root, l, r);
    }

    void show() {
        cerr << "Size = " << size() << " ";
        cerr << "[";
        show(root);
        cerr << "]\n";
    }
};
```

7.2 Big Integer

```
typedef vector<int> bigInt;
const int BASE = 1000;
const int LENGTH = 3;

// * Refine function
bigInt& fix(bigInt &a) {
    a.push_back(0);
    for (int i = 0; i + 1 < a.size(); ++i) {
        a[i + 1] += a[i] / BASE; a[i] %= BASE;
        if (a[i] < 0) a[i] += BASE, --a[i + 1];
    }
    while (a.size() > 1 && a.back() == 0) a.pop_back();
    return a;
}

// * Constructors
bigInt big(int x) {
```

```
    bigInt result;
    while (x > 0) {
        result.push_back(x % BASE);
        x /= BASE;
    }
    return result;
}

bigInt big(string s) {
    bigInt result(s.size() / LENGTH + 1);
    for (int i = 0; i < s.size(); ++i) {
        int pos = (s.size() - i - 1) / LENGTH;
        result[pos] = result[pos] * 10 + s[i] - '0';
    }
    return fix(result), result;
}

// * Compare operators

int compare(bigInt &a, bigInt &b) {
    if (a.size() != b.size()) return (int)a.size() - (int)b.size();
    for (int i = (int) a.size() - 1; i >= 0; --i)
        if (a[i] != b[i]) return a[i] - b[i];
    return 0;
}

#define DEFINE_OPERATOR(x) bool operator x (bigInt &a, bigInt &b) { return
      compare(a, b) x 0; }
DEFINE_OPERATOR(==)
DEFINE_OPERATOR(!=)
DEFINE_OPERATOR(>)
DEFINE_OPERATOR(<)
DEFINE_OPERATOR(>=)
DEFINE_OPERATOR(<=)
#undef DEFINE_OPERATOR

// * Arithmetic operators

void operator += (bigInt &a, bigInt b) {
    a.resize(max(a.size(), b.size()));
    for (int i = 0; i < b.size(); ++i)
        a[i] += b[i];
    fix(a);
}

void operator -= (bigInt &a, bigInt b) {
    for (int i = 0; i < b.size(); ++i)
        a[i] -= b[i];
    fix(a);
}

void operator *= (bigInt &a, int b) {
    for (int i = 0; i < a.size(); ++i)
        a[i] *= b;
    fix(a);
}

void divide(bigInt a, int b, bigInt &q, int &r) {
    for (int i = int(a.size()) - 1; i >= 0; --i) {
        r = r * BASE + a[i];
        q.push_back(r / b); r %= b;
    }
    reverse(q.begin(), q.end());
    fix(q);
}

bigInt operator + (bigInt a, bigInt b) { a += b; return a; }
bigInt operator - (bigInt a, bigInt b) { a -= b; return a; }
bigInt operator * (bigInt a, int b) { a *= b; return a; }

bigInt operator / (bigInt a, int b) {
    bigInt q; int r = 0;
    divide(a, b, q, r);
    return q;
}
int operator % (bigInt a, int b) {
    bigInt q; int r = 0;
    divide(a, b, q, r);
    return r;
}

bigInt operator * (bigInt a, bigInt b) {
    bigInt result (a.size() + b.size());
    for (int i = 0; i < a.size(); ++i)
        for (int j = 0; j < b.size(); ++j)
            result[i + j] += a[i] * b[j];
    return fix(result);
}

// * I/O routines

istream& operator >> (istream& cin, bigInt &a) {
    string s; cin >> s;
    a = big(s);
    return cin;
}

ostream& operator << (ostream& cout, const bigInt &a) {
    cout << a.back();
    for (int i = (int)a.size() - 2; i >= 0; --i)
        cout << setw(LENGTH) << setfill('0') << a[i];
    return cout;
}
```

7.3 Convex Hull IT

```
struct Line {
    long long a, b; // y = ax + b
    Line(long long a = 0, long long b = -INF): a(a), b(b) {}
    long long eval(long long x) {
        return a * x + b;
    }
};

struct Node {
    Line line;
    int l, r;
    Node *left, *right;

    Node(int l, int r): l(l), r(r), left(NULL), right(NULL), line() {}

    void update(int i, int j, Line newLine) {
        if (r < i || j < l) return;
        if (i <= l && r <= j) {
            Line AB = line, CD = newLine;
            if (AB.eval(valueX[l]) < CD.eval(valueX[l])) swap(AB, CD);
            if (AB.eval(valueX[r]) >= CD.eval(valueX[r])) {
                line = AB;
                return;
            }
            int mid = valueX[l + r >> 1];
            if (AB.eval(mid) < CD.eval(mid))
                line = CD, left->update(i, j, AB);
            else
                line = AB, right->update(i, j, CD);
            return;
        }
        left->update(i, j, newLine);
        right->update(i, j, newLine);
    }

    long long getMax(int i) {
        if (l == r) return line.eval(valueX[i]);
        if (i <= (l + r >> 1)) return max(line.eval(valueX[i]), left->getMax(
            i));
        return max(line.eval(valueX[i]), right->getMax(i));
    }
};

Node* build(int l, int r) {
    Node *x = new Node(l, r);
    if (l == r) return x;
    x->left = build(l, l + r >> 1);
    x->right = build((l + r >> 1) + 1, r);
    return x;
}
```

7.4 Link Cut Tree

```
// treequery returns sum weight of child in subtree
// to change it to sum weight of child in root->u
// comment all update on w and return x->s instead
struct node_t {
    node_t *p, *l, *r;
    int size, rev;
    int s, w;
    node_t() : p(0), l(0), r(0), size(1), rev(0), s(1), w(1) {}
};

int isrt(node_t* x) {
    return !(x->p) || (x->p->l != x && x->p->r != x);
}

int left(node_t* x) {
    return x->p->l == x;
}

void setchild(node_t* x, node_t* p, int l) {
    (l ? p->l : p->r) = x;
    if (x) x->p = p;
}

void push(node_t* x) {
    node_t* u = x->l;
    node_t* v = x->r;
    if (x->rev) {
        if (u) swap(u->l, u->r), u->rev ^= 1;
        if (v) swap(v->l, v->r), v->rev ^= 1;
        x->rev = 0;
    }
}

int size(node_t* x) {
    return x ? x->size : 0;
}

int sum(node_t* x) {
    return x ? x->s : 0;
}

void pull(node_t* x) {
    x->size = size(x->l) + 1 + size(x->r);
    x->s = sum(x->l) + x->w + sum(x->r);
}

void rotate(node_t* x) {
    node_t *p = x->p, *g = p->p;
    int l = left(x);
    setchild(l ? x->r : x->l, p, l);
    if (!isrt(p)) setchild(x, g, left(p));
    else x->p = g;
    setchild(p, x, !l);
    pull(p);
}

node_t* splay(node_t* x) {
    push(x);
    while (!isrt(x)) {
        node_t *p = x->p, *g = p->p;
        if (g) push(g);
        push(p), push(x);
        if (!isrt(p)) rotate(left(x) != left(p) ? x : p);
        rotate(x);
    }
    pull(x);
    return x;
```

```cpp
}

node_t* access(node_t* x) {
    node_t* z = 0;
    for (node_t* y = x; y; y = y->p) {
        splay(y);
        y->w += sum(y->r);
        y->r = z;
        y->w -= sum(y->r);
        pull(z = y);
    }
    splay(x);
    return z;
}

void link(node_t* x, node_t* p) {
    access(x), access(p);
    x->p = p;
    p->w += sum(x);
}

void cut(node_t* x) {
    access(x);
    x->l->p = 0, x->l = 0;
    pull(x);
}

void makeroot(node_t* x) {
    access(x);
    x->rev ^= 1;
    swap(x->l, x->r);
}

node_t* findroot(node_t* x) {
    access(x);
    while (x->l) push(x), x = x->l;
    push(x);
    return splay(x);
}

node_t* lca(node_t* x, node_t* y) {
    if (findroot(x) != findroot(y)) return 0;
    access(x);
    return access(y);
}

int connect(node_t* x, node_t* y) {
    if (x == y) return 1;
    access(x), access(y);
    return x->p != 0;
}

int treequery(node_t* x) {
    access(x);
    return x->w;
}
```

7.5 Ordered Set

```cpp
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
/*
    change null_type to int if we want to use map instead
    find_by_order(k) returns an iterator to the k-th element (0-indexed)
    order_of_key(k) returns numbers of item being strictly smaller than k
*/
template<typename T = int>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
      tree_order_statistics_node_update>;
```

8 Miscellaneous

8.1 RNG

```cpp
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
//use mt19937_64 if we want 64-bit number
```

8.2 SQRT forloop

```cpp
for (int i = 1, la; i <= n; i = la + 1) {
    la = n / (n / i);
    //n / x yields the same value for i <= x <= la.
}
```

1 Chinese remainder theorem

Let m, n, a, b be any integers, let $g = \gcd(m, n)$, and consider the system of congruences:

$$x \equiv a \pmod{n}$$
$$x \equiv b \pmod{n}$$

If $a \equiv b \pmod{g}$, then this system of equations has a unique solution modulo $\operatorname{lcm}(m,n) = \frac{mn}{g}$. Otherwise, it has no solutions.
If we use Bézout's identity to write $g = um + vn$, then the solution is

$$x = \frac{avn + bum}{g}.$$

This defines an integer, as g divides both m and n. Otherwise, the proof is very similar to that for coprime moduli.

2 Eigen Decomposition

A (non-zero) vector v of dimension N is an eigenvector of a square $N \times N$ matrix A if it satisfies the linear equation

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$$

where λ is a scalar, termed the eigenvalue corresponding to v.
This yields an equation for the eigenvalues

$$p(\lambda) = \det(\mathbf{A} - \lambda\mathbf{I}) = 0$$

.
This equation will have $N\lambda$ distinct solutions, where $1 \le N\lambda \le N$. The set of solutions, that is, the eigenvalues, is called the spectrum of A.
We can factor p as

$$p(\lambda) = (\lambda - \lambda_1)^{n_1} (\lambda - \lambda_2)^{n_2} \cdots (\lambda - \lambda_{N_\lambda})^{n_{N_\lambda}} = 0$$

.
The integer n_i is termed the algebraic multiplicity of eigenvalue λ_i. If the field of scalars is algebraically closed, the algebraic multiplicities sum to N:

$$\sum_{i=1}^{N_\lambda} n_i = N.$$

For each eigenvalue λ_i, we have a specific eigenvalue equation

$$(\mathbf{A} - \lambda_i\mathbf{I})\,\mathbf{v} = 0.$$

There will be $1 \le m_i \le n_i$ linearly independent solutions to each eigenvalue equation. The linear combinations of the m_i solutions are the eigenvectors associated with the eigenvalue λ_i. The integer m_i is termed the geometric multiplicity of λ_i. It is important to keep in mind that the algebraic multiplicity n_i and geometric multiplicity m_i may or may not be equal, but we always have $m_i \le n_i$. The simplest case is of course when $m_i = n_i = 1$. The total number of linearly independent eigenvectors, N_v, can be calculated by summing the geometric multiplicities

$$\sum_{i=1}^{N_\lambda} m_i = N_{\mathbf{v}}.$$

The eigenvectors can be indexed by eigenvalues, using a double index, with v_{ij} being the jth eigenvector for the ith eigenvalue. The eigenvectors can also be indexed using the simpler notation of a single index v_k, with $k = 1, 2, \ldots, N_v$.
Let A be a square $n \times n$ matrix with n linearly independent eigenvectors q_i (where $i = 1, \ldots, n$). Then A can be factorized as

$$\mathbf{A} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{-1}$$

where Q is the square $n \times n$ matrix whose ith column is the eigenvector q_i of A, and Λ is the diagonal matrix whose diagonal elements are the corresponding eigenvalues, $\lambda_{ii} = \lambda_i$.
The n eigenvectors q_i are usually normalized, but they need not be. A non-normalized set of n eigenvectors, v_i can also be used as the columns of Q. That can be understood by noting that the magnitude of the eigenvectors in Q gets canceled in the decomposition by the presence of $Q - 1$.
The decomposition can be derived from the fundamental property of eigenvectors:

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$$
$$\mathbf{A}\mathbf{Q} = \mathbf{Q}\mathbf{\Lambda}$$
$$\mathbf{A} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{-1}.$$

If a matrix A can be eigendecomposed and if none of its eigenvalues are zero, then A is nonsingular and its inverse is given by

$$\mathbf{A}^{-1} = \mathbf{Q}\mathbf{\Lambda}^{-1}\mathbf{Q}^{-1}$$

If $\mathbf{A}$ is a symmetric matrix, since $\mathbf{Q}$ is formed from the eigenvectors of $\mathbf{A}$ it is guaranteed to be an orthogonal matrix, therefore $\mathbf{Q}^{-1} = \mathbf{Q}^{\mathrm{T}}$. Furthermore, because Λ is a diagonal matrix, its inverse is easy to calculate:

$$\left[\Lambda^{-1}\right]_{ii} = \frac{1}{\lambda_i}$$

3 Generating function

$$\sum_{n=0}^{\infty} a^n \binom{n+k}{k} x^n = \frac{1}{(1-ax)^{k+1}}.$$

4 Partition

The number of partitions of n is the partition function $p(n)$ having generating function:

$$\sum_{n=0}^{\infty} p(n)x^n = \prod_{k=1}^{\infty} (1 - x^k)^{-1}$$

$$p_n = p_{n-1} + p_{n-2} - p_{n-5} - p_{n-7} + p_{n-12} + p_{n-15} - p_{n-22} - \ldots$$

$$p_k = k(3k-1)/2 \text{ with } k = 1, -1, 2, -2, 3, -3, \ldots$$

5 Center of mass + Green theorem

Let C be a positively oriented, piecewise smooth, simple closed curve in a plane, and let D be the region bounded by C. If L and M are functions of (x, y) defined on an open region containing D and having continuous partial derivatives there, then

$$\oint_C (L\,dx + M\,dy) = \iint_D \left(\frac{\partial M}{\partial x} - \frac{\partial L}{\partial y}\right) dx\,dy$$

where the path of integration along C is anticlockwise.
The centroid of a non-self-intersecting closed polygon defined by n vertices $(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})$ is the point (C_x, C_y) where

$$C_\mathrm{x} = \frac{1}{6A} \sum_{i=0}^{n-1} (x_i + x_{i+1})(x_i\ y_{i+1} - x_{i+1}\ y_i), \text{and}$$
$$C_\mathrm{y} = \frac{1}{6A} \sum_{i=0}^{n-1} (y_i + y_{i+1})(x_i\ y_{i+1} - x_{i+1}\ y_i),$$

and where A is the polygon's signed area, as described by the shoelace formula:

$$A = \frac{1}{2} \sum_{i=0}^{n-1} (x_i\ y_{i+1} - x_{i+1}\ y_i).$$

In these formulae, the vertices are assumed to be numbered in order of their occurrence along the polygon's perimeter; furthermore, the vertex (x_n, y_n) is assumed to be the same as (x_0, y_0), meaning $i + 1$ on the last case must loop around to $i = 0$. (If the points are numbered in clockwise order, the area A, computed as above, will be negative; however, the centroid coordinates will be correct even in this case.)

6 Fibonacci mod $10^9 + 9$

$$F_n \equiv 276601605(691504013^n - 308495997^n) \pmod{10^9 + 9}$$
$$F_n = \frac{\varphi^n - \psi^n}{\varphi - \psi} = \frac{\varphi^n - \psi^n}{\sqrt{5}}$$
where
$$\varphi = \frac{1 + \sqrt{5}}{2} \approx 1.61803\,39887\ldots$$
$$\psi = \frac{1 - \sqrt{5}}{2} = 1 - \varphi = -\frac{1}{\varphi} \approx -0.61803\,39887\ldots.$$

Properties

$$(-1)^n = F_{n+1}F_{n-1} - F_n^2.$$

$$F_m F_n + F_{m-1}F_{n-1} = F_{m+n-1},$$
$$F_m F_{n+1} + F_{m-1}F_n = F_{m+n}.$$

In particular, with $m = n$,

$$F_{2n-1} = F_n^2 + F_{n-1}^2$$
$$F_{2n} = (F_{n-1} + F_{n+1})F_n$$
$$= (2F_{n-1} + F_n)F_n.$$
$$\sum_{i=1}^{n} F_i = F_{n+2} - 1$$
$$\sum_{i=0}^{n-1} F_{2i+1} = F_{2n}$$
$$\sum_{i=1}^{n} F_{2i} = F_{2n+1} - 1.$$
$$\sum_{i=1}^{n} {F_i}^2 = F_n F_{n+1}$$

7 Möbius inversion formula

The classic version states that if g and f are arithmetic functions satisfying

$$g(n) = \sum_{d|n} f(d) \quad \text{for every integer } n \ge 1$$

then

$$f(n) = \sum_{d|n} \mu(d) g\left(\frac{n}{d}\right) \quad \text{for every integer } n \ge 1$$

- ε is the multiplicative identity: $\varepsilon(1) = 1$, otherwise 0.
- Id is the identity function with value n: $\mathrm{Id}(n) = n$.
- $1 * \mu = \varepsilon$, the Dirichlet inverse of the constant function 1 is the Möbius function.
- $g = f * 1$ if and only if $f = g * \mu$, the Möbius inversion formula
- $\phi * 1 = \mathrm{Id}$, proved under Euler's totient function

8 Planar graph

Euler's formula:

$$v - e + f = 2.$$

In a finite, connected, simple, planar graph, any face (except possibly the outer one) is bounded by at least three edges and every edge touches at most two faces; using Euler's formula, one can then show that these graphs are sparse in the sense that if $v \ge 3$:

$$e \le 3v - 6.$$

The **dual graph** of a plane graph G is a graph that has a vertex for each face of G.
In the complement dual graph: (removed egdes in the original =¿ edges in dual): a **connected component** is equivalent to a **face** in dual graph.

9 Pell equation

$$x^2 - 2y^2 = 1$$

If x_1, y_1 is the minimal solution then:

$$x_{k+1} = x_1 x_k + n y_1 y_k,$$
$$y_{k+1} = x_1 y_k + y_1 x_k.$$

10 Burnside lemma

let G be a finite group that acts on a set X. For each g in G let X^g denote the set of elements in X that are fixed by g (also said to be left invariant by g), i.e. $X^g = \{x \in X | g.x = x\}$. Burnside's lemma asserts the following formula for the number of orbits, denoted $|X/G|$:

$$|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|.$$

11 Euler function

Gamma:

$$\Gamma(z) = \int_0^\infty x^{z-1} e^{-x}\, dx, \qquad \Re(z) > 0 \ .$$
$$\Gamma(n) = (n-1)! \ .$$
$$\Gamma(n+1) = n\Gamma(n)$$
$$\Gamma(1-z)\Gamma(z) = \frac{\pi}{\sin(\pi z)}, \qquad z \notin \mathbb{Z}$$
$$\Gamma\left(\tfrac{1}{2}\right) = \sqrt{\pi},$$

Beta

$$\mathrm{B}(x,y) = \int_0^1 t^{x-1}(1-t)^{y-1}\, dt$$
$$\mathrm{B}(x,y) = \mathrm{B}(y,x)$$
$$\mathrm{B}(x,y) = \frac{\Gamma(x)\,\Gamma(y)}{\Gamma(x+y)}.$$
$$\Gamma(x)\Gamma(y) = \int_{\mathbb{R}} f(u)\, du \cdot \int_{\mathbb{R}} g(u)\, du = \int_{\mathbb{R}} (f * g)(u)\, du = \mathrm{B}(x,y)\,\Gamma(x+y).$$

12 3 mutually tangent circles

Given 3 mutually tangent circles. Find inner circle (touching all 3) and outer circle (touching all 3). The radius is given by:

$$k4 = |k1 + k2 + k3 \pm 2 * \sqrt{k1 * k2 + k2 * k3 + k3 * k1}|$$

where $ki = 1/r_i$
Minus → Outer
Plus → Inner
Special cases: If 1 circle → line, change k_i to 0, the radius:

$$k4 = k1 + k2 \pm 2 * \sqrt{k1 * k2}$$

13 Hacken Bush

Green Hacken Bush: subtree of u: $g(u) = \bigoplus_v g(v) + 1$ with v is a child of u.
RB Hacken Bush:

- *Rooted tree* u: $g(u) = \sum f(g(v))$ with v is a child of u.
  - If color of u, v is blue: $f(x) = \frac{x+i}{2^{i-1}}$ with smallest $i \geq 1$ such that $x + i > +1$
  - If color of u, v is red: $f(x) = \frac{x-i}{2^{i-1}}$ with smallest $i \geq 1$ such that $x - i < -1$
- *Loop*: find 2 nearest 2 points where segment change color, cut the rest in half the value of loop is sum of the 2 segments. If there are an odd number, cut the middle segment in half and treat it as two segments
- *Stalk*: Count the number of blue (or red) edges that are connected in one continuous path. If there are n of them, start with the number n. For each new edge going up, assign that value of that edge to be half of the one below it. If it is a blue edge, make it positive. If it is a red edge, make it negative.

14 Prüfer sequence

- Get prufer code of a tree
  - Find a leaf of lowest label x, connect to y. Remove x, add y to the sequence
  - Repeat until we are left with 2 nodes
- Construct a tree
  - Let the first element is X, find a node which doesn't appear in the sequence L
  - Add edge X, L
  - Remove X

Cayley's formula

- The number of trees on n labeled vertices is n^{n-2}.
- The number of labelled rooted forests on n vertices, namely $(n+1)^{n-1}$.
- The number of labelled forests on n vertices with k connected components, such that vertices $1, 2, \ldots, k$ all belong to different connected components is kn^{n-k-1}.

15 Graph realization

Erdős–Gallai theorem
A sequence of non-negative integers $d_1 \geq \cdots \geq d_n$ can be represented as the degree sequence of a finite simple graph on n vertices if and only if $d_1 + \cdots + d_n$ is even and

$$\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k)$$

holds for every k in $1 \leq k \leq n$.
Fulkerson–Chen–Anstee theorem
A sequence $((a_1, b_1), \ldots, (a_n, b_n))$ of nonnegative integer pairs with $a_1 \geq \cdots \geq a_n$ is digraphic if and only if $\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} b_i$ and the following inequality holds for k such that $1 \leq k \leq n$:

$$\sum_{i=1}^{k} a_i \le \sum_{i=1}^{k} \min(b_i, k-1) + \sum_{i=k+1}^{n} \min(b_i, k)$$

Gale–Ryser theorem
A pair of sequences of nonnegative integers $(a_1, \dots, a_n)$ and $(b_1, \dots, b_n)$ with $a_1 \ge \dots \ge a_n$ is bigraphic if and only if $\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} b_i$ and the following inequality holds for k such that $1 \le k \le n$:

$$\sum_{i=1}^{k} a_i \le \sum_{i=1}^{n} \min(b_i, k).$$

16 Binomial coefficient

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}.$$

$$\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$$

$$\binom{n}{h}\binom{n-h}{k} = \binom{n}{k}\binom{n-k}{h}$$

$$\sum_{j=0}^{k}\binom{m}{j}\binom{n-m}{k-j} = \binom{n}{k}$$

$$\sum_{j=0}^{m}\binom{m}{j}^2 = \binom{2m}{m},$$

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

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

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

$$\sum_{k=0}^{\lfloor n/2 \rfloor}\binom{n-k}{k} = F(n+1).$$

17 Kőnig's theorem

Kőnig's theorem states that, in any bipartite graph, the **minimum vertex cover set** and the **maximum matching set** have in fact the same size.

Constructive proof
The following proof provides a way of constructing a minimum vertex cover from a maximum matching. Let $G = (V, E)$ be a bipartite graph and let L, R be the two parts of the vertex set V. Suppose that M is a maximum matching for G. No vertex in a vertex cover can cover more than one edge of M (because the edge half-overlap would prevent M from being a matching in the first place), so if a vertex cover with $|M|$ vertices can be constructed, it must be a minimum cover. To construct such a cover, let U be the set of unmatched vertices in L (possibly empty), and let Z be the set of vertices that are either in U or are connected to U by alternating paths (paths that alternate between edges that are in the matching and edges that are not in the matching). Let

$$K = (L \setminus Z) \cup (R \cap Z).$$

Every edge e in E either belongs to an alternating path (and has a right endpoint in K), or it has a left endpoint in K. For, if e is matched but not in an alternating path, then its left endpoint cannot be in an alternating path (because two matched edges can not share a vertex) and thus belongs to $L \setminus Z$. Alternatively, if e is unmatched but not in an alternating path, then its left endpoint cannot be in an alternating path, for such a path could be extended by adding e to it. Thus, K forms a vertex cover.

Additionally, every vertex in K is an endpoint of a matched edge. For, every vertex in $L \setminus Z$ is matched because Z is a superset of U, the set of unmatched left vertices. And every vertex in $R \cap Z$ must also be matched, for if there existed an alternating path to an unmatched vertex then changing the matching by removing the matched edges from this path and adding the unmatched edges in their place would increase the size of the matching. However, no matched edge can have both of its endpoints in K. Thus, K is a vertex cover of cardinality equal to M, and must be a minimum vertex cover.

18 Dilworth's theorem

Dilworth's theorem states that, in any finite partially ordered set, the largest antichain has the same size as the smallest chain decomposition. Here, the size of the antichain is its number of elements, and the size of the chain decomposition is its number of chains.

19 3D Transformation

- **Rotation** We can perform 3D rotation about X, Y, and Z axes (**counter-clockwise**). They are represented in the matrix form as below:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & -sin\theta & 0 \\ 0 & sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} cos\theta & 0 & sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -sin\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} cos\theta & -sin\theta & 0 & 0 \\ sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- **Scaling**:

$$S = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- **Shear**

$$Sh = \begin{bmatrix} 1 & sh_x^y & sh_x^z & 0 \\ sh_y^x & 1 & sh_y^z & 0 \\ sh_z^x & sh_z^y & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

20 Matroid intersection

Matroid is a pair $<X, I>$ where X is called ground set and I is set of all independent subsets of X. In other words matroid $<X, I>$ gives a classification for each subset of X to be either independent or dependent (included in I or not included in I).
Of course, we are not speaking about arbitrary classifications. These 3 properties must hold for any matroid:

- Empty set is independent.
- Any subset of independent set is independent.
- If independent set A has smaller size than independent set B, there exist at least one element in B that can be added into A without loss of independency.

Some types of matroid:

- **Uniform matroid**: Matroid that considers subset S independent if size of S is not greater than some constant k ($|S| \leq k$).
- **Linear (algebra) matroid**
- **Colorful matroid**: Set of elements is independent if no pair of included elements share a color
- **Graphic matroid**:This matroid is defined on edges of some undirected graph. Set of edges is independent if it does not contain a cycle
- **Truncated matroid**: We can limit rank of any matroid by some number k without breaking matroid properties
- **Matroid on a subset of ground set**. We can limit ground set of matroid to its subset without breaking matroid properties
- **Expanded matroid. Direct matroid sum.** We can consider two matroids as one big matroid without any difficulties if elements of ground set of first matroid does not affect independence, neither intersect with elements of ground set of second matroid and vise versa. Think of two graphic matroids on two connected graphs. We can unite their graphs together resulting in graph with two connected components, but it is clear that including some edges in one component have no effect on other component. This is called direct matroid sum. Formally, $M_1 = <X_1, I_1>, M_2 = <X_2, I_2>, M_1 + M_2 = <X_1 \bigcup X_2, I_1 \times I_2>$, where $\times$ means cartesian product of two sets. You can unite as many matroids of as many different types without restrictions as you want (if you can find some use for the result).

Matroid intersection solution We are given two matroids $M_1 = <X, I_1>$ and $M_2 = <X, I_2>$ with ranking functions r_1 and r_2 respectively and independence oracles with running times $C1$ and $C2$ respectively. We need to find largest set S that is independent for both matroids.

According to algorithm we need to start with empty S and then repeat the following until we fail to do this:

- Build exchange graph $D_{(M1,M2)}(S)$
- Find "free to include vertices" sets Y_1 and Y_2
- Find **Shortest** augmenting path without shortcuts P from any element in Y_1 to any element in Y_2
- Alternate inclusion into S of all elements in P

We do this at most $O(|S|)$ times.

Exchange graph: Split elements in half: S and X
S. If we exchange $v \in X$
S and $u \in S$, add edge $u \to v$ in matroid $M_1 = <X, I_1>$ and $v \to u$ in matroid $M_2 = <X_2, I_2>$

Theoretical Computer Science Cheat Sheet

Definitions

$f(n) = O(g(n))$	iff $\exists$ positive c, n_0 such that $0 \le f(n) \le cg(n)\ \forall n \ge n_0$.
$f(n) = \Omega(g(n))$	iff $\exists$ positive c, n_0 such that $f(n) \ge cg(n) \ge 0\ \forall n \ge n_0$.
$f(n) = \Theta(g(n))$	iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.
$f(n) = o(g(n))$	iff $\lim_{n\to\infty} f(n)/g(n) = 0$.
$\lim_{n\to\infty} a_n = a$	iff $\forall \epsilon > 0$, $\exists n_0$ such that $\lvert a_n - a\rvert < \epsilon$, $\forall n \ge n_0$.
$\sup S$	least $b \in \mathbb{R}$ such that $b \ge s$, $\forall s \in S$.
$\inf S$	greatest $b \in \mathbb{R}$ such that $b \le s$, $\forall s \in S$.
$\liminf_{n\to\infty} a_n$	$\lim_{n\to\infty} \inf\{a_i \mid i \ge n, i \in \mathbb{N}\}$.
$\limsup_{n\to\infty} a_n$	$\lim_{n\to\infty} \sup\{a_i \mid i \ge n, i \in \mathbb{N}\}$.
$\binom{n}{k}$	Combinations: Size k subsets of a size n set.
$\genfrac{[}{]}{0pt}{}{n}{k}$	Stirling numbers (1st kind): Arrangements of an n element set into k cycles.
$\genfrac{\{}{\}}{0pt}{}{n}{k}$	Stirling numbers (2nd kind): Partitions of an n element set into k non-empty sets.
$\genfrac{\langle}{\rangle}{0pt}{}{n}{k}$	1st order Eulerian numbers: Permutations $\pi_1\pi_2\dots\pi_n$ on $\{1, 2, \dots, n\}$ with k ascents.
$\left\langle\!\!\left\langle{n \atop k}\right\rangle\!\!\right\rangle$	2nd order Eulerian numbers.
C_n	Catalan Numbers: Binary trees with $n+1$ vertices.

Series

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}, \qquad \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}, \qquad \sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4}.$$

In general:

$$\sum_{i=1}^{n} i^m = \frac{1}{m+1}\left[(n+1)^{m+1} - 1 - \sum_{i=1}^{n}\left((i+1)^{m+1} - i^{m+1} - (m+1)i^m\right)\right]$$

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

Geometric series:

$$\sum_{i=0}^{n} c^i = \frac{c^{n+1}-1}{c-1}, \quad c \ne 1, \qquad \sum_{i=0}^{\infty} c^i = \frac{1}{1-c}, \qquad \sum_{i=1}^{\infty} c^i = \frac{c}{1-c}, \quad |c| < 1,$$

$$\sum_{i=0}^{n} ic^i = \frac{nc^{n+2} - (n+1)c^{n+1} + c}{(c-1)^2}, \quad c \ne 1, \qquad \sum_{i=0}^{\infty} ic^i = \frac{c}{(1-c)^2}, \quad |c| < 1.$$

Harmonic series:

$$H_n = \sum_{i=1}^{n}\frac{1}{i}, \qquad \sum_{i=1}^{n} iH_i = \frac{n(n+1)}{2}H_n - \frac{n(n-1)}{4}.$$

$$\sum_{i=1}^{n} H_i = (n+1)H_n - n, \qquad \sum_{i=1}^{n}\binom{i}{m}H_i = \binom{n+1}{m+1}\left(H_{n+1} - \frac{1}{m+1}\right).$$

1. $\displaystyle\binom{n}{k} = \frac{n!}{(n-k)!k!}$, **2.** $\displaystyle\sum_{k=0}^{n}\binom{n}{k} = 2^n$, **3.** $\displaystyle\binom{n}{k} = \binom{n}{n-k}$,

4. $\displaystyle\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$, **5.** $\displaystyle\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$,

6. $\displaystyle\binom{n}{m}\binom{m}{k} = \binom{n}{k}\binom{n-k}{m-k}$, **7.** $\displaystyle\sum_{k=0}^{n}\binom{r+k}{k} = \binom{r+n+1}{n}$,

8. $\displaystyle\sum_{k=0}^{n}\binom{k}{m} = \binom{n+1}{m+1}$, **9.** $\displaystyle\sum_{k=0}^{n}\binom{r}{k}\binom{s}{n-k} = \binom{r+s}{n}$,

10. $\displaystyle\binom{n}{k} = (-1)^k\binom{k-n-1}{k}$, **11.** $\displaystyle\genfrac{\{}{\}}{0pt}{}{n}{1} = \genfrac{\{}{\}}{0pt}{}{n}{n} = 1$,

12. $\displaystyle\genfrac{\{}{\}}{0pt}{}{n}{2} = 2^{n-1} - 1$, **13.** $\displaystyle\genfrac{\{}{\}}{0pt}{}{n}{k} = k\genfrac{\{}{\}}{0pt}{}{n-1}{k} + \genfrac{\{}{\}}{0pt}{}{n-1}{k-1}$,

14. $\displaystyle\genfrac{[}{]}{0pt}{}{n}{1} = (n-1)!$, **15.** $\displaystyle\genfrac{[}{]}{0pt}{}{n}{2} = (n-1)!H_{n-1}$, **16.** $\displaystyle\genfrac{[}{]}{0pt}{}{n}{n} = 1$, **17.** $\displaystyle\genfrac{[}{]}{0pt}{}{n}{k} \ge \genfrac{\{}{\}}{0pt}{}{n}{k}$,

18. $\displaystyle\genfrac{[}{]}{0pt}{}{n}{k} = (n-1)\genfrac{[}{]}{0pt}{}{n-1}{k} + \genfrac{[}{]}{0pt}{}{n-1}{k-1}$, **19.** $\displaystyle\genfrac{\{}{\}}{0pt}{}{n}{n-1} = \genfrac{[}{]}{0pt}{}{n}{n-1} = \binom{n}{2}$, **20.** $\displaystyle\sum_{k=0}^{n}\genfrac{[}{]}{0pt}{}{n}{k} = n!$, **21.** $\displaystyle C_n = \frac{1}{n+1}\binom{2n}{n}$,

22. $\displaystyle\genfrac{\langle}{\rangle}{0pt}{}{n}{0} = \genfrac{\langle}{\rangle}{0pt}{}{n}{n-1} = 1$, **23.** $\displaystyle\genfrac{\langle}{\rangle}{0pt}{}{n}{k} = \genfrac{\langle}{\rangle}{0pt}{}{n}{n-1-k}$, **24.** $\displaystyle\genfrac{\langle}{\rangle}{0pt}{}{n}{k} = (k+1)\genfrac{\langle}{\rangle}{0pt}{}{n-1}{k} + (n-k)\genfrac{\langle}{\rangle}{0pt}{}{n-1}{k-1}$,

25. $\displaystyle\genfrac{\langle}{\rangle}{0pt}{}{0}{k} = \begin{cases} 1 & \text{if } k = 0, \\ 0 & \text{otherwise} \end{cases}$ **26.** $\displaystyle\genfrac{\langle}{\rangle}{0pt}{}{n}{1} = 2^n - n - 1$, **27.** $\displaystyle\genfrac{\langle}{\rangle}{0pt}{}{n}{2} = 3^n - (n+1)2^n + \binom{n+1}{2}$,

28. $\displaystyle x^n = \sum_{k=0}^{n}\genfrac{\langle}{\rangle}{0pt}{}{n}{k}\binom{x+k}{n}$, **29.** $\displaystyle\genfrac{\langle}{\rangle}{0pt}{}{n}{m} = \sum_{k=0}^{m}\binom{n+1}{k}(m+1-k)^n(-1)^k$, **30.** $\displaystyle m!\genfrac{\{}{\}}{0pt}{}{n}{m} = \sum_{k=0}^{n}\genfrac{\langle}{\rangle}{0pt}{}{n}{k}\binom{k}{n-m}$,

31. $\displaystyle\genfrac{\langle}{\rangle}{0pt}{}{n}{m} = \sum_{k=0}^{n}\genfrac{\{}{\}}{0pt}{}{n}{k}\binom{n-k}{m}(-1)^{n-k-m}k!$, **32.** $\displaystyle\left\langle\!\!\left\langle{n \atop 0}\right\rangle\!\!\right\rangle = 1$, **33.** $\displaystyle\left\langle\!\!\left\langle{n \atop n}\right\rangle\!\!\right\rangle = 0$ for $n \ne 0$,

34. $\displaystyle\left\langle\!\!\left\langle{n \atop k}\right\rangle\!\!\right\rangle = (k+1)\left\langle\!\!\left\langle{n-1 \atop k}\right\rangle\!\!\right\rangle + (2n-1-k)\left\langle\!\!\left\langle{n-1 \atop k-1}\right\rangle\!\!\right\rangle$, **35.** $\displaystyle\sum_{k=0}^{n}\left\langle\!\!\left\langle{n \atop k}\right\rangle\!\!\right\rangle = \frac{(2n)^{\underline{n}}}{2^n}$,

36. $\displaystyle\genfrac{\{}{\}}{0pt}{}{x}{x-n} = \sum_{k=0}^{n}\left\langle\!\!\left\langle{n \atop k}\right\rangle\!\!\right\rangle\binom{x+n-1-k}{2n}$, **37.** $\displaystyle\genfrac{\{}{\}}{0pt}{}{n+1}{m+1} = \sum_{k}\binom{n}{k}\genfrac{\{}{\}}{0pt}{}{k}{m} = \sum_{k=0}^{n}\genfrac{\{}{\}}{0pt}{}{k}{m}(m+1)^{n-k}$,

Theoretical Computer Science Cheat Sheet

Identities Cont.

38. $\begin{bmatrix} n+1 \\ m+1 \end{bmatrix} = \sum_k \begin{bmatrix} n \\ k \end{bmatrix} \binom{k}{m} = \sum_{k=0}^{n} \begin{bmatrix} k \\ m \end{bmatrix} n^{\underline{n-k}} = n! \sum_{k=0}^{n} \frac{1}{k!} \begin{bmatrix} k \\ m \end{bmatrix},$

39. $\begin{bmatrix} x \\ x-n \end{bmatrix} = \sum_{k=0}^{n} \left\langle\!\!\left\langle \begin{matrix} n \\ k \end{matrix} \right\rangle\!\!\right\rangle \binom{x+k}{2n},$

40. $\begin{Bmatrix} n \\ m \end{Bmatrix} = \sum_k \binom{n}{k} \begin{Bmatrix} k+1 \\ m+1 \end{Bmatrix} (-1)^{n-k},$

41. $\begin{bmatrix} n \\ m \end{bmatrix} = \sum_k \begin{bmatrix} n+1 \\ k+1 \end{bmatrix} \binom{k}{m} (-1)^{m-k},$

42. $\begin{Bmatrix} m+n+1 \\ m \end{Bmatrix} = \sum_{k=0}^{m} k \begin{Bmatrix} n+k \\ k \end{Bmatrix},$

43. $\begin{bmatrix} m+n+1 \\ m \end{bmatrix} = \sum_{k=0}^{m} k(n+k) \begin{bmatrix} n+k \\ k \end{bmatrix},$

44. $\binom{n}{m} = \sum_k \begin{Bmatrix} n+1 \\ k+1 \end{Bmatrix} \begin{bmatrix} k \\ m \end{bmatrix} (-1)^{m-k},$

45. $(n-m)! \binom{n}{m} = \sum_k \begin{bmatrix} n+1 \\ k+1 \end{bmatrix} \begin{Bmatrix} k \\ m \end{Bmatrix} (-1)^{m-k}, \quad$ for $n \geq m$,

46. $\begin{Bmatrix} n \\ n-m \end{Bmatrix} = \sum_k \binom{m-n}{m+k} \binom{m+n}{n+k} \begin{bmatrix} m+k \\ k \end{bmatrix},$

47. $\begin{bmatrix} n \\ n-m \end{bmatrix} = \sum_k \binom{m-n}{m+k} \binom{m+n}{n+k} \begin{Bmatrix} m+k \\ k \end{Bmatrix},$

48. $\begin{Bmatrix} n \\ \ell+m \end{Bmatrix} \binom{\ell+m}{\ell} = \sum_k \begin{Bmatrix} k \\ \ell \end{Bmatrix} \begin{Bmatrix} n-k \\ m \end{Bmatrix} \binom{n}{k},$

49. $\begin{bmatrix} n \\ \ell+m \end{bmatrix} \binom{\ell+m}{\ell} = \sum_k \begin{bmatrix} k \\ \ell \end{bmatrix} \begin{bmatrix} n-k \\ m \end{bmatrix} \binom{n}{k}.$

Trees

Every tree with n vertices has $n-1$ edges.

Kraft inequality: If the depths of the leaves of a binary tree are $d_1, \ldots, d_n$:

$$\sum_{i=1}^{n} 2^{-d_i} \leq 1,$$

and equality holds only if every internal node has 2 sons.

Recurrences

Master method:

$$T(n) = aT(n/b) + f(n), \quad a \geq 1, b > 1$$

If $\exists \epsilon > 0$ such that $f(n) = O(n^{\log_b a - \epsilon})$ then

$$T(n) = \Theta(n^{\log_b a}).$$

If $f(n) = \Theta(n^{\log_b a})$ then

$$T(n) = \Theta(n^{\log_b a} \log_2 n).$$

If $\exists \epsilon > 0$ such that $f(n) = \Omega(n^{\log_b a + \epsilon})$, and $\exists c < 1$ such that $af(n/b) \leq cf(n)$ for large n, then

$$T(n) = \Theta(f(n)).$$

Substitution (example): Consider the following recurrence

$$T_{i+1} = 2^{2^i} \cdot T_i^2, \quad T_1 = 2.$$

Note that T_i is always a power of two. Let $t_i = \log_2 T_i$. Then we have

$$t_{i+1} = 2^i + 2t_i, \quad t_1 = 1.$$

Let $u_i = t_i/2^i$. Dividing both sides of the previous equation by 2^{i+1} we get

$$\frac{t_{i+1}}{2^{i+1}} = \frac{2^i}{2^{i+1}} + \frac{t_i}{2^i}.$$

Substituting we find

$$u_{i+1} = \tfrac{1}{2} + u_i, \qquad u_1 = \tfrac{1}{2},$$

which is simply $u_i = i/2$. So we find that T_i has the closed form $T_i = 2^{i2^{i-1}}$. Summing factors (example): Consider the following recurrence

$$T(n) = 3T(n/2) + n, \quad T(1) = 1.$$

Rewrite so that all terms involving T are on the left side

$$T(n) - 3T(n/2) = n.$$

Now expand the recurrence, and choose a factor which makes the left side "telescope"

$$1\bigl(T(n) - 3T(n/2) = n\bigr)$$
$$3\bigl(T(n/2) - 3T(n/4) = n/2\bigr)$$
$$\vdots \qquad \vdots \qquad \vdots$$
$$3^{\log_2 n - 1}\bigl(T(2) - 3T(1) = 2\bigr)$$

Let $m = \log_2 n$. Summing the left side we get $T(n) - 3^m T(1) = T(n) - 3^m = T(n) - n^k$ where $k = \log_2 3 \approx 1.58496$. Summing the right side we get

$$\sum_{i=0}^{m-1} \frac{n}{2^i} 3^i = n \sum_{i=0}^{m-1} \left(\tfrac{3}{2}\right)^i.$$

Let $c = \frac{3}{2}$. Then we have

$$\begin{aligned} n \sum_{i=0}^{m-1} c^i &= n \left(\frac{c^m - 1}{c - 1} \right) \\ &= 2n(c^{\log_2 n} - 1) \\ &= 2n(c^{(k-1)\log_c n} - 1) \\ &= 2n^k - 2n, \end{aligned}$$

and so $T(n) = 3n^k - 2n$. Full history recurrences can often be changed to limited history ones (example): Consider

$$T_i = 1 + \sum_{j=0}^{i-1} T_j, \quad T_0 = 1.$$

Note that

$$T_{i+1} = 1 + \sum_{j=0}^{i} T_j.$$

Subtracting we find

$$\begin{aligned} T_{i+1} - T_i &= 1 + \sum_{j=0}^{i} T_j - 1 - \sum_{j=0}^{i-1} T_j \\ &= T_i. \end{aligned}$$

And so $T_{i+1} = 2T_i = 2^{i+1}$.

Generating functions:

1. Multiply both sides of the equation by x^i.
2. Sum both sides over all i for which the equation is valid.
3. Choose a generating function $G(x)$. Usually $G(x) = \sum_{i=0}^{\infty} x^i g_i$.
3. Rewrite the equation in terms of the generating function $G(x)$.
4. Solve for $G(x)$.
5. The coefficient of x^i in $G(x)$ is g_i.

Example:

$$g_{i+1} = 2g_i + 1, \quad g_0 = 0.$$

Multiply and sum:

$$\sum_{i \geq 0} g_{i+1} x^i = \sum_{i \geq 0} 2g_i x^i + \sum_{i \geq 0} x^i.$$

We choose $G(x) = \sum_{i \geq 0} x^i g_i$. Rewrite in terms of $G(x)$:

$$\frac{G(x) - g_0}{x} = 2G(x) + \sum_{i \geq 0} x^i.$$

Simplify:

$$\frac{G(x)}{x} = 2G(x) + \frac{1}{1-x}.$$

Solve for $G(x)$:

$$G(x) = \frac{x}{(1-x)(1-2x)}.$$

Expand this using partial fractions:

$$\begin{aligned} G(x) &= x \left(\frac{2}{1-2x} - \frac{1}{1-x} \right) \\ &= x \left(2 \sum_{i \geq 0} 2^i x^i - \sum_{i \geq 0} x^i \right) \\ &= \sum_{i \geq 0} (2^{i+1} - 1) x^{i+1}. \end{aligned}$$

So $g_i = 2^i - 1$.

Theoretical Computer Science Cheat Sheet

$\pi \approx 3.14159$, $e \approx 2.71828$, $\gamma \approx 0.57721$, $\phi = \frac{1+\sqrt{5}}{2} \approx 1.61803$, $\hat{\phi} = \frac{1-\sqrt{5}}{2} \approx -.61803$

i	2^i	p_i
1	2	2
2	4	3
3	8	5
4	16	7
5	32	11
6	64	13
7	128	17
8	256	19
9	512	23
10	1,024	29
11	2,048	31
12	4,096	37
13	8,192	41
14	16,384	43
15	32,768	47
16	65,536	53
17	131,072	59
18	262,144	61
19	524,288	67
20	1,048,576	71
21	2,097,152	73
22	4,194,304	79
23	8,388,608	83
24	16,777,216	89
25	33,554,432	97
26	67,108,864	101
27	134,217,728	103
28	268,435,456	107
29	536,870,912	109
30	1,073,741,824	113
31	2,147,483,648	127
32	4,294,967,296	131

Pascal's Triangle

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1

General

Bernoulli Numbers ($B_i = 0$, odd $i \neq 1$):

$$B_0 = 1,\ B_1 = -\tfrac{1}{2},\ B_2 = \tfrac{1}{6},\ B_4 = -\tfrac{1}{30},$$
$$B_6 = \tfrac{1}{42},\ B_8 = -\tfrac{1}{30},\ B_{10} = \tfrac{5}{66}.$$

Change of base, quadratic formula:

$$\log_b x = \frac{\log_a x}{\log_a b}, \qquad \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Euler's number e:

$$e = 1 + \tfrac{1}{2} + \tfrac{1}{6} + \tfrac{1}{24} + \tfrac{1}{120} + \cdots$$
$$\lim_{n\to\infty}\left(1 + \frac{x}{n}\right)^n = e^x.$$
$$\left(1+\tfrac{1}{n}\right)^n < e < \left(1+\tfrac{1}{n}\right)^{n+1}.$$
$$\left(1+\tfrac{1}{n}\right)^n = e - \frac{e}{2n} + \frac{11e}{24n^2} - O\left(\frac{1}{n^3}\right).$$

Harmonic numbers:

$$1,\ \tfrac{3}{2},\ \tfrac{11}{6},\ \tfrac{25}{12},\ \tfrac{137}{60},\ \tfrac{49}{20},\ \tfrac{363}{140},\ \tfrac{761}{280},\ \tfrac{7129}{2520}, \ldots$$

$$\ln n < H_n < \ln n + 1,$$
$$H_n = \ln n + \gamma + O\left(\frac{1}{n}\right).$$

Factorial, Stirling's approximation:

1, 2, 6, 24, 120, 720, 5040, 40320, 362880, ...

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1 + \Theta\left(\frac{1}{n}\right)\right).$$

Ackermann's function and inverse:

$$a(i,j) = \begin{cases} 2^j & i = 1 \\ a(i-1, 2) & j = 1 \\ a(i-1, a(i, j-1)) & i, j \geq 2 \end{cases}$$
$$\alpha(i) = \min\{j \mid a(j,j) \geq i\}.$$

Binomial distribution:

$$\Pr[X = k] = \binom{n}{k} p^k q^{n-k}, \qquad q = 1 - p,$$
$$\mathrm{E}[X] = \sum_{k=1}^{n} k \binom{n}{k} p^k q^{n-k} = np.$$

Poisson distribution:

$$\Pr[X = k] = \frac{e^{-\lambda}\lambda^k}{k!}, \quad \mathrm{E}[X] = \lambda.$$

Normal (Gaussian) distribution:

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2}, \quad \mathrm{E}[X] = \mu.$$

The "coupon collector": We are given a random coupon each day, and there are n different types of coupons. The distribution of coupons is uniform. The expected number of days to pass before we to collect all n types is

$$nH_n.$$

Probability

Continuous distributions: If

$$\Pr[a < X < b] = \int_a^b p(x)\,dx,$$

then p is the probability density function of X. If

$$\Pr[X < a] = P(a),$$

then P is the distribution function of X. If P and p both exist then

$$P(a) = \int_{-\infty}^{a} p(x)\,dx.$$

Expectation: If X is discrete

$$\mathrm{E}[g(X)] = \sum_x g(x)\Pr[X = x].$$

If X continuous then

$$\mathrm{E}[g(X)] = \int_{-\infty}^{\infty} g(x)p(x)\,dx = \int_{-\infty}^{\infty} g(x)\,dP(x).$$

Variance, standard deviation:

$$\mathrm{VAR}[X] = \mathrm{E}[X^2] - \mathrm{E}[X]^2,$$
$$\sigma = \sqrt{\mathrm{VAR}[X]}.$$

For events A and B:

$$\Pr[A \vee B] = \Pr[A] + \Pr[B] - \Pr[A \wedge B]$$
$$\Pr[A \wedge B] = \Pr[A]\cdot\Pr[B],$$

iff A and B are independent.

$$\Pr[A|B] = \frac{\Pr[A \wedge B]}{\Pr[B]}$$

For random variables X and Y:

$$\mathrm{E}[X \cdot Y] = \mathrm{E}[X]\cdot\mathrm{E}[Y],$$

if X and Y are independent.

$$\mathrm{E}[X + Y] = \mathrm{E}[X] + \mathrm{E}[Y],$$
$$\mathrm{E}[cX] = c\,\mathrm{E}[X].$$

Bayes' theorem:

$$\Pr[A_i|B] = \frac{\Pr[B|A_i]\Pr[A_i]}{\sum_{j=1}^{n}\Pr[A_j]\Pr[B|A_j]}.$$

Inclusion-exclusion:

$$\Pr\left[\bigvee_{i=1}^{n} X_i\right] = \sum_{i=1}^{n}\Pr[X_i] + \sum_{k=2}^{n}(-1)^{k+1}\sum_{i_i < \cdots < i_k}\Pr\left[\bigwedge_{j=1}^{k} X_{i_j}\right].$$

Moment inequalities:

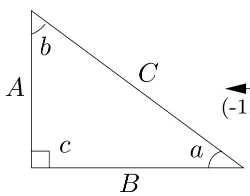
$$\Pr\left[|X| \geq \lambda\,\mathrm{E}[X]\right] \leq \frac{1}{\lambda},$$
$$\Pr\left[\left|X - \mathrm{E}[X]\right| \geq \lambda\cdot\sigma\right] \leq \frac{1}{\lambda^2}.$$

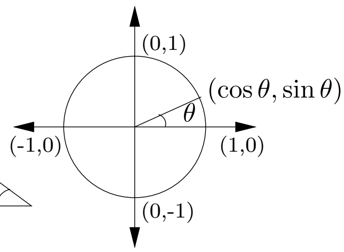
Geometric distribution:

$$\Pr[X = k] = pq^{k-1}, \qquad q = 1 - p,$$
$$\mathrm{E}[X] = \sum_{k=1}^{\infty} kpq^{k-1} = \frac{1}{p}.$$

Theoretical Computer Science Cheat Sheet

Trigonometry





Pythagorean theorem:

$$C^2 = A^2 + B^2.$$

Definitions:

$$\sin a = A/C, \quad \cos a = B/C,$$
$$\csc a = C/A, \quad \sec a = C/B,$$
$$\tan a = \frac{\sin a}{\cos a} = \frac{A}{B}, \quad \cot a = \frac{\cos a}{\sin a} = \frac{B}{A}.$$

Area, radius of inscribed circle:

$$\tfrac{1}{2}AB, \quad \frac{AB}{A+B+C}.$$

Identities:

$$\sin x = \frac{1}{\csc x}, \qquad \cos x = \frac{1}{\sec x},$$
$$\tan x = \frac{1}{\cot x}, \qquad \sin^2 x + \cos^2 x = 1,$$
$$1 + \tan^2 x = \sec^2 x, \qquad 1 + \cot^2 x = \csc^2 x,$$
$$\sin x = \cos\left(\tfrac{\pi}{2} - x\right), \qquad \sin x = \sin(\pi - x),$$
$$\cos x = -\cos(\pi - x), \qquad \tan x = \cot\left(\tfrac{\pi}{2} - x\right),$$
$$\cot x = -\cot(\pi - x), \qquad \csc x = \cot\tfrac{x}{2} - \cot x,$$
$$\sin(x \pm y) = \sin x \cos y \pm \cos x \sin y,$$
$$\cos(x \pm y) = \cos x \cos y \mp \sin x \sin y,$$
$$\tan(x \pm y) = \frac{\tan x \pm \tan y}{1 \mp \tan x \tan y},$$
$$\cot(x \pm y) = \frac{\cot x \cot y \mp 1}{\cot x \pm \cot y},$$
$$\sin 2x = 2 \sin x \cos x, \qquad \sin 2x = \frac{2\tan x}{1 + \tan^2 x},$$
$$\cos 2x = \cos^2 x - \sin^2 x, \qquad \cos 2x = 2\cos^2 x - 1,$$
$$\cos 2x = 1 - 2\sin^2 x, \qquad \cos 2x = \frac{1 - \tan^2 x}{1 + \tan^2 x},$$
$$\tan 2x = \frac{2\tan x}{1 - \tan^2 x}, \qquad \cot 2x = \frac{\cot^2 x - 1}{2\cot x},$$
$$\sin(x+y)\sin(x-y) = \sin^2 x - \sin^2 y,$$
$$\cos(x+y)\cos(x-y) = \cos^2 x - \sin^2 y.$$

Euler's equation:

$$e^{ix} = \cos x + i \sin x, \qquad e^{i\pi} = -1.$$

Matrices

Multiplication:

$$C = A \cdot B, \quad c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}.$$

Determinants: $\det A \neq 0$ iff A is non-singular.

$$\det A \cdot B = \det A \cdot \det B,$$
$$\det A = \sum_{\pi} \prod_{i=1}^{n} \operatorname{sign}(\pi) a_{i,\pi(i)}.$$

2×2 and 3×3 determinant:

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc,$$
$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = g \begin{vmatrix} b & c \\ e & f \end{vmatrix} - h \begin{vmatrix} a & c \\ d & f \end{vmatrix} + i \begin{vmatrix} a & b \\ d & e \end{vmatrix}$$
$$= \begin{matrix} aei + bfg + cdh \\ -ceg - fha - ibd. \end{matrix}$$

Permanents:

$$\operatorname{perm} A = \sum_{\pi} \prod_{i=1}^{n} a_{i,\pi(i)}.$$

Hyperbolic Functions

Definitions:

$$\sinh x = \frac{e^x - e^{-x}}{2}, \qquad \cosh x = \frac{e^x + e^{-x}}{2},$$
$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \qquad \operatorname{csch} x = \frac{1}{\sinh x},$$
$$\operatorname{sech} x = \frac{1}{\cosh x}, \qquad \coth x = \frac{1}{\tanh x}.$$

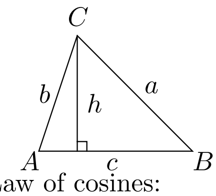
Identities:

$$\cosh^2 x - \sinh^2 x = 1, \qquad \tanh^2 x + \operatorname{sech}^2 x = 1,$$
$$\coth^2 x - \operatorname{csch}^2 x = 1, \qquad \sinh(-x) = -\sinh x,$$
$$\cosh(-x) = \cosh x, \qquad \tanh(-x) = -\tanh x,$$
$$\sinh(x+y) = \sinh x \cosh y + \cosh x \sinh y,$$
$$\cosh(x+y) = \cosh x \cosh y + \sinh x \sinh y,$$
$$\sinh 2x = 2 \sinh x \cosh x,$$
$$\cosh 2x = \cosh^2 x + \sinh^2 x,$$
$$\cosh x + \sinh x = e^x, \qquad \cosh x - \sinh x = e^{-x},$$
$$(\cosh x + \sinh x)^n = \cosh nx + \sinh nx, \quad n \in \mathbb{Z},$$
$$2\sinh^2 \tfrac{x}{2} = \cosh x - 1, \qquad 2\cosh^2 \tfrac{x}{2} = \cosh x + 1.$$

θ	$\sin\theta$	$\cos\theta$	$\tan\theta$
0	0	1	0
$\frac{\pi}{6}$	$\frac{1}{2}$	$\frac{\sqrt{3}}{2}$	$\frac{\sqrt{3}}{3}$
$\frac{\pi}{4}$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{2}}{2}$	1
$\frac{\pi}{3}$	$\frac{\sqrt{3}}{2}$	$\frac{1}{2}$	$\sqrt{3}$
$\frac{\pi}{2}$	1	0	∞

. . . in mathematics you don't understand things, you just get used to them.
– J. von Neumann

More Trig.



Law of cosines:

$$c^2 = a^2 + b^2 - 2ab\cos C.$$

Area:

$$A = \tfrac{1}{2}hc,$$
$$= \tfrac{1}{2}ab\sin C,$$
$$= \frac{c^2 \sin A \sin B}{2 \sin C}.$$

Heron's formula:

$$A = \sqrt{s \cdot s_a \cdot s_b \cdot s_c},$$
$$s = \tfrac{1}{2}(a + b + c),$$
$$s_a = s - a,$$
$$s_b = s - b,$$
$$s_c = s - c.$$

More identities:

$$\sin\tfrac{x}{2} = \sqrt{\frac{1 - \cos x}{2}},$$
$$\cos\tfrac{x}{2} = \sqrt{\frac{1 + \cos x}{2}},$$
$$\tan\tfrac{x}{2} = \sqrt{\frac{1 - \cos x}{1 + \cos x}},$$
$$= \frac{1 - \cos x}{\sin x},$$
$$= \frac{\sin x}{1 + \cos x},$$
$$\cot\tfrac{x}{2} = \sqrt{\frac{1 + \cos x}{1 - \cos x}},$$
$$= \frac{1 + \cos x}{\sin x},$$
$$= \frac{\sin x}{1 - \cos x},$$
$$\sin x = \frac{e^{ix} - e^{-ix}}{2i},$$
$$\cos x = \frac{e^{ix} + e^{-ix}}{2},$$
$$\tan x = -i\frac{e^{ix} - e^{-ix}}{e^{ix} + e^{-ix}},$$
$$= -i\frac{e^{2ix} - 1}{e^{2ix} + 1},$$
$$\sin x = \frac{\sinh ix}{i},$$
$$\cos x = \cosh ix,$$
$$\tan x = \frac{\tanh ix}{i}.$$

v2.02 ©1994 by Steve Seiden
sseiden@acm.org
http://www.csc.lsu.edu/~seiden

Theoretical Computer Science Cheat Sheet

Number Theory

The Chinese remainder theorem: There exists a number C such that:

$$\begin{aligned} C &\equiv r_1 \bmod m_1 \\ \vdots \quad &\vdots \quad \vdots \\ C &\equiv r_n \bmod m_n \end{aligned}$$

if m_i and m_j are relatively prime for $i \neq j$.

Euler's function: $\phi(x)$ is the number of positive integers less than x relatively prime to x. If $\prod_{i=1}^n p_i^{e_i}$ is the prime factorization of x then

$$\phi(x) = \prod_{i=1}^n p_i^{e_i-1}(p_i - 1).$$

Euler's theorem: If a and b are relatively prime then

$$1 \equiv a^{\phi(b)} \bmod b.$$

Fermat's theorem:

$$1 \equiv a^{p-1} \bmod p.$$

The Euclidean algorithm: if $a > b$ are integers then

$$\gcd(a, b) = \gcd(a \bmod b, b).$$

If $\prod_{i=1}^n p_i^{e_i}$ is the prime factorization of x then

$$S(x) = \sum_{d|x} d = \prod_{i=1}^n \frac{p_i^{e_i+1} - 1}{p_i - 1}.$$

Perfect Numbers: x is an even perfect number iff $x = 2^{n-1}(2^n - 1)$ and $2^n - 1$ is prime.

Wilson's theorem: n is a prime iff

$$(n-1)! \equiv -1 \bmod n.$$

Möbius inversion:

$$\mu(i) = \begin{cases} 1 & \text{if } i = 1. \\ 0 & \text{if } i \text{ is not square-free.} \\ (-1)^r & \text{if } i \text{ is the product of} \\ & r \text{ distinct primes.} \end{cases}$$

If

$$G(a) = \sum_{d|a} F(d),$$

then

$$F(a) = \sum_{d|a} \mu(d) G\Big(\frac{a}{d}\Big).$$

Prime numbers:

$$p_n = n \ln n + n \ln\ln n - n + n \frac{\ln\ln n}{\ln n} + O\Big(\frac{n}{\ln n}\Big),$$

$$\pi(n) = \frac{n}{\ln n} + \frac{n}{(\ln n)^2} + \frac{2!n}{(\ln n)^3} + O\Big(\frac{n}{(\ln n)^4}\Big).$$

Graph Theory

Definitions:

Loop	An edge connecting a vertex to itself.
Directed	Each edge has a direction.
Simple	Graph with no loops or multi-edges.
Walk	A sequence $v_0 e_1 v_1 \dots e_\ell v_\ell$.
Trail	A walk with distinct edges.
Path	A trail with distinct vertices.
Connected	A graph where there exists a path between any two vertices.
Component	A maximal connected subgraph.
Tree	A connected acyclic graph.
Free tree	A tree with no root.
DAG	Directed acyclic graph.
Eulerian	Graph with a trail visiting each edge exactly once.
Hamiltonian	Graph with a cycle visiting each vertex exactly once.
Cut	A set of edges whose removal increases the number of components.
Cut-set	A minimal cut.
Cut edge	A size 1 cut.
k-Connected	A graph connected with the removal of any $k-1$ vertices.
k-Tough	$\forall S \subseteq V, S \neq \emptyset$ we have $k \cdot c(G - S) \leq \|S\|$.
k-Regular	A graph where all vertices have degree k.
k-Factor	A k-regular spanning subgraph.
Matching	A set of edges, no two of which are adjacent.
Clique	A set of vertices, all of which are adjacent.
Ind. set	A set of vertices, none of which are adjacent.
Vertex cover	A set of vertices which cover all edges.
Planar graph	A graph which can be embeded in the plane.
Plane graph	An embedding of a planar graph.

$$\sum_{v \in V} \deg(v) = 2m.$$

If G is planar then $n - m + f = 2$, so

$$f \leq 2n - 4, \quad m \leq 3n - 6.$$

Any planar graph has a vertex with degree ≤ 5.

Notation:

$E(G)$	Edge set
$V(G)$	Vertex set
$c(G)$	Number of components
$G[S]$	Induced subgraph
$\deg(v)$	Degree of v
$\Delta(G)$	Maximum degree
$\delta(G)$	Minimum degree
$\chi(G)$	Chromatic number
$\chi_E(G)$	Edge chromatic number
G^c	Complement graph
K_n	Complete graph
K_{n_1,n_2}	Complete bipartite graph
$\mathrm{r}(k,\ell)$	Ramsey number

Geometry

Projective coordinates: triples (x, y, z), not all x, y and z zero.

$$(x, y, z) = (cx, cy, cz) \quad \forall c \neq 0.$$

Cartesian	Projective
(x, y)	$(x, y, 1)$
$y = mx + b$	$(m, -1, b)$
$x = c$	$(1, 0, -c)$

Distance formula, L_p and L_∞ metric:

$$\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2},$$

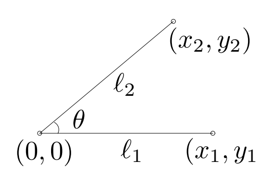
$$\big[|x_1 - x_0|^p + |y_1 - y_0|^p\big]^{1/p},$$

$$\lim_{p \to \infty} \big[|x_1 - x_0|^p + |y_1 - y_0|^p\big]^{1/p}.$$

Area of triangle (x_0, y_0), (x_1, y_1) and (x_2, y_2):

$$\tfrac{1}{2} \operatorname{abs} \begin{vmatrix} x_1 - x_0 & y_1 - y_0 \\ x_2 - x_0 & y_2 - y_0 \end{vmatrix}.$$

Angle formed by three points:



$$\cos\theta = \frac{(x_1, y_1) \cdot (x_2, y_2)}{\ell_1 \ell_2}.$$

Line through two points (x_0, y_0) and (x_1, y_1):

$$\begin{vmatrix} x & y & 1 \\ x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \end{vmatrix} = 0.$$

Area of circle, volume of sphere:

$$A = \pi r^2, \qquad V = \tfrac{4}{3}\pi r^3.$$

If I have seen farther than others, it is because I have stood on the shoulders of giants.
– Issac Newton

Theoretical Computer Science Cheat Sheet

π

Wallis' identity:

$$\pi = 2 \cdot \frac{2 \cdot 2 \cdot 4 \cdot 4 \cdot 6 \cdot 6 \cdots}{1 \cdot 3 \cdot 3 \cdot 5 \cdot 5 \cdot 7 \cdots}$$

Brouncker's continued fraction expansion:

$$\tfrac{\pi}{4} = 1 + \cfrac{1^2}{2 + \cfrac{3^2}{2 + \cfrac{5^2}{2 + \cfrac{7^2}{2 + \cdots}}}}$$

Gregrory's series:

$$\tfrac{\pi}{4} = 1 - \tfrac{1}{3} + \tfrac{1}{5} - \tfrac{1}{7} + \tfrac{1}{9} - \cdots$$

Newton's series:

$$\tfrac{\pi}{6} = \frac{1}{2} + \frac{1}{2 \cdot 3 \cdot 2^3} + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5 \cdot 2^5} + \cdots$$

Sharp's series:

$$\tfrac{\pi}{6} = \frac{1}{\sqrt{3}}\left(1 - \frac{1}{3^1 \cdot 3} + \frac{1}{3^2 \cdot 5} - \frac{1}{3^3 \cdot 7} + \cdots\right)$$

Euler's series:

$$\tfrac{\pi^2}{6} = \tfrac{1}{1^2} + \tfrac{1}{2^2} + \tfrac{1}{3^2} + \tfrac{1}{4^2} + \tfrac{1}{5^2} + \cdots$$
$$\tfrac{\pi^2}{8} = \tfrac{1}{1^2} + \tfrac{1}{3^2} + \tfrac{1}{5^2} + \tfrac{1}{7^2} + \tfrac{1}{9^2} + \cdots$$
$$\tfrac{\pi^2}{12} = \tfrac{1}{1^2} - \tfrac{1}{2^2} + \tfrac{1}{3^2} - \tfrac{1}{4^2} + \tfrac{1}{5^2} - \cdots$$

Partial Fractions

Let $N(x)$ and $D(x)$ be polynomial functions of x. We can break down $N(x)/D(x)$ using partial fraction expansion. First, if the degree of N is greater than or equal to the degree of D, divide N by D, obtaining

$$\frac{N(x)}{D(x)} = Q(x) + \frac{N'(x)}{D(x)},$$

where the degree of N' is less than that of D. Second, factor $D(x)$. Use the following rules: For a non-repeated factor:

$$\frac{N(x)}{(x-a)D(x)} = \frac{A}{x-a} + \frac{N'(x)}{D(x)},$$

where

$$A = \left[\frac{N(x)}{D(x)}\right]_{x=a}.$$

For a repeated factor:

$$\frac{N(x)}{(x-a)^m D(x)} = \sum_{k=0}^{m-1} \frac{A_k}{(x-a)^{m-k}} + \frac{N'(x)}{D(x)},$$

where

$$A_k = \frac{1}{k!}\left[\frac{d^k}{dx^k}\left(\frac{N(x)}{D(x)}\right)\right]_{x=a}.$$

The reasonable man adapts himself to the world; the unreasonable persists in trying to adapt the world to himself. Therefore all progress depends on the unreasonable.
– George Bernard Shaw

Calculus

Derivatives:

1. $\dfrac{d(cu)}{dx} = c\dfrac{du}{dx}$, **2.** $\dfrac{d(u+v)}{dx} = \dfrac{du}{dx} + \dfrac{dv}{dx}$, **3.** $\dfrac{d(uv)}{dx} = u\dfrac{dv}{dx} + v\dfrac{du}{dx}$,

4. $\dfrac{d(u^n)}{dx} = nu^{n-1}\dfrac{du}{dx}$, **5.** $\dfrac{d(u/v)}{dx} = \dfrac{v\left(\frac{du}{dx}\right) - u\left(\frac{dv}{dx}\right)}{v^2}$, **6.** $\dfrac{d(e^{cu})}{dx} = ce^{cu}\dfrac{du}{dx}$,

7. $\dfrac{d(c^u)}{dx} = (\ln c)c^u\dfrac{du}{dx}$, **8.** $\dfrac{d(\ln u)}{dx} = \dfrac{1}{u}\dfrac{du}{dx}$,

9. $\dfrac{d(\sin u)}{dx} = \cos u\dfrac{du}{dx}$, **10.** $\dfrac{d(\cos u)}{dx} = -\sin u\dfrac{du}{dx}$,

11. $\dfrac{d(\tan u)}{dx} = \sec^2 u\dfrac{du}{dx}$, **12.** $\dfrac{d(\cot u)}{dx} = \csc^2 u\dfrac{du}{dx}$,

13. $\dfrac{d(\sec u)}{dx} = \tan u \sec u\dfrac{du}{dx}$, **14.** $\dfrac{d(\csc u)}{dx} = -\cot u \csc u\dfrac{du}{dx}$,

15. $\dfrac{d(\arcsin u)}{dx} = \dfrac{1}{\sqrt{1-u^2}}\dfrac{du}{dx}$, **16.** $\dfrac{d(\arccos u)}{dx} = \dfrac{-1}{\sqrt{1-u^2}}\dfrac{du}{dx}$,

17. $\dfrac{d(\arctan u)}{dx} = \dfrac{1}{1+u^2}\dfrac{du}{dx}$, **18.** $\dfrac{d(\operatorname{arccot} u)}{dx} = \dfrac{-1}{1+u^2}\dfrac{du}{dx}$,

19. $\dfrac{d(\operatorname{arcsec} u)}{dx} = \dfrac{1}{u\sqrt{1-u^2}}\dfrac{du}{dx}$, **20.** $\dfrac{d(\operatorname{arccsc} u)}{dx} = \dfrac{-1}{u\sqrt{1-u^2}}\dfrac{du}{dx}$,

21. $\dfrac{d(\sinh u)}{dx} = \cosh u\dfrac{du}{dx}$, **22.** $\dfrac{d(\cosh u)}{dx} = \sinh u\dfrac{du}{dx}$,

23. $\dfrac{d(\tanh u)}{dx} = \operatorname{sech}^2 u\dfrac{du}{dx}$, **24.** $\dfrac{d(\coth u)}{dx} = -\operatorname{csch}^2 u\dfrac{du}{dx}$,

25. $\dfrac{d(\operatorname{sech} u)}{dx} = -\operatorname{sech} u \tanh u\dfrac{du}{dx}$, **26.** $\dfrac{d(\operatorname{csch} u)}{dx} = -\operatorname{csch} u \coth u\dfrac{du}{dx}$,

27. $\dfrac{d(\operatorname{arcsinh} u)}{dx} = \dfrac{1}{\sqrt{1+u^2}}\dfrac{du}{dx}$, **28.** $\dfrac{d(\operatorname{arccosh} u)}{dx} = \dfrac{1}{\sqrt{u^2-1}}\dfrac{du}{dx}$,

29. $\dfrac{d(\operatorname{arctanh} u)}{dx} = \dfrac{1}{1-u^2}\dfrac{du}{dx}$, **30.** $\dfrac{d(\operatorname{arccoth} u)}{dx} = \dfrac{1}{u^2-1}\dfrac{du}{dx}$,

31. $\dfrac{d(\operatorname{arcsech} u)}{dx} = \dfrac{-1}{u\sqrt{1-u^2}}\dfrac{du}{dx}$, **32.** $\dfrac{d(\operatorname{arccsch} u)}{dx} = \dfrac{-1}{|u|\sqrt{1+u^2}}\dfrac{du}{dx}$.

Integrals:

1. $\displaystyle\int cu\,dx = c\int u\,dx$, **2.** $\displaystyle\int (u+v)\,dx = \int u\,dx + \int v\,dx$,

3. $\displaystyle\int x^n\,dx = \frac{1}{n+1}x^{n+1}, \quad n \neq -1$, **4.** $\displaystyle\int \frac{1}{x}dx = \ln x$, **5.** $\displaystyle\int e^x\,dx = e^x$,

6. $\displaystyle\int \frac{dx}{1+x^2} = \arctan x$, **7.** $\displaystyle\int u\frac{dv}{dx}dx = uv - \int v\frac{du}{dx}dx$,

8. $\displaystyle\int \sin x\,dx = -\cos x$, **9.** $\displaystyle\int \cos x\,dx = \sin x$,

10. $\displaystyle\int \tan x\,dx = -\ln|\cos x|$, **11.** $\displaystyle\int \cot x\,dx = \ln|\cos x|$,

12. $\displaystyle\int \sec x\,dx = \ln|\sec x + \tan x|$, **13.** $\displaystyle\int \csc x\,dx = \ln|\csc x + \cot x|$,

14. $\displaystyle\int \arcsin\tfrac{x}{a}dx = \arcsin\tfrac{x}{a} + \sqrt{a^2-x^2}, \quad a > 0$,

Calculus Cont.

15. $\int \arccos\frac{x}{a}dx = \arccos\frac{x}{a} - \sqrt{a^2-x^2}, \quad a>0,$

16. $\int \arctan\frac{x}{a}dx = x\arctan\frac{x}{a} - \frac{a}{2}\ln(a^2+x^2), \quad a>0,$

17. $\int \sin^2(ax)dx = \frac{1}{2a}\big(ax - \sin(ax)\cos(ax)\big),$

18. $\int \cos^2(ax)dx = \frac{1}{2a}\big(ax + \sin(ax)\cos(ax)\big),$

19. $\int \sec^2 x\,dx = \tan x,$

20. $\int \csc^2 x\,dx = -\cot x,$

21. $\int \sin^n x\,dx = -\frac{\sin^{n-1}x\cos x}{n} + \frac{n-1}{n}\int \sin^{n-2}x\,dx,$

22. $\int \cos^n x\,dx = \frac{\cos^{n-1}x\sin x}{n} + \frac{n-1}{n}\int \cos^{n-2}x\,dx,$

23. $\int \tan^n x\,dx = \frac{\tan^{n-1}x}{n-1} - \int \tan^{n-2}x\,dx, \quad n\neq 1,$

24. $\int \cot^n x\,dx = -\frac{\cot^{n-1}x}{n-1} - \int \cot^{n-2}x\,dx, \quad n\neq 1,$

25. $\int \sec^n x\,dx = \frac{\tan x\sec^{n-1}x}{n-1} + \frac{n-2}{n-1}\int \sec^{n-2}x\,dx, \quad n\neq 1,$

26. $\int \csc^n x\,dx = -\frac{\cot x\csc^{n-1}x}{n-1} + \frac{n-2}{n-1}\int \csc^{n-2}x\,dx, \quad n\neq 1,$

27. $\int \sinh x\,dx = \cosh x,$

28. $\int \cosh x\,dx = \sinh x,$

29. $\int \tanh x\,dx = \ln|\cosh x|,$

30. $\int \coth x\,dx = \ln|\sinh x|,$

31. $\int \operatorname{sech} x\,dx = \arctan\sinh x,$

32. $\int \operatorname{csch} x\,dx = \ln\left|\tanh\frac{x}{2}\right|,$

33. $\int \sinh^2 x\,dx = \frac14\sinh(2x) - \frac12 x,$

34. $\int \cosh^2 x\,dx = \frac14\sinh(2x) + \frac12 x,$

35. $\int \operatorname{sech}^2 x\,dx = \tanh x,$

36. $\int \operatorname{arcsinh}\frac{x}{a}dx = x\operatorname{arcsinh}\frac{x}{a} - \sqrt{x^2+a^2}, \quad a>0,$

37. $\int \operatorname{arctanh}\frac{x}{a}dx = x\operatorname{arctanh}\frac{x}{a} + \frac{a}{2}\ln|a^2-x^2|,$

38. $$\int \operatorname{arccosh}\frac{x}{a}dx = \begin{cases} x\operatorname{arccosh}\dfrac{x}{a} - \sqrt{x^2+a^2}, & \text{if } \operatorname{arccosh}\frac{x}{a}>0 \text{ and } a>0,\\ x\operatorname{arccosh}\dfrac{x}{a} + \sqrt{x^2+a^2}, & \text{if } \operatorname{arccosh}\frac{x}{a}<0 \text{ and } a>0,\end{cases}$$

39. $\int \frac{dx}{\sqrt{a^2+x^2}} = \ln\left(x+\sqrt{a^2+x^2}\right), \quad a>0,$

40. $\int \frac{dx}{a^2+x^2} = \frac1a\arctan\frac{x}{a}, \quad a>0,$

41. $\int \sqrt{a^2-x^2}\,dx = \frac{x}{2}\sqrt{a^2-x^2} + \frac{a^2}{2}\arcsin\frac{x}{a}, \quad a>0,$

42. $\int (a^2-x^2)^{3/2}dx = \frac{x}{8}(5a^2-2x^2)\sqrt{a^2-x^2} + \frac{3a^4}{8}\arcsin\frac{x}{a}, \quad a>0,$

43. $\int \frac{dx}{\sqrt{a^2-x^2}} = \arcsin\frac{x}{a}, \quad a>0,$

44. $\int \frac{dx}{a^2-x^2} = \frac{1}{2a}\ln\left|\frac{a+x}{a-x}\right|,$

45. $\int \frac{dx}{(a^2-x^2)^{3/2}} = \frac{x}{a^2\sqrt{a^2-x^2}},$

46. $\int \sqrt{a^2\pm x^2}\,dx = \frac{x}{2}\sqrt{a^2\pm x^2} \pm \frac{a^2}{2}\ln\left|x+\sqrt{a^2\pm x^2}\right|,$

47. $\int \frac{dx}{\sqrt{x^2-a^2}} = \ln\left|x+\sqrt{x^2-a^2}\right|, \quad a>0,$

48. $\int \frac{dx}{ax^2+bx} = \frac1a\ln\left|\frac{x}{a+bx}\right|,$

49. $\int x\sqrt{a+bx}\,dx = \frac{2(3bx-2a)(a+bx)^{3/2}}{15b^2},$

50. $\int \frac{\sqrt{a+bx}}{x}dx = 2\sqrt{a+bx} + a\int\frac{1}{x\sqrt{a+bx}}dx,$

51. $\int \frac{x}{\sqrt{a+bx}}dx = \frac{1}{\sqrt2}\ln\left|\frac{\sqrt{a+bx}-\sqrt a}{\sqrt{a+bx}+\sqrt a}\right|, \quad a>0,$

52. $\int \frac{\sqrt{a^2-x^2}}{x}dx = \sqrt{a^2-x^2} - a\ln\left|\frac{a+\sqrt{a^2-x^2}}{x}\right|,$

53. $\int x\sqrt{a^2-x^2}\,dx = -\frac13(a^2-x^2)^{3/2},$

54. $\int x^2\sqrt{a^2-x^2}\,dx = \frac{x}{8}(2x^2-a^2)\sqrt{a^2-x^2} + \frac{a^4}{8}\arcsin\frac{x}{a}, \quad a>0,$

55. $\int \frac{dx}{\sqrt{a^2-x^2}} = -\frac1a\ln\left|\frac{a+\sqrt{a^2-x^2}}{x}\right|,$

56. $\int \frac{x\,dx}{\sqrt{a^2-x^2}} = -\sqrt{a^2-x^2},$

57. $\int \frac{x^2\,dx}{\sqrt{a^2-x^2}} = -\frac{x}{2}\sqrt{a^2-x^2} + \frac{a^2}{2}\arcsin\frac{x}{a}, \quad a>0,$

58. $\int \frac{\sqrt{a^2+x^2}}{x}dx = \sqrt{a^2+x^2} - a\ln\left|\frac{a+\sqrt{a^2+x^2}}{x}\right|,$

59. $\int \frac{\sqrt{x^2-a^2}}{x}dx = \sqrt{x^2-a^2} - a\arccos\frac{a}{|x|}, \quad a>0,$

60. $\int x\sqrt{x^2\pm a^2}\,dx = \frac13(x^2\pm a^2)^{3/2},$

61. $\int \frac{dx}{x\sqrt{x^2+a^2}} = \frac1a\ln\left|\frac{x}{a+\sqrt{a^2+x^2}}\right|,$

Calculus Cont.

62. $\displaystyle\int \frac{dx}{x\sqrt{x^2-a^2}} = \tfrac{1}{a}\arccos\tfrac{a}{|x|}, \quad a>0,$

63. $\displaystyle\int \frac{dx}{x^2\sqrt{x^2\pm a^2}} = \mp\frac{\sqrt{x^2\pm a^2}}{a^2x},$

64. $\displaystyle\int \frac{x\,dx}{\sqrt{x^2\pm a^2}} = \sqrt{x^2\pm a^2},$

65. $\displaystyle\int \frac{\sqrt{x^2\pm a^2}}{x^4}\,dx = \mp\frac{(x^2+a^2)^{3/2}}{3a^2x^3},$

66. $$\int \frac{dx}{ax^2+bx+c} = \begin{cases} \dfrac{1}{\sqrt{b^2-4ac}}\ln\left|\dfrac{2ax+b-\sqrt{b^2-4ac}}{2ax+b+\sqrt{b^2-4ac}}\right|, & \text{if } b^2>4ac,\\[2ex] \dfrac{2}{\sqrt{4ac-b^2}}\arctan\dfrac{2ax+b}{\sqrt{4ac-b^2}}, & \text{if } b^2<4ac, \end{cases}$$

67. $$\int \frac{dx}{\sqrt{ax^2+bx+c}} = \begin{cases} \dfrac{1}{\sqrt{a}}\ln\left|2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}\right|, & \text{if } a>0,\\[2ex] \dfrac{1}{\sqrt{-a}}\arcsin\dfrac{-2ax-b}{\sqrt{b^2-4ac}}, & \text{if } a<0, \end{cases}$$

68. $$\int \sqrt{ax^2+bx+c}\,dx = \frac{2ax+b}{4a}\sqrt{ax^2+bx+c} + \frac{4ax-b^2}{8a}\int\frac{dx}{\sqrt{ax^2+bx+c}},$$

69. $$\int \frac{x\,dx}{\sqrt{ax^2+bx+c}} = \frac{\sqrt{ax^2+bx+c}}{a} - \frac{b}{2a}\int\frac{dx}{\sqrt{ax^2+bx+c}},$$

70. $$\int \frac{dx}{x\sqrt{ax^2+bx+c}} = \begin{cases} \dfrac{-1}{\sqrt{c}}\ln\left|\dfrac{2\sqrt{c}\sqrt{ax^2+bx+c}+bx+2c}{x}\right|, & \text{if } c>0,\\[2ex] \dfrac{1}{\sqrt{-c}}\arcsin\dfrac{bx+2c}{|x|\sqrt{b^2-4ac}}, & \text{if } c<0, \end{cases}$$

71. $\displaystyle\int x^3\sqrt{x^2+a^2}\,dx = (\tfrac{1}{3}x^2-\tfrac{2}{15}a^2)(x^2+a^2)^{3/2},$

72. $\displaystyle\int x^n\sin(ax)\,dx = -\tfrac{1}{a}x^n\cos(ax) + \tfrac{n}{a}\int x^{n-1}\cos(ax)\,dx,$

73. $\displaystyle\int x^n\cos(ax)\,dx = \tfrac{1}{a}x^n\sin(ax) - \tfrac{n}{a}\int x^{n-1}\sin(ax)\,dx,$

74. $\displaystyle\int x^n e^{ax}\,dx = \frac{x^n e^{ax}}{a} - \tfrac{n}{a}\int x^{n-1}e^{ax}\,dx,$

75. $\displaystyle\int x^n\ln(ax)\,dx = x^{n+1}\left(\frac{\ln(ax)}{n+1} - \frac{1}{(n+1)^2}\right),$

76. $\displaystyle\int x^n(\ln ax)^m\,dx = \frac{x^{n+1}}{n+1}(\ln ax)^m - \frac{m}{n+1}\int x^n(\ln ax)^{m-1}\,dx.$

$$\begin{aligned}
x^1 &= x^{\underline{1}} &&= x^{\overline{1}}\\
x^2 &= x^{\underline{2}}+x^{\underline{1}} &&= x^{\overline{2}}-x^{\overline{1}}\\
x^3 &= x^{\underline{3}}+3x^{\underline{2}}+x^{\underline{1}} &&= x^{\overline{3}}-3x^{\overline{2}}+x^{\overline{1}}\\
x^4 &= x^{\underline{4}}+6x^{\underline{3}}+7x^{\underline{2}}+x^{\underline{1}} &&= x^{\overline{4}}-6x^{\overline{3}}+7x^{\overline{2}}-x^{\overline{1}}\\
x^5 &= x^{\underline{5}}+15x^{\underline{4}}+25x^{\underline{3}}+10x^{\underline{2}}+x^{\underline{1}} &&= x^{\underline{5}}-15x^{\overline{4}}+25x^{\overline{3}}-10x^{\overline{2}}+x^{\overline{1}}
\end{aligned}$$

$$\begin{aligned}
x^{\overline{1}} &= x^1 & x^{\underline{1}} &= x^1\\
x^{\overline{2}} &= x^2+x^1 & x^{\underline{2}} &= x^2-x^1\\
x^{\overline{3}} &= x^3+3x^2+2x^1 & x^{\underline{3}} &= x^3-3x^2+2x^1\\
x^{\overline{4}} &= x^4+6x^3+11x^2+6x^1 & x^{\underline{4}} &= x^4-6x^3+11x^2-6x^1\\
x^{\overline{5}} &= x^5+10x^4+35x^3+50x^2+24x^1 & x^{\underline{5}} &= x^5-10x^4+35x^3-50x^2+24x^1
\end{aligned}$$

Finite Calculus

Difference, shift operators:

$$\Delta f(x) = f(x+1) - f(x),$$
$$\mathrm{E}\, f(x) = f(x+1).$$

Fundamental Theorem:

$$f(x) = \Delta F(x) \Leftrightarrow \sum f(x)\delta x = F(x) + C.$$

$$\sum_a^b f(x)\delta x = \sum_{i=a}^{b-1} f(i).$$

Differences:

$\Delta(cu) = c\Delta u, \qquad \Delta(u+v) = \Delta u + \Delta v,$

$\Delta(uv) = u\Delta v + \mathrm{E}\, v\Delta u,$

$\Delta(x^{\underline{n}}) = nx^{\underline{n-1}},$

$\Delta(H_x) = x^{\underline{-1}}, \qquad \Delta(2^x) = 2^x,$

$\Delta(c^x) = (c-1)c^x, \qquad \Delta\binom{x}{m} = \binom{x}{m-1}.$

Sums:

$\sum cu\,\delta x = c\sum u\,\delta x,$

$\sum (u+v)\,\delta x = \sum u\,\delta x + \sum v\,\delta x,$

$\sum u\Delta v\,\delta x = uv - \sum \mathrm{E}\, v\Delta u\,\delta x,$

$\sum x^{\underline{n}}\,\delta x = \frac{x^{\underline{n+1}}}{m+1}, \qquad \sum x^{\underline{-1}}\,\delta x = H_x,$

$\sum c^x\,\delta x = \frac{c^x}{c-1}, \qquad \sum \binom{x}{m}\,\delta x = \binom{x}{m+1}.$

Falling Factorial Powers:

$$x^{\underline{n}} = x(x-1)\cdots(x-n+1), \quad n>0,$$
$$x^{\underline{0}} = 1,$$
$$x^{\underline{n}} = \frac{1}{(x+1)\cdots(x+|n|)}, \quad n<0,$$
$$x^{\underline{n+m}} = x^{\underline{m}}(x-m)^{\underline{n}}.$$

Rising Factorial Powers:

$$x^{\overline{n}} = x(x+1)\cdots(x+n-1), \quad n>0,$$
$$x^{\overline{0}} = 1,$$
$$x^{\overline{n}} = \frac{1}{(x-1)\cdots(x-|n|)}, \quad n<0,$$
$$x^{\overline{n+m}} = x^{\overline{m}}(x+m)^{\overline{n}}.$$

Conversion:

$$x^{\underline{n}} = (-1)^n(-x)^{\overline{n}} = (x-n+1)^{\overline{n}}$$
$$= 1/(x+1)^{\overline{-n}},$$
$$x^{\overline{n}} = (-1)^n(-x)^{\underline{n}} = (x+n-1)^{\underline{n}}$$
$$= 1/(x-1)^{\underline{-n}},$$
$$x^n = \sum_{k=1}^n \left\{ n \atop k \right\} x^{\underline{k}} = \sum_{k=1}^n \left\{ n \atop k \right\} (-1)^{n-k} x^{\overline{k}},$$
$$x^{\underline{n}} = \sum_{k=1}^n \left[n \atop k \right] (-1)^{n-k} x^k,$$
$$x^{\overline{n}} = \sum_{k=1}^n \left[n \atop k \right] x^k.$$

Theoretical Computer Science Cheat Sheet

Series

Taylor's series:

$$f(x) = f(a) + (x-a)f'(a) + \frac{(x-a)^2}{2}f''(a) + \cdots = \sum_{i=0}^{\infty} \frac{(x-a)^i}{i!} f^{(i)}(a).$$

Expansions:

$$\frac{1}{1-x} = 1 + x + x^2 + x^3 + x^4 + \cdots = \sum_{i=0}^{\infty} x^i,$$

$$\frac{1}{1-cx} = 1 + cx + c^2x^2 + c^3x^3 + \cdots = \sum_{i=0}^{\infty} c^i x^i,$$

$$\frac{1}{1-x^n} = 1 + x^n + x^{2n} + x^{3n} + \cdots = \sum_{i=0}^{\infty} x^{ni},$$

$$\frac{x}{(1-x)^2} = x + 2x^2 + 3x^3 + 4x^4 + \cdots = \sum_{i=0}^{\infty} ix^i,$$

$$x^k \frac{d^n}{dx^n}\left(\frac{1}{1-x}\right) = x + 2^n x^2 + 3^n x^3 + 4^n x^4 + \cdots = \sum_{i=0}^{\infty} i^n x^i,$$

$$e^x = 1 + x + \tfrac{1}{2}x^2 + \tfrac{1}{6}x^3 + \cdots = \sum_{i=0}^{\infty} \frac{x^i}{i!},$$

$$\ln(1+x) = x - \tfrac{1}{2}x^2 + \tfrac{1}{3}x^3 - \tfrac{1}{4}x^4 - \cdots = \sum_{i=1}^{\infty} (-1)^{i+1}\frac{x^i}{i},$$

$$\ln\frac{1}{1-x} = x + \tfrac{1}{2}x^2 + \tfrac{1}{3}x^3 + \tfrac{1}{4}x^4 + \cdots = \sum_{i=1}^{\infty} \frac{x^i}{i},$$

$$\sin x = x - \tfrac{1}{3!}x^3 + \tfrac{1}{5!}x^5 - \tfrac{1}{7!}x^7 + \cdots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)!},$$

$$\cos x = 1 - \tfrac{1}{2!}x^2 + \tfrac{1}{4!}x^4 - \tfrac{1}{6!}x^6 + \cdots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i}}{(2i)!},$$

$$\tan^{-1} x = x - \tfrac{1}{3}x^3 + \tfrac{1}{5}x^5 - \tfrac{1}{7}x^7 + \cdots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)},$$

$$(1+x)^n = 1 + nx + \tfrac{n(n-1)}{2}x^2 + \cdots = \sum_{i=0}^{\infty} \binom{n}{i} x^i,$$

$$\frac{1}{(1-x)^{n+1}} = 1 + (n+1)x + \tbinom{n+2}{2}x^2 + \cdots = \sum_{i=0}^{\infty} \binom{i+n}{i} x^i,$$

$$\frac{x}{e^x - 1} = 1 - \tfrac{1}{2}x + \tfrac{1}{12}x^2 - \tfrac{1}{720}x^4 + \cdots = \sum_{i=0}^{\infty} \frac{B_i x^i}{i!},$$

$$\frac{1}{2x}(1 - \sqrt{1-4x}) = 1 + x + 2x^2 + 5x^3 + \cdots = \sum_{i=0}^{\infty} \frac{1}{i+1}\binom{2i}{i} x^i,$$

$$\frac{1}{\sqrt{1-4x}} = 1 + x + 2x^2 + 6x^3 + \cdots = \sum_{i=0}^{\infty} \binom{2i}{i} x^i,$$

$$\frac{1}{\sqrt{1-4x}}\left(\frac{1-\sqrt{1-4x}}{2x}\right)^n = 1 + (2+n)x + \tbinom{4+n}{2}x^2 + \cdots = \sum_{i=0}^{\infty} \binom{2i+n}{i} x^i,$$

$$\frac{1}{1-x}\ln\frac{1}{1-x} = x + \tfrac{3}{2}x^2 + \tfrac{11}{6}x^3 + \tfrac{25}{12}x^4 + \cdots = \sum_{i=1}^{\infty} H_i x^i,$$

$$\frac{1}{2}\left(\ln\frac{1}{1-x}\right)^2 = \tfrac{1}{2}x^2 + \tfrac{3}{4}x^3 + \tfrac{11}{24}x^4 + \cdots = \sum_{i=2}^{\infty} \frac{H_{i-1}x^i}{i},$$

$$\frac{x}{1-x-x^2} = x + x^2 + 2x^3 + 3x^4 + \cdots = \sum_{i=0}^{\infty} F_i x^i,$$

$$\frac{F_n x}{1 - (F_{n-1} + F_{n+1})x - (-1)^n x^2} = F_n x + F_{2n}x^2 + F_{3n}x^3 + \cdots = \sum_{i=0}^{\infty} F_{ni} x^i.$$

Ordinary power series:

$$A(x) = \sum_{i=0}^{\infty} a_i x^i.$$

Exponential power series:

$$A(x) = \sum_{i=0}^{\infty} a_i \frac{x^i}{i!}.$$

Dirichlet power series:

$$A(x) = \sum_{i=1}^{\infty} \frac{a_i}{i^x}.$$

Binomial theorem:

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k.$$

Difference of like powers:

$$x^n - y^n = (x-y)\sum_{k=0}^{n-1} x^{n-1-k} y^k.$$

For ordinary power series:

$$\alpha A(x) + \beta B(x) = \sum_{i=0}^{\infty} (\alpha a_i + \beta b_i) x^i,$$

$$x^k A(x) = \sum_{i=k}^{\infty} a_{i-k} x^i,$$

$$\frac{A(x) - \sum_{i=0}^{k-1} a_i x^i}{x^k} = \sum_{i=0}^{\infty} a_{i+k} x^i,$$

$$A(cx) = \sum_{i=0}^{\infty} c^i a_i x^i,$$

$$A'(x) = \sum_{i=0}^{\infty} (i+1) a_{i+1} x^i,$$

$$xA'(x) = \sum_{i=1}^{\infty} i a_i x^i,$$

$$\int A(x)\, dx = \sum_{i=1}^{\infty} \frac{a_{i-1}}{i} x^i,$$

$$\frac{A(x) + A(-x)}{2} = \sum_{i=0}^{\infty} a_{2i} x^{2i},$$

$$\frac{A(x) - A(-x)}{2} = \sum_{i=0}^{\infty} a_{2i+1} x^{2i+1}.$$

Summation: If $b_i = \sum_{j=0}^{i} a_i$ then

$$B(x) = \frac{1}{1-x} A(x).$$

Convolution:

$$A(x)B(x) = \sum_{i=0}^{\infty} \left(\sum_{j=0}^{i} a_j b_{i-j}\right) x^i.$$

God made the natural numbers;
all the rest is the work of man.
– Leopold Kronecker

Theoretical Computer Science Cheat Sheet

Series

Expansions:

$$\frac{1}{(1-x)^{n+1}}\ln\frac{1}{1-x} = \sum_{i=0}^{\infty}(H_{n+i}-H_n)\binom{n+i}{i}x^i,$$

$$\left(\frac{1}{x}\right)^{\overline{-n}} = \sum_{i=0}^{\infty}\left\{ {i \atop n} \right\}x^i,$$

$$x^{\overline{n}} = \sum_{i=0}^{\infty}\left[{n \atop i} \right]x^i,$$

$$(e^x-1)^n = \sum_{i=0}^{\infty}\left\{ {i \atop n} \right\}\frac{n!x^i}{i!},$$

$$\left(\ln\frac{1}{1-x}\right)^n = \sum_{i=0}^{\infty}\left[{i \atop n} \right]\frac{n!x^i}{i!},$$

$$x\cot x = \sum_{i=0}^{\infty}\frac{(-4)^i B_{2i}x^{2i}}{(2i)!},$$

$$\tan x = \sum_{i=1}^{\infty}(-1)^{i-1}\frac{2^{2i}(2^{2i}-1)B_{2i}x^{2i-1}}{(2i)!},$$

$$\zeta(x) = \sum_{i=1}^{\infty}\frac{1}{i^x},$$

$$\frac{1}{\zeta(x)} = \sum_{i=1}^{\infty}\frac{\mu(i)}{i^x},$$

$$\frac{\zeta(x-1)}{\zeta(x)} = \sum_{i=1}^{\infty}\frac{\phi(i)}{i^x},$$

$$\zeta(x) = \prod_p\frac{1}{1-p^{-x}},$$

$$\zeta^2(x) = \sum_{i=1}^{\infty}\frac{d(i)}{x^i} \quad \text{where } d(n)=\textstyle\sum_{d|n}1,$$

$$\zeta(x)\zeta(x-1) = \sum_{i=1}^{\infty}\frac{S(i)}{x^i} \quad \text{where } S(n)=\textstyle\sum_{d|n}d,$$

$$\zeta(2n) = \frac{2^{2n-1}|B_{2n}|}{(2n)!}\pi^{2n}, \quad n\in\mathbb{N},$$

$$\frac{x}{\sin x} = \sum_{i=0}^{\infty}(-1)^{i-1}\frac{(4^i-2)B_{2i}x^{2i}}{(2i)!},$$

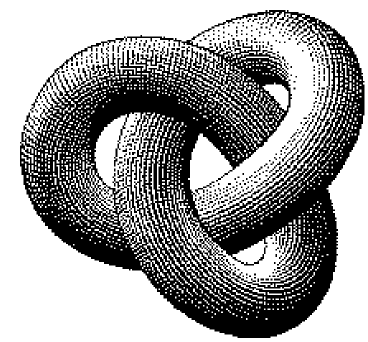
$$\left(\frac{1-\sqrt{1-4x}}{2x}\right)^n = \sum_{i=0}^{\infty}\frac{n(2i+n-1)!}{i!(n+i)!}x^i,$$

$$e^x\sin x = \sum_{i=1}^{\infty}\frac{2^{i/2}\sin\frac{i\pi}{4}}{i!}x^i,$$

$$\sqrt{\frac{1-\sqrt{1-x}}{x}} = \sum_{i=0}^{\infty}\frac{(4i)!}{16^i\sqrt{2}(2i)!(2i+1)!}x^i,$$

$$\left(\frac{\arcsin x}{x}\right)^2 = \sum_{i=0}^{\infty}\frac{4^i i!^2}{(i+1)(2i+1)!}x^{2i}.$$

Escher's Knot



Stieltjes Integration

If G is continuous in the interval $[a,b]$ and F is nondecreasing then

$$\int_a^b G(x)\,dF(x)$$

exists. If $a\le b\le c$ then

$$\int_a^c G(x)\,dF(x) = \int_a^b G(x)\,dF(x) + \int_b^c G(x)\,dF(x).$$

If the integrals involved exist

$$\int_a^b \big(G(x)+H(x)\big)\,dF(x) = \int_a^b G(x)\,dF(x) + \int_a^b H(x)\,dF(x),$$

$$\int_a^b G(x)\,d\big(F(x)+H(x)\big) = \int_a^b G(x)\,dF(x) + \int_a^b G(x)\,dH(x),$$

$$\int_a^b c\cdot G(x)\,dF(x) = \int_a^b G(x)\,d\big(c\cdot F(x)\big) = c\int_a^b G(x)\,dF(x),$$

$$\int_a^b G(x)\,dF(x) = G(b)F(b) - G(a)F(a) - \int_a^b F(x)\,dG(x).$$

If the integrals involved exist, and F possesses a derivative F' at every point in $[a,b]$ then

$$\int_a^b G(x)\,dF(x) = \int_a^b G(x)F'(x)\,dx.$$

Cramer's Rule

If we have equations:

$$\begin{matrix} a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n = b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n = b_2 \\ \vdots \qquad\quad \vdots \qquad\qquad \vdots \\ a_{n,1}x_1 + a_{n,2}x_2 + \cdots + a_{n,n}x_n = b_n \end{matrix}$$

Let $A=(a_{i,j})$ and B be the column matrix (b_i). Then there is a unique solution iff $\det A\neq 0$. Let A_i be A with column i replaced by B. Then

$$x_i = \frac{\det A_i}{\det A}.$$

00	47	18	76	29	93	85	34	61	52
86	11	57	28	70	39	94	45	02	63
95	80	22	67	38	71	49	56	13	04
59	96	81	33	07	48	72	60	24	15
73	69	90	82	44	17	58	01	35	26
68	74	09	91	83	55	27	12	46	30
37	08	75	19	92	84	66	23	50	41
14	25	36	40	51	62	03	77	88	99
21	32	43	54	65	06	10	89	97	78
42	53	64	05	16	20	31	98	79	87

The Fibonacci number system: Every integer n has a unique representation

$$n = F_{k_1} + F_{k_2} + \cdots + F_{k_m},$$

where $k_i \ge k_{i+1}+2$ for all i, $1\le i<m$ and $k_m\ge 2$.

Fibonacci Numbers

$1,1,2,3,5,8,13,21,34,55,89,\ldots$

Definitions:

$$F_i = F_{i-1}+F_{i-2}, \quad F_0=F_1=1,$$

$$F_{-i} = (-1)^{i-1}F_i,$$

$$F_i = \frac{1}{\sqrt{5}}\left(\phi^i - \hat{\phi}^i\right),$$

Cassini's identity: for $i>0$:

$$F_{i+1}F_{i-1} - F_i^2 = (-1)^i.$$

Additive rule:

$$F_{n+k} = F_kF_{n+1} + F_{k-1}F_n,$$

$$F_{2n} = F_nF_{n+1} + F_{n-1}F_n.$$

Calculation by matrices:

$$\begin{pmatrix} F_{n-2} & F_{n-1} \\ F_{n-1} & F_n \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n.$$

Improvement makes strait roads, but the crooked roads without Improvement, are roads of Genius.
– William Blake (The Marriage of Heaven and Hell)