Team Notebook

HSG SQRT

November 11, 2022

Contents

1 Combinatorics

1.1 Permutations

1.1.1 Factorial

n	1	2	3	4	5	6	7	8	9	10
$n!$	1	2	6	24	120	720	5040	40320	362880	3628800

n	11	12	13	14	15	16	17
$n!$	4.0e7	4.8e8	6.2e9	8.7e10	1.3e12	2.1e13	3.6e14

n	20	25	30	40	50	100	150	171
$n!$	2e18	2e25	3e32	8e47	3e64	9e157	6e262	>DBL_MAX

1.1.2 Cycles

Let $g_S(n)$ be the number of n-permutations whose cycle lengths all belong to the set S. Then

$$\sum_{n=0}^{\infty} g_S(n)\frac{x^n}{n!} = \exp\left(\sum_{n\in S}\frac{x^n}{n}\right)$$

1.1.3 Derangements

Permutations of a set such that none of the elements appear in their original position.

$$D(n) = (n-1)(D(n-1)+D(n-2)) = nD(n-1)+(-1)^n = \left\lfloor \frac{n!}{e} \right\rceil$$

1.1.4 Burnside's lemma

Given a group G of symmetries and a set X, the number of elements of X *up to symmetry* equals

$$\frac{1}{|G|}\sum_{g\in G}|X^g|,$$

where X^g are the elements fixed by g ($g.x = x$).

If $f(n)$ counts "configurations" (of some sort) of length n, we can ignore rotational symmetry using $G = Z_n$ to get

$$g(n) = \frac{1}{n}\sum_{k=0}^{n-1} f(\gcd(n,k)) = \frac{1}{n}\sum_{k|n} f(k)\phi(n/k).$$

1.2 Partitions and subsets

1.2.1 Partition function

Number of ways of writing n as a sum of positive integers, disregarding the order of the summands.

$$p(0) = 1,\ p(n) = \sum_{k\in Z\setminus\{0\}} (-1)^{k+1} p(n-k(3k-1)/2)$$

$$p(n) \sim 0.145/n \cdot \exp(2.56\sqrt{n})$$

n	0	1	2	3	4	5	6	7	8	9	20	50	100
$p(n)$	1	1	2	3	5	7	11	15	22	30	627	∼2e5	∼2e8

1.2.2 Lucas' Theorem

Let n, m be non-negative integers and p a prime. Write $n = n_k p^k + ... + n_1 p + n_0$ and $m = m_k p^k + ... + m_1 p + m_0$. Then $\binom{n}{m} \equiv \prod_{i=0}^{k}\binom{n_i}{m_i} \pmod p$.

1.3 General purpose numbers

1.3.1 Bernoulli numbers

EGF of Bernoulli numbers is $B(t) = \frac{t}{e^t - 1}$ (FFT-able). $B[0,\ldots] = [1, -\frac{1}{2}, \frac{1}{6}, 0, -\frac{1}{30}, 0, \frac{1}{42}, \ldots]$

Sums of powers:

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

Euler-Maclaurin formula for infinite sums:

$$\sum_{i=m}^{\infty} f(i) = \int_m^{\infty} f(x)dx - \sum_{k=1}^{\infty}\frac{B_k}{k!}f^{(k-1)}(m)$$

$$\approx \int_m^{\infty} f(x)dx + \frac{f(m)}{2} - \frac{f'(m)}{12} + \frac{f'''(m)}{720} + O(f^{(5)}(m))$$

1.3.2 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

$$c(n,k) = c(n-1,k-1) + (n-1)c(n-1,k),\ c(0,0) = 1$$
$$\textstyle\sum_{k=0}^{n} c(n,k)x^k = x(x+1)\ldots(x+n-1)$$

$c(8,k) = 8, 0, 5040, 13068, 13132, 6769, 1960, 322, 28, 1$
$c(n,2) = 0, 0, 1, 3, 11, 50, 274, 1764, 13068, 109584, \ldots$

1.3.3 Eulerian numbers

Number of permutations $\pi \in S_n$ in which exactly k elements are greater than the previous element. k j:s s.t. $\pi(j) > \pi(j+1)$, $k+1$ j:s s.t. $\pi(j) \geq j$, k j:s s.t. $\pi(j) > j$.

$$E(n,k) = (n-k)E(n-1,k-1) + (k+1)E(n-1,k)$$
$$E(n,0) = E(n,n-1) = 1$$
$$E(n,k) = \sum_{j=0}^{k}(-1)^j\binom{n+1}{j}(k+1-j)^n$$

1.3.4 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

1.3.5 Bell numbers

Total number of partitions of n distinct elements. $B(n) = 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, \ldots$. For p prime,

$$B(p^m + n) \equiv mB(n) + B(n+1) \pmod p$$

1.3.6 Labeled unrooted trees

\# on n vertices: n^{n-2}
\# on k existing trees of size n_i: $n_1 n_2 \cdots n_k n^{k-2}$
\# with degrees d_i: $(n-2)!/((d_1-1)!\cdots(d_n-1)!)$

1.3.7 Catalan numbers

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \frac{(2n)!}{(n+1)!n!}$$

$$C_0 = 1,\ C_{n+1} = \frac{2(2n+1)}{n+2}C_n,\ C_{n+1} = \sum C_i C_{n-i}$$

$C_n = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, \ldots$

- sub-diagonal monotone paths in an $n \times n$ grid.
- strings with n pairs of parenthesis, correctly nested.
- binary trees with with $n + 1$ leaves (0 or 2 children).
- ordered trees with $n + 1$ vertices.
- ways a convex polygon with $n+2$ sides can be cut into triangles by connecting vertices with straight lines.
- permutations of $[n]$ with no 3-term increasing subseq.

2 Data Structure

2.1 Gilbert Order

```
inline int64_t gilbertOrder(int x, int y, int pow = 21, int
    rotate = 0) {
 if (pow == 0) {
  return 0;
 }
 int hpow = 1 << (pow-1);
 int seg = (x < hpow) ? (
  (y < hpow) ? 0 : 3
 ) : (
  (y < hpow) ? 1 : 2
 );
 seg = (seg + rotate) & 3;
 const int rotateDelta[4] = {3, 0, 0, 1};
 int nx = x & (x ^ hpow), ny = y & (y ^ hpow);
 int nrot = (rotate + rotateDelta[seg]) & 3;
 int64_t subSquareSize = int64_t(1) << (2*pow - 2);
 int64_t ans = seg * subSquareSize;
 int64_t add = gilbertOrder(nx, ny, pow-1, nrot);
 ans += (seg == 1 || seg == 2) ? add : (subSquareSize - add
     - 1);
 return ans;
}
```

2.2 Li Chao Tree

```
const int N = 2e5 + 5;

struct line {
    ll a, b;
    ll val(ll x) { return a * x + b; }
};

line seg[4 * N];

void upd(int id, int l, int r, line lin) {
    if (l == r) {
        if (lin.val(l) > seg[id].val(l))
            seg[id] = lin;
        return;
    }
    int mid = (l + r) >> 1;
    if (seg[id].a > lin.a)
        swap(seg[id], lin);
    // sap xep lai do doc duong thang de tong quat hoa bai
        toan
    if (lin.val(mid) > seg[id].val(mid)) {
        swap(seg[id], lin);
        upd(id << 1, l, mid, lin);
    } else
        upd(id << 1 | 1, mid + 1, r, lin);
}

void upd_range(int id, int l, int r, int u, int v, line lin)
     {
    if (l > v || r < u)
        return;
    if (l >= u && r <= v) {
        upd(id, l, r, lin);
        return;
    }
    int mid = (l + r) >> 1;
    upd_range(id << 1, l, mid, u, v, lin);
    upd_range(id << 1 | 1, mid + 1, r, u, v, lin);
}

ll get(int id, int l, int r, int pos) {
    if (l == pos && l == r) {
        return seg[id].val(pos);
    }
    int mid = (l + r) >> 1;
    if (mid >= pos)
        return max(seg[id].val(pos), get(id << 1, l, mid, pos
            ));
    else
        return max(seg[id].val(pos), get(id << 1 | 1, mid +
            1, r, pos));
}
```

2.3 Persistent Segment Tree

```
struct Node{
    int mi;
    Node *l, *r;

    Node(int x){
        mi = x; l = nullptr; r = nullptr;
    }
    Node(Node *u, Node *v){
        l = u; r = v;
        mi = MAXA;
        if (l) mi = l -> mi;
        if (r) mi = min(mi, r -> mi);
    }
    Node(Node *u){
        mi = u -> mi;
        l = u -> l;
        r = u -> r;
    }
};

int a[N], last[N];
pair<int, int> r[N];
vector <int> v;
Node* root[N];

Node* build(int l, int r){
    if (l == r) return new Node(MAXA);

    int mid = (l + r) >> 1;
    return new Node(build(l, mid), build(mid + 1, r));
}

Node* upd(Node* node, int l, int r, int pos, int val){
    if (l == r) return new Node(val);

    int mid = (l + r) >> 1;
    if (pos > mid) return new Node(node -> l, upd(node -> r,
         mid + 1, r, pos, val));
    else return new Node(upd(node -> l, l, mid, pos, val),
         node -> r);
}
```

```
int get(Node* node, int l, int r, int u, int v){
    if (node == nullptr || l > v || r < u) return MAXA;
    if (l >= u && r <= v) return node -> mi;

    int mid = (l + r) >> 1;
    int v1 = get(node -> l, l, mid, u, v);
    int v2 = get(node -> r, mid + 1, r, u, v);
    return min(v1, v2);
}
```

2.4 Persistent Trie

```
/*
Usage: printf("%d\n", query(version[r], ~x, l));
-> Trie version[r] contains the trie for [1...r] elements
*/

struct node_t;
typedef node_t * pnode;

struct node_t {
  int time;
  pnode to[2];
  node_t() : time(0) {
    to[0] = to[1] = 0;
  }
  bool go(int l) const {
    if (!this) return false;
    return time >= l;
  }
  pnode clone() {
    pnode cur = new node_t();
    if (this) {
      cur->time = time;
      cur->to[0] = to[0];
      cur->to[1] = to[1];
    }
    return cur;
  }
};

pnode last;
pnode version[N];

void insert(int a, int time) {
  pnode v = version[time] = last = last->clone();
  for (int i = K - 1; i >= 0; --i) {
    int bit = (a >> i) & 1;
    pnode &child = v->to[bit];
    child = child->clone();
    v = child;
    v->time = time;
  }
}

int query(pnode v, int x, int l) {
  int ans = 0;
  for (int i = K - 1; i >= 0; --i) {
    int bit = (x >> i) & 1;
    if (v->to[bit]->go(l)) { // checking if this bit was
        inserted before the range
      ans |= 1 << i;
      v = v->to[bit];
    } else {
      v = v->to[bit ^ 1];
    }
  }
  return ans;
}
```

2.5 Segment Tree 2D

```
void build_y(int vx, int lx, int rx, int vy, int ly, int ry)
     {
    if (ly == ry) {
        if (lx == rx)
            segtree[vx][vy] = 0;
        else
            segtree[vx][vy] = max(segtree[vx*2][vy],segtree[
                 vx*2+1][vy]);
    } else {
        int my = (ly + ry) / 2;
        build_y(vx, lx, rx, vy*2, ly, my);
        build_y(vx, lx, rx, vy*2+1, my+1, ry);
        segtree[vx][vy] = max(segtree[vx][vy*2],segtree[vx][
             vy*2+1]);
    }
}

void build_x(int vx, int lx, int rx) {
    if (lx != rx) {
        int mx = (lx + rx) / 2;
        build_x(vx*2, lx, mx);
        build_x(vx*2+1, mx+1, rx);
    }
    build_y(vx, lx, rx, 1, 0, m - 1);
}
int query_y(int vx, int vy, int tly, int try_, int ly, int
     ry) {
    if (ly > ry)
        return 0;
    if (ly == tly && try_ == ry)
        return segtree[vx][vy];
    int tmy = (tly + try_) / 2;
    return max(query_y(vx, vy*2, tly, tmy, ly, min(ry, tmy))
        ,query_y(vx, vy*2+1, tmy+1, try_, max(ly, tmy+1), ry)
            );
}

int query_x(int vx, int tlx, int trx, int lx, int rx, int ly
     , int ry) {
    if (lx > rx)
        return 0;
    if (lx == tlx && trx == rx)
        return query_y(vx, 1, 0, m-1, ly, ry);
    int tmx = (tlx + trx) / 2;
    return max(query_x(vx*2, tlx, tmx, lx, min(rx, tmx), ly,
         ry)
         ,query_x(vx*2+1, tmx+1, trx, max(lx, tmx+1), rx, ly,
               ry));
}
void update_y(int vx, int lx, int rx, int vy, int ly, int ry
     , int x, int y, int new_val) {
    if (ly == ry) {
        if (lx == rx)
            segtree[vx][vy] = new_val;
        else
            segtree[vx][vy] = max(segtree[vx*2][vy], segtree[
                 vx*2+1][vy]);
    } else {
        int my = (ly + ry) / 2;
        if (y <= my)
            update_y(vx, lx, rx, vy*2, ly, my, x, y, new_val)
                ;
        else
            update_y(vx, lx, rx, vy*2+1, my+1, ry, x, y,
                 new_val);
        segtree[vx][vy] = max(segtree[vx][vy*2],segtree[vx][
             vy*2+1]);
    }
}

void update_x(int vx, int lx, int rx, int x, int y, int
     new_val) {
    if (lx != rx) {
        int mx = (lx + rx) / 2;
        if (x <= mx)
```

```
        update_x(vx*2, lx, mx, x, y, new_val);
      else
        update_x(vx*2+1, mx+1, rx, x, y, new_val);
    }
    update_y(vx, lx, rx, 1, 0, m-1, x, y, new_val);
}
```

2.6 STL

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

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/rope>

using namespace __gnu_pbds;
using namespace __gnu_cxx;
using namespace std;

typedef tree<int, null_type, less<int>, rb_tree_tag,
     tree_order_statistics_node_update> ordered_set;

unsigned hash_f(unsigned x) {
    x = ((x >> 16) ^ x) * 0x45d9f3b;
    x = ((x >> 16) ^ x) * 0x45d9f3b;
    x = (x >> 16) ^ x;
    return x;
}
struct chash {
    int operator()(int x) const { return hash_f(x); }
};

ordered_set s; //ordered_set
//s.find_by_order(x)
//s.order_of_key(x)
gp_hash_table<int, int, chash> mp; //hash map
rope <int> v; //rope (almost like string...)
```

2.7 Treap

```
struct node{
    node *l, *r;
    int sz, pri;
    int val, sum;
    bool rev;

    node(){}
    node(int c){
        l = r = nullptr;
        sz = 1;
        rev = false;
        pri = rng();
        val = c;
        sum = c;
    }
};

void push(node *&treap){
    if (treap == nullptr) return;
    if (!(treap -> rev)) return;
    swap(treap -> l, treap -> r);
    if (treap -> l){
        (treap -> l) -> rev ^= 1;
    }
    if (treap -> r){
        (treap -> r) -> rev ^= 1;
    }
    treap -> rev = false;
}

int get_sz(node *treap){
    if (treap == nullptr) return 0;
    else return treap -> sz;
}

int get_sum(node *treap){
    if (treap == nullptr) return 0;
    else return treap -> sum;
}

void split(node *treap, node *&l, node *&r, int k){ //[1, k]
     [k + 1, sz]
    if (treap == nullptr){
        l = r = nullptr;
        return;
    }
    push(treap);
    if (get_sz(treap -> l) < k){
        split(treap -> r, treap -> r, r, k - get_sz(treap ->
            l) - 1);
        l = treap;
    } else {
        split(treap -> l, l, treap -> l, k);
        r = treap;
    }
    treap -> sz = get_sz(treap -> l) + get_sz(treap -> r) +
         1;
    treap -> sum = get_sum(treap -> l) + get_sum(treap -> r)
         + treap -> val;
}

void merge(node *&treap, node *l, node *r){
    if (l == nullptr){
        treap = r;
        return;
    }
    if (r == nullptr){
        treap = l;
        return;
    }
    push(l); push(r);
    if (l -> pri < r -> pri){
        merge(l -> r, l -> r, r);
        treap = l;
    } else {
        merge(r -> l, l, r -> l);
        treap = r;
    }
    treap -> sz = get_sz(treap -> l) + get_sz(treap -> r) +
         1;
    treap -> sum = get_sum(treap -> l) + get_sum(treap -> r)
         + treap -> val;
}

void print(node *treap){
    if (treap == nullptr) return;
    push(treap);
    print(treap -> l);
    cout << treap -> val;
    print(treap -> r);
}
```

3 Dynamic Programming

3.1 DP Convex Hull Trick

```
// Decreasing Insertion, Query Min
struct CHT {
     vector<long long> a, b;

     bool cross(int i, int j, int k) {
          return 1.d*(a[j] - a[i])*(b[k] - b[i]) >= 1.d*(a[
               k] - a[i])*(b[j] - b[i]);
     }
```

```
    void add(long long A, long long B) {
        a.push_back(A);
        b.push_back(B);

        while (a.size() > 2 && cross(a.size() - 3, a.size
            () - 2, a.size() - 1)) {
        a.erase(a.end() - 2);
  b.erase(b.end() - 2);
 }
    }

    long long query(long long x) {
        int l = 0, r = a.size() - 1;

        while (l < r) {
            int mid = l + (r - l)/2;
  long long f1 = a[mid] * x + b[mid];
  long long f2 = a[mid + 1] * x + b[mid + 1];

  if (f1 > f2) l = mid + 1;
  else r = mid;
        }

        return a[l]*x + b[l];
    }
};
```

3.2 DP Divide and Conquer

```
void divide(int i, int L, int R, int optL, int optR) {
    if (L > R) return;
    int mid = (L+R) / 2, cut = optL;
    f[i][mid] = INF;
    for (int k = optL; k <= min(mid, optR); k++) {
        long long cur = f[i - 1][k] + Cost(k+1,mid);
        if (f[i][mid] > cur) {
            f[i][mid] = cur;
            cut = k;
        }
    }
    divide(i, L, mid - 1, optL, cut);
    divide(i, mid + 1, R, cut, optR);
}
```

3.3 DP Knuth

```
/*
* Complexity: O(N^2)
* f[i][j] = min(f[i][k] + f[k][j] + c[i][j], i < k < j)
* a[i][j] = min(k | i < k < j && f[i][j] = f[i][k] + f[k][j]
    + c[i][j])
* Sufficient condition: a[i][j - 1] <= a[i][j] <= a[i + 1][j
   ] or
* c[x][z] + c[y][t] <= c[x][t] + c[y][z] (quadrangle
   inequality) and c[y][z] <= c[x][t] (monotonicity), x <=
    y <= z <= t
*/

void knuth() {
    for (int i = 1; i <= n; i++) {
        f[i][i] = 0;
        a[i][i] = i;
    }
    for(int len = 1; len <= n-1;len++)
            for(int i = 1; i <= n-len;i++)
        {
            int j = i + len;
            f[i][j] = INF;
            for (int k = a[i][j - 1]; k <= a[i + 1][j]; k++)
            {
                if (f[i][j] > f[i][k-1] + f[k][j] + c[i][j])
                {
                    f[i][j] = f[i][k-1] + f[k][j] + c[i][j];
                    a[i][j] = k;
                }
            }
        }
    cout << f[1][n] << '\n';
}
```

3.4 Subset Sum Convolution

```
// Make fhat[][] = {0} and ghat[][] = {0}
for(int mask = 0; mask < (1 << N); mask++) {
    fhat[__builtin_popcount(mask)][mask] = f[mask];
    ghat[__builtin_popcount(mask)][mask] = g[mask];
}

// Apply zeta transform on fhat[][] and ghat[][]
for(int i = 0; i < N; i++) {
    for(int j = 0; j < N; j++) {
        for(int mask = 0; mask < (1 << N); mask++) {
            if((mask & (1 << j)) != 0) {
                fhat[i][mask] += fhat[i][mask ^ (1 << j)];
                ghat[i][mask] += ghat[i][mask ^ (1 << j)];
            }
        }
    }
}

// Do the convolution and store into h[][] = {0}
for(int mask = 0; mask < (1 << N); mask++) {
    for(int i = 0; i < N; i++) {
        for(int j = 0; j <= i; j++) {
            h[i][mask] += fhat[j][mask] * ghat[i - j][mask];
        }
    }
}

// Apply inverse SOS dp on h[][]
for(int i = 0; i < N; i++) {
    for(int j = 0; j < N; j++) {
        for(int mask = 0; mask < (1 << N); mask++) {
            if((mask & (1 << j)) != 0) {
                h[i][mask] -= h[i][mask ^ (1 << j)];
            }
        }
    }
}
for(int mask = 0; mask < (1 << N); mask++) fog[mask] = h[
    __builtin_popcount(mask)][mask];
```

4 Formula

4.1 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.

4.2 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. $a_n = (d_1 n + d_2) r^n$.

4.3 Trigonometry

$$\sin(v+w) = \sin v \cos w + \cos v \sin w$$
$$\cos(v+w) = \cos v \cos w - \sin v \sin w$$

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

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

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

$$a \cos x + b \sin x = r \cos(x - \phi)$$
$$a \sin x + b \cos x = r \sin(x + \phi)$$

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

4.4 Geometry

4.4.1 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): $s_a = \sqrt{bc\left[1 - \left(\frac{a}{b+c}\right)^2\right]}$

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: $\dfrac{a+b}{a-b} = \dfrac{\tan\frac{\alpha+\beta}{2}}{\tan\frac{\alpha-\beta}{2}}$

4.4.2 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 $A = \sqrt{(p-a)(p-b)(p-c)(p-d)}$.

4.4.3 Spherical coordinates

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

4.5 Sums

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

4.6 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) $\mu = E(X) = \sum_x x p_X(x)$ and variance $\sigma^2 = V(X) = E(X^2) - (E(X))^2 = \sum_x (x - E(X))^2 p_X(x)$ 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:

$$E(aX + bY) = aE(X) + bE(Y)$$

For independent X and Y,

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

4.6.1 Discrete distributions

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.

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 $\mathrm{Fs}(p)$, $0 \leq p \leq 1$.

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

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 $\mathrm{Po}(\lambda)$, $\lambda = t\kappa$.

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

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

5 Geometry

5.1 Angle

```
/**
 * Description: A class for ordering angles (as represented
     by int points and
 *  a number of rotations around the origin). Useful for
     rotational sweeping.
 *  Sometimes also represents points or vectors.
 * Usage:
 *  vector<Angle> v = {w[0], w[0].t360() ...}; // sorted
 *  int j = 0; rep(i,0,n) { while (v[j] < v[i].t180()) ++j; }
 *  // sweeps j such that (j-i) represents the number of
     positively oriented triangles with vertices at 0 and i
 * Status: Used, works well
 */
#pragma once

struct Angle {
 int x, y;
 int t;
 Angle(int x, int y, int t=0) : x(x), y(y), t(t) {}
 Angle operator-(Angle b) const { return {x-b.x, y-b.y, t};
     }
 int half() const {
  assert(x || y);
  return y < 0 || (y == 0 && x < 0);
 }
 Angle t90() const { return {-y, x, t + (half() && x >= 0)};
     }
 Angle t180() const { return {-x, -y, t + half()}; }
 Angle t360() const { return {x, y, t + 1}; }
};
bool operator<(Angle a, Angle b) {
 // add a.dist2() and b.dist2() to also compare distances
 return make_tuple(a.t, a.half(), a.y * (ll)b.x) <
       make_tuple(b.t, b.half(), a.x * (ll)b.y);
}

// Given two points, this calculates the smallest angle
    between
// them, i.e., the angle that covers the defined line
    segment.
pair<Angle, Angle> segmentAngles(Angle a, Angle b) {
 if (b < a) swap(a, b);
 return (b < a.t180() ?
        make_pair(a, b) : make_pair(b, a.t360()));
}
Angle operator+(Angle a, Angle b) { // point a + vector b
 Angle r(a.x + b.x, a.y + b.y, a.t);
 if (a.t180() < r) r.t--;
 return r.t180() < a ? r.t360() : r;
}
Angle angleDiff(Angle a, Angle b) { // angle b - angle a
 int tu = b.t - a.t; a.t = b.t;
 return {a.x*b.x + a.y*b.y, a.x*b.y - a.y*b.x, tu - (b < a)
     };
}
```

5.2 CircleIntersection

```
/**
 * Description: Computes the pair of points at which two
     circles intersect. Returns false in case of no
     intersection.
 * Status: stress-tested
 */
#pragma once

#include "Point.h"

typedef Point<double> P;
bool circleInter(P a,P b,double r1,double r2,pair<P, P>* out
     ) {
 if (a == b) { assert(r1 != r2); return false; }
 P vec = b - a;
 double d2 = vec.dist2(), sum = r1+r2, dif = r1-r2,
        p = (d2 + r1*r1 - r2*r2)/(d2*2), h2 = r1*r1 - p*p*d2;
 if (sum*sum < d2 || dif*dif > d2) return false;
 P mid = a + vec*p, per = vec.perp() * sqrt(fmax(0, h2) / d2
     );
 *out = {mid + per, mid - per};
 return true;
}
```

5.3 CircleLine

```
/**
 * Description: Finds the intersection between a circle and
     a line.
 * Returns a vector of either 0, 1, or 2 intersection points
     .
 * P is intended to be Point<double>.
 */

#pragma once

#include "Point.h"

template<class P>
vector<P> circleLine(P c, double r, P a, P b) {
 P ab = b - a, p = a + ab * (c-a).dot(ab) / ab.dist2();
 double s = a.cross(b, c), h2 = r*r - s*s / ab.dist2();
 if (h2 < 0) return {};
 if (h2 == 0) return {p};
 P h = ab.unit() * sqrt(h2);
 return {p - h, p + h};
}
```

5.4 CirclePolygonIntersection

```
/**
 * Description: Returns the area of the intersection of a
     circle with a
 * ccw polygon.
 * Time: O(n)
 * Status: Tested on GNYR 2019 Gerrymandering, stress-tested
 */
#pragma once

#include "../../content/geometry/Point.h"

typedef Point<double> P;
#define arg(p, q) atan2(p.cross(q), p.dot(q))
double circlePoly(P c, double r, vector<P> ps) {
 auto tri = [&](P p, P q) {
  auto r2 = r * r / 2;
  P d = q - p;
  auto a = d.dot(p)/d.dist2(), b = (p.dist2()-r*r)/d.dist2()
      ;
  auto det = a * a - b;
  if (det <= 0) return arg(p, q) * r2;
  auto s = max(0., -a-sqrt(det)), t = min(1., -a+sqrt(det));
  if (t < 0 || 1 <= s) return arg(p, q) * r2;
  P u = p + d * s, v = p + d * t;
  return arg(p,u) * r2 + u.cross(v)/2 + arg(v,q) * r2;
 };
 auto sum = 0.0;
 rep(i,0,sz(ps))
  sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)] - c);
 return sum;
}
```

5.5 CircleTangents

```
/**
 * Description: Finds the external tangents of two circles,
     or internal if r2 is negated.
 * Can return 0, 1, or 2 tangents -- 0 if one circle
     contains the other (or overlaps it, in the internal
     case, or if the circles are the same);
 * 1 if the circles are tangent to each other (in which case
      .first = .second and the tangent line is perpendicular
      to the line between the centers).
 * .first and .second give the tangency points at circle 1
     and 2 respectively.
 * To find the tangents of a circle with a point set r2 to
     0.
 * Status: tested
 */
#pragma once

#include "Point.h"

template<class P>
vector<pair<P, P>> tangents(P c1, double r1, P c2, double r2
    ) {
 P d = c2 - c1;
 double dr = r1 - r2, d2 = d.dist2(), h2 = d2 - dr * dr;
 if (d2 == 0 || h2 < 0) return {};
 vector<pair<P, P>> out;
 for (double sign : {-1, 1}) {
  P v = (d * dr + d.perp() * sqrt(h2) * sign) / d2;
  out.push_back({c1 + v * r1, c2 + v * r2});
 }
 if (h2 == 0) out.pop_back();
 return out;
}
```

5.6 circumcircle

```
/**
 * Description:\\
 * The circumcirle of a triangle is the circle intersecting
     all three vertices. ccRadius returns the radius of the
     circle going through points A, B and C and ccCenter
     returns the center of the same circle.

 * Status: tested
 */
#pragma once

#include "Point.h"

typedef Point<double> P;
double ccRadius(const P& A, const P& B, const P& C) {
 return (B-A).dist()*(C-B).dist()*(A-C).dist()/
   abs((B-A).cross(C-A))/2;
}
P ccCenter(const P& A, const P& B, const P& C) {
 P b = C-A, c = B-A;
 return A + (b*c.dist2()-c*b.dist2()).perp()/b.cross(c)/2;
}
```

5.7 ClosestPair

```
/**
 * Source: https://codeforces.com/blog/entry/58747
 * Description: Finds the closest pair of points.
 */
#pragma once

#include "Point.h"

typedef Point<ll> P;
pair<P, P> closest(vector<P> v) {
 assert(sz(v) > 1);
 set<P> S;
 sort(all(v), [](P a, P b) { return a.y < b.y; });
 pair<ll, pair<P, P>> ret{LLONG_MAX, {P(), P()}};
 int j = 0;
 for (P p : v) {
  P d{1 + (ll)sqrt(ret.first), 0};
  while (v[j].y <= p.y - d.x) S.erase(v[j++]);
  auto lo = S.lower_bound(p - d), hi = S.upper_bound(p + d);
  for (; lo != hi; ++lo)
   ret = min(ret, {(*lo - p).dist2(), {*lo, p}});
  S.insert(p);
 }
 return ret.second;
}
```

5.8 ConvexHull

```
/**
Returns a vector of the points of the convex hull in counter
     -clockwise order.
Points on the edge of the hull between two other points are
     not considered part of the hull.
*/
#pragma once

#include "Point.h"

typedef Point<ll> P;
vector<P> convexHull(vector<P> pts) {
 if (sz(pts) <= 1) return pts;
 sort(all(pts));
 vector<P> h(sz(pts)+1);
 int s = 0, t = 0;
 for (int it = 2; it--; s = --t, reverse(all(pts)))
  for (P p : pts) {
   while (t >= s + 2 && h[t-2].cross(h[t-1], p) <= 0) t--;
   h[t++] = p;
  }
 return {h.begin(), h.begin() + t - (t == 2 && h[0] == h[1])
     };
}
```

5.9 DelaunayTriangulation

```
/**
 * Description: Computes the Delaunay triangulation of a set
      of points.
 *  Each circumcircle contains none of the input points.
 *  If any three points are collinear or any four are on the
      same circle, behavior is undefined.
 * Time: O(n^2)
 * Status: stress-tested
 */
#pragma once

#include "Point.h"
#include "3dHull.h"

template<class P, class F>
void delaunay(vector<P>& ps, F trifun) {
 if (sz(ps) == 3) { int d = (ps[0].cross(ps[1], ps[2]) < 0);
  trifun(0,1+d,2-d); }
 vector<P3> p3;
 for (P p : ps) p3.emplace_back(p.x, p.y, p.dist2());
 if (sz(ps) > 3) for(auto t:hull3d(p3)) if ((p3[t.b]-p3[t.a
      ]).
   cross(p3[t.c]-p3[t.a]).dot(P3(0,0,1)) < 0)
  trifun(t.a, t.c, t.b);
}
```

5.10 FastDelaunay

```
/**
 * Description: Fast Delaunay triangulation.
 * Each circumcircle contains none of the input points.
 * There must be no duplicate points.
 * If all points are on a line, no triangles will be
     returned.
 * Should work for doubles as well, though there may be
     precision issues in 'circ'.
 * Returns triangles in order \{t[0][0], t[0][1], t[0][2], t
     [1][0], \dots\}, all counter-clockwise.
 * Time: O(n \log n)
 */
#pragma once

#include "Point.h"

typedef Point<ll> P;
typedef struct Quad* Q;
typedef __int128_t lll; // (can be ll if coords are < 2e4)
P arb(LLONG_MAX,LLONG_MAX); // not equal to any other point

struct Quad {
 Q rot, o; P p = arb; bool mark;
 P& F() { return r()->p; }
 Q& r() { return rot->rot; }
 Q prev() { return rot->o->rot; }
 Q next() { return r()->prev(); }
} *H;

bool circ(P p, P a, P b, P c) { // is p in the circumcircle?
 lll p2 = p.dist2(), A = a.dist2()-p2,
     B = b.dist2()-p2, C = c.dist2()-p2;
 return p.cross(a,b)*C + p.cross(b,c)*A + p.cross(c,a)*B >
     0;
}
Q makeEdge(P orig, P dest) {
 Q r = H ? H : new Quad{new Quad{new Quad{new Quad{0}}}};
 H = r->o; r->r()->r() = r;
 rep(i,0,4) r = r->rot, r->p = arb, r->o = i & 1 ? r : r->r
     ();
 r->p = orig; r->F() = dest;
 return r;
}
void splice(Q a, Q b) {
 swap(a->o->rot->o, b->o->rot->o); swap(a->o, b->o);
}
Q connect(Q a, Q b) {
 Q q = makeEdge(a->F(), b->p);
 splice(q, a->next());
 splice(q->r(), b);
 return q;
}

pair<Q,Q> rec(const vector<P>& s) {
 if (sz(s) <= 3) {
  Q a = makeEdge(s[0], s[1]), b = makeEdge(s[1], s.back());
  if (sz(s) == 2) return { a, a->r() };
  splice(a->r(), b);
  auto side = s[0].cross(s[1], s[2]);
  Q c = side ? connect(b, a) : 0;
  return {side < 0 ? c->r() : a, side < 0 ? c : b->r() };
 }

#define H(e) e->F(), e->p
#define valid(e) (e->F().cross(H(base)) > 0)
 Q A, B, ra, rb;
 int half = sz(s) / 2;
 tie(ra, A) = rec({all(s) - half});
 tie(B, rb) = rec({sz(s) - half + all(s)});
 while ((B->p.cross(H(A)) < 0 && (A = A->next())) ||
        (A->p.cross(H(B)) > 0 && (B = B->r()->o)));
 Q base = connect(B->r(), A);
 if (A->p == ra->p) ra = base->r();
 if (B->p == rb->p) rb = base;

#define DEL(e, init, dir) Q e = init->dir; if (valid(e)) \
  while (circ(e->dir->F(), H(base), e->F())) { \
   Q t = e->dir; \
   splice(e, e->prev()); \
   splice(e->r(), e->r()->prev()); \
   e->o = H; H = e; e = t; \
  }
 for (;;) {
  DEL(LC, base->r(), o); DEL(RC, base, prev());
  if (!valid(LC) && !valid(RC)) break;
  if (!valid(LC) || (valid(RC) && circ(H(RC), H(LC))))
   base = connect(RC, base->r());
  else
   base = connect(base->r(), LC->r());
 }
 return { ra, rb };
}

vector<P> triangulate(vector<P> pts) {
 sort(all(pts)); assert(unique(all(pts)) == pts.end());
 if (sz(pts) < 2) return {};
 Q e = rec(pts).first;
 vector<Q> q = {e};
 int qi = 0;
 while (e->o->F().cross(e->F(), e->p) < 0) e = e->o;
#define ADD { Q c = e; do { c->mark = 1; pts.push_back(c->p)
     ; \
 q.push_back(c->r()); c = c->next(); } while (c != e); }
 ADD; pts.clear();
 while (qi < sz(q)) if (!(e = q[qi++])->mark) ADD;
 return pts;
}
```

5.11 HullDiameter

```
/**
 * Description: Returns the two points with max distance on
     a convex hull (ccw,
 * no duplicate/collinear points).
 */
#pragma once
#include "Point.h"

typedef Point<ll> P;
array<P, 2> hullDiameter(vector<P> S) {
 int n = sz(S), j = n < 2 ? 0 : 1;
 pair<ll, array<P, 2>> res({0, {S[0], S[0]}});
 rep(i,0,j)
  for (;; j = (j + 1) % n) {
   res = max(res, {(S[i] - S[j]).dist2(), {S[i], S[j]}});
   if ((S[(j + 1) % n] - S[j]).cross(S[i + 1] - S[i]) >= 0)
    break;
  }
 return res.second;
}
```

5.12 InsidePolygon

```
/**
 * Description: Returns true if p lies within the polygon.
     If strict is true,
 * it returns false for points on the boundary. The
     algorithm uses
 * products in intermediate steps so watch out for overflow.
 * Time: O(n)
 * Usage:
 * vector<P> v = {P{4,4}, P{1,2}, P{2,1}};
 * bool in = inPolygon(v, P{3, 3}, false);
 */
#pragma once
```

```cpp
#include "Point.h"
#include "OnSegment.h"
#include "SegmentDistance.h"

template<class P>
bool inPolygon(vector<P> &p, P a, bool strict = true) {
 int cnt = 0, n = sz(p);
 rep(i,0,n) {
  P q = p[(i + 1) % n];
  if (onSegment(p[i], q, a)) return !strict;
  //or: if (segDist(p[i], q, a) <= eps) return !strict;
  cnt ^= ((a.y<p[i].y) - (a.y<q.y)) * a.cross(p[i], q) > 0;
 }
 return cnt;
}
```

5.13 linearTransformation

```cpp
/**
 Apply the linear transformation (translation, rotation and
     scaling) which takes line p0-p1 to line q0-q1 to point
     r.
 */
#pragma once

#include "Point.h"

typedef Point<double> P;
P linearTransformation(const P& p0, const P& p1,
  const P& q0, const P& q1, const P& r) {
 P dp = p1-p0, dq = q1-q0, num(dp.cross(dq), dp.dot(dq));
 return q0 + P((r-p0).cross(num), (r-p0).dot(num))/dp.dist2
     ();
}
```

5.14 lineDistance

```cpp
/**
 * Returns the signed distance between point p and the line
     containing points a and b. Positive value on left side
     and negative on right as seen from a towards b. a==b
     gives nan. P is supposed to be Point<T> or Point3D<T>
     where T is e.g. double or long long. It uses products
     in intermediate steps so watch out for overflow if
     using int or long long. Using Point3D will always give
     a non-negative distance. For Point3D, call .dist on the
      result of the cross product.
 * Status: tested
 */
#pragma once

#include "Point.h"

template<class P>
double lineDist(const P& a, const P& b, const P& p) {
 return (double)(b-a).cross(p-a)/(b-a).dist();
}
```

5.15 lineIntersection

```cpp
/**
If a unique intersection point of the lines going through s1
     ,e1 and s2,e2 exists \{1, point\} is returned.
If no intersection point exists \{0, (0,0)\} is returned and
      if infinitely many exists \{-1, (0,0)\} is returned.
The wrong position will be returned if P is Point<ll> and
     the intersection point does not have integer
     coordinates.
Products of three coordinates are used in intermediate steps
      so watch out for overflow if using int or ll.
 * Usage:
 *  auto res = lineInter(s1,e1,s2,e2);
 *  if (res.first == 1)
 *   cout << "intersection point at " << res.second << endl;
 * Status: stress-tested, and tested through half-plane
      tests
 */
#pragma once

#include "Point.h"

template<class P>
pair<int, P> lineInter(P s1, P e1, P s2, P e2) {
 auto d = (e1 - s1).cross(e2 - s2);
 if (d == 0) // if parallel
  return {-(s1.cross(e1, s2) == 0), P(0, 0)};
 auto p = s2.cross(e1, e2), q = s2.cross(e2, s1);
 return {1, (s1 * p + e1 * q) / d};
}
```

5.16 LineProjectionReflection

```cpp
/**
 * Description: Projects point p onto line ab. Set refl=true
      to get reflection
 * of point p across line ab insted. The wrong point will be
      returned if P is
 * an integer point and the desired point doesn't have
     integer coordinates.
 * Products of three coordinates are used in intermediate
     steps so watch out
 * for overflow.
 */
#pragma once

#include "Point.h"

template<class P>
P lineProj(P a, P b, P p, bool refl=false) {
 P v = b - a;
 return p - v.perp()*(1+refl)*v.cross(p-a)/v.dist2();
}
```

5.17 ManhattanMST

```cpp
/**
 * Description: Given N points, returns up to 4*N edges,
     which are guaranteed
 * to contain a minimum spanning tree for the graph with
     edge weights w(p, q) =
 * |p.x - q.x| + |p.y - q.y|. Edges are in the form (
     distance, src, dst). Use a
 * standard MST algorithm on the result to find the final
     MST.
 * Time: O(N \log N)
 */
#pragma once
#include "Point.h"

typedef Point<int> P;
vector<array<int, 3>> manhattanMST(vector<P> ps) {
 vi id(sz(ps));
 iota(all(id), 0);
 vector<array<int, 3>> edges;
 rep(k,0,4) {
  sort(all(id), [&](int i, int j) {
      return (ps[i]-ps[j]).x < (ps[j]-ps[i]).y;});
  map<int, int> sweep;
  for (int i : id) {
   for (auto it = sweep.lower_bound(-ps[i].y);
```

```
        it != sweep.end(); sweep.erase(it++)) {
    int j = it->second;
    P d = ps[i] - ps[j];
    if (d.y > d.x) break;
    edges.push_back({d.y + d.x, i, j});
   }
   sweep[-ps[i].y] = i;
  }
  for (P& p : ps) if (k & 1) p.x = -p.x; else swap(p.x, p.y)
      ;
 }
 return edges;
}
```

5.18 MinimumEnclosingCircle

```
/**
 * Description: Computes the minimum circle that encloses a
     set of points.
 * Time: expected O(n)
 * Status: stress-tested
 */
#pragma once

#include "circumcircle.h"

pair<P, double> mec(vector<P> ps) {
 shuffle(all(ps), mt19937(time(0)));
 P o = ps[0];
 double r = 0, EPS = 1 + 1e-8;
 rep(i,0,sz(ps)) if ((o - ps[i]).dist() > r * EPS) {
  o = ps[i], r = 0;
  rep(j,0,i) if ((o - ps[j]).dist() > r * EPS) {
   o = (ps[i] + ps[j]) / 2;
   r = (o - ps[i]).dist();
   rep(k,0,j) if ((o - ps[k]).dist() > r * EPS) {
    o = ccCenter(ps[i], ps[j], ps[k]);
    r = (o - ps[i]).dist();
   }
  }
 }
 return {o, r};
}
```

5.19 OnSegment

```
/**
 * Description: Returns true iff p lies on the line segment
     from s to e.
 * Use \texttt{(segDist(s,e,p)<=epsilon)} instead when using
      Point<double>.
 * Status:
 */
#pragma once

#include "Point.h"

template<class P> bool onSegment(P s, P e, P p) {
 return p.cross(s, e) == 0 && (s - p).dot(e - p) <= 0;
}
```

5.20 Point

```
/**
 * Description: Class to handle points in the plane.
 *  T can be e.g. double or long long. (Avoid int.)
 * Status: Works fine, used a lot
 */
#pragma once

template <class T> int sgn(T x) { return (x > 0) - (x < 0);
    }
template<class T>
struct Point {
 typedef Point P;
 T x, y;
 explicit Point(T x=0, T y=0) : x(x), y(y) {}
 bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y);
      }
 bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y);
      }
 P operator+(P p) const { return P(x+p.x, y+p.y); }
 P operator-(P p) const { return P(x-p.x, y-p.y); }
 P operator*(T d) const { return P(x*d, y*d); }
 P operator/(T d) const { return P(x/d, y/d); }
 T dot(P p) const { return x*p.x + y*p.y; }
 T cross(P p) const { return x*p.y - y*p.x; }
 T cross(P a, P b) const { return (a-*this).cross(b-*this);
     }
 T dist2() const { return x*x + y*y; }
 double dist() const { return sqrt((double)dist2()); }
 // angle to x-axis in interval [-pi, pi]
 double angle() const { return atan2(y, x); }
 P unit() const { return *this/dist(); } // makes dist()=1
 P perp() const { return P(-y, x); } // rotates +90 degrees
 P normal() const { return perp().unit(); }
 // returns point rotated 'a' radians ccw around the origin
 P rotate(double a) const {
  return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a)); }
 friend ostream& operator<<(ostream& os, P p) {
  return os << "(" << p.x << "," << p.y << ")"; }
};
```

5.21 PointInsideHull

```
/**
 * Description: Determine whether a point t lies inside a
     convex hull (CCW
 * order, with no collinear points). Returns true if point
     lies within
 * the hull. If strict is true, points on the boundary aren'
     t included.
 * Usage:
 * Status: stress-tested
 * Time: O(\log N)
 */
#pragma once

#include "Point.h"
#include "sideOf.h"
#include "OnSegment.h"

typedef Point<ll> P;

bool inHull(const vector<P>& l, P p, bool strict = true) {
 int a = 1, b = sz(l) - 1, r = !strict;
 if (sz(l) < 3) return r && onSegment(l[0], l.back(), p);
 if (sideOf(l[0], l[a], l[b]) > 0) swap(a, b);
 if (sideOf(l[0], l[a], p) >= r || sideOf(l[0], l[b], p)<= -
     r)
  return false;
 while (abs(a - b) > 1) {
  int c = (a + b) / 2;
  (sideOf(l[0], l[c], p) > 0 ? b : a) = c;
 }
 return sgn(l[a].cross(l[b], p)) < r;
}
```

5.22 PolygonArea

```
/**
 * Description: Returns twice the signed area of a polygon.
```

```
 *  Clockwise enumeration gives negative area. Watch out for
     overflow if using int as T!
 * Status: Stress-tested and tested on kattis:polygonarea
 */
#pragma once

#include "Point.h"

template<class T>
T polygonArea2(vector<Point<T>>& v) {
 T a = v.back().cross(v[0]);
 rep(i,0,sz(v)-1) a += v[i].cross(v[i+1]);
 return a;
}
```

5.23 PolygonCenter

```
/**
 * Description: Returns the center of mass for a polygon.
 * Time: O(n)
 * Status: Tested
 */
#pragma once

#include "Point.h"

typedef Point<double> P;
P polygonCenter(const vector<P>& v) {
 P res(0, 0); double A = 0;
 for (int i = 0, j = sz(v) - 1; i < sz(v); j = i++) {
  res = res + (v[i] + v[j]) * v[j].cross(v[i]);
  A += v[j].cross(v[i]);
 }
 return res / A / 3;
}
```

5.24 PolygonCut

```
/**
 Returns a vector with the vertices of a polygon with
     everything to the left of the line going from s to e
     cut away.
 * Usage:
 *  vector<P> p = ...;
 *  p = polygonCut(p, P(0,0), P(1,0));
 */
#pragma once

#include "Point.h"
#include "lineIntersection.h"

typedef Point<double> P;
vector<P> polygonCut(const vector<P>& poly, P s, P e) {
 vector<P> res;
 rep(i,0,sz(poly)) {
  P cur = poly[i], prev = i ? poly[i-1] : poly.back();
  bool side = s.cross(e, cur) < 0;
  if (side != (s.cross(e, prev) < 0))
   res.push_back(lineInter(s, e, cur, prev).second);
  if (side)
   res.push_back(cur);
 }
 return res;
}
```

5.25 PolygonUnion

```
/**
 * Description: Calculates the area of the union of $n$
     polygons (not necessarily
 * convex). The points within each polygon must be given in
     CCW order.
 * (Epsilon checks may optionally be added to sideOf/sgn,
     but shouldn't be needed.)
 * Time: $O(N^2)$, where $N$ is the total number of points
 * Status: stress-tested, Submitted on ECNA 2017 Problem A
 */
#pragma once

#include "Point.h"
#include "sideOf.h"

typedef Point<double> P;
double rat(P a, P b) { return sgn(b.x) ? a.x/b.x : a.y/b.y;
     }
double polyUnion(vector<vector<P>>& poly) {
 double ret = 0;
 rep(i,0,sz(poly)) rep(v,0,sz(poly[i])) {
  P A = poly[i][v], B = poly[i][(v + 1) % sz(poly[i])];
  vector<pair<double, int>> segs = {{0, 0}, {1, 0}};
  rep(j,0,sz(poly)) if (i != j) {
   rep(u,0,sz(poly[j])) {
    P C = poly[j][u], D = poly[j][(u + 1) % sz(poly[j])];
    int sc = sideOf(A, B, C), sd = sideOf(A, B, D);
    if (sc != sd) {
     double sa = C.cross(D, A), sb = C.cross(D, B);
     if (min(sc, sd) < 0)
      segs.emplace_back(sa / (sa - sb), sgn(sc - sd));
    } else if (!sc && !sd && j<i && sgn((B-A).dot(D-C))>0){
     segs.emplace_back(rat(C - A, B - A), 1);
     segs.emplace_back(rat(D - A, B - A), -1);
    }
   }
  }
  sort(all(segs));
  for (auto& s : segs) s.first = min(max(s.first, 0.0), 1.0)
      ;
  double sum = 0;
  int cnt = segs[0].second;
  rep(j,1,sz(segs)) {
   if (!cnt) sum += segs[j].first - segs[j - 1].first;
   cnt += segs[j].second;
  }
  ret += A.cross(B) * sum;
 }
 return ret / 2;
}
```

5.26 SegmentDistance

```
/**
Returns the shortest distance between point p and the line
     segment from point s to e.
 * Usage:
 *  Point<double> a, b(2,2), p(1,1);
 *  bool onSegment = segDist(a,b,p) < 1e-10;
 * Status: tested
 */
#pragma once

#include "Point.h"

typedef Point<double> P;
double segDist(P& s, P& e, P& p) {
 if (s==e) return (p-s).dist();
 auto d = (e-s).dist2(), t = min(d,max(.0,(p-s).dot(e-s)));
 return ((p-s)*d-(e-s)*t).dist()/d;
}
```

5.27 SegmentIntersection

```
/**
```

```
If a unique intersection point between the line segments
    going from s1 to e1 and from s2 to e2 exists then it is
     returned.
If no intersection point exists an empty vector is returned.
     If infinitely many exist a vector with 2 elements is
    returned, containing the endpoints of the common line
    segment.
The wrong position will be returned if P is Point<ll> and
    the intersection point does not have integer
    coordinates.
Products of three coordinates are used in intermediate steps
     so watch out for overflow if using int or long long.
 * Usage:
 * vector<P> inter = segInter(s1,e1,s2,e2);
 * if (sz(inter)==1)
 *   cout << "segments intersect at " << inter[0] << endl;
 * Status: stress-tested, tested on kattis:intersection
 */
#pragma once

#include "Point.h"
#include "OnSegment.h"

template<class P> vector<P> segInter(P a, P b, P c, P d) {
 auto oa = c.cross(d, a), ob = c.cross(d, b),
      oc = a.cross(b, c), od = a.cross(b, d);
 // Checks if intersection is single non-endpoint point.
 if (sgn(oa) * sgn(ob) < 0 && sgn(oc) * sgn(od) < 0)
  return {(a * ob - b * oa) / (ob - oa)};
 set<P> s;
 if (onSegment(c, d, a)) s.insert(a);
 if (onSegment(c, d, b)) s.insert(b);
 if (onSegment(a, b, c)) s.insert(c);
 if (onSegment(a, b, d)) s.insert(d);
 return {all(s)};
}
```

5.28 sideOf

```
/**
 * Description: Returns where $p$ is as seen from $s$
     towards $e$. 1/0/-1 $\Leftrightarrow$ left/on line/
     right. If the optional argument $eps$ is given 0 is
     returned if $p$ is within distance $eps$ from the line.
      P is supposed to be Point<T> where T is e.g. double or
      long long. It uses products in intermediate steps so
     watch out for overflow if using int or long long.
 * Usage:
 *  bool left = sideOf(p1,p2,q)==1;
 * Status: tested
 */
#pragma once

#include "Point.h"

template<class P>
int sideOf(P s, P e, P p) { return sgn(s.cross(e, p)); }

template<class P>
int sideOf(const P& s, const P& e, const P& p, double eps) {
 auto a = (e-s).cross(p-s);
 double l = (e-s).dist()*eps;
 return (a > l) - (a < -l);
}
```

6 Graph

6.1 2SAT

```
//task : n people, each people have 2 request : + x or - x
//Ask : Is there a way build array m elements that for each
     people,
//at least one of two request is satisfied. If yes, print it
     .

void tarjan(int u) {
    st[++top] = u;
    in[u] = low[u] = ++tme;
    was_tarjan[u] = true;
    for (int v : g[u]) {
        if (!was_tarjan[v]) {
            tarjan(v);
            low[u] = min(low[u], low[v]);
        }
        else {
            low[u] = min(low[u], in[v]);
        }
    }

    if (low[u] == in[u]) {
        ++scc_cnt;
        while (st[top] != u) {
            scc_id[st[top]] = scc_cnt;
            in[st[top]] = low[st[top]] = N;
            --top;
        }
        scc_id[st[top--]] = scc_cnt;
        in[u] = low[u] = N;
    }
}

void compress_scc_to_dag() {
    for (int u = 1; u <= m; ++u)
        for (int v : g[u]) if (scc_id[v] != scc_id[u])
            g2[scc_id[u]].emplace_back(scc_id[v]);
}

void dfs(int u) { // toposort
    was[u] = true;
    for (int v : g2[u]) if (!was[v]) dfs(v);

    topo[u] = top--;
}

void toposort_g2() {
    top = scc_cnt;
    fill(was + 1, was + scc_cnt + 1, false);
    for (int i = 1; i <= scc_cnt; ++i) if (!was[i]) dfs(i);
}

int main() {
 //io()
 m = n << 1;
    for (int i = 1; i <= m; ++i) if (!was_tarjan[i]) tarjan(i
        );

    for (int i = 1; i <= n; ++i) if (scc_id[i] == scc_id[i +
        n]) {
        cout << "IMPOSSIBLE";
        return 0;
    }

    compress_scc_to_dag();
    toposort_g2();

    for (int i = 1; i <= n; ++i) {
        cout << (topo[scc_id[i]] > topo[scc_id[i + n]] ? '+'
             : '-') << ' ';
    }
}
```

6.2 Blocc Cut Tree

```
struct block_cut_tree {
    int h[maxn], p[20][maxn], in[maxn], low[maxn], cnt = 0,
        node;
```

```
vector<int> adj2[maxn], st;
void DFS(int u = 1, int p = 0)
{
    in[u] = low[u] = ++cnt;
    st.emplace_back(u);
    for (int v: adj[u])
    {
        if (v == p) continue;
        if (in[v]) low[u] = min(low[u], in[v]);
        else
        {
            DFS(v, u);
            low[u] = min(low[u], low[v]);
            if (low[v] >= in[u])
            {
                node++;
                int x;
                do
                    x = st.back(), st.pop_back(),
                    adj2[node].emplace_back(x),
                    adj2[x].emplace_back(node);
                while (x != v);
                adj2[node].emplace_back(u);
                adj2[u].emplace_back(node);
            }
        }
    }
}
void DFS2(int u = 1, int pa = 0)
{
    for (int v: adj2[u])
    {
        if (v == pa) continue;
        h[v] = h[u] + 1;
        p[0][v] = u;
        DFS2(v, u);
    }
}
int lca(int u, int v)
{
    if (h[u] > h[v]) swap(u, v);
    int dis = h[v] - h[u];
    for (int i=19; i>=0; i--) if ((dis>>i)&1) v = p[i][v
        ];
    if (v == u) return u;
    for (int i=19; i>=0; i--) if (p[i][u] != p[i][v]) u =
          p[i][u], v = p[i][v];
    return p[0][u];
}
    int dis(int u, int v) {return h[u] + h[v] - 2 * h[lca(u,
        v)];}
    int query(int u, int v)
    {
        return dis(u, v)/2 + 1;
    }
    void init()
    {
        memset(p, -1, sizeof p);
        memset(in, 0, sizeof in);
        memset(low, 0, sizeof low);
        memset(h, 0, sizeof h);
        st.clear();
        for (int i=1; i<=n; i++) adj2[i].clear();
        node = n;
        DFS();
        DFS2();
        for (int i=1; (1<<i) <= node; i++)
            for (int j=1; j<=node; j++)
                if (p[i-1][j] != -1) p[i][j] = p[i-1][p[i-1][j
                    ]];
    }

} bctree;
```

6.3 DeMen Bipartite Matching

```
vector <int> a[N];
int mr[N], cttme;
bool ml[N];
char f[N];

bool dfs(int u){
    if (f[u] == cttme) return false;
    f[u] = cttme;
    for(int i : a[u]){
        if (!mr[i] || dfs(mr[i])){
            mr[i] = u;
            return true;
        }
    }
    return false;
}

int maximum_matching(){
    int cnt = 0;
    for(bool run = true; run;){
        cttme++;
        run = false;
        for(int i = 1; i <= n; ++i){
            if (ml[i]) continue;
            if (dfs(i)){
                ml[i] = run = true;
                ++cnt;
            }
        }
    }
    return cnt;
}
```

6.4 General Matching

```
struct GeneralMatching {
    GeneralMatching(int _n) : n(_n), match(_n, -1), g(_n),
          timer(-1), label(_n), parent(_n), orig(_n), aux(
              _n, -1) {}

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

    int get_match() {
        for (int i = 0; i < n; i++) {
            if (match[i] == -1) bfs(i);
        }
        int res = 0;
        for (int i = 0; i < n; i++) {
            if (match[i] >= 0) ++res;
        }
        return res / 2;
    }

    int n;
    vector<int> match;

private:
    int lca(int x, int y) {
        for (timer++; ; swap(x, y)) {
            if (x == -1) continue;
            if (aux[x] == timer) return x;
            aux[x] = timer;
            x = (match[x] == -1 ? -1 : orig[parent[match[x
                ]]]);
        }
    }

    void blossom(int v, int w, int a) {
```

```cpp
        while (orig[v] != a) {
            parent[v] = w;
            w = match[v];
            if (label[w] == 1) {
                label[w] = 0;
                q.push_back(w);
            }
            orig[v] = orig[w] = a;
            v = parent[w];
        }
    }

    void augment(int v) {
        while (v != -1) {
            int pv = parent[v], nv = match[pv];
            match[v] = pv; match[pv] = v; v = nv;
        }
    }

    int bfs(int root) {
        fill(label.begin(), label.end(), -1);
        iota(orig.begin(), orig.end(), 0);
        q.clear();
        label[root] = 0;
        q.push_back(root);
        for (int i = 0; i < (int) q.size(); ++i) {
            int v = q[i];
            for (auto x : g[v]) {
                if (label[x] == -1) {
                    label[x] = 1;
                    parent[x] = v;
                    if (match[x] == -1) {
                        augment(x);
                        return 1;
                    }
                    label[match[x]] = 0;
                    q.push_back(match[x]);
                } else if (label[x] == 0 && orig[v] != orig[x
                    ]) {
                    int a = lca(orig[v], orig[x]);
                    blossom(x, v, a);
                    blossom(v, x, a);
                }
            }
        }
        return 0;
    }

private:
    vector<vector<int>> g;
    int timer;
    vector<int> label, parent, orig, aux, q;
};
```

6.5 Gomory Hu Tree

```cpp
//0-based

struct PushRelabel {
 struct Edge {
  int dest, back;
  int f, c;
 };
 vector<vector<Edge>> g;
 vector<int> ec;
 vector<Edge*> cur;
 vector<vector<int>> hs; vector<int> H;
 PushRelabel(int n) : g(n), ec(n), cur(n), hs(2*n), H(n) {}

 void addEdge(int s, int t, int cap, int rcap=0) {
  if (s == t) return;
  g[s].push_back({t, (int)g[t].size(), 0, cap});
  g[t].push_back({s, (int)g[s].size()-1, 0, rcap});
 }

 void addFlow(Edge& e, int f) {
  Edge &back = g[e.dest][e.back];
  if (!ec[e.dest] && f) hs[H[e.dest]].push_back(e.dest);
  e.f += f; e.c -= f; ec[e.dest] += f;
  back.f -= f; back.c += f; ec[back.dest] -= f;
 }
 int calc(int s, int t) {
  int v = g.size(); H[s] = v; ec[t] = 1;
  vector<int> co(2*v); co[0] = v-1;
  for(int i = 0; i < v; ++i) cur[i] = g[i].data();
  for (Edge& e : g[s]) addFlow(e, e.c);

  for (int hi = 0;;) {
   while (hs[hi].empty()) if (!hi--) return -ec[s];
   int u = hs[hi].back(); hs[hi].pop_back();
   while (ec[u] > 0) // discharge u
    if (cur[u] == g[u].data() + g[u].size()) {
     H[u] = 1e9;
     for (Edge& e : g[u]) if (e.c && H[u] > H[e.dest]+1)
      H[u] = H[e.dest]+1, cur[u] = &e;
     if (++co[H[u]], !--co[hi] && hi < v)
      for(int i = 0; i < v; ++i) if (hi < H[i] && H[i] < v)
       --co[H[i]], H[i] = v + 1;
     hi = H[u];
    } else if (cur[u]->c && H[u] == H[cur[u]->dest]+1)
     addFlow(*cur[u], min(ec[u], cur[u]->c));
    else ++cur[u];
  }
 }
 bool leftOfMinCut(int a) { return H[a] >= (int)g.size(); }
};

vector<array<int, 3>> gomoryHu(int N, vector<array<int, 3>>
     ed) {
 vector<array<int, 3>> tree;
 vector<int> par(N);
 for(int i = 0; i < N; ++i) {
  PushRelabel D(N); // Dinic also works
  for (array<int, 3> t : ed) D.addEdge(t[0], t[1], t[2], t
      [2]);

      int flow = D.calc(i, par[i]);
  if (flow != -1) tree.push_back({i, par[i], flow});
  for(int j = i + 1; j < N; ++j)
   if (par[j] == par[i] && D.leftOfMinCut(j)) par[j] = i;
 }
 return tree;
}
```

6.6 Max Cost General Matching

```cpp
#include <bits/stdc++.h>
using namespace std;
// Cap ghep co trong so lon nhat - Thay Hoang
#define MCM MaxCostMatching
namespace MaxCostMatching {
#define dist(e) (lab[e.u] + lab[e.v] - g[e.u][e.v].w * 2)
    const int maxn = 1e3 + 5;
    const int oo = (int) 1e9;
    struct Edge {
        int u, v, w;
    } g[maxn][maxn];

    int n, m, n_x;
    int lab[maxn], match[maxn], slack[maxn], st[maxn], pa[
        maxn];
    int flower_from[maxn][maxn], s[maxn], vis[maxn];
    vector<int> flower[maxn];
    deque<int> q;

    void init(int _n) {
        n = _n;
        for (int u = 1; u <= n; u++) {
```

```cpp
            for (int v = 1; v <= n; v++) {
                g[u][v] = Edge{u, v, 0};
            }
        }
    }
    void add(int u, int v, int w) {
        g[u][v].w = max(g[u][v].w, w);
        g[v][u].w = max(g[v][u].w, w);
    }
    void update_slack(int u,int x) {
        if (!slack[x] || dist(g[u][x]) < dist(g[slack[x]][x])
            ) slack[x] = u;
    }
    void set_slack(int x) {
        slack[x] = 0;
        for (int u = 1; u <= n; u++) {
            if (g[u][x].w > 0 && st[u] != x && s[st[u]] == 0)
                update_slack(u, x);
        }
    }
    void q_push(int x) {
        if (x <= n) return q.push_back(x);
        for (int i = 0; i < flower[x].size(); i++) q_push(
            flower[x][i]);
    }
    void set_st(int x,int b) {
        st[x] = b;
        if (x <= n) return;
        for(int i = 0; i < flower[x].size(); i++) set_st(
            flower[x][i], b);
    }
    int get_pr(int b,int xr) {
        int pr = find(flower[b].begin(), flower[b].end(), xr)
             - flower[b].begin();
        if (pr % 2 == 1) {
            reverse(flower[b].begin() + 1, flower[b].end());
            return (int) flower[b].size() - pr;
        }
        else {
            return pr;
        }
    }
    void set_match(int u, int v) {
        match[u] = g[u][v].v;
        if (u <= n) return;
        Edge e = g[u][v];
        int xr = flower_from[u][e.u], pr = get_pr(u, xr);
        for (int i = 0; i < pr; i++) set_match(flower[u][i],
            flower[u][i ^ 1]);
        set_match(xr, v);
        rotate(flower[u].begin(), flower[u].begin() + pr,
            flower[u].end());
    }
    void augment(int u, int v) {
        int xnv = st[match[u]];
        set_match(u, v);
        if (!xnv) return;
        set_match(xnv, st[pa[xnv]]);
        augment(st[pa[xnv]], xnv);
    }
    int get_lca(int u, int v) {
        static int t = 0;
        for (t++; u || v; swap(u, v)) {
            if (u == 0) continue;
            if(vis[u] == t) return u;
            vis[u] = t;
            u = st[match[u]];
            if (u) u = st[pa[u]];
        }
        return 0;
    }
    void add_blossom(int u, int lca, int v) {
        int b = n + 1;
        while (b <= n_x && st[b]) b++;
        if (b > n_x) n_x++;
        lab[b] = 0, s[b] = 0;
        match[b] = match[lca];
        flower[b].clear();
        flower[b].push_back(lca);
        for (int x = u, y; x != lca; x = st[pa[y]]) {
            flower[b].push_back(x), flower[b].push_back(y =
                st[match[x]]), q_push(y);
        }
        reverse(flower[b].begin() + 1, flower[b].end());
        for (int x = v, y; x != lca; x = st[pa[y]]) {
            flower[b].push_back(x), flower[b].push_back(y =
                st[match[x]]), q_push(y);
        }
        set_st(b, b);
        for (int x = 1; x <= n_x; x++) g[b][x].w = g[x][b].w
            = 0;
        for (int x = 1; x <= n; x++) flower_from[b][x] = 0;
        for (int i = 0; i < flower[b].size(); i++) {
            int xs = flower[b][i];
            for (int x = 1; x <= n_x; x++) {
                if (g[b][x].w == 0 || dist(g[xs][x]) < dist(g[
                    b][x])) {
                    g[b][x] = g[xs][x], g[x][b] = g[x][xs];
                }
            }
            for (int x = 1; x <= n; x++) {
                if (flower_from[xs][x]) flower_from[b][x] = xs
                    ;
            }
        }
        set_slack(b);
    }
    void expand_blossom(int b) {
        for (int i = 0; i < flower[b].size(); i++) {
            set_st(flower[b][i], flower[b][i]);
        }
        int xr = flower_from[b][g[b][pa[b]].u], pr=get_pr(b,
             xr);
        for (int i = 0; i < pr; i += 2) {
            int xs = flower[b][i], xns = flower[b][i + 1];
            pa[xs] = g[xns][xs].u;
            s[xs] = 1, s[xns] = 0;
            slack[xs] = 0, set_slack(xns);
            q_push(xns);
        }
        s[xr] = 1, pa[xr] = pa[b];
        for (int i = pr + 1; i < flower[b].size(); i++) {
            int xs = flower[b][i];
            s[xs] = -1, set_slack(xs);
        }
        st[b] = 0;
    }
    int on_found_Edge(const Edge &e) {
        int u = st[e.u], v = st[e.v];
        if (s[v] == -1) {
            pa[v] = e.u, s[v] = 1;
            int nu = st[match[v]];
            slack[v] = slack[nu] = 0;
            s[nu] = 0, q_push(nu);
        }
        else if (s[v] == 0) {
            int lca = get_lca(u, v);
            if (!lca) return augment(u, v), augment(v, u), 1;
            else add_blossom(u, lca, v);
        }
        return 0;
    }
    int matching() {
        fill(s, s + n_x + 1, -1), fill(slack, slack + n_x +
             1, 0);
        q.clear();
        for(int x = 1; x <= n_x; x++) {
            if (st[x] == x && !match[x]) pa[x] = 0, s[x]= 0,
                q_push(x);
        }
```

```
        if (q.empty()) return 0;
        while (1) {
            while (q.size()) {
                int u = q.front();
                q.pop_front();
                if (s[st[u]] == 1) continue;
                for (int v = 1; v <= n; v++) {
                    if (g[u][v].w > 0 && st[u] != st[v]) {
                        if (dist(g[u][v]) == 0) {
                            if (on_found_Edge(g[u][v])) return
                                 1;
                        }
                        else update_slack(u, st[v]);
                    }
                }
            }
            int d = oo;
            for (int b = n + 1; b <= n_x; b++) {
                if (st[b] == b && s[b] == 1) d = min(d, lab[b]
                     / 2);
            }
            for (int x = 1; x <= n_x; x++) {
                if (st[x] == x && slack[x]) {
                    if (s[x] == -1) d = min(d, dist(g[slack[x
                         ]][x]));
                    else if (s[x] == 0) d = min(d, dist(g[
                         slack[x]][x]) / 2);
                }
            }
            for (int u = 1; u <= n; u++) {
                if (s[st[u]] == 0) {
                    if (lab[u] <= d) return 0;
                    lab[u] -= d;
                }
                else if (s[st[u]] == 1) lab[u] += d;
            }
            for (int b = n + 1; b <= n_x; b++) {
                if (st[b] == b) {
                    if (s[st[b]] == 0) lab[b] += d * 2;
                    else if (s[st[b]] == 1) lab[b] -= d * 2;
                }
            }
            q.clear();
            for (int x = 1; x <= n_x; x++) {
                if (st[x] == x && slack[x] && st[slack[x]] !=x
                     && dist(g[slack[x]][x]) == 0) {
                    if (on_found_Edge(g[slack[x]][x])) return
                         1;
                }
            }
            for (int b = n + 1; b <= n_x; b++) {
                if (st[b]==b && s[b] == 1 && lab[b] == 0)
                    expand_blossom(b);
            }
        }
        return 0;
    }
    int maxcost() {
        fill(match, match + n + 1, 0);
        n_x = n;
        int tot_weight = 0;
        int n_matches = 0;
        for (int u = 0; u <= n; u++) st[u] = u, flower[u].
             clear();
        int w_max = 0;
        for (int u = 1; u <= n; u++) {
            for (int v = 1; v <= n; v++) {
                flower_from[u][v] = (u == v ? u : 0);
                w_max = max(w_max, g[u][v].w);
            }
        }
        for (int u = 1; u <= n; u++) lab[u] = w_max;
        while (matching()) n_matches++;
        for (int u = 1; u <= n; u++) {
            if (match[u] && match[u] < u) {
                tot_weight += g[u][match[u]].w;
            }
        }
        return tot_weight;
    }
}

int main() {
    MCM::init(4);
    MCM::add(1, 2, 5);
    MCM::add(2, 3, 10);
    MCM::add(3, 4, 2);
    cout << MCM::maxcost() << "\n";
    return 0;
}
```

6.7 Max Flow

```
struct FlowEdge {
    int v, u;
    long long cap, flow = 0;
    FlowEdge(int v, int u, long long cap) : v(v), u(u), cap(
         cap) {}
};

struct Dinic {
    const long long flow_inf = 1e18;
    vector<FlowEdge> edges;
    vector<vector<int>> adj;
    int n, m = 0;
    int s, t;
    vector<int> level, ptr;
    queue<int> q;

    Dinic(int n, int s, int t) : n(n), s(s), t(t) {
        adj.resize(n);
        level.resize(n);
        ptr.resize(n);
    }

    void add_edge(int v, int u, long long cap) {
        edges.emplace_back(v, u, cap);
        edges.emplace_back(u, v, 0);
        adj[v].push_back(m);
        adj[u].push_back(m + 1);
        m += 2;
    }

    bool bfs() {
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            for (int id : adj[v]) {
                if (edges[id].cap - edges[id].flow < 1)
                    continue;
                if (level[edges[id].u] != -1)
                    continue;
                level[edges[id].u] = level[v] + 1;
                q.push(edges[id].u);
            }
        }
        return level[t] != -1;
    }

    long long dfs(int v, long long pushed) {
        if (pushed == 0)
            return 0;
        if (v == t)
            return pushed;
        for (int& cid = ptr[v]; cid < (int)adj[v].size(); cid
             ++) {
            int id = adj[v][cid];
            int u = edges[id].u;
```

```
        if (level[v] + 1 != level[u] || edges[id].cap -
            edges[id].flow < 1)
            continue;
        long long tr = dfs(u, min(pushed, edges[id].cap -
             edges[id].flow));
        if (tr == 0)
            continue;
        edges[id].flow += tr;
        edges[id ^ 1].flow -= tr;
        return tr;
      }
      return 0;
    }

    long long flow() {
      long long f = 0;
      while (true) {
        fill(level.begin(), level.end(), -1);
        level[s] = 0;
        q.push(s);
        if (!bfs())
            break;
        fill(ptr.begin(), ptr.end(), 0);
        while (long long pushed = dfs(s, flow_inf)) {
            f += pushed;
        }
      }
      return f;
    }
};
```

6.8 Maximum Independent Set

```
void cal(int node, bool col){
    if(col)
    {
        if(visitb[node]) return;
    }
    else
    {
        if(visita[node]) return;
    }
    if(col)
        visitb[node] = 1;
    else
        visita[node] = 1;
    if(col){    // node from the right side, can only
         traverse matched edge
        cal(pb[node], col ^ 1);
        return;
    }
    for(auto nx:adj[node]){
        //cout << nx << ' ' << pa[node] << '\n';
        if(nx==pa[node])continue;
        cal(nx, col ^ 1);
    }
}
int main(){
   DemenMatching();
   for(int i=1;i<=n;i++){
       if(pa[i])continue;     // matched node from the left
            side
       cal(i,0);
   }
   vector<int> MaxISa, MaxISb, MVCa, MVCb; // find max cover
        and minimum cover
   for(int i=1;i<=n;i++){
       if(visita[i]) MaxISa.pb(i); // Minimum indepedent set
            is visted on the left
       else MVCa.pb(i); // Max vertex cover is not visited
            on left
   }
   for(int i=1;i<=k;i++){
       if(!visitb[i])MaxISb.pb(i); // Minimum indepedent set
            is not visted on the right
       else MVCb.pb(i); // Max vertex cover is visited on
            right
   }
}
```

6.9 Prüfer Decode

```
vector<pair<int, int>> pruefer_decode(vector<int> const&
     code) {
    int n = code.size() + 2;
    vector<int> degree(n, 1);
    for (int i : code)
        degree[i]++;

    int ptr = 0;
    while (degree[ptr] != 1)
        ptr++;
    int leaf = ptr;

    vector<pair<int, int>> edges;
    for (int v : code) {
        edges.emplace_back(leaf, v);
        if (--degree[v] == 1 && v < ptr) {
            leaf = v;
        } else {
            ptr++;
            while (degree[ptr] != 1)
                ptr++;
            leaf = ptr;
        }
    }
    edges.emplace_back(leaf, n-1);
    return edges;
}
```

6.10 Prüfer Encode

```
vector<vector<int>> adj;
vector<int> parent;

void dfs(int v) {
    for (int u : adj[v]) {
        if (u != parent[v]) {
            parent[u] = v;
            dfs(u);
        }
    }
}

vector<int> pruefer_code() {
    int n = adj.size();
    parent.resize(n);
    parent[n-1] = -1;
    dfs(n-1);

    int ptr = -1;
    vector<int> degree(n);
    for (int i = 0; i < n; i++) {
        degree[i] = adj[i].size();
        if (degree[i] == 1 && ptr == -1)
            ptr = i;
    }

    vector<int> code(n - 2);
    int leaf = ptr;
    for (int i = 0; i < n - 2; i++) {
        int next = parent[leaf];
        code[i] = next;
        if (--degree[next] == 1 && next < ptr) {
            leaf = next;
        } else {
            ptr++;
```

```
            while (degree[ptr] != 1)
                ptr++;
            leaf = ptr;
        }
    }

    return code;
}
```

7 Mathematics

7.1 CRT

```
//Given m1, m2, r1, r2, find x :
//- x mod m1 = r1
//- x mod m2 = r2

int cal_crt(int r1, int r2, int m1, int m2)){
    ii ans = ExEuclid(m1, m2);
    bool f = false;
    int g = __gcd(m1, m2);
    if ((r2 - r1) % g) return 1e18; //No solution

    int k = ans.x * ((r2 - r1) / g);
    k %= (m2 / g);
    return (r1 + k * m1) % lc;
    //all_ans = {ans + k*LCM(m1, m2)}
}
```

7.2 Extended Euclid

```
ii ExEuclid(int a, int b){
    int x0 = 1, y0 = 0;
    int x1 = 0, y1 = 1;
    int x2, y2;
    while (b){
        int q = a / b;
        int r = a % b;
        a = b; b = r;
        x2 = x0 - q * x1;
        y2 = y0 - q * y1;
        x0 = x1; y0 = y1;
        x1 = x2; y1 = y2;
    }
    return {x0, y0};
}
```

7.3 Fast Fourier Transform - Mod

```
/**
 * Description: Higher precision FFT, can be used for
     convolutions modulo arbitrary integers
 * as long as $N\log_2N\cdot \text{mod} < 8.6 \cdot 10^{14}$
      (in practice $10^{16}$ or higher).
 * Inputs must be in $[0, \text{mod})$.
 * Time: O(N \log N), where $N = |A|+|B|$ (twice as slow as
     NTT or FFT)
 */

typedef vector<ll> vl;
template<int M> vl convMod(const vl &a, const vl &b) {
 if (a.empty() || b.empty()) return {};
 vl res(sz(a) + sz(b) - 1);
 int B=32-__builtin_clz(sz(res)), n=1<<B, cut=int(sqrt(M));
 vector<C> L(n), R(n), outs(n), outl(n);
 rep(i,0,sz(a)) L[i] = C((int)a[i] / cut, (int)a[i] % cut);
 rep(i,0,sz(b)) R[i] = C((int)b[i] / cut, (int)b[i] % cut);
 fft(L), fft(R);
 rep(i,0,n) {
  int j = -i & (n - 1);
  outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
  outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / 1i;
 }
 fft(outl), fft(outs);
 rep(i,0,sz(res)) {
  ll av = ll(real(outl[i])+.5), cv = ll(imag(outs[i])+.5);
  ll bv = ll(imag(outl[i])+.5) + ll(real(outs[i])+.5);
  res[i] = ((av % M * cut + bv) % M * cut + cv) % M;
 }
 return res;
}
```

7.4 Fast Fourier Transform

```
/**
 * Time: O(N \log N) with $N = |A|+|B|$ ($\tilde 1s$ for $N
     =2^{22}$)
 */

typedef complex<double> C;
typedef vector<double> vd;
void fft(vector<C>& a) {
 int n = sz(a), L = 31 - __builtin_clz(n);
 static vector<complex<long double>> R(2, 1);
 static vector<C> rt(2, 1); // (^ 10% faster if double)
 for (static int k = 2; k < n; k *= 2) {
  R.resize(n); rt.resize(n);
  auto x = polar(1.0L, acos(-1.0L) / k);
  rep(i,k,2*k) rt[i] = R[i] = i&1 ? R[i/2] * x : R[i/2];
 }
 vi rev(n);
 rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
 rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
 for (int k = 1; k < n; k *= 2)
  for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
   // C z = rt[j+k] * a[i+j+k]; // (25% faster if hand-
       rolled) /// include-line
   auto x = (double *)&rt[j+k], y = (double *)&a[i+j+k];  ///
       exclude-line
   C z(x[0]*y[0] - x[1]*y[1], x[0]*y[1] + x[1]*y[0]);     ///
       exclude-line
   a[i + j + k] = a[i + j] - z;
   a[i + j] += z;
  }
}
vd conv(const vd& a, const vd& b) {
 if (a.empty() || b.empty()) return {};
 vd res(sz(a) + sz(b) - 1);
 int L = 32 - __builtin_clz(sz(res)), n = 1 << L;
 vector<C> in(n), out(n);
 copy(all(a), begin(in));
 rep(i,0,sz(b)) in[i].imag(b[i]);
 fft(in);
 for (C& x : in) x *= x;
 rep(i,0,n) out[i] = in[-i & (n - 1)] - conj(in[i]);
 fft(out);
 rep(i,0,sz(res)) res[i] = imag(out[i]) / (4 * n);
 return res;
}
```

7.5 Fast Walsh–Hadamard Transform

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

int fpow(int n, long long k, int p = (int) 1e9 + 7) {
    int r = 1;
    for (; k; k >>= 1) {
        if (k & 1) r = (long long) r * n % p;
        n = (long long) n * n % p;
    }
    return r;
}

/*
```

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

7.6 Gaussian Elimination

```
const double EPS = 1e-9;
const int INF = 2; // it doesn't actually have to be
     infinity or a big number

int gauss (vector < vector<double> > a, vector<double> & ans
    ) {
    int n = (int) a.size();
    int m = (int) a[0].size() - 1;

    vector<int> where (m, -1);
    for (int col=0, row=0; col<m && row<n; ++col) {
        int sel = row;
        for (int i=row; i<n; ++i)
            if (abs (a[i][col]) > abs (a[sel][col]))
                sel = i;
        if (abs (a[sel][col]) < EPS)
            continue;
        for (int i=col; i<=m; ++i)
            swap (a[sel][i], a[row][i]);
        where[col] = row;

        for (int i=0; i<n; ++i)
            if (i != row) {
                double c = a[i][col] / a[row][col];
                for (int j=col; j<=m; ++j)
                    a[i][j] -= a[row][j] * c;
            }
        ++row;
    }

    ans.assign (m, 0);
    for (int i=0; i<m; ++i)
        if (where[i] != -1)
            ans[i] = a[where[i]][m] / a[where[i]][i];
    for (int i=0; i<n; ++i) {
        double sum = 0;
        for (int j=0; j<m; ++j)
            sum += ans[j] * a[i][j];
        if (abs (sum - a[i][m]) > EPS)
            return 0;
    }

    for (int i=0; i<m; ++i)
        if (where[i] == -1)
            return INF;
    return 1;
}
```

7.7 Karatsuba

```
void convo(int a[], int b[], int res[], int h1, int h2, int
     n){
 if (n <= 8){
  for (int i = h1; i < h1 + n; i++) {
   for (int j = h1; j < h1 + n; j++) {
    add(res[i + j], prod(a[i], b[j]));
   }
  }
 } else {
  const int mid = n >> 1;
  int atmp[mid], btmp[mid], E[n + 1];
  memset(E, 0, sizeof E);
  for(int i = h1; i < h1 + mid; ++i){
   atmp[i - h1] = sum(a[i], a[i + mid]);
   btmp[i - h1] = sum(b[i], b[i + mid]);
  }

  convo(atmp, btmp, E, 0, 0, mid);
  convo(a, b, res, h1, h2, mid);
  convo(a, b, res, h1 + mid, h2 + n, mid);

  for(int i = h2; i < h2 + mid; ++i){
   const int tmp = res[i + mid];
   add(res[i + mid], E[i - h2] - res[i] - res[i + 2 * mid]);
   add(res[i + 2 * mid], E[i - h2 + mid] - tmp - res[i + 3 *
        mid]);
```

```
  }
 }
}
```

7.8 Lagrange

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

7.9 Pisano

```
bool check(int r, int p)
{
   if (p == 1) return false;
   swap(m, r);
   matrix q; q.build1();
   matrix dv = q;
   q = bpow(q, p + 1);
   for(int i = 0; i < 2; ++i){
      for(int j = 0; j < 2; ++j){
         if (q(i, j) != dv(i, j)) {swap(r, m); return
             false;}
      }
   }
   swap(r, m);
   return true;
}

int pisano(int m)
{
   int p = 1;
   int tmp = m;
   for (int i : prime_factors(m)) {
      int expo = 0, fact = 1;
      for (; tmp % i == 0; ++expo, fact *= i)
         tmp /= i;

      int q = 1;
      if (i == 2) {
         q = (int) fact / 2 * 3;
      }
      else if (i == 5) {
         q = (int) fact * 4;
      }
      else {
         vector<int> cands;
         if (i % 10 == 1 || i % 10 == 9)
            cands = factorize(i - 1);
         else
            cands = factorize(2 * (i + 1));
         for (int x : cands)
            if (check(i, x)) {
               q = x;
               break;
            }
         q = (int) fact / i * q;
      }
      p = lcm(p, q);
   }
   return p;
}
```

7.10 Rho and Miller Rabin

```
mt19937 rng(chrono::steady_clock::now().time_since_epoch().
    count());
vector <int> b;
ll binpower(ll base, ll e, ll mod) {
   ll result = 1;
   base %= mod;
   while (e){
      if (e & 1)result = (ll)result * base % mod;
      base = (ll)base * base % mod;e >>= 1;
   }
   return result;
}
bool check_composite(ll n, ll a, ll d, int s){
   ll x = binpower(a, d, n);
   if (x == 1 or x == n - 1)return false;
   for (int r = 1; r < s; r++) {
      x = (ll)x * x % n;
      if (x == n - 1)return false;
   }return true;
}

bool MillerRabin_checkprime(ll n) {
   if (n < 2)return false;
   int r = 0;
   ll d = n - 1;
   while ((d & 1) == 0) {
      d >>= 1;
      r++;
   }
   for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31,
       37}) {
      if (n == a)return true;
      if (check_composite(n, a, d, r))return false;
   }
   return true;
```

```
}

int mult(int a, int b, int mod) {
    return (ll)a * b % mod;
}

int F(int x, int c, int mod) {
    return (mult(x, x, mod) + c) % mod;
}
int rho(int n, int x0=2, int c=1) {
    int x = x0;
    int y = x0;
    int g = 1;
    if (n % 2 == 0) return 2;
    while (g == 1) {
        x = F(x, c, n);
        y = F(y, c, n);
        y = F(y, c, n);
        g = gcd(abs(x - y), n);
    }
    return g;
}

void Rho_factorization(int n){
    set <int> s;
    if (n == 1) {return;}
    while (!MillerRabin_checkprime(n)){
        ll k;
        while (1){
            int p = (rng()%(n-2))+2, q = (rng()%(n-1))+1;
            k = rho(n,p,q);
            if (MillerRabin_checkprime(k)) break;
        }
        s.insert(k);
        n/=k;
    }
    s.insert(n);
    fora(i,s) b.pb(i);
}
```

7.11 Tonelli Shanks

```
mt19937 rng(std::chrono::system_clock::now().
     time_since_epoch().count());
// sqrt mod p
int Tonelli_Shanks(int n, int p){
    int q = p - 1, s = 0;
    while (q % 2 == 0){
        ++ s,
        q >>= 1;
    }
    int z;
    do {
        z = rng() % p;
    } while(binpow(z, (p-1)/2, p) != p-1);

    int m = s,
        c = binpow(z, q, p),
        t = binpow(n, q, p),
        r = binpow(n, (q+1)/2, p);

    while (t > 1) {
        i = 0;
        do {
            i ++;
            t = 1ll * t * t % p;
        } while (t != 1);

        if (t == m)
            return -1;

        int b = binpow(c, (1<<(m-i-1)), p);
        m = i;
        c = binpow(b, 2, p);
        t = 1ll * t * c % p;
        r = 1ll * r * b;
    }

    if (t == 0)
        return 0;
    else
        return r;
}
```

8 Number Theory

8.1 Bézout's identity

For $a \neq$, $b \neq 0$, then $d = gcd(a, b)$ is the smallest positive integer for which there are integer solutions to

$$ax + by = d$$

If (x, y) is one solution, then all solutions are given by

$$\left(x + \frac{kb}{\gcd(a,b)}, y - \frac{ka}{\gcd(a,b)}\right), \quad k \in Z$$

8.2 Pythagorean Triples

The Pythagorean triples are uniquely generated by

$$a = k \cdot (m^2 - n^2), \quad b = k \cdot (2mn), \quad c = k \cdot (m^2 + n^2),$$

with $m > n > 0$, $k > 0$, $m \perp n$, and either m or n even.

8.3 Primes

$p = 962592769$ is such that $2^{21} \mid p - 1$, which may be useful. For hashing use 970592641 (31-bit number), 31443539979727 (45-bit), 3006703054056749 (52-bit). There are 78498 primes less than 1 000 000.

Primitive roots exist modulo any prime power p^a, except for $p = 2, a > 2$, and there are $\phi(\phi(p^a))$ many. For $p = 2, a > 2$, the group $Z_{2^a}^\times$ is instead isomorphic to $Z_2 \times Z_{2^{a-2}}$.

8.4 Estimates

$\sum_{d|n} d = O(n \log \log n)$.

The number of divisors of n is at most around 100 for $n < 5e4$, 500 for $n < 1e7$, 2000 for $n < 1e10$, 200 000 for $n < 1e19$.

8.5 Mobius Function

$$\mu(n) = \begin{cases} 0 & n \text{ is not square free} \\ 1 & n \text{ has even number of prime factors} \\ -1 & n \text{ has odd number of prime factors} \end{cases}$$

Mobius Inversion:

$$g(n) = \sum_{d|n} f(d) \Leftrightarrow f(n) = \sum_{d|n} \mu(d) g(n/d)$$

Other useful formulas/forms:

$\sum_{d|n} \mu(d) = [n = 1]$ (very useful)

$g(n) = \sum_{n|d} f(d) \Leftrightarrow f(n) = \sum_{n|d} \mu(d/n) g(d)$

$g(n) = \sum_{1 \le m \le n} f(\lfloor \frac{n}{m} \rfloor) \Leftrightarrow f(n) = \sum_{1 \le m \le n} \mu(m) g(\lfloor \frac{n}{m} \rfloor)$

9 String

9.1 Aho Corasick

```
const int K = 26;

struct Vertex {
    int next[K];
    bool leaf = false;
    int p = -1;
    char pch;
    int link = -1;
    int go[K];

    Vertex(int p=-1, char ch='$') : p(p), pch(ch) {
        fill(begin(next), end(next), -1);
        fill(begin(go), end(go), -1);
    }
};

vector<Vertex> t(1);

void add_string(string const& s) {
    int v = 0;
    for (char ch : s) {
        int c = ch - 'a';
        if (t[v].next[c] == -1) {
            t[v].next[c] = t.size();
            t.emplace_back(v, ch);
        }
        v = t[v].next[c];
    }
    t[v].leaf = true;
}

int go(int v, char ch);

int get_link(int v) {
    if (t[v].link == -1) {
        if (v == 0 || t[v].p == 0)
            t[v].link = 0;
        else
            t[v].link = go(get_link(t[v].p), t[v].pch);
    }
    return t[v].link;
}

int go(int v, char ch) {
    int c = ch - 'a';
    if (t[v].go[c] == -1) {
        if (t[v].next[c] != -1)
            t[v].go[c] = t[v].next[c];
        else
            t[v].go[c] = v == 0 ? 0 : go(get_link(v), ch);
    }
    return t[v].go[c];
}
```

9.2 Suffix Array

```
string s;
vector <int> p(400007), c(400007), lcp(400007);
int n,k;

void build(int n){
    vector<pair<int,int>> a(n);
    for(int i = 0; i < n; ++i) a[i] = {s[i], i};
    sort(a.begin(), a.end());
    for(int i = 0; i < n; ++i) p[i] = a[i].y;
    c[p[0]] = 0;
    for(int i = 1; i < n; ++i){
        c[p[i]] = c[p[i-1]];
        if (a[i].x != a[i - 1].x) c[p[i]] += 1;
    }
    k = 0;
    while ((1 << k) < n){
        vector<pair<pair<int,int>,int>> a(n);
        for(int i = 0; i < n; ++i){
            a[i] = {{c[i], c[(i + (1 << k)) % n]}, i};
        }
        //Radix sort
        vector <int> cnt(n);
        for(auto i : a){
            cnt[i.x.y]++;
        }
        vector <pair<pair<int,int>,int>> b(n);
        vector <int> pos(n);
        pos[0] = 0;
        for(int i = 1; i < n; ++i) pos[i] = pos[i-1] + cnt[i
            -1];
        for(auto i : a){
            b[pos[i.x.y]] = i;
            pos[i.x.y]++;
        }
        a=b;
        //////////////////////////////////
        vector <int> cnt2(n);
        for(auto i : a){
            cnt2[i.x.x]++;
        }
        vector <pair<pair<int,int>,int>> f(n);
        vector <int> pos2(n);
        pos2[0] = 0;
        for(int i = 1; i < n; ++i) pos2[i] = pos2[i-1] + cnt2
            [i-1];
        for(auto i : a){
            f[pos2[i.x.x]] = i;
            pos2[i.x.x]++;
        }
        a = f;
        /////////////
        for(int i = 0; i < n; ++i) p[i] = a[i].y;
        c[p[0]] = 0;
        for(int i = 1; i < n; ++i){
            c[p[i]] = c[p[i-1]];
            if (a[i].x != a[i-1].x) c[p[i]]++;
        }
        k++;
    }
}

void buildlcp(int n){
    k=0;
    for(i = 0; i < n - 1; ++i){
        k=max(0,k - 1);
        lcp[c[i]] = k;
        int s1 = i, s2 = p[c[i]-1];
        for(int j = k; j <= n - i + 1; ++j){
            if (s[s1 + j] == s[s2 + j]){
                k++;
                lcp[c[i]] = k;
            } else break;
        }
    }
}
```