Contents

1 Java

```
// BigInteger
gcd, modPow, modInverse, not, and, or, xor, andNot, shiftLeft(int), shiftRight(
    int), clearBit(int), setBit(n), flipBit(n), testBit(n), bitBount, bitLength
    , getLowestSetBit, equals, signum, doubleValue, intValue, longValue,
    toString(base)
// Regex
import java.util.regex.*
Pattern pattern = Pattern.compile(regex);
Matcher matcher = pattern.matcher(string);
Matcher functions:
matches(), find(), find(start), group() // returns input subsequence matched by
     previous match (find?), replaceAll(), replaceFirst()
// DecimalFormat
import java.text.*
DecimalFormat df = new DecimalFormat("0.00###E0"); // #=skip if possible
```

2 HexagonalGrid

```
int roundCount(int round) {
    return (6*round);
}
int roundSum(int round) {
    return (6*round*(round+1)/2);
}
int findRound(int n) {
    int res=1;
    while (roundSum(res)<n) res++;
    return (res);
}
pair<int,int> cord(int n) {
    if (n==0) return (make_pair(0,0));
    int c=findRound(n);
    int prev=roundSum(c-1);
    if (n<=prev+c) return (make_pair(c,n-prev));
    if (n<=prev+2*c) return (make_pair(prev+2*c-n,c));
    if (n<=prev+3*c) return (make_pair(prev+2*c-n,prev+3*c-n));
    if (n<=prev+4*c) return (make_pair(-c,prev+3*c-n));
    if (n<=prev+5*c) return (make_pair(n-prev-5*c,-c));
    return (make_pair(n-prev-5*c,n-prev-6*c));
}
bool inRound(int x,int y,int c) {
    if (0<=y && y<=c && x==c) return (true);
    if (0<=x && x<=c && y==c) return (true);
    if (0<=y && y<=c && y-x==c) return (true);
    if (-c<=y && y<=0 && x==-c) return (true);
    if (-c<=x && x<=0 && y==-c) return (true);
    if (0<=x && x<=c && x-y==c) return (true);
    return (false);
}
int findRound(int x,int y) {
    int res=1;
    while (!inRound(x,y,res)) res++;
    return (res);
}
int number(int x,int y) {
    if (x==0 && y==0) return (0);
    int c=findRound(x,y);
    int prev=roundSum(c-1);
    if (1<=y && y<=c && x==c) return (prev+y);
    if (0<=x && x<=c && y==c) return (prev+2*c-x);
    if (0<=y && y<=c && y-x==c) return (prev+2*c-x);
    if (-c<=y && y<=0 && x==-c) return (prev+3*c-y);
    if (-c<=x && x<=0 && y==-c) return (prev+5*c+x);
    return (prev+5*c+x);
}
```

3 GaussJordan

```
// Gauss-Jordan elimination with full pivoting.
//
// Uses:
//   (1) solving systems of linear equations (AX=B)
//   (2) inverting matrices (AX=I)
//   (3) computing determinants of square matrices
//
// Running time: O(n^3)
//
// INPUT:    a[][] = an nxn matrix
//           b[][] = an nxm matrix
//
// OUTPUT:   X      = an nxm matrix (stored in b[][])
//           A^{-1} = an nxn matrix (stored in a[][])
//           returns determinant of a[][]

const double EPS = 1e-10;

typedef vector<int> VI;
typedef double T;
typedef vector<T> VT;
typedef vector<VT> VVT;

T GaussJordan(VVT &a, VVT &b) {
    const int n = a.size();
    const int m = b[0].size();
    VI irow(n), icol(n), ipiv(n);
    T det = 1;
    for (int i = 0; i < n; i++) {
        int pj = -1, pk = -1;
        for (int j = 0; j < n; j++) if (!ipiv[j])
            for (int k = 0; k < n; k++) if (!ipiv[k])
                if (pj == -1 || fabs(a[j][k]) > fabs(a[pj][pk])) { pj = j; pk =
                    k; }
        if (fabs(a[pj][pk]) < EPS) { cerr << "Matrix is singular." << endl;
            exit(0); }
        ipiv[pk]++;
        swap(a[pj], a[pk]);
        swap(b[pj], b[pk]);
        if (pj != pk) det *= -1;
        irow[i] = pj;
        icol[i] = pk;
        T c = 1.0 / a[pk][pk];
        det *= a[pk][pk];
        a[pk][pk] = 1.0;
        for (int p = 0; p < n; p++) a[pk][p] *= c;
        for (int p = 0; p < m; p++) b[pk][p] *= c;
        for (int p = 0; p < n; p++) if (p != pk) {
            c = a[p][pk];
            a[p][pk] = 0;
            for (int q = 0; q < n; q++) a[p][q] -= a[pk][q] * c;
            for (int q = 0; q < m; q++) b[p][q] -= b[pk][q] * c;
        }
    }
    for (int p = n-1; p >= 0; p--) if (irow[p] != icol[p]) {
        for (int k = 0; k < n; k++) swap(a[k][irow[p]], a[k][icol[p]]);
    }
    return det;
}

int main() {
    const int n = 4, m = 2;
    double A[n][n] = { {1,2,3,4},{1,0,1,0},{5,3,2,4},{6,1,4,6} };
    double B[n][m] = { {1,2},{4,3},{5,6},{8,7} };
    VVT a(n), b(n);
    for (int i = 0; i < n; i++) {
        a[i] = VT(A[i], A[i] + n);
        b[i] = VT(B[i], B[i] + m);
    }
    double det = GaussJordan(a, b);
    cout << "Determinant: " << det << endl; // expected: 60
    // expected: -0.233333 0.166667 0.133333 0.0666667
    //           0.166667 0.166667 0.333333 -0.333333
    //           0.233333 0.833333 -0.133333 -0.0666667
    //           0.05 -0.75 -0.1 0.2
    cout << "Inverse: " << endl;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
            cout << a[i][j] << ' ';
        cout << endl;
    }
    // expected: 1.63333 1.3
    //           -0.166667 0.5
    //           2.36667 1.7
    //           -1.85 -1.35
    cout << "Solution: " << endl;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++)
            cout << b[i][j] << ' ';
        cout << endl;
    }
}
```

4 Simplex

```
// Two-phase simplex algorithm for solving linear programs of the form
//
//     maximize     c^T x
//     subject to   Ax <= b
//                  x >= 0
//
// INPUT: A -- an m x n matrix
//        b -- an m-dimensional vector
//        c -- an n-dimensional vector
//        x -- a vector where the optimal solution will be stored
//
// OUTPUT: value of the optimal solution (infinity if unbounded
//         above, nan if infeasible)
//
// To use this code, create an LPSolver object with A, b, and c as
// arguments.  Then, call Solve(x).

typedef long double DOUBLE;
typedef vector<DOUBLE> VD;
typedef vector<VD> VVD;
typedef vector<int> VI;

const DOUBLE EPS = 1e-9;

struct LPSolver {
    int m, n;
    VI B, N;
    VVD D;

    LPSolver(const VVD &A, const VD &b, const VD &c) :
        m(b.size()), n(c.size()), B(m), N(n + 1), D(m + 2, VD(n + 2)) {
            for (int i = 0; i < m; i++) for (int j = 0; j < n; j++) D[i][j] = A
                [i][j];
            for (int i = 0; i < m; i++) { B[i] = n + i; D[i][n] = -1; D[i][n +
                1] = b[i]; }
            for (int j = 0; j < n; j++) { N[j] = j; D[m][j] = -c[j]; }
            N[n] = -1; D[m + 1][n] = 1;
        }

    void Pivot(int r, int s) {
        for (int i = 0; i < m + 2; i++) if (i != r)
            for (int j = 0; j < n + 2; j++) if (j != s)
                D[i][j] -= D[r][j] * D[i][s] / D[r][s];
        for (int j = 0; j < n + 2; j++) if (j != s) D[r][j] /= D[r][s];
        for (int i = 0; i < m + 2; i++) if (i != r) D[i][s] /= -D[r][s];
        D[r][s] = 1.0 / D[r][s];
        swap(B[r], N[s]);
    }

    bool Simplex(int phase) {
        int x = phase == 1 ? m + 1 : m;
        while (true) {
            int s = -1;
            for (int j = 0; j <= n; j++) {
                if (phase == 2 && N[j] == -1) continue;
                if (s == -1 || D[x][j] < D[x][s] || (D[x][j] == D[x][s] && N[j]
                     < N[s])) s = j;
            }
            if (D[x][s] > -EPS) return true;
            int r = -1;
            for (int i = 0; i < m; i++) {
                if (D[i][s] < EPS) continue;
                if (r == -1 || D[i][n + 1] / D[i][s] < D[r][n + 1] / D[r][s] ||
                     ((D[i][n + 1] / D[i][s]) == (D[r][n + 1] / D[r][s]) && B[i
                    ] < B[r])) r = i;
            }
            if (r == -1) return false;
            Pivot(r, s);
        }
    }

    DOUBLE Solve(VD &x) {
        int r = 0;
        for (int i = 1; i < m; i++) if (D[i][n + 1] < D[r][n + 1]) r = i;
        if (D[r][n + 1] < -EPS) {
            Pivot(r, n);
            if (!Simplex(1) || D[m + 1][n + 1] < -EPS) return -numeric_limits<
                DOUBLE>::infinity();
            for (int i = 0; i < m; i++) if (B[i] == -1) {
                int s = -1;
                for (int j = 0; j <= n; j++)
                    if (s == -1 || D[i][j] < D[i][s] || (D[i][j] == D[i][s] &&
                        N[j] < N[s])) s = j;
                Pivot(i, s);
            }
        }
        if (!Simplex(2)) return numeric_limits<DOUBLE>::infinity();
        x = VD(n);
        for (int i = 0; i < m; i++) if (B[i] < n) x[B[i]] = D[i][n + 1];
        return D[m][n + 1];
    }
};

int main() {

    const int m = 4;
    const int n = 3;
    DOUBLE _A[m][n] = {
        { 6, -1, 0 },
        { -1, -5, 0 },
        { 1, 5, 1 },
        { -1, -5, -1 }
    };
    DOUBLE _b[m] = { 10, -4, 5, -5 };
    DOUBLE _c[n] = { 1, -1, 0 };

    VVD A(m);
    VD b(_b, _b + m);
    VD c(_c, _c + n);
    for (int i = 0; i < m; i++) A[i] = VD(_A[i], _A[i] + n);

    LPSolver solver(A, b, c);
    VD x;
    DOUBLE value = solver.Solve(x);

    cerr << "VALUE: " << value << endl; // VALUE: 1.29032
    cerr << "SOLUTION:"; // SOLUTION: 1.74194 0.451613 1
    for (size_t i = 0; i < x.size(); i++) cerr << " " << x[i];
    cerr << endl;
    return 0;
}
```

5 ReducedRowEchelonForm

```
// Reduced row echelon form via Gauss-Jordan elimination
// with partial pivoting.  This can be used for computing
// the rank of a matrix.
//
// Running time: O(n^3)
//
// INPUT:    a[][] = an nxm matrix
//
// OUTPUT:   rref[][] = an nxm matrix (stored in a[][])
//           returns rank of a[][]

const double EPSILON = 1e-10;

typedef double T;
typedef vector<T> VT;
typedef vector<VT> VVT;

int rref(VVT &a) {
    int n = a.size();
    int m = a[0].size();
    int r = 0;
    for (int c = 0; c < m && r < n; c++) {
        int j = r;
        for (int i = r + 1; i < n; i++)
            if (fabs(a[i][c]) > fabs(a[j][c])) j = i;
        if (fabs(a[j][c]) < EPSILON) continue;
        swap(a[j], a[r]);

        T s = 1.0 / a[r][c];
        for (int j = 0; j < m; j++) a[r][j] *= s;
        for (int i = 0; i < n; i++) if (i != r) {
            T t = a[i][c];
            for (int j = 0; j < m; j++) a[i][j] -= t * a[r][j];
        }
        r++;
    }
    return r;
}

int main() {
    const int n = 5, m = 4;
    double A[n][m] = {
        {16,  2,  3, 13},
        { 5, 11, 10,  8},
        { 9,  7,  6, 12},
        { 4, 14, 15,  1},
        {13, 21, 21, 13}};
    VVT a(n);
    for (int i = 0; i < n; i++)
        a[i] = VT(A[i], A[i] + m);

    int rank = rref(a);

    // expected: 3
    cout << "Rank: " << rank << endl;

    // expected: 1 0 0 1
    //           0 1 0 3
    //           0 0 1 -3
    //           0 0 0 3.10862e-15
    //           0 0 0 2.22045e-15
    cout << "rref: " << endl;
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 4; j++)
            cout << a[i][j] << ' ';
        cout << endl;
    }
}
```

6 FFT

```
typedef complex<double> Complex;

template<class T> int size(const T &a) {
    return a.size();
}

unsigned roundUp(unsigned v) {
    --v;
    v |= v >> 1;
    v |= v >> 2;
    v |= v >> 4;
    v |= v >> 8;
    v |= v >> 16;
    return v + 1;
}

int reverse(int num, int lg) {
    int res = 0;
    for(int i = 0; i < lg; ++i) if(num & 1 << i)
        res |= 1 << (lg - i - 1);
    return res;
}

template<class T> ostream& operator << (ostream& out, const vector<T> &a) {
    for(int i = 0; i < size(a); ++i) {
        if(i > 0) out << ' ';
        out << a[i];
    }
    return out;
}

vector<Complex> fft(vector<Complex> a, bool invert) {
    int n = size(a), lg = 0;
    while(1 << lg < n) ++lg;
    vector<Complex> roots (n);
    for(int i = 0; i < n; ++i) {
        double alpha = 2 * M_PI / n * i * (invert ? -1 : 1);
        roots[i] = Complex(cos(alpha), sin(alpha));
    }
    for(int i = 0; i < n; ++i) {
        int rev = reverse(i, lg);
        if(i < rev) swap(a[i], a[rev]);
    }
    for(int len = 2; len <= n; len <<= 1)
        for(int i = 0; i < n; i += len)
            for(int j = 0; j < len >> 1; ++j) {
                Complex u = a[i + j], v = a[i + j + (len >> 1)] * roots[n / len
                     * j];
                a[i + j] = u + v;
                a[i + j + (len >> 1)] = u - v;
            }
    if(invert) for(int i = 0; i < n; ++i) a[i] /= n;
    return a;
}

vector<long long> multiply(const vector<int> &a, const vector<int> &b) {
    int n = roundUp(size(a) + size(b) - 1);
    vector<Complex> pa (n), pb (n);
    for(int i = 0; i < size(a); ++i) pa[i] = a[i];
    for(int i = 0; i < size(b); ++i) pb[i] = b[i];
    pa = fft(pa, false); pb = fft(pb, false);
    for(int i = 0; i < n; ++i) pa[i] *= pb[i];
    pa = fft(pa, true);
    vector<long long> res (n);
    for(int i = 0; i < n; ++i) res[i] = round(real(pa[i]));
    return res;
}
```

7 FFTMod

```
const int MODULO = 998244353;
const int ROOT = 3; // Primitive root

void fft(vector<int> &a, bool invert) {
    int n = a.size();
    assert((n & (n - 1)) == 0);
    int lg = __builtin_ctz(n);
    for (int i = 0; i < n; ++i) {
        int j = 0;
        for (int k = 0; k < lg; ++k) if ((i&1<<k)!=0) j |= 1 << (lg-k-1);
        if (i < j) swap(a[i], a[j]);
    }
    for (int len = 2; len <= n; len *= 2) {
        int wlen = power(ROOT, (MODULO - 1) / len);
        if (invert) wlen = inverse(wlen);
        for (int i = 0; i < n; i += len) {
            int w = 1;
            for (int j = 0; j < len / 2; ++j) {
                int u = a[i + j];
                int v = 1LL * a[i + j + len / 2] * w % MODULO;
                a[i + j] = (u + v) % MODULO;
                a[i + j + len / 2] = (u - v + MODULO) % MODULO;
                w = 1LL * w * wlen % MODULO;
            }
        }
    }
    if (invert) {
        int mul = inverse(n);
        for (auto &x : a) x = 1LL * x * mul % MODULO;
    }
}
```

8 EulerTotient

```
int phi[n];
for (int i = 0; i < n; i++) phi[i] = i;
for (int i = 1; i < n; i++)
        for (int j = 2 * i; j < n; j += i)
                phi[j] -= phi[i];
```

9 Euclid

```
// This is a collection of useful code for solving problems that
// involve modular linear equations.  Note that all of the
// algorithms described here work on nonnegative integers.

typedef vector<int> VI;
typedef pair<int,int> PII;

int mod(int a, int b) { // return a % b (positive value)
    return ((a%b)+b)%b;
}

int gcd(int a, int b) { // computes gcd(a,b)
    int tmp;
    while(b){a%=b; tmp=a; a=b; b=tmp;}
    return a;
}

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

// returns d = gcd(a,b); finds x,y such that d = ax + by
int extended_euclid(int a, int b, int &x, int &y) {
    int xx = y = 0;
    int yy = x = 1;
    while (b) {
        int q = a/b;
        int t = b; b = a%b; a = t;
        t = xx; xx = x-q*xx; x = t;
        t = yy; yy = y-q*yy; y = t;
    }
    return a;
}

// finds all solutions to ax = b (mod n)
VI modular_linear_equation_solver(int a, int b, int n) {
    int x, y;
    VI solutions;
    int d = extended_euclid(a, n, x, y);
    if (!(b%d)) {
        x = mod (x*(b/d), n);
        for (int i = 0; i < d; i++)
            solutions.push_back(mod(x + i*(n/d), n));
    }
    return solutions;
}

// computes b such that ab = 1 (mod n), returns -1 on failure
int mod_inverse(int a, int n) {
    int x, y;
    int d = extended_euclid(a, n, x, y);
    if (d > 1) return -1;
    return mod(x,n);
}

// Chinese remainder theorem (special case): find z such that
// z % x = a, z % y = b.  Here, z is unique modulo M = lcm(x,y).
// Return (z,M).  On failure, M = -1.
PII chinese_remainder_theorem(int x, int a, int y, int b) {
    int s, t;
    int d = extended_euclid(x, y, s, t);
    if (a%d != b%d) return make_pair(0, -1);
    return make_pair(mod(s*b*x+t*a*y,x*y)/d, x*y/d);
}

// Chinese remainder theorem: find z such that
// z % x[i] = a[i] for all i.  Note that the solution is
// unique modulo M = lcm_i (x[i]).  Return (z,M).  On
// failure, M = -1.  Note that we do not require the a[i]'s
// to be relatively prime.
PII chinese_remainder_theorem(const VI &x, const VI &a) {
    PII ret = make_pair(a[0], x[0]);
    for (int i = 1; i < (int) x.size(); i++) {
        ret = chinese_remainder_theorem(ret.second, ret.first, x[i], a[i]);
        if (ret.second == -1) break;
    }
    return ret;
}

// computes x and y such that ax + by = c; on failure, x = y =-1
void linear_diophantine(int a, int b, int c, int &x, int &y) {
    int d = gcd(a,b);
    if (c%d) {
        x = y = -1;
    } else {
        x = c/d * mod_inverse(a/d, b/d);
        y = (c-a*x)/b;
    }
}

int main() {
    cout << gcd(14, 30) << endl; // 2
    int x, y, d = extended_euclid(14, 30, x, y);
    cout << d << " " << x << " " << y << endl; // 2 -2 1
    VI sols = modular_linear_equation_solver(14, 30, 100); // 95 45
    for (int i = 0; i < (int) sols.size(); i++) cout << sols[i] << " ";
    cout << endl;
    cout << mod_inverse(8, 9) << endl; // 8
    int xs[] = {3, 5, 7, 4, 6};
    int as[] = {2, 3, 2, 3, 5};
    PII ret = chinese_remainder_theorem(VI (xs, xs+3), VI(as, as+3));
    cout << ret.first << " " << ret.second << endl; // 23 56
    ret = chinese_remainder_theorem (VI(xs+3, xs+5), VI(as+3, as+5));
    cout << ret.first << " " << ret.second << endl; // 11 12
    linear_diophantine(7, 2, 5, x, y);
    cout << x << " " << y << endl; // expected: 5 -15
}
```

10 PrimitiveRoot

```
int generator(int p) {
    vector<int> fact;
    int phi = p-1, n = phi;
    for (int i=2; i*i<=n; ++i) if (n % i == 0) {
        fact.push_back(i);
        while (n % i == 0) n /= i;
    }
    if (n > 1) fact.push_back(n);
    for (int res=2; res<=p; ++res) {
        bool ok = true;
        for (size_t i=0; i<fact.size() && ok; ++i)
            ok &= powmod (res, phi / fact[i], p) != 1;
        if (ok)  return res;
    }
    return -1;
}
```

11 RabinMiller

```
bool suspect(ll a, ll s, ll d, ll n) {
   ll x = powMod(a, d, n);
   if (x == 1) return true;
   for (int r = 0; r < s; ++r) {
      if (x == n - 1) return true;
      x = x * x % n;
   }
   return false;
}
// {2,7,61,-1}                 is for n < 4759123141 (= 2^32)
// {2,3,5,7,11,13,17,19,23,-1} is for n < 10^16 (at least)
bool isPrime(ll n) {
   if (n <= 1 || (n > 2 && n % 2 == 0)) return false;
   ll test[] = {2,3,5,7,11,13,17,19,23,-1};
   ll d = n - 1, s = 0;
   while (d % 2 == 0) ++s, d /= 2;
   for (int i = 0; test[i] < n && test[i] != -1; ++i)
      if (!suspect(test[i], s, d, n)) return false;
   return true;
}
```

12 SqrtMod

```
// Jacobi Symbol (m/n), m,n0  and n is odd
// (m/n)==1 x^2 == m (mod n) solvable, -1 unsolvable
#define NEGPOW(e) ((e) % 2 ? -1 : 1)
int jacobi(int a, int m) {
   if (a == 0) return m == 1 ? 1 : 0;
   if (a % 2)  return NEGPOW((a-1)*(m-1)/4)*jacobi(m%a, a);
   else return NEGPOW((m*m-1)/8)*jacobi(a/2, m);
}
// No solution when: n(p-1)/2 = -1 mod p
int sqrtMod(int n, int p) { //find x: x2 = n (mod p) p is prime
   int S, Q, W, i, m = invMod(n, p);
   for (Q = p - 1, S = 0; Q % 2 == 0; Q /= 2, ++S);
   do { W = rand() % p; } while (W == 0 || jacobi(W, p) != -1);
   for (int R = powMod(n, (Q+1)/2, p), V = powMod(W, Q, p); ;) {
      int z = R * R * m % p;
      for (i = 0; i < S && z % p != 1; z *= z, ++i);
      if (i == 0) return R;
      R = (R * powMod(V, 1 << (S-i-1), p)) % p;
   }
}
```

13 kdTree

```
// -----------------------------------------------------------------
// A straightforward, but probably sub-optimal KD-tree implmentation
// that's probably good enough for most things (current it's a
// 2D-tree)
//
//  - constructs from n points in O(n lg^2 n) time
//  - handles nearest-neighbor query in O(lg n) if points are well
//    distributed
//  - worst case for nearest-neighbor may be linear in pathological
//    case
//
// Sonny Chan, Stanford University, April 2009
// -----------------------------------------------------------------

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

// number type for coordinates, and its maximum value
typedef long long ntype;
const ntype sentry = numeric_limits<ntype>::max();

// point structure for 2D-tree, can be extended to 3D
struct point {
    ntype x, y;
    point(ntype xx = 0, ntype yy = 0) : x(xx), y(yy) {}
};

bool operator==(const point &a, const point &b) {
    return a.x == b.x && a.y == b.y;
}

// sorts points on x-coordinate
bool on_x(const point &a, const point &b) {
    return a.x < b.x;
}

// sorts points on y-coordinate
bool on_y(const point &a, const point &b) {
    return a.y < b.y;
}

// squared distance between points
ntype pdist2(const point &a, const point &b) {
    ntype dx = a.x-b.x, dy = a.y-b.y;
    return dx*dx + dy*dy;
}

// bounding box for a set of points
struct bbox {
    ntype x0, x1, y0, y1;

    bbox() : x0(sentry), x1(-sentry), y0(sentry), y1(-sentry) {}

    // computes bounding box from a bunch of points
    void compute(const vector<point> &v) {
        for (int i = 0; i < (int) v.size(); ++i) {
            x0 = min(x0, v[i].x);   x1 = max(x1, v[i].x);
            y0 = min(y0, v[i].y);   y1 = max(y1, v[i].y);
        }
    }

    // squared distance between a point and this bbox, 0 if inside
    ntype distance(const point &p) {
        if (p.x < x0) {
            if (p.y < y0)       return pdist2(point(x0, y0), p);
            else if (p.y > y1)  return pdist2(point(x0, y1), p);
            else                return pdist2(point(x0, p.y), p);
        }
        else if (p.x > x1) {
            if (p.y < y0)       return pdist2(point(x1, y0), p);
            else if (p.y > y1)  return pdist2(point(x1, y1), p);
            else                return pdist2(point(x1, p.y), p);
        }
        else {
            if (p.y < y0)       return pdist2(point(p.x, y0), p);
            else if (p.y > y1)  return pdist2(point(p.x, y1), p);
            else                return 0;
        }
    }
};

// stores a single node of the kd-tree, either internal or leaf
struct kdnode {
    bool leaf;      // true if this is a leaf node (has one point)
    point pt;       // the single point of this is a leaf
    bbox bound;     // bounding box for set of points in children

    kdnode *first, *second; // two children of this kd-node

    kdnode() : leaf(false), first(0), second(0) {}
    ~kdnode() { if (first) delete first; if (second) delete second; }

    // intersect a point with this node (returns squared distance)
    ntype intersect(const point &p) {
        return bound.distance(p);
    }

    // recursively builds a kd-tree from a given cloud of points
    void construct(vector<point> &vp) {
        // compute bounding box for points at this node
        bound.compute(vp);

        // if we're down to one point, then we're a leaf node
        if (vp.size() == 1) {
            leaf = true;
            pt = vp[0];
        } else {
            // split on x if the bbox is wider than high (not best heuristic
                ...)
            if (bound.x1-bound.x0 >= bound.y1-bound.y0)
                sort(vp.begin(), vp.end(), on_x);
            // otherwise split on y-coordinate
            else
                sort(vp.begin(), vp.end(), on_y);

            // divide by taking half the array for each child
            // (not best performance if many duplicates in the middle)
            int half = vp.size()/2;
            vector<point> vl(vp.begin(), vp.begin()+half);
            vector<point> vr(vp.begin()+half, vp.end());
            first = new kdnode();   first->construct(vl);
            second = new kdnode();  second->construct(vr);
        }
    }
};

// simple kd-tree class to hold the tree and handle queries
struct kdtree {
    kdnode *root;

    // constructs a kd-tree from a points (copied here, as it sorts them)
    kdtree(const vector<point> &vp) {
        vector<point> v(vp.begin(), vp.end());
        root = new kdnode();
        root->construct(v);
    }
    ~kdtree() { delete root; }

    // recursive search method returns squared distance to nearest point
    ntype search(kdnode *node, const point &p) {
        if (node->leaf) {
            // commented special case tells a point not to find itself
//            if (p == node->pt) return sentry;
//            else
                return pdist2(p, node->pt);
        }

        ntype bfirst = node->first->intersect(p);
        ntype bsecond = node->second->intersect(p);

        // choose the side with the closest bounding box to search first
        // (note that the other side is also searched if needed)
        if (bfirst < bsecond) {
            ntype best = search(node->first, p);
            if (bsecond < best)
                best = min(best, search(node->second, p));
            return best;
        } else {
            ntype best = search(node->second, p);
            if (bfirst < best)
                best = min(best, search(node->first, p));
            return best;
        }
    }
    // squared distance to the nearest
    ntype nearest(const point &p) {
        return search(root, p);
    }
};

// --------------------------------------------------------------------------
// some basic test code here

int main() {
    // generate some random points for a kd-tree
    vector<point> vp;
    for (int i = 0; i < 100000; ++i) {
        vp.push_back(point(rand()%100000, rand()%100000));
    }
    kdtree tree(vp);

    // query some points
    for (int i = 0; i < 10; ++i) {
        point q(rand()%100000, rand()%100000);
        cout << "Closest squared distance to (" << q.x << ", " << q.y << ")"
             << " is " << tree.nearest(q) << endl;
    }

    return 0;
}

// --------------------------------------------------------------------------
```

14 FenwickTree

```
struct FenwickTree { // 1-based index
    vector<int> tmul, tadd;
    int n;

    FenwickTree(int _n): tmul (_n + 1), tadd (_n + 1), n (_n) {
    }

    void update_point(int x, int mul, int add) {
        while (1<=x && x<=n) {
            tmul[x]+=mul;
            tadd[x]+=add;
            x+=x&(-x);
        }
    }

    void update_range(int l, int r, int val) { // l to r (inclusive)
        update_point(l,val,-val*(l-1));
        update_point(r,-val,val*r);
    }

    int get(int x) {
        int mul = 0, add = 0, fst = x;
        while (1<=x && x<=n) {
            mul+=tmul[x];
            add+=tadd[x];
            x=x&(x-1);
        }
        return (mul*fst+add);
    }

    int value(int x) {
        return (get(x)-get(x-1));
    }
};
```

15 SplayTree

```
struct Node {
    Node * child[2], * parent;
    bool reverse;
    int value, size;
    long long sum;
};

Node * nil, * root;

void initTree() {
    nil = new Node();
    nil->child[0] = nil->child[1] = nil->parent = nil;
    nil->value = nil->size = nil->sum = 0;
    nil->reverse = false;
    root = nil;
}

void pushDown(Node * x) {
    if(x == nil) return;
    if(x->reverse) {
        swap(x->child[0], x->child[1]);
        x->child[0]->reverse = !x->child[0]->reverse;
        x->child[1]->reverse = !x->child[1]->reverse;
        x->reverse = false;
    }
}

void update(Node * x) {
    pushDown(x->child[0]); pushDown(x->child[1]);
    x->size = x->child[0]->size + x->child[1]->size + 1;
    x->sum = x->child[0]->sum + x->child[1]->sum + x->value;
}

void setLink(Node * x, Node * y, int d) {
    x->child[d] = y;
    y->parent = x;
}

int getDir(Node * x, Node * y) {
    return x->child[0] == y ? 0 : 1;
}

void rotate(Node * x, int d) {
    Node * y = x->child[d], * z = x->parent;
    setLink(x, y->child[d ^ 1], d);
    setLink(y, x, d ^ 1);
    setLink(z, y, getDir(z, x));
    update(x); update(y);
}

void splay(Node * x) {
    while(x->parent != nil) {
        Node * y = x->parent, * z = y->parent;
        int dy = getDir(y, x), dz = getDir(z, y);
        if(z == nil) rotate(y, dy);
        else if(dy == dz) rotate(z, dz), rotate(y, dy);
        else rotate(y, dy), rotate(z, dz);
    }
}

Node * nodeAt(Node * x, int pos) {
    while(pushDown(x), x->child[0]->size != pos)
        if(pos < x->child[0]->size) x = x->child[0];
        else pos -= x->child[0]->size + 1, x = x->child[1];
    return splay(x), x;
}

void split(Node * x, int left, Node * &t1, Node * &t2) {
    if(left == 0) t1 = nil, t2 = x;
    else {
        t1 = nodeAt(x, left - 1);
        t2 = t1->child[1];
        t1->child[1] = t2->parent = nil;
        update(t1);
    }
}

Node * join(Node * x, Node * y) {
    if(x == nil) return y;
    x = nodeAt(x, x->size - 1);
    setLink(x, y, 1);
    update(x);
    return x;
}
```

16 Geometry

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

#define REP(i,n) for(int i=0;i<(int)n;++i)
#define FOR(i,c) for(__typeof((c).begin())i=(c).begin();i!=(c).end();++i)
#define ALL(c) (c).begin(), (c).end()
#define chmax(a,b) (a<b?(a=b,1):0)
#define chmin(a,b) (a>b?(a=b,1):0)
#define valid(y,x,h,w) (0<=y&&y<h&&0<=x&&x<w)
typedef long long ll;
typedef pair<int,int> pii;
const int INF = 1<<29;
const double PI = acos(-1);
const double EPS = 1e-8;

typedef complex<double> P;
namespace std {
    bool operator < (const P& a, const P& b) {
        // if (abs(a-b)<EPS) return 0;
        return real(a) != real(b) ? real(a) < real(b) : imag(a) < imag(b);
    }
}
double cross(const P& a, const P& b) {
    return imag(conj(a)*b);
}
double dot(const P& a, const P& b) {
    return real(conj(a)*b);
}
struct L : public vector<P> {
    L(const P &a, const P &b) {
        push_back(a); push_back(b);
    }
    L() {}
};
typedef vector<P> G;
#define curr(P, i) P[i]
#define next(P, i) P[(i+1)%P.size()]
struct C {
    P p; double r;
    C(const P &p, double r) : p(p), r(r) { }
};

int ccw(P a, P b, P c) {
    b -= a; c -= a;
    if (cross(b, c) > 0)   return +1;       // counter clockwise
    if (cross(b, c) < 0)   return -1;       // clockwise
    if (dot(b, c) < 0)     return +2;       // c--a--b on line
    if (norm(b) < norm(c)) return -2;       // a--b--c on line
    return 0;
}

bool intersectLL(const L &l, const L &m) {
    return abs(cross(l[1]-l[0], m[1]-m[0])) > EPS || // non-parallel
        abs(cross(l[1]-l[0], m[0]-l[0])) < EPS;   // same line
}
bool intersectLS(const L &l, const L &s) {
    return cross(l[1]-l[0], s[0]-l[0])*       // s[0] is left of l
        cross(l[1]-l[0], s[1]-l[0]) < EPS; // s[1] is right of l
}
bool intersectLP(const L &l, const P &p) {
    return abs(cross(l[1]-p, l[0]-p)) < EPS;
}
```

```
bool intersectSS(const L &s, const L &t) {
    return ccw(s[0],s[1],t[0])*ccw(s[0],s[1],t[1]) <= 0 &&
        ccw(t[0],t[1],s[0])*ccw(t[0],t[1],s[1]) <= 0;
}
bool intersectSS2(const L &s, const L &t) { // 0 if touching
    REP(i, 2) {
        if (ccw(s[0], s[1], t[i]) == 0) {
            int c = ccw(s[0],s[1],t[!i]);
            if (s[0] == t[i]) {
                if (c!=-2&&c) return 0;
            } else if (s[1] == t[i]) {
                if (c!=2&&c) return 0;
            } else if (abs(c)==1) return 0;
        }
    }
    return ccw(s[0],s[1],t[0])*ccw(s[0],s[1],t[1]) <= 0 &&
        ccw(t[0],t[1],s[0])*ccw(t[0],t[1],s[1]) <= 0;
}
bool intersectSP(const L &s, const P &p) {
    return abs(s[0]-p)+abs(s[1]-p)-abs(s[1]-s[0]) < EPS; // triangle inequality
}

P projection(const L &l, const P &p) {
    double t = dot(p-l[0], l[0]-l[1]) / norm(l[0]-l[1]);
    return l[0] + t*(l[0]-l[1]);
}
P reflection(const L &l, const P &p) {
    return p + P(2,0) * (projection(l, p) - p);
}
double distanceLP(const L &l, const P &p) {
    return abs(p - projection(l, p));
}
double distanceLL(const L &l, const L &m) {
    return intersectLL(l, m) ? 0 : distanceLP(l, m[0]);
}
double istanceLS(const L &l, const L &s) {
    if (intersectLS(l, s)) return 0;
    return min(distanceLP(l, s[0]), distanceLP(l, s[1]));
}
double distanceSP(const L &s, const P &p) {
    const P r = projection(s, p);
    if (intersectSP(s, r)) return abs(r - p);
    return min(abs(s[0] - p), abs(s[1] - p));
}
double distanceSS(const L &s, const L &t) {
    if (intersectSS(s, t)) return 0;
    return min(min(distanceSP(s, t[0]), distanceSP(s, t[1])),
            min(distanceSP(t, s[0]), distanceSP(t, s[1])));
}
P crosspoint(const L &l, const L &m) {
    double A = cross(l[1] - l[0], m[1] - m[0]);
    double B = cross(l[1] - l[0], l[1] - m[0]);
    if (abs(A) < EPS && abs(B) < EPS) return m[0]; // same line
    if (abs(A) < EPS) assert(false); // !!!PRECONDITION NOT SATISFIED!!!
    return m[0] + B / A * (m[1] - m[0]);
}
double area(const G& g) {
    double A = 0;
    for (int i = 0; i < g.size(); ++i) {
        A += cross(g[i], next(g, i));
    }
    return abs(A/2);
}

G convex_cut(const G& g, const L& l) {
    G Q;
    REP(i, g.size()) {
        P A = curr(g, i), B = next(g, i);
        if (ccw(l[0], l[1], A) != -1) Q.push_back(A);
        if (ccw(l[0], l[1], A)*ccw(l[0], l[1], B) < 0)
            Q.push_back(crosspoint(L(A, B), l));
    }
    return Q;
}
///////////////////////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////

// Centroid of a polygon
P centroid(const vector<P> &v) {
    double S = 0;
    P res;
    REP(i,v.size()) {
        int j = i+1;
        if (j == v.size()) j = 0;
        double tmp = cross(v[i], v[j]);
        S += tmp;
        res += (v[i] + v[j]) * tmp;
    }
    S /= 2;
    res /= 6*S;
    return res;
}

double manDistanceSP(const L &l, const P &p) {
    double res = INF;
    L xl = L(p, p + P(1,0));
    if (intersectLS(xl, l)){
        P cp = crosspoint(xl, l);
        double d = abs(p-cp);
        res = min(res, d);
    }
    L yl = L(p, p + P(0,1));
    if (intersectLS(yl, l)) {
        P cp = crosspoint(yl, l);
        double d = abs(p-cp);
        res = min(res, d);
    }
    res = min(res, abs(l[0].real()-p.real()) + abs(l[0].imag()-p.imag()));
    res = min(res, abs(l[1].real()-p.real()) + abs(l[1].imag()-p.imag()));
    return res;
}

// Check if a (counter-clockwise) convex polygoncontains a point
bool convex_contain(const G &g, const P &p) {
    REP(i,g.size())
        if (ccw(g[i], next(g, i), p) == -1) return 0;
    return 1;
}
// Check if two polygons have common point
bool intersectGG(const G &g1, const G &g2) {
    if (convex_contain(g1, g2[0])) return 1;
    if (convex_contain(g2, g1[0])) return 1;
    REP(i,g1.size()) REP(j,g2.size()) {
        if (intersectSS(L(g1[i], next(g1, i)), L(g2[j],next(g2, j)))) return 1;
    }
    return 0;
}
// Distance between a point and a polygon
double distanceGP(const G &g, const P &p) {
    if (convex_contain(g, p)) return 0;
    double res = INF;
    REP(i, g.size()) {
        res = min(res, distanceSP(L(g[i], next(g, i)), p));
    }
    return res;
}
// Duong phan giac
L bisector(const P &a, const P &b) {
    P A = (a+b)*P(0.5,0);
    return L(A, A+(b-a)*P(0, PI/2));
}
// Voronoi
G voronoi_cell(G g, const vector<P> &v, int s) {
    REP(i, v.size())
        if (i!=s)
            g = convex_cut(g, bisector(v[s], v[i]));
    return g;
}
// Angle-related
double angle(const P &a, const P &b) { // Goc dinh huong a -> b [0,2pi)
    double ret = arg(b)-arg(a);
    return (ret>=0) ? ret : ret + 2*PI;
}
double angle2(const P &a, const P &b) { // Goc giua a va b
    return min(angle(a,b), angle(b,a));
}
double rtod(double rad) { // Radian to degree
    return rad*180/PI;
}
double dtor(double deg) { // Degree to radian
    return deg*PI/180;
}
// Rotation
P rotate(P p, double ang) {
    return p * P(cos(ang), sin(ang));
}
// Rotate a line around O
L rotate(L l, double ang) {
    return L(rotate(l[0], ang),rotate(l[1], ang));
}
```

17 ConvexHull

```
struct Point {
    long long x, y;
    bool operator < (const Point &v) const {
        return x == v.x ? y < v.y : x < v.x;
    }
    long long cross(const Point &p, const Point &q) const {
        return (p.x - x) * (q.y - y) - (p.y - y) * (q.x - x);
    }
};

vector<Point> convexHull(vector<Point> p) {
    sort(p.begin(), p.end());
    int k = 0, n = p.size();
    vector<Point> poly (2 * n);
    for(int i = 0; i < n; ++i) {
        while(k >= 2 && poly[k-2].cross(poly[k-1], p[i]) < 0) --k;
        poly[k++] = p[i];
    }
    for(int i = n-2, t = k+1; i >= 0; --i) {
        while(k >= t && poly[k-2].cross(poly[k-1], p[i]) < 0) --k;
        poly[k++] = p[i];
    }
    poly.resize(min(n, max(0, k - 1)));
    return poly;
}
```

18 BiConComps

```
const int N = 1024;

int count, parent[N], n; //n vertices 0..n-1
bool visited[N];
vector<int> G[N];
stack<pair<int, int> > s;

void OutputComp(int u, int v) {
    pair<int, int> edge;
    do {
        edge = s.top(); s.pop();
        printf("%d %d\n", edge.first, edge.second);
    } while (edge != make_pair(u, v));
    printf("\n");
}

void dfs(int u) {
    visited[u] = true;
    count++;
    low[u] = num[u] = count;
    for (int v : G[u]) {
        if (!visited[v]) {
            s.push({u, v});
            parent[v] = u;
            dfs(v);
            if (low[v] > num[u]) OutputComp(u, v);
            low[u] = min(low[u], low[v]);
        } else if (parent[u] != v && num[v] < num[u]) {
            s.push({u, v});
            low[u] = min(low[u], num[v]);
        }
    }
}

void BiconnectedComponents {
    count = 0;
    memset(parent, -1, sizeof parent);
    for (int i = 0; i < n; i++)
        if (!visited[i]) dfs(i);
}
```

19 DirectedMST

```
const int maxe = 100111, maxv = 100;

// Index from 0, running time O(E*V)
namespace chuliu {
    struct Cost;
    vector<Cost> costlist;

    struct Cost {
        int id, val, used, a, b, pos;
        Cost() { val = -1; used = 0; }
        Cost(int _id, int _val, bool temp) {
            a = b = -1; id = _id; val = _val; 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, int c) {
            u = _u; v = _v; cost = Cost(id, c, 0);
        }
    } edge[maxe];

    int n, m, root, pre[maxv], node[maxv], vis[maxv], best[maxv];

    void init(int _n) {
        n = _n; m = 0;
        costlist.clear();
    }

    void add(int id, int u, int v, int c) {
        edge[m++] = Edge(id, u, v, c);
    }

    int mst(int root) {
        int ret = 0;
        while (true) {
            REP(i, n) best[i] = -1;
            REP(e, m) {
                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;
                }
            }
            REP(i, n) if (i != root && best[i] == -1) return -1;
            int cntnode = 0;
            memset(node, -1, sizeof node); memset(vis, -1, sizeof vis);
            REP(i, n) if (i != root) {
                ret += 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;
            REP(i, n) if (node[i] == -1) node[i] = cntnode++;
            REP(e, m) {
                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 ret;
    }

    vector<int> trace() {
        vector<int> ret;
        FORD(i, costlist.size()-1,0) costlist[i].push();
        REP(i, costlist.size()) {
            Cost cost = costlist[i];
            if (cost.id != -1 && cost.used > 0) ret.push_back(cost.id);
        }
        return ret;
    }
}
```

20 StableMarriage

```
/*
 *      Takes a set of m men and n women, where each person has
 *      an integer preference for each of the persons of the opposite
 *      sex. Produces a matching of each man to some woman. The matching
 *      will have the following properties:
 *          - Each man is assigned a different woman (n must be at least m).
 *          - No two couples M1W1 and M2W2 will be unstable.
 *      Two couples are unstable if
 *          - M1 prefers W2 over W1 and
 *          - W1 prefers M2 over M1.
 * INPUTS:
 *   - m:        number of men.
 *   - n:        number of women (must be at least as large as m).
 *   - L[i][]:   the list of women in order of decreasing preference of man i.
 *   - R[j][i]:  the attractiveness of i to j.
 * OUTPUTS:
 *   - L2R[]:    the mate of man i (always between 0 and n-1)
 *   - R2L[]:    the mate of woman j (or -1 if single)
 * ALGORITHM:
 *   The algorithm is greedy and runs in time O(m^2).
 */

#define MAXM 1024
#define MAXW 1024
int m, n;
int L[MAXM][MAXW], R[MAXW][MAXM];
int L2R[MAXM], R2L[MAXW];
int p[MAXM];
void stableMarriage(){
    static int p[128];
    memset( R2L, -1, sizeof( R2L ) );
    memset( p, 0, sizeof( p ) );
    // Each man proposes...
    for( int i = 0; i < m; i++ ) {
        int man = i;
        while( man >= 0 ) {
            // to the next woman on his list in order of decreasing preference,
            // until one of them accepts;
            int wom;
            while( 1 ){
                wom = L[man][p[man]++];
                if( R2L[wom] < 0 || R[wom][man] > R[wom][R2L[wom]] ) break;
            }
            // Remember the old husband of wom.
            int hubby = R2L[wom];
            // Marry man and wom.
            R2L[L2R[man] = wom] = man;
            // If a guy was dumped in the process, remarry him now.
            man = hubby;
        }
    }
}
```

21 GeneralMatching

```
// General matching on graph

const int maxv = 1000;
const int maxe = 50000;

// Index from 1
// Directed
struct EdmondsLawler {
    int n, E, start, finish, newRoot, qsize, adj[maxe], next[maxe], last[maxv],
         mat[maxv], que[maxv], dad[maxv], root[maxv];
    bool inque[maxv], inpath[maxv], inblossom[maxv];

    void init(int _n) {
        n = _n; E = 0;
        for(int x=1; x<=n; ++x) { last[x] = -1; mat[x] = 0; }
    }
    void add(int u, int v) {
        adj[E] = v; next[E] = last[u]; last[u] = E++;
    }
    int lca(int u, int v) {
        for(int x=1; x<=n; ++x) inpath[x] = false;
        while (true) {
            u = root[u];
            inpath[u] = true;
            if (u == start) break;
            u = dad[mat[u]];
        }
        while (true) {
            v = root[v];
            if (inpath[v]) break;
            v = dad[mat[v]];
        }
        return v;
    }
    void trace(int u) {
        while (root[u] != newRoot) {
            int v = mat[u];

            inblossom[root[u]] = true;
            inblossom[root[v]] = true;

            u = dad[v];
            if (root[u] != newRoot) dad[u] = v;
        }
    }
    void blossom(int u, int v) {
        for(int x=1; x<=n; ++x) inblossom[x] = false;

        newRoot = lca(u, v);
        trace(u); trace(v);

        if (root[u] != newRoot) dad[u] = v;
        if (root[v] != newRoot) dad[v] = u;

        for(int x=1; x<=n; ++x) if (inblossom[root[x]]) {
            root[x] = newRoot;
            if (!inque[x]) {
                inque[x] = true;
                que[qsize++] = x;
            }
        }
    }
    bool bfs() {
        for(int x=1; x<=n; ++x){
            inque[x] = false;
            dad[x] = 0;
            root[x] = x;
        }
        qsize = 0;
        que[qsize++] = start;
        inque[start] = true;
        finish = 0;

        for(int i=0; i<qsize; ++i) {
            int u = que[i];
            for (int e = last[u]; e != -1; e = next[e]) {
                int v = adj[e];
                if (root[v] != root[u] && v != mat[u]) {
                    if (v == start || (mat[v] > 0 && dad[mat[v]] > 0)) blossom(
                        u, v);
                    else if (dad[v] == 0) {
                        dad[v] = u;
                        if (mat[v] > 0) que[qsize++] = mat[v];
                        else {
                            finish = v;
                            return true;
                        }
                    }
                }
            }
        }
        return false;
    }
    void enlarge() {
        int u = finish;
        while (u > 0) {
            int v = dad[u], x = mat[v];
            mat[v] = u;
            mat[u] = v;
            u = x;
        }
    }
    int maxmat() {
        for(int x=1; x<=n; ++x) if (mat[x] == 0) {
            start = x;
            if (bfs()) enlarge();
        }

        int ret = 0;
        for(int x=1; x<=n; ++x) if (mat[x] > x) ++ret;
        return ret;
    }
} edmonds;
```

22 BipartiteMatching

```
const int N = 50000, INF = 1e9;
int mx[N], my[N], d[N], nx, ny;
vector<int> g[N];

bool dfs(int u, int mind) {
    for (int v : g[u]) if(my[v] == -1 ? d[u] == mind : d[my[v]] == d[u]+1 &&
        dfs(my[v], mind)) return mx[u] = v, my[v] = u, true;
    d[u] = INF;
    return false;
}

int maxMatch() {
    int matching = 0;
    fill(mx, mx + nx, -1); fill(my, my + ny, -1);
    while(true) {
        fill(d, d + nx, INF); queue<int> q;
        for(int u = 0; u < nx; ++u) if(mx[u] == -1) d[u] = 0, q.push(u);
        int mind = INF;
        while(!q.empty()) {
            int u = q.front(); q.pop();
            for (int v : g[u]) if(my[v] == -1) mind = min(mind, d[u]);
            else if(d[my[v]] == INF) d[my[v]] = d[u]+1, q.push(my[v]);
        }
        if(mind == INF) break;
        for(int u = 0; u < nx; ++u) if(d[u] > mind) d[u] = INF;
        for(int u = 0; u < nx; ++u)
```

```
            if(mx[u] == -1 && dfs(u, mind)) ++matching;
    }
    return matching;
}
```

23 KuhnMunkresFast

```
const int V = 1000, INF = 1e9;
int g[V][V], mx[V], my[V], fx[V], fy[V], d[V], ar[V], tr[V], p;

int slack(int u, int v) {
    return g[u][v] - fx[u] - fy[v];
}

int augment(int s) {
    queue<int> q; q.push(s);
    fill_n(tr, p, -1);
    for(int i = 0; i < p; ++i)
        d[i] = slack(s, i), ar[i] = s;
    while(true) {
        while(!q.empty()) {
            int u = q.front(); q.pop();
            for(int v = 0; v < p; ++v) if(tr[v] == -1) {
                int w = slack(u, v);
                if(w == 0) {
                    tr[v] = u;
                    if(my[v] == -1) return v;
                    q.push(my[v]);
                }
                if(d[v] > w) d[v] = w, ar[v] = u;
            }
        }
        int delta = INF;
        for(int v = 0; v < p; ++v) if(tr[v] == -1) delta = min(delta, d[v]);
        fx[s] += delta;
        for(int v = 0; v < p; ++v)
            if(tr[v] == -1) d[v] -= delta;
            else fx[my[v]] += delta, fy[v] -= delta;
        for(int v = 0; v < p; ++v) if(tr[v] == -1 && d[v] == 0) {
            tr[v] = ar[v];
            if(my[v] == -1) return v;
            q.push(my[v]);
        }
    }
}

void maxMatchMinCost() {
    fill_n(mx, p, -1); fill_n(my, p, -1);
    for(int i = 0; i < p; ++i) fx[i] = *min_element(g[i], g[i]+p);
    for(int s = 0; s < p; ++s) {
        int f = augment(s);
        while(f != -1) {
            int x = tr[f], nx = mx[x];
            mx[x] = f; my[f] = x; f = nx;
        }
    }
}
```

24 KuhnMunkres

```
const int INF = (int) 1e9;
const int M = 1000;
int c[M][M], fx[M], fy[M], my[M], m; // m = number of vertices
bool vx[M], vy[M];

int slack(int u, int v) {
    return c[u][v] - fx[u] - fy[v];
}

bool dfs(int u) {
    vx[u] = true;
    for(int v = 0; v < m; ++v)
        if(slack(u, v) == 0 && !vy[v]) {
            vy[v] = true;
            if(my[v] == -1 || dfs(my[v]))
                return my[v] = u, true;
        }
    return false;
}

void maxMatchMinCost() {
    memset(my, -1, sizeof my);
    for(int i = 0; i < m; ++i)
        fx[i] = *min_element(c[i], c[i] + m);
    for(int i = 0; i < m; ++i) {
        memset(vx, 0, sizeof vx);
        memset(my, 0, sizeof vy);
        while(!dfs(i)) {
            int delta = INF;
            for(int u = 0; u < m; ++u) if(vx[u])
                for(int v = 0; v < m; ++v) if(!vy[v])
                    delta = min(delta, slack(u, v));
            for(int u = 0; u < m; ++u) if(vx[u])
                fx[u] += delta, vx[u] = false;
            for(int u = 0; u < m; ++u) if(vy[u])
                fy[u] -= delta, vy[u] = false;
        }
    }
}
```

25 MaxFlow

```
struct DinicFlow {
    static const int INF=(int)1e9+7;
    vector<int> dist,head,q,work;
    vector<int> point,capa,flow,next;
    int n,m;
    DinicFlow() {
        n=0;m=0;
    }
    DinicFlow(int n) {
        this->n=n;m=0;
        dist.assign(n+7,0);
        head.assign(n+7,-1);
        q.assign(n+7,0);
        work.assign(n+7,0);
    }
    void addEdge(int u,int v,int c1,int c2) {
        point.push_back(v);capa.push_back(c1);flow.push_back(0);next.push_back(
            head[u]);head[u]=m++;
        point.push_back(u);capa.push_back(c2);flow.push_back(0);next.push_back(
            head[v]);head[v]=m++;
    }
    bool bfs(int s,int t) {
        fill(dist.begin(), dist.end(), -1);
        int sz=0;
        q[sz++]=s;dist[s]=0;
        for (int x=0;x<sz;x=x+1) {
            int u=q[x];
            for (int i=head[u];i>=0;i=next[i])
                if (dist[point[i]]<0 && flow[i]<capa[i]) {
                    dist[point[i]]=dist[u]+1;
                    q[sz++]=point[i];
                }
        }
        return (dist[t]>=0);
    }
    int dfs(int s,int t,int f) {
        if (s==t) return (f);
        for (int &i=work[s];i>=0;i=next[i])
            if (dist[point[i]]==dist[s]+1 && flow[i]<capa[i]) {
                int d=dfs(point[i],t,min(f,capa[i]-flow[i]));
                if (d>0) {
                    flow[i]+=d;
                    flow[i^1]-=d;
                    return (d);
                }
            }
        return (0);
    }
    int maxFlow(int s,int t) {
        int totflow=0;
        while (bfs(s,t)) {
            copy(head.begin(), head.end(), work.begin());
            while (true) {
                int d=dfs(s,t,INF);
                if (d<=0) break;
                totflow+=d;
            }
        }
        return (totflow);
    }
    void resetFlow() {
        fill(flow.begin(), flow.end(), 0);
    }
};
```

26 MaxFlowFast

```
// Fastest flow
// Index from 0, directed
// To use:
// MaxFlow flow(n)
// For each edge: flow.addEdge(u, v, c)
// result = flow.getFlow(s, t)

struct Edge {
    int u, v, c, f;
    int next;
};

struct MaxFlow {
    int n, s, t;
    vector< Edge > edges;
    vector<int> head, current, h, avail;
    vector<long long> excess;

    MaxFlow(int n) : n(n), head(n, -1), current(n, -1), h(n), avail(n), excess(
        n) {
        edges.clear();
    }

    int addEdge(int u, int v, int c, bool bi = false) {
        Edge xuoi = {u, v, c, 0, head[u]};
        head[u] = edges.size(); edges.push_back(xuoi);
        Edge nguoc = {v, u, bi ? c : 0, 0, head[v]};
        head[v] = edges.size(); edges.push_back(nguoc);
        return v == u ? head[u] - 1 : head[u];
    }

    long long getFlow(int _s, int _t) {
        s = _s; t = _t;
        init();

        int now = 0;
        queue<int> qu[2];
        REP(x,n)
            if (x != s && x != t && excess[x] > 0)
                qu[now].push(x);

        globalLabeling();

        int cnt = 0;
        while (!qu[now].empty()) {
            while (!qu[1-now].empty()) qu[1-now].pop();
            while (!qu[now].empty()) {
                int x = qu[now].front(); qu[now].pop();
                while (current[x] >= 0) {
                    int p = current[x];
                    if (edges[p].c > edges[p].f && h[edges[p].u] > h[edges[p].v
                        ]) {
                        bool need = (edges[p].v != s && edges[p].v != t && !
                            excess[edges[p].v]);
                        push(current[x]);
                        if (need) qu[1-now].push(edges[p].v);
                        if (!excess[x]) break;
                    }
                    current[x] = edges[current[x]].next;
                }

                if (excess[x] > 0) {
                    lift(x);
                    current[x] = head[x];
                    qu[1-now].push(x);
                    cnt++;
                    if (cnt == n) {
                        globalLabeling();
                        cnt=0;
                    }
                }
            }
            now = 1 - now;
        }
        return excess[t];
    }

private:
    void init() {
        REP(i,n) current[i] = head[i];

        int p = head[s];
        while (p >= 0) {
            edges[p].f = edges[p].c;
            edges[p^1].f -= edges[p].c;
            excess[edges[p].v] += edges[p].c;
            excess[s] -= edges[p].c;
            p = edges[p].next;
        }
        for(int v = 0; v < n; ++v) h[v] = 1;
        h[s] = n; h[t] = 0;
    }

    void push(int i) {
        long long delta = min(excess[edges[i].u], (long long) edges[i].c -
            edges[i].f);
        edges[i].f += delta; edges[i^1].f -= delta;
        excess[edges[i].u] -= delta;
        excess[edges[i].v] += delta;
    }

    void lift(int u) {
        int minH = 2 * n;
        int p = head[u];
        while (p>=0) {
            if (edges[p].c > edges[p].f)
                minH = min(minH, h[edges[p].v]);
            p = edges[p].next;
        }
        h[u] = minH + 1;
    }

    void globalLabeling() {
        REP(i,n) avail[i] = 1, h[i] = 0;
        h[s] = n; h[t] = 0;
        queue<int> q; q.push(t); avail[t] = false;

        while (!q.empty()) {
            int x = q.front(); q.pop();
            int p = head[x];
            while (p >= 0) {
                int pp = p^1;
                if (avail[edges[pp].u] && edges[pp].f < edges[pp].c) {
                    h[edges[pp].u] = h[x] + 1;
                    avail[edges[pp].u] = 0;
                    q.push(edges[pp].u);
                }
                p = edges[p].next;
            }
            if (q.empty() && avail[s]) {
                avail[s] = false;
                q.push(s);
            }
        }
        REP(x,n) current[x] = head[x];
    }
};
```

27 GomoryHu

```
vector<vector<int> > gomoryHu(DinicFlow &network) {
    int n = network.n;
    vector<vector<int> > d (n, vector<int>(n, network.INF));
    vector<int> p (n, 0);
        for(int i = 1; i < n; ++i) {
                network.resetFlow();
                int flow = network.maxFlow(i, p[i]);
                for(int j = i + 1; j < n; ++j)
                        if(network.dist[j] >= 0 && p[j] == p[i])
                                p[j] = i;
                d[i][p[i]] = d[p[i]][i] = flow;
                for(int j = 0; j < i; ++j) d[i][j] = d[j][i] = min(flow, d[p[i
                    ]][j]);
        }
    return d;
}
```

28 GlobalMinCut

```
// Adjacency matrix implementation of Stoer-Wagner min cut algorithm.
//
// Running time:
//     O(|V|^3)
//
// INPUT:
//     - graph, constructed using AddEdge()
//
// OUTPUT:
//     - (min cut value, nodes in half of min cut)

typedef vector<int> VI;
typedef vector<VI> VVI;

const int INF = 1000000000;

pair<int, VI> GetMinCut(VVI &weights) {
    int N = weights.size();
    VI used(N), cut, best_cut;
    int best_weight = -1;

    for (int phase = N-1; phase >= 0; phase--) {
        VI w = weights[0];
        VI added = used;
        int prev, last = 0;
        for (int i = 0; i < phase; i++) {
            prev = last;
            last = -1;
            for (int j = 1; j < N; j++)
                if (!added[j] && (last == -1 || w[j] > w[last])) last = j;
            if (i == phase-1) {
                for (int j=0; j<N; j++) weights[prev][j] += weights[last][j];
                for (int j=0; j<N; j++) weights[j][prev] = weights[prev][j];
                used[last] = true;
                cut.push_back(last);
                if (best_weight == -1 || w[last] < best_weight) {
                    best_cut = cut;
                    best_weight = w[last];
                }
            } else {
                for (int j = 0; j < N; j++)
                    w[j] += weights[last][j];
                added[last] = true;
            }
        }
    }
    return make_pair(best_weight, best_cut);
}
```

29 MaxFlowMinCost

```
#define FORE(i,v) for (__typeof((v).begin()) i=(v).begin();i!=(v).end();i++)

class MaxFlowMinCost {
    private:
    static const int INF=(int)1e9+7;
    struct Edge {
        int from,to,capa,flow,cost;
        Edge() {
            from=to=capa=flow=cost=0;
        }
        Edge(int u,int v,int ca,int co) {
            from=u;to=v;capa=ca;flow=0;cost=co;
        }
        int residental(void) const {
            return (capa-flow);
        }
    };
    int n;
    vector<vector<int> > adj;
    vector<Edge> edge;
    vector<int> dist,trace;
    bool FordBellman(int s,int t) {
        fill(dist.begin(), dist.end(), INF);
        fill(trace.begin(), trace.end(), -1);
        vector<bool> inQueue(n+7,false);
        queue<int> q;
        dist[s]=0;q.push(s);inQueue[s]=true;
        while (!q.empty()) {
            int u=q.front();q.pop();inQueue[u]=false;
            FORE(it,adj[u]) if (edge[*it].residental()>0) {
                int v=edge[*it].to;
                if (dist[v]>dist[u]+edge[*it].cost) {
                    dist[v]=dist[u]+edge[*it].cost;
                    trace[v]=*it;
                    if (!inQueue[v]) {
                        q.push(v);inQueue[v]=true;
                    }
                }
            }
        }
        return (dist[t]<INF);
    }
    public:
    MaxFlowMinCost() {
        n=0;
    }
    MaxFlowMinCost(int n) {
        this->n=n;
        adj.assign(n+7,vector<int>());
        dist.assign(n+7,0);
        trace.assign(n+7,0);
    }
    void addEdge(int u,int v,int ca,int co) {
        adj[u].push_back(edge.size());
        edge.push_back(Edge(u,v,ca,co));
        adj[v].push_back(edge.size());
        edge.push_back(Edge(v,u,0,-co));
    }
    pair<int,int> getFlow(int s,int t) {
        int totFlow=0;
        int totCost=0;
        while (FordBellman(s,t)) {
            int delta=INF;
            for (int u=t;u!=s;u=edge[trace[u]].from)
                delta=min(delta,edge[trace[u]].residental());
            for (int u=t;u!=s;u=edge[trace[u]].from) {
                edge[trace[u]].flow+=delta;
                edge[trace[u]^1].flow-=delta;
            }
            totFlow+=delta;
            totCost+=delta*dist[t];
        }
        return (make_pair(totFlow,totCost));
    }
};
```

30 MaxFlowMinCostDijkstra

```
// Source: rng_58: http://codeforces.com/contest/277/submission/3212642
// Fast min cost max flow (Dijkstra)
// Index from 0
// NOTE!!!!!! Flow through both direction can be > 0
// --> need to be careful when trace
// Does not work when cost < 0

#define REP(i,n) for(int i=0,_n=(n);i<_n;++i)
#define F_INF 1000111000LL
#define C_INF 1000111000LL

template<class Flow = long long, class Cost = long long>
struct MinCostFlow {
    int V, E;
    vector<Flow> cap;
    vector<Cost> cost;
    vector<int> to, prev;

    vector<Cost> dist, pot;
    vector<int> last, path, used;
    priority_queue< pair<Cost, int> > q;

    MinCostFlow(int V, int E) : V(V), E(0), cap(E*2,0), cost(E*2,0), to(E*2,0),
         prev(E*2,0), dist(V,0), pot(V,0), last(V, -1), path(V,0), used(V,0) {}

    int addEdge(int x, int y, Flow f, Cost c) {
        cap[E]=f; cost[E]=c; to[E]=y; prev[E]=last[x]; last[x]=E; E++;
        cap[E]=0; cost[E]=-c; to[E]=x; prev[E]=last[y]; last[y]=E; E++;
        return E - 2;
    }

    pair<Flow, Cost> search(int s, int t) {
        Flow ansf = 0; Cost ansc = 0;
        REP(i,V) used[i] = false;
        REP(i,V) dist[i] = C_INF;

        dist[s] = 0; path[s] = -1; q.push(make_pair(0, s));
        while (!q.empty()) {
            int x = q.top().second; q.pop();
            if (used[x]) continue; used[x] = true;
            for(int e = last[x]; e >= 0; e = prev[e]) if (cap[e] > 0) {
                Cost tmp = dist[x] + cost[e] + pot[x] - pot[to[e]];
                if (tmp < dist[to[e]] && !used[to[e]]) {
                    dist[to[e]] = tmp;
                    path[to[e]] = e;
                    q.push(make_pair(-dist[to[e]], to[e]));
                }
            }
        }
        REP(i,V) pot[i] += dist[i];
        if (used[t]) {
            ansf = F_INF;
            for(int e=path[t];e>=0;e=path[to[e^1]]) ansf=min(ansf, cap[e]);
            for(int e=path[t]; e >= 0; e=path[to[e^1]]) { ansc += cost[e] *
                 ansf; cap[e] -= ansf; cap[e^1] += ansf; }
        }
        return make_pair(ansf, ansc);
    }
    pair<Flow, Cost> minCostFlow(int s, int t) {
        Flow ansf = 0; Cost ansc = 0;
        while (1) {
            pair<Flow, Cost> p = search(s, t);
            if (!used[t]) break;
            ansf += p.first; ansc += p.second;
        }
        return make_pair(ansf, ansc);
    }
};
```

31 AhoCorasick

```
const int NODE = (int) 1e6 + 1;
const int NC = 26;

int nextNode[NODE][NC];
int chr[NODE];
int parent[NODE];
int prefix[NODE];
int numNodes;
set<int> match[NODE];

int getPrefix(int);
int go(int u, int c) {
    if (nextNode[u][c] != -1) return nextNode[u][c];
    if (u == 0) return 0;
    return nextNode[u][c] = go(getPrefix(u), c);
}

int getPrefix(int u) {
    if (prefix[u] != -1) return prefix[u];
    if (u == 0 || parent[u] == 0) return prefix[u] = 0;
    return prefix[u] = go(getPrefix(parent[u]), chr[u]);
}

void add(const string &s, int id) {
    int u = 0;
    for (int i = 0; i < (int) s.size(); ++i) {
        int c = s[i] - 'A';
        if (nextNode[u][c] == -1) {
            nextNode[u][c] = numNodes;
            fill(nextNode[numNodes], nextNode[numNodes] + NC, -1);
            chr[numNodes] = c;
            parent[numNodes] = u;
            prefix[numNodes] = -1;
            match[numNodes].clear();
            match[numNodes].insert(-1);
            ++numNodes;
        }
        u = nextNode[u][c];
    }
    match[u].insert(id);
}

set<int>& getMatch(int u) {
    if (match[u].count(-1) == 0) return match[u];
    const set<int> &foo = getMatch(getPrefix(u));
    match[u].insert(foo.begin(), foo.end());
    match[u].erase(-1);
    return match[u];
}

void init() {
    fill(nextNode[0], nextNode[0] + NC, -1);
    numNodes = 1;
}
```

32 SuffixArrayPrefixDoubling

```
struct SuffixArray {
    string a;
    int N, m;
    vector<int> SA, LCP, x, y, w, c;

    SuffixArray(string _a, int m) : a(" " + _a), N(a.length()), m(m),
            SA(N), LCP(N), x(N), y(N), w(max(m, N)), c(N) {
        a[0] = 0;
        DA();
        kasaiLCP();
        #define REF(X) { rotate(X.begin(), X.begin()+1, X.end()); X.pop_back();
             }
        REF(SA); REF(LCP);
        a = a.substr(1, a.size());
        for(int i = 0; i < (int) SA.size(); ++i) --SA[i];
        #undef REF
    }

    inline bool cmp (const int a, const int b, const int l) { return (y[a] == y
        [b] && y[a + l] == y[b + l]); }

    void Sort() {
        for(int i = 0; i < m; ++i) w[i] = 0;
        for(int i = 0; i < N; ++i) ++w[x[y[i]]];
        for(int i = 0; i < m - 1; ++i) w[i + 1] += w[i];
        for(int i = N - 1; i >= 0; --i) SA[--w[x[y[i]]]] = y[i];
    }

    void DA() {
        for(int i = 0; i < N; ++i) x[i] = a[i], y[i] = i;
        Sort();
        for(int i, j = 1, p = 1; p < N; j <<= 1, m = p) {
            for(p = 0, i = N - j; i < N; i++) y[p++] = i;
            for (int k = 0; k < N; ++k) if (SA[k] >= j) y[p++] = SA[k] - j;
            Sort();
            for(swap(x, y), p = 1, x[SA[0]] = 0, i = 1; i < N; ++i)
                x[SA[i]] = cmp(SA[i - 1], SA[i], j) ? p - 1 : p++;
        }
    }

    void kasaiLCP() {
        for (int i = 0; i < N; i++) c[SA[i]] = i;
        for (int i = 0, j, k = 0; i < N; LCP[c[i++]] = k)
            if (c[i] > 0) for (k ? k-- : 0, j = SA[c[i] - 1]; a[i + k] == a[j +
                 k]; k++);
            else k = 0;
    }
};

int main() {
    SuffixArray sa ("mississippi", 256);
    for (int i = 0; i < sa.N - 1; ++i) cout << sa.SA[i] << ' '; cout << '\n';
    for (int i = 0; i < sa.N - 1; ++i) cout << sa.LCP[i] << ' '; cout << '\n';
    // 10 7 4 1 0 9 8 6 3 5 2
    // 0 1 1 4 0 0 1 0 2 1 3
    return 0;
}
```

33 LexicographicallyMaximalStringRotation

```
vector<int> maxLex(vector<int> s) {
        s.insert(s.end(), s.begin(), s.end());
        s.push_back(numeric_limits<int>::min());
        int i = 0, j = 1, n = s.size();
        while(j < n) {
                int k = 0;
                while(j + k < n - 1 && s[i + k] == s[j + k]) ++k;
                if(s[i + k] < s[j + k]) i += k + 1;
                else j += k + 1;
                if(i == j) ++j;
        }
        return vector<int>(s.begin() + i, s.begin() + i + n / 2);
}
```

34 Lyndon

```
// Decompose s = w1w2..wk : k max and w1 >= w2 >= ...

vector<string> lyndon(string s) {
    int n = (int) s.length();
    int i = 0;
    vector<string> result;
    while (i < n) {
        int j = i + 1, k = i;
        while (j < n && s[k] <= s[j]) {
            if (s[k] < s[j]) k = i;
            else ++k;
            ++j;
        }
        while (i <= k) {
            result.push_back(s.substr(i, j - k));
            i += j - k;
        }
    }
    return result;
}

int main() {
    auto f = [](vector<string>x){for(auto y:x)cout<<y<<' ';cout<<'\n';};
    f(lyndon("abcdef")); // abcdef
    f(lyndon("fedcba")); // f e d c b a
    f(lyndon("aaaaaa")); // a a a a a a
    f(lyndon("ababab")); // ab ab ab
    return 0;
}
```

35 ZFunction

```
vector<int> calcZ(const string &s) {
    int n = s.size();
    vector<int> z (n);
    for (int i = 1, j = 0; i < n; ++i) {
        if (j + z[j] > i) z[i] = min(j + z[j] - i, z[i - j]);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]]) ++z[i];
        if (j + z[j] <= i || i + z[i] > j + z[j]) j = i;
    }
    return z;
}
```

36 SuffixArrayDC3

```
#include <bits/stdc++.h>
#define FOR(i,a,b) for (int i=(a),_b=(b);i<=_b;i=i+1)
#define REP(i,n) for (int i=0,_n=(n);i<_n;i=i+1)
#define MASK(i) (1LL<<(i))
#define BIT(x,i) (((x)>>(i))&1)
#define tget(i) BIT(t[(i) >> 3], (i) & 7)
#define tset(i, b) { if (b) t[(i) >> 3] |= MASK((i) & 7); else t[(i) >> 3] &= ~
    MASK((i) & 7); }
#define chr(i) (cs == sizeof(int) ? ((int *)s)[i] : ((unc *)s)[i])
#define isLMS(i) ((i) > 0 && tget(i) && !tget((i) - 1))

typedef unsigned char unc;
class SuffixArray {
    public:
    int *sa, *lcp, *rank, n;
    unc *s;
    void getbuckets(unc s[], vector<int> &bkt, int n, int k, int cs, bool end)
        {
        FOR(i, 0, k) bkt[i] = 0;
        REP(i, n) bkt[chr(i)]++;
        int sum = 0;
        FOR(i, 0, k) {
            sum += bkt[i];
            bkt[i] = end ? sum : sum - bkt[i];
        }
    }
    void inducesal(vector<unc> &t, int sa[], unc s[], vector<int> &bkt, int n,
        int k, int cs, bool end) {
        getbuckets(s, bkt, n, k, cs, end);
        REP(i, n) {
            int j = sa[i] - 1;
            if (j >= 0 && !tget(j)) sa[bkt[chr(j)]++] = j;
        }
    }
    void inducesas(vector<unc> &t, int sa[], unc s[], vector<int> &bkt, int n,
        int k, int cs, bool end) {
        getbuckets(s, bkt, n, k, cs, end);
        FORD(i, n - 1, 0) {
            int j = sa[i] - 1;
            if (j >= 0 && tget(j)) sa[--bkt[chr(j)]]=j;
        }
    }
    void build(unc s[], int sa[], int n, int k, int cs) {
        int j;
        vector<unc> t = vector<unc>(n / 8 + 1, 0);
        tset(n - 2, 0);
        tset(n - 1, 1);
        FORD(i, n - 3, 0) tset(i, chr(i) < chr(i+1) || (chr(i) == chr(i+1) &&
            tget(i+1)));
        vector<int> bkt = vector<int> (k + 1, 0);
        getbuckets(s, bkt, n, k, cs, true);
        REP(i, n) sa[i] = -1;
        REP(i, n) if (isLMS(i)) sa[--bkt[chr(i)]] = i;
        inducesal(t, sa, s, bkt, n, k, cs, false);
        inducesas(t, sa, s, bkt, n, k, cs, true);
        bkt.clear();
        int n1 = 0;
        REP(i, n) if (isLMS(sa[i])) sa[n1++] = sa[i];
        FOR(i, n1, n - 1) sa[i] = -1;
        int name = 0;
        int prev = -1;
        REP(i, n1) {
            int pos = sa[i];
            bool diff = false;
            REP(d, n) {
                if (prev < 0 || chr(prev + d) != chr(pos + d) || tget(prev + d)
                     != tget(pos + d)) {
                    diff = true;
                    break;
                }
                else if (d > 0 && (isLMS(prev + d) || isLMS(pos + d))) break;
            }
            if (diff) {
                name++;
                prev = pos;
            }
            sa[n1 + pos / 2] = name - 1;
        }
        j = n - 1;
        FORD(i, n - 1, n1) if (sa[i] >= 0) sa[j--] = sa[i];
        int *sa1 = sa;
        int *s1 = sa + n - n1;
        if (name < n1) build((unc *)s1, sa1, n1, name-1, sizeof(int));
        else REP(i, n1) sa1[s1[i]] = i;
        bkt.assign(k + 1, 0);
        getbuckets(s, bkt, n, k, cs, true);
        j = 0;
        REP(i, n) if (isLMS(i)) s1[j++] = i;
        REP(i, n1) sa1[i] = s1[sa1[i]];
        FOR(i, n1, n - 1) sa[i] = -1;
        FORD(i, n1 - 1, 0) {
            j = sa[i];
            sa[i] = -1;
            sa[--bkt[chr(j)]] = j;
        }
        inducesal(t, sa, s, bkt, n, k, cs, false);
        inducesas(t, sa, s, bkt, n, k, cs, true);
        bkt.clear();
        t.clear();
    }
    void calc_lcp(void) {
        FOR(i,1,n) rank[sa[i]] = i;
        int h = 0;
        REP(i, n) if (rank[i] < n) {
            int j = sa[rank[i] + 1];
            while (s[i + h] == s[j + h]) h++;
            lcp[rank[i]] = h;
            if (h > 0) h--;
        }
    }
    SuffixArray() {
        n = 0;
        sa = lcp = rank = NULL;
        s=NULL;
    }
    SuffixArray(string ss) {
        n = ss.size();
        sa = new int[n + 7];
        lcp = new int [n + 7];
        rank = new int [n + 7];
        s = (unc *)ss.c_str();
        build(s, sa, n + 1, 256, sizeof(char));
        calc_lcp();
    }
};

//Sorted suffices are SA[1] to SA[N]. The values of SA[1], SA[2], ..., SA[N]
    are 0, 1, ..., N - 1
//The longest common prefix of SA[i] and SA[i + 1] is LCP[i]

int main(void) {
    string s = "mississippi";
    SuffixArray suffixArray(s);
    FOR(i, 1, 11) printf("%d %s %d\n", suffixArray.sa[i], s.substr(suffixArray.
        sa[i]).c_str(), suffixArray.lcp[i]);
}
```

37 Manacher

```
vector<int> manacher(const string &os) {
    int n = os.size();
    string s;
    for(int i = 0; i < n; ++i) {
        s += os[i];
        if(i != n - 1) s += '$';
    }
    int mx = 0, id = 0;
    n = s.size();
    vector<int> p (n);
    for(int i = 0; i < n; ++i) {
        p[i] = mx > i ? min(p[2 * id - i], mx - i) : 1;
        while(p[i] <= i && i + p[i] < n && s[i - p[i]] == s[i + p[i]]) ++p[i];
        if(i + p[i] > mx) {
            mx = i + p[i];
            id = i;
        }
    }
    return p;
}
```

38 SuffixAutomaton

```
const int MAX_N = int(1e5) + 4;
const int MAX_SAM = 2 * MAX_N;
// a node in the "directed acyclic word graph" (or simply "DAWG")
struct State {
    // len: length of the path from root to this node (number of edges)
    // link: link to a node which is a suffix of this state
    // nexts: the node which is adjanced with this node by an edge ('a'..'z',
         '0'..'9', ...)
    int len, link;
    map<int, int> nexts;
    State() {
        len = 0;
        link = -1;
        nexts.clear();
    }
    void operator = (const State &other) {
        len = other.len;
        link = other.link;
        nexts = other.nexts;
    }
    bool hasNext(int x) {
        return nexts.find(x) != nexts.end();
    }
};
void sam_init() {
    nSAM = 1;       // number of nodes
    last = 0;       // id of the last node (start from 0)
    sam[0] = State();   // this is the root node
    f[0] = 0;           // for some applications
}
void sam_extend(int x) {
    int cur = nSAM++;       // id of new node
    sam[cur] = State();
    sam[cur].len = sam[last].len + 1;
    f[cur] = 1;
    int p = last;
    for (; p != -1 && !sam[p].hasNext(x); p = sam[p].link)
        sam[p].nexts[x] = cur;
    if (p == -1) sam[cur].link = 0;
    else {
        int q = sam[p].nexts[x];
        if (sam[q].len == sam[p].len + 1) sam[cur].link = q;
        else {
            int clone = nSAM++;     // create a clone node of q
            sam[clone] = sam[q];
            sam[clone].len = sam[p].len + 1;
            f[clone] = 0;
            for (; p != -1 && sam[p].nexts[x] == q; p = sam[p].link)
                sam[p].nexts[x] = clone;
            sam[cur].link = sam[q].link = clone;
        }
    }
    last = cur;
}
// // APLICATIONS
// // we should do the topo sort (by length) before implement other features
// for (int i = 0; i <= n; ++i) c[i] = 0;
// for (int i = 0; i < nSAM; ++i) ++c[sam[i].len];
// for (int i = 1; i <= n; ++i) c[i] += c[i-1];
// for (int i = 0; i < nSAM; ++i) id[--c[sam[i].len]] = i;
// // number of occurrents of state u, which corresponding with
// // number of occurrents of each substrings from root to u
// for (int i = nSAM-1; i >= 0; --i) {
//  int u = id[i];
//  f[sam[u].link] += f[u];
// }
// // number of ways to go from root to state u, which corresponding with
// // number of different substrings end at u.
// // If we sort these strings increasing by their lengths,
// // then the i-th string is a suffix of the (i+1)-th string.
// fill(g, 0);
// g[0] = 1;
// for (int i = 0; i < nSAM; ++i) {
//  int u = id[i];
//  tr(sam[u].nexts, it)
//      g[it->second] += g[u];
// }
// // number of substrings which have state u as its prefix
// for (int i = nSAM-1; i >= 0; --i) {
//  int u = id[i];
//  f[u] = 1;
//  tr(sam[u].nexts, it)
//      f[u] += [it->second];
// }
```

$$\pi(x) = \lfloor x \rfloor - \sum_{i=1}^{a} \left\lfloor \frac{x}{p_i} \right\rfloor + \sum_{1 \le i \le j \le a} \left\lfloor \frac{x}{p_i p_j} \right\rfloor - \ldots + \frac{1}{2}(b+a-2)(b-a+1) - \sum_{a < i \le b} \pi\left(\frac{x}{p_i}\right) - \sum_{i=a+1}^{c} \sum_{j=i}^{b_i} \left[\pi\left(\frac{x}{p_i p_j}\right) - (j-1)\right], a = \pi\left(x^{1/4}\right), b = \pi\left(x^{1/2}\right), b_i = \pi\left(\sqrt{x/p_i}\right), c = \pi\left(x^{1/3}\right)$$

$$C_n = \binom{2n}{n} - \binom{2n}{n+1} = \frac{1}{n+1}\binom{2n}{n}; C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i} = \frac{2(2n+1)}{n+2} C_n$$

$C = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440$

Number of permutations of length n with k cycles:

$$s(n+1,k) = ns(n,k) + s(n,k-1)$$

Number of ways to partition a set of n labelled objects into k nonempty subsets:

$$S(n,k) = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j} j^n = kS(n-1,k) + S(n,k-1)$$

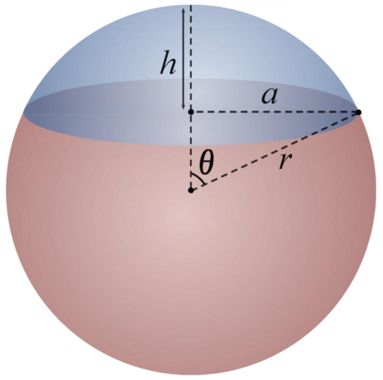
$$H_n = \sum_{k=1}^{n} \frac{1}{k} \approx \ln n + \gamma + \frac{1}{2n} - \frac{1}{12n^2} + \frac{1}{120n^4} - \frac{1}{252n^6} + \ldots$$

$$\frac{1}{2(n+1)} < H_n - \ln n - \gamma < \frac{1}{2n}; \frac{1}{24(n+1)^2} < H_n - \ln\left(n + \frac{1}{2}\right) - \gamma < \frac{1}{24n^2}$$

$$\gamma = 0.5772156649015328606065120900824024310421593359 3992$$

Sphere: $V = \frac{4}{3}\pi r^3; A = 4\pi r^2$

$V = \frac{\pi h}{6}\left(3a^2 + h^2\right); A = 2\pi r h = 2\pi r^2 (1 - \cos\theta) = \pi\left(a^2 + h^2\right); r = \frac{a^2 + h^2}{2h}$



Maximum Flows with Edge Demands: $c'(s' \to v) = \sum_{u \in V} d(u \to v)$, $c'(v \to t') = \sum_{w \in V} d(v \to w)$, $c'(u \to v) = c(u \to v) - d(u \to v)$, $c'(t \to s) = \infty$. If feasible: $c_f(u \to v) = c(u \to v) - f(u \to v)$ if $u \to v \in E$; $f(v \to u) - d(v \to u)$ if $v \to u \in E$, 0 otherwise.