

# Contents

1	Reminder	1
1.1	Bug List	1
1.2	OwO	1
2	Basic	1
2.1	Vimrc	1
2.2	Runcpp.sh	1
2.3	Stress	1
2.4	PBDS	1
2.5	Random	1
3	Python	1
3.1	I/O	1
3.2	Decimal	2
4	Data Structure	2
4.1	Heavy Light Decomposition	2
4.2	Skew Heap	2
4.3	Leftist Heap	3
4.4	Treap	3
4.5	Persistent Treap	3
4.6	Li Chao Tree	3
4.7	Time Segment Tree	4
5	DP	4
5.1	Aliens	4
6	Graph	5
6.1	Bellman-Ford + SPFA	5
6.2	BCC - AP	5
6.3	BCC - Bridge	6
6.4	SCC - Tarjan	6
6.5	SCC - Kosaraju	7
6.6	Eulerian Path - Undir	7
6.7	Eulerian Path - Dir	7
6.8	Hamilton Path	8
6.9	Kth Shortest Path	8
6.10	System of Difference Constraints	9
7	String	9
7.1	Rolling Hash	9
7.2	Trie	9
7.3	KMP	9
7.4	Z Value	9
7.5	Manacher	10
7.6	Suffix Array	10
7.7	SA-IS	10
7.8	Minimum Rotation	11
7.9	Aho Corasick	11
8	Geometry	11
8.1	Basic Operations	11
8.2	InPoly	11
8.3	Sort by Angle	11
8.4	Line Intersect Check	11
8.5	Line Intersection	11
8.6	Convex Hull	12
8.7	Lower Concave Hull	12
8.8	Polygon Area	12
8.9	Pick's Theorem	12
8.10	Minimum Enclosing Circle	12
8.11	PolyUnion	12
8.12	Minkowski Sum	13
9	Number Theory	13
9.1	FFT	13
9.2	Pollard's rho	14
9.3	Miller Rabin	14
9.4	Fast Power	14
9.5	Extend GCD	14
9.6	Mu + Phi	14
9.7	Other Formulas	15
9.8	Polynomial	15
10	Linear Algebra	16
10.1	Gaussian-Jordan Elimination	16
10.2	Determinant	17
11	Flow / Matching	17
11.1	Dinic	17
11.2	ISAP	17
11.3	MCMF	17
11.4	Hopcroft-Karp	17
11.5	Cover / Independent Set	18
11.6	KM	18
12	Combinatorics	18
12.1	Catalan Number	19
12.2	Burnside's Lemma	19
13	Special Numbers	19
13.1	Fibonacci Series	19
13.2	Prime Numbers	19

## 1 Reminder

### 1.1 Bug List

- 沒開 long long
- 陣列戳出界／開不夠大／開太大本地 compile 噴怪 error
- 傳之前先確定選對檔案
- 寫好的函式忘記呼叫
- 變數打錯
- 0-base / 1-base
- 忘記初始化
- == 打成 =
- <= 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
- 線段樹改值懶標初始值不能設為 0
- DFS 的時候不小心覆寫到全域變數
- 浮點數誤差
- unsigned int128
- 多筆測資不能沒讀完直接 return
- 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!

## 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
"set guifont Hack:h16
":set guifont?

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

vmap <C-c> "+y
inoremap <C-v> <Esc>p
nnoremap <C-v> p

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 Runcpp.sh

```

#!/bin/bash
clear
echo "Start compiling $1..."
echo
g++ -O2 -std=c++20 -Wall -Wextra -Wshadow $2/$1 -o $2/
out
if [ "$?" -ne 0 ]

```

```

7 then
8     exit 1
9 fi
10 echo
11 echo "Done compiling"
12 echo "=====
13 echo
14 echo "Input file:"
15 echo
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"

```

## 2.3 Stress

```

1 g++ gen.cpp -o gen.out
2 g++ ac.cpp -o ac.out
3 g++ wa.cpp -o wa.out
4 for ((i=0;;i++))
5 do
6     echo "$i"
7     ./gen.out > in.txt
8     ./ac.out < in.txt > ac.txt
9     ./wa.out < in.txt > wa.txt
10    diff ac.txt wa.txt || break
11 done

```

## 2.4 PBDS

```

1 #include <bits/extc++.h>
2 using namespace __gnu_pbds;
3
4 // map
5 tree<int, int, less<>, rb_tree_tag,
6     tree_order_statistics_node_update> tr;
7 tr.order_of_key(element);
8 tr.find_by_order(rank);
9
10 // set
11 tree<int, null_type, less<>, rb_tree_tag,
12     tree_order_statistics_node_update> tr;
13 tr.order_of_key(element);
14 tr.find_by_order(rank);
15
16 // priority queue
17 __gnu_pbds::priority_queue<int, less<int> > big_q; //
18     Big First
19 __gnu_pbds::priority_queue<int, greater<int> > small_q;
20 // Small First
21 q1.join(q2); // join

```

## 2.5 Random

```

1 mt19937 gen(chrono::steady_clock::now().
2     time_since_epoch().count());
3 uniform_int_distribution<int> dis(1, 100);
4 cout << dis(gen) << endl;
5 shuffle(v.begin(), v.end(), gen);

```

# 3 Python

## 3.1 I/O

```

1 import sys
2 input = sys.stdin.readline
3
4 # Input
5 def readInt():

```

```

6     return int(input())
7 def readList():
8     return list(map(int,input().split()))
9 def readStr():
10    s = input()
11    return list(s[:len(s) - 1])
12 def readVars():
13    return map(int,input().split())
14
15 # Output
16 sys.stdout.write(string)
17
18 # faster
19 def main():
20     pass
21 main()

```

## 3.2 Decimal

```

1 from decimal import *
2 getcontext().prec = 2500000
3 getcontext().Emax = 2500000
4 a,b = Decimal(input()),Decimal(input())
5 a*=b
6 print(a)

```

# 4 Data Structure

## 4.1 Heavy Light Decomposition

```

1 const int N=2e5+5;
2 int n,dfn[N],son[N],top[N],num[N],dep[N],p[N];
3 vector<int>path[N];
4 struct node
5 {
6     int mx,sum;
7 }seg[N<<2];
8 void update(int x,int l,int r,int qx,int val)
9 {
10    if(l==r)
11    {
12        seg[x].mx=seg[x].sum=val;
13        return;
14    }
15    int mid=(l+r)>>1;
16    if(qx<=mid)update(x<<1,l,mid,qx,val);
17    else update(x<<1|1,mid+1,r,qx,val);
18    seg[x].mx=max(seg[x<<1].mx,seg[x<<1|1].mx);
19    seg[x].sum=seg[x<<1].sum+seg[x<<1|1].sum;
20 }
21 int big(int x,int l,int r,int ql,int qr)
22 {
23    if(ql<=l&&r<=qr)return seg[x].mx;
24    int mid=(l+r)>>1;
25    int res=-INF;
26    if(ql<=mid)res=max(res,big(x<<1,l,mid,ql,qr));
27    if(mid<qr)res=max(res,big(x<<1|1,mid+1,r,ql,qr));
28    return res;
29 }
30 int ask(int x,int l,int r,int ql,int qr)
31 {
32    if(ql<=l&&r<=qr)return seg[x].sum;
33    int mid=(l+r)>>1;
34    int res=0;
35    if(ql<=mid)res+=ask(x<<1,l,mid,ql,qr);
36    if(mid<qr)res+=ask(x<<1|1,mid+1,r,ql,qr);
37    return res;
38 }
39 void dfs1(int now)
40 {
41    son[now]=-1;
42    num[now]=1;
43    for(auto i:path[now])
44    {
45        if(!dep[i])
46        {
47            dep[i]=dep[now]+1;
48            p[i]=now;
49            dfs1(i);

```

```

50         num[now]+=num[i];
51         if(son[now]==-1||num[i]>num[son[now]])son[
            now]=i;
52     }
53 }
54 }
55 int cnt;
56 void dfs2(int now,int t)
57 {
58     top[now]=t;
59     cnt++;
60     dfn[now]=cnt;
61     if(son[now]==-1)return;
62     dfs2(son[now],t);
63     for(auto i:path[now])
64         if(i!=p[now]&&i!=son[now])
65             dfs2(i,i);
66 }
67 int path_big(int x,int y)
68 {
69     int res=-INF;
70     while(top[x]!=top[y])
71     {
72         if(dep[top[x]]<dep[top[y]])swap(x,y);
73         res=max(res,big(1,1,n,dfn[top[x]],dfn[x]));
74         x=p[top[x]];
75     }
76     if(dfn[x]>dfn[y])swap(x,y);
77     res=max(res,big(1,1,n,dfn[x],dfn[y]));
78     return res;
79 }
80 int path_sum(int x,int y)
81 {
82     int res=0;
83     while(top[x]!=top[y])
84     {
85         if(dep[top[x]]<dep[top[y]])swap(x,y);
86         res+=ask(1,1,n,dfn[top[x]],dfn[x]);
87         x=p[top[x]];
88     }
89     if(dfn[x]>dfn[y])swap(x,y);
90     res+=ask(1,1,n,dfn[x],dfn[y]);
91     return res;
92 }
93 void buildTree()
94 {
95     FOR(i,0,n-1)
96     {
97         int a,b;cin>>a>>b;
98         path[a].pb(b);
99         path[b].pb(a);
100     }
101 }
102 void buildHLD(int root)
103 {
104     dep[root]=1;
105     dfs1(root);
106     dfs2(root,root);
107     FOR(i,1,n+1)
108     {
109         int now;cin>>now;
110         update(1,1,n,dfn[i],now);
111     }
112 }

```

## 4.2 Skew Heap

```

1 struct node{
2     node *l,*r;
3     int v;
4     node(int x):v(x){
5         l=r=nullptr;
6     }
7 };
8 node* merge(node* a,node* b){
9     if(!a||!b) return a?:b;
10    // min heap
11    if(a->v>b->v) swap(a,b);
12    a->r=merge(a->r,b);
13    swap(a->l,a->r);
14    return a;

```

## 4.3 Leftist Heap

```

1 struct node{
2     node *l,*r;
3     int d, v;
4     node(int x):d(1),v(x){
5         l=r=nullptr;
6     }
7 };
8 static inline int d(node* x){return x?x->d:0;}
9 node* merge(node* a,node* b){
10     if(!a||!b) return a?:b;
11    // min heap
12    if(a->v>b->v) swap(a,b);
13    a->r=merge(a->r,b);
14    if(d(a->l)<d(a->r))
15        swap(a->l,a->r);
16    a->d=d(a->r)+1;
17    return a;
18 }

```

## 4.4 Treap

```

1 mt19937 rng(random_device{}());
2 struct Treap
3 {
4     Treap *l,*r;
5     int val,num,pri;
6     Treap(int k)
7     {
8         l=r=NULL;
9         val=k;
10        num=1;
11        pri=rng();
12    }
13 };
14 int siz(Treap *now){return now?now->num:0;}
15 void pull(Treap *&now)
16 {
17     now->num=siz(now->l)+siz(now->r)+1;
18 }
19 Treap* merge(Treap *a,Treap *b)
20 {
21     if(!a||!b)return a?:b;
22     else if(a->pri>b->pri)
23     {
24         a->r=merge(a->r,b);
25         pull(a);
26         return a;
27     }
28     else
29     {
30         b->l=merge(a,b->l);
31         pull(b);
32         return b;
33     }
34 }
35 void split_size(Treap *rt,Treap *&a,Treap *&b,int val)
36 {
37     if(!rt)
38     {
39         a=b=NULL;
40         return;
41     }
42     if(siz(rt->l)+1>val)
43     {
44         b=rt;
45         split_size(rt->l,a,b->l,val);
46         pull(b);
47     }
48     else
49     {
50         a=rt;
51         split_size(rt->r,a->r,b,val-siz(a->l)-1);
52         pull(a);
53     }
54 }
55 void split_val(Treap *rt,Treap *&a,Treap *&b,int val)

```

```

56 {
57     if(!rt)
58     {
59         a=b=NULL;
60         return;
61     }
62     if(rt->val<=val)
63     {
64         a=rt;
65         split_val(rt->r,a->r,b,val);
66         pull(a);
67     }
68     else
69     {
70         b=rt;
71         split_val(rt->l,a,b->l,val);
72         pull(b);
73     }
74 }
75 void treap_dfs(Treap *now)
76 {
77     if(!now) return;
78     treap_dfs(now->l);
79     cout<<now->val<<" ";
80     treap_dfs(now->r);
81 }

```

## 4.5 Persistent Treap

```

1 struct node {
2     node *l, *r;
3     char c; int v, sz;
4     node(char x = '$'): c(x), v(mt()), sz(1) {
5         l = r = nullptr;
6     }
7     node(node* p) { *this = *p; }
8     void pull() {
9         sz = 1;
10        for (auto i : {l, r})
11            if (i) sz += i->sz;
12    }
13 } arr[maxn], *ptr = arr;
14 inline int size(node* p) { return p ? p->sz : 0; }
15 node* merge(node* a, node* b) {
16     if (!a || !b) return a ? b;
17     if (a->v < b->v) {
18         node* ret = new(ptr++) node(a);
19         ret->r = merge(ret->r, b); ret->pull();
20         return ret;
21     }
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     }
36     else {
37         auto [a, b] = split(p->l, k);
38         node* ret = new(ptr++) node(p);
39         ret->l = b; ret->pull();
40         return {a, ret};
41     }
42 }

```

## 4.6 Li Chao Tree

```

1 constexpr int maxn = 5e4 + 5;
2 struct line {
3     ld a, b;
4     ld operator()(ld x) { return a * x + b; }
5 } arr[(maxn + 1) << 2];
6 bool operator<(line a, line b) { return a.a < b.a; }

```

```

7 #define m ((l+r)>>1)
8 void insert(line x, int i = 1, int l = 0, int r = maxn)
9 {
10     if (r - l == 1) {
11         if (x(l) > arr[i](l))
12             arr[i] = x;
13         return;
14     }
15     line a = max(arr[i], x), b = min(arr[i], x);
16     if (a(m) > b(m))
17         arr[i] = a, insert(b, i << 1, l, m);
18     else
19         arr[i] = b, insert(a, i << 1 | 1, m, r);
20 }
21 ld query(int x, int i = 1, int l = 0, int r = maxn) {
22     if (x < l || r <= x) return -numeric_limits<ld>::
23         max();
24     if (r - l == 1) return arr[i](x);
25     return max({arr[i](x), query(x, i << 1, l, m),
26         query(x, i << 1 | 1, m, r)});
27 }
28 #undef m

```

## 4.7 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(); his.pop_back();
19     dsu[a] = a, sz[b] = s;
20 }
21 #define m ((l + r) >> 1)
22 void insert(int ql, int qr, P<int> x, int i = 1, int l
23     = 0, int r = q) {
24     // debug(ql, qr, x); return;
25     if (qr <= l || r <= ql) return;
26     if (ql <= l && r <= qr) {arr[i].push_back(x);
27         return;}
28     if (qr <= m)
29         insert(ql, qr, x, i << 1, l, m);
30     else if (m <= ql)
31         insert(ql, qr, x, i << 1 | 1, m, r);
32     else {
33         insert(ql, qr, x, i << 1, l, m);
34         insert(ql, qr, x, i << 1 | 1, m, r);
35     }
36 }
37 void traversal(V<int>& ans, int i = 1, int l = 0, int r
38     = q) {
39     int opcnt = 0;
40     // debug(i, l, r);
41     for (auto [a, b] : arr[i])
42         if (merge(a, b))
43             opcnt++, cnt--;
44     if (r - l == 1) ans[l] = cnt;
45     else {
46         traversal(ans, i << 1, l, m);
47         traversal(ans, i << 1 | 1, m, r);
48     }
49     while (opcnt--)
50         undo(), cnt++;
51     arr[i].clear();
52 }
53 #undef m
54 inline void solve() {
55     int n, m; cin>>n>>m>>q,q++;
56     dsu.resize(cnt = n), sz.assign(n, 1);

```



```

64 } } } } }
65
66
67 // Negative Cycle Detection
68 void NegCycleDetect() {
69     /* No Neg Cycle: NO
70     Exist Any Neg Cycle:
71     YES
72     v0 v1 v2 ... vk v0 */
73
74     vector<int> src;
75     for (int i = 1; i <= n; i++)
76         src.emplace_back(i);
77
78     SPFA(src);
79     // BellmanFord(src);
80
81     int ptr = -1;
82     for (int i = 1; i <= n; i++) if (negCycle[i])
83         { ptr = i; break; }
84
85     if (ptr == -1) { return cout << "NO" << endl, void
86         (); }
87
88     cout << "YES\n";
89     vector<int> ans;
90     vector<bool> vis(n+1, false);
91
92     while (true) {
93         ans.emplace_back(ptr);
94         if (vis[ptr]) break;
95         vis[ptr] = true;
96         ptr = pa[ptr];
97     }
98     reverse(ans.begin(), ans.end());
99
100     vis.assign(n+1, false);
101     for (auto& x : ans) {
102         cout << x << ' ';
103         if (vis[x]) break;
104         vis[x] = true;
105     }
106     cout << endl;
107 }
108
109 // Distance Calculation
110 void calcDis(int s) {
111     vector<int> src;
112     src.emplace_back(s);
113     SPFA(src);
114     // BellmanFord(src);
115
116     while (!q.empty()) q.pop();
117     for (int i = 1; i <= n; i++)
118         if (negCycle[i]) q.push(i);
119
120     while (!q.empty()) {
121         int u = q.front(); q.pop();
122         for (auto& e : g[u]) {
123             int v = e.first;
124             if (!negCycle[v]) {
125                 q.push(v);
126                 negCycle[v] = true;
127             }
128         }
129     }
130 }

```

## 6.2 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<maxn> 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++; dfs(v);
30             low[u] = min(low[u], low[v]);
31             if (!rt && low[v] >= dfn[u]) {
32                 // bcc found: u is ap
33                 isap[u] = true;
34                 popout(u);
35             }
36         } else {
37             // back edge
38             low[u] = min(low[u], dfn[v]);
39         }
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 }

```

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

```

## 6.4 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
6 stack<int> stk;
7 bitset<maxn> ins, vis;
8
9 int n, m;
10
11 void init() {
12     cin >> m >> n;
13     E.clear();
14     fill(g, g+maxn, vector<int>());
15     fill(low, low+maxn, INF);
16     memset(in, 0, sizeof(in));
17     instp = 1;
18     sccnt = 0;
19     memset(sccid, 0, sizeof(sccid));
20     ins.reset();
21     vis.reset();
22 }
23
24 inline int no(int u) {
25     return (u > n ? u-n : u+n);
26 }
27
28 int ecnt = 0;
29 inline void clause(int u, int v) {
30     E.eb(no(u)^v);
31     g[no(u)].eb(ecnt++);
32     E.eb(no(v)^u);
33     g[no(v)].eb(ecnt++);
34 }

```

```

35
36 void dfs(int u) {
37     in[u] = instp++;
38     low[u] = in[u];
39     stk.push(u);
40     ins[u] = true;
41
42     Each(e, g[u]) {
43         if (vis[e]) continue;
44         vis[e] = true;
45
46         int v = E[e]^u;
47         if (ins[v]) low[u] = min(low[u], in[v]);
48         else if (!in[v]) {
49             dfs(v);
50             low[u] = min(low[u], low[v]);
51         }
52     }
53
54     if (low[u] == in[u]) {
55         sccnt++;
56         while (!stk.empty()) {
57             int v = stk.top();
58             stk.pop();
59             ins[v] = false;
60             sccid[v] = sccnt;
61             if (u == v) break;
62         }
63     }
64 }
65
66 int main() {
67     WiwiHorz
68     init();
69
70     REP(i, m) {
71         char su, sv;
72         int u, v;
73         cin >> su >> u >> sv >> v;
74         if (su == '-') u = no(u);
75         if (sv == '-') v = no(v);
76         clause(u, v);
77     }
78
79     FOR(i, 1, 2*n+1, 1) {
80         if (!in[i]) dfs(i);
81     }
82
83     FOR(u, 1, n+1, 1) {
84         int du = no(u);
85         if (sccid[u] == sccid[du]) {
86             return cout << "IMPOSSIBLE\n", 0;
87         }
88     }
89
90     FOR(u, 1, n+1, 1) {
91         int du = no(u);
92         cout << (sccid[u] < sccid[du] ? '+' : '-') << '
93         ' << ' ';
94     }
95     cout << endl;
96
97     return 0;
98 }

```

## 6.5 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 {
11     vis[x] = 1;
12     for(int i : ed[x]) {
13         if(vis[i]) continue;

```



```

14     dfs(i);
15 }
16 pre.push_back(x);
17 }
18
19 void dfs2(int x)
20 {
21     vis[x] = 1;
22     SCC_cnt;
23     for(int i : ed_b[x]) {
24         if(vis[i]) continue;
25         dfs2(i);
26     }
27 }
28
29 void kosaraju()
30 {
31     for(int i = 1; i <= n; i++) {
32         if(!vis[i]) {
33             dfs(i);
34         }
35     }
36     SCC_cnt = 0;
37     vis = 0;
38     for(int i = n - 1; i >= 0; i--) {
39         if(!vis[pre[i]]) {
40             SCC_cnt++;
41             dfs2(pre[i]);
42         }
43     }
44 }

```

## 6.6 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; cin >> u >> v;
13        inodd[u] = inodd[u] ^ true;
14        inodd[v] = inodd[v] ^ true;
15        g[u].emplace_back(v);
16        g[v].emplace_back(u);
17    }
18    stack<int> stk;
19    void dfs(int u) {
20        while (!g[u].empty()) {
21            int v = g[u].back();
22            g[u].pop_back();
23            dfs(v);
24        }
25        stk.push(u);

```

## 6.7 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; cin >> u >> v;
13        g[u].emplace_back(v);
14        out[u]++, in[v]++;
15    }
16    for (int i = 1; i <= n; i++) {
17        if (i == 1 && out[i]-in[i] != 1) gg;
18        if (i == n && in[i]-out[i] != 1) gg;

```

```

19     if (i != 1 && i != n && in[i] != out[i]) gg;
20 } }
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 }
29 void solve() {
30     dfs(1)
31     for (int i = 1; i <= n; i++)
32         if ((int)g[i].size()) gg;
33     while (!stk.empty()) {
34         int u = stk.top();
35         stk.pop();
36         cout << u << ' ';
37     } }

```

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

```

## 6.9 Kth Shortest Path

```

1 // time: O(|E| \lg |E|+|V| \lg |V|+K)
2 // memory: O(|E| \lg |E|+|V|)
3 struct KSP{ // 1-base
4     struct nd{

```



```

5   int u,v; ll d;
6   nd(int ui=0,int vi=0,ll di=INF){ u=ui; v=vi; d=di;
7   }
8   };
9   struct heap{ nd* edge; int dep; heap* chd[4]; };
10  static int cmp(heap* a,heap* b)
11  { return a->edge->d > b->edge->d; }
12  struct node{
13      int v; ll d; heap* H; nd* E;
14      node(){}
15      node(ll _d,int _v,nd* _E){ d=_d; v=_v; E=_E; }
16      node(heap* _H,ll _d){ H=_H; d=_d; }
17      friend bool operator<(node a,node b)
18      { return a.d>b.d; }
19  };
20  int n,k,s,t,dst[N]; nd *nxt[N];
21  vector<nd*> g[N],rg[N]; heap *nullNd,*head[N];
22  void init(int _n,int _k,int _s,int _t){
23      n=_n; k=_k; s=_s; t=_t;
24      for(int i=1;i<=n;i++){
25          g[i].clear(); rg[i].clear();
26          nxt[i]=NULL; head[i]=NULL; dst[i]=-1;
27      }
28  }
29  void addEdge(int ui,int vi,ll di){
30      nd* e=new nd(ui,vi,di);
31      g[ui].push_back(e); rg[vi].push_back(e);
32  }
33  queue<int> dfsQ;
34  void dijkstra(){
35      while(dfsQ.size()) dfsQ.pop();
36      priority_queue<node> Q; Q.push(node(0,t,NULL));
37      while (!Q.empty()){
38          node p=Q.top(); Q.pop(); if(dst[p.v]!=-1) continue;
39          dst[p.v]=p.d; nxt[p.v]=p.E; dfsQ.push(p.v);
40          for(auto e:rg[p.v]) Q.push(node(p.d+e->d,e->u,e));
41      }
42  }
43  heap* merge(heap* curNd,heap* newNd){
44      if(curNd==nullNd) return newNd;
45      heap* root=new heap; memcpy(root,curNd,sizeof(heap));
46      ;
47      if(newNd->edge->d<curNd->edge->d){
48          root->edge=newNd->edge;
49          root->chd[2]=newNd->chd[2];
50          root->chd[3]=newNd->chd[3];
51          newNd->edge=curNd->edge;
52          newNd->chd[2]=curNd->chd[2];
53          newNd->chd[3]=curNd->chd[3];
54      }
55      if(root->chd[0]->dep<root->chd[1]->dep)
56          root->chd[0]=merge(root->chd[0],newNd);
57      else root->chd[1]=merge(root->chd[1],newNd);
58      root->dep=max(root->chd[0]->dep,
59                  root->chd[1]->dep)+1;
60      return root;
61  }
62  vector<heap*> V;
63  void build(){
64      nullNd=new heap; nullNd->dep=0; nullNd->edge=new nd
65      ;
66      fill(nullNd->chd,nullNd->chd+4,nullNd);
67      while(not dfsQ.empty()){
68          int u=dfsQ.front(); dfsQ.pop();
69          if(!nxt[u]) head[u]=nullNd;
70          else head[u]=head[nxt[u]->v];
71          V.clear();
72          for(auto&& e:g[u]){
73              int v=e->v;
74              if(dst[v]==-1) continue;
75              e->d+=dst[v]-dst[u];
76              if(nxt[u]!=e){
77                  heap* p=new heap; fill(p->chd,p->chd+4,nullNd);
78                  ;
79                  p->dep=1; p->edge=e; V.push_back(p);
80              }
81          }
82          if(V.empty()) continue;
83          make_heap(V.begin(),V.end(),cmp);
84  }
85  #define L(X) ((X<<1)+1)

```

```

81 #define R(X) ((X<<1)+2)
82 for(size_t i=0;i<V.size();i++){
83     if(L(i)<V.size()) V[i]->chd[2]=V[L(i)];
84     else V[i]->chd[2]=nullNd;
85     if(R(i)<V.size()) V[i]->chd[3]=V[R(i)];
86     else V[i]->chd[3]=nullNd;
87 }
88 head[u]=merge(head[u],V.front());
89 }
90 }
91 vector<ll> ans;
92 void first_K(){
93     ans.clear(); priority_queue<node> Q;
94     if(dst[s]==-1) return;
95     ans.push_back(dst[s]);
96     if(head[s]!=nullNd)
97         Q.push(node(head[s],dst[s]+head[s]->edge->d));
98     for(int _=1;_<k and not Q.empty();_++){
99         node p=Q.top(); q; Q.pop(); ans.push_back(p.d);
100         if(head[p.H->edge->v]!=nullNd){
101             q.H=head[p.H->edge->v]; q.d=p.d+q.H->edge->d;
102             Q.push(q);
103         }
104     }
105     for(int i=0;i<4;i++){
106         if(p.H->chd[i]!=nullNd){
107             q.H=p.H->chd[i];
108             q.d=p.d-p.H->edge->d+p.H->chd[i]->edge->d;
109             Q.push(q);
110         }
111     }
112     void solve(){ // ans[i] stores the i-th shortest path
113         dijkstra(); build();
114         first_K(); // ans.size() might less than k
115     }
116 } solver;

```

## 6.10 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.
- $\frac{x_u}{x_v} \leq c \Rightarrow \log x_u - \log x_v \leq \log c$

## 7 String

### 7.1 Rolling Hash

```

1 const ll C = 27;
2 inline int id(char c) {return c-'a'+1;}
3 struct RollingHash {
4     string s; int n; ll mod;
5     vector<ll> Cexp, hs;
6     RollingHash(string& _s, ll _mod):
7         s(_s), n((int)_s.size()), mod(_mod)
8     {
9         Cexp.assign(n, 0);
10        hs.assign(n, 0);
11        Cexp[0] = 1;
12        for (int i = 1; i < n; i++) {
13            Cexp[i] = Cexp[i-1] * C;

```

```

14         if (Cexp[i] >= mod) Cexp[i] %= mod;
15     }
16     hs[0] = id(s[0]);
17     for (int i = 1; i < n; i++) {
18         hs[i] = hs[i-1] * C + id(s[i]);
19         if (hs[i] >= mod) hs[i] %= mod;
20     }
21     inline ll query(int l, int r) {
22         ll res = hs[r] - (l ? hs[l-1] * Cexp[r-l+1] :
23             0);
24         res = (res % mod + mod) % mod;
25         return res; }

```

## 7.2 Trie

```

1 struct node {
2     int c[26]; ll cnt;
3     node(): cnt(0) {memset(c, 0, sizeof(c));}
4     node(ll x): cnt(x) {memset(c, 0, sizeof(c));}
5 };
6 struct Trie {
7     vector<node> t;
8     void init() {
9         t.clear();
10        t.emplace_back(node());
11    }
12    void insert(string s) { int ptr = 0;
13        for (auto& i : s) {
14            if (!t[ptr].c[i-'a']) {
15                t.emplace_back(node());
16                t[ptr].c[i-'a'] = (int)t.size()-1; }
17            ptr = t[ptr].c[i-'a']; }
18        t[ptr].cnt++; }
19 } trie;

```

## 7.3 KMP

```

1 int n, m;
2 string s, p;
3 vector<int> f;
4 void build() {
5     f.clear(); f.resize(m, 0);
6     int ptr = 0; for (int i = 1; i < m; i++) {
7         while (ptr && p[i] != p[ptr]) ptr = f[ptr-1];
8         if (p[i] == p[ptr]) ptr++;
9         f[i] = ptr;
10    }
11    void init() {
12        cin >> s >> p;
13        n = (int)s.size();
14        m = (int)p.size();
15        build(); }
16    void solve() {
17        int ans = 0, pi = 0;
18        for (int si = 0; si < n; si++) {
19            while (pi && s[si] != p[pi]) pi = f[pi-1];
20            if (s[si] == p[pi]) pi++;
21            if (pi == m) ans++, pi = f[pi-1];
22        }
23    cout << ans << endl; }

```

## 7.4 Z Value

```

1 string is, it, s;
2 int n; vector<int> z;
3 void init() {
4     cin >> is >> it;
5     s = it+'0'+is;
6     n = (int)s.size();
7     z.resize(n, 0); }
8 void solve() {
9     int ans = 0; z[0] = n;
10    for (int i = 1, l = 0, r = 0; i < n; i++) {
11        if (i <= r) z[i] = min(z[i-l], r-i+1);
12        while (i+z[i] < n && s[z[i]] == s[i+z[i]]) z[i]
13            ++;
14        if (i+z[i]-1 > r) l = i, r = i+z[i]-1;
15        if (z[i] == (int)it.size()) ans++;

```

```

15     }
16     cout << ans << endl; }

```

## 7.5 Manacher

```

1 int n; string S, s;
2 vector<int> m;
3 void manacher() {
4     s.clear(); s.resize(2*n+1, '.');
5     for (int i = 0, j = 1; i < n; i++, j += 2) s[j] = S[i];
6     m.clear(); m.resize(2*n+1, 0);
7     // m[i] := max k such that s[i-k, i+k] is palindrome
8     int mx = 0, mxk = 0;
9     for (int i = 1; i < 2*n+1; i++) {
10        if (mx-(i-mx) >= 0) m[i] = min(m[mx-(i-mx)], mx+mxk
11            -i);
12        while (0 <= i-m[i]-1 && i+m[i]+1 < 2*n+1 &&
13            s[i-m[i]-1] == s[i+m[i]+1]) m[i]++;
14        if (i+m[i] > mx+mxk) mx = i, mxk = m[i];
15    }
16    void init() { cin >> S; n = (int)S.size(); }
17    void solve() {
18        manacher();
19        int mx = 0, ptr = 0;
20        for (int i = 0; i < 2*n+1; i++) if (mx < m[i])
21            { mx = m[i]; ptr = i; }
22        for (int i = ptr-mx; i <= ptr+mx; i++)
23            if (s[i] != '.') cout << s[i];
24    }

```

## 7.6 Suffix Array

```

1 #define F first
2 #define S second
3 struct SuffixArray { // don't forget s += "$";
4     int n; string s;
5     vector<int> suf, lcp, rk;
6     vector<int> cnt, pos;
7     vector<pair<pii, int>> buc[2];
8     void init(string _s) {
9         s = _s; n = (int)s.size();
10        // resize(n): suf, rk, cnt, pos, lcp, buc[0~1]
11    }
12    void radix_sort() {
13        for (int t : {0, 1}) {
14            fill(cnt.begin(), cnt.end(), 0);
15            for (auto& i : buc[t]) cnt[ (t ? i.F.F : i.
16                F.S) ]++;
17            for (int i = 0; i < n; i++)
18                pos[i] = (i ? 0 : pos[i-1] + cnt[i-1])
19                    ;
20            for (auto& i : buc[t])
21                buc[t^1][pos[ (t ? i.F.F : i.F.S) ]++]
22                    = i;
23        }
24        bool fill_suf() {
25            bool end = true;
26            for (int i = 0; i < n; i++) suf[i] = buc[0][i].
27                S;
28            rk[suf[0]] = 0;
29            for (int i = 1; i < n; i++) {
30                int dif = (buc[0][i].F != buc[0][i-1].F);
31                end &= dif;
32                rk[suf[i]] = rk[suf[i-1]] + dif;
33            } return end;
34        }
35        void sa() {
36            for (int i = 0; i < n; i++)
37                buc[0][i] = make_pair(make_pair(s[i], s[i]),
38                    i);
39            sort(buc[0].begin(), buc[0].end());
40            if (fill_suf()) return;
41            for (int k = 0; (1<<k) < n; k++) {
42                for (int i = 0; i < n; i++)
43                    buc[0][i] = make_pair(make_pair(rk[i],
44                        rk[(i + (1<<k)) % n]), i);
45                radix_sort();
46                if (fill_suf()) return;
47            }
48            void LCP() { int k = 0;

```

```

43     for (int i = 0; i < n-1; i++) {
44         if (rk[i] == 0) continue;
45         int pi = rk[i];
46         int j = suf[pi-1];
47         while (i+k < n && j+k < n && s[i+k] == s[j+
48             k]) k++;
49         lcp[pi] = k;
50         k = max(k-1, 0);
51     }
52 };
SuffixArray suffixarray;

```

## 7.7 SA-IS

```

1  const int N=300010;
2  struct SA{
3      #define REP(i,n) for(int i=0;i<int(n);i++)
4      #define REP1(i,a,b) for(int i=a;i<=int(b);i++)
5      bool _t[N*2]; int _s[N*2],_sa[N*2];
6      int _c[N*2],x[N],_p[N],_q[N*2],hei[N],r[N];
7      int operator [](int i){ return _sa[i]; }
8      void build(int *s,int n,int m){
9          memcpy(_s,s,sizeof(int)*n);
10         sais(_s,_sa,_p,_q,_t,_c,n,m); mkhei(n);
11     }
12     void mkhei(int n){
13         REP(i,n) r[_sa[i]]=i;
14         hei[0]=0;
15         REP(i,n) if(r[i]) {
16             int ans=i>0?max(hei[r[i-1]]-1,0):0;
17             while(_s[i+ans]==_s[_sa[r[i]-1]+ans]) ans++;
18             hei[r[i]]=ans;
19         }
20     }
21     void sais(int *s,int *sa,int *p,int *q,bool *t,int *c
22         ,int n,int z){
23         bool uniq=t[n-1]=true,neq;
24         int nn=0,nmxz=-1,*nsa=sa+n,*ns=s+n,lst=-1;
25         #define MS0(x,n) memset((x),0,n*sizeof(*(x)))
26         #define MAGIC(XD) MS0(sa,n);\
27         memcpy(x,c,sizeof(int)*z); XD;\
28         memcpy(x+1,c,sizeof(int)*(z-1));\
29         REP(i,n) if(sa[i]&&!t[sa[i]-1]) sa[x[s[sa[i]-1]]++]=sa[
30             i]-1;\
31         memcpy(x,c,sizeof(int)*z);\
32         for(int i=n-1;i>=0;i--) if(sa[i]&&t[sa[i]-1]) sa[--x[s[
33             sa[i]-1]]]=sa[i]-1;
34         MS0(c,z); REP(i,n) uniq&=++c[s[i]]<2;
35         REP(i,z-1) c[i+1]+=c[i];
36         if(uniq) { REP(i,n) sa[--c[s[i]]]=i; return; }
37         for(int i=n-2;i>=0;i--){
38             t[i]=(s[i]==s[i+1]?t[i+1]:s[i]<s[i+1]);
39             MAGIC(REP1(i,1,n-1) if(t[i]&&!t[i-1]) sa[--x[s[i
40                 ]]]]=p[q[i]=nn++]=i);
41             REP(i,n) if(sa[i]&&t[sa[i]]&&t[sa[i]-1]){
42                 neq=lst<0||memcmp(s+sa[i],s+lst,(p[q[sa[i]]+1]-sa
43                     [i])*sizeof(int));
44                 ns[q[lst=sa[i]]]=nmxz+=neq;
45             }
46         }
47         sais(ns,nsa,p+nn,q+n,t+n,c+z,nn,nmxz+1);
48         MAGIC(for(int i=nn-1;i>=0;i--) sa[--x[s[p[nsa[i
49             ]]]]=p[nsa[i]]);
50     }
51 };
52 int H[N],SA[N],RA[N];
53 void suffix_array(int* ip,int len){
54     // should padding a zero in the back
55     // ip is int array, len is array length
56     // ip[0..n-1] != 0, and ip[len]=0
57     ip[len++]=0; sa.build(ip,len,128);
58     memcpy(H,sa.hei+1,len<2); memcpy(SA,sa._sa+1,len<2);
59     for(int i=0;i<len;i++) RA[i]=sa.r[i]-1;
60     // resulting height, sa array \in [0,len)
61 }

```

## 7.8 Minimum Rotation

```

1 //rotate(begin(s), begin(s)+minRotation(s), end(s))
2 int minRotation(string s) {

```

```

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

```

## 7.9 Aho Corasick

```

1 struct ACautomata{
2     struct Node{
3         int cnt;
4         Node *go[26], *fail, *dic;
5         Node(){
6             cnt = 0; fail = 0; dic=0;
7             memset(go,0,sizeof(go));
8         }
9     }pool[1048576],*root;
10    int nMem;
11    Node* new_Node(){
12        pool[nMem] = Node();
13        return &pool[nMem++];
14    }
15    void init() { nMem = 0; root = new_Node(); }
16    void add(const string &str) { insert(root,str,0); }
17    void insert(Node *cur, const string &str, int pos){
18        for(int i=pos;i<str.size();i++){
19            if(!cur->go[str[i]-'a'])
20                cur->go[str[i]-'a'] = new_Node();
21            cur=cur->go[str[i]-'a'];
22        }
23        cur->cnt++;
24    }
25    void make_fail(){
26        queue<Node*> que;
27        que.push(root);
28        while (!que.empty()){
29            Node* fr=que.front(); que.pop();
30            for (int i=0; i<26; i++){
31                if (fr->go[i]){
32                    Node *ptr = fr->fail;
33                    while (ptr && !ptr->go[i]) ptr = ptr->fail;
34                    fr->go[i]->fail=ptr=(ptr?ptr->go[i]:root);
35                    fr->go[i]->dic=(ptr->cnt?ptr:ptr->dic);
36                    que.push(fr->go[i]);
37                } } } }
38 }AC;

```

## 8 Geometry

### 8.1 Basic Operations

```

1 typedef long long T;
2 // typedef long double T;
3 const long double eps = 1e-8;
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)
20         { return x < a.x || (x == a.x && y < a.y); }
21     //return sgn(x-a.x) < 0 || (sgn(x-a.x) == 0 && sgn(y-a.
22         y) < 0); }
23     bool operator==(Pt a)

```

```

23 { return sgn(x-a.x) == 0 && sgn(y-a.y) == 0; }
24 };
25
26 Pt mv(Pt a, Pt b) { return b-a; }
27 T len2(Pt a) { return a*a; }
28 T dis2(Pt a, Pt b) { return len2(b-a); }
29
30 short ori(Pt a, Pt b) { return ((a^b)>0) - ((a^b)<0); }
31 bool onseg(Pt p, Pt l1, Pt l2) {
32     Pt a = mv(p, l1), b = mv(p, l2);
33     return ((a^b) == 0) && ((a*b) <= 0);
34 }

```

## 8.2 InPoly

```

1 short inPoly(Pt p) {
2     // 0=Bound 1=In -1=Out
3     REP(i, n) if (onseg(p, E[i], E[(i+1)%n])) return 0;
4     int cnt = 0;
5     REP(i, n) if (banana(p, Pt(p.x+1, p.y+2e9),
6                          E[i], E[(i+1)%n])) cnt ^= 1;
7     return (cnt ? 1 : -1);
8 }

```

## 8.3 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(ALL(E), [&](const Pt& a, const Pt& b){
7     if (ud(a) != ud(b)) return ud(a) < ud(b);
8     return (a^b) > 0;
9 });

```

## 8.4 Line Intersect Check

```

1 inline bool banana(Pt p1, Pt p2, Pt q1, Pt q2) {
2     if (onseg(p1, q1, q2) || onseg(p2, q1, q2) ||
3         onseg(q1, p1, p2) || onseg(q2, p1, p2)) {
4         return true;
5     }
6     Pt p = mv(p1, p2), q = mv(q1, q2);
7     return (ori(p, mv(p1, q1)) * ori(p, mv(p1, q2)) < 0 &&
8             ori(q, mv(q1, p1)) * ori(q, mv(q1, p2)) < 0);
9 }

```

## 8.5 Line Intersection

```

1 // T: long double
2 Pt bananaPoint(Pt p1, Pt p2, Pt q1, Pt q2) {
3     if (onseg(q1, p1, p2)) return q1;
4     if (onseg(q2, p1, p2)) return q2;
5     if (onseg(p1, q1, q2)) return p1;
6     if (onseg(p2, q1, q2)) return p2;
7     double s = abs(mv(p1, p2) ^ mv(p1, q1));
8     double t = abs(mv(p1, p2) ^ mv(p1, q2));
9     return q2 * (s/(s+t)) + q1 * (t/(s+t));
10 }

```

## 8.6 Convex Hull

```

1 vector<Pt> hull;
2 void convexHull() {
3     hull.clear(); sort(ALL(E));
4     REP(t, 2) {
5         int b = SZ(hull);
6         Each(ei, E) {
7             while (SZ(hull) - b >= 2 &&
8                     ori(mv(hull[SZ(hull)-2], hull.back()),
9                         mv(hull[SZ(hull)-2], ei)) == -1) {
10                 hull.pop_back();
11             }
12             hull.pb(ei);
13         }
14         hull.pop_back();
15         reverse(ALL(E));
16     } }

```

## 8.7 Lower Concave Hull

```

1 struct Line {
2     mutable ll m, b, p;
3     bool operator<(const Line& o) const { return m < o.m; }
4     bool operator<(ll x) const { return p < x; }
5 };
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 isect(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    }
31 };

```

## 8.8 Polygon Area

```

1 T dbarea(vector<Pt>& e) {
2     ll res = 0;
3     REP(i, SZ(e)) res += e[i]^e[(i+1)%SZ(e)];
4     return abs(res);
5 }

```

## 8.9 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:

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

## 8.10 Minimum Enclosing Circle

```

1 Pt circumcenter(Pt A, Pt B, Pt C) {
2     // a1(x-A.x) + b1(y-A.y) = c1
3     // a2(x-A.x) + b2(y-A.y) = c2
4     // solve using Cramer's rule
5     T a1 = B.x-A.x, b1 = B.y-A.y, c1 = dis2(A, B)/2.0;
6     T a2 = C.x-A.x, b2 = C.y-A.y, c2 = dis2(A, C)/2.0;
7     T D = Pt(a1, b1) ^ Pt(a2, b2);
8     T Dx = Pt(c1, b1) ^ Pt(c2, b2);
9     T Dy = Pt(a1, c1) ^ Pt(a2, c2);
10    if (D == 0) return Pt(-INF, -INF);
11    return A + Pt(Dx/D, Dy/D);
12 }
13 Pt center; T r2;
14 void minEncloseCircle() {
15     mt19937 gen(chrono::steady_clock::now().
16         time_since_epoch().count());
17     shuffle(ALL(E), gen);
18     center = E[0], r2 = 0;
19     for (int i = 0; i < n; i++) {
20         if (dis2(center, E[i]) <= r2) continue;
21         center = E[i], r2 = 0;
22         for (int j = 0; j < i; j++) {

```

```

23     if (dis2(center, E[j]) <= r2) continue;
24     center = (E[i] + E[j]) / 2.0;
25     r2 = dis2(center, E[i]);
26     for (int k = 0; k < j; k++) {
27         if (dis2(center, E[k]) <= r2) continue;
28         center = circumcenter(E[i], E[j], E[k]);
29         r2 = dis2(center, E[i]);
30     }
31 }
32 }

```

## 8.11 PolyUnion

```

1 struct PY{
2     int n; Pt pt[5]; double area;
3     Pt& operator[](const int x){ return pt[x]; }
4     void init(){ //n,pt[0~n-1] must be filled
5         area=pt[n-1]^pt[0];
6         for(int i=0;i<n-1;i++) area+=pt[i]^pt[i+1];
7         if((area/=2)<0)reverse(pt,pt+n),area=-area;
8     }
9 };
10 PY py[500]; pair<double,int> c[5000];
11 inline double segP(Pt &p,Pt &p1,Pt &p2){
12     if(dcmp(p1.x-p2.x)==0) return (p.y-p1.y)/(p2.y-p1.y);
13     return (p.x-p1.x)/(p2.x-p1.x);
14 }
15 double polyUnion(int n){ //py[0~n-1] must be filled
16     int i,j,ii,jj,ta,tb,r,d; double z,w,s,sum=0,tc,td;
17     for(i=0;i<n;i++) py[i][py[i].n]=py[i][0];
18     for(i=0;i<n;i++){
19         for(ii=0;ii<py[i].n;ii++){
20             r=0;
21             c[r++]=make_pair(0.0,0); c[r++]=make_pair(1.0,0);
22             for(j=0;j<n;j++){
23                 if(i==j) continue;
24                 for(jj=0;jj<py[j].n;jj++){
25                     ta=dcmp(tri(py[i][ii],py[i][ii+1],py[j][jj]))
26                     ;
27                     tb=dcmp(tri(py[i][ii],py[i][ii+1],py[j][jj
28                     +1]));
29                     if(ta==0 && tb==0){
30                         if((py[j][jj+1]-py[j][jj])*(py[i][ii+1]-py[
31                         i][ii])>0&&j<i){
32                             c[r++]=make_pair(segP(py[j][jj],py[i][ii
33                             ],py[i][ii+1]),1);
34                             c[r++]=make_pair(segP(py[j][jj+1],py[i][
35                             ii],py[i][ii+1]),-1);
36                         }
37                     }else if(ta>0 && tb<0){
38                         tc=tri(py[j][jj],py[j][jj+1],py[i][ii]);
39                         td=tri(py[j][jj],py[j][jj+1],py[i][ii+1]);
40                         c[r++]=make_pair(tc/(tc-td),1);
41                     }else if(ta<0 && tb>0){
42                         tc=tri(py[j][jj],py[j][jj+1],py[i][ii]);
43                         td=tri(py[j][jj],py[j][jj+1],py[i][ii+1]);
44                         c[r++]=make_pair(tc/(tc-td),-1);
45                     }
46                 }
47             }
48             sort(c,c+r);
49             z=min(max(c[0].first,0.0),1.0); d=c[0].second; s
50             =0;
51             for(j=1;j<r;j++){
52                 w=min(max(c[j].first,0.0),1.0);
53                 if(!d) s+=w-z;
54                 d+=c[j].second; z=w;
55             }
56             sum+=(py[i][ii]^py[i][ii+1])*s;
57         }
58     }
59     return sum/2;
60 }

```

## 8.12 Minkowski Sum

```

1 /* convex hull Minkowski Sum*/
2 #define INF 1000000000000000LL
3 int pos( const Pt& tp ){
4     if( tp.Y == 0 ) return tp.X > 0 ? 0 : 1;
5     return tp.Y > 0 ? 0 : 1;
6 }

```

```

7 #define N 300030
8 Pt pt[ N ], qt[ N ], rt[ N ];
9 LL Lx,Rx;
10 int dn,un;
11 inline bool cmp( Pt a, Pt b ){
12     int pa=pos( a ),pb=pos( b );
13     if(pa==pb) return (a^b)>0;
14     return pa<pb;
15 }
16 int minkowskiSum(int n,int m){
17     int i,j,r,p,q,fi,fj;
18     for(i=1,p=0;i<n;i++){
19         if( pt[i].Y<pt[p].Y ||
20             (pt[i].Y==pt[p].Y && pt[i].X<pt[p].X) ) p=i; }
21     for(i=1,q=0;i<m;i++){
22         if( qt[i].Y<qt[q].Y ||
23             (qt[i].Y==qt[q].Y && qt[i].X<qt[q].X) ) q=i; }
24     rt[0]=pt[p]+qt[q];
25     r=1; i=p; j=q; fi=fj=0;
26     while(1){
27         if((fj&&j==q) ||
28             ( (!fi||i==p) &&
29               cmp(pt[(p+1)%n]-pt[p],qt[(q+1)%n]-qt[q]) ) ){
30             rt[r]=rt[r-1]+pt[(p+1)%n]-pt[p];
31             p=(p+1)%n;
32             fi=1;
33         }else{
34             rt[r]=rt[r-1]+qt[(q+1)%n]-qt[q];
35             q=(q+1)%m;
36             fj=1;
37         }
38         if(r<=1 || ((rt[r]-rt[r-1])^(rt[r-1]-rt[r-2]))
39             !=0) r++;
40         else rt[r-1]=rt[r];
41         if(i==p && j==q) break;
42     }
43     return r-1;
44 }
45 void initInConvex(int n){
46     int i,p,q;
47     LL Ly,Ry;
48     Lx=INF; Rx=-INF;
49     for(i=0;i<n;i++){
50         if(pt[i].X<Lx) Lx=pt[i].X;
51         if(pt[i].X>Rx) Rx=pt[i].X;
52     }
53     Ly=Ry=INF;
54     for(i=0;i<n;i++){
55         if(pt[i].X==Lx && pt[i].Y<Ly){ Ly=pt[i].Y; p=i; }
56         if(pt[i].X==Rx && pt[i].Y>Ry){ Ry=pt[i].Y; q=i; }
57     }
58     for(dn=0,i=p;i!=q;i=(i+1)%n){ qt[dn++]=pt[i]; }
59     qt[dn]=pt[q]; Ly=Ry=-INF;
60     for(i=0;i<n;i++){
61         if(pt[i].X==Lx && pt[i].Y>Ly){ Ly=pt[i].Y; p=i; }
62         if(pt[i].X==Rx && pt[i].Y>Ry){ Ry=pt[i].Y; q=i; }
63     }
64     for(un=0,i=p;i!=q;i=(i+n-1)%n){ rt[un++]=pt[i]; }
65     rt[un]=pt[q];
66 }
67 inline int inConvex(Pt p){
68     int L,R,M;
69     if(p.X<Lx || p.X>Rx) return 0;
70     L=0;R=dn;
71     while(L<R-1){ M=(L+R)/2;
72         if(p.X<qt[M].X) R=M; else L=M; }
73     if(tri(qt[L],qt[R],p)<0) return 0;
74     L=0;R=un;
75     while(L<R-1){ M=(L+R)/2;
76         if(p.X<rt[M].X) R=M; else L=M; }
77     if(tri(rt[L],rt[R],p)>0) return 0;
78     return 1;
79 }
80 int main(){
81     int n,m,i;
82     Pt p;
83     scanf("%d",&n);

```

```

83     for(i=0;i<n;i++) scanf("%lld%lld",&pt[i].X,&pt[i].Y);
84     scanf("%d",&m);
85     for(i=0;i<m;i++) scanf("%lld%lld",&qt[i].X,&qt[i].Y);
86     n=minkowskiSum(n,m);
87     for(i=0;i<n;i++) pt[i]=rt[i];
88     scanf("%d",&m);
89     for(i=0;i<m;i++) scanf("%lld%lld",&qt[i].X,&qt[i].Y);
90     n=minkowskiSum(n,m);
91     for(i=0;i<n;i++) pt[i]=rt[i];
92     initInConvex(n);
93     scanf("%d",&m);
94     for(i=0;i<m;i++){
95         scanf("%lld %lld",&p.X,&p.Y);
96         p.X*=3; p.Y*=3;
97         puts(inConvex(p)?"YES":"NO");
98     }
99 }

```

## 9 Number Theory

### 9.1 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) 3(011)
11      4(100) 5(101) 6(110) 7(111)
12      1th recursion 0(000) 2(010) 4(100) 6(110)
13      | 1(011) 3(011) 5(101) 7(111)
14      2th recursion 0(000) 4(100) | 2(010) 6(110)
15      | 1(011) 5(101) | 3(011) 7(111)
16      3th recursion 0(000) | 4(100) | 2(010) | 6(110)
17      | 1(011) | 5(101) | 3(011) | 7(111)
18      All the bits are reversed => We can save the
19      reverse of the numbers in an array!
20      */
21      int n, rev[NN];
22      cp omega[NN], iomega[NN];
23      void init(int n_){
24          n = n_;
25          for(int i = 0; i < n; i++){
26              //Calculate the nth roots of unity
27              omega[i] = cp(cos(2*pi*i/n_), sin(2*pi*i/n_));
28              iomega[i] = conj(omega[i]);
29          }
30          int k = __lg(n_);
31          for(int i = 0; i < n; i++){
32              int t = 0;
33              for(int j = 0; j < k; j++){
34                  if(i & (1<<j)) t |= (1<<(k-j-1));
35              }
36              rev[i] = t;
37          }
38      }
39
40      void transform(vector<cp> &a, cp* xomega){
41          for(int i = 0; i < n; i++){
42              if(i < rev[i]) swap(a[i], a[rev[i]]);
43          }
44          for(int len = 2; len <= n; len <= 1){
45              int mid = len >> 1;
46              int r = n/len;
47              for(int j = 0; j < n; j += len)
48                  for(int i = 0; i < mid; i++){
49                      cp tmp = xomega[r*i] * a[j+mid+i];
50                      a[j+mid+i] = a[j+i] - tmp;
51                      a[j+i] = a[j+i] + tmp;
52                  }
53          }
54      }
55  };
56
57  void fft(vector<cp> &a){ transform(a, omega); }
58  void ifft(vector<cp> &a){ transform(a, iomega); }
59  for(int i = 0; i < n; i++) a[i] /= n;
60  } FFT;
61
62  const int MAXN = 262144;
63  // (must be 2^k)
64  // 262144, 524288, 1048576, 2097152, 4194304
65  // before any usage, run pre_fft() first
66  typedef long double ld;
67  typedef complex<ld> cplx; //real() ,imag()
68  const ld PI = acos(-1);
69  const cplx I(0, 1);
70  cplx omega[MAXN+1];
71  void pre_fft(){
72      for(int i=0; i<=MAXN; i++) {
73          omega[i] = exp(i * 2 * PI / MAXN * I);
74      }
75  }
76
77  // n must be 2^k
78  void fft(int n, cplx a[], bool inv=false){
79      int basic = MAXN / n;
80      int theta = basic;
81      for (int m = n; m >= 2; m >= 1) {
82          int mh = m >> 1;
83          for (int i = 0; i < mh; i++) {
84              cplx w = omega[inv ? MAXN - (i * theta %
85                  MAXN) : i * theta % MAXN];
86              for (int j = i; j < n; j += m) {
87                  int k = j + mh;
88                  cplx x = a[j] - a[k];
89                  a[j] += a[k];
90                  a[k] = w * x;
91              }
92          }
93          theta = (theta * 2) % MAXN;
94      }
95      int i = 0;
96      for (int j = 1; j < n - 1; j++) {
97          for (int k = n >> 1; k > (i ^= k); k >>= 1);
98          if (j < i) swap(a[i], a[j]);
99      }
100      if(inv) {
101          for (i = 0; i < n; i++) a[i] /= n;
102      }
103  }
104
105  cplx arr[MAXN + 1];
106  inline void mul(int _n, long long a[], int _m, long long b
107  [], long long ans[]){
108      int n=1, sum = _n + _m - 1;
109      while(n < sum) n <= 1;
110      for(int i = 0; i < n; i++) {
111          double x = (i < _n ? a[i] : 0), y = (i < _m ? b[i]
112              : 0);
113          arr[i] = complex<double>(x + y, x - y);
114      }
115      fft(n, arr);
116      for(int i = 0; i < n; i++) arr[i]=arr[i]*arr[i];
117      fft(n, arr, true);
118      for(int i=0; i<sum; i++) ans[i]=(long long int)(arr[i
119          ].real() / 4 + 0.5);
120  }
121
122  long long a[MAXN];
123  long long b[MAXN];
124  long long ans[MAXN];
125  int a_length;
126  int b_length;

```

### 9.2 Pollard's rho

```

1  from itertools import count
2  from math import gcd
3  from sys import stdin
4
5  for s in stdin:
6      number, x = int(s), 2
7      break2 = False
8      for cycle in count(1):
9          y = x

```



```

10     if break2:
11         break
12     for i in range(1 << cycle):
13         x = (x * x + 1) % number
14         factor = gcd(x - y, number)
15         if factor > 1:
16             print(factor)
17             break2 = True
18             break

```

### 9.3 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 : pirmes <= 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; int t=0;
22     while(!(u&1)) u>>=1, t++;
23     while(s--){
24         ll a=randll()%(n-1)+1;
25         if(witness(a,n,u,t)) return 0;
26     }
27     return 1;
28 }

```

### 9.4 Fast Power

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

### 9.5 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 }
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 }

```

### 9.6 Mu + Phi

```

1 const int maxn = 1e6 + 5;
2 ll f[maxn];
3 vector<int> lpf, prime;
4 void build() {
5     lpf.clear(); lpf.resize(maxn, 1);
6     prime.clear();
7     f[1] = ...; /* mu[1] = 1, phi[1] = 1 */

```

```

8 for (int i = 2; i < maxn; i++) {
9     if (lpf[i] == 1) {
10         lpf[i] = i; prime.emplace_back(i);
11         f[i] = ...; /* mu[i] = 1, phi[i] = i-1 */
12     }
13     for (auto& j : prime) {
14         if (i*j >= maxn) break;
15         lpf[i*j] = j;
16         if (i % j == 0) f[i*j] = ...; /* 0, phi[i]*j
17                                     */
18         else f[i*j] = ...; /* -mu[i], phi[i]*phi[j] */
19         if (j >= lpf[i]) break;
20     }
21 }

```

### 9.7 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:**  
 $\sigma_x(n) = \sum_{d|n} d^x$ .  $n = \prod_{i=1}^r p_i^{a_i}$ .  
 $\sigma_x(n) = \prod_{i=1}^r \frac{p_i^{(a_i+1)x} - 1}{p_i^x - 1}$  if  $x \neq 0$ .  $\sigma_0(n) = \prod_{i=1}^r (a_i + 1)$ .
- Chinese remainder theorem (Coprime Moduli):**  
 $x \equiv a_i \pmod{m_i}$ .  
 $M = \prod m_i$ .  $M_i = M/m_i$ .  $t_i = M_i^{-1}$ .  
 $x = kM + \sum a_i t_i M_i$ ,  $k \in \mathbb{Z}$ .
- 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:**  $(f * g)(n) = \sum_{d|n} f(d)g(n/d)$
- Important Multiplicative Functions + Properties:**
  - $\epsilon(n) = [n = 1]$
  - $1(n) = 1$
  - $id(n) = n$
  - $\mu(n) = 0$  if  $n$  has squared prime factor
  - $\mu(n) = (-1)^k$  if  $n = p_1 p_2 \cdots p_k$
  - $\epsilon = \mu * 1$
  - $\phi = \mu * id$
  - $[n = 1] = \sum_{d|n} \mu(d)$
  - $[gcd = 1] = \sum_{d|gcd} \mu(d)$
- Möbius inversion:**  $f = g * 1 \Leftrightarrow g = f * \mu$



## 9.8 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
71 template<typename T>
72 vector<T>& operator-=(vector<T>& a, const vector<T>& b) {
73     {
74         if (siz(a) < siz(b)) a.resize(siz(b));
75         for (int i = 0; i < min(siz(a), siz(b)); i++) {
76             a[i] -= b[i];
77             a[i] += a[i] < 0 ? MOD : 0;
78         }
79         return a;
80     }
81 }
82
83 template<typename T>
84 vector<T> operator-(const vector<T>& a) {
85     vector<T> ret(siz(a));
86     for (int i = 0; i < siz(a); i++) {
87         ret[i] = -a[i] < 0 ? -a[i] + MOD : -a[i];
88     }
89     return ret;
90 }
91
92 vector<ll> X, iX;
93 vector<int> rev;
94
95 void init_ntt() {
96     X.clear(); X.resize(maxn, 1); // x1 = g^((p-1)/n)
97     iX.clear(); iX.resize(maxn, 1);
98
99     ll u = pw(g, (MOD-1)/maxn);
100    ll iu = pw(u, MOD-2);
101
102    for (int i = 1; i < maxn; i++) {
103        X[i] = X[i-1] * u;
104        iX[i] = iX[i-1] * iu;
105        if (X[i] >= MOD) X[i] %= MOD;
106        if (iX[i] >= MOD) iX[i] %= MOD;
107    }
108
109    rev.clear(); rev.resize(maxn, 0);
110    for (int i = 1, hb = -1; i < maxn; i++) {
111        if (!(i & (i-1))) hb++;
112        rev[i] = rev[i ^ (1<<hb)] | (1<<(maxk-hb-1));
113    }
114 }
115
116 template<typename T>
117 void NTT(vector<T>& a, bool inv=false) {
118
119     int _n = (int)a.size();
120     int k = __lg(_n) + ((1<<__lg(_n)) != _n);
121     int n = 1<<k;
122     a.resize(n, 0);
123
124     short shift = maxk-k;
125     for (int i = 0; i < n; i++)
126         if (i > (rev[i]>>shift))
127             swap(a[i], a[rev[i]>>shift]);
128
129     for (int len = 2, half = 1, div = maxn>>1; len <= n; len<=1, half<=1, div>>=1) {
130         for (int i = 0; i < n; i += len) {
131             for (int j = 0; j < half; j++) {
132                 T u = a[i+j];
133                 T v = a[i+j+half] * (inv ? iX[j*div] : X[j*div]) % MOD;
134                 a[i+j] = (u+v >= MOD ? u+v-MOD : u+v);
135                 a[i+j+half] = (u-v < 0 ? u-v+MOD : u-v);
136             }
137         }
138     }
139
140     if (inv) {
141         T dn = pw(n, MOD-2);
142         for (auto& x : a) {
143             x *= dn;
144             if (x >= MOD) x %= MOD;
145         }
146     }
147 }
148
149 template<typename T>
150 inline void resize(vector<T>& a) {
151     int cnt = (int)a.size();
152     for (; cnt > 0; cnt--) if (a[cnt-1]) break;
153     a.resize(max(cnt, 1));
154 }
155
156 template<typename T>
157 vector<T>& operator*=(vector<T>& a, vector<T> b) {
158     int na = (int)a.size();
159     int nb = (int)b.size();
160     a.resize(na + nb - 1, 0);
161     b.resize(na + nb - 1, 0);
162
163     NTT(a); NTT(b);
164     for (int i = 0; i < (int)a.size(); i++) {
165         a[i] *= b[i];
166         if (a[i] >= MOD) a[i] %= MOD;
167     }
168 }

```

```

158     }
159     NTT(a, true);
160
161     resize(a);
162     return 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 -> 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-3
195         -m+1));
196     reverse(rb.begin(), rb.end()); rb.resize(min(m+1, n-5
197         -m+1));
198
199     inv(rb, n-m+1);
200
201     vector<T> q = move(ra);
202     q *= rb;
203     q.resize(n-m+1);
204     reverse(q.begin(), q.end());
205
206     q *= b;
207     a -= q;
208     resize(a);
209 }
210
211 /* Kitamasa Method (Fast Linear Recurrence):
212 Find a[K] (Given a[j] = c[0]a[j-N] + ... + c[N-1]a[j-1])
213 Let B(x) = x^N - c[N-1]x^(N-1) - ... - c[1]x^1 - c[0]
214 Let R(x) = x^K mod B(x) (get x^K using fast pow and use poly mod to get R(x))
215 Let r[i] = the coefficient of x^i in R(x)
216 => a[K] = a[0]r[0] + a[1]r[1] + ... + a[N-1]r[N-1] */

```

## 10 Linear Algebra

### 10.1 Gaussian-Jordan Elimination

```

1 int n; vector<vector<ll>> > v;
2 void gauss(vector<vector<ll>>& v) {
3     int r = 0;
4     for (int i = 0; i < n; i++) {
5         bool ok = false;
6         for (int j = r; j < n; j++) {
7             if (v[j][i] == 0) continue;
8             swap(v[j], v[r]);
9             ok = true; break;
10        }
11        if (!ok) continue;
12        ll div = inv(v[r][i]);

```

```

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

```

### 10.2 Determinant

1. Use GJ Elimination, if there's any row consists of only 0, then det = 0, otherwise det = 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  $\vec{r}_j$ : Unchanged

## 11 Flow / Matching

### 11.1 Dinic

```

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

```

```

47         {
48             e.cap-=res;
49             path[e.to][e.rev].cap+=res;
50             return res;
51         }
52     }
53 }
54 return 0;
55 }
56 int dinic()
57 {
58     int res=0;
59     while(true)
60     {
61         bfs();
62         if(level[t]==-1)break;
63         memset(iter,0,sizeof(iter));
64         int now=0;
65         while((now=dfs(s,INF))>0)res+=now;
66     }
67     return res;
68 }
69 };

```

## 11.2 ISAP

```

1 #define SZ(c) ((int)(c).size())
2 struct Maxflow{
3     static const int MAXV=50010;
4     static const int INF =1000000;
5     struct Edge{
6         int v,c,r;
7         Edge(int _v,int _c,int _r):v(_v),c(_c),r(_r){}
8     };
9     int s,t; vector<Edge> G[MAXV];
10    int iter[MAXV],d[MAXV],gap[MAXV],tot;
11    void init(int n,int _s,int _t){
12        tot=n,s=_s,t=_t;
13        for(int i=0;i<=tot;i++){
14            G[i].clear(); iter[i]=d[i]=gap[i]=0;
15        }
16    }
17    void addEdge(int u,int v,int c){
18        G[u].push_back(Edge(v,c,SZ(G[v])));
19        G[v].push_back(Edge(u,0,SZ(G[u])-1));
20    }
21    int DFS(int p,int flow){
22        if(p==t) return flow;
23        for(int &i=iter[p];i<SZ(G[p]);i++){
24            Edge &e=G[p][i];
25            if(e.c>0&&d[p]==d[e.v]+1){
26                int f=DFS(e.v,min(flow,e.c));
27                if(f){ e.c-=f; G[e.v][e.r].c+=f; return f; }
28            }
29        }
30        if(--gap[d[p]]==0) d[s]=tot;
31        else{ d[p]++; iter[p]=0; ++gap[d[p]]; }
32        return 0;
33    }
34    int flow(){
35        int res=0;
36        for(res=0,gap[0]=tot;d[s]<tot;res+=DFS(s,INF));
37        return res;
38    } // reset: set iter,d,gap to 0
39 } flow;

```

## 11.3 MCMF

```

1 struct MCMF
2 {
3     int n,s,t,par[N+5],p_i[N+5],dis[N+5],vis[N+5];
4     struct edge{int to, cap, rev, cost;};
5     vector<edge> path[N];
6     void init(int _n,int _s,int _t)
7     {
8         n=_n,s=_s,t=_t;
9         FOR(i,0,2*n+5)par[i]=p_i[i]=vis[i]=0;
10    }
11    void add(int a,int b,int c,int d)
12    {

```

```

13        path[a].pb({b,c,sz(path[b]),d});
14        path[b].pb({a,0,sz(path[a])-1,-d});
15    }
16    void spfa()
17    {
18        FOR(i,0,n*2+5)dis[i]=INF,vis[i]=0;
19        dis[s]=0;
20        queue<int>q;q.push(s);
21        while(!q.empty())
22        {
23            int now=q.front();
24            q.pop();
25            vis[now]=0;
26            for(int i=0;i<sz(path[now]);i++)
27            {
28                edge e=path[now][i];
29                if(e.cap>0&&dis[e.to]>dis[now]+e.cost)
30                {
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                    {
36                        vis[e.to]=1;
37                        q.push(e.to);
38                    }
39                }
40            }
41        }
42    }
43    pii flow()
44    {
45        int flow=0,cost=0;
46        while(true)
47        {
48            spfa();
49            if(dis[t]==INF)break;
50            int mn=INF;
51            for(int i=t;i!=s;i=par[i])
52                mn=min(mn,path[par[i]][p_i[i]].cap);
53            flow+=mn;cost+=dis[t]*mn;
54            for(int i=t;i!=s;i=par[i])
55            {
56                edge &now=path[par[i]][p_i[i]];
57                now.cap-=mn;
58                path[i][now.rev].cap+=mn;
59            }
60        }
61        return mp(flow,cost);
62    }
63 };

```

## 11.4 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(); g.resize(n);
10    }
11    void add(int x, int y) {
12        g[x].emplace_back(y);
13        g[y].emplace_back(x);
14    }
15    bool dfs(int x) {
16        vis[x] = true;
17        Each(y, g[x]) {
18            int px = my[y];
19            if (px == -1 ||
20                (dis[px] == dis[x]+1 &&
21                 !vis[px] && dfs(px))) {
22                mx[x] = y;
23                my[y] = x;
24                return true;
25            }
26        }
27        return false;

```

```

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

```

## 11.5 Cover / Independent Set

1 V(E) Cover: choose some V(E) to cover all E(V)  
2 V(E) Independ: set of V(E) not adj to each other

3  
4 M = Max Matching  
5 Cv = Min V Cover  
6 Ce = Min E Cover  
7 Iv = Max V Ind  
8 Ie = Max E Ind (equiv to M)

9  
10 M = Cv (Konig Theorem)  
11 Iv = V \ Cv  
12 Ce = V - M

13  
14 Construct Cv:  
15 1. Run Dinic  
16 2. Find s-t min cut  
17 3. Cv = {X in T} + {Y in S}

## 11.6 KM

```

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

```

```

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

```

## 12 Combinatorics

### 12.1 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

### 12.2 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:

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

## 13 Special Numbers

### 13.1 Fibonacci Series

1	1	1	2	3
5	5	8	13	21
9	34	55	89	144
13	233	377	610	987
17	1597	2584	4181	6765
21	10946	17711	28657	46368
25	75025	121393	196418	317811
29	514229	832040	1346269	2178309
33	3524578	5702887	9227465	14930352

$$f(45) \approx 10^9, f(88) \approx 10^{18}$$

### 13.2 Prime Numbers

- First 50 prime numbers:

1	2	3	5	7	11
6	13	17	19	23	29
11	31	37	41	43	47
16	53	59	61	67	71
21	73	79	83	89	97
26	101	103	107	109	113
31	127	131	137	139	149
36	151	157	163	167	173
41	179	181	191	193	197
46	199	211	223	227	229

- Very large prime numbers:

1000001333    1000500889    2500001909  
 2000000659    900004151    850001359

- $\pi(n) \equiv \text{Number of primes} \leq n \approx n/((\ln n) - 1)$

$$\pi(100) = 25, \pi(200) = 46$$

$$\pi(500) = 95, \pi(1000) = 168$$

$$\pi(2000) = 303, \pi(4000) = 550$$

$$\pi(10^4) = 1229, \pi(10^5) = 9592$$

$$\pi(10^6) = 78498, \pi(10^7) = 664579$$













