Team Notes

Contents

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
        ];
}
```

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;
}
```

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;
}
```

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;
    }
}
```

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;
}
```

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
```

7 Tonelli-Shanks

```
long pow_mod(long x, long n, long p) {
  if (n == 0) return 1;
  if (n & 1)
    return (pow_mod(x, n-1, p) * x) % p;
  x = pow_mod(x, n/2, p);
  return (x * x) % p;
}

/* Takes as input an odd prime p and n < p and
      returns r
 * such that r * r = n [mod p]. */
long tonelli_shanks(long n, long p) {
  long s = 0;
  long q = p - 1;
  while ((q & 1) == 0) { q /= 2; ++s; }
  if (s == 1) {
    long r = pow_mod(n, (p+1)/4, p);
    if ((r * r) % p == n) return r;
    return 0;
  }
  // Find the first quadratic non-residue z by
      brute-force search
  long z = 1;
  while (pow_mod(++z, (p-1)/2, p) != p - 1);
  long c = pow_mod(z, q, p);
  long r = pow_mod(n, (q+1)/2, p);
  long t = pow_mod(n, q, p);
  long m = s;
  while (t != 1) {
    long tt = t;
    long i = 0;
    while (tt != 1) {
      tt = (tt * tt) % p;
      ++i;
      if (i == m) return 0;
    }
    long b = pow_mod(c, pow_mod(2, m-i-1, p-1), p
        );
    long b2 = (b * b) % p;
    r = (r * b) % p;
```

```cpp
        t = (t * b2) % p;
        c = b2;
        m = i;
    }
    if ((r * r) % p == n) return r;
    return 0;
}
```

8 Suffix Array

```cpp
#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--;
            }
        }
    }
};
```

9 Z algorithm

```
Suppose we are given a string s of length n. The
     Z-function for this string is an array of
     length  where the i-th element is equal to
      the greatest number of characters
     starting from the position i that coincide
      with the first characters of s.
```

```cpp
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;
}
```

10 Manacher

```cpp
//Given string s with length n. Find all the pairs (i, j) such that substring s[i, j] is a palindrome.

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;
    }
}
```

11 Suffix Automaton

```cpp
//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);
        }
    }
};
```

12 Palindromic Tree

```cpp
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];
}
```

13 Geometry

```cpp
#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));
}

// 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;
}
```

14 Gauus Elimination

```
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
}
```

15 Simplex Algorithm

```
/**
 * 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;
    }
};
```

16 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;
    }
};
```

17 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
```

```cpp
 * +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;
                }
            }
        }
    }
}
```

18 FFT chemthan

```cpp
#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;
    }
}
```

19 Binary vector space

```cpp
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
}
```

20 DiophanteMod

```cpp
// 1 <= a * k <= r (k > 0) (1 <= 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);
}
```

21 Linear Sieve

```cpp
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;
        }
    }
}
```

22 Arborescence

```cpp
//find mst in directed graph

namespace Arborescence {
    const int maxv = 2550;
    const int maxe = maxv * maxv;
    const long long INF = (long long) 1e18;

    struct edge_t {
        int u, v;
        long long w;
        edge_t(int u = 0, int v = 0, long long w = 0) : u(u), v(v), w(w) {}
    } edge[maxe];

    int ec;
    int id[maxv], pre[maxv];
    long long in[maxv];
    int vis[maxv];

    void init() {
        ec = 0;
    }

    void add(int u, int v, int w) {
        // 1-indexed
        edge[++ec] = edge_t(u, v, w);
    }

    long long mst(int n, int rt) {
        long long res = 0;
        int idx;
        while (1) {
            for (int i = 1; i <= n; i++) {
                in[i] = INF, vis[i] = -1, id[i] = -1;
            }
            for (int i = 1; i <= ec; i++) {
                int u = edge[i].u, v = edge[i].v;
                if (u == v || in[v] <= edge[i].w) continue;
                in[v] = edge[i].w, pre[v] = u;
            }
            pre[rt] = rt, in[rt] = 0;
            for (int i = 1; i <= n; i++) {
                res += in[i];
                if (in[i] == INF) return -1;
            }

            idx = 0;
            for (int i = 1; i <= n; i++) {
                if (vis[i] != -1) continue;
                int u = i, v;
                while (vis[u] == -1) {
                    vis[u] = i;
                    u = pre[u];
                }
                if (vis[u] != i || u == rt) continue;
                for (v = u, u = pre[u], idx++; u != v; u = pre[u]) id[u] = idx;
                id[v] = idx;
            }
            if (idx == 0) return res;
            for (int i = 1; i <= n; i++) if (id[i] == -1) id[i] = ++idx;
            for (int i = 1; i <= ec; i++) {
                int u = edge[i].u, v = edge[i].v;
                edge[i].u = id[u], edge[i].v = id[v];
                edge[i].w -= in[v];
            }
            n = idx, rt = id[rt];
        }
        return res;
    }
}
```

23 Arborescence With Trace

```
namespace Arborescence {
    static const int maxv = 2555 + 5;
    static const int maxe = maxv * maxv;
    int n, m, root;
    int pre[maxv], node[maxv], vis[maxv], best[maxv];
    struct Cost;
    vector<Cost> costlist;
    struct Cost {
        int id;
        long long val;
        int used, a, b, pos;
        Cost() {val = -1; used = 0;}
        Cost(int id, long long val, bool temp) : id(id), val(val) {
            a = b = -1, used = 0;
            pos = costlist.size();
            costlist.push_back(*this);
        }
        Cost(int a, int b) : a(a), b(b) {
            id = -1;
            val = costlist[a].val - costlist[b].val;
            used = 0; pos = costlist.size();
            costlist.push_back(*this);
        }
        void push() {
            if (id == -1) {
                costlist[a].used += used;
                costlist[b].used -= used;
            }
        }
    };
    struct Edge {
        int u, v;
        Cost cost;
        Edge() {}
        Edge(int id, int u, int v, long long c) : u(u), v(v) {
            cost = Cost(id, c, 0);
        }
    } edge[maxe];

    void init(int _n) {
        n = _n; m = 0;
        costlist.clear();
    }
    void add(int id, int u, int v, long long c) {
        // zero indexed
        edge[m++] = Edge(id, u, v, c);
    }
    long long mst(int root) {
        long long res = 0;
        while (1) {
            for (int i = 0; i < n; i++) best[i] = -1;
            for (int e = 0; e < m; e++) {
                int u = edge[e].u, v = edge[e].v;
                if ((best[v] == -1 || edge[e].cost.val < costlist[best[v]].val) && u != v) {
                    pre[v] = u;
                    best[v] = edge[e].cost.pos;
                }
            }
            for (int i = 0; i < n; i++) if (i != root && best[i] == -1) return -1;
            int cntnode = 0;
            memset(node, -1, sizeof(node));
            memset(vis, -1, sizeof(vis));
            for (int i = 0; i < n; i++) if (i != root) {
                res += costlist[best[i]].val;
                costlist[best[i]].used++;
                int v = i;
                while (vis[v] != i && node[v] == -1 && v != root) {
                    vis[v] = i;
                    v = pre[v];
                }
                if (v != root && node[v] == -1) {
                    for (int u = pre[v]; u != v; u = pre[u]) node[u] = cntnode;
                    node[v] = cntnode++;
                }
            }
            if (cntnode == 0) break;
            for (int i = 0; i < n; i++) if (node[i] == -1) node[i] = cntnode++;
            for (int e = 0; e < m; e++) {
                int v = edge[e].v;
                edge[e].u = node[edge[e].u];
                edge[e].v = node[edge[e].v];
                if (edge[e].u != edge[e].v) edge[e].cost = Cost(edge[e].cost.pos, best[v]);
            }
            n = cntnode;
            root = node[root];
        }
        return res;
    }
    vector<int> trace() {
        vector<int> res;
        for (int i = costlist.size() - 1; i >= 0; i--) costlist[i].push();
        for (int i = 0; i < costlist.size(); i++) {
            Cost cost = costlist[i];
            if (cost.id != -1 && cost.used > 0)
                res.push_back(cost.id);
        }
        return res;
    }
}
```

24 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;
```

25 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;
    }
};
```

26 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]];
```

```cpp
                    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;
}
}
```

27 Dinic Flow

```cpp
/*
    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;
    }
};
```

28 Dinic Flow With Scaling

```cpp
/*
    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;
    }
};
```

29 Gomory Hu Tree

```cpp
// 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;
}
```

30 Min Cost-Max Flow

```cpp
/*
    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};
    }
};
```

31 Min Cost Max Flow Potential

```cpp
/*
    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 <--
*/
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;
                    }
                }
            }
        }
    }
};
```

32 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;
    }
};
```

33 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;
    }
};
```

34 Undirected mincut

```cpp
/*
 * 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;
    }
}
```

35 Eulerian Path/Circuit

```
In graph theory, an Eulerian trail (or Eulerian
    path) is a trail in a finite graph that
    visits every edge exactly once (allowing
    for revisiting vertices).

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;
    }
};
```

36 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;
    }
};
```

37 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];

    }
};
```

38 Treap

```
class Treap {
    struct Node {
        int key, prior, size;
        int now;
        Node *l, *r;

        Node(int key): key(key), prior((rand() &
            0xFFFF) << 10 | (rand())), size(1)
            , l(NULL), r(NULL), now(1) {}
        ~Node() { delete l; delete r; }
    };

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

    void update(Node *x) {
        if (!x) return;
        x->size = size(x->l) + size(x->r) + x->
            now;
    }

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

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

    void show(Node *x) {
        if (!x) return;
        show(x->l);
        cout << x->key << '-' << x->now << " | ";
        show(x->r);
    }

    Node *root;

public:
    Treap(): root(NULL) {}
    ~Treap() { delete root; }

    bool insert(int x) {
        Node *l, *mid, *r;
        split(root, x, l, mid);
        split(mid, x + 1, mid, r);
        if (mid) {
            mid->now++;
            mid->size++;
        } else {
            mid = new Node(x);
        }
        root = join(join(l, mid), r);
        return true;
    }

    int countSmaller(int x) {
        Node *l, *r;
        split(root, x, l, r);
        int res = size(l);
        root = join(l, r);
        return res;
    }

    int countSmallerOrEqual(int x) {
        return countSmaller(x + 1);
```

```
    }

    int countGreaterOrEqual(int x) {
        return size() - countSmaller(x);
    }

    int size() const { return root ? root->size :
         0; }

    void show() {
        cout << "{";
        show(root);
        cout << "}\n";
    }
};
```

39 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;
}
```

40 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;
}
```

41 Ordered Set

```
#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>;
```

42 Unordered Map

```
struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        // http://xorshift.di.unimi.it/splitmix64
            .c
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM =
            chrono::steady_clock::now().
            time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
};
unordered_map<long long, int, custom_hash>
      safe_map;
gp_hash_table<long long, int, custom_hash>
      safe_hash_table;
```

43 RNG

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

44 SQRT forloop

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

45 IT(Hotamago)

```
struct IT{
    int n;
    vector<int> it;
    void init(int _n){
        n = _n;
        it = vector<int>(4*n + 1, 0);
    }
    void update(int i, int l, int r, int u, int
        val){
        if(l > u || r < u) return;
        if(l >= r){
            it[i] = val;
            return;
        }
        int mid = (l+r)/2;
        update(i*2, l, mid, u, val);
        update(i*2 + 1, mid + 1, r, u, val);
        it[i] = min(it[i*2], it[i*2 + 1]);
    }
    int query(int i, int l, int r, int u, int v){
        if(l > v || r < u) return 1000000007;
        if(l >= u && r <= v) return it[i];
        int mid = (l+r)/2;
        return min(query(i*2, l, mid, u, v),
            query(i*2 + 1, mid + 1, r, u, v));
    }
    void upd(int u, int val){
        update(1,1,n,u,val);
    }
    int quy(int l, int r){
        return query(1, 1, n, l, r);
    }
} se;
```

46 ITLAZY(Hotamago)

```cpp
struct sem{
    int l, r, w;
} sems[300005];
struct node{
    int val, lazy;
};
struct IT{
    int n;
    vector<node> it;
    void downnode(int i){
        it[i*2].val += it[i].lazy;
        it[i*2 + 1].val += it[i].lazy;
        it[i*2].lazy += it[i].lazy;
        it[i*2 + 1].lazy += it[i].lazy;
        it[i].lazy = 0;
    }
    void init(int _n){
        n = _n;
        it = vector<node>(4*n + 1, {0, 0});
    }
    void update(int i, int l, int r, int u, int v
        , int val){
        if(l > v || r < u) return;
        if(l >= u && r <= v){
            it[i].val += val;
            it[i].lazy += val;
            return;
        }
        int mid = (l+r)/2;
        downnode(i);
        update(i*2, l, mid, u, v, val);
        update(i*2 + 1, mid + 1, r, u, v, val);
        it[i].val = min(it[i*2].val, it[i*2 + 1].
            val);
    }
    int query(int i, int l, int r, int u, int v){
        if(l > v || r < u) return 1000000007;
        if(l >= u && r <= v) return it[i].val;
        int mid = (l+r)/2;
        downnode(i);
        return min(query(i*2, l, mid, u, v),
            query(i*2 + 1, mid + 1, r, u, v));
    }
    void upd(int l, int r, int val){
        update(1,1,n,l,r,val);
    }
    int quy(int l, int r){
        return query(1, 1, n, l, r);
    }
} se;
```

47 BgInt(Hotamago)

```cpp
const int base = 1000000000; const int
    base_digits = 9;
struct bigint {
    vector<int> a; int sign;

    bigint() :
        sign(1) {
    }

    bigint(long long v) {
        *this = v;
    }

    bigint(const string &s) {
        read(s);
    }

    void operator=(const bigint &v) {
        sign = v.sign;
        a = v.a;
    }

    void operator=(long long v) {
        sign = 1;
        if (v < 0)
            sign = -1, v = -v;
        for (; v > 0; v = v / base)
            a.push_back(v % base);
    }

    bigint operator+(const bigint &v) const {
        if (sign == v.sign) {
            bigint res = v;

            for (int i = 0, carry = 0; i < (int)
                max(a.size(), v.a.size()) ||
                carry; ++i) {
                if (i == (int) res.a.size())
                    res.a.push_back(0);
                res.a[i] += carry + (i < (int) a.
                    size() ? a[i] : 0);
                carry = res.a[i] >= base;
                if (carry)
                    res.a[i] -= base;
            }
            return res;
        }
        return *this - (-v);
    }

    bigint operator-(const bigint &v) const {
        if (sign == v.sign) {
            if (abs() >= v.abs()) {
                bigint res = *this;
                for (int i = 0, carry = 0; i < (
                    int) v.a.size() || carry;
                    ++i) {
                    res.a[i] -= carry + (i < (int
                        ) v.a.size() ? v.a[i]
                        : 0);
                    carry = res.a[i] < 0;
                    if (carry)
                        res.a[i] += base;
                }
                res.trim();
                return res;
            }
            return -(v - *this);
        }
        return *this + (-v);
    }

    void operator*=(int v) {
        if (v < 0)
            sign = -sign, v = -v;
        for (int i = 0, carry = 0; i < (int) a.
            size() || carry; ++i) {
            if (i == (int) a.size())
                a.push_back(0);
            long long cur = a[i] * (long long) v
                + carry;
            carry = (int) (cur / base);
            a[i] = (int) (cur % base);
            //asm("divl %%ecx" : "=a"(carry), "=d
                "(a[i]) : "A"(cur), "c"(base))
                ;
        }
        trim();
    }

    bigint operator*(int v) const {
        bigint res = *this;
        res *= v;
        return res;
    }

    friend pair<bigint, bigint> divmod(const
        bigint &a1, const bigint &b1) {
        int norm = base / (b1.a.back() + 1);
        bigint a = a1.abs() * norm;
        bigint b = b1.abs() * norm;
        bigint q, r;
        q.a.resize(a.a.size());

        for (int i = a.a.size() - 1; i >= 0; i--)
            {
            r *= base;
            r += a.a[i];
            int s1 = r.a.size() <= b.a.size() ? 0
                : r.a[b.a.size()];
            int s2 = r.a.size() <= b.a.size() - 1
                ? 0 : r.a[b.a.size() - 1];
            int d = ((long long) base * s1 + s2)
                / b.a.back();
            r -= b * d;
            while (r < 0)
                r += b, --d;
            q.a[i] = d;
        }

        q.sign = a1.sign * b1.sign;
        r.sign = a1.sign;
        q.trim();
        r.trim();
        return make_pair(q, r / norm);
    }

    bigint operator/(const bigint &v) const {
        return divmod(*this, v).first;
    }

    bigint operator%(const bigint &v) const {
        return divmod(*this, v).second;
    }

    void operator/=(int v) {
        if (v < 0)
            sign = -sign, v = -v;
        for (int i = (int) a.size() - 1, rem = 0;
            i >= 0; --i) {
            long long cur = a[i] + rem * (long
                long) base;
            a[i] = (int) (cur / v);
            rem = (int) (cur % v);
        }
        trim();
    }

    bigint operator/(int v) const {
        bigint res = *this;
        res /= v;
        return res;
    }

    int operator%(int v) const {
        if (v < 0)
            v = -v;
        int m = 0;
        for (int i = a.size() - 1; i >= 0; --i)
            m = (a[i] + m * (long long) base) % v
                ;
        return m * sign;
    }

    void operator+=(const bigint &v) {
        *this = *this + v;
    }
    void operator-=(const bigint &v) {
        *this = *this - v;
    }
    void operator*=(const bigint &v) {
        *this = *this * v;
    }
    void operator/=(const bigint &v) {
        *this = *this / v;
    }

    bool operator<(const bigint &v) const {
        if (sign != v.sign)
            return sign < v.sign;
        if (a.size() != v.a.size())
            return a.size() * sign < v.a.size() *
                v.sign;
        for (int i = a.size() - 1; i >= 0; i--)
            if (a[i] != v.a[i])
                return a[i] * sign < v.a[i] *
                    sign;
        return false;
    }

    bool operator>(const bigint &v) const {
        return v < *this;
    }
    bool operator<=(const bigint &v) const {
        return !(v < *this);
    }
    bool operator>=(const bigint &v) const {
        return !(*this < v);
    }
    bool operator==(const bigint &v) const {
        return !(*this < v) && !(v < *this);
    }
    bool operator!=(const bigint &v) const {
        return *this < v || v < *this;
    }

    void trim() {
        while (!a.empty() && !a.back())
            a.pop_back();
        if (a.empty())
            sign = 1;
    }

    bool isZero() const {
        return a.empty() || (a.size() == 1 && !a
            [0]);
    }

    bigint operator-() const {
        bigint res = *this;
        res.sign = -sign;
        return res;
    }

    bigint abs() const {
        bigint res = *this;
        res.sign *= res.sign;
        return res;
    }

    long long longValue() const {
        long long res = 0;
        for (int i = a.size() - 1; i >= 0; i--)
            res = res * base + a[i];
        return res * sign;
    }

    friend bigint gcd(const bigint &a, const
        bigint &b) {
        return b.isZero() ? a : gcd(b, a % b);
    }
    friend bigint lcm(const bigint &a, const
        bigint &b) {
        return a / gcd(a, b) * b;
    }

    void read(const string &s) {
        sign = 1;
        a.clear();
        int pos = 0;
        while (pos < (int) s.size() && (s[pos] ==
            '-' || s[pos] == '+')) {
            if (s[pos] == '-')
                sign = -sign;
            ++pos;
        }
        for (int i = s.size() - 1; i >= pos; i -=
            base_digits) {
            int x = 0;
            for (int j = max(pos, i - base_digits
                + 1); j <= i; j++)
                x = x * 10 + s[j] - '0';
            a.push_back(x);
        }
        trim();
    }

    friend istream& operator>>(istream &stream,
        bigint &v) {
        string s;
        stream >> s;
        v.read(s);
        return stream;
    }

    friend ostream& operator<<(ostream &stream,
        const bigint &v) {
        if (v.sign == -1)
            stream << '-';
        stream << (v.a.empty() ? 0 : v.a.back());
        for (int i = (int) v.a.size() - 2; i >=
            0; --i)
            stream << setw(base_digits) <<
                setfill('0') << v.a[i];
        return stream;
    }

    static vector<int> convert_base(const vector<
        int> &a, int old_digits, int
        new_digits) {
        vector<long long> p(max(old_digits,
            new_digits) + 1);
        p[0] = 1;
        for (int i = 1; i < (int) p.size(); i++)
            p[i] = p[i - 1] * 10;
```

```cpp
        vector<int> res;
        long long cur = 0;
        int cur_digits = 0;
        for (int i = 0; i < (int) a.size(); i++)
            {
            cur += a[i] * p[cur_digits];
            cur_digits += old_digits;
            while (cur_digits >= new_digits) {
                res.push_back(int(cur % p[
                    new_digits]));
                cur /= p[new_digits];
                cur_digits -= new_digits;
            }
        }
        res.push_back((int) cur);
        while (!res.empty() && !res.back())
            res.pop_back();
        return res;
    }

    typedef vector<long long> vll;

    static vll karatsubaMultiply(const vll &a,
        const vll &b) {
        int n = a.size();
        vll res(n + n);
        if (n <= 32) {
            for (int i = 0; i < n; i++)
                for (int j = 0; j < n; j++)
                    res[i + j] += a[i] * b[j];
            return res;
        }

        int k = n >> 1;
        vll a1(a.begin(), a.begin() + k);
        vll a2(a.begin() + k, a.end());
        vll b1(b.begin(), b.begin() + k);
        vll b2(b.begin() + k, b.end());

        vll a1b1 = karatsubaMultiply(a1, b1);
        vll a2b2 = karatsubaMultiply(a2, b2);

        for (int i = 0; i < k; i++)
            a2[i] += a1[i];
        for (int i = 0; i < k; i++)
            b2[i] += b1[i];

        vll r = karatsubaMultiply(a2, b2);
        for (int i = 0; i < (int) a1b1.size(); i
            ++)
            r[i] -= a1b1[i];
        for (int i = 0; i < (int) a2b2.size(); i
            ++)
            r[i] -= a2b2[i];

        for (int i = 0; i < (int) r.size(); i++)
            res[i + k] += r[i];
        for (int i = 0; i < (int) a1b1.size(); i
            ++)
            res[i] += a1b1[i];
        for (int i = 0; i < (int) a2b2.size(); i
            ++)
            res[i + n] += a2b2[i];
        return res;
    }

    bigint operator*(const bigint &v) const {
        vector<int> a6 = convert_base(this->a,
            base_digits, 6);
        vector<int> b6 = convert_base(v.a,
            base_digits, 6);
        vll a(a6.begin(), a6.end());
        vll b(b6.begin(), b6.end());
        while (a.size() < b.size())
            a.push_back(0);
        while (b.size() < a.size())
            b.push_back(0);
        while (a.size() & (a.size() - 1))
            a.push_back(0), b.push_back(0);
        vll c = karatsubaMultiply(a, b);
        bigint res;
        res.sign = sign * v.sign;
        for (int i = 0, carry = 0; i < (int) c.
            size(); i++) {
            long long cur = c[i] + carry;
            res.a.push_back((int) (cur % 1000000)
                );
            carry = (int) (cur / 1000000);
        }
        res.a = convert_base(res.a, 6,
            base_digits);
        res.trim();
        return res;
    }
};
```

48 LCA(SegTree)

```cpp
int n; // The number of nodes in the graph
vector<int> graph[100000];
int timer = 0, tin[100000], euler_tour[200000];
int segtree[800000]; // Segment tree for RMQ

void dfs(int node = 0, int parent = -1) {
        tin[node] = timer; // The time when we
            first visit a node
        euler_tour[timer++] = node;
        for (int i : graph[node]) {
                if (i != parent) {
                        dfs(i, node);
                        euler_tour[timer++] =
                            node;
                }
        }
}

int mn_tin(int x, int y) {
        if (x == -1) return y;
        if (y == -1) return x;
        return (tin[x] < tin[y] ? x : y);
}

// Build the segment tree: run 'build()' after
    running dfs'
void build(int node = 1, int l = 0, int r = timer
    - 1) {
        if (l == r) segtree[node] = euler_tour[l
            ];
        else {
                int mid = (l + r) / 2;
                build(node * 2, l, mid);
                build(node * 2 + 1, mid + 1, r);
                segtree[node] = mn_tin(segtree[
                    node * 2], segtree[node *
                    2 + 1]);
        }
}

int query(int a, int b, int node = 1, int l = 0,
    int r = timer - 1) {
        if (l > b || r < a) return -1;
        if (l >= a && r <= b) return segtree[node
            ];
        int mid = (l + r) / 2;
        return mn_tin(query(a, b, node * 2, l,
            mid), query(a, b, node * 2 + 1,
            mid + 1, r));
}

// Make sure you run $dfs' and 'build()' before
    you run this
int lca(int a, int b) {
        if (tin[a] > tin[b]) swap(a, b);
        return query(tin[a], tin[b]);
}
```

49 EulerTotientFunction

```cpp
void phi_1_to_n(int n) {
    vector<int> phi(n + 1);
    for (int i = 0; i <= n; i++)
        phi[i] = i;

    for (int i = 2; i <= n; i++) {
        if (phi[i] == i) {
            for (int j = i; j <= n; j += i)
                phi[j] -= phi[j] / i;
        }
    }
}
```

50 BigInt(Hotamago)

```cpp
const int base = 1000000000; const int
    base_digits = 9;
struct bigint {
    vector<int> a; int sign;

    bigint() :
        sign(1) {
    }

    bigint(long long v) {
        *this = v;
    }

    bigint(const string &s) {
        read(s);
    }

    void operator=(const bigint &v) {
        sign = v.sign;
        a = v.a;
    }

    void operator=(long long v) {
        sign = 1;
        if (v < 0)
            sign = -1, v = -v;
        for (; v > 0; v = v / base)
            a.push_back(v % base);
    }

    bigint operator+(const bigint &v) const {
        if (sign == v.sign) {
            bigint res = v;

            for (int i = 0, carry = 0; i < (int)
                max(a.size(), v.a.size()) ||
                carry; ++i) {
                if (i == (int) res.a.size())
                    res.a.push_back(0);
                res.a[i] += carry + (i < (int) a.
                    size() ? a[i] : 0);
                carry = res.a[i] >= base;
                if (carry)
                    res.a[i] -= base;
            }
            return res;
        }
        return *this - (-v);
    }

    bigint operator-(const bigint &v) const {
        if (sign == v.sign) {
            if (abs() >= v.abs()) {
                bigint res = *this;
                for (int i = 0, carry = 0; i < (
                    int) v.a.size() || carry;
                    ++i) {
                    res.a[i] -= carry + (i < (int
                        ) v.a.size() ? v.a[i]
                        : 0);
                    carry = res.a[i] < 0;
                    if (carry)
                        res.a[i] += base;
                }
                res.trim();
                return res;
            }
            return -(v - *this);
        }
        return *this + (-v);
    }

    void operator*=(int v) {
        if (v < 0)
            sign = -sign, v = -v;
        for (int i = 0, carry = 0; i < (int) a.
            size() || carry; ++i) {
            if (i == (int) a.size())
                a.push_back(0);
            long long cur = a[i] * (long long) v
                + carry;
            carry = (int) (cur / base);
            a[i] = (int) (cur % base);
            //asm("divl %%ecx" : "=a"(carry), "=d
                "(a[i]) : "A"(cur), "c"(base))
                ;
        }
        trim();
    }

    bigint operator*(int v) const {
        bigint res = *this;
        res *= v;
        return res;
    }

    friend pair<bigint, bigint> divmod(const
        bigint &a1, const bigint &b1) {
        int norm = base / (b1.a.back() + 1);
        bigint a = a1.abs() * norm;
        bigint b = b1.abs() * norm;
        bigint q, r;
        q.a.resize(a.a.size());

        for (int i = a.a.size() - 1; i >= 0; i--)
            {
            r *= base;
            r += a.a[i];
            int s1 = r.a.size() <= b.a.size() ? 0
                : r.a[b.a.size()];
            int s2 = r.a.size() <= b.a.size() - 1
                ? 0 : r.a[b.a.size() - 1];
            int d = ((long long) base * s1 + s2)
                / b.a.back();
            r -= b * d;
            while (r < 0)
                r += b, --d;
            q.a[i] = d;
        }

        q.sign = a1.sign * b1.sign;
        r.sign = a1.sign;
        q.trim();
        r.trim();
        return make_pair(q, r / norm);
    }

    bigint operator/(const bigint &v) const {
        return divmod(*this, v).first;
    }

    bigint operator%(const bigint &v) const {
        return divmod(*this, v).second;
    }

    void operator/=(int v) {
        if (v < 0)
            sign = -sign, v = -v;
        for (int i = (int) a.size() - 1, rem = 0;
            i >= 0; --i) {
            long long cur = a[i] + rem * (long
                long) base;
            a[i] = (int) (cur / v);
            rem = (int) (cur % v);
        }
        trim();
    }

    bigint operator/(int v) const {
        bigint res = *this;
        res /= v;
        return res;
    }

    int operator%(int v) const {
        if (v < 0)
            v = -v;
        int m = 0;
        for (int i = a.size() - 1; i >= 0; --i)
            m = (a[i] + m * (long long) base) % v
                ;
        return m * sign;
    }
```

```
    void operator+=(const bigint &v) {
        *this = *this + v;
    }
    void operator-=(const bigint &v) {
        *this = *this - v;
    }
    void operator*=(const bigint &v) {
        *this = *this * v;
    }
    void operator/=(const bigint &v) {
        *this = *this / v;
    }

    bool operator<(const bigint &v) const {
        if (sign != v.sign)
            return sign < v.sign;
        if (a.size() != v.a.size())
            return a.size() * sign < v.a.size() *
                 v.sign;
        for (int i = a.size() - 1; i >= 0; i--)
            if (a[i] != v.a[i])
                return a[i] * sign < v.a[i] *
                     sign;
        return false;
    }

    bool operator>(const bigint &v) const {
        return v < *this;
    }
    bool operator<=(const bigint &v) const {
        return !(v < *this);
    }
    bool operator>=(const bigint &v) const {
        return !(*this < v);
    }
    bool operator==(const bigint &v) const {
        return !(*this < v) && !(v < *this);
    }
    bool operator!=(const bigint &v) const {
        return *this < v || v < *this;
    }

    void trim() {
        while (!a.empty() && !a.back())
            a.pop_back();
        if (a.empty())
            sign = 1;
    }

    bool isZero() const {
        return a.empty() || (a.size() == 1 && !a
             [0]);
    }

    bigint operator-() const {
        bigint res = *this;
        res.sign = -sign;
        return res;
    }

    bigint abs() const {
        bigint res = *this;
        res.sign *= res.sign;
        return res;
    }

    long long longValue() const {
        long long res = 0;
        for (int i = a.size() - 1; i >= 0; i--)
            res = res * base + a[i];
        return res * sign;
    }

    friend bigint gcd(const bigint &a, const
         bigint &b) {
        return b.isZero() ? a : gcd(b, a % b);
    }
    friend bigint lcm(const bigint &a, const
         bigint &b) {
        return a / gcd(a, b) * b;
    }

    void read(const string &s) {
        sign = 1;
        a.clear();
        int pos = 0;
        while (pos < (int) s.size() && (s[pos] ==
             '-' || s[pos] == '+')) {
            if (s[pos] == '-')
                sign = -sign;
            ++pos;
        }
        for (int i = s.size() - 1; i >= pos; i -=
             base_digits) {
            int x = 0;
            for (int j = max(pos, i - base_digits
                 + 1); j <= i; j++)
                x = x * 10 + s[j] - '0';
            a.push_back(x);
        }
        trim();
    }

    friend istream& operator>>(istream &stream,
         bigint &v) {
        string s;
        stream >> s;
        v.read(s);
        return stream;
    }

    friend ostream& operator<<(ostream &stream,
         const bigint &v) {
        if (v.sign == -1)
            stream << '-';
        stream << (v.a.empty() ? 0 : v.a.back());
        for (int i = (int) v.a.size() - 2; i >=
             0; --i)
            stream << setw(base_digits) <<
                 setfill('0') << v.a[i];
        return stream;
    }

    static vector<int> convert_base(const vector<
         int> &a, int old_digits, int
         new_digits) {
        vector<long long> p(max(old_digits,
             new_digits) + 1);
        p[0] = 1;
        for (int i = 1; i < (int) p.size(); i++)
            p[i] = p[i - 1] * 10;
        vector<int> res;
        long long cur = 0;
        int cur_digits = 0;
        for (int i = 0; i < (int) a.size(); i++)
             {
            cur += a[i] * p[cur_digits];
            cur_digits += old_digits;
            while (cur_digits >= new_digits) {
                res.push_back(int(cur % p[
                     new_digits]));
                cur /= p[new_digits];
                cur_digits -= new_digits;
            }
        }
        res.push_back((int) cur);
        while (!res.empty() && !res.back())
            res.pop_back();
        return res;
    }

    typedef vector<long long> vll;

    static vll karatsubaMultiply(const vll &a,
         const vll &b) {
        int n = a.size();
        vll res(n + n);
        if (n <= 32) {
            for (int i = 0; i < n; i++)
                for (int j = 0; j < n; j++)
                    res[i + j] += a[i] * b[j];
            return res;
        }

        int k = n >> 1;
        vll a1(a.begin(), a.begin() + k);
        vll a2(a.begin() + k, a.end());
        vll b1(b.begin(), b.begin() + k);
        vll b2(b.begin() + k, b.end());

        vll a1b1 = karatsubaMultiply(a1, b1);
        vll a2b2 = karatsubaMultiply(a2, b2);

        for (int i = 0; i < k; i++)
            a2[i] += a1[i];
        for (int i = 0; i < k; i++)
            b2[i] += b1[i];

        vll r = karatsubaMultiply(a2, b2);
        for (int i = 0; i < (int) a1b1.size(); i
             ++)
            r[i] -= a1b1[i];
        for (int i = 0; i < (int) a2b2.size(); i
             ++)
            r[i] -= a2b2[i];

        for (int i = 0; i < (int) r.size(); i++)
            res[i + k] += r[i];
        for (int i = 0; i < (int) a1b1.size(); i
             ++)
            res[i] += a1b1[i];
        for (int i = 0; i < (int) a2b2.size(); i
             ++)
            res[i + n] += a2b2[i];
        return res;
    }

    bigint operator*(const bigint &v) const {
        vector<int> a6 = convert_base(this->a,
             base_digits, 6);
        vector<int> b6 = convert_base(v.a,
             base_digits, 6);
        vll a(a6.begin(), a6.end());
        vll b(b6.begin(), b6.end());
        while (a.size() < b.size())
            a.push_back(0);
        while (b.size() < a.size())
            b.push_back(0);
        while (a.size() & (a.size() - 1))
            a.push_back(0), b.push_back(0);
        vll c = karatsubaMultiply(a, b);
        bigint res;
        res.sign = sign * v.sign;
        for (int i = 0, carry = 0; i < (int) c.
             size(); i++) {
            long long cur = c[i] + carry;
            res.a.push_back((int) (cur % 1000000)
                 );
            carry = (int) (cur / 1000000);
        }
        res.a = convert_base(res.a, 6,
             base_digits);
        res.trim();
        return res;
    }
};
```

51 DSU

```
struct DSU {
    vector<int> e;
    DSU(int N) { e = vector<int>(N, -1); }

    // get representive component (uses path
         compression)
    int get(int x) { return e[x] < 0 ? x : e[x] =
         get(e[x]); }

    bool same_set(int a, int b) { return get(a) ==
         get(b); }

    int size(int x) { return -e[get(x)]; }

    bool unite(int x, int y) {  // union by size
        x = get(x), y = get(y);
        if (x == y) return false;
        if (e[x] > e[y]) swap(x, y);
        e[x] += e[y]; e[y] = x;
        return true;
    }
};
```

52 Mod

```
//
     -----------------------------------------------//
//                Auto mod number
     //
//
     -----------------------------------------------//

template<ll M>
struct modint {

    static ll _pow(ll n, ll k) {
        ll r = 1;
        for (; k > 0; k >>= 1, n = (n*n)%M)
            if (k&1) r = (r*n)%M;
        return r;
    }

    ll v; modint(ll n = 0) : v(n%M) { v += (M
         &(0-(v<0))); }

    friend string to_string(const modint n) {
         return to_string(n.v); }
    friend istream& operator>>(istream& i, modint
         & n) { return i >> n.v; }
    friend ostream& operator<<(ostream& o, const
         modint n) { return o << n.v; }
    template<typename T> explicit operator T() {
         return T(v); }

    friend bool operator==(const modint n, const
         modint m) { return n.v == m.v; }
    friend bool operator!=(const modint n, const
         modint m) { return n.v != m.v; }
    friend bool operator<(const modint n, const
         modint m) { return n.v < m.v; }
    friend bool operator<=(const modint n, const
         modint m) { return n.v <= m.v; }
    friend bool operator>(const modint n, const
         modint m) { return n.v > m.v; }
    friend bool operator>=(const modint n, const
         modint m) { return n.v >= m.v; }

    modint& operator+=(const modint n) { v += n.v
         ; v -= (M&(0-(v>=M))); return *this; }
    modint& operator-=(const modint n) { v -= n.v
         ; v += (M&(0-(v<0))); return *this; }
    modint& operator*=(const modint n) { v = (v*n
         .v)%M; return *this; }
    modint& operator/=(const modint n) { v = (v*
         _pow(n.v, M-2))%M; return *this; }
    friend modint operator+(const modint n, const
         modint m) { return modint(n) += m; }
    friend modint operator-(const modint n, const
         modint m) { return modint(n) -= m; }
    friend modint operator*(const modint n, const
         modint m) { return modint(n) *= m; }
    friend modint operator/(const modint n, const
         modint m) { return modint(n) /= m; }
    modint& operator++() { return *this += 1; }
    modint& operator--() { return *this -= 1; }
    modint operator++(int) { modint t = *this;
         return *this += 1, t; }
    modint operator--(int) { modint t = *this;
         return *this -= 1, t; }
    modint operator+() { return *this; }
    modint operator-() { return modint(0) -= *
         this; }

    // O(logk) modular exponentiation
    modint pow(const ll k) const {
        return k < 0 ? _pow(v, M-1-(-k%(M-1))) :
             _pow(v, k);
    }
    modint inv() const { return _pow(v, M-2); }
};

using mod = modint<998244353>;
//
     -----------------------------------------------//
//                    Factorial
     //
//
     -----------------------------------------------//

struct fac{
    int n; ll mod_fac; vector<ll> f;
```

```
    fac(){}
    fac(ll _mod_fac){ n = 0; f = vector<ll>(1); f[0] = 1; mod_fac = _mod_fac; }
    fac(ll _mod_fac, int _n){ n = _n; f = vector<ll>(n + 1, 0); f[0] = 1; mod_fac = _mod_fac; }
    ll get(int index){ if(index <= n) return f[index]; for(int i = n + 1; i<=index; i++){ f.push_back(f.back()*i%mod_fac); n++; } return f[index]; }
} ft;

//----------------------------------------------//
//                 Math prox                    //
//----------------------------------------------//

ll pow_hota(ll ax, ll bx){
    ll cx = 1;
    while(bx > 0) {
      if(bx & 1)
        cx = 1ll * cx * ax % mod;
      ax = 1ll * ax * ax % mod;
      bx >>= 1;
    }
    return cx;
}

ll divmod(ll ax, ll bx){
    ax%=mod; bx%=mod;
    return (ax*pow_hota(bx, mod-2))%mod;
}

ll Cx(ll _n, ll _k){
    return divmod(ft.get(_n), ft.get(_k)*ft.get(_n - _k)%mod)%mod;
}

//----------------------------------------------//
//                 LCM MOD                      //
//----------------------------------------------//

const int MAX = 200000;
int prime[MAX];
unordered_map<int, int> max_map;
ll pow_hota(ll ax, ll bx){
    ll cx = 1;
    while(bx > 0) {
      if(bx & 1)
        cx = 1ll * cx * ax % mod;
      ax = 1ll * ax * ax % mod;
      bx >>= 1;
    }
    return cx;
}
void sieve()
{
    prime[0] = prime[1] = 1;
    for (int i = 2; i < MAX; i++) {
        if (prime[i] == 0) {
            for (int j = i * 2; j < MAX; j += i)
            {
                if (prime[j] == 0) {
                    prime[j] = i;
                }
            }
            prime[i] = i;
        }
    }
}
// LCM MOD
ll lcmModuloM(const int* ar, int n){
    for (int i = 0; i < n; i++) {
        int num = ar[i];
        unordered_map<int, int> temp;
        while (num > 1) {
            int factor = prime[num];
            temp[factor]++;
            num /= factor;
        }

        for (auto it : temp) {
            max_map[it.first] = max(max_map[it.first], it.second);
        }
    }
    ll ans = 1;
    for (auto it : max_map) {
        ans = (ans * pow_hota(it.F, it.S)) % mod;
    }
    return ans;
}
```

53 Tarjan(SCC)

```
int Num[200005], Low[200005], Time = 0;
int Count = 0;
stack<int> st;

void tarjan(int u) {
    Low[u] = Num[u] = ++Time;
    st.push(u);
    for (int v : ed[u]){
        if (Num[v] != 0)
            Low[u] = min(Low[u], Num[v]);
        else {
            tarjan(v);
            Low[u] = min(Low[u], Low[v]);
        }
    }
    if (Num[u] == Low[u]) {  // found one
        Count++;
        int v;
        do {
            v = st.top();
            st.pop();
            Num[v] = Low[v] = ooii;  // remove v from graph
        } while (v != u);
    }
}
```

54 Matrix

```
//----------------------------------------------//
//                 Matrix                       //
//----------------------------------------------//

#define ele_maxtrix ll
struct Matrix{
  int n, m;
  vector<vector<ele_maxtrix>> ma;
  Matrix(){}
  Matrix(int _n){
    n = _n; m = _n;
    ele_maxtrix ei;
    ei.set(0,0);
    ma = vector<vector<ele_maxtrix>>(n, vector<ele_maxtrix>(m, ei));
    for(int i = 0; i<n; i++){
      ma[i][i].set(0, 1);
    }
  }
  Matrix(int _n, int _m){
    n = _n; m = _m;
    ele_maxtrix ei;
    ei.set(0,0);
    ma = vector<vector<ele_maxtrix>>(n, vector<ele_maxtrix>(m, ei));
  }
};
Matrix operator*(Matrix ax, Matrix bx){
  if(ax.m != bx.n) return Matrix(0,0);
  Matrix cx = Matrix(ax.n, bx.m);
  for(int i = 0; i<ax.n; i++){
    for(int j = 0; j<bx.m; j++){
      for(int c = 0; c<ax.m; c++){
        cx.ma[i][j] = cx.ma[i][j] + (ax.ma[i][c]*bx.ma[c][j]);
      }
    }
  }
  return cx;
}
void print_matrix(Matrix ax){
  for(int i = 0; i<ax.n; i++){
    for(int j = 0; j<ax.m; j++){
      cout << ax.ma[i][j] << " ";
    }
    cout << "\n";
  }
}
```

55 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.

56 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

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

.

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:

$$\begin{aligned} \mathbf{A}\mathbf{v} &= \lambda\mathbf{v} \\ \mathbf{A}\mathbf{Q} &= \mathbf{Q}\mathbf{\Lambda} \\ \mathbf{A} &= \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{-1}. \end{aligned}$$

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:

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

57 Generating function

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

58 Partition

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

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

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

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

59 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.)

60 Fibonacci mod $10^9 + 9$

$$F_n \equiv 276601605(691504013^n - 308495997^n)$$

$$F_n = \frac{\varphi^n - \psi^n}{\varphi - \psi} = \frac{\varphi^n - \psi^n}{\sqrt{5}}$$

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

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

$$\begin{aligned} F_m F_n + F_{m-1} F_{n-1} &= F_{m+n-1}, \\ F_m F_{n+1} + F_{m-1} F_n &= F_{m+n}. \end{aligned}$$

In particular, with $m = n$,

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

61 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

62 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.

63 Pell equation

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

If x_1, y_1 is the minimal solution then:

$$\begin{aligned} x_{k+1} &= x_1 x_k + n y_1 y_k, \\ y_{k+1} &= x_1 y_k + y_1 x_k. \end{aligned}$$

64 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 $|X/G|$:

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

65 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

$$\mathrm{B}(x,y) = \int_0^1 t^{x-1}(1-t)^{y-1}\, dt$$

$$\mathrm{B}(x,y) = \mathrm{B}(y,x)$$

$$\mathrm{B}(x,y) = \frac{\Gamma(x)\,\Gamma(y)}{\Gamma(x+y)}.$$

$$\Gamma(x)\Gamma(y) = \int_{\mathbb{R}} f(u)\, du \cdot \int_{\mathbb{R}} g(u)\, du = \int_{\mathbb{R}} (f *$$

66 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:

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

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

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

67 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 $i \ge 1$ 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.

68 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}.

69 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 $1 \leq k \leq n$:

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

Gale–Ryser theorem

A pair of sequences of nonnegative integers $(a_1, \ldots, a_n)$ and $(b_1, \ldots, b_n)$ with $a_1 \geq \cdots \geq a_n$ 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 $1 \leq k \leq n$:

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

70 Binomial coefficient

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

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

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

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

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

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

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

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

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

71 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.

72 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.

73 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**

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

74 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 ($|S| \le k$).
- **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 $M_2 = <X_2, I_2>$

75 Equations

$$ax^2 + bx + c = 0 \Rightarrow x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The extremum is given by $x = -b/2a$.

$$\begin{aligned} ax + by &= e \\ cx + dy &= f \end{aligned} \Rightarrow \begin{aligned} x &= \frac{ed - bf}{ad - bc} \\ y &= \frac{af - ec}{ad - bc} \end{aligned}$$

In general, given an equation $Ax = b$, the solution to a variable x_i is given by

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

where A_i' is A with the i'th column replaced by b.

76 Recurrences

If $a_n = c_1 a_{n-1} + \cdots + c_k a_{n-k}$, and $r_1, \ldots, r_k$ are distinct roots of $x^k + c_1 x^{k-1} + \cdots + c_k$, there are $d_1, \ldots, d_k$ s.t.

$$a_n = d_1 r_1^n + \cdots + d_k r_k^n.$$

Non-distinct roots r become polynomial factors, e.g. $a_n = (d_1 n + d_2) r^n$.

77 Trigonometry

$$\begin{aligned} \sin(v + w) &= \sin v \cos w + \cos v \sin w \\ cos(v + w) &= \cos v \cos w - \sin v \sin w \end{aligned}$$

$$\begin{aligned} \tan(v + w) &= \frac{\tan v + \tan w}{1 - \tan v \tan w} \\ \sin v + \sin w &= 2 \sin \frac{v + w}{2} \cos \frac{v - w}{2} \\ \cos v + \cos w &= 2 \cos \frac{v + w}{2} \cos \frac{v - w}{2} \end{aligned}$$

$$(V + W) \tan(v - w)/2 = (V - W) \tan(v + w)/2$$

where V, W are lengths of sides opposite angles v, w.

$$\begin{aligned} a \cos x + b \sin x &= r \cos(x - \phi) \\ a \sin x + b \cos x &= r \sin(x + \phi) \end{aligned}$$

where $r = \sqrt{a^2 + b^2}, \phi = \operatorname{atan2}(b, a)$.

78 Geometry

78.1 Triangles

Side lengths: a, b, c

Semiperimeter: $p = \dfrac{a + b + c}{2}$

Area: $A = \sqrt{p(p - a)(p - b)(p - c)}$

Circumradius: $R = \dfrac{abc}{4A}$

Inradius: $r = \dfrac{A}{p}$

Length of median (divides triangle into two equal-area triangles): $m_a = \frac{1}{2}\sqrt{2b^2 + 2c^2 - a^2}$

Length of bisector (divides angles in two): $s_a = \sqrt{bc\left[1 - \left(\dfrac{a}{b + c}\right)^2\right]}$

Law of sines: $\dfrac{\sin \alpha}{a} = \dfrac{\sin \beta}{b} = \dfrac{\sin \gamma}{c} = \dfrac{1}{2R}$

Law of cosines: $a^2 = b^2 + c^2 - 2bc \cos \alpha$

Law of tangents: $\dfrac{a + b}{a - b} = \dfrac{\tan \dfrac{\alpha + \beta}{2}}{\tan \dfrac{\alpha - \beta}{2}}$

78.2 Quadrilaterals

With side lengths a, b, c, d, diagonals e, f, diagonals angle θ, area A and magic flux $F = b^2 + d^2 - a^2 - c^2$:

$$4A = 2ef \cdot \sin \theta = F \tan \theta = \sqrt{4e^2 f^2 - F^2}$$

For cyclic quadrilaterals the sum of opposite angles is 180°, $ef = ac + bd$, and $A = \sqrt{(p - a)(p - b)(p - c)(p - d)}$.

78.3 Spherical coordinates

$$\begin{aligned} x &= r \sin \theta \cos \phi & r &= \sqrt{x^2 + y^2 + z^2} \\ y &= r \sin \theta \sin \phi & \theta &= \operatorname{acos}(z/\sqrt{x^2 + y^2 + z^2}) \\ z &= r \cos \theta & \phi &= \operatorname{atan2}(y, x) \end{aligned}$$

79 Derivatives/Integrals

$$\frac{d}{dx} \arcsin x = \frac{1}{\sqrt{1 - x^2}} \qquad \frac{d}{dx} \arccos x = -\frac{}{\sqrt{1}}$$

$$\frac{d}{dx} \tan x = 1 + \tan^2 x \qquad \frac{d}{dx} \arctan x = \frac{1}{1 + x}$$

$$\int \tan ax = -\frac{\ln |\cos ax|}{a} \qquad \int x \sin ax = \frac{\sin a}{}$$

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

Integration by parts:

$$\int_a^b f(x) g(x) dx = [F(x) g(x)]_a^b - \int_a^b F(x) g'($$

80 Sums

$$c^a + c^{a+1} + \cdots + c^b = \frac{c^{b+1} - c^a}{c-1}, c \neq 1$$

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$

$$1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(2n+1)(n+1)}{6}$$

$$1^3 + 2^3 + 3^3 + \cdots + n^3 = \frac{n^2(n+1)^2}{4}$$

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

81 Series

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots, \ (-\infty < x < \infty)$$

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots, \ (-1 < x \leq 1)$$

$$\sqrt{1+x} = 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{2x^3}{32} - \frac{5x^4}{128} + \ldots, \ (-1 \leq x \leq 1)$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots, \ (-\infty < x < \infty)$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots, \ (-\infty < x < \infty)$$

82 Probability theory

Let X be a discrete random variable with probability $p_X(x)$ of assuming the value x. It will then have an expected value (mean) $\mu = \mathbb{E}(X) = \sum_x x p_X(x)$ and variance $\sigma^2 = V(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2 = \sum_x (x - \mathbb{E}(X))^2 p_X(x)$ where σ is the standard deviation. If X is instead continuous it will have a probability density function $f_X(x)$ and the sums above will instead be integrals with $p_X(x)$ replaced by $f_X(x)$.

Expectation is linear:

$$\mathbb{E}(aX + bY) = a\mathbb{E}(X) + b\mathbb{E}(Y)$$

For independent X and Y,

$$V(aX + bY) = a^2 V(X) + b^2 V(Y).$$

82.1 Discrete distributions

82.1.1 Binomial distribution

The number of successes in n independent yes/no experiments, each which yields success with probability p is $\mathrm{Bin}(n,p)$, $n = 1, 2, \ldots$, $0 \leq p \leq 1$.

$$p(k) = \binom{n}{k} p^k (1-p)^{n-k}$$

$$\mu = np, \ \sigma^2 = np(1-p)$$

$\mathrm{Bin}(n,p)$ is approximately $\mathrm{Po}(np)$ for small p.

82.1.2 First success distribution

The number of trials needed to get the first success in independent yes/no experiments, each wich yields success with probability p is $\mathrm{Fs}(p)$, $0 \leq p \leq 1$.

$$p(k) = p(1-p)^{k-1}, \ k = 1, 2, \ldots$$

$$\mu = \frac{1}{p}, \ \sigma^2 = \frac{1-p}{p^2}$$

82.1.3 Poisson distribution

The number of events occurring in a fixed period of time t if these events occur with a known average rate κ and independently of the time since the last event is $\mathrm{Po}(\lambda)$, $\lambda = t\kappa$.

$$p(k) = e^{-\lambda} \frac{\lambda^k}{k!}, k = 0, 1, 2, \ldots$$

$$\mu = \lambda, \ \sigma^2 = \lambda$$

82.2 Continuous distributions

82.2.1 Uniform distribution

If the probability density function is constant between a and b and 0 elsewhere it is $\mathrm{U}(a,b)$, $a < b$.

$$f(x) = \begin{cases} \frac{1}{b-a} & a < x < b \\ 0 & \text{otherwise} \end{cases}$$

$$\mu = \frac{a+b}{2}, \ \sigma^2 = \frac{(b-a)^2}{12}$$

82.2.2 Exponential distribution

The time between events in a Poisson process is $\mathrm{Exp}(\lambda)$, $\lambda > 0$.

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

$$\mu = \frac{1}{\lambda}, \ \sigma^2 = \frac{1}{\lambda^2}$$

82.2.3 Normal distribution

Most real random values with mean μ and variance σ^2 are well described by $\mathcal{N}(\mu, \sigma^2)$, $\sigma > 0$.

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

If $X_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $X_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$ then

$$aX_1 + bX_2 + c \sim \mathcal{N}(\mu_1 + \mu_2 + c, a^2\sigma_1^2 + b^2\sigma_2^2)$$

83 Markov chains

A *Markov chain* is a discrete random process with the property that the next state depends only on the current state. Let $X_1, X_2, \ldots$ be a sequence of random variables generated by the Markov process. Then there is a transition matrix $\mathbf{P} = (p_{ij})$, with $p_{ij} = \Pr(X_n = i | X_{n-1} = j)$, and $\mathbf{p}^{(n)} = \mathbf{P}^n \mathbf{p}^{(0)}$ is the probability distribution for X_n (i.e., $p_i^{(n)} = \Pr(X_n = i)$), where $\mathbf{p}^{(0)}$ is the initial distribution.

π is a stationary distribution if $\pi = \pi\mathbf{P}$. If the Markov chain is *irreducible* (it is possible to get to any state from any state), then $\pi_i = \frac{1}{\mathbb{E}(T_i)}$ where $\mathbb{E}(T_i)$ is the expected time between two visits in state i. π_j / π_i is the expected number of visits in state j between two visits in state i.

For a connected, undirected and non-bipartite graph, where the transition probability is uniform among all neighbors, π_i is proportional to node i's degree.

A Markov chain is *ergodic* if the asymptotic distribution is independent of the initial distribution. A finite Markov chain is ergodic iff it is irreducible and *aperiodic* (i.e., the gcd of cycle lengths is 1). $\lim_{k\to\infty} \mathbf{P}^k = \mathbf{1}\pi$.

A Markov chain is an A-chain if the states can be partitioned into two sets $\mathbf{A}$ and $\mathbf{G}$, such that all states in $\mathbf{A}$ are absorbing ($p_{ii} = 1$), and all states in $\mathbf{G}$ leads to an absorbing state in $\mathbf{A}$. The probability for absorption in state $i \in \mathbf{A}$, when the initial state is j, is $a_{ij} = p_{ij} + \sum_{k\in\mathbf{G}} a_{ik}p_{kj}$. The expected time until absorption, when the initial state is i, is $t_i = 1 + \sum_{k\in\mathbf{G}} p_{ki}t_k$.