

## Contents

<b>1 Reminder</b>	<b>1</b>	<b>6 String</b>	<b>13</b>
1.1 Bug List . . . . .	1	6.1 Aho Corasick . . . . .	13
1.2 OwO . . . . .	1	6.2 KMP . . . . .	13
<b>2 Basic</b>	<b>1</b>	6.3 Z Value . . . . .	13
2.1 Vimrc . . . . .	1	6.4 Manacher . . . . .	13
2.2 Stress . . . . .	1	6.5 Suffix Array . . . . .	14
2.3 Run Sample . . . . .	1	6.6 Suffix Automaton . . . . .	14
2.4 Runcpp.sh . . . . .	1	6.7 Minimum Rotation . . . . .	15
2.5 Others . . . . .	1	6.8 Lyndon Factorization . . . . .	15
<b>3 Data Structure</b>	<b>2</b>	6.9 Rolling Hash . . . . .	15
3.1 BIT . . . . .	2	6.10 Trie . . . . .	15
3.2 Lazy Propagation Segment Tree . . . . .	2	<b>7 Geometry</b>	<b>15</b>
3.3 Treap . . . . .	2	7.1 Basic Operations . . . . .	15
3.4 Persistent Treap . . . . .	3	7.2 Sort by Angle . . . . .	16
3.5 Li Chao Tree . . . . .	3	7.3 Intersection . . . . .	16
3.6 LineContainer . . . . .	4	7.4 Polygon Area . . . . .	16
3.7 Sparse Table . . . . .	4	7.5 Convex Hull . . . . .	16
3.8 Time Segment Tree . . . . .	4	7.6 Point In Convex . . . . .	16
3.9 Dynamic Median . . . . .	4	7.7 Point Segment Distance . . . . .	17
3.10 SOS DP . . . . .	5	7.8 Point in Polygon . . . . .	17
<b>4 Flow / Matching</b>	<b>5</b>	7.9 Minimum Euclidean Distance . . . . .	17
4.1 Dinic . . . . .	5	7.10 Minkowski Sum . . . . .	17
4.2 MCMF . . . . .	5	7.11 Lower Concave Hull . . . . .	17
4.3 KM . . . . .	6	7.12 Pick's Theorem . . . . .	17
4.4 Hopcroft-Karp . . . . .	6	7.13 Rotating SweepLine . . . . .	17
4.5 Blossom . . . . .	7	7.14 Half Plane Intersection . . . . .	18
4.6 Cover / Independent Set . . . . .	7	7.15 Minimum Enclosing Circle . . . . .	18
4.7 Hungarian Algorithm . . . . .	7	7.16 Union of Circles . . . . .	18
<b>5 Graph</b>	<b>7</b>	7.17 Area Of Circle Polygon . . . . .	18
5.1 Heavy-Light Decomposition . . . . .	7	7.18 3D Point . . . . .	19
5.2 Centroid Decomposition . . . . .	8	<b>8 Number Theory</b>	<b>19</b>
5.3 Bellman-Ford + SPFA . . . . .	8	8.1 FFT . . . . .	19
5.4 BCC - AP . . . . .	9	8.2 Pollard's rho . . . . .	20
5.5 BCC - Bridge . . . . .	9	8.3 Miller Rabin . . . . .	20
5.6 SCC - Tarjan . . . . .	10	8.4 Fast Power . . . . .	20
5.7 SCC - Kosaraju . . . . .	10	8.5 Extend GCD . . . . .	20
5.8 Eulerian Path - Undir . . . . .	11	8.6 Mu + Phi . . . . .	20
5.9 Eulerian Path - Dir . . . . .	11	8.7 Discrete Log . . . . .	20
5.10 Hamilton Path . . . . .	11	8.8 sqrt mod . . . . .	21
5.11 Kth Shortest Path . . . . .	11	8.9 Primitive Root . . . . .	21
5.12 System of Difference Constraints . . . . .	13	8.10 LinearSieve . . . . .	21

## 2 Basic

### 2.1 Vimrc

```

set number relativenumber ai t_Co=256 tabstop=4
set mouse=a shiftwidth=4 encoding=utf8
set bs=2 ruler laststatus=2 cmdheight=2
set clipboard=unnamedplus showcmd autoread
set belloff=all
filetype indent on

inoremap ( ()<Esc>i
inoremap " ""<Esc>i
inoremap [ []<Esc>i
inoremap ' ''<Esc>i
inoremap { {<CR>}<Esc>k0

nnoremap <tab> gt
nnoremap <S-tab> gT
inoremap <C-n> <Esc>:tabnew<CR>
nnoremap <C-n> :tabnew<CR>

inoremap <F9> <Esc>:w<CR>:!~/runcpp.sh %:p:t %:p:h<CR>
nnoremap <F9> :w<CR>:!~/runcpp.sh %:p:t %:p:h<CR>

syntax on
colorscheme desert
set filetype=cpp
set background=dark
hi Normal ctermfg=white ctermbg=black

```

### 2.2 Stress

```

g++ gen.cpp -o gen.out
g++ ac.cpp -o ac.out
g++ wa.cpp -o wa.out
for ((i=0;;i++))
do
  echo "$i"
  ./gen.out > in.txt
  ./ac.out < in.txt > ac.txt
  ./wa.out < in.txt > wa.txt
  if [ "$?" -ne 0 ]; then
    exit 1
  fi
  diff ac.txt wa.txt || break
done

```

### 2.3 Run Sample

```

prog=$1
shift
g++ -O2 -std=c++20 -fsanitize=address -Wall -Wextra -Wshadow ${prog}.cpp -o ${prog}.out
for f in "$@"; do
  out=${prog}_${(basename "$f")}.out
  echo "input: $f"
  cat "$f"
  echo "output: $out"
  ./${prog}.out < "$f" | tee "$out"
  echo
done

```

### 2.4 Runcpp.sh

```

#!/bin/bash
clear
echo "Start compiling $1..."
echo
g++ -O2 -std=c++20 -fsanitize=address -Wall -Wextra -Wshadow $2/$1 -o $2/out
if [ "$?" -ne 0 ]
then
  exit 1
fi
echo
echo "Done compiling"
echo =====
echo "Input file:"
echo

```

## 1 Reminder

### 1.1 Bug List

- 沒開 long long
- 本地編譯請開-Wall -Wextra -Wshadow -fsanitize=address
- 陣列越出界 / 開不夠大 / 開太大本地 compile 噴怪 error
- 傳之前先確定選對檔案
- 變數打錯
- 0-base / 1-base
- 忘記初始化
- == 打成 =
- dp[i] 從 dp[i-1] 轉移時忘記特判 i > 0
- std::sort 比較運算子寫成 < 或是讓 = 的情況為 true
- 漏 case / 分 case 要好好想
- 線段樹改值懶標初始值不能設為 0
- 少碰動態開點，能離散化就離散化
- 能不用浮點數運算就不用
- DFS 的時候不小心覆寫到全域變數
- 記得刪 cerr

### 1.2 OwO

- 可以構造複雜點的測資幫助思考
- 真的卡太久請跳題
- Enjoy The Contest!

```

16 cat $2/in.txt
17 echo
18 echo "===== "
19 echo
20 declare startTime=`date +%s%N`
21 $2/out < $2/in.txt > $2/out.txt
22 declare endTime=`date +%s%N`
23 delta=`expr $endTime - $startTime`
24 delta=`expr $delta / 1000000`
25 cat $2/out.txt
26 echo
27 echo "time: $delta ms"

```

## 2.5 Others

```

1 #pragma GCC optimize("Ofast,unroll-loops,no-stack-
2   protector,fast-math")
3 #pragma GCC target("see,see2,see3,see4,avx2,bmi,bmi2,
4   lzcnt,popcnt,tune=native")
5 #pragma GCC optimize("trapv")
6 mt19937 gen(chrono::steady_clock::now(),
7   time_since_epoch().count());
8 uniform_int_distribution<int> dis(1, 100);
9 cout << dis(gen) << endl;
10 shuffle(v.begin(), v.end(), gen);
11
12 struct edge {
13   int a, b, w;
14   friend istream& operator>>(istream& in, edge& x) {
15     in >> x.a >> x.b >> x.w; }
16   friend ostream& operator<<(ostream& out, const edge
17   & x) {
18     out << "(" << x.a << ", " << x.b << ", " << x.w
19     << ")";
20     return out;
21   }
22 };
23 struct cmp {
24   bool operator()(const edge& x, const edge& y) const
25   { return x.w < y.w; }
26 };
27 set<edge, cmp> st; // 遞增
28 map<edge, long long, cmp> mp; // 遞增
29 priority_queue<edge, vector<edge>, cmp> pq; // 遞減
30
31 #include <bits/extc++.h>
32 #include <ext/pb_ds/assoc_container.hpp>
33 #include <ext/pb_ds/tree_policy.hpp>
34 using namespace __gnu_pbds;
35
36 // map
37 tree<int, int, less<>, rb_tree_tag,
38   tree_order_statistics_node_update> tr;
39 tr.order_of_key(element);
40 tr.find_by_order(rank);
41
42 // set
43 tree<int, null_type, less<>, rb_tree_tag,
44   tree_order_statistics_node_update> tr;
45 tr.order_of_key(element);
46 tr.find_by_order(rank);
47
48 gp_hash_table<int, int> ht;
49 ht.find(element);
50 ht.insert({key, value});
51 ht.erase(element);
52
53 // priority queue Big First
54 __gnu_pbds::priority_queue<int, less<int>> big_q;
55 __gnu_pbds::priority_queue<int, greater<int>> small_q;
56   // Small First
57 q1.join(q2); // join

```

## 3 Data Structure

### 3.1 BIT

```

1 struct BIT {
2   int n;
3   long long bit[N];
4

```

```

5   void init(int x, vector<long long> &a) {
6     n = x;
7     for (int i = 1, j; i <= n; i++) {
8       bit[i] += a[i - 1], j = i + (i & -i);
9       if (j <= n) bit[j] += bit[i];
10    }
11  }
12
13  void update(int x, long long dif) {
14    while (x <= n) bit[x] += dif, x += x & -x;
15  }
16
17  long long query(int l, int r) {
18    if (l != 1) return query(1, r) - query(1, l - 1);
19
20    long long ret = 0;
21    while (l <= r) ret += bit[r], r -= r & -r;
22  }
23 } bm;

```

### 3.2 Lazy Propagation Segment Tree

```

1 struct lazy_propagation{
2   // 0-based, [l, r], tg[0]->add, tg[1]->set
3   ll seg[N * 4], tg[2][N*4];
4   void assign (bool op, ll val, int idx){
5     if (op == 0){
6       if (tg[1][idx]) tg[1][idx] += val;
7       else tg[0][idx] += val;
8     }
9     else seg[idx] = 0, tg[0][idx] = 0, tg[1][idx]
10      = val;
11   }
12   sum (int idx, int len){
13     if (tg[1][idx]) return tg[1][idx] * len;
14     return tg[0][idx] * len + seg[idx];
15   }
16   void pull (int idx, int len){
17     seg[idx] = sum(2*idx, (len+1)/2) + sum(2*idx+1,
18       len/2);
19   }
20   void push (int idx){
21     if (!tg[0][idx] && !tg[1][idx]) return ;
22     if (tg[0][idx]){
23       assign(0, tg[0][idx], 2*idx);
24       assign(0, tg[0][idx], 2*idx+1);
25       tg[0][idx] = 0;
26     }
27     else{
28       assign(1, tg[1][idx], 2*idx);
29       assign(1, tg[1][idx], 2*idx+1);
30       tg[1][idx] = 0;
31     }
32   }
33   void update (bool op, ll val, int gl, int gr, int l
34 , int r, int idx){
35     if (r < l || gr < l || r < gl) return ;
36     if (gl <= l && r <= gr){
37       assign(op, val, idx);
38       return ;
39     }
40
41     int mid = (l + r) / 2;
42     push(idx);
43     update(op, val, gl, gr, l, mid, 2*idx);
44     update(op, val, gl, gr, mid+1, r, 2*idx+1);
45     pull(idx, r-l+1);
46   }
47   query (int gl, int gr, int l, int r, int idx){
48     if (r < l || gr < l || r < gl) return 0;
49     if (gl <= l && r <= gr) return sum(idx, r-l+1);
50
51     push(idx), pull(idx, r-l+1);
52     int mid = (l + r) / 2;
53     return query(gl, gr, l, mid, 2*idx) + query(gl,
54       gr, mid+1, r, 2*idx+1);
55   }
56 }bm;

```

### 3.3 Treap

```

1 mt19937 rng(random_device{}());
2 struct Treap {
3     Treap *l, *r;
4     int val, sum, real, tag, num, pri, rev;
5     Treap(int k) {
6         l = r = NULL;
7         val = sum = k;
8         num = 1;
9         real = -1;
10        tag = 0;
11        rev = 0;
12        pri = rng();
13    }
14    int siz(Treap *now) { return now ? now->num : 0ll; }
15    int sum(Treap *now) {
16        if (!now) return 0;
17        if (now->real != -1) return (now->real + now->tag)
18            * now->num;
19        return now->sum + now->tag * now->num;
20    }
21    void pull(Treap *&now) {
22        now->num = siz(now->l) + siz(now->r) + 1ll;
23        now->sum = sum(now->l) + sum(now->r) + now->val +
24            now->tag;
25    }
26    void push(Treap *&now) {
27        if (now->rev) {
28            swap(now->l, now->r);
29            now->l->rev ^= 1;
30            now->r->rev ^= 1;
31            now->rev = 0;
32        }
33        if (now->real != -1) {
34            now->real += now->tag;
35            if (now->l) {
36                now->l->tag = 0;
37                now->l->real = now->real;
38                now->l->val = now->real;
39            }
40            if (now->r) {
41                now->r->tag = 0;
42                now->r->real = now->real;
43                now->r->val = now->real;
44            }
45            now->val = now->real;
46            now->sum = now->real * now->num;
47            now->real = -1;
48            now->tag = 0;
49        } else {
50            if (now->l) now->l->tag += now->tag;
51            if (now->r) now->r->tag += now->tag;
52            now->sum += sum(now);
53            now->val += now->tag;
54            now->tag = 0;
55        }
56    Treap *merge(Treap *a, Treap *b) {
57        if (!a || !b) return a ?: b;
58        else if (a->pri > b->pri) {
59            push(a);
60            a->r = merge(a->r, b);
61            pull(a);
62            return a;
63        } else {
64            push(b);
65            b->l = merge(a, b->l);
66            pull(b);
67            return b;
68        }
69    }
70    void split_size(Treap *rt, Treap *&a, Treap *&b, int
71        val) {
72        if (!rt) {
73            a = b = NULL;
74            return;
75        }
76        push(rt);
77        if (siz(rt->l) + 1 > val) {

```

```

78            split_size(rt->l, a, b->l, val);
79            pull(b);
80        } else {
81            a = rt;
82            split_size(rt->r, a->r, b, val - siz(a->l) - 1)
83            ;
84            pull(a);
85        }
86    void split_val(Treap *rt, Treap *&a, Treap *&b, int val
87        ) {
88        if (!rt) {
89            a = b = NULL;
90            return;
91        }
92        push(rt);
93        if (rt->val <= val) {
94            a = rt;
95            split_val(rt->r, a->r, b, val);
96            pull(a);
97        } else {
98            b = rt;
99            split_val(rt->l, a, b->l, val);
100           pull(b);
101      }

```

### 3.4 Persistent Treap

```

1 struct node {
2     node *l, *r;
3     char c;
4     int v, sz;
5     node(char x = '$') : c(x), v(mt()), sz(1) {
6         l = r = nullptr;
7     }
8     node(node* p) { *this = *p; }
9     void pull() {
10        sz = 1;
11        for (auto i : {l, r})
12            if (i) sz += i->sz;
13    }
14 } arr[maxn], *ptr = arr;
15 inline int size(node* p) { return p ? p->sz : 0; }
16 node* merge(node* a, node* b) {
17     if (!a || !b) return a ?: b;
18     if (a->v < b->v) {
19         node* ret = new (ptr++) node(a);
20         ret->r = merge(ret->r, b), ret->pull();
21         return ret;
22     } else {
23         node* ret = new (ptr++) node(b);
24         ret->l = merge(a, ret->l), ret->pull();
25         return ret;
26     }
27 }
28 P<node*> split(node* p, int k) {
29     if (!p) return {nullptr, nullptr};
30     if (k >= size(p->l) + 1) {
31         auto [a, b] = split(p->r, k - size(p->l) - 1);
32         node* ret = new (ptr++) node(p);
33         ret->r = a, ret->pull();
34         return {ret, b};
35     } else {
36         auto [a, b] = split(p->l, k);
37         node* ret = new (ptr++) node(p);
38         ret->l = b, ret->pull();
39         return {a, ret};
40     }
41 }

```

### 3.5 Li Chao Tree

```

1 const int eps=1e-9;
2 struct line
3 {
4     double m,k;
5 }p[N];
6 int seg[N<<2],cnt;
7 double cal(int id,int x){return 1.0*p[id].m*x+p[id].k;}
8 void add(int x0,int y0,int x1,int y1)
9 {

```

```

10    cnt++;
11    if(x0==x1)p[cnt].m=0,p[cnt].k=max(y0,y1);
12    else p[cnt].m=1.0*(y1-y0)/(x1-x0),p[cnt].k=y0-p[
13        cnt].m*x0;
14}
15 void update(int x,int l,int r,int ql,int qr,int u)
16 {
17     int v=seg[x],mid=(l+r)>>1;
18     double resu=cal(u,mid),resv=cal(v,mid);
19     if(qr<l||r<ql) return;
20     if(ql<=l&&r<=qr)
21     {
22         if(l==r)
23         {
24             if(resu>resv)seg[x]=u;
25             return;
26         }
27         if(p[u].m-p[v].m>eps)
28         {
29             if(resu-resv>eps)
30             {
31                 seg[x]=u;
32                 update(x<<1,l,mid,ql,qr,v);
33             }
34             else update(x<<1|1,mid+1,r,ql,qr,u);
35         }
36         else if(p[v].m-p[u].m>eps)
37         {
38             if(resu-resv>eps)
39             {
40                 seg[x]=u;
41                 update(x<<1|1,mid+1,r,ql,qr,v);
42             }
43             else update(x<<1,l,mid,ql,qr,u);
44         }
45         else if(p[u].k-p[v].k>eps)seg[x]=u;
46         return;
47     }
48     update(x<<1,l,mid,ql,qr,u);
49     update(x<<1|1,mid+1,r,ql,qr,u);
50 }
51 double ask(int x,int l,int r,int qx)
52 {
53     if(r<qx||qx<l) return{0,0};
54     int mid=(l+r)>>1;
55     double res=cal(seg[x],qx);
56     if(l==r) return res;
57     return max({res,seg[x],ask(x<<1,l,mid,qx),ask(x
      <<1|1,mid+1,r,qx)});
```

## 3.6 LineContainer

```

1 struct Line {
2     mutable ll k, m, p;
3     bool operator<(const Line& o) const { return k < o.
4         k; }
5     bool operator<(ll x) const { return p < x; }
6 };
7
8 struct LineContainer : multiset<Line, less<> {
9     // (for doubles, use inf = 1/.0, div(a,b) = a/b)
10    static const ll inf = LONG_MAX;
11    ll div(ll a, ll b) { // floored division
12        return a / b - ((a ^ b) < 0 && a % b); }
13    bool isect(iterator x, iterator y) {
14        if (y == end()) return x->p = inf, 0;
15        if (x->k == y->k) x->p = x->m > y->m ? inf : -
16            inf;
17        else x->p = div(y->m - x->m, x->k - y->k);
18        return x->p >= y->p;
19    }
20    void add(ll k, ll m) {
21        auto z = insert({k, m, 0}), y = z++, x = y;
22        while (isect(y, z)) z = erase(z);
23        if (x != begin() && isect(--x, y)) isect(x, y =
24            erase(y));
25        while ((y = x) != begin() && (--x)->p >= y->p)
26            isect(x, erase(y));
27    }
28    ll query(ll x) {
```

```

27        assert(!empty());
28        auto l = *lower_bound(x);
29        return l.k * x + l.m;
30    }
31};
```

## 3.7 Sparse Table

```

1 int a[N];
2 int st[N][30];
3 void pre(int n)
4 {
5     FOR(i,1,n+1)st[i][0]=a[i];
6     for(int j=1;(1<<j)<=n+1;j++)
7         for(int i=0;i+(1<<j)<=n+1;i++)
8             st[i][j]=min(st[i][j-1],st[i+(1<<(j-1))
9                ][j-1]);
10 }
11 int ask(int l,int r)
12 {
13     int k=__lg(r-l+1);
14     return min(st[1][k],st[r-(1<<k)+1][k]);
15 }
```

## 3.8 Time Segment Tree

```

1 constexpr int maxn = 1e5 + 5;
2 V<P<int>> arr[(maxn + 1) << 2];
3 V<int> dsu, sz;
4 V<tuple<int, int, int>> his;
5 int cnt, q;
6 int find(int x) {
7     return x == dsu[x] ? x : find(dsu[x]);
8 };
9 inline bool merge(int x, int y) {
10     int a = find(x), b = find(y);
11     if (a == b) return false;
12     if (sz[a] > sz[b]) swap(a, b);
13     his.emplace_back(a, b, sz[b]), dsu[a] = b, sz[b] +=
14         sz[a];
15     return true;
16 };
17 inline void undo() {
18     auto [a, b, s] = his.back();
19     his.pop_back();
20     dsu[a] = a, sz[b] = s;
21 }
22 #define m ((l + r) >> 1)
23 void insert(int ql, int qr, P<int> x, int i = 1, int l
24 = 0, int r = q) {
25     // debug(ql, qr, x); return;
26     if (qr <= l || r <= ql) return;
27     if (ql <= l && r <= qr) {
28         arr[i].push_back(x);
29         return;
30     }
31     if (qr <= m)
32         insert(ql, qr, x, i << 1, l, m);
33     else if (m <= ql)
34         insert(ql, qr, x, i << 1 | 1, m, r);
35     else {
36         insert(ql, qr, x, i << 1, l, m);
37         insert(ql, qr, x, i << 1 | 1, m, r);
38     }
39 }
40 void traversal(V<int>& ans, int i = 1, int l = 0, int r
41 = q) {
42     int opcnt = 0;
43     // debug(i, l, r);
44     for (auto [a, b] : arr[i])
45         if (merge(a, b))
46             opcnt++, cnt--;
47     if (r - l == 1)
48         ans[l] = cnt;
49     else {
50         traversal(ans, i << 1, l, m);
51         traversal(ans, i << 1 | 1, m, r);
52     }
53     while (opcnt--)
54         undo(), cnt++;
55     arr[i].clear();
56 }
```

```

54 #undef m
55 inline void solve() {
56     int n, m;
57     cin >> n >> m >> q, q++;
58     dsu.resize(cnt = n), sz.assign(n, 1);
59     iota(dsu.begin(), dsu.end(), 0);
60     // a, b, time, operation
61     unordered_map<ll, V<int>> s;
62     for (int i = 0; i < m; i++) {
63         int a, b;
64         cin >> a >> b;
65         if (a > b) swap(a, b);
66         s[((ll)a << 32) | b].emplace_back(0);
67     }
68     for (int i = 1; i < q; i++) {
69         int op, a, b;
70         cin >> op >> a >> b;
71         if (a > b) swap(a, b);
72         switch (op) {
73             case 1:
74                 s[((ll)a << 32) | b].push_back(i);
75                 break;
76             case 2:
77                 auto tmp = s[((ll)a << 32) | b].back();
78                 s[((ll)a << 32) | b].pop_back();
79                 insert(tmp, i, P<int>{a, b});
80             }
81     }
82     for (auto [p, v] : s) {
83         int a = p >> 32, b = p & -1;
84         while (v.size()) {
85             insert(v.back(), q, P<int>{a, b});
86             v.pop_back();
87         }
88     }
89     V<int> ans(q);
90     traversal(ans);
91     for (auto i : ans)
92         cout << i << ' ';
93     cout << endl;
94 }
```

### 3.9 Dynamic Median

```

1 struct Dynamic_Median {
2     multiset<long long> lo, hi;
3     long long slo = 0, shi = 0;
4     void rebalance() {
5         // keep sz(lo) >= sz(hi) and sz(lo) - sz(hi) <= 1
6         while((int)lo.size() > (int)hi.size() + 1) {
7             auto it = prev(lo.end());
8             long long x = *it;
9             lo.erase(it); slo -= x;
10            hi.insert(x); shi += x;
11        }
12        while((int)lo.size() < (int)hi.size()) {
13            auto it = hi.begin();
14            long long x = *it;
15            hi.erase(it); shi -= x;
16            lo.insert(x); slo += x;
17        }
18    }
19    void add(long long x) {
20        if(lo.empty() || x <= *prev(lo.end())) {
21            lo.insert(x); slo += x;
22        } else {
23            hi.insert(x); shi += x;
24        }
25        rebalance();
26    }
27    void remove_one(long long x) {
28        if(!lo.empty() && x <= *prev(lo.end())) {
29            auto it = lo.find(x);
30            if(it != lo.end()) {
31                lo.erase(it); slo -= x;
32            }
33        } else {
34            auto it2 = hi.find(x);
35            hi.erase(it2); shi -= x;
36        }
37    }
38 }
```

```

38     }
39     else {
40         auto it = hi.find(x);
41         if(it != hi.end()) {
42             hi.erase(it); shi -= x;
43         }
44     } else {
45         auto it2 = lo.find(x);
46         lo.erase(it2); slo -= x;
47     }
48     rebalance();
49 }
50 };
51 }
```

### 3.10 SOS DP

```

1 for (int mask = 0; mask < (1 << n); mask++) {
2     for (int submask = mask; submask != 0; submask = (submask - 1) & mask) {
3         int subset = mask ^ submask;
4     }
5 }
```

## 4 Flow / Matching

### 4.1 Dinic

```

1 using namespace std;
2 const int N = 2000 + 5;
3 int n, m, s, t, level[N], iter[N];
4 struct edge {int to, cap, rev;};
5 vector<edge> path[N];
6 void add(int a, int b, int c) {
7     path[a].pb({b, c, sz(path[b])});
8     path[b].pb({a, 0, sz(path[a]) - 1});
9 }
10 void bfs() {
11     memset(level, -1, sizeof(level));
12     level[s] = 0;
13     queue<int> q;
14     q.push(s);
15     while (q.size()) {
16         int now = q.front(); q.pop();
17         for (edge e : path[now]) if (e.cap > 0 && level[e.to] == -1) {
18             level[e.to] = level[now] + 1;
19             q.push(e.to);
20         }
21     }
22 }
23 int dfs(int now, int flow) {
24     if (now == t) return flow;
25     for (int &i = iter[now]; i < sz(path[now]); i++) {
26         edge &e = path[now][i];
27         if (e.cap > 0 && level[e.to] == level[now] + 1) {
28             int res = dfs(e.to, min(flow, e.cap));
29             if (res > 0) {
30                 e.cap -= res;
31                 path[e.to][e.rev].cap += res;
32                 return res;
33             }
34         }
35     }
36     return 0;
37 }
38 int dinic() {
39     int res = 0;
40     while (true) {
41         bfs();
42         if (level[t] == -1) break;
43         memset(iter, 0, sizeof(iter));
44         int now = 0;
45         while ((now = dfs(s, INF)) > 0) res += now;
46     }
47     return res;
48 }
```

### 4.2 MCMF

```

1 struct MCMF {
```

```

2   int n, s, t, par[N + 5], p_i[N + 5], dis[N + 5],
3     vis[N + 5];
4   struct edge {
5     int to, cap, rev, cost;
6   };
7   vector<edge> path[N];
8   void init(int _n, int _s, int _t) {
9     n = _n, s = _s, t = _t;
10    FOR(i, 0, 2 * n + 5)
11      par[i] = p_i[i] = vis[i] = 0;
12  }
13  void add(int a, int b, int c, int d) {
14    path[a].pb({b, c, sz(path[b]), d});
15    path[b].pb({a, 0, sz(path[a]) - 1, -d});
16  }
17  void spfa() {
18    FOR(i, 0, n * 2 + 5)
19      dis[i] = INF,
20      vis[i] = 0;
21    dis[s] = 0;
22    queue<int> q;
23    q.push(s);
24    while (!q.empty()) {
25      int now = q.front();
26      q.pop();
27      vis[now] = 0;
28      for (int i = 0; i < sz(path[now]); i++) {
29        edge e = path[now][i];
30        if (e.cap > 0 && dis[e.to] > dis[now] + e.cost) {
31          dis[e.to] = dis[now] + e.cost;
32          par[e.to] = now;
33          p_i[e.to] = i;
34          if (vis[e.to] == 0) {
35            vis[e.to] = 1;
36            q.push(e.to);
37          }
38        }
39      }
40    }
41    pii flow() {
42      int flow = 0, cost = 0;
43      while (true) {
44        spfa();
45        if (dis[t] == INF)
46          break;
47        int mn = INF;
48        for (int i = t; i != s; i = par[i])
49          mn = min(mn, path[par[i]][p_i[i]].cap);
50        flow += mn;
51        cost += dis[t] * mn;
52        for (int i = t; i != s; i = par[i]) {
53          edge &now = path[par[i]][p_i[i]];
54          now.cap -= mn;
55          path[i][now.rev].cap += mn;
56        }
57      }
58      return mp(flow, cost);
59    }
60  };

```

### 4.3 KM

```

1 struct KM {
2   int n, mx[1005], my[1005], pa[1005];
3   int g[1005][1005], lx[1005], ly[1005], sy[1005];
4   bool vx[1005], vy[1005];
5   void init(int _n) {
6     n = _n;
7     FOR(i, 1, n + 1)
8       fill(g[i], g[i] + 1 + n, 0);
9   }
10  void add(int a, int b, int c) { g[a][b] = c; }
11  void augment(int y) {
12    for (int x, z; y; y = z)
13      x = pa[y], z = mx[x], my[y] = x, mx[x] = y;
14  }
15  void bfs(int st) {
16    FOR(i, 1, n + 1)
17      sy[i] = INF,
18      vx[i] = vy[i] = 0;
19  }

```

```

19  queue<int> q;
20  q.push(st);
21  for (;;) {
22    while (!q.empty()) {
23      int x = q.front();
24      q.pop();
25      vx[x] = 1;
26      FOR(y, 1, n + 1)
27        if (!vy[y]) {
28          int t = lx[x] + ly[y] - g[x][y];
29          if (t == 0) {
30            pa[y] = x;
31            if (!my[y])
32              augment(y);
33            return;
34          }
35          vy[y] = 1, q.push(my[y]);
36        } else if (sy[y] > t)
37          pa[y] = x, sy[y] = t;
38      }
39      int cut = INF;
40      FOR(y, 1, n + 1)
41        if (!vy[y] && cut > sy[y]) cut = sy[y];
42      FOR(j, 1, n + 1) {
43        if (vx[j]) lx[j] -= cut;
44        if (vy[j])
45          ly[j] += cut;
46        else
47          sy[j] -= cut;
48      }
49      FOR(y, 1, n + 1) {
50        if (!vy[y] && sy[y] == 0) {
51          if (!my[y])
52            augment(y);
53            return;
54        }
55        vy[y] = 1;
56        q.push(my[y]);
57      }
58    }
59  }
60  int solve() {
61    fill(mx, mx + n + 1, 0);
62    fill(my, my + n + 1, 0);
63    fill(ly, ly + n + 1, 0);
64    fill(lx, lx + n + 1, 0);
65    FOR(x, 1, n + 1)
66      FOR(y, 1, n + 1)
67        lx[x] = max(lx[x], g[x][y]);
68    FOR(x, 1, n + 1)
69      bfs(x);
70    int ans = 0;
71    FOR(y, 1, n + 1)
72      ans += g[my[y]][y];
73    return ans;
74  }
75  };
76  };
77  };

```

### 4.4 Hopcroft-Karp

```

1 struct HopcroftKarp {
2   // id: X = [1, nx], Y = [nx+1, nx+ny]
3   int n, nx, ny, m, MXCNT;
4   vector<vector<int>> g;
5   vector<int> mx, my, dis, vis;
6   void init(int nnx, int nny, int mm) {
7     nx = nnx, ny = nny, m = mm;
8     n = nx + ny + 1;
9     g.clear();
10    g.resize(n);
11  }
12  void add(int x, int y) {
13    g[x].emplace_back(y);
14    g[y].emplace_back(x);
15  }
16  bool dfs(int x) {
17    vis[x] = true;
18    Each(y, g[x]) {
19      int px = my[y];
20      if (px == -1 ||

```

```

21     (dis[px] == dis[x] + 1 &&
22      !vis[px] && dfs(px))) {
23         mx[x] = y;
24         my[y] = x;
25         return true;
26     }
27 }
28 return false;
29 }
30 void get() {
31     mx.clear();
32     mx.resize(n, -1);
33     my.clear();
34     my.resize(n, -1);
35
36     while (true) {
37         queue<int> q;
38         dis.clear();
39         dis.resize(n, -1);
40         for (int x = 1; x <= nx; x++) {
41             if (mx[x] == -1) {
42                 dis[x] = 0;
43                 q.push(x);
44             }
45         }
46         while (!q.empty()) {
47             int x = q.front();
48             q.pop();
49             Each(y, g[x]) {
50                 if (my[y] != -1 && dis[my[y]] ==
51                     -1) {
52                     dis[my[y]] = dis[x] + 1;
53                     q.push(my[y]);
54                 }
55             }
56
57             bool brk = true;
58             vis.clear();
59             vis.resize(n, 0);
60             for (int x = 1; x <= nx; x++)
61                 if (mx[x] == -1 && dfs(x))
62                     brk = false;
63
64             if (brk) break;
65         }
66         MXCNT = 0;
67         for (int x = 1; x <= nx; x++)
68             if (mx[x] != -1) MXCNT++;
69     }
70 } hk;

```

## 4.5 Blossom

```

1 const int N=5e2+10;
2 struct Graph{
3     int to[N],bro[N],head[N],e;
4     int lnk[N],vis[N],stp,n;
5     void init(int _n){
6         stp=0;e=1;n=_n;
7         FOR(i,0,n+1)head[i]=lnk[i]=vis[i]=0;
8     }
9     void add(int u,int v){
10        to[e]=v,bro[e]=head[u],head[u]=e++;
11        to[e]=u,bro[e]=head[v],head[v]=e++;
12    }
13    bool dfs(int x){
14        vis[x]=stp;
15        for(int i=head[x];i;i=bro[i])
16        {
17            int v=to[i];
18            if(!lnk[v])
19            {
20                lnk[x]=v;lnk[v]=x;
21                return true;
22            }
23            else if(vis[lnk[v]]<stp)
24            {
25                int w=lnk[v];
26                lnk[x]=v,lnk[v]=x,lnk[w]=0;
27                if(dfs(w))return true;
28                lnk[w]=v,lnk[v]=w,lnk[x]=0;
29            }
30        }
31    }
32 }
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
717
718
719
719
720
721
722
723
724
725
726
727
727
728
729
729
730
731
732
733
734
735
735
736
737
737
738
739
739
740
741
742
742
743
744
744
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
```

```

9     seg[x].mx = seg[x].sum = val;
10    return;
11}
12int mid = (l + r) >> 1;
13if (qx <= mid) update(x << 1, l, mid, qx, val);
14else update(x << 1 | 1, mid + 1, r, qx, val);
15seg[x].mx = max(seg[x << 1].mx, seg[x << 1 | 1].mx);
16;
17seg[x].sum = seg[x << 1].sum + seg[x << 1 | 1].sum;
18}
19int big(int x, int l, int r, int ql, int qr) {
20if (ql <= l && r <= qr) return seg[x].mx;
21int mid = (l + r) >> 1;
22int res = -INF;
23if (ql <= mid) res = max(res, big(x << 1, l, mid,
24    ql, qr));
25if (mid < qr) res = max(res, big(x << 1 | 1, mid +
26    1, r, ql, qr));
27return res;
28}
29int ask(int x, int l, int r, int ql, int qr) {
30if (ql <= l && r <= qr) return seg[x].sum;
31int mid = (l + r) >> 1;
32int res = 0;
33if (ql <= mid) res += ask(x << 1, l, mid, ql, qr);
34if (mid < qr) res += ask(x << 1 | 1, mid + 1, r, ql
35    , qr);
36return res;
37}
38void dfs1(int now) {
39    son[now] = -1;
40    num[now] = 1;
41    for (auto i : path[now]) {
42        if (!dep[i]) {
43            dep[i] = dep[now] + 1;
44            p[i] = now;
45            dfs1(i);
46            num[now] += num[i];
47            if (son[now] == -1 || num[i] > num[son[now
48                ]]) son[now] = i;
49        }
50    }
51}
52int cnt;
53void dfs2(int now, int t) {
54    top[now] = t;
55    cnt++;
56    dfn[now] = cnt;
57    if (son[now] == -1) return;
58    dfs2(son[now], t);
59    for (auto i : path[now])
60        if (i != p[now] && i != son[now])dfs2(i, i);
61}
62int path_big(int x, int y) {
63    int res = -INF;
64    while (top[x] != top[y]) {
65        if (dep[top[x]] < dep[top[y]]) swap(x, y);
66        res = max(res, big(1, 1, n, dfn[top[x]], dfn[x
67            ]));
68        x = p[top[x]];
69    }
70    if (dfn[x] > dfn[y]) swap(x, y);
71    res = max(res, big(1, 1, n, dfn[x], dfn[y]));
72    return res;
73}
74int path_sum(int x, int y) {
75    int res = 0;
76    while (top[x] != top[y]) {
77        if (dep[top[x]] < dep[top[y]]) swap(x, y);
78        res += ask(1, 1, n, dfn[top[x]], dfn[x]);
79        x = p[top[x]];
80    }
81    if (dfn[x] > dfn[y]) swap(x, y);
82    res += ask(1, 1, n, dfn[x], dfn[y]);
83    return res;
84}
85void buildTree() {
86    FOR(i, 0, n - 1) {
87        int a, b;
88        cin >> a >> b;
89        path[a].pb(b);
90        path[b].pb(a);
91    }
92}
93int main() {
94    buildHLD(1);
95    cout << "ans: " << ans << endl;
96}

```

```

85    }
86}
87void buildHLD(int root) {
88    dep[root] = 1;
89    dfs1(root);
90    dfs2(root, root);
91    FOR(i, 1, n + 1) {
92        int now;
93        cin >> now;
94        update(1, 1, n, dfn[i], now);
95    }
96}

```

## 5.2 Centroid Decomposition

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int N = 1e5 + 5;
4 vector<int> a[N];
5 int sz[N], lv[N];
6 bool used[N];
7 int f_sz(int x, int p) {
8    sz[x] = 1;
9    for (int i : a[x])
10        if (i != p && !used[i])
11            sz[x] += f_sz(i, x);
12    return sz[x];
13}
14 int f_cen(int x, int p, int total) {
15    for (int i : a[x]) {
16        if (i != p && !used[i] && 2 * sz[i] > total)
17            return f_cen(i, x, total);
18    }
19    return x;
20}
21 void cd(int x, int p) {
22    int total = f_sz(x, p);
23    int cen = f_cen(x, p, total);
24    lv[cen] = lv[p] + 1;
25    used[cen] = 1;
26    // cout << "cd: " << x << " " << p << " " << cen <<
27    "\n";
28    for (int i : a[cen]) {
29        if (!used[i])
30            cd(i, cen);
31    }
32 int main() {
33    ios_base::sync_with_stdio(0);
34    cin.tie(0);
35    int n;
36    cin >> n;
37    for (int i = 0, x, y; i < n - 1; i++) {
38        cin >> x >> y;
39        a[x].push_back(y);
40        a[y].push_back(x);
41    }
42    cd(1, 0);
43    for (int i = 1; i <= n; i++)
44        cout << (char)('A' + lv[i] - 1) << " ";
45    cout << "\n";
46}

```

## 5.3 Bellman-Ford + SPFA

```

1 int n, m;
2
3 // Graph
4 vector<vector<pair<int, ll>>> g;
5 vector<ll> dis;
6 vector<bool> negCycle;
7
8 // SPFA
9 vector<int> rlx;
10 queue<int> q;
11 vector<bool> inq;
12 vector<int> pa;
13 void SPFA(vector<int>& src) {
14    dis.assign(n + 1, LINF);
15    negCycle.assign(n + 1, false);
16    rlx.assign(n + 1, 0);
17    while (!q.empty()) q.pop();
18}

```

```

18     inq.assign(n + 1, false);
19     pa.assign(n + 1, -1);
20
21     for (auto& s : src) {
22         dis[s] = 0;
23         q.push(s);
24         inq[s] = true;
25     }
26
27     while (!q.empty()) {
28         int u = q.front();
29         q.pop();
30         inq[u] = false;
31         if (rlx[u] >= n) {
32             negCycle[u] = true;
33         } else {
34             for (auto& e : g[u]) {
35                 int v = e.first;
36                 ll w = e.second;
37                 if (dis[v] > dis[u] + w) {
38                     dis[v] = dis[u] + w;
39                     rlx[v] = rlx[u] + 1;
40                     pa[v] = u;
41                     if (!inq[v]) {
42                         q.push(v);
43                         inq[v] = true;
44                     }
45                 }
46             }
47         }
48     }
49
50 // Bellman-Ford
51 queue<int> q;
52 vector<int> pa;
53 void BellmanFord(vector<int>& src) {
54     dis.assign(n + 1, LINF);
55     negCycle.assign(n + 1, false);
56     pa.assign(n + 1, -1);
57
58     for (auto& s : src) dis[s] = 0;
59
60     for (int rlx = 1; rlx <= n; rlx++) {
61         for (int u = 1; u <= n; u++) {
62             if (dis[u] == LINF) continue; // Important
63             if (rlx == n) negCycle[u] = true;
64         }
65     }
66
67     for (auto& e : g[u]) {
68         int v = e.first;
69         ll w = e.second;
70         if (dis[v] > dis[u] + w) {
71             dis[v] = dis[u] + w;
72             pa[v] = u;
73             if (rlx == n) negCycle[v] = true;
74         }
75     }
76
77 // Negative Cycle Detection
78 void NegCycleDetect() {
79     /* No Neg Cycle: NO
80     Exist Any Neg Cycle:
81     YES
82     v0 v1 v2 ... vk v0 */
83
84     vector<int> src;
85     for (int i = 1; i <= n; i++)
86         src.emplace_back(i);
87
88     SPFA(src);
89     // BellmanFord(src);
90
91     int ptr = -1;
92     for (int i = 1; i <= n; i++)
93         if (negCycle[i]) {
94             ptr = i;
95             break;
96         }
97
98     if (ptr == -1) {
99         cout << "NO" << endl, void();
100    }
101
102    cout << "YES\n";
103    vector<int> ans;
104    vector<bool> vis(n + 1, false);
105
106    while (true) {
107        ans.emplace_back(ptr);
108        if (vis[ptr]) break;
109        vis[ptr] = true;
110        ptr = pa[ptr];
111    }
112    reverse(ans.begin(), ans.end());
113
114    vis.assign(n + 1, false);
115    for (auto& x : ans) {
116        cout << x << ' ';
117        if (vis[x]) break;
118        vis[x] = true;
119    }
120    cout << endl;
121
122 // Distance Calculation
123 void calcDis(int s) {
124     vector<int> src;
125     src.emplace_back(s);
126     SPFA(src);
127     // BellmanFord(src);
128
129     while (!q.empty()) q.pop();
130     for (int i = 1; i <= n; i++)
131         if (negCycle[i]) q.push(i);
132
133     while (!q.empty()) {
134         int u = q.front();
135         q.pop();
136         for (auto& e : g[u]) {
137             int v = e.first;
138             if (!negCycle[v]) {
139                 q.push(v);
140                 negCycle[v] = true;
141             }
142         }
143     }
144 }

```

## 5.4 BCC - AP

```

1 int n, m;
2 int low[maxn], dfn[maxn], instp;
3 vector<int> E, g[maxn];
4 bitset<maxn> isap;
5 bitset<maxm> vis;
6 stack<int> stk;
7 int bccnt;
8 vector<int> bcc[maxn];
9 inline void popout(int u) {
10     bccnt++;
11     bcc[bccnt].emplace_back(u);
12     while (!stk.empty()) {
13         int v = stk.top();
14         if (u == v) break;
15         stk.pop();
16         bcc[bccnt].emplace_back(v);
17     }
18 }
19 void dfs(int u, bool rt = 0) {
20     stk.push(u);
21     low[u] = dfn[u] = ++instp;
22     int kid = 0;
23     Each(e, g[u]) {
24         if (vis[e]) continue;
25         vis[e] = true;
26         int v = E[e] ^ u;
27         if (!dfn[v]) {
28             // tree edge
29             kid++;
30             dfs(v);
31             low[u] = min(low[u], low[v]);
32             if (!rt && low[v] >= dfn[u]) {
33                 // bcc found: u is ap
34             }
35         }
36     }
37 }

```

```

34         isap[u] = true;
35         popout(u);
36     }
37 } else {
38     // back edge
39     low[u] = min(low[u], dfn[v]);
40 }
41 // special case: root
42 if (rt) {
43     if (kid > 1) isap[u] = true;
44     popout(u);
45 }
46 }
47 void init() {
48     cin >> n >> m;
49     fill(low, low + maxn, INF);
50     REP(i, m) {
51         int u, v;
52         cin >> u >> v;
53         g[u].emplace_back(i);
54         g[v].emplace_back(i);
55         E.emplace_back(u ^ v);
56     }
57 }
58 void solve() {
59     FOR(i, 1, n + 1, 1) {
60         if (!dfn[i]) dfs(i, true);
61     }
62     vector<int> ans;
63     int cnt = 0;
64     FOR(i, 1, n + 1, 1) {
65         if (isap[i]) cnt++, ans.emplace_back(i);
66     }
67     cout << cnt << endl;
68     Each(i, ans) cout << i << ' ';
69     cout << endl;
70 }
71 }
```

## 5.5 BCC - Bridge

```

1 int n, m;
2 vector<int> g[maxn], E;
3 int low[maxn], dfn[maxn], instp;
4 int bccnt, bccid[maxn];
5 stack<int> stk;
6 bitset<maxm> vis, isbreg;
7 void init() {
8     cin >> n >> m;
9     REP(i, m) {
10         int u, v;
11         cin >> u >> v;
12         E.emplace_back(u ^ v);
13         g[u].emplace_back(i);
14         g[v].emplace_back(i);
15     }
16     fill(low, low + maxn, INF);
17 }
18 void popout(int u) {
19     bccnt++;
20     while (!stk.empty()) {
21         int v = stk.top();
22         if (v == u) break;
23         stk.pop();
24         bccid[v] = bccnt;
25     }
26 }
27 void dfs(int u) {
28     stk.push(u);
29     low[u] = dfn[u] = ++instp;
30
31     Each(e, g[u]) {
32         if (vis[e]) continue;
33         vis[e] = true;
34
35         int v = E[e] ^ u;
36         if (dfn[v]) {
37             // back edge
38             low[u] = min(low[u], dfn[v]);
39         } else {
40             // tree edge
41             dfs(v);
42         }
43     }
44 }
```

```

42     low[u] = min(low[u], low[v]);
43     if (low[v] == dfn[v]) {
44         isbreg[e] = true;
45         popout(u);
46     }
47 }
48 }
49 void solve() {
50     FOR(i, 1, n + 1, 1) {
51         if (!dfn[i]) dfs(i);
52     }
53     vector<pii> ans;
54     vis.reset();
55     FOR(u, 1, n + 1, 1) {
56         Each(e, g[u]) {
57             if (!isbreg[e] || vis[e]) continue;
58             vis[e] = true;
59             int v = E[e] ^ u;
60             ans.emplace_back(mp(u, v));
61         }
62     }
63     cout << (int)ans.size() << endl;
64     Each(e, ans) cout << e.F << ' ' << e.S << endl;
65 }
66 }
```

## 5.6 SCC - Tarjan

```

1 // 2-SAT
2 vector<int> E, g[maxn]; // 1~n, n+1~2n
3 int low[maxn], in[maxn], instp;
4 int sccnt, sccid[maxn];
5 stack<int> stk;
6 bitset<maxn> ins, vis;
7 int n, m;
8 void init() {
9     cin >> m >> n;
10    E.clear();
11    fill(g, g + maxn, vector<int>());
12    fill(low, low + maxn, INF);
13    memset(in, 0, sizeof(in));
14    instp = 1;
15    sccnt = 0;
16    memset(sccid, 0, sizeof(sccid));
17    ins.reset();
18    vis.reset();
19 }
20 inline int no(int u) {
21     return (u > n ? u - n : u + n);
22 }
23 int ecnt = 0;
24 inline void clause(int u, int v) {
25     E.eb(no(u) ^ v);
26     g[no(u)].eb(ecnt++);
27     E.eb(no(v) ^ u);
28     g[no(v)].eb(ecnt++);
29 }
30 void dfs(int u) {
31     in[u] = instp++;
32     low[u] = in[u];
33     stk.push(u);
34     ins[u] = true;
35
36     Each(e, g[u]) {
37         if (vis[e]) continue;
38         vis[e] = true;
39
40         int v = E[e] ^ u;
41         if (ins[v])
42             low[u] = min(low[u], in[v]);
43         else if (!in[v]) {
44             dfs(v);
45             low[u] = min(low[u], low[v]);
46         }
47     }
48     if (low[u] == in[u]) {
49         sccnt++;
50         while (!stk.empty()) {
51             int v = stk.top();
52             stk.pop();
53             ins[v] = false;
54             sccid[v] = sccnt;
55         }
56     }
57 }
```

```

55         if (u == v) break;
56     }
57 }
58 int main() {
59     init();
60     REP(i, m) {
61         char su, sv;
62         int u, v;
63         cin >> su >> u >> sv >> v;
64         if (su == '-') u = no(u);
65         if (sv == '-') v = no(v);
66         clause(u, v);
67     }
68     FOR(i, 1, 2 * n + 1, 1) {
69         if (!in[i]) dfs(i);
70     }
71     FOR(u, 1, n + 1, 1) {
72         int du = no(u);
73         if (sccid[u] == sccid[du]) {
74             return cout << "IMPOSSIBLE\n", 0;
75         }
76     }
77     FOR(u, 1, n + 1, 1) {
78         int du = no(u);
79         cout << (sccid[u] < sccid[du] ? '+' : '-')
80             ;
81     }
82     cout << endl;
83 }
```

## 5.7 SCC - Kosaraju

```

1 const int N = 1e5 + 10;
2 vector<int> ed[N], ed_b[N]; // 反邊
3 vector<int> SCC(N); // 最後SCC的分組
4 bitset<N> vis;
5 int SCC_cnt;
6 int n, m;
7 vector<int> pre; // 後序遍歷
8
9 void dfs(int x) {
10    vis[x] = 1;
11    for (int i : ed[x]) {
12        if (vis[i]) continue;
13        dfs(i);
14    }
15    pre.push_back(x);
16}
17
18 void dfs2(int x) {
19    vis[x] = 1;
20    SCC[x] = SCC_cnt;
21    for (int i : ed_b[x]) {
22        if (vis[i]) continue;
23        dfs2(i);
24    }
25}
26
27 void kosaraju() {
28    for (int i = 1; i <= n; i++) {
29        if (!vis[i]) {
30            dfs(i);
31        }
32    }
33    SCC_cnt = 0;
34    vis = 0;
35    for (int i = n - 1; i >= 0; i--) {
36        if (!vis[pre[i]]) {
37            SCC_cnt++;
38            dfs2(pre[i]);
39        }
40    }
41}
```

## 5.8 Eulerian Path - Undir

```

1 // from 1 to n
2 #define gg return cout << "IMPOSSIBLE\n", void();
3
4 int n, m;
```

```

5 vector<int> g[maxn];
6 bitset<maxn> inodd;
7
8 void init() {
9     cin >> n >> m;
10    inodd.reset();
11    for (int i = 0; i < m; i++) {
12        int u, v;
13        cin >> u >> v;
14        inodd[u] = inodd[u] ^ true;
15        inodd[v] = inodd[v] ^ true;
16        g[u].emplace_back(v);
17        g[v].emplace_back(u);
18    }
19 }
20 stack<int> stk;
21 void dfs(int u) {
22     while (!g[u].empty()) {
23         int v = g[u].back();
24         g[u].pop_back();
25         dfs(v);
26     }
27     stk.push(u);
28 }
```

## 5.9 Eulerian Path - Dir

```

1 // from node 1 to node n
2 #define gg return cout << "IMPOSSIBLE\n", 0
3
4 int n, m;
5 vector<int> g[maxn];
6 stack<int> stk;
7 int in[maxn], out[maxn];
8
9 void init() {
10    cin >> n >> m;
11    for (int i = 0; i < m; i++) {
12        int u, v;
13        cin >> u >> v;
14        g[u].emplace_back(v);
15        out[u]++, in[v]++;
16    }
17    for (int i = 1; i <= n; i++) {
18        if (i == 1 && out[i] - in[i] != 1) gg;
19        if (i == n && in[i] - out[i] != 1) gg;
20        if (i != 1 && i != n && in[i] != out[i]) gg;
21    }
22 }
23 void dfs(int u) {
24     while (!g[u].empty()) {
25         int v = g[u].back();
26         g[u].pop_back();
27         dfs(v);
28     }
29     stk.push(u);
30 }
31 void solve() {
32    dfs(1) for (int i = 1; i <= n; i++) if ((int)g[i].size()) gg;
33    while (!stk.empty()) {
34        int u = stk.top();
35        stk.pop();
36        cout << u << ' ';
37    }
38 }
```

## 5.10 Hamilton Path

```

1 // top down DP
2 // Be Aware Of Multiple Edges
3 int n, m;
4 ll dp[maxn][1<<maxn];
5 int adj[maxn][maxn];
6
7 void init() {
8    cin >> n >> m;
9    fill(dp[0], dp[maxn-1]+(1<<maxn), -1);
10}
11
12 void DP(int i, int msk) {
13    if (dp[i][msk] != -1) return;
```

```

14    dp[i][msk] = 0;
15    REP(j, n) if (j != i && (msk & (1<<j)) && adj[j][i]
16        ]) {
17        int sub = msk ^ (1<<i);
18        if (dp[j][sub] == -1) DP(j, sub);
19        dp[i][msk] += dp[j][sub] * adj[j][i];
20        if (dp[i][msk] >= MOD) dp[i][msk] %= MOD;
21    }
22
23
24 int main() {
25     WiwiHorz
26     init();
27
28     REP(i, m) {
29         int u, v;
30         cin >> u >> v;
31         if (u == v) continue;
32         adj[--u][--v]++;
33     }
34
35     dp[0][1] = 1;
36     FOR(i, 1, n, 1) {
37         dp[i][1] = 0;
38         dp[i][1|(1<<i)] = adj[0][i];
39     }
40     FOR(msk, 1, (1<<n), 1) {
41         if (msk == 1) continue;
42         dp[0][msk] = 0;
43     }
44
45     DP(n-1, (1<<n)-1);
46     cout << dp[n-1][(1<<n)-1] << endl;
47
48     return 0;
49 }

```

## 5.11 Kth Shortest Path

```

1 // time: O(|E| \lg |E| + |V| \lg |V| + K)
2 // memory: O(|E| \lg |E| + |V| + |V|K)
3 struct KSP { // 1-base
4     struct nd {
5         int u, v;
6         ll d;
7         nd(int ui = 0, int vi = 0, ll di = INF) {
8             u = ui;
9             v = vi;
10            d = di;
11        }
12    };
13    struct heap {
14        nd* edge;
15        int dep;
16        heap* chd[4];
17    };
18    static int cmp(heap* a, heap* b) { return a->edge->d > b->edge->d; }
19    struct node {
20        int v;
21        ll d;
22        heap* H;
23        nd* E;
24        node() {}
25        node(ll _d, int _v, nd* _E) {
26            d = _d;
27            v = _v;
28            E = _E;
29        }
30        node(heap* _H, ll _d) {
31            H = _H;
32            d = _d;
33        }
34        friend bool operator<(node a, node b) { return a.d > b.d; }
35    };
36    int n, k, s, t, dst[N];
37    nd* nxt[N];
38    vector<nd*> g[N], rg[N];
39    heap *nullNd, *head[N];

```

```

40    void init(int _n, int _k, int _s, int _t) {
41        n = _n;
42        k = _k;
43        s = _s;
44        t = _t;
45        for (int i = 1; i <= n; i++) {
46            g[i].clear();
47            rg[i].clear();
48            nxt[i] = NULL;
49            head[i] = NULL;
50            dst[i] = -1;
51        }
52    }
53    void addEdge(int ui, int vi, ll di) {
54        nd* e = new nd(ui, vi, di);
55        g[ui].push_back(e);
56        rg[vi].push_back(e);
57    }
58    queue<int> dfsQ;
59    void dijkstra() {
60        while (!dfsQ.size()) dfsQ.pop();
61        priority_queue<node> Q;
62        Q.push(node(0, t, NULL));
63        while (!Q.empty()) {
64            node p = Q.top();
65            Q.pop();
66            if (dst[p.v] != -1) continue;
67            dst[p.v] = p.d;
68            nxt[p.v] = p.E;
69            dfsQ.push(p.v);
70            for (auto e : rg[p.v]) Q.push(node(p.d + e->d, e->u, e));
71        }
72    }
73    heap* merge(heap* curNd, heap* newNd) {
74        if (curNd == nullNd) return newNd;
75        heap* root = new heap;
76        memcpy(root, curNd, sizeof(heap));
77        if (newNd->edge->d < curNd->edge->d) {
78            root->edge = newNd->edge;
79            root->chd[2] = newNd->chd[2];
80            root->chd[3] = newNd->chd[3];
81            newNd->edge = curNd->edge;
82            newNd->chd[2] = curNd->chd[2];
83            newNd->chd[3] = curNd->chd[3];
84        }
85        if (root->chd[0]->dep < root->chd[1]->dep)
86            root->chd[0] = merge(root->chd[0], newNd);
87        else
88            root->chd[1] = merge(root->chd[1], newNd);
89        root->dep = max(root->chd[0]->dep,
90                          root->chd[1]->dep) +
91        1;
92        return root;
93    }
94    vector<heap*> V;
95    void build() {
96        nullNd = new heap;
97        nullNd->dep = 0;
98        nullNd->edge = new nd;
99        fill(nullNd->chd, nullNd->chd + 4, nullNd);
100       while (not dfsQ.empty()) {
101           int u = dfsQ.front();
102           dfsQ.pop();
103           if (!nxt[u])
104               head[u] = nullNd;
105           else
106               head[u] = head[nxt[u]->v];
107           V.clear();
108           for (auto&& e : g[u]) {
109               int v = e->v;
110               if (dst[v] == -1) continue;
111               e->d += dst[v] - dst[u];
112               if (nxt[u] != e) {
113                   heap* p = new heap;
114                   fill(p->chd, p->chd + 4, nullNd);
115                   p->dep = 1;
116                   p->edge = e;
117                   V.push_back(p);
118               }
119           }
120       }
121       if (V.empty()) continue;

```

```

121     make_heap(V.begin(), V.end(), cmp);
122 #define L(X) ((X << 1) + 1)
123 #define R(X) ((X << 1) + 2)
124     for (size_t i = 0; i < V.size(); i++) {
125         if (L(i) < V.size())
126             V[i]->chd[2] = V[L(i)];
127         else
128             V[i]->chd[2] = nullNd;
129         if (R(i) < V.size())
130             V[i]->chd[3] = V[R(i)];
131         else
132             V[i]->chd[3] = nullNd;
133     }
134     head[u] = merge(head[u], V.front());
135 }
136 vector<ll> ans;
137 void first_K() {
138     ans.clear();
139     priority_queue<node> Q;
140     if (dst[s] == -1) return;
141     ans.push_back(dst[s]);
142     if (head[s] != nullNd)
143         Q.push(node(head[s], dst[s] + head[s]->edge
144             ->d));
145     for (int _ = 1; _ < k and not Q.empty(); _++) {
146         node p = Q.top(), q;
147         Q.pop();
148         ans.push_back(p.d);
149         if (head[p.H->edge->v] != nullNd) {
150             q.H = head[p.H->edge->v];
151             q.d = p.d + q.H->edge->d;
152             Q.push(q);
153         }
154         for (int i = 0; i < 4; i++)
155             if (p.H->chd[i] != nullNd) {
156                 q.H = p.H->chd[i];
157                 q.d = p.d - p.H->edge->d + p.H->chd
158                     [i]->edge->d;
159                 Q.push(q);
160             }
161     }
162     void solve() { // ans[i] stores the i-th shortest
163         path
164         dijkstra();
165         build();
166         first_K(); // ans.size() might less than k
167     }
} solver;

```

## 5.12 System of Difference Constraints

```

1 vector<vector<pair<int, ll>>> G;
2 void add(int u, int v, ll w) {
3     G[u].emplace_back(make_pair(v, w));
4 }

```

- $x_u - x_v \leq c \Rightarrow \text{add}(v, u, c)$
- $x_u - x_v \geq c \Rightarrow \text{add}(u, v, -c)$
- $x_u - x_v = c \Rightarrow \text{add}(v, u, c), \text{add}(u, v, -c)$
- $x_u \geq c \Rightarrow \text{add super vertex } x_0 = 0, \text{then } x_u - x_0 \geq c \Rightarrow \text{add}(u, 0, -c)$
- Don't forget non-negative constraints for every variable if specified implicitly.

- Interval sum  $\Rightarrow$  Use prefix sum to transform into differential constraints. Don't forget  $S_{i+1} - S_i \geq 0$  if  $x_i$  needs to be non-negative.

- $\frac{x_u}{x_v} \leq c \Rightarrow \log x_u - \log x_v \leq \log c$

## 6 String

### 6.1 Aho Corasick

```

1 struct ACautomata {
2     struct Node {
3         int cnt; // 停在此節點的數量
4         Node *go[26], *fail, *dic;
5         // 子節點 fail 指標 最近的模式結尾
6         Node() {
7             cnt = 0;
8             fail = 0;
9             dic = 0;
10            memset(go, 0, sizeof(go));
11        }
12    } pool[1048576], *root;
13    int nMem;
14    Node *new_Node() {
15        pool[nMem] = Node();
16        return &pool[nMem++];
17    }
18    void init() {
19        nMem = 0;
20        root = new_Node();
21    }
22    void add(const string &str) { insert(root, str, 0); }
23    void insert(Node *cur, const string &str, int pos)
24    {
25        for (int i = pos; i < str.size(); i++) {
26            if (!cur->go[str[i] - 'a'])
27                cur->go[str[i] - 'a'] = new_Node();
28            cur = cur->go[str[i] - 'a'];
29        }
30        cur->cnt++;
31    }
32    void make_fail() { // 全部 add 完做
33        queue<Node *> que;
34        que.push(root);
35        while (!que.empty()) {
36            Node *fr = que.front();
37            que.pop();
38            for (int i = 0; i < 26; i++) {
39                if (fr->go[i]) {
40                    Node *ptr = fr->fail;
41                    while (ptr && !ptr->go[i]) ptr =
42                        ptr->fail;
43                    fr->go[i]->fail = ptr = (ptr ? ptr
44                        ->go[i] : root);
45                    fr->go[i]->dic = (ptr->cnt ? ptr :
46                        ptr->dic);
47                    que.push(fr->go[i]);
48                }
49            }
50        }
51    }
52    // 出現過不同string的總數
53    int query_unique(const string& text) {
54        Node* p = root;
55        int ans = 0;
56        for(char ch : text) {
57            int i = ch - 'a';
58            while(p && !p->go[i]) p = p->fail;
59            p = p ? p->go[i] : root;
60            if(p->cnt) {ans += p->cnt, p->cnt = 0;}
61            for(Node* t = p->dic; t; t = t->dic) if(t->
62                cnt) {
63                    ans += t->cnt; t->cnt = 0;
64                }
65        }
66        return ans;
67    }
} AC;

```

### 6.2 KMP

```

1 vector<int> f;
2 // 沒匹配到可以退回哪裡
3 void buildFailFunction(string &s) {
4     f.resize(s.size(), -1);
5     for (int i = 1; i < s.size(); i++) {

```

```

6     int now = f[i - 1];
7     while (now != -1 and s[now + 1] != s[i]) now =
8         f[now];
9     if (s[now + 1] == s[i]) f[i] = now + 1;
10    }
11
12 void KMPmatching(string &a, string &b) {
13     for (int i = 0, now = -1; i < a.size(); i++) {
14         while (a[i] != b[now + 1] and now != -1) now =
15             f[now];
16         if (a[i] == b[now + 1]) now++;
17         if (now + 1 == b.size()) {
18             cout << "found a match start at position "
19                 << i - now << endl;
20             now = f[now];
21         }
22     }
23
24     int mx = m[i];
25     ptr = i;
26     }
27     for (int i = ptr - mx; i <= ptr + mx; i++)
28         if (s[i] != '.') cout << s[i];
29     cout << endl;
30 }
31
32
33
34
35 }
```

## 6.3 Z Value

```
1 string is, it, s;
2 // is: 被搜尋 it: 要找的
3 int n;
4 vector<int> z;
5 // 計算每個位置 i 開始的字串，和 s 的共農前綴長度
6 void init() {
7     cin >> is >> it;
8     s = it + '0' + is;
9     n = (int)s.size();
10    z.resize(n, 0);
11 }
12 void solve() {
13     int ans = 0;
14     z[0] = n;
15     for (int i = 1, l = 0, r = 0; i < n; i++) {
16         if (i <= r) z[i] = min(z[i - 1], r - i + 1);
17         while (i + z[i] < n && s[z[i]] == s[i + z[i]]) {
18             z[i]++;
19         }
20         if (i + z[i] - 1 > r) l = i, r = i + z[i];
21         if (z[i] == (int)it.size()) ans++;
22     }
23     cout << ans << endl;
24 }
```

## 6.4 Manacher

```

1 // 找最長回文
2 int n;
3 string s, s;
4 vector<int> m;
5 void manacher() {
6     s.clear();
7     s.resize(2 * n + 1, '.');
8     for (int i = 0, j = 1; i < n; i++, j += 2) s[j] = s[i];
9     m.clear();
10    m.resize(2 * n + 1, 0);
11    // m[i] := max k such that s[i-k, i+k] is
12    // palindrome
13    int mx = 0, mxk = 0;
14    for (int i = 1; i < 2 * n + 1; i++) {
15        if (mx - (i - mx) >= 0) m[i] = min(m[mx - (i - mx)], mx + mxk - i);
16        while (0 <= i - m[i] - 1 && i + m[i] + 1 < 2 * n + 1 &&
17              s[i - m[i] - 1] == s[i + m[i] + 1]) m[i]++;
18        if (i + m[i] > mx + mxk) mx = i, mxk = m[i];
19    }
20    void init() {
21        cin >> s;
22        n = (int)s.size();
23    }
24    void solve() {
25        manacher();
26        int mx = 0, ptr = 0;
27        for (int i = 0; i < 2 * n + 1; i++) {
28            if (mx < m[i]) {
29                _ = true;
30                end = true;
31                for (int i = 0; i < n; i++) suf[i] = buc[0][i].S;
32                rk[suf[0]] = 0;
33                for (int i = 1; i < n; i++) {
34                    int dif = (buc[0][i].F != buc[0][i - 1].F);
35                    end &= dif;
36                    rk[suf[i]] = rk[suf[i - 1]] + dif;
37                }
38                return end;
39            }
40            void sa() {
41                for (int i = 0; i < n; i++)
42                    buc[0][i] = make_pair(make_pair(s[i], s[i]), i);
43                sort(buc[0].begin(), buc[0].end());
44                if (fill_suf()) return;
45                for (int k = 0; (1 << k) < n; k++) {
46                    for (int i = 0; i < n; i++)
47                        buc[0][i] = make_pair(make_pair(rk[i], rk[(i + (1 << k)) % n]), i);
48                    radix_sort();
49                    if (fill_suf()) return;
50                }
51            }
52            void LCP() {
53                int k = 0;
54                for (int i = 0; i < n - 1; i++) {
55                    if (rk[i] == 0) continue;
56                    int pi = rk[i];
57                    int j = suf[pi - 1];
58                    while (i + k < n && j + k < n && s[i + k]
59                          == s[j + k]) k++;
60                }
61            }
62        }
63    }

```

```

64     lcp[pi] = k;
65     k = max(k - 1, 0);
66   }
67 }
SuffixArray suffixarray;

```

## 6.6 Suffix Automaton

```

1 struct SAM {
2   struct State {
3     int next[26];
4     int link, len;
5     // suffix link, 指向最長真後綴所對應的狀態
6     // 該狀態代表的字串集合中的最長字串長度
7     State() : link(-1), len(0) { memset(next, -1,
8       sizeof(next)); }
9   };
10  vector<State> st;
11  int last;
12  vector<long long> occ; // 每個狀態的出現次數 (
13    endpos 個數)
14  vector<int> first_bkpos; // 出現在哪裡
15  SAM(int maxlen = 0) {
16    st.reserve(2 * maxlen + 5); st.push_back(State()
17      ());
18    last = 0;
19    occ.reserve(2 * maxlen + 5); occ.push_back(0);
20    first_bkpos.push_back(-1);
21  }
22  void extend(int c) {
23    int cur = (int)st.size();
24    st.push_back(State());
25    occ.push_back(0);
26    first_bkpos.push_back(0);
27    st[cur].len = st[last].len + 1;
28    first_bkpos[cur] = st[cur].len - 1;
29    int p = last;
30    while (p != -1 && st[p].next[c] == -1) {
31      st[p].next[c] = cur;
32      p = st[p].link;
33    }
34    if (p == -1) {
35      st[cur].link = 0;
36    } else {
37      int q = st[p].next[c];
38      if (st[p].len + 1 == st[q].len) {
39        st[cur].link = q;
40      } else {
41        int clone = (int)st.size();
42        st.push_back(st[q]);
43        first_bkpos.push_back(first_bkpos[q]);
44        occ.push_back(0);
45        st[clone].len = st[p].len + 1;
46        while (p != -1 && st[p].next[c] == q) {
47          st[p].next[c] = clone;
48          p = st[p].link;
49        }
50        st[q].link = st[cur].link = clone;
51      }
52    }
53    last = cur;
54    occ[cur] += 1;
55  }
56  void finalize_occ() {
57    int m = (int)st.size();
58    vector<int> order(m);
59    iota(order.begin(), order.end(), 0);
60    sort(order.begin(), order.end(), [&](int a, int
61      b){ return st[a].len > st[b].len; });
62    for (int v : order) {
63      int p = st[v].link;
64      if (p != -1) occ[p] += occ[v];
65    }
66  }
67 };

```

## 6.7 Minimum Rotation

```

1 // rotate(begin(s), begin(s)+minRotation(s), end(s))
2 // 找出字串的最小字典序旋轉
3 int minRotation(string s) {

```

```

4   int a = 0, n = s.size();
5   s += s;
6   for (int b = 0; b < n; b++) {
7     for (int k = 0; k < n; k++) {
8       if (a + k == b || s[a + k] < s[b + k]) {
9         b += max(0, k - 1);
10        break;
11      }
12      if (s[a + k] > s[b + k]) {
13        a = b;
14        break;
15      }
16    }
17  }
18  return a;

```

## 6.8 Lyndon Factorization

```

1 // Duval: 將字串唯一分解為字典序非遞增的 Lyndon 子字串
2 vector<string> duval(string const& s) {
3   int n = s.size();
4   int i = 0;
5   vector<string> factorization;
6   while (i < n) {
7     int j = i + 1, k = i;
8     while (j < n && s[k] <= s[j]) {
9       if (s[k] < s[j])
10         k = i;
11       else
12         k++;
13       j++;
14     }
15     while (i <= k) {
16       factorization.push_back(s.substr(i, j - k));
17       i += j - k;
18     }
19   }
20   return factorization; // O(n)
21 }

```

## 6.9 Rolling Hash

```

1 const ll C = 27;
2 inline int id(char c) { return c - 'a' + 1; }
3 struct RollingHash {
4   string s;
5   int n;
6   ll mod;
7   vector<ll> Cexp, hs;
8   RollingHash(string& _s, ll _mod) : s(_s), n((int)_s
9     .size()), mod(_mod) {
10     Cexp.assign(n, 0);
11     hs.assign(n, 0);
12     Cexp[0] = 1;
13     for (int i = 1; i < n; i++) {
14       Cexp[i] = Cexp[i - 1] * C;
15       if (Cexp[i] >= mod) Cexp[i] %= mod;
16     }
17     hs[0] = id(s[0]);
18     for (int i = 1; i < n; i++) {
19       hs[i] = hs[i - 1] * C + id(s[i]);
20       if (hs[i] >= mod) hs[i] %= mod;
21     }
22   }
23   inline ll query(int l, int r) {
24     ll res = hs[r] - (l ? hs[l - 1] * Cexp[r - l +
25       1] : 0);
26     res = (res % mod + mod) % mod;
27   }
28 };

```

## 6.10 Trie

```

1 pii a[N][26];
2
3 void build(string &s) {
4   static int idx = 0;
5   int n = s.size();
6   for (int i = 0, v = 0; i < n; i++) {

```

```

7     pii &now = a[v][s[i] - 'a'];
8     if (now.first != -1)
9         v = now.first;
10    else
11        v = now.first = ++idx;
12    if (i == n - 1)
13        now.second++;
14 }
15 }
```

## 7 Geometry

### 7.1 Basic Operations

```

1 // typedef long long T;
2 typedef long double T;
3 const long double eps = 1e-12;
4
5 short sgn(T x) {
6     if (abs(x) < eps) return 0;
7     return x < 0 ? -1 : 1;
8 }
9
10 struct Pt {
11     T x, y;
12     Pt(T _x = 0, T _y = 0) : x(_x), y(_y) {}
13     Pt operator+(Pt a) { return Pt(x + a.x, y + a.y); }
14     Pt operator-(Pt a) { return Pt(x - a.x, y - a.y); }
15     Pt operator*(T a) { return Pt(x * a, y * a); }
16     Pt operator/(T a) { return Pt(x / a, y / a); }
17     T operator*(Pt a) { return x * a.x + y * a.y; }
18     T operator^(Pt a) { return x * a.y - y * a.x; }
19     bool operator<(Pt a) { return x < a.x || (x == a.x
20         && y < a.y); }
21     // return sgn(x-a.x) < 0 || (sgn(x-a.x) == 0 && sgn
22     // (y-a.y) < 0); }
23     bool operator==(Pt a) { return sgn(x - a.x) == 0 &&
24         sgn(y - a.y) == 0; }
25 }
26
27 Pt mv(Pt a, Pt b) { return b - a; }
28 T len2(Pt a) { return a * a; }
29 T dis2(Pt a, Pt b) { return len2(b - a); }
30 Pt rotate(Pt u) { return {-u.y, u.x}; }
31 Pt unit(Pt x) { return x / sqrtl(x * x); }
32 short ori(Pt a, Pt b) { return ((a ^ b) > 0) - ((a ^ b)
33     < 0); }
34 bool onseg(Pt p, Pt l1, Pt l2) {
35     Pt a = mv(p, l1), b = mv(p, l2);
36     return ((a ^ b) == 0) && ((a * b) <= 0);
37 }
38 inline T cross(const Pt &a, const Pt &b, const Pt &c) {
39     return (b.x - a.x) * (c.y - a.y)
40         - (b.y - a.y) * (c.x - a.x);
41 }
42 long double polar_angle(Pt ori, Pt pt){
43     return atan2(pt.y - ori.y, pt.x - ori.x);
44 }
45 // slope to degree atan(Slope) * 180.0 / acos(-1.0);
46 bool argcmp(Pt u, Pt v) {
47     auto half = []()>>const Pt& p) {
48         return p.y > 0 || (p.y == 0 && p.x >= 0);
49     };
50     if (half(u) != half(v)) return half(u) < half(v);
51     return sgn(u ^ v) > 0;
52 }
53 int ori(Pt& o, Pt& a, Pt& b) {
54     return sgn((a - o) ^ (b - o));
55 }
56 struct Line {
57     Pt a, b;
58     Pt dir() { return b - a; }
59 };
60 int PtSide(Pt p, Line L) {
61     return sgn(ori(L.a, L.b, p)); // for int
62     return sgn(ori(L.a, L.b, p) / sqrt(len2(L.a - L.b))
63         );
64 }
```

```

62     return PtSide(p, L) == 0 and sgn((p - L.a) * (p - L
63         .b)) <= 0;
64 }
65 Pt proj(Pt& p, Line& l) {
66     Pt d = l.b - l.a;
67     T d2 = len2(d);
68     if (sgn(d2) == 0) return l.a;
69     T t = ((p - l.a) * d) / d2;
70     return l.a + d * t;
71 }
72 struct Cir {
73     Pt o;
74     T r;
75 };
76 bool disjunct(Cir a, Cir b) {
77     return sgn(sqrtl(len2(a.o - b.o)) - a.r - b.r) >=
78         0;
79 }
80 bool contain(Cir a, Cir b) {
81     return sgn(a.r - b.r - sqrtl(len2(a.o - b.o))) >=
82         0;
83 }
```

### 7.2 Sort by Angle

```

1 int ud(Pt a) { // up or down half plane
2     if (a.y > 0) return 0;
3     if (a.y < 0) return 1;
4     return (a.x >= 0 ? 0 : 1);
5 }
6 sort(pts.begin(), pts.end(), [&](const Pt& a, const Pt&
7     b) {
8     if (ud(a) != ud(b)) return ud(a) < ud(b);
9     return (a ^ b) > 0;
10 });
11 }
```

### 7.3 Intersection

```

1 bool line_intersect_check(Pt p1, Pt p2, Pt q1, Pt q2) {
2     if (onseg(p1, q1, q2) || onseg(p2, q1, q2) || onseg
3         (q1, p1, p2) || onseg(q2, p1, p2)) return true;
4     Pt p = mv(p1, p2), q = mv(q1, q2);
5     return (ori(p, mv(p1, q1)) * ori(p, mv(p1, q2)) <
6         0) && (ori(q, mv(q1, p1)) * ori(q, mv(q1, p2))
7         < 0);
8 }
9 // long double
10 Pt line_intersect(Pt a1, Pt a2, Pt b1, Pt b2) {
11     Pt da = mv(a1, a2), db = mv(b1, b2);
12     T det = da ^ db;
13     if (sgn(det) == 0) { // parallel
14         // return Pt(NAN, NAN);
15     }
16     T t = ((b1 - a1) ^ db) / det;
17     return a1 + da * t;
18 }
19 vector<Pt> CircleInter(Cir a, Cir b) {
20     double d2 = len2(a.o - b.o), d = sqrt(d2);
21     if (d < max(a.r, b.r) - min(a.r, b.r) || d > a.r +
22         b.r) return {};
23     Pt u = (a.o + b.o) / 2 + (a.o - b.o) * ((b.r * b.r
24         - a.r * a.r) / (2 * d2));
25     double A = sqrt((a.r + b.r + d) * (a.r - b.r + d) *
26         (a.r + b.r - d) * (-a.r + b.r + d));
27     Pt v = rotate(b.o - a.o) * A / (2 * d2);
28     if (sgn(v.x) == 0 and sgn(v.y) == 0) return {u};
29     return {u - v, u + v}; // counter clockwise of a
30 }
31 vector<Pt> CircleLineInter(Cir c, Line l) {
32     Pt H = proj(c.o, l);
33     Pt dir = unit(l.b - l.a);
34     T h = sqrtl(len2(H - c.o));
35     if (sgn(h - c.r) > 0) return {};
36     T d = sqrtl(max((T)0, c.r * c.r - h * h));
37     if (sgn(d) == 0) return {H};
38     return {H - dir * d, H + dir * d};
39 }
40
41 // 2 * area
42 }
```

### 7.4 Polygon Area

```

1 T dbPoly_area(vector<Pt>& e) {
2     T res = 0;
3     int sz = e.size();
4     for (int i = 0; i < sz; i++) {
5         res += e[i] ^ e[(i + 1) % sz];
6     }
7     return abs(res);
8 }
```

## 7.5 Convex Hull

```

1 vector<Pt> convexHull(vector<Pt> pts) {
2     vector<Pt> hull;
3     sort(pts.begin(), pts.end());
4     for (int i = 0; i < 2; i++) {
5         int b = hull.size();
6         for (auto ei : pts) {
7             while (hull.size() - b >= 2 && ori(mv(hull[7
8                 hull.size() - 2], hull.back()), mv(hull[8
9                     [hull.size() - 2], ei)) == -1) {
10                     hull.pop_back();
11                 }
12                 hull.emplace_back(ei);
13             }
14             hull.pop_back();
15             reverse(pts.begin(), pts.end());
16         }
17         return hull;
18 }
```

## 7.6 Point In Convex

```

1 bool point_in_convex(const vector<Pt> &C, Pt p, bool
2 strict = true) {
3     // only works when no three point are collinear
4     int n = C.size();
5     int a = 1, b = n - 1, r = !strict;
6     if (n == 0) return false;
7     if (n < 3) return r && onseg(p, C[0], C.back());
8     if (ori(mv(C[0], C[a]), mv(C[0], C[b])) > 0) swap(a
9         , b);
10    if (ori(mv(C[0], C[a]), mv(C[0], p)) >= r || ori(mv
11        (C[0], C[b]), mv(C[0], p)) <= -r) return false;
12    while (abs(a - b) > 1) {
13        int c = (a + b) / 2;
14        if (ori(mv(C[0], C[c]), mv(C[0], p)) > 0) b = c
15            ;
16        else a = c;
17    }
18    return ori(mv(C[a], C[b]), mv(C[a], p)) < r;
19 }
```

## 7.7 Point Segment Distance

```

1 double point_segment_dist(Pt q0, Pt q1, Pt p) {
2     if (q0 == q1) {
3         double dx = double(p.x - q0.x);
4         double dy = double(p.y - q0.y);
5         return sqrt(dx * dx + dy * dy);
6     }
7     T d1 = (q1 - q0) * (p - q0);
8     T d2 = (q0 - q1) * (p - q1);
9     if (d1 >= 0 && d2 >= 0) {
10        double area = fabs(double((q1 - q0) ^ (p - q0)))
11            );
12        double base = sqrt(double(dis2(q0, q1)));
13        return area / base;
14    }
15    double dx0 = double(p.x - q0.x), dy0 = double(p.y -
16        q0.y);
17    double dx1 = double(p.x - q1.x), dy1 = double(p.y -
18        q1.y);
19    return min(sqrt(dx0 * dx0 + dy0 * dy0), sqrt(dx1 * 
20        dx1 + dy1 * dy1));
21 }
```

## 7.8 Point in Polygon

```

1 short inPoly(vector<Pt>& pts, Pt p) {
2     // 0=Bound 1=In -1=Out
3     int n = pts.size();
```

```

4     for (int i = 0; i < pts.size(); i++) if (onseg(p,
5         pts[i], pts[(i + 1) % n])) return 0;
6     int cnt = 0;
7     for (int i = 0; i < pts.size(); i++) if (
8         line_intersect_check(p, Pt(p.x + 1, p.y + 2e9),
9             pts[i], pts[(i + 1) % n])) cnt ^= 1;
10    return (cnt ? 1 : -1);
11 }
```

## 7.9 Minimum Euclidean Distance

```

1 long long Min_Euclidean_Dist(vector<Pt> &pts) {
2     sort(pts.begin(), pts.end());
3     set<pair<long long, long long>> s;
4     s.insert({pts[0].y, pts[0].x});
5     long long l = 0, best = LLONG_MAX;
6     for (int i = 1; i < (int)pts.size(); i++) {
7         Pt now = pts[i];
8         long long lim = (long long)ceil(sqrtl((long
9             double)best));
10        while (now.x - pts[1].x > lim) {
11            s.erase({pts[1].y, pts[1].x}); l++;
12        }
13        auto low = s.lower_bound({now.y - lim,
14            LLONG_MIN});
15        auto high = s.upper_bound({now.y + lim,
16            LLONG_MAX});
17        for (auto it = low; it != high; it++) {
18            long long dy = it->first - now.y;
19            long long dx = it->second - now.x;
20            best = min(best, dx * dx + dy * dy);
21        }
22        s.insert({now.y, now.x});
23    }
24    return best;
25 }
```

## 7.10 Minkowski Sum

```

1 void reorder(vector <Pt> &P) {
2     rotate(P.begin(), min_element(P.begin(), P.end(),
3         [&](Pt a, Pt b) { return make_pair(a.y, a.x) <
4             make_pair(b.y, b.x); }), P.end());
5 }
6 vector <Pt> Minkowski(vector <Pt> P, vector <Pt> Q) {
7     // P, Q: convex polygon
8     reorder(P), reorder(Q);
9     int n = P.size(), m = Q.size();
10    P.push_back(P[0]), P.push_back(P[1]), Q.push_back(Q
11        [0]), Q.push_back(Q[1]);
12    vector <Pt> ans;
13    for (int i = 0, j = 0; i < n || j < m; ) {
14        ans.push_back(P[i] + Q[j]);
15        auto val = (P[i + 1] - P[i]) ^ (Q[j + 1] - Q[j]);
16        if (val >= 0) i++;
17        if (val <= 0) j++;
18    }
19    return ans;
20 }
```

## 7.11 Lower Concave Hull

```

1 struct Line {
2     mutable ll m, b, p;
3     bool operator<(const Line& o) const { return m < o.m;
4         }
5     bool operator<(ll x) const { return p < x; }
6 };
7 struct LineContainer : multiset<Line, less<>> {
8     // (for doubles, use inf = 1./0., div(a,b) = a/b)
9     const ll inf = LLONG_MAX;
10    ll div(ll a, ll b) { // floored division
11        return a / b - ((a ^ b) < 0 && a % b); }
12    bool intersect(iterator x, iterator y) {
13        if (y == end()) { x->p = inf; return false; }
14        if (x->m == y->m) x->p = x->b > y->b ? inf : -inf;
15        else x->p = div(y->b - x->b, x->m - y->m);
16        return x->p >= y->p;
17    }
18    void add(ll m, ll b) {
```

```

19    auto z = insert({m, b, 0}), y = z++, x = y;
20    while (isect(y, z)) z = erase(z);
21    if (x != begin() && isect(--x, y)) isect(x, y =
22        erase(y));
23    while ((y = x) != begin() && (--x)->p >= y->p)
24        isect(x, erase(y));
25    }
26    ll query(ll x) {
27        assert(!empty());
28        auto l = *lower_bound(x);
29        return l.m * x + l.b;
30    }

```

## 7.12 Pick's Theorem

Consider a polygon which vertices are all lattice points.

Let  $i$  = number of points inside the polygon.

Let  $b$  = number of points on the boundary of the polygon.

Then we have the following formula:

$$\text{Area} = i + \frac{b}{2} - 1$$

## 7.13 Rotating SweepLine

```

1 double cross(const Pt &a, const Pt &b) {
2     return a.x*b.y - a.y*b.x;
3 }
4 int rotatingCalipers(const vector<Pt>& hull) {
5     int m = hull.size();
6     if (m < 2) return 0;
7     int j = 1;
8     T maxd = 0;
9     for (int i = 0; i < m; ++i) {
10         int ni = (i + 1) % m;
11         while (abs(cross({hull[ni].x - hull[i].x, hull[ni].y -
12             hull[i].y, {hull[(j+1)%m].x - hull[j].x, hull[(j+1)%m].y -
13             hull[j].y, {hull[i].x - hull[i].y}}) > abs(
14             (cross({hull[ni].x - hull[i].x, hull[ni].y -
15                 hull[i].y}, {hull[j].x - hull[i].x, hull[j].y -
16                 hull[i].y})) {
17             j = (j + 1) % m;
18         }
19         maxd = max(maxd, dis2(hull[i], hull[j]));
20         maxd = max(maxd, dis2(hull[ni], hull[j]));
21     }
22     return maxd; // TODO
23 }

```

## 7.14 Half Plane Intersection

```

1 bool cover(Line& L, Line& P, Line& Q) {
2     long double u = (Q.a - P.a) ^ Q.dir();
3     long double v = P.dir() ^ Q.dir();
4     long double x = P.dir().x * u + (P.a - L.a).x * v;
5     long double y = P.dir().y * u + (P.a - L.a).y * v;
6     return sgn(x * L.dir().y - y * L.dir().x) * sgn(v)
7         >= 0;
8 }
9 vector<Line> HPI(vector<Line> P) {
10    sort(P.begin(), P.end(), [&](Line& l, Line& m) {
11        if (argcmp(l.dir(), m.dir())) return true;
12        if (argcmp(m.dir(), l.dir())) return false;
13        return ori(m.a, m.b, l.a) > 0;
14    });
15    int l = 0, r = -1;
16    for (size_t i = 0; i < P.size(); ++i) {
17        if (i && !argcmp(P[i - 1].dir(), P[i].dir()))
18            continue;
19        while (l < r && cover(P[i], P[r - 1], P[r])) --r;
20        r;
21        while (l < r && cover(P[i], P[l], P[l + 1])) ++
22            l;
23        P[++r] = P[i];
24    }
25    while (l < r && cover(P[l], P[r - 1], P[r])) --r;
26    while (l < r && cover(P[r], P[l], P[l + 1])) ++l;
27 }

```

```

25    if (r - 1 <= 1 || !argcmp(P[1].dir(), P[r].dir()))
26        return {};
27    if (cover(P[1 + 1], P[1], P[r])) return {};
28
29    return vector<Line>(P.begin() + 1, P.begin() + r + 1);
30 }

```

## 7.15 Minimum Enclosing Circle

```

1 const int INF = 1e9;
2 Pt circumcenter(Pt A, Pt B, Pt C) {
3     // a1(x-A.x) + b1(y-A.y) = c1
4     // a2(x-A.x) + b2(y-A.y) = c2
5     // solve using Cramer's rule
6     T a1 = B.x - A.x, b1 = B.y - A.y, c1 = dis2(A, B) /
7         2.0;
8     T a2 = C.x - A.x, b2 = C.y - A.y, c2 = dis2(A, C) /
9         2.0;
10    T D = Pt(a1, b1) ^ Pt(a2, b2);
11    T Dx = Pt(c1, b1) ^ Pt(c2, b2);
12    T Dy = Pt(a1, c1) ^ Pt(a2, c2);
13    if (D == 0) return Pt(-INF, -INF);
14    return A + Pt(Dx / D, Dy / D);
15 }
16 Pt center;
17 T r2;
18 void minEncloseCircle(vector<Pt> pts) {
19     mt19937 gen(chrono::steady_clock::now());
20     time_since_epoch().count());
21     shuffle(pts.begin(), pts.end(), gen);
22     center = pts[0], r2 = 0;
23
24     for (int i = 0; i < pts.size(); i++) {
25         if (dis2(center, pts[i]) <= r2) continue;
26         center = pts[i], r2 = 0;
27         for (int j = 0; j < i; j++) {
28             if (dis2(center, pts[j]) <= r2) continue;
29             center = (pts[i] + pts[j]) / 2.0;
30             r2 = dis2(center, pts[i]);
31             for (int k = 0; k < j; k++) {
32                 if (dis2(center, pts[k]) <= r2)
33                     continue;
34                 center = circumcenter(pts[i], pts[j],
35                     pts[k]);
36                 r2 = dis2(center, pts[i]);
37             }
38         }
39     }
40 }

```

## 7.16 Union of Circles

```

1 // Area[i] : area covered by at least i circle
2 vector<T> CircleUnion(const vector<Cir> &C) {
3     const int n = C.size();
4     vector<T> Area(n + 1);
5     auto check = [&](int i, int j) {
6         if (!contain(C[i], C[j]))
7             return false;
8         return sgn(C[i].r - C[j].r) > 0 or (sgn(C[i].r
9             - C[j].r) == 0 and i < j);
10    };
11    struct Teve {
12        double ang; int add; Pt p;
13        bool operator<(const Teve &b) { return ang < b.
14            ang; }
15    };
16    auto ang = [&](Pt p) { return atan2(p.y, p.x); };
17    for (int i = 0; i < n; i++) {
18        int cov = 1;
19        vector<Teve> event;
20        for (int j = 0; j < n; j++) if (i != j) {
21            if (check(j, i)) cov++;
22            else if (!check(i, j) and !disjunct(C[i], C
23                [j])) {
24                auto I = CircleInter(C[i], C[j]);
25                assert(I.size() == 2);
26                double a1 = ang(I[0] - C[i].o), a2 =
27                    ang(I[1] - C[i].o);
28                event.push_back({a1, 1, I[0]});
29                event.push_back({a2, -1, I[1]});
30            }
31        }
32    }
33 }

```

```

26         if (a1 > a2) cov++;
27     }
28
29     if (event.empty()) {
30         Area[cov] += acos(-1) * C[i].r * C[i].r;
31         continue;
32     }
33     sort(event.begin(), event.end());
34     event.push_back(event[0]);
35     for (int j = 0; j + 1 < event.size(); j++) {
36         cov += event[j].add;
37         Area[cov] += (event[j].p ^ event[j + 1].p)
38             / 2.;
39         double theta = event[j + 1].ang - event[j].ang;
40         if (theta < 0) theta += 2 * acos(-1);
41         Area[cov] += (theta - sin(theta)) * C[i].r
42             * C[i].r / 2;
43     }
44 }
45 return Area;
}

```

## 7.17 Area Of Circle Polygon

```

1 double AreaOfCirclePoly(Cir C, vector<Pt> &P) {
2     auto arg = [&](Pt p, Pt q) { return atan2l(p ^ q, p
3         * q); };
4     double r2 = (double)(C.r * C.r / 2);
5     auto tri = [&](Pt p, Pt q) {
6         Pt d = q - p;
7         T a = (d * p) / (d * d);
8         T b = ((p * p) - C.r * C.r) / (d * d);
9         T det = a * a - b;
10        if (det <= 0) return (double)(arg(p, q) * r2);
11        T s = max((T)0.0L, -a - sqrtl(det));
12        T t = min((T)1.0L, -a + sqrtl(det));
13        if (t < 0 || 1 <= s) return (double)(arg(p, q)
14            * r2);
15        Pt u = p + d * s, v = p + d * t;
16        return (double)(arg(p, u) * r2 + (u ^ v) / 2 +
17            arg(v, q) * r2);
18    };
19    long double sum = 0.0L;
20    for (int i = 0; i < (int)P.size(); i++)
21        sum += tri(P[i] - C.o, P[(i + 1) % P.size()] -
22            C.o);
23    return (double)fabsl(sum);
}

```

## 7.18 3D Point

```

1 struct Pt {
2     double x, y, z;
3     Pt(double _x = 0, double _y = 0, double _z = 0): x(_x
4         ), y(_y), z(_z){}
5     Pt operator + (const Pt &o) const
6     { return Pt(x + o.x, y + o.y, z + o.z); }
7     Pt operator - (const Pt &o) const
8     { return Pt(x - o.x, y - o.y, z - o.z); }
9     Pt operator * (const double &k) const
10    { return Pt(x * k, y * k, z * k); }
11    Pt operator / (const double &k) const
12    { return Pt(x / k, y / k, z / k); }
13    double operator * (const Pt &o) const
14    { return x * o.x + y * o.y + z * o.z; }
15    Pt operator ^ (const Pt &o) const
16    { return {Pt(y * o.z - z * o.y, z * o.x - x * o.z, x
17        * o.y - y * o.x)}; }
18 };
19 double abs2(Pt o) { return o * o; }
20 double abs(Pt o) { return sqrt(abs2(o)); }
21 Pt cross3(Pt a, Pt b, Pt c)
22 { return (b - a) ^ (c - a); }
23 double area(Pt a, Pt b, Pt c)
24 { return abs(cross3(a, b, c)); }
25 double volume(Pt a, Pt b, Pt c, Pt d)
26 { return cross3(a, b, c) * (d - a); }
27 bool coplaner(Pt a, Pt b, Pt c, Pt d)
28 { return sign(volume(a, b, c, d)) == 0; }
29 Pt proj(Pt o, Pt a, Pt b, Pt c) // o proj to plane abc
30 { Pt n = cross3(a, b, c);
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
}

```

```

29     return o - n * ((o - a) * (n / abs2(n)));
30 Pt line_plane_intersect(Pt u, Pt v, Pt a, Pt b, Pt c) {
31     // intersection of line uv and plane abc
32     Pt n = cross3(a, b, c);
33     double s = n * (u - v);
34     if (sign(s) == 0) return {-1, -1, -1}; // not found
35     return v + (u - v) * ((n * (a - v)) / s);
36 Pt rotateAroundAxis(Pt v, Pt axis, double theta) {
37     axis = axis / abs(axis); // axis must be unit
38     vector
39     double cosT = cos(theta);
40     double sinT = sin(theta);
41     Pt term1 = v * cosT;
42     Pt term2 = (axis ^ v) * sinT;
43     Pt term3 = axis * ((axis * v) * (1 - cosT));
44     return term1 + term2 + term3;
}

```

## 8 Number Theory

### 8.1 FFT

```

1 typedef complex<double> cp;
2
3 const double pi = acos(-1);
4 const int NN = 131072;
5
6 struct FastFourierTransform {
7     /*
8      * Iterative Fast Fourier Transform
9      * How this works? Look at this
10     0th recursion 0(000) 1(001) 2(010)
11     3(011) 4(100) 5(101) 6(110)
12     7(111)
13     1th recursion 0(000) 2(010) 4(100)
14     6(110) / 1(011) 3(011) 5(101)
15     7(111)
16     2th recursion 0(000) 4(100) / 2(010)
17     6(110) / 1(011) 5(101) / 3(011)
18     7(111)
19     3th recursion 0(000) / 4(100) / 2(010) /
20     6(110) / 1(011) / 5(101) / 3(011) /
21     7(111)
22     All the bits are reversed => We can save
23     the reverse of the numbers in an array!
24 */
25     int n, rev[NN];
26     cp omega[NN], iomega[NN];
27     void init(int n_) {
28         n = n_;
29         for (int i = 0; i < n_; i++) {
30             // Calculate the nth roots of unity
31             omega[i] = cp(cos(2 * pi * i / n_), sin(2 *
32                 pi * i / n_));
33             iomega[i] = conj(omega[i]);
34         }
35         int k = __lg(n_);
36         for (int i = 0; i < n_; i++) {
37             int t = 0;
38             for (int j = 0; j < k; j++) {
39                 if (i & (1 << j)) t |= (1 << (k - j -
40                     1));
41             }
42             rev[i] = t;
43         }
44     }
45
46     void transform(vector<cp> &a, cp *xomega) {
47         for (int i = 0; i < n; i++)
48             if (i < rev[i]) swap(a[i], a[rev[i]]);
49         for (int len = 2; len <= n; len <= 1) {
50             int mid = len >> 1;
51             int r = n / len;
52             for (int j = 0; j < n; j += len)
53                 for (int i = 0; i < mid; i++)
54                     cp tmp = xomega[r * i] * a[j + mid
55                         + i];
56                     a[j + mid + i] = a[j + i] - tmp;
57                     a[j + i] = a[j + i] + tmp;
58         }
59     }
}

```

```

48 }
49
50 void fft(vector<cp> &a) { transform(a, omega); }
51 void ifft(vector<cp> &a) {
52     transform(a, iomega);
53     for (int i = 0; i < n; i++) a[i] /= n;
54 }
55 } FFT;
56
57 const int MAXN = 262144;
58 // (must be 2^k)
59 // 262144, 524288, 1048576, 2097152, 4194304
60 // before any usage, run pre_fft() first
61 typedef long double ld;
62 typedef complex<ld> cplx; // real(), imag()
63 const ld PI = acosl(-1);
64 const cplx I(0, 1);
65 cplx omega[MAXN + 1];
66 void pre_fft() {
67     for (int i = 0; i <= MAXN; i++) {
68         omega[i] = exp(i * 2 * PI / MAXN * I);
69     }
70 }
71 // n must be 2^k
72 void fft(int n, cplx a[], bool inv = false) {
73     int basic = MAXN / n;
74     int theta = basic;
75     for (int m = n; m >= 2; m >>= 1) {
76         int mh = m >> 1;
77         for (int i = 0; i < mh; i++) {
78             cplx w = omega[inv ? MAXN - (i * theta % MAXN) : i * theta % MAXN];
79             for (int j = i; j < n; j += m) {
80                 int k = j + mh;
81                 cplx x = a[j] - a[k];
82                 a[j] += a[k];
83                 a[k] = w * x;
84             }
85         }
86         theta = (theta * 2) % MAXN;
87     }
88     int i = 0;
89     for (int j = 1; j < n - 1; j++) {
90         for (int k = n >> 1; k > (i ^= k); k >>= 1);
91         if (j < i) swap(a[i], a[j]);
92     }
93     if (inv) {
94         for (int i = 0; i < n; i++) a[i] /= n;
95     }
96 }
97 cplx arr[MAXN + 1];
98 inline void mul(int _n, long long a[], int _m, long long b[], long long ans[]) {
99     int n = 1, sum = _n + _m - 1;
100    while (n < sum) n <<= 1;
101    for (int i = 0; i < n; i++) {
102        double x = (i < _n ? a[i] : 0), y = (i < _m ? b[i] : 0);
103        arr[i] = complex<double>(x + y, x - y);
104    }
105    fft(n, arr);
106    for (int i = 0; i < n; i++) arr[i] = arr[i] * arr[i];
107    fft(n, arr, true);
108    for (int i = 0; i < sum; i++) ans[i] = (long long int)(arr[i].real() / 4 + 0.5);
109 }
110
111 long long a[MAXN];
112 long long b[MAXN];
113 long long ans[MAXN];
114 int a_length;
115 int b_length;

```

## 8.2 Pollard's rho

```

1 ll add(ll x, ll y, ll p) {
2     return (x + y) % p;
3 }
4 ll qMul(ll x, ll y, ll mod) {
5     ll ret = x * y - (ll((long double)x / mod * y) *
       mod;

```

```

6         return ret < 0 ? ret + mod : ret;
7 }
8 ll f(ll x, ll mod) { return add(qMul(x, x, mod), 1, mod);
9     );
10 }
11 ll pollard_rho(ll n) {
12     if (!(n & 1)) return 2;
13     while (true) {
14         ll y = 2, x = rand() % (n - 1) + 1, res = 1;
15         for (int sz = 2; res == 1; sz *= 2) {
16             for (int i = 0; i < sz && res <= 1; i++) {
17                 x = f(x, n);
18                 res = __gcd(llabs(x - y), n);
19             }
20             y = x;
21         }
22         if (res != 0 && res != n) return res;
23     }
24 }
25 vector<ll> ret;
26 void fact(ll x) {
27     if (miller_rabin(x)) {
28         ret.push_back(x);
29         return;
30     }
31     ll f = pollard_rho(x);
32     fact(f);
33     fact(x / f);
34 }

```

## 8.3 Miller Rabin

```

1 // n < 4,759,123,141      3 : 2, 7, 61
2 // n < 1,122,004,669,633   4 : 2, 13, 23, 1662803
3 // n < 3,474,749,660,383   6 : primes <= 13
4 // n < 2^64                  7 :
5 // 2, 325, 9375, 28178, 450775, 9780504, 1795265022
6 bool witness(ll a, ll n, ll u, int t) {
7     if (!(a % n)) return 0;
8     ll x = mypow(a, u, n);
9     for (int i = 0; i < t; i++) {
10         ll nx = mul(x, x, n);
11         if (nx == 1 && x != 1 && x != n - 1) return 1;
12         x = nx;
13     }
14     return x != 1;
15 }
16 bool miller_rabin(ll n, int s = 100) {
17     // iterate s times of witness on n
18     // return 1 if prime, 0 otherwise
19     if (n < 2) return 0;
20     if (!(n & 1)) return n == 2;
21     ll u = n - 1;
22     int t = 0;
23     while (!(u & 1)) u >>= 1, t++;
24     while (s--) {
25         ll a = randll() % (n - 1) + 1;
26         if (witness(a, n, u, t)) return 0;
27     }
28     return 1;
29 }

```

## 8.4 Fast Power

Note:  $a^n \equiv a^{(n \bmod (p-1))} (mod p)$

## 8.5 Extend GCD

```

1 ll GCD;
2 pll extgcd(ll a, ll b) {
3     if (b == 0) {
4         GCD = a;
5         return pll{1, 0};
6     }
7     pll ans = extgcd(b, a % b);
8     return pll{ans.S, ans.F - a / b * ans.S};
9 }
10 pll bezout(ll a, ll b, ll c) {
11     bool negx = (a < 0), negy = (b < 0);
12     pll ans = extgcd(abs(a), abs(b));
13     if (c % GCD != 0) return pll{-LLINF, -LLINF};
14     return pll{ans.F * c / GCD * (negx ? -1 : 1),
15                ans.S * c / GCD * (negy ? -1 : 1)};
16 }

```

```

16 }
17 ll inv(ll a, ll p) {
18     if (p == 1) return -1;
19     pll ans = bezout(a % p, -p, 1);
20     if (ans == pll{-LLINF, -LLINF}) return -1;
21     return (ans.F % p + p) % p;
22 }
```

## 8.6 Mu + Phi

```

1 const int maxn = 1e6 + 5;
2 ll f[maxn];
3 vector<int> lpf, prime;
4 void build() {
5     lpf.clear();
6     lpf.resize(maxn, 1);
7     prime.clear();
8     f[1] = ... /* mu[1] = 1, phi[1] = 1 */
9     for (int i = 2; i < maxn; i++) {
10         if (lpf[i] == 1) {
11             lpf[i] = i;
12             prime.emplace_back(i);
13             f[i] = ... /* mu[i] = 1, phi[i] = i-1 */
14         }
15         for (auto& j : prime) {
16             if (i * j >= maxn) break;
17             lpf[i * j] = j;
18             if (i % j == 0)
19                 f[i * j] = ... /* 0, phi[i]*j */
20             else
21                 f[i * j] = ... /* -mu[i], phi[i]*phi[j] */
22             if (j >= lpf[i]) break;
23         }
24     }
25 }
```

## 8.7 Discrete Log

```

1 long long mod_pow(long long a, long long e, long long p) {
2     long long r = 1 % p;
3     while(e){
4         if(e & 1) r = (__int128)r * a % p;
5         a = (__int128)a * a % p;
6         e >>= 1;
7     }
8     return r;
9 }
10 long long mod_inv(long long a, long long p){
11     return mod_pow((a%p+p)%p, p-2, p);
12 }
13 // BSGS: solve a^x = y (mod p), gcd(a,p)=1, p prime,
14 // return minimal x>=0, or -1 if no solution
15 long long bsgs(long long a, long long y, long long p){
16     a%=p; y%=p;
17     if(y==1%p) return 0; // x=0
18     long long m = (long long)ceil(sqrt((long double)p));
19     // baby steps: a^j
20     unordered_map<long long, long long> table;
21     table.reserve(m*2);
22     long long cur = 1%p;
23     for(long long j=0;j<m;++j){
24         if(!table.count(cur)) table[cur]=j;
25         cur = (__int128)cur * a % p;
26     }
27     long long am = mod_pow(a, m, p);
28     long long am_inv = mod_inv(am, p);
29     long long gamma = y % p;
30     for(long long i=0;i<=m;++i){
31         auto it = table.find(gamma);
32         if(it != table.end()){
33             long long x = i*m + it->second;
34             return x;
35         }
36         gamma = (__int128)gamma * am_inv % p;
37     }
38     return -1;
39 }
```

## 8.8 sqrt mod

```

1 // the Jacobi symbol is a generalization of the
2 // Legendre symbol,
3 // such that the bottom doesn't need to be prime.
4 // (n/p) -> same as legendre
5 // (n/ab) = (n/a)(n/b)
6 // work with long long
7 int Jacobi(int a, int m) {
8     int s = 1;
9     for (; m > 1; ) {
10        a %= m;
11        if (a == 0) return 0;
12        const int r = __builtin_ctz(a);
13        if ((r & 1) && ((m + 2) & 4)) s = -s;
14        a >>= r;
15        if (a & m & 2) s = -s;
16        swap(a, m);
17    }
18    return s;
19 }
20 // solve x^2 = a (mod p)
21 // 0: a == 0
22 // -1: a isn't a quad res of p
23 // else: return X with X^2 % p == a
24 // doesn't work with long long
25 int QuadraticResidue(int a, int p) {
26     if (p == 2) return a & 1;
27     if (int jc = Jacobi(a, p); jc <= 0) return jc;
28     int b, d;
29     for (; ; ) {
30         b = rand() % p;
31         d = (1LL * b * b + p - a) % p;
32         if (Jacobi(d, p) == -1) break;
33     }
34     int f0 = b, f1 = 1, g0 = 1, g1 = 0, tmp;
35     for (int e = (1LL + p) >> 1; e; e >>= 1) {
36         if (e & 1) {
37             tmp = (1LL * g0 * f0 + 1LL * d * (1LL * g1
38             * f1 % p)) % p;
39             g1 = (1LL * g0 * f1 + 1LL * g1 * f0) % p;
40             g0 = tmp;
41         }
42         tmp = (1LL * f0 * f0 + 1LL * d * (1LL * f1 * f1
43             % p)) % p;
44         f1 = (2LL * f0 * f1) % p;
45         f0 = tmp;
46     }
47     return g0;
48 }
```

## 8.9 Primitive Root

```

1 unsigned long long primitiveRoot(ull p) {
2     auto fac = factor(p - 1);
3     sort(all(fac));
4     fac.erase(unique(all(fac)), fac.end());
5     auto test = [p, fac](ull x) {
6         for(ull d : fac)
7             if (modpow(x, (p - 1) / d, p) == 1)
8                 return false;
9         return true;
10    };
11    uniform_int_distribution<unsigned long long> unif
12        (1, p - 1);
13    unsigned long long root;
14    while(!test(root = unif(rng)));
15    return root;
16 }
```

## 8.10 LinearSieve

```

1 const int C = 1e7 + 2;
2 int mo[C], lp[C], phi[C], isp[C];
3 vector<int> prime;
4 void sieve() {
5     mo[1] = phi[1] = 1;
6     for(int i = 1; i < C; i++) lp[i] = 1;
7     for(int i = 2; i < C; i++) {
8         if(lp[i] == 1) {
9             lp[i] = i;
10            prime.push_back(i);
11        }
12    }
13 }
```

```

11     isp[i] = 1;
12     mo[i] = -1;
13     phi[i] = i - 1;
14   }
15   for(int p : prime) {
16     if(i * p >= C) break;
17     lp[i * p] = p;
18     if(i % p == 0) {
19       phi[p * i] = phi[i] * p;
20       break;
21     }
22     phi[i * p] = phi[i] * (p - 1);
23     mo[i * p] = mo[i] * mo[p];
24   }
25 }
```

## 8.11 Other Formulas

- Inversion:  
 $aa^{-1} \equiv 1 \pmod{m}$ .  $a^{-1}$  exists iff  $\gcd(a, m) = 1$ .

- Linear inversion:  
 $a^{-1} \equiv (m - \lfloor \frac{m}{a} \rfloor) \times (m \bmod a)^{-1} \pmod{m}$

- Fermat's little theorem:  
 $a^p \equiv a \pmod{p}$  if  $p$  is prime.

- Euler function:  
 $\phi(n) = n \prod_{p|n} \frac{p-1}{p}$

- Euler theorem:  
 $a^{\phi(n)} \equiv 1 \pmod{n}$  if  $\gcd(a, n) = 1$ .

- Extended Euclidean algorithm:

$$ax + by = \gcd(a, b) = \gcd(b, a \bmod b) = \gcd(b, a - \lfloor \frac{a}{b} \rfloor b) = bx_1 + (a - \lfloor \frac{a}{b} \rfloor b)y_1 = ay_1 + b(x_1 - \lfloor \frac{a}{b} \rfloor y_1)$$

- Divisor function:

$$\sigma_x(n) = \sum_{d|n} d^x \cdot n = \prod_{i=1}^r p_i^{a_i} \cdot n$$

$$\sigma_x(n) = \prod_{i=1}^r \frac{p_i^{(a_i+1)x} - 1}{p_i^x - 1} \text{ if } x \neq 0. \quad \sigma_0(n) = \prod_{i=1}^r (a_i + 1).$$

- Chinese remainder theorem (Coprime Moduli):

$$x \equiv a_i \pmod{m_i}.$$

$$M = \prod m_i. \quad M_i = M/m_i. \quad t_i = M_i^{-1}.$$

$$x = kM + \sum a_i t_i M_i, \quad k \in \mathbb{Z}.$$

- Chinese remainder theorem:

$$x \equiv a_1 \pmod{m_1}, x \equiv a_2 \pmod{m_2} \Rightarrow x = m_1 p + a_1 =$$

$$m_2 q + a_2 \Rightarrow m_1 p - m_2 q = a_2 - a_1$$

Solve for  $(p, q)$  using ExtGCD.

$$x \equiv m_1 p + a_1 \equiv m_2 q + a_2 \pmod{\text{lcm}(m_1, m_2)}$$

- Avoiding Overflow:  $ca \bmod cb = c(a \bmod b)$

- Dirichlet Convolution:  $(f * g)(n) = \sum_{d|n} f(n)g(n/d)$

- Important Multiplicative Functions + Properties:

1.  $\epsilon(n) = [n = 1]$
2.  $1(n) = 1$
3.  $id(n) = n$
4.  $\mu(n) = 0$  if  $n$  has squared prime factor
5.  $\mu(n) = (-1)^k$  if  $n = p_1 p_2 \cdots p_k$
6.  $\epsilon = \mu * 1$
7.  $\phi = \mu * id$
8.  $[n = 1] = \sum_{d|n} \mu(d)$
9.  $[gcd = 1] = \sum_{d|gcd} \mu(d)$

- Möbius inversion:  $f = g * 1 \Leftrightarrow g = f * \mu$

## 8.12 Polynomial

```

1 const int maxk = 20;
2 const int maxn = 1<<maxk;
3 const ll LINF = 1e18;
4
5 /* P = r*2^k + 1
6 P r k g
7 998244353 119 23 3
8 1004535809 479 21 3
9
10 P r k g
11 3 1 1 2
12 5 1 2 2
13 17 1 4 3
14 97 3 5 5
15 193 3 6 5
16 257 1 8 3
17 7681 15 9 17
18 12289 3 12 11
19 40961 5 13 3
20 65537 1 16 3
21 786433 3 18 10
22 5767169 11 19 3
23 7340033 7 20 3
24 23068673 11 21 3
25 104857601 25 22 3
26 167772161 5 25 3
27 469762049 7 26 3
28 1004535809 479 21 3
29 2013265921 15 27 31
30 2281701377 17 27 3
31 3221225473 3 30 5
32 75161927681 35 31 3
33 77309411329 9 33 7
34 206158430209 3 36 22
35 2061584302081 15 37 7
36 2748779069441 5 39 3
37 6597069766657 3 41 5
38 39582418599937 9 42 5
39 79164837199873 9 43 5
40 263882790666241 15 44 7
41 1231453023109121 35 45 3
42 1337006139375617 19 46 3
43 3799912185593857 27 47 5
44 4222124650659841 15 48 19
45 7881299347898369 7 50 6
46 31525197391593473 7 52 3
47 180143985094819841 5 55 6
48 1945555039024054273 27 56 5
49 4179340454199820289 29 57 3
50 9097271247288401921 505 54 6 */
51
52 const int g = 3;
53 const ll MOD = 998244353;
54
55 ll pw(ll a, ll n) { /* fast pow */ }
56
57 #define siz(x) (int)x.size()
58
59 template<typename T>
60 vector<T>& operator+=(vector<T>& a, const vector<T>& b)
61 {
62   if (siz(a) < siz(b)) a.resize(siz(b));
63   for (int i = 0; i < min(siz(a), siz(b)); i++) {
64     a[i] += b[i];
65     a[i] -= a[i] >= MOD ? MOD : 0;
66   }
67   return a;
68 }
69
70 template<typename T>
71 vector<T>& operator-=(vector<T>& a, const vector<T>& b)
72 {
73   if (siz(a) < siz(b)) a.resize(siz(b));
74   for (int i = 0; i < min(siz(a), siz(b)); i++) {
75     a[i] -= b[i];
76     a[i] += a[i] < 0 ? MOD : 0;
77   }
78 }
```

```

79 template<typename T>
80 vector<T> operator-(const vector<T>& a) {
81     vector<T> ret(siz(a));
82     for (int i = 0; i < siz(a); i++) {
83         ret[i] = -a[i] < 0 ? -a[i] + MOD : -a[i];
84     }
85     return ret;
86 }
87
88 vector<ll> X, iX;
89 vector<int> rev;
90
91 void init_ntt() {
92     X.clear(); X.resize(maxn, 1); //  $x_1 = g^{(p-1)/n}$ 
93     iX.clear(); iX.resize(maxn, 1);
94
95     ll u = pw(g, (MOD-1)/maxn);
96     ll iu = pw(u, MOD-2);
97
98     for (int i = 1; i < maxn; i++) {
99         X[i] = X[i-1] * u;
100        iX[i] = iX[i-1] * iu;
101        if (X[i] >= MOD) X[i] %= MOD;
102        if (iX[i] >= MOD) iX[i] %= MOD;
103    }
104
105    rev.clear(); rev.resize(maxn, 0);
106    for (int i = 1, hb = -1; i < maxn; i++) {
107        if (!(i & (i-1))) hb++;
108        rev[i] = rev[i ^ (1<<hb)] | (1<<(maxk-hb-1));
109    }
110}
111
112 template<typename T>
113 void NTT(vector<T>& a, bool inv=false) {
114     int _n = (int)a.size();
115     int k = __lg(_n) + ((1<<__lg(_n)) != _n);
116     int n = 1<<k;
117     a.resize(n, 0);
118
119     short shift = maxk-k;
120     for (int i = 0; i < n; i++)
121         if (i > (rev[i]>>shift))
122             swap(a[i], a[rev[i]>>shift]);
123
124     for (int len = 2, half = 1, div = maxn>>1; len <= n;
125         ; len<<=1, half<<=1, div>>=1) {
126         for (int i = 0; i < n; i += len) {
127             for (int j = 0; j < half; j++) {
128                 T u = a[i+j];
129                 T v = a[i+j+half] * (inv ? iX[j*div] : X[j*div]) % MOD;
130                 a[i+j] = (u+v >= MOD ? u+v-MOD : u+v);
131                 a[i+j+half] = (u-v < 0 ? u-v+MOD : u-v);
132             }
133         }
134         if (inv) {
135             T dn = pw(n, MOD-2);
136             for (auto& x : a) {
137                 x *= dn;
138                 if (x >= MOD) x %= MOD;
139             }
140
141         template<typename T>
142         inline void resize(vector<T>& a) {
143             int cnt = (int)a.size();
144             for (; cnt > 0; cnt--) if (a[cnt-1]) break;
145             a.resize(max(cnt, 1));
146         }
147
148         template<typename T>
149         vector<T>& operator*=(vector<T>& a, vector<T> b) {
150             int na = (int)a.size();
151             int nb = (int)b.size();
152             a.resize(na + nb - 1, 0);
153             b.resize(na + nb - 1, 0);
154
155             NTT(a); NTT(b);
156             for (int i = 0; i < (int)a.size(); i++) {
157                 a[i] *= b[i];
158                 if (a[i] >= MOD) a[i] %= MOD;
159             }
160             NTT(a, true);
161
162             resize(a);
163         }
164
165         template<typename T>
166         void inv(vector<T>& ia, int N) {
167             vector<T> _a(move(ia));
168             ia.resize(1, pw(_a[0], MOD-2));
169             vector<T> a(1, -_a[0] + (-_a[0] < 0 ? MOD : 0));
170
171             for (int n = 1; n < N; n<<=1) {
172                 //  $n \rightarrow 2^n$ 
173                 //  $ia' = ia(2-a*ia)$ ;
174
175                 for (int i = n; i < min(siz(_a), (n<<1)); i++)
176                     a.emplace_back(-_a[i] + (-_a[i] < 0 ? MOD : 0));
177
178                 vector<T> tmp = ia;
179                 ia *= a;
180                 ia.resize(n<<1);
181                 ia[0] = ia[0] + 2 >= MOD ? ia[0] + 2 - MOD : ia[0] + 2;
182                 ia *= tmp;
183                 ia.resize(n<<1);
184             }
185             ia.resize(N);
186         }
187
188         template<typename T>
189         void mod(vector<T>& a, vector<T>& b) {
190             int n = (int)a.size()-1, m = (int)b.size()-1;
191             if (n < m) return;
192
193             vector<T> ra = a, rb = b;
194             reverse(ra.begin(), ra.end()); ra.resize(min(n+1, n-m+1));
195             reverse(rb.begin(), rb.end()); rb.resize(min(m+1, n-m+1));
196
197             inv(rb, n-m+1);
198
199             vector<T> q = move(ra);
200             q *= rb;
201             q.resize(n-m+1);
202             reverse(q.begin(), q.end());
203
204             q *= b;
205             a -= q;
206             resize(a);
207         }
208
209         /* Kitamasa Method (Fast Linear Recurrence):
210         Find  $a[K]$  (Given  $a[j] = c[0]a[j-N] + \dots + c[N-1]a[-1]$ )
211         Let  $B(x) = x^N - c[N-1]x^{N-1} - \dots - c[1]x^1 - c[0]$ 
212         Let  $R(x) = x^K \bmod B(x)$  (get  $x^K$  using fast pow and
213         use poly mod to get  $R(x)$ )
214         Let  $r[i]$  = the coefficient of  $x^i$  in  $R(x)$ 
215          $\Rightarrow a[K] = a[0]r[0] + a[1]r[1] + \dots + a[N-1]r[N-1]$  */
216     }
217
218     vector<vector<ll>> v;
219     void gauss(vector<vector<ll>>& v) {
220         int r = 0;
221         for (int i = 0; i < n; i++) {
222             bool ok = false;
223             for (int j = r; j < n; j++) {
224                 if (v[j][i] == 0) continue;
225                 swap(v[j], v[r]);
226                 ok = true;
227             }
228             if (!ok) continue;
229             for (int j = r+1; j < n; j++)
230                 v[j][i] /= v[r][i];
231         }
232     }
233
234     vector<vector<ll>> v;
235     void gauss(vector<vector<ll>>& v) {
236         int r = 0;
237         for (int i = 0; i < n; i++) {
238             bool ok = false;
239             for (int j = r; j < n; j++) {
240                 if (v[j][i] == 0) continue;
241                 swap(v[j], v[r]);
242                 ok = true;
243             }
244             if (!ok) continue;
245             for (int j = r+1; j < n; j++)
246                 v[j][i] /= v[r][i];
247         }
248     }
249
250     vector<vector<ll>> v;
251     void gauss(vector<vector<ll>>& v) {
252         int r = 0;
253         for (int i = 0; i < n; i++) {
254             bool ok = false;
255             for (int j = r; j < n; j++) {
256                 if (v[j][i] == 0) continue;
257                 swap(v[j], v[r]);
258                 ok = true;
259             }
260             if (!ok) continue;
261             for (int j = r+1; j < n; j++)
262                 v[j][i] /= v[r][i];
263         }
264     }
265
266     vector<vector<ll>> v;
267     void gauss(vector<vector<ll>>& v) {
268         int r = 0;
269         for (int i = 0; i < n; i++) {
270             bool ok = false;
271             for (int j = r; j < n; j++) {
272                 if (v[j][i] == 0) continue;
273                 swap(v[j], v[r]);
274                 ok = true;
275             }
276             if (!ok) continue;
277             for (int j = r+1; j < n; j++)
278                 v[j][i] /= v[r][i];
279         }
280     }
281
282     vector<vector<ll>> v;
283     void gauss(vector<vector<ll>>& v) {
284         int r = 0;
285         for (int i = 0; i < n; i++) {
286             bool ok = false;
287             for (int j = r; j < n; j++) {
288                 if (v[j][i] == 0) continue;
289                 swap(v[j], v[r]);
290                 ok = true;
291             }
292             if (!ok) continue;
293             for (int j = r+1; j < n; j++)
294                 v[j][i] /= v[r][i];
295         }
296     }
297
298     vector<vector<ll>> v;
299     void gauss(vector<vector<ll>>& v) {
300         int r = 0;
301         for (int i = 0; i < n; i++) {
302             bool ok = false;
303             for (int j = r; j < n; j++) {
304                 if (v[j][i] == 0) continue;
305                 swap(v[j], v[r]);
306                 ok = true;
307             }
308             if (!ok) continue;
309             for (int j = r+1; j < n; j++)
310                 v[j][i] /= v[r][i];
311         }
312     }
313
314     vector<vector<ll>> v;
315     void gauss(vector<vector<ll>>& v) {
316         int r = 0;
317         for (int i = 0; i < n; i++) {
318             bool ok = false;
319             for (int j = r; j < n; j++) {
320                 if (v[j][i] == 0) continue;
321                 swap(v[j], v[r]);
322                 ok = true;
323             }
324             if (!ok) continue;
325             for (int j = r+1; j < n; j++)
326                 v[j][i] /= v[r][i];
327         }
328     }
329
330     vector<vector<ll>> v;
331     void gauss(vector<vector<ll>>& v) {
332         int r = 0;
333         for (int i = 0; i < n; i++) {
334             bool ok = false;
335             for (int j = r; j < n; j++) {
336                 if (v[j][i] == 0) continue;
337                 swap(v[j], v[r]);
338                 ok = true;
339             }
340             if (!ok) continue;
341             for (int j = r+1; j < n; j++)
342                 v[j][i] /= v[r][i];
343         }
344     }
345
346     vector<vector<ll>> v;
347     void gauss(vector<vector<ll>>& v) {
348         int r = 0;
349         for (int i = 0; i < n; i++) {
350             bool ok = false;
351             for (int j = r; j < n; j++) {
352                 if (v[j][i] == 0) continue;
353                 swap(v[j], v[r]);
354                 ok = true;
355             }
356             if (!ok) continue;
357             for (int j = r+1; j < n; j++)
358                 v[j][i] /= v[r][i];
359         }
360     }
361
362     vector<vector<ll>> v;
363     void gauss(vector<vector<ll>>& v) {
364         int r = 0;
365         for (int i = 0; i < n; i++) {
366             bool ok = false;
367             for (int j = r; j < n; j++) {
368                 if (v[j][i] == 0) continue;
369                 swap(v[j], v[r]);
370                 ok = true;
371             }
372             if (!ok) continue;
373             for (int j = r+1; j < n; j++)
374                 v[j][i] /= v[r][i];
375         }
376     }
377
378     vector<vector<ll>> v;
379     void gauss(vector<vector<ll>>& v) {
380         int r = 0;
381         for (int i = 0; i < n; i++) {
382             bool ok = false;
383             for (int j = r; j < n; j++) {
384                 if (v[j][i] == 0) continue;
385                 swap(v[j], v[r]);
386                 ok = true;
387             }
388             if (!ok) continue;
389             for (int j = r+1; j < n; j++)
390                 v[j][i] /= v[r][i];
391         }
392     }
393
394     vector<vector<ll>> v;
395     void gauss(vector<vector<ll>>& v) {
396         int r = 0;
397         for (int i = 0; i < n; i++) {
398             bool ok = false;
399             for (int j = r; j < n; j++) {
400                 if (v[j][i] == 0) continue;
401                 swap(v[j], v[r]);
402                 ok = true;
403             }
404             if (!ok) continue;
405             for (int j = r+1; j < n; j++)
406                 v[j][i] /= v[r][i];
407         }
408     }
409
410     vector<vector<ll>> v;
411     void gauss(vector<vector<ll>>& v) {
412         int r = 0;
413         for (int i = 0; i < n; i++) {
414             bool ok = false;
415             for (int j = r; j < n; j++) {
416                 if (v[j][i] == 0) continue;
417                 swap(v[j], v[r]);
418                 ok = true;
419             }
420             if (!ok) continue;
421             for (int j = r+1; j < n; j++)
422                 v[j][i] /= v[r][i];
423         }
424     }
425
426     vector<vector<ll>> v;
427     void gauss(vector<vector<ll>>& v) {
428         int r = 0;
429         for (int i = 0; i < n; i++) {
430             bool ok = false;
431             for (int j = r; j < n; j++) {
432                 if (v[j][i] == 0) continue;
433                 swap(v[j], v[r]);
434                 ok = true;
435             }
436             if (!ok) continue;
437             for (int j = r+1; j < n; j++)
438                 v[j][i] /= v[r][i];
439         }
440     }
441
442     vector<vector<ll>> v;
443     void gauss(vector<vector<ll>>& v) {
444         int r = 0;
445         for (int i = 0; i < n; i++) {
446             bool ok = false;
447             for (int j = r; j < n; j++) {
448                 if (v[j][i] == 0) continue;
449                 swap(v[j], v[r]);
450                 ok = true;
451             }
452             if (!ok) continue;
453             for (int j = r+1; j < n; j++)
454                 v[j][i] /= v[r][i];
455         }
456     }
457
458     vector<vector<ll>> v;
459     void gauss(vector<vector<ll>>& v) {
460         int r = 0;
461         for (int i = 0; i < n; i++) {
462             bool ok = false;
463             for (int j = r; j < n; j++) {
464                 if (v[j][i] == 0) continue;
465                 swap(v[j], v[r]);
466                 ok = true;
467             }
468             if (!ok) continue;
469             for (int j = r+1; j < n; j++)
470                 v[j][i] /= v[r][i];
471         }
472     }
473
474     vector<vector<ll>> v;
475     void gauss(vector<vector<ll>>& v) {
476         int r = 0;
477         for (int i = 0; i < n; i++) {
478             bool ok = false;
479             for (int j = r; j < n; j++) {
480                 if (v[j][i] == 0) continue;
481                 swap(v[j], v[r]);
482                 ok = true;
483             }
484             if (!ok) continue;
485             for (int j = r+1; j < n; j++)
486                 v[j][i] /= v[r][i];
487         }
488     }
489
490     vector<vector<ll>> v;
491     void gauss(vector<vector<ll>>& v) {
492         int r = 0;
493         for (int i = 0; i < n; i++) {
494             bool ok = false;
495             for (int j = r; j < n; j++) {
496                 if (v[j][i] == 0) continue;
497                 swap(v[j], v[r]);
498                 ok = true;
499             }
500             if (!ok) continue;
501             for (int j = r+1; j < n; j++)
502                 v[j][i] /= v[r][i];
503         }
504     }
505
506     vector<vector<ll>> v;
507     void gauss(vector<vector<ll>>& v) {
508         int r = 0;
509         for (int i = 0; i < n; i++) {
510             bool ok = false;
511             for (int j = r; j < n; j++) {
512                 if (v[j][i] == 0) continue;
513                 swap(v[j], v[r]);
514                 ok = true;
515             }
516             if (!ok) continue;
517             for (int j = r+1; j < n; j++)
518                 v[j][i] /= v[r][i];
519         }
520     }
521
522     vector<vector<ll>> v;
523     void gauss(vector<vector<ll>>& v) {
524         int r = 0;
525         for (int i = 0; i < n; i++) {
526             bool ok = false;
527             for (int j = r; j < n; j++) {
528                 if (v[j][i] == 0) continue;
529                 swap(v[j], v[r]);
530                 ok = true;
531             }
532             if (!ok) continue;
533             for (int j = r+1; j < n; j++)
534                 v[j][i] /= v[r][i];
535         }
536     }
537
538     vector<vector<ll>> v;
539     void gauss(vector<vector<ll>>& v) {
540         int r = 0;
541         for (int i = 0; i < n; i++) {
542             bool ok = false;
543             for (int j = r; j < n; j++) {
544                 if (v[j][i] == 0) continue;
545                 swap(v[j], v[r]);
546                 ok = true;
547             }
548             if (!ok) continue;
549             for (int j = r+1; j < n; j++)
550                 v[j][i] /= v[r][i];
551         }
552     }
553
554     vector<vector<ll>> v;
555     void gauss(vector<vector<ll>>& v) {
556         int r = 0;
557         for (int i = 0; i < n; i++) {
558             bool ok = false;
559             for (int j = r; j < n; j++) {
560                 if (v[j][i] == 0) continue;
561                 swap(v[j], v[r]);
562                 ok = true;
563             }
564             if (!ok) continue;
565             for (int j = r+1; j < n; j++)
566                 v[j][i] /= v[r][i];
567         }
568     }
569
570     vector<vector<ll>> v;
571     void gauss(vector<vector<ll>>& v) {
572         int r = 0;
573         for (int i = 0; i < n; i++) {
574             bool ok = false;
575             for (int j = r; j < n; j++) {
576                 if (v[j][i] == 0) continue;
577                 swap(v[j], v[r]);
578                 ok = true;
579             }
580             if (!ok) continue;
581             for (int j = r+1; j < n; j++)
582                 v[j][i] /= v[r][i];
583         }
584     }
585
586     vector<vector<ll>> v;
587     void gauss(vector<vector<ll>>& v) {
588         int r = 0;
589         for (int i = 0; i < n; i++) {
590             bool ok = false;
591             for (int j = r; j < n; j++) {
592                 if (v[j][i] == 0) continue;
593                 swap(v[j], v[r]);
594                 ok = true;
595             }
596             if (!ok) continue;
597             for (int j = r+1; j < n; j++)
598                 v[j][i] /= v[r][i];
599         }
600     }
601
602     vector<vector<ll>> v;
603     void gauss(vector<vector<ll>>& v) {
604         int r = 0;
605         for (int i = 0; i < n; i++) {
606             bool ok = false;
607             for (int j = r; j < n; j++) {
608                 if (v[j][i] == 0) continue;
609                 swap(v[j], v[r]);
610                 ok = true;
611             }
612             if (!ok) continue;
613             for (int j = r+1; j < n; j++)
614                 v[j][i] /= v[r][i];
615         }
616     }
617
618     vector<vector<ll>> v;
619     void gauss(vector<vector<ll>>& v) {
620         int r = 0;
621         for (int i = 0; i < n; i++) {
622             bool ok = false;
623             for (int j = r; j < n; j++) {
624                 if (v[j][i] == 0) continue;
625                 swap(v[j], v[r]);
626                 ok = true;
627             }
628             if (!ok) continue;
629             for (int j = r+1; j < n; j++)
630                 v[j][i] /= v[r][i];
631         }
632     }
633
634     vector<vector<ll>> v;
635     void gauss(vector<vector<ll>>& v) {
636         int r = 0;
637         for (int i = 0; i < n; i++) {
638             bool ok = false;
639             for (int j = r; j < n; j++) {
640                 if (v[j][i] == 0) continue;
641                 swap(v[j], v[r]);
642                 ok = true;
643             }
644             if (!ok) continue;
645             for (int j = r+1; j < n; j++)
646                 v[j][i] /= v[r][i];
647         }
648     }
649
650     vector<vector<ll>> v;
651     void gauss(vector<vector<ll>>& v) {
652         int r = 0;
653         for (int i = 0; i < n; i++) {
654             bool ok = false;
655             for (int j = r; j < n; j++) {
656                 if (v[j][i] == 0) continue;
657                 swap(v[j], v[r]);
658                 ok = true;
659             }
660             if (!ok) continue;
661             for (int j = r+1; j < n; j++)
662                 v[j][i] /= v[r][i];
663         }
664     }
665
666     vector<vector<ll>> v;
667     void gauss(vector<vector<ll>>& v) {
668         int r = 0;
669         for (int i = 0; i < n; i++) {
670             bool ok = false;
671             for (int j = r; j < n; j++) {
672                 if (v[j][i] == 0) continue;
673                 swap(v[j], v[r]);
674                 ok = true;
675             }
676             if (!ok) continue;
677             for (int j = r+1; j < n; j++)
678                 v[j][i] /= v[r][i];
679         }
680     }
681
682     vector<vector<ll>> v;
683     void gauss(vector<vector<ll>>& v) {
684         int r = 0;
685         for (int i = 0; i < n; i++) {
686             bool ok = false;
687             for (int j = r; j < n; j++) {
688                 if (v[j][i] == 0) continue;
689                 swap(v[j], v[r]);
690                 ok = true;
691             }
692             if (!ok) continue;
693             for (int j = r+1; j < n; j++)
694                 v[j][i] /= v[r][i];
695         }
696     }
697
698     vector<vector<ll>> v;
699     void gauss(vector<vector<ll>>& v) {
700         int r = 0;
701         for (int i = 0; i < n; i++) {
702             bool ok = false;
703             for (int j = r; j < n; j++) {
704                 if (v[j][i] == 0) continue;
705                 swap(v[j], v[r]);
706                 ok = true;
707             }
708             if (!ok) continue;
709             for (int j = r+1; j < n; j++)
710                 v[j][i] /= v[r][i];
711         }
712     }
713
714     vector<vector<ll>> v;
715     void gauss(vector<vector<ll>>& v) {
716         int r = 0;
717         for (int i = 0; i < n; i++) {
718             bool ok = false;
719             for (int j = r; j < n; j++) {
720                 if (v[j][i] == 0) continue;
721                 swap(v[j], v[r]);
722                 ok = true;
723             }
724             if (!ok) continue;
725             for (int j = r+1; j < n; j++)
726                 v[j][i] /= v[r][i];
727         }
728     }
729
730     vector<vector<ll>> v;
731     void gauss(vector<vector<ll>>& v) {
732         int r = 0;
733         for (int i = 0; i < n; i++) {
734             bool ok = false;
735             for (int j = r; j < n; j++) {
736                 if (v[j][i] == 0) continue;
737                 swap(v[j], v[r]);
738                 ok = true;
739             }
740             if (!ok) continue;
741             for (int j = r+1; j < n; j++)
742                 v[j][i] /= v[r][i];
743         }
744     }
745
746     vector<vector<ll>> v;
747     void gauss(vector<vector<ll>>& v) {
748         int r = 0;
749         for (int i = 0; i < n; i++) {
750             bool ok = false;
751             for (int j = r; j < n; j++) {
752                 if (v[j][i] == 0) continue;
753                 swap(v[j], v[r]);
754                 ok = true;
755             }
756             if (!ok) continue;
757             for (int j = r+1; j < n; j++)
758                 v[j][i] /= v[r][i];
759         }
760     }
761
762     vector<vector<ll>> v;
763     void gauss(vector<vector<ll>>& v) {
764         int r = 0;
765         for (int i = 0; i < n; i++) {
766             bool ok = false;
767             for (int j = r; j < n; j++) {
768                 if (v[j][i] == 0) continue;
769                 swap(v[j], v[r]);
770                 ok = true;
771             }
772             if (!ok) continue;
773             for (int j = r+1; j < n; j++)
774                 v[j][i] /= v[r][i];
775         }
776     }
777
778     vector<vector<ll>> v;
779     void gauss(vector<vector<ll>>& v) {
780         int r = 0;
781         for (int i = 0; i < n; i++) {
782             bool ok = false;
783             for (int j = r; j < n; j++) {
784                 if (v[j][i] == 0) continue;
785                 swap(v[j], v[r]);
786                 ok = true;
787             }
788             if (!ok) continue;
789             for (int j = r+1; j < n; j++)
790                 v[j][i] /= v[r][i];
791         }
792     }
793
794     vector<vector<ll>> v;
795     void gauss(vector<vector<ll>>& v) {
796         int r = 0;
797         for (int i = 0; i < n; i++) {
798             bool ok = false;
799             for (int j = r; j < n; j++) {
800                 if (v[j][i] == 0) continue;
801                 swap(v[j], v[r]);
802                 ok = true;
803             }
804             if (!ok) continue;
805             for (int j = r+1; j < n; j++)
806                 v[j][i] /= v[r][i];
807         }
808     }
809
810     vector<vector<ll>> v;
811     void gauss(vector<vector<ll>>& v) {
812         int r = 0;
813         for (int i = 0; i < n; i++) {
814             bool ok = false;
815             for (int j = r; j < n; j++) {
816                 if (v[j][i] == 0) continue;
817                 swap(v[j], v[r]);
818                 ok = true;
819             }
820             if (!ok) continue;
821             for (int j = r+1; j < n; j++)
822                 v[j][i] /= v[r][i];
823         }
824     }
825
826     vector<vector<ll>> v;
827     void gauss(vector<vector<ll>>& v) {
828         int r = 0;
829         for (int i = 0; i < n; i++) {
830             bool ok = false;
831             for (int j = r; j < n; j++) {
832                 if (v[j][i] == 0) continue;
833                 swap(v[j], v[r]);
834                 ok = true;
835             }
836             if (!ok) continue;
837             for (int j = r+1; j < n; j++)
838                 v[j][i] /= v[r][i];
839         }
840     }
841
842     vector<vector<ll>> v;
843     void gauss(vector<vector<ll>>& v) {
844         int r = 0;
845         for (int i = 0; i < n; i++) {
846             bool ok = false;
847             for (int j = r; j < n; j++) {
848                 if (v[j][i] == 0) continue;
849                 swap(v[j], v[r]);
850                 ok = true;
851             }
852             if (!ok) continue;
853             for (int j = r+1; j < n; j++)
854                 v[j][i] /= v[r][i];
855         }
856     }
857
858     vector<vector<ll>> v;
859     void gauss(vector<vector<ll>>& v) {
860         int r = 0;
861         for (int i = 0; i < n; i++) {
862             bool ok = false;
863             for (int j = r; j < n; j++) {
864                 if (v[j][i] == 0) continue;
865                 swap(v[j], v[r]);
866                 ok = true;
867             }
868             if (!ok) continue;
869             for (int j = r+1; j < n; j++)
870                 v[j][i] /= v[r][i];
871         }
872     }
873
874     vector<vector<ll>> v;
875     void gauss(vector<vector<ll>>& v) {
876         int r = 0;
877         for (int i = 0; i < n; i++) {
878             bool ok = false;
879             for (int j = r; j < n; j++) {
880                 if (v[j][i] == 0) continue;
881                 swap(v[j], v[r]);
882                 ok = true;
883             }
884             if (!ok) continue;
885             for (int j = r+1; j < n; j++)
886                 v[j][i] /= v[r][i];
887         }
888     }
889
890     vector<vector<ll>> v;
891     void gauss(vector<vector<ll>>& v) {
892         int r = 0;
893         for (int i = 0; i < n; i++) {
894             bool ok = false;
895             for (int j = r; j < n; j++) {
896                 if (v[j][i] == 0) continue;
897                 swap(v[j], v[r]);
898                 ok = true;
899             }
900             if (!ok) continue;
901             for (int j = r+1; j < n; j++)
902                 v[j][i] /= v[r][i];
903         }
904     }
905
906     vector<vector<ll>> v;
907     void gauss(vector<vector<ll>>& v) {
908         int r = 0;
909         for (int i = 0; i < n; i++) {
910             bool ok = false;
911             for (int j = r; j < n; j++) {
912                 if (v[j][i] == 0) continue;
913                 swap(v[j], v[r]);
914                 ok = true;
915             }
916             if (!ok) continue;
917             for (int j = r+1; j < n; j++)
918                 v[j][i] /= v[r][i];
919         }
920     }
921
922     vector<vector<ll>> v;
923     void gauss(vector<vector<ll>>& v) {
924         int r = 0;
925         for (int i = 0; i < n; i++) {
926             bool ok = false;
927             for (int j = r; j < n; j++) {
928                 if (v[j][i] == 0) continue;
929                 swap(v[j], v[r]);
930                 ok = true;
931             }
932             if (!ok) continue;
933             for (int j = r+1; j < n; j++)
934                 v[j][i] /= v[r][i];
935         }
936     }
937
938     vector<vector<ll>> v;
939     void
```

```

14     ll div = inv(v[r][i]);
15     for (int j = 0; j < n + 1; j++) {
16         v[r][j] *= div;
17         if (v[r][j] >= MOD) v[r][j] %= MOD;
18     }
19     for (int j = 0; j < n; j++) {
20         if (j == r) continue;
21         ll t = v[j][i];
22         for (int k = 0; k < n + 1; k++) {
23             v[j][k] -= v[r][k] * t % MOD;
24             if (v[j][k] < 0) v[j][k] += MOD;
25         }
26     }
27     r++;
28 }
29 }
```

## 9.2 Determinant

1. Use GJ Elimination, if there's any row consists of only 0, then  $\det = 0$ , otherwise  $\det = \text{product of diagonal elements}$ .

2. Properties of  $\det$ :

- Transpose: Unchanged
- Row Operation 1 - Swap 2 rows:  $-\det$
- Row Operation 2 -  $k\vec{r}_i$ :  $k \times \det$
- Row Operation 3 -  $k\vec{r}_i$  add to  $\vec{r}_j$ : Unchaged

# 10 Combinatorics

## 10.1 Catalan Number

$$C_0 = 1, C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i}, C_n = C_n^{2n} - C_{n-1}^{2n}$$

0	1	1	2	5
4	14	42	132	429
8	1430	4862	16796	58786
12	208012	742900	2674440	9694845

## 10.2 Burnside's Lemma

Let  $X$  be the original set.

Let  $G$  be the group of operations acting on  $X$ .

Let  $X^g$  be the set of  $x$  not affected by  $g$ .

Let  $X/G$  be the set of orbits.

Then the following equation holds:

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

