

## 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	Others	1
3	Data Structure	2
3.1	BIT	2
3.2	Lazy Propagation Segment Tree	2
3.3	Treap	2
3.4	Persistent Treap	3
3.5	Li Chao Tree	3
3.6	Sparse Table	3
3.7	Time Segment Tree	3
3.8	Dynamic Median	4
3.9	SOS DP	4
4	Flow / Matching	4
4.1	Dinic	4
4.2	MCMF	5
4.3	KM	5
4.4	Hopcroft-Karp	6
4.5	Blossom	6
4.6	Cover / Independent Set	6
4.7	Hungarian Algorithm	6
5	Graph	7
5.1	Heavy-Light Decomposition	7
5.2	Centroid Decomposition	7
5.3	Bellman-Ford + SPFA	8
5.4	BCC - AP	9
5.5	BCC - Bridge	9
5.6	SCC - Tarjan	9
5.7	SCC - Kosaraju	10
5.8	Eulerian Path - Undir	10
5.9	Eulerian Path - Dir	10
5.10	Hamilton Path	11
5.11	Kth Shortest Path	11
5.12	System of Difference Constraints	12
6	String	12
6.1	Aho Corasick	12
6.2	KMP	13
6.3	Z Value	13
6.4	Manacher	13
6.5	Suffix Array	13
6.6	Suffix Automaton	14
6.7	Minimum Rotation	14

6.8	Lyndon Factorization	14
6.9	Rolling Hash	14
6.10	Trie	15
7	Geometry	15
7.1	Basic Operations	15
7.2	Sort by Angle	15
7.3	Intersection	15
7.4	Polygon Area	16
7.5	Convex Hull	16
7.6	Point In Convex	16
7.7	Point Segment Distance	16
7.8	Point in Polygon	16
7.9	Minimum Euclidean Distance	16
7.10	Minkowski Sum	16
7.11	Lower Concave Hull	17
7.12	Pick's Theorem	17
7.13	Rotating SweepLine	17
7.14	Half Plane Intersection	17
7.15	Minimum Enclosing Circle	17
7.16	Union of Circles	18
7.17	Area Of Circle Polygon	18
7.18	3D Point	18
8	Number Theory	18
8.1	FFT	18
8.2	Pollard's rho	19
8.3	Miller Rabin	19
8.4	Fast Power	20
8.5	Extend GCD	20
8.6	Mu + Phi	20
8.7	Discrete Log	20
8.8	sqrt mod	20
8.9	Primitive Root	21
8.10	Other Formulas	21
8.11	Polynomial	21
9	Linear Algebra	22
9.1	Gaussian-Jordan Elimination	22
9.2	Determinant	23
10	Combinatorics	23
10.1	Catalan Number	23
10.2	Burnside's Lemma	23
11	Special Numbers	23
11.1	Fibonacci Series	23
11.2	Prime Numbers	23

## 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>ko

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++ -Ofast -std=c++20 -Wall -Wextra -Wshadow $2/$1 -o $2/out
if [ "$?" -ne 0 ]
then
    exit 1
fi
echo
echo "Done compiling"
echo "===== "
echo "Input file:"
echo
cat $2/in.txt
echo
echo "===== "
echo
declare startTime=`date +%s%N`
$2/out < $2/in.txt > $2/out.txt
declare endTime=`date +%s%N`
delta=`expr $endTime - $startTime`
delta=`expr $delta / 1000000`
cat $2/out.txt
echo
echo "time: $delta ms"
```

### 2.3 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
    diff ac.txt wa.txt || break
done
```

### 2.4 Others

```
#pragma GCC optimize("O3,unroll-loops")
#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
#pragma GCC optimize("trapv")
```

## 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 / 分 case 要好好想
- 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
- 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!

```

4 mt19937 gen(chrono::steady_clock::now().
    time_since_epoch().count());
5 uniform_int_distribution<int> dis(1, 100);
6 cout << dis(gen) << endl;
7 shuffle(v.begin(), v.end(), gen);
8
9 struct edge {
10     int a, b, w;
11     friend istream& operator>>(istream& in, edge& x) {
12         in >> x.a >> x.b >> x.w; }
13     friend ostream& operator<<(ostream& out, const edge
14         & x) {
15         out << "(" << x.a << ", " << x.b << ", " << x.w
16         << ")";
17         return out;
18     }
19 };
20 struct cmp {
21     bool operator()(const edge& x, const edge& y) const
22     { return x.w < y.w; }
23 };
24 set<edge, cmp> st; // 遞增
25 map<edge, long long, cmp> mp; // 遞增
26 priority_queue<edge, vector<edge>, cmp> pq; // 遞減
27
28 #include <bits/extc++.h>
29 #include <ext/pb_ds/assoc_container.hpp>
30 #include <ext/pb_ds/tree_policy.hpp>
31 using namespace __gnu_pbds;
32
33 // map
34 tree<int, int, less<>, rb_tree_tag,
35     tree_order_statistics_node_update> tr;
36 tr.order_of_key(element);
37 tr.find_by_order(rank);
38
39 // set
40 tree<int, null_type, less<>, rb_tree_tag,
41     tree_order_statistics_node_update> tr;
42 tr.order_of_key(element);
43 tr.find_by_order(rank);
44
45 // hash table
46 gp_hash_table<int, int> ht;
47 ht.find(element);
48 ht.insert({key, value});
49 ht.erase(element);
50
51 // priority queue
52 __gnu_pbds::priority_queue<int, less<int>> big_q;
53 // Big First
54 __gnu_pbds::priority_queue<int, greater<int>> small_q;
55 // Small First
56 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 -
19            1);
20
21        long long ret = 0;
22        while (l <= r) ret += bit[r], r -= r & -r;

```

```

22     return ret;
23 }
24 } 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    ll 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    ll 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 };
15 int siz(Treap *now) { return now ? now->num : 0; }
16 int sum(Treap *now) {
17     if (!now) return 0;

```

```

18     if (now->real != -1) return (now->real + now->tag)
19         * now->num;
20     return now->sum + now->tag * now->num;
21 }
22 void pull(Treap *&now) {
23     now->num = siz(now->l) + siz(now->r) + 1ll;
24     now->sum = sum(now->l) + sum(now->r) + now->val +
25         now->tag;
26 }
27 void push(Treap *&now) {
28     if (now->rev) {
29         swap(now->l, now->r);
30         now->l->rev ^= 1;
31         now->r->rev ^= 1;
32         now->rev = 0;
33     }
34     if (now->real != -1) {
35         now->real += now->tag;
36         if (now->l) {
37             now->l->tag = 0;
38             now->l->real = now->real;
39             now->l->val = now->real;
40         }
41         if (now->r) {
42             now->r->tag = 0;
43             now->r->real = now->real;
44             now->r->val = now->real;
45         }
46         now->val = now->real;
47         now->sum = now->real * now->num;
48         now->real = -1;
49         now->tag = 0;
50     } else {
51         if (now->l) now->l->tag += now->tag;
52         if (now->r) now->r->tag += now->tag;
53         now->sum += sum(now);
54         now->val += now->tag;
55         now->tag = 0;
56     }
57 }
58 Treap *merge(Treap *a, Treap *b) {
59     if (!a || !b) return a ? a : b;
60     else if (a->pri > b->pri) {
61         push(a);
62         a->r = merge(a->r, b);
63         pull(a);
64         return a;
65     } else {
66         push(b);
67         b->l = merge(a, b->l);
68         pull(b);
69         return b;
70     }
71 }
72 void split_size(Treap *rt, Treap *&a, Treap *&b, int
73     val) {
74     if (!rt) {
75         a = b = NULL;
76         return;
77     }
78     push(rt);
79     if (siz(rt->l) + 1 > val) {
80         b = rt;
81         split_size(rt->l, a, b->l, val);
82         pull(b);
83     } else {
84         a = rt;
85         split_size(rt->r, a->r, b, val - siz(a->l) - 1);
86         pull(a);
87     }
88 }
89 void split_val(Treap *rt, Treap *&a, Treap *&b, int val)
90 {
91     if (!rt) {
92         a = b = NULL;
93         return;
94     }
95     push(rt);
96     if (rt->val <= val) {
97         a = rt;
98         split_val(rt->r, a->r, b, val);
99     }

```

```

95     pull(a);
96 } else {
97     b = rt;
98     split_val(rt->l, a, b->l, val);
99     pull(b);
100 }
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 ? 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 = a, ret->pull();
39         return {a, ret};
40     }
41 }

```

### 3.5 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

```

### 3.6 Sparse Table

```

1 const int lgmx = 19;
2
3 int n, q;
4 int spt[lgmx][maxn];
5
6 void build() {
7     FOR(k, 1, lgmx, 1) {
8         for (int i = 0; i + (1 << k) - 1 < n; i++) {
9             spt[k][i] = min(spt[k - 1][i], spt[k - 1][i
10                 + (1 << (k - 1))]);
11         }
12     }
13
14 int query(int l, int r) {
15     int ln = len(l, r);
16     int lg = __lg(ln);
17     return min(spt[lg][l], spt[lg][r - (1 << lg) + 1]);
18 }

```

### 3.7 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();
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 }
57 #undef m
58 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 }

```

### 3.8 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) <=
6             1
7         while((int)lo.size() > (int)hi.size() + 1) {
8             auto it = prev(lo.end());
9             long long x = *it;
10            lo.erase(it); slo -= x;
11            hi.insert(x); shi += x;
12        }
13        while((int)lo.size() < (int)hi.size()) {
14            auto it = hi.begin();
15            long long x = *it;
16            hi.erase(it); shi -= x;
17            lo.insert(x); slo += x;
18        }
19    }
20    void add(long long x) {
21        if(lo.empty() || x <= *prev(lo.end())) {
22            lo.insert(x); slo += x;
23        }
24        else {
25            hi.insert(x); shi += x;
26        }
27        rebalance();
28    }
29    void remove_one(long long x) {
30        if(!lo.empty() && x <= *prev(lo.end())) {
31            auto it = lo.find(x);
32            if(it != lo.end()) {
33                lo.erase(it); slo -= x;
34            }
35            else {
36                auto it2 = hi.find(x);
37                hi.erase(it2); shi -= x;
38            }
39        }
40        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 }
49 rebalance();
50 }
51 };

```

### 3.9 SOS DP

```

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

```

## 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
18             [e.to] == -1) {
19             level[e.to] = level[now] + 1;
20             q.push(e.to);
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];

```

```

3 struct edge {
4     int to, cap, rev, cost;
5 };
6 vector<edge> path[N];
7 void init(int _n, int _s, int _t) {
8     n = _n, s = _s, t = _t;
9     FOR(i, 0, 2 * n + 5)
10         par[i] = p_i[i] = vis[i] = 0;
11 }
12 void add(int a, int b, int c, int d) {
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     FOR(i, 0, n * 2 + 5)
18         dis[i] = INF,
19         vis[i] = 0;
20     dis[s] = 0;
21     queue<int> q;
22     q.push(s);
23     while (!q.empty()) {
24         int now = q.front();
25         q.pop();
26         vis[now] = 0;
27         for (int i = 0; i < sz(path[now]); i++) {
28             edge e = path[now][i];
29             if (e.cap > 0 && dis[e.to] > dis[now] +
30                 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 }
42 pii flow() {
43     int flow = 0, cost = 0;
44     while (true) {
45         spfa();
46         if (dis[t] == INF)
47             break;
48         int mn = INF;
49         for (int i = t; i != s; i = par[i])
50             mn = min(mn, path[par[i]][p_i[i]].cap);
51         flow += mn;
52         cost += dis[t] * mn;
53         for (int i = t; i != s; i = par[i]) {
54             edge &now = path[par[i]][p_i[i]];
55             now.cap -= mn;
56             path[i][now.rev].cap += mn;
57         }
58     }
59     return mp(flow, cost);
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        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             }
40             int cut = INF;
41             FOR(y, 1, n + 1)
42                 if (!vy[y] && cut > sy[y]) cut = sy[y];
43             FOR(j, 1, n + 1) {
44                 if (vx[j]) lx[j] -= cut;
45                 if (vy[j])
46                     ly[j] += cut;
47                 else
48                     sy[j] -= cut;
49             }
50             FOR(y, 1, n + 1) {
51                 if (!vy[y] && sy[y] == 0) {
52                     if (!my[y]) {
53                         augment(y);
54                         return;
55                     }
56                     vy[y] = 1;
57                     q.push(my[y]);
58                 }
59             }
60         }
61     }
62     int solve() {
63         fill(mx, mx + n + 1, 0);
64         fill(my, my + n + 1, 0);
65         fill(ly, ly + n + 1, 0);
66         fill(lx, lx + n + 1, 0);
67         FOR(x, 1, n + 1)
68             FOR(y, 1, n + 1)
69                 lx[x] = max(lx[x], g[x][y]);
70         FOR(x, 1, n + 1)
71             bfs(x);
72         int ans = 0;
73         FOR(y, 1, n + 1)
74             ans += g[my[y]][y];
75         return ans;
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     if (brk) break;
64 }
65 MXCNT = 0;
66 for (int x = 1; x <= nx; x++)
67     if (mx[x] != -1) MXCNT++;
68 }
69 } hk;
70

```

## 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     return false;
32 }
33 int solve(){
34     int ans=0;
35     FOR(i,1,n+1){
36         if(!lnk[i]){
37             stp++;
38             ans+=dfs(i);
39         }
40     }
41     return ans;
42 }
43 void print_matching(){
44     FOR(i,1,n+1)
45         if(i<graph.lnk[i])
46             cout<<i<<" "<<graph.lnk[i]<<endl;
47 }
48 };

```

## 4.6 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}

```

## 4.7 Hungarian Algorithm

```

1 const int N = 2e3;
2 int match[N];
3 bool vis[N];
4 int n;
5 vector<int> ed[N];
6 int match_cnt;
7 bool dfs(int u) {
8     vis[u] = 1;
9     for(int i : ed[u]) {
10         if(match[i] == 0 || !vis[match[i]] && dfs(match[i])) {
11             match[i] = u;
12             return true;
13         }
14     }
15     return false;
16 }
17 void hungary() {
18     memset(match, 0, sizeof(match));
19     match_cnt = 0;
20     for(int i = 1; i <= n; i++) {
21         memset(vis, 0, sizeof(vis));
22         if(dfs(i)) match_cnt++;
23     }
24 }

```

# 5 Graph

## 5.1 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     int mx, sum;
6 } seg[N << 2];
7 void update(int x, int l, int r, int qx, int val) {
8     if (l == r) {
9         seg[x].mx = seg[x].sum = val;
10        return;

```

```

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

```

```

87 void 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 }
33 int main() {
34     ios_base::sync_with_stdio(0);
35     cin.tie(0);
36     int n;
37     cin >> n;
38     for (int i = 0, x, y; i < n - 1; i++) {
39         cin >> x >> y;
40         a[x].push_back(y);
41         a[y].push_back(x);
42     }
43     cd(1, 0);
44     for (int i = 1; i <= n; i++)
45         cout << (char)('A' + lv[i] - 1) << " ";
46     cout << "\n";
47 }

```

## 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     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             //
64             for (auto& e : g[u]) {
65                 int v = e.first;
66                 ll w = e.second;
67                 if (dis[v] > dis[u] + w) {
68                     dis[v] = dis[u] + w;
69                     pa[v] = u;
70                     if (rlx == n) negCycle[v] = true;
71                 }
72             }
73         }
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         return cout << "NO" << endl, void();
100     }

```



```

101 cout << "YES\n";
102 vector<int> ans;
103 vector<bool> vis(n + 1, false);
104
105 while (true) {
106     ans.emplace_back(ptr);
107     if (vis[ptr]) break;
108     vis[ptr] = true;
109     ptr = pa[ptr];
110 }
111 reverse(ans.begin(), ans.end());
112
113 vis.assign(n + 1, false);
114 for (auto& x : ans) {
115     cout << x << ' ';
116     if (vis[x]) break;
117     vis[x] = true;
118 }
119 cout << endl;
120 }
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<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++;
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                 isap[u] = true;
35                 popout(u);

```

```

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

```

## 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             if (u == v) break;
56         }
57     }
58 }

```

```

57 }
58 }
59 int main() {
60     init();
61     REP(i, m) {
62         char su, sv;
63         int u, v;
64         cin >> su >> u >> sv >> v;
65         if (su == '-') u = no(u);
66         if (sv == '-') v = no(v);
67         clause(u, v);
68     }
69     FOR(i, 1, 2 * n + 1, 1) {
70         if (!in[i]) dfs(i);
71     }
72     FOR(u, 1, n + 1, 1) {
73         int du = no(u);
74         if (sccid[u] == sccid[du]) {
75             return cout << "IMPOSSIBLE\n", 0;
76         }
77     }
78     FOR(u, 1, n + 1, 1) {
79         int du = no(u);
80         cout << (sccid[u] < sccid[du] ? '+' : '-') << '
81         ' << endl;
82     }
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].
33         size()) gg;
34     while (!stk.empty()) {
35         int u = stk.top();
36         stk.pop();
37         cout << u << ' ';
38     }
39 }

```

## 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|)
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->
19        d > b->edge->d; }
20    struct node {
21        int v;
22        ll d;
23        heap* H;
24        nd* E;
25        node() {}
26        node(ll _d, int _v, nd* _E) {
27            d = _d;
28            v = _v;
29            E = _E;
30        }
31        node(heap* _H, ll _d) {
32            H = _H;
33            d = _d;
34        }
35        friend bool operator<(node a, node b) { return
36            a.d > b.d; }
37    };
38    int n, k, s, t, dst[N];
39    nd* nxt[N];
40    vector<nd*> g[N], rg[N];
41    heap *nullNd, *head[N];
42    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
71             ->d, e->u, e));
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             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 }
137 vector<ll> ans;
138 void first_K() {
139     ans.clear();
140     priority_queue<node> Q;
141     if (dst[s] == -1) return;
142     ans.push_back(dst[s]);
143     if (head[s] != nullNd)
144         Q.push(node(head[s], dst[s] + head[s]->edge
145             ->d));
146     for (int _ = 1; _ < k and not Q.empty(); _++) {
147         node p = Q.top(), q;
148         Q.pop();
149         ans.push_back(p.d);
150         if (head[p.H->edge->v] != nullNd) {
151             q.H = head[p.H->edge->v];
152             q.d = p.d + q.H->edge->d;
153             Q.push(q);
154         }
155         for (int i = 0; i < 4; i++)
156             if (p.H->chd[i] != nullNd) {
157                 q.H = p.H->chd[i];
158                 q.d = p.d - p.H->edge->d + p.H->chd
159                     [i]->edge->d;
160                 Q.push(q);
161             }
162     }
163 }
164 void solve() { // ans[i] stores the i-th shortest
165     // path
166     dijkstra();
167     build();
168     first_K(); // ans.size() might less than k
169 }
170 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        for (int i = pos; i < str.size(); i++) {
25            if (!cur->go[str[i] - 'a'])
26                cur->go[str[i] - 'a'] = new_Node();
27            cur = cur->go[str[i] - 'a'];
28        }
29        cur->cnt++;
30    }
31    void make_fail() { // 全部 add 完做
32        queue<Node> que;
33        que.push(root);
34        while (!que.empty()) {
35            Node *fr = que.front();
36            que.pop();
37            for (int i = 0; i < 26; i++) {
38                if (fr->go[i]) {
39                    Node *ptr = fr->fail;
40                    while (ptr && !ptr->go[i]) ptr = ptr->fail;
41                    fr->go[i]->fail = ptr = (ptr ? ptr->go[i] : root);
42                    fr->go[i]->dic = (ptr->cnt ? ptr : ptr->dic);
43                    que.push(fr->go[i]);
44                }
45            }
46        }
47    }
48    // 出現過不同string的總數
49    int query_unique(const string& text) {
50        Node* p = root;
51        int ans = 0;
52        for(char ch : text) {
53            int i = ch - 'a';
54            while(p && !p->go[i]) p = p->fail;
55            p = p ? p->go[i] : root;
56            if(p->cnt) {ans += p->cnt; p->cnt = 0;}
57            for(Node* t = p->dic; t; t = t->dic) if(t->cnt) {
58                ans += t->cnt; t->cnt = 0;
59            }
60        }
61        return ans;
62    }
63 } 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 = f[now];
8         if (s[now + 1] == s[i]) f[i] = now + 1;

```

```

9     }
10 }
11 void KMPmatching(string &a, string &b) {
12     for (int i = 0, now = -1; i < a.size(); i++) {
13         while (a[i] != b[now + 1] and now != -1) now = f[now];
14         if (a[i] == b[now + 1]) now++;
15         if (now + 1 == b.size()) {
16             cout << "found a match start at position "
17                  << i - now << endl;
18             now = f[now];
19         }
20     }
21 }

```

## 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 - l], r - i + 1);
17         while (i + z[i] < n && s[z[i]] == s[i + z[i]])
18             z[i]++;
19         if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
20         if (z[i] == (int)it.size()) ans++;
21     }
22     cout << ans << endl;

```

## 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 && s[i - m[i] - 1] == s[i + m[i] + 1]) m[i]++;
17        if (i + m[i] > mx + mxk) mx = i, mxk = m[i];
18    }
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             mx = m[i];
30             ptr = i;
31         }
32     for (int i = ptr - mx; i <= ptr + mx; i++)

```

```

33     if (s[i] != '.') cout << s[i];
34     cout << endl;
35 }

```

## 6.5 Suffix Array

```

1  #define F first
2  #define S second
3  struct SuffixArray { // don't forget s += "$";
4      int n;
5      string s;
6      vector<int> suf, lcp, rk;
7      // 後綴陣列: suf[i] = 第 i 小的後綴起點
8      // LCP 陣列: lcp[i] = suf[i] 與 suf[i-1] 的最長共同
9      // 前綴長度
10     // rank 陣列: rk[i] = 起點在 i 的後綴的名次
11     vector<int> cnt, pos;
12     vector<pair<int, int>, int> > buc[2];
13     void init(string _s) {
14         s = _s;
15         n = (int)s.size();
16         // resize(n): suf, rk, cnt, pos, lcp, buc[0~1]
17         suf.assign(n, 0);
18         rk.assign(n, 0);
19         lcp.assign(n, 0);
20         cnt.assign(n, 0);
21         pos.assign(n, 0);
22         buc[0].assign(n, {{0,0},0});
23         buc[1].assign(n, {{0,0},0});
24     }
25     void radix_sort() {
26         for (int t : {0, 1}) {
27             fill(cnt.begin(), cnt.end(), 0);
28             for (auto& i : buc[t]) cnt[(t ? i.F.F : i.F.S)++]++;
29             for (int i = 0; i < n; i++)
30                 pos[i] = (i ? 0 : pos[i - 1] + cnt[i - 1]);
31             for (auto& i : buc[t])
32                 buc[t ^ 1][pos[(t ? i.F.F : i.F.S)++] + 1] = i;
33         }
34     }
35     bool fill_suf() {
36         bool end = true;
37         for (int i = 0; i < n; i++) suf[i] = buc[0][i].S;
38         rk[suf[0]] = 0;
39         for (int i = 1; i < n; i++) {
40             int dif = (buc[0][i].F != buc[0][i - 1].F);
41             end &= dif;
42             rk[suf[i]] = rk[suf[i - 1]] + dif;
43         }
44         return end;
45     }
46     void sa() {
47         for (int i = 0; i < n; i++)
48             buc[0][i] = make_pair(make_pair(s[i], s[i]), i);
49         sort(buc[0].begin(), buc[0].end());
50         if (fill_suf()) return;
51         for (int k = 0; (1 << k) < n; k++) {
52             for (int i = 0; i < n; i++)
53                 buc[0][i] = make_pair(make_pair(rk[i], rk[(i + (1 << k)) % n]), i);
54             radix_sort();
55             if (fill_suf()) return;
56         }
57     }
58     void LCP() {
59         int k = 0;
60         for (int i = 0; i < n - 1; i++) {
61             if (rk[i] == 0) continue;
62             int pi = rk[i];
63             int j = suf[pi - 1];
64             while (i + k < n && j + k < n && s[i + k] == s[j + k]) k++;
65             lcp[pi] = k;
66             k = max(k - 1, 0);
67         }
68     }
69 }

```

```

68 };
69 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         last = 0;
18         occ.reserve(2 * maxlen + 5); occ.push_back(0);
19         first_bkpos.push_back(-1);
20     }
21     void extend(int c) {
22         int cur = (int)st.size();
23         st.push_back(State());
24         occ.push_back(0);
25         first_bkpos.push_back(0);
26         st[cur].len = st[last].len + 1;
27         first_bkpos[cur] = st[cur].len - 1;
28         int p = last;
29         while (p != -1 && st[p].next[c] == -1) {
30             st[p].next[c] = cur;
31             p = st[p].link;
32         }
33         if (p == -1) {
34             st[cur].link = 0;
35         } else {
36             int q = st[p].next[c];
37             if (st[p].len + 1 == st[q].len) {
38                 st[cur].link = q;
39             } else {
40                 int clone = (int)st.size();
41                 st.push_back(st[q]);
42                 first_bkpos.push_back(first_bkpos[q]);
43                 occ.push_back(0);
44                 st[clone].len = st[p].len + 1;
45                 while (p != -1 && st[p].next[c] == q) {
46                     st[p].next[c] = clone;
47                     p = st[p].link;
48                 }
49                 st[q].link = st[cur].link = clone;
50             }
51         }
52         last = cur;
53         occ[cur] += 1;
54     }
55     void finalize_occ() {
56         int m = (int)st.size();
57         vector<int> order(m);
58         iota(order.begin(), order.end(), 0);
59         sort(order.begin(), order.end(), [&](int a, int b) {
60             return st[a].len > st[b].len; });
61         for (int v : order) {
62             int p = st[v].link;
63             if (p != -1) occ[p] += occ[v];
64         }
65     }
66 };

```

## 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    return a;
18 }

```

## 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         return res;
28     }
29 };

```

## 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 / sqrt1(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
39 inline T cross(const Pt &a, const Pt &b, const Pt &c) {
40     return (b.x - a.x) * (c.y - a.y)
41         - (b.y - a.y) * (c.x - a.x);
42 }
43
44 long double polar_angle(Pt ori, Pt pt) {
45     return atan2(pt.y - ori.y, pt.x - ori.x);
46 }
47
48 // slope to degree atan(Slope) * 180.0 / acos(-1.0);
49 bool argcmp(Pt u, Pt v) {
50     auto half = [](const Pt &p) {
51         return p.y > 0 || (p.y == 0 && p.x >= 0);
52     };
53     if (half(u) != half(v)) return half(u) < half(v);
54     return sgn(u ^ v) > 0;
55 }
56
57 int ori(Pt& o, Pt& a, Pt& b) {
58     return sgn((a - o) ^ (b - o));
59 }
60
61 struct Line {
62     Pt a, b;
63     Pt dir() { return b - a; }
64 };
65
66 int PtSide(Pt p, Line L) {
67     return sgn(ori(L.a, L.b, p)); // for int
68     return sgn(ori(L.a, L.b, p) / sqrt(len2(L.a - L.b))
69         );
70 }
71
72 bool PtOnSeg(Pt p, Line L) {
73     return PtSide(p, L) == 0 and sgn((p - L.a) * (p - L
74         .b)) <= 0;
75 }
76
77 Pt proj(Pt& p, Line& l) {
78     Pt d = l.b - l.a;

```

```

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

```

## 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 });

```

## 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 }

```

## 7.4 Polygon Area

```

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

```

```

7 }
8 return abs(res);
9 }

```

## 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[
8                 hull.size() - 2], hull.back()), mv(hull[
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 }

```

```

7   return (cnt ? 1 : -1);
8 }

```

## 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(sqrt(1.0 * ((long
9             double)best)));
10        while (now.x - pts[l].x > lim) {
11            s.erase({pts[l].y, pts[l].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
8 struct LineContainer : multiset<Line, less<>> {
9     // (for doubles, use inf = 1/.0, div(a,b) = a/b)
10    const ll inf = LLONG_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()) { x->p = inf; return false; }
15        if (x->m == y->m) x->p = x->b > y->b ? inf : -inf;
16        else x->p = div(y->b - x->b, x->m - y->m);
17        return x->p >= y->p;
18    }
19    void add(ll m, ll b) {
20        auto z = insert({m, b, 0}); y = z++, x = y;
21        while (isect(y, z)) z = erase(z);
22        if (x != begin() && isect(--x, y)) isect(x, y =
23            erase(y));
24        while ((y = x) != begin() && (--x)->p >= y->p)
25            isect(x, erase(y));
26    }
27 }

```

```

24 }
25 ll query(ll x) {
26     assert(!empty());
27     auto l = *lower_bound(x);
28     return l.m * x + l.b;
29 }
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:

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

## 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[
12            ni].y - hull[i].y}, {hull[(j+1)%m].x - hull[
13                i].x, hull[(j+1)%m].y - 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,
16                    hull[j].y - 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        while (l < r && cover(P[i], P[l], P[l + 1])) ++l;
21        P[++r] = P[i];
22    }
23    while (l < r && cover(P[l], P[r - 1], P[r])) --r;
24    while (l < r && cover(P[r], P[l], P[l + 1])) ++l;
25    if (r - l <= 1 || !argcmp(P[l].dir(), P[r].dir()))
26        return {};
27    if (cover(P[l + 1], P[l], P[r])) return {};
28    return vector<Line>(P.begin() + l, P.begin() + r +
29        1);
30 }

```

29 }

## 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     for (int i = 0; i < pts.size(); i++) {
24         if (dis2(center, pts[i]) <= r2) continue;
25         center = pts[i], r2 = 0;
26         for (int j = 0; j < i; j++) {
27             if (dis2(center, pts[j]) <= r2) continue;
28             center = (pts[i] + pts[j]) / 2.0;
29             r2 = dis2(center, pts[i]);
30             for (int k = 0; k < j; k++) {
31                 if (dis2(center, pts[k]) <= r2)
32                     continue;
33                 center = circumcenter(pts[i], pts[j],
34                     pts[k]);
35                 r2 = dis2(center, pts[i]);
36             }
37         }
38     }
39 }

```

## 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                 if (a1 > a2) cov++;
31             }
32         }
33     }
34     if (event.empty()) {
35         Area[cov] += acos(-1) * C[i].r * C[i].r;
36         continue;
37     }
38 }

```

```

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].
40             ang;
41         if (theta < 0) theta += 2 * acos(-1);
42         Area[cov] += (theta - sin(theta)) * C[i].r
43             * C[i].r / 2.;
44     }
45 }
46 return Area;
47 }

```

## 7.17 Area Of Circle Polygon

```

1  double AreaOfCirclePoly(Cir C, vector<Pt> &P) {
2      auto arg = [&](Pt p, Pt q) { return atan2(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)fabs1(sum);
24 }

```

## 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     return o - n * ((o - a) * (n / abs2(n))); }
32 Pt line_plane_intersect(Pt u, Pt v, Pt a, Pt b, Pt c) {
33     // intersection of line uv and plane abc
34     Pt n = cross3(a, b, c);
35     double s = n * (u - v);
36     if (sign(s) == 0) return {-1, -1, -1}; // not found
37 }

```

```

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;
45 }

```

## 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          }
60      }
61
62      void fft(vector<cp> &a) { transform(a, omega); }
63      void ifft(vector<cp> &a) {
64          transform(a, iomega);
65          for (int i = 0; i < n; i++) a[i] /= n;
66      }
67  }

```

```

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 = acos(-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 %
79                 MAXN) : i * theta % MAXN];
80             for (int j = i; j < n; j += m) {
81                 int k = j + mh;
82                 cplx x = a[j] - a[k];
83                 a[j] += a[k];
84                 a[k] = w * x;
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 (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
99         long b[], long long ans[]) {
100         int n = 1, sum = _n + _m - 1;
101         while (n < sum) n <= 1;
102         for (int i = 0; i < n; i++) {
103             double x = (i < _n ? a[i] : 0), y = (i < _m ? b
104                 [i] : 0);
105             arr[i] = complex<double>(x + y, x - y);
106         }
107         fft(n, arr);
108         for (int i = 0; i < n; i++) arr[i] = arr[i] * arr[i
109             ];
110         fft(n, arr, true);
111         for (int i = 0; i < sum; i++) ans[i] = (long long
112             int)(arr[i].real() / 4 + 0.5);
113     }
114 }
115
116 long long a[MAXN];
117 long long b[MAXN];
118 long long ans[MAXN];
119 int a_length;
120 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) *
6          mod;
7      return ret < 0 ? ret + mod : ret;
8  }
9  ll f(ll x, ll mod) { return add(qMul(x, x, mod), 1, mod
10      ); }
11  ll pollard_rho(ll n) {
12      if (!(n & 1)) return 2;

```



```

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

```

### 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 : pimes <= 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:  $a^n \equiv a^{(n \bmod (p-1))} \pmod{p}$

### 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 }
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 ){
3     long long r = 1 % p;
4     while(e){
5         if(e & 1) r = (__int128)r * a % p;
6         a = (__int128)a * a % p;
7         e >>= 1;
8     }
9     return r;
10 }
11 long long mod_inv(long long a, long long p){
12     return mod_pow((a%p+p)%p, p-2, p);
13 }
14 // BSGS: solve a^x = y (mod p), gcd(a,p)=1, p prime,
15 // return minimal x>=0, or -1 if no solution
16 long long bsgs(long long a, long long y, long long p){
17     a%p; y%p;
18     if(y==1%p) return 0; // x=0
19     long long m = (long long)ceil(sqrt((long double)p));
20     ;
21     // baby steps: a^j
22     unordered_map<long long, long long> table;
23     table.reserve(m*2);
24     long long cur = 1%p;
25     for(long long j=0; j<m; ++j){
26         if(!table.count(cur)) table[cur]=j;
27         cur = (__int128)cur * a % p;
28     }
29     long long am = mod_pow(a, m, p);
30     long long am_inv = mod_inv(am, p);
31     long long gamma = y % p;
32     for(long long i=0; i<m; ++i){
33         auto it = table.find(gamma);
34         if(it != table.end()){
35             long long x = i*m + it->second;
36             return x;
37         }
38         gamma = (__int128)gamma * am_inv % p;
39     }
40     return -1;
41 }

```

## 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) {

```



```

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

```

## 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 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:

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

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

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

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

## 8.11 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   27    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  3958241859937  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     int _n = (int)a.size();
119     int k = __lg(_n) + ((1<<__lg(_n)) != _n);
120     int n = 1<<k;
121     a.resize(n, 0);
122
123     short shift = maxk-k;
124     for (int i = 0; i < n; i++)
125         if (i > (rev[i]>>shift))
126             swap(a[i], a[rev[i]>>shift]);
127
128     for (int len = 2, half = 1, div = maxn>>1; len <= n; len<<=1, half<<=1, div>>=1) {
129         for (int i = 0; i < n; i += len) {
130             for (int j = 0; j < half; j++) {
131                 T u = a[i+j];
132                 T v = a[i+j+half] * (inv ? iX[j*div] : X[j*div]) % MOD;
133                 a[i+j] = (u+v >= MOD ? u+v-MOD : u+v);
134                 a[i+j+half] = (u-v < 0 ? u-v+MOD : u-v);
135             }
136         }
137     }
138
139     if (inv) {
140         T dn = pw(n, MOD-2);
141         for (auto& x : a) {
142             x *= dn;
143             if (x >= MOD) x %= MOD;
144         }
145     }
146 }
147
148 template<typename T>
149 inline void resize(vector<T>& a) {
150     int cnt = (int)a.size();
151     for (; cnt > 0; cnt--) if (a[cnt-1]) break;
152     a.resize(max(cnt, 1));
153 }
154
155 template<typename T>
156 vector<T>& operator*=(vector<T>& a, vector<T> b) {
157     int na = (int)a.size();
158     int nb = (int)b.size();
159     a.resize(na + nb - 1, 0);
160     b.resize(na + nb - 1, 0);
161
162     NTT(a); NTT(b);
163     for (int i = 0; i < (int)a.size(); i++) {
164         a[i] *= b[i];
165         if (a[i] >= MOD) a[i] %= MOD;
166     }
167     NTT(a, true);
168
169     resize(a);
170     return a;
171 }
172
173 template<typename T>
174 void inv(vector<T>& ia, int N) {
175     vector<T> _a(move(ia));
176     ia.resize(1, pw(_a[0], MOD-2));
177     vector<T> a(1, -_a[0] + (-_a[0] < 0 ? MOD : 0));
178
179     for (int n = 1; n < N; n<<=1) {
180         // n -> 2*n
181         // ia' = ia(2-a*ia);
182
183         for (int i = n; i < min(siz(_a), (n<<1)); i++)
184             a.emplace_back(-_a[i] + (-_a[i] < 0 ? MOD : 0));
185
186         vector<T> tmp = ia;
187         ia *= a;
188         ia.resize(n<<1);
189         ia[0] = ia[0] + 2 >= MOD ? ia[0] + 2 - MOD : ia[0] + 2;
190         ia *= tmp;
191         ia.resize(n<<1);
192     }
193     ia.resize(N);
194 }
195
196 template<typename T>
197 void mod(vector<T>& a, vector<T>& b) {
198     int n = (int)a.size()-1, m = (int)b.size()-1;
199     if (n < m) return;
200
201     vector<T> ra = a, rb = b;
202     reverse(ra.begin(), ra.end()); ra.resize(min(n+1, n-m+1));
203     reverse(rb.begin(), rb.end()); rb.resize(min(m+1, n-m+1));

```

```

196 inv(rb, n-m+1);
197
198 vector<T> q = move(ra);
199 q *= rb;
200 q.resize(n-m+1);
201 reverse(q.begin(), q.end());
202
203 q *= b;
204 a -= q;
205 resize(a);
206 }
207
208 /* Kitamasa Method (Fast Linear Recurrence):
209 Find a[K] (Given a[j] = c[0]a[j-N] + ... + c[N-1]a[j
210 -1])
211 Let B(x) = x^N - c[N-1]x^(N-1) - ... - c[1]x^1 - c[0]
212 Let R(x) = x^K mod 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 => a[K] = a[0]r[0] + a[1]r[1] + ... + a[N-1]r[N-1] */

```

## 9 Linear Algebra

### 9.1 Gaussian-Jordan Elimination

```

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

## 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:

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

## 11 Special Numbers

### 11.1 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

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

### 11.2 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

- $\pi(n) \equiv \text{Number of primes } \leq n \approx n/((\ln n) - 1)$   
 $\pi(100) = 25, \pi(200) = 46$   
 $\pi(500) = 95, \pi(1000) = 168$   
 $\pi(2000) = 303, \pi(4000) = 550$   
 $\pi(10^4) = 1229, \pi(10^5) = 9592$   
 $\pi(10^6) = 78498, \pi(10^7) = 664579$











