

Contents

1	Reminder	1
1.1	Bug List	1
1.2	OwO	1
2	Basic	1
2.1	Vimrc	1
2.2	Runcpp.sh	1
2.3	Stress	1
2.4	PBDS	1
2.5	Random	1
3	Data Structure	2
3.1	BIT	2
3.2	DSU	2
3.3	Segment Tree	2
3.4	Treap	2
3.5	Persistent Treap	2
3.6	Li Chao Tree	3
3.7	Sparse Table	3
3.8	Time Segment Tree	3
4	Flow / Matching	4
4.1	Dinic	4
4.2	MCMF	4
4.3	KM	5
4.4	Hopcroft-Karp	5
4.5	Blossom	5
4.6	Weighted Blossom	6
4.7	Cover / Independent Set	7
5	Graph	7
5.1	Heavy-Light Decomposition	7
5.2	Centroid Decomposition	8
5.3	Bellman-Ford + SPFA	9
5.4	BCC - AP	9
5.5	BCC - Bridge	10
5.6	SCC - Tarjan	10
5.7	SCC - Kosaraju	11
5.8	Eulerian Path - Undir	11
5.9	Eulerian Path - Dir	11
5.10	Hamilton Path	12
5.11	Kth Shortest Path	12
5.12	System of Difference Constraints	13
6	String	13
6.1	Rolling Hash	13
6.2	Trie	13
6.3	KMP	13
6.4	Z Value	13
6.5	Manacher	14
6.6	Suffix Array	14
6.7	SA-IS	14
6.8	Minimum Rotation	14
6.9	Aho Corasick	15
7	Geometry	15
7.1	Basic Operations	15
7.2	InPoly	15
7.3	Sort by Angle	15
7.4	Line Intersect Check	15
7.5	Line Intersection	15
7.6	Convex Hull	15
7.7	Lower Concave Hull	16
7.8	Polygon Area	16
7.9	Pick's Theorem	16
7.10	Minimum Enclosing Circle	16
7.11	PolyUnion	16
7.12	Minkowski Sum	16
8	Number Theory	17
8.1	FFT	17
8.2	Pollard's rho	18
8.3	Miller Rabin	18
8.4	Fast Power	18
8.5	Extend GCD	18
8.6	Mu + Phi	19
8.7	Other Formulas	19
8.8	Polynomial	19
9	Linear Algebra	20
9.1	Gaussian-Jordan Elimination	20
9.2	Determinant	20
10	Combinatorics	21
10.1	Catalan Number	21
10.2	Burnside's Lemma	21
11	Special Numbers	21
11.1	Fibonacci Series	21
11.2	Prime Numbers	21

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!

2 Basic

2.1 Vimrc

```

set number relativenumber ai t_Co=256 tabstop=4
set mouse=a shiftwidth=4 encoding=utf8
set bs=2 ruler laststatus=2 cmdheight=2
set clipboard=unnamedplus showcmd autoread
set belloff=all
filetype indent on
"set guifont Hack:h16
":set guifont?

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

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

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

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

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

```

2.2 Runcpp.sh

```

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

```

```

8     exit 1
9 fi
10 echo
11 echo "Done compiling"
12 echo "=====
13 echo
14 echo "Input file:"
15 echo
16 cat $2/in.txt
17 echo
18 echo "=====
19 echo
20 declare startTime=`date +%s%N`
21 $2/out < $2/in.txt > $2/out.txt
22 declare endTime=`date +%s%N`
23 delta=`expr $endTime - $startTime`
24 delta=`expr $delta / 1000000`
25 cat $2/out.txt
26 echo
27 echo "time: $delta ms"

```

2.3 Stress

```

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

```

2.4 PBDS

```

1 #include <bits/extc++.h>
2 using namespace __gnu_pbds;
3
4 // map
5 tree<int, int, less<>, rb_tree_tag,
6     tree_order_statistics_node_update> tr;
7 tr.order_of_key(element);
8 tr.find_by_order(rank);
9
10 // set
11 tree<int, null_type, less<>, rb_tree_tag,
12     tree_order_statistics_node_update> tr;
13 tr.order_of_key(element);
14 tr.find_by_order(rank);
15
16 // hash table
17 gp_hash_table<int, int> ht;
18 ht.find(element);
19 ht.insert({key, value});
20 ht.erase(element);
21
22 // priority queue
23 __gnu_pbds::priority_queue<int, less<int>> big_q;
24 // Big First
25 __gnu_pbds::priority_queue<int, greater<int>> small_q;
26 // Small First
27 q1.join(q2);
28
29 // join

```

2.5 Random

```

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

```

3 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;
23        return ret;
24    }
25 } bm;

```

3.2 DSU

```

1 struct DSU {
2     int h[N], s[N];
3
4     void init(int n) { iota(h, h + n + 1, 0), fill(s, s +
5         n + 1, 1); }
6
7     int fh(int x) { return (h[x] == x ? x : h[x] = fh(h
8         [x])); }
9
10    bool mer(int x, int y) {
11        x = fh(x), y = fh(y);
12        if (x == y) return 0;
13        if (s[x] < s[y]) swap(x, y);
14        s[x] += s[y], s[y] = 0;
15        h[y] = x;
16        return 1;
17    }
18 } bm;

```

3.3 Segment Tree

```

1 struct segtree {
2     int n, seg[1 << 19];
3
4     void init(int x) {
5         n = 1 << (lg(x) + 1);
6         for (int i = 1; i < 2 * n; i++)
7             seg[i] = inf;
8     }
9
10    void update(int x, int val) {
11        x += n;
12        seg[x] = val, x /= 2;
13        while (x)
14            seg[x] = min(seg[2 * x], seg[2 * x + 1]), x
15                /= 2;
16    }
17
18    int query(int l, int r) {
19        l += n, r += n;
20        int ret = inf;
21        while (l < r) {
22            if (l & 1)
23                ret = min(ret, seg[l++]);
24            if (r & 1)
25                ret = min(ret, seg[--r]);
26            l /= 2, r /= 2;
27        }
28        return ret;
29    }
30 } bm;

```

3.4 Treap

```

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

```

3.5 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);
21         ret->pull();
22         return ret;
23     } else {
24         node* ret = new (ptr++) node(b);
25         ret->l = merge(a, ret->l);
26         ret->pull();
27         return ret;
28     }
29 }
30 P<node*> split(node* p, int k) {
31     if (!p) return {nullptr, nullptr};
32     if (k >= size(p->l) + 1) {
33         auto [a, b] = split(p->r, k - size(p->l) - 1);
34         node* ret = new (ptr++) node(p);
35         ret->r = a;
36         ret->pull();
37         return {ret, b};
38     } else {
39         auto [a, b] = split(p->l, k);
40         node* ret = new (ptr++) node(p);
41         ret->l = b;
42         ret->pull();
43         return {a, ret};
44     }
45 }

```

3.6 Li Chao Tree

```

1 constexpr int maxn = 5e4 + 5;
2 struct line {
3     ld a, b;
4     ld operator()(ld x) { return a * x + b; }
5 } arr[(maxn + 1) << 2];
6 bool operator<(line a, line b) { return a.a < b.a; }
7 #define m ((l + r) >> 1)
8 void insert(line x, int i = 1, int l = 0, int r = maxn)
    {
9     if (r - l == 1) {
10         if (x(l) > arr[i](l))
11             arr[i] = x;
12         return;
13     }
14     line a = max(arr[i], x), b = min(arr[i], x);
15     if (a(m) > b(m))
16         arr[i] = a, insert(b, i << 1, l, m);
17     else
18         arr[i] = b, insert(a, i << 1 | 1, m, r);
19 }
20 ld query(int x, int i = 1, int l = 0, int r = maxn) {
21     if (x < l || r <= x) return -numeric_limits<ld>::
22         max();
23     if (r - l == 1) return arr[i](x);
24     return max({arr[i](x), query(x, i << 1, l, m),
25         query(x, i << 1 | 1, m, r)});
26 }
27 #undef m

```

3.7 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 }

```

3.8 Time Segment Tree

```

1 constexpr int maxn = 1e5 + 5;
2 V<P<int>> arr[(maxn + 1) << 2];
3 V<int> dsu, sz;
4 V<tuple<int, int, int>> his;
5 int cnt, q;
6 int find(int x) {
7     return x == dsu[x] ? x : find(dsu[x]);
8 };
9 inline bool merge(int x, int y) {
10     int a = find(x), b = find(y);
11     if (a == b) return false;
12     if (sz[a] > sz[b]) swap(a, b);
13     his.emplace_back(a, b, sz[b]), dsu[a] = b, sz[b] +=
14         sz[a];
15     return true;
16 };
17 inline void undo() {
18     auto [a, b, s] = his.back();
19     his.pop_back();
20     dsu[a] = a, sz[b] = s;
21 };
22 #define m ((l + r) >> 1)
23 void insert(int ql, int qr, P<int> x, int i = 1, int l
24     = 0, int r = q) {
25     // debug(ql, qr, x); return;
26     if (qr <= l || r <= ql) return;
27     if (ql <= l && r <= qr) {
28         arr[i].push_back(x);
29         return;
30     }
31     if (qr <= m)
32         insert(ql, qr, x, i << 1, l, m);
33     else if (m <= ql)
34         insert(ql, qr, x, i << 1 | 1, m, r);
35     else {
36         insert(ql, qr, x, i << 1, l, m);
37         insert(ql, qr, x, i << 1 | 1, m, r);
38     }
39 }
40 void traversal(V<int>& ans, int i = 1, int l = 0, int r
41     = q) {
42     int opcnt = 0;
43     // debug(i, l, r);
44     for (auto [a, b] : arr[i])
45         if (merge(a, b))
46             opcnt++, cnt--;
47     if (r - l == 1)
48         ans[l] = cnt;
49     else {
50         traversal(ans, i << 1, l, m);
51         traversal(ans, i << 1 | 1, m, r);
52     }
53     while (opcnt--)
54         undo(), cnt++;
55     arr[i].clear();
56 }
57 #undef m
58 inline void solve() {
59     int n, m;
60     cin >> n >> m >> q, q++;
61     dsu.resize(cnt = n), sz.assign(n, 1);
62     iota(dsu.begin(), dsu.end(), 0);
63     // a, b, time, operation
64     unordered_map<ll, V<int>> s;
65     for (int i = 0; i < m; i++) {
66         int a, b;
67         cin >> a >> b;
68         if (a > b) swap(a, b);
69         s[((1ll)a << 32) | b].emplace_back(0);
70     }
71     for (int i = 1; i < q; i++) {
72         int op, a, b;
73         cin >> op >> a >> b;

```

```

71     if (a > b) swap(a, b);
72     switch (op) {
73     case 1:
74         s[((1ll)a << 32) | b].push_back(i);
75         break;
76     case 2:
77         auto tmp = s[((1ll)a << 32) | b].back();
78         s[((1ll)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 }

```

4 Flow / Matching

4.1 Dinic

```

1 struct Dinic {
2     int n, s, t, level[N], iter[N];
3     struct edge {
4         int to, cap, rev;
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, n + 1)
10             path[i].clear();
11     }
12     void add(int a, int b, int c) {
13         edge now;
14         now.to = b, now.cap = c, now.rev = sz(path[b]);
15         path[a].pb(now);
16         now.to = a, now.cap = 0, now.rev = sz(path[a])
17             - 1;
18         path[b].pb(now);
19     }
20     void bfs() {
21         memset(level, -1, sizeof(level));
22         level[s] = 0;
23         queue<int> q;
24         q.push(s);
25         while (q.size()) {
26             int now = q.front();
27             q.pop();
28             for (edge e : path[now]) {
29                 if (e.cap > 0 && level[e.to] == -1) {
30                     level[e.to] = level[now] + 1;
31                     q.push(e.to);
32                 }
33             }
34         }
35     }
36     int dfs(int now, int flow) {
37         if (now == t) return flow;
38         for (int &i = iter[now]; i < sz(path[now]); i++) {
39             edge &e = path[now][i];
40             if (e.cap > 0 && level[e.to] == level[now]
41                 + 1) {
42                 int res = dfs(e.to, min(flow, e.cap));
43                 if (res > 0) {
44                     e.cap -= res;
45                     path[e.to][e.rev].cap += res;
46                     return res;
47                 }
48             }
49         }
50         return 0;
51     }
52 }

```

```

49     }
50     int dinic() {
51         int res = 0;
52         while (true) {
53             bfs();
54             if (level[t] == -1) break;
55             memset(iter, 0, sizeof(iter));
56             int now = 0;
57             while ((now = dfs(s, INF)) > 0) res += now;
58         }
59         return res;
60     }
61 };

```

4.2 MCMF

```

1 struct MCMF {
2     int n, s, t, par[N + 5], p_i[N + 5], dis[N + 5],
3         vis[N + 5];
4     struct edge {
5         int to, cap, rev, cost;
6     };
7     vector<edge> path[N];
8     void init(int _n, int _s, int _t) {
9         n = _n, s = _s, t = _t;
10        FOR(i, 0, 2 * n + 5)
11            par[i] = p_i[i] = vis[i] = 0;
12    }
13    void add(int a, int b, int c, int d) {
14        path[a].pb({b, c, sz(path[b]), d});
15        path[b].pb({a, 0, sz(path[a]) - 1, -d});
16    }
17    void spfa() {
18        FOR(i, 0, n * 2 + 5)
19            dis[i] = INF,
20            vis[i] = 0;
21        dis[s] = 0;
22        queue<int> q;
23        q.push(s);
24        while (!q.empty()) {
25            int now = q.front();
26            q.pop();
27            vis[now] = 0;
28            for (int i = 0; i < sz(path[now]); i++) {
29                edge e = path[now][i];
30                if (e.cap > 0 && dis[e.to] > dis[now] +
31                    e.cost) {
32                    dis[e.to] = dis[now] + e.cost;
33                    par[e.to] = now;
34                    p_i[e.to] = i;
35                    if (vis[e.to] == 0) {
36                        vis[e.to] = 1;
37                        q.push(e.to);
38                    }
39                }
40            }
41        }
42    }
43    pii flow() {
44        int flow = 0, cost = 0;
45        while (true) {
46            spfa();
47            if (dis[t] == INF)
48                break;
49            int mn = INF;
50            for (int i = t; i != s; i = par[i])
51                mn = min(mn, path[par[i]][p_i[i]].cap);
52            flow += mn;
53            cost += dis[t] * mn;
54            for (int i = t; i != s; i = par[i]) {
55                edge &now = path[par[i]][p_i[i]];
56                now.cap -= mn;
57                path[i][now.rev].cap += mn;
58            }
59            return mp(flow, cost);
60        }
61 };

```

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]) ly[j] += cut;
46            } else sy[j] -= cut;
47        }
48        FOR(y, 1, n + 1) {
49            if (!vy[y] && sy[y] == 0) {
50                if (!my[y]) {
51                    augment(y);
52                    return;
53                }
54                vy[y] = 1;
55                q.push(my[y]);
56            }
57        }
58    }
59 }
60
61 int solve() {
62     fill(mx, mx + n + 1, 0);
63     fill(my, my + n + 1, 0);
64     fill(ly, ly + n + 1, 0);
65     fill(lx, lx + n + 1, 0);
66     FOR(x, 1, n + 1)
67         FOR(y, 1, n + 1)
68             lx[x] = max(lx[x], g[x][y]);
69     bfs(x);
70     int ans = 0;
71     FOR(y, 1, n + 1)
72         ans += g[my[y]][y];
73     return ans;
74 }
75
76 };
77

```

4.4 Hopcroft-Karp

```

1 struct HopcroftKarp {

```

```

2 // id: X = [1, nx], Y = [nx+1, nx+ny]
3 int n, nx, ny, m, MXCNT;
4 vector<vector<int>> > g;
5 vector<int> mx, my, dis, vis;
6 void init(int nnx, int nny, int mm) {
7     nx = nnx, ny = nny, m = mm;
8     n = nx + ny + 1;
9     g.clear();
10    g.resize(n);
11 }
12 void add(int x, int y) {
13     g[x].emplace_back(y);
14     g[y].emplace_back(x);
15 }
16 bool dfs(int x) {
17     vis[x] = true;
18     Each(y, g[x]) {
19         int px = my[y];
20         if (px == -1 ||
21             (dis[px] == dis[x] + 1 &&
22              !vis[px] && dfs(px))) {
23             mx[x] = y;
24             my[y] = x;
25             return true;
26         }
27     }
28     return false;
29 }
30 void get() {
31     mx.clear();
32     mx.resize(n, -1);
33     my.clear();
34     my.resize(n, -1);
35
36     while (true) {
37         queue<int> q;
38         dis.clear();
39         dis.resize(n, -1);
40         for (int x = 1; x <= nx; x++) {
41             if (mx[x] == -1) {
42                 dis[x] = 0;
43                 q.push(x);
44             }
45         }
46         while (!q.empty()) {
47             int x = q.front();
48             q.pop();
49             Each(y, g[x]) {
50                 if (my[y] != -1 && dis[my[y]] ==
51                     -1) {
52                     dis[my[y]] = dis[x] + 1;
53                     q.push(my[y]);
54                 }
55             }
56         }
57         bool brk = true;
58         vis.clear();
59         vis.resize(n, 0);
60         for (int x = 1; x <= nx; x++)
61             if (mx[x] == -1 && dfs(x))
62                 brk = false;
63
64         if (brk) break;
65     }
66     MXCNT = 0;
67     for (int x = 1; x <= nx; x++)
68         if (mx[x] != -1) MXCNT++;
69 }
70 } hk;

```

4.5 Blossom

```

1 const int N=5e2+10;
2 struct Graph{
3     int to[N],bro[N],head[N],e;
4     int lnk[N],vis[N],stp,n;
5     void init(int _n){
6         stp=0;e=1;n=_n;
7         FOR(i,0,n+1)head[i]=lnk[i]=vis[i]=0;
8     }

```

```

9     void add(int u,int v){
10         to[e]=v,bro[e]=head[u],head[u]=e++;
11         to[e]=u,bro[e]=head[v],head[v]=e++;
12     }
13     bool dfs(int x){
14         vis[x]=stp;
15         for(int i=head[x];i;i=bro[i])
16             {
17                 int v=to[i];
18                 if(!lnk[v])
19                     {
20                         lnk[x]=v;lnk[v]=x;
21                         return true;
22                     }
23                 else if(vis[lnk[v]]<stp)
24                     {
25                         int w=lnk[v];
26                         lnk[x]=v,lnk[v]=x,lnk[w]=0;
27                         if(dfs(w))return true;
28                         lnk[w]=v,lnk[v]=w,lnk[x]=0;
29                     }
30             }
31         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 Weighted Blossom

```

1 struct WeightGraph { // 1-based
2     static const int inf = INT_MAX;
3     static const int maxn = 514;
4     struct edge {
5         int u, v, w;
6         edge() {}
7         edge(int u, int v, int w) : u(u), v(v), w(w) {}
8     };
9     int n, n_x;
10    edge g[maxn * 2][maxn * 2];
11    int lab[maxn * 2];
12    int match[maxn * 2], slack[maxn * 2], st[maxn * 2],
13        pa[maxn * 2];
14    int flo_from[maxn * 2][maxn + 1], S[maxn * 2], vis[
15        maxn * 2];
16    vector<int> flo[maxn * 2];
17    queue<int> q;
18    int e_delta(const edge &e) { return lab[e.u] + lab[
19        e.v] - g[e.u][e.v].w * 2; }
20    void update_slack(int u, int x) {
21        if (!slack[x] || e_delta(g[u][x]) < e_delta(g[
22            slack[x]][x])) slack[x] = u;
23    }
24    void set_slack(int x) {
25        slack[x] = 0;
26        for (int u = 1; u <= n; ++u)
27            if (g[u][x].w > 0 && st[u] != x && S[st[u]]
28                == 0)
29                update_slack(u, x);
30    }
31    void q_push(int x) {
32        if (x <= n)
33            q.push(x);
34        else
35            for (size_t i = 0; i < flo[x].size(); i++)
36                q_push(flo[x][i]);
37    }
38    void set_st(int x, int b) {

```

```

33     st[x] = b;
34     if (x > n)
35         for (size_t i = 0; i < flo[x].size(); ++i)
36             set_st(flo[x][i], b);
37 }
38 int get_pr(int b, int xr) {
39     int pr = find(flo[b].begin(), flo[b].end(), xr) - flo[b].begin();
40     if (pr % 2 == 1) {
41         reverse(flo[b].begin() + 1, flo[b].end());
42         return (int)flo[b].size() - pr;
43     }
44     return pr;
45 }
46 void set_match(int u, int v) {
47     match[u] = g[u][v].v;
48     if (u <= n) return;
49     edge e = g[u][v];
50     int xr = flo_from[u][e.u], pr = get_pr(u, xr);
51     for (int i = 0; i < pr; ++i) set_match(flo[u][i], flo[u][i ^ 1]);
52     set_match(xr, v);
53     rotate(flo[u].begin(), flo[u].begin() + pr, flo[u].end());
54 }
55 void augment(int u, int v) {
56     for (;;) {
57         int xnv = st[match[u]];
58         set_match(u, v);
59         if (!xnv) return;
60         set_match(xnv, st[pa[xnv]]);
61         u = st[pa[xnv]], v = xnv;
62     }
63 }
64 int get_lca(int u, int v) {
65     static int t = 0;
66     for (++t; u || v; swap(u, v)) {
67         if (u == 0) continue;
68         if (vis[u] == t) return u;
69         vis[u] = t;
70         u = st[match[u]];
71         if (u) u = st[pa[u]];
72     }
73     return 0;
74 }
75 void add_blossom(int u, int lca, int v) {
76     int b = n + 1;
77     while (b <= n_x && st[b]) ++b;
78     if (b > n_x) ++n_x;
79     lab[b] = 0, S[b] = 0;
80     match[b] = match[lca];
81     flo[b].clear();
82     flo[b].push_back(lca);
83     for (int x = u, y; x != lca; x = st[pa[y]])
84         flo[b].push_back(x), flo[b].push_back(y = st[match[x]]), q_push(y);
85     reverse(flo[b].begin() + 1, flo[b].end());
86     for (int x = v, y; x != lca; x = st[pa[y]])
87         flo[b].push_back(x), flo[b].push_back(y = st[match[x]]), q_push(y);
88     set_st(b, b);
89     for (int x = 1; x <= n_x; ++x) g[b][x].w = g[x][b].w = 0;
90     for (int x = 1; x <= n; ++x) flo_from[b][x] = 0;
91     for (size_t i = 0; i < flo[b].size(); ++i) {
92         int xs = flo[b][i];
93         for (int x = 1; x <= n_x; ++x)
94             if (g[b][x].w == 0 || e_delta(g[xs][x]) < e_delta(g[b][x]))
95                 g[b][x] = g[xs][x], g[x][b] = g[x][xs];
96         for (int x = 1; x <= n; ++x)
97             if (flo_from[xs][x]) flo_from[b][x] = xs;
98     }
99     set_slack(b);
100 }
101 void expand_blossom(int b) {
102     for (size_t i = 0; i < flo[b].size(); ++i)
103         set_st(flo[b][i], flo[b][i]);
104 }
105 int xr = flo_from[b][g[b][pa[b]].u], pr = get_pr(b, xr);
106 for (int i = 0; i < pr; i += 2) {
107     int xs = flo[b][i], xns = flo[b][i + 1];
108     pa[xs] = g[xns][xs].u;
109     S[xs] = 1, S[xns] = 0;
110     slack[xs] = 0, set_slack(xns);
111     q_push(xns);
112 }
113 S[xr] = 1, pa[xr] = pa[b];
114 for (size_t i = pr + 1; i < flo[b].size(); ++i) {
115     int xs = flo[b][i];
116     S[xs] = -1, set_slack(xs);
117 }
118 st[b] = 0;
119 }
120 bool on_found_edge(const edge &e) {
121     int u = st[e.u], v = st[e.v];
122     if (S[v] == -1) {
123         pa[v] = e.u, S[v] = 1;
124         int nu = st[match[v]];
125         slack[v] = slack[nu] = 0;
126         S[nu] = 0, q_push(nu);
127     } else if (S[v] == 0) {
128         int lca = get_lca(u, v);
129         if (!lca)
130             return augment(u, v), augment(v, u), true;
131         else
132             add_blossom(u, lca, v);
133     }
134     return false;
135 }
136 bool matching() {
137     memset(S + 1, -1, sizeof(int) * n_x);
138     memset(slack + 1, 0, sizeof(int) * n_x);
139     q = queue<int>();
140     for (int x = 1; x <= n_x; ++x)
141         if (st[x] == x && !match[x]) pa[x] = 0, S[x] = 0, q_push(x);
142     if (q.empty()) return false;
143     for (;;) {
144         while (q.size()) {
145             int u = q.front();
146             q.pop();
147             if (S[st[u]] == 1) continue;
148             for (int v = 1; v <= n; ++v)
149                 if (g[u][v].w > 0 && st[u] != st[v] && (e_delta(g[u][v]) == 0 || (on_found_edge(g[u][v]))))
150                     return true;
151                 else
152                     update_slack(u, st[v]);
153         }
154     }
155     int d = inf;
156     for (int b = n + 1; b <= n_x; ++b)
157         if (st[b] == b && S[b] == 1) d = min(d, lab[b] / 2);
158     for (int x = 1; x <= n_x; ++x)
159         if (st[x] == x && slack[x]) {
160             if (S[x] == -1)
161                 d = min(d, e_delta(g[slack[x]](x)));
162             else if (S[x] == 0)
163                 d = min(d, e_delta(g[slack[x]](x)) / 2);
164         }
165     for (int u = 1; u <= n; ++u) {
166         if (S[st[u]] == 0) {
167             if (lab[u] <= d) return 0;
168             lab[u] -= d;
169         } else if (S[st[u]] == 1)
170             lab[u] += d;
171     }
172     for (int b = n + 1; b <= n_x; ++b)
173         if (st[b] == b) {
174             if (S[st[b]] == 0)
175                 lab[b] += d * 2;
176             else if (S[st[b]] == 1)

```



```

176         lab[b] -= d * 2;
177     }
178     q = queue<int>();
179     for (int x = 1; x <= n_x; ++x)
180         if (st[x] == x && slack[x] && st[slack[
181             x]] != x && e_delta(g[slack[x]][x])
182             == 0)
183             if (on_found_edge(g[slack[x]][x]))
184                 return true;
185     for (int b = n + 1; b <= n_x; ++b)
186         if (st[b] == b && S[b] == 1 && lab[b]
187             == 0) expand_blossom(b);
188 }
189 return false;
190 }
191 pair<long long, int> solve() {
192     memset(match + 1, 0, sizeof(int) * n);
193     n_x = n;
194     int n_matches = 0;
195     long long tot_weight = 0;
196     for (int u = 0; u <= n; ++u) st[u] = u, flo[u].
197         clear();
198     int w_max = 0;
199     for (int u = 1; u <= n; ++u)
200         for (int v = 1; v <= n; ++v) {
201             flo_from[u][v] = (u == v ? u : 0);
202             w_max = max(w_max, g[u][v].w);
203         }
204     for (int u = 1; u <= n; ++u) lab[u] = w_max;
205     while (matching()) ++n_matches;
206     for (int u = 1; u <= n; ++u)
207         if (match[u] && match[u] < u)
208             tot_weight += g[u][match[u]].w;
209     return make_pair(tot_weight, n_matches);
210 }
211 void add_edge(int ui, int vi, int wi) { g[ui][vi].w
212     = g[vi][ui].w = wi; }
213 void init(int _n) {
214     n = _n;
215     for (int u = 1; u <= n; ++u)
216         for (int v = 1; v <= n; ++v)
217             g[u][v] = edge(u, v, 0);
218 }
219 }
220 }
221 }
222 }
223 }
224 }
225 }
226 }
227 }
228 }
229 }
230 }
231 }
232 }
233 }
234 }
235 }
236 }
237 }
238 }
239 }
240 }
241 }
242 }
243 }
244 }
245 }
246 }
247 }
248 }
249 }
250 }
251 }
252 }
253 }
254 }
255 }
256 }
257 }
258 }
259 }
260 }
261 }
262 }
263 }
264 }
265 }
266 }
267 }
268 }
269 }
270 }
271 }
272 }
273 }
274 }
275 }
276 }
277 }
278 }
279 }
280 }
281 }
282 }
283 }
284 }
285 }
286 }
287 }
288 }
289 }
290 }
291 }
292 }
293 }
294 }
295 }
296 }
297 }
298 }
299 }
300 }
301 }
302 }
303 }
304 }
305 }
306 }
307 }
308 }
309 }
310 }
311 }
312 }
313 }
314 }
315 }
316 }
317 }
318 }
319 }
320 }
321 }
322 }
323 }
324 }
325 }
326 }
327 }
328 }
329 }
330 }
331 }
332 }
333 }
334 }
335 }
336 }
337 }
338 }
339 }
340 }
341 }
342 }
343 }
344 }
345 }
346 }
347 }
348 }
349 }
350 }
351 }
352 }
353 }
354 }
355 }
356 }
357 }
358 }
359 }
360 }
361 }
362 }
363 }
364 }
365 }
366 }
367 }
368 }
369 }
370 }
371 }
372 }
373 }
374 }
375 }
376 }
377 }
378 }
379 }
380 }
381 }
382 }
383 }
384 }
385 }
386 }
387 }
388 }
389 }
390 }
391 }
392 }
393 }
394 }
395 }
396 }
397 }
398 }
399 }
400 }
401 }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
467 }
468 }
469 }
470 }
471 }
472 }
473 }
474 }
475 }
476 }
477 }
478 }
479 }
480 }
481 }
482 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
501 }
502 }
503 }
504 }
505 }
506 }
507 }
508 }
509 }
510 }
511 }
512 }
513 }
514 }
515 }
516 }
517 }
518 }
519 }
520 }
521 }
522 }
523 }
524 }
525 }
526 }
527 }
528 }
529 }
530 }
531 }
532 }
533 }
534 }
535 }
536 }
537 }
538 }
539 }
540 }
541 }
542 }
543 }
544 }
545 }
546 }
547 }
548 }
549 }
550 }
551 }
552 }
553 }
554 }
555 }
556 }
557 }
558 }
559 }
560 }
561 }
562 }
563 }
564 }
565 }
566 }
567 }
568 }
569 }
570 }
571 }
572 }
573 }
574 }
575 }
576 }
577 }
578 }
579 }
580 }
581 }
582 }
583 }
584 }
585 }
586 }
587 }
588 }
589 }
590 }
591 }
592 }
593 }
594 }
595 }
596 }
597 }
598 }
599 }
600 }
601 }
602 }
603 }
604 }
605 }
606 }
607 }
608 }
609 }
610 }
611 }
612 }
613 }
614 }
615 }
616 }
617 }
618 }
619 }
620 }
621 }
622 }
623 }
624 }
625 }
626 }
627 }
628 }
629 }
630 }
631 }
632 }
633 }
634 }
635 }
636 }
637 }
638 }
639 }
640 }
641 }
642 }
643 }
644 }
645 }
646 }
647 }
648 }
649 }
650 }
651 }
652 }
653 }
654 }
655 }
656 }
657 }
658 }
659 }
660 }
661 }
662 }
663 }
664 }
665 }
666 }
667 }
668 }
669 }
670 }
671 }
672 }
673 }
674 }
675 }
676 }
677 }
678 }
679 }
680 }
681 }
682 }
683 }
684 }
685 }
686 }
687 }
688 }
689 }
690 }
691 }
692 }
693 }
694 }
695 }
696 }
697 }
698 }
699 }
700 }
701 }
702 }
703 }
704 }
705 }
706 }
707 }
708 }
709 }
710 }
711 }
712 }
713 }
714 }
715 }
716 }
717 }
718 }
719 }
720 }
721 }
722 }
723 }
724 }
725 }
726 }
727 }
728 }
729 }
730 }
731 }
732 }
733 }
734 }
735 }
736 }
737 }
738 }
739 }
740 }
741 }
742 }
743 }
744 }
745 }
746 }
747 }
748 }
749 }
750 }
751 }
752 }
753 }
754 }
755 }
756 }
757 }
758 }
759 }
760 }
761 }
762 }
763 }
764 }
765 }
766 }
767 }
768 }
769 }
770 }
771 }
772 }
773 }
774 }
775 }
776 }
777 }
778 }
779 }
780 }
781 }
782 }
783 }
784 }
785 }
786 }
787 }
788 }
789 }
790 }
791 }
792 }
793 }
794 }
795 }
796 }
797 }
798 }
799 }
800 }
801 }
802 }
803 }
804 }
805 }
806 }
807 }
808 }
809 }
810 }
811 }
812 }
813 }
814 }
815 }
816 }
817 }
818 }
819 }
820 }
821 }
822 }
823 }
824 }
825 }
826 }
827 }
828 }
829 }
830 }
831 }
832 }
833 }
834 }
835 }
836 }
837 }
838 }
839 }
840 }
841 }
842 }
843 }
844 }
845 }
846 }
847 }
848 }
849 }
850 }
851 }
852 }
853 }
854 }
855 }
856 }
857 }
858 }
859 }
860 }
861 }
862 }
863 }
864 }
865 }
866 }
867 }
868 }
869 }
870 }
871 }
872 }
873 }
874 }
875 }
876 }
877 }
878 }
879 }
880 }
881 }
882 }
883 }
884 }
885 }
886 }
887 }
888 }
889 }
890 }
891 }
892 }
893 }
894 }
895 }
896 }
897 }
898 }
899 }
900 }
901 }
902 }
903 }
904 }
905 }
906 }
907 }
908 }
909 }
910 }
911 }
912 }
913 }
914 }
915 }
916 }
917 }
918 }
919 }
920 }
921 }
922 }
923 }
924 }
925 }
926 }
927 }
928 }
929 }
930 }
931 }
932 }
933 }
934 }
935 }
936 }
937 }
938 }
939 }
940 }
941 }
942 }
943 }
944 }
945 }
946 }
947 }
948 }
949 }
950 }
951 }
952 }
953 }
954 }
955 }
956 }
957 }
958 }
959 }
960 }
961 }
962 }
963 }
964 }
965 }
966 }
967 }
968 }
969 }
970 }
971 }
972 }
973 }
974 }
975 }
976 }
977 }
978 }
979 }
980 }
981 }
982 }
983 }
984 }
985 }
986 }
987 }
988 }
989 }
990 }
991 }
992 }
993 }
994 }
995 }
996 }
997 }
998 }
999 }
1000 }

```

4.7 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}

```

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

```

```

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

```



```

92 }
93 void buildTree()
94 {
95     FOR(i,0,n-1)
96     {
97         int a,b;cin>>a>>b;
98         path[a].pb(b);
99         path[b].pb(a);
100     }
101 }
102 void buildHLD(int root)
103 {
104     dep[root]=1;
105     dfs1(root);
106     dfs2(root,root);
107     FOR(i,1,n+1)
108     {
109         int now;cin>>now;
110         update(1,1,n,dfn[i],now);
111     }
112 }

```

5.2 Centroid Decomposition

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 const int N = 1e5+5;
5
6 vector<int> a[N];
7
8 int sz[N], lv[N];
9 bool used[N];
10
11 int f_sz(int x, int p)
12 {
13     sz[x] = 1;
14     for(int i: a[x])
15         if(i != p && !used[i])
16             sz[x] += f_sz(i, x);
17     return sz[x];
18 }
19
20 int f_cen(int x, int p, int total)
21 {
22     for(int i: a[x])
23     {
24         if(i != p && !used[i] && 2 * sz[i] > total)
25             return f_cen(i, x, total);
26     }
27     return x;
28 }
29
30 void cd(int x, int p)
31 {
32     int total = f_sz(x, p);
33     int cen = f_cen(x, p, total);
34     lv[cen] = lv[p] + 1;
35     used[cen] = 1;
36     //cout << "cd: " << x << " " << p << " " << cen <<
37     //      "\n";
38     for(int i: a[cen])
39     {
40         if(!used[i])
41             cd(i, cen);
42     }
43 }
44
45 int main()
46 {
47     ios_base::sync_with_stdio(0);
48     cin.tie(0);
49
50     int n;
51     cin >> n;
52     for(int i=0, x, y; i<n-1; i++)
53     {
54         cin >> x >> y;
55         a[x].push_back(y);
56         a[y].push_back(x);
57     }

```

```

57     cd(1, 0);
58
59     for(int i=1; i<=n; i++)
60         cout << (char)('A' + lv[i] - 1) << " ";
61     cout << "\n";
62 }

```

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); inq[s] = true;
24     }
25
26     while (!q.empty()) {
27         int u = q.front();
28         q.pop(); inq[u] = false;
29         if (rlx[u] >= n) {
30             negCycle[u] = true;
31         }
32         else for (auto& e : g[u]) {
33             int v = e.first;
34             ll w = e.second;
35             if (dis[v] > dis[u] + w) {
36                 dis[v] = dis[u] + w;
37                 rlx[v] = rlx[u] + 1;
38                 pa[v] = u;
39                 if (!inq[v]) {
40                     q.push(v);
41                     inq[v] = true;
42                 }
43             }
44         }
45     }
46
47 // Bellman-Ford
48 queue<int> q;
49 vector<int> pa;
50 void BellmanFord(vector<int>& src) {
51     dis.assign(n+1, LINF);
52     negCycle.assign(n+1, false);
53     pa.assign(n+1, -1);
54
55     for (auto& s : src) dis[s] = 0;
56
57     for (int rlx = 1; rlx <= n; rlx++) {
58         for (int u = 1; u <= n; u++) {
59             if (dis[u] == LINF) continue; // Important
60             !!
61             for (auto& e : g[u]) {
62                 int v = e.first; ll w = e.second;
63                 if (dis[v] > dis[u] + w) {
64                     dis[v] = dis[u] + w;
65                     pa[v] = u;
66                     if (rlx == n) negCycle[v] = true;
67                 }
68             }
69         }
70     }
71
72 // Negative Cycle Detection
73 void NegCycleDetect() {
74     /* No Neg Cycle: NO
75     Exist Any Neg Cycle:
76     YES

```

```

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

```

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

```

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
6 stack<int> stk;
7 bitset<maxn> ins, vis;
8
9 int n, m;
10
11 void init() {
12     cin >> m >> n;
13     E.clear();
14     fill(g, g+maxn, vector<int>());
15     fill(low, low+maxn, INF);
16     memset(in, 0, sizeof(in));
17     instp = 1;
18     sccnt = 0;
19     memset(sccid, 0, sizeof(sccid));
20     ins.reset();
21     vis.reset();
22 }
23
24 inline int no(int u) {
25     return (u > n ? u-n : u+n);
26 }
27
28 int ecnt = 0;
29 inline void clause(int u, int v) {
30     E eb(no(u)^v);
31     g[no(u)].eb(ecnt++);
32     E eb(no(v)^u);
33     g[no(v)].eb(ecnt++);
34 }
35
36 void dfs(int u) {
37     in[u] = instp++;
38     low[u] = in[u];
39     stk.push(u);
40     ins[u] = true;
41
42     Each(e, g[u]) {

```

```

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

```

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 {
11     vis[x] = 1;
12     for(int i : ed[x]) {
13         if(vis[i]) continue;
14         dfs(i);
15     }
16     pre.push_back(x);
17 }
18
19 void dfs2(int x)
20 {
21     vis[x] = 1;

```

```

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

```

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

```

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; cin >> u >> v;
13        g[u].emplace_back(v);
14        out[u]++, in[v]++;
15    }
16    for (int i = 1; i <= n; i++) {
17        if (i == 1 && out[i]-in[i] != 1) gg;
18        if (i == n && in[i]-out[i] != 1) gg;
19        if (i != 1 && i != n && in[i] != out[i]) gg;
20    }
21    void dfs(int u) {
22        while (!g[u].empty()) {
23            int v = g[u].back();
24            g[u].pop_back();
25            dfs(v);
26        }

```

```

27        stk.push(u);
28    }
29    void solve() {
30        dfs(1)
31        for (int i = 1; i <= n; i++)
32            if ((int)g[i].size()) gg;
33        while (!stk.empty()) {
34            int u = stk.top();
35            stk.pop();
36            cout << u << ' ';
37    } }

```

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

```

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; ll d;
6         nd(int ui=0,int vi=0,ll di=INF){ u=ui; v=vi; d=di;
7     };
8     struct heap{ nd* edge; int dep; heap* chd[4]; };
9     static int cmp(heap* a,heap* b)
10     { return a->edge->d > b->edge->d; }
11     struct node{

```

```

12 int v; ll d; heap* H; nd* E;
13 node(){
14 node(ll _d, int _v, nd* _E){ d=_d; v=_v; E=_E; }
15 node(heap* _H, ll _d){ H=_H; d=_d; }
16 friend bool operator<(node a, node b)
17 { return a.d>b.d; }
18 };
19 int n, k, s, t, dst[N]; nd *nxt[N];
20 vector<nd*> g[N], rg[N]; heap *nullNd, *head[N];
21 void init(int _n, int _k, int _s, int _t){
22 n=_n; k=_k; s=_s; t=_t;
23 for(int i=1; i<=n; i++){
24 g[i].clear(); rg[i].clear();
25 nxt[i]=NULL; head[i]=NULL; dst[i]=-1;
26 }
27 }
28 void addEdge(int ui, int vi, ll di){
29 nd* e=new nd(ui, vi, di);
30 g[ui].push_back(e); rg[vi].push_back(e);
31 }
32 queue<int> dfsQ;
33 void dijkstra(){
34 while(dfsQ.size()) dfsQ.pop();
35 priority_queue<node> Q; Q.push(node(0, t, NULL));
36 while (!Q.empty()){
37 node p=Q.top(); Q.pop(); if(dst[p.v]!=-1) continue;
38 dst[p.v]=p.d; nxt[p.v]=p.E; dfsQ.push(p.v);
39 for(auto e: rg[p.v]) Q.push(node(p.d+e->d, e->u, e));
40 }
41 }
42 heap* merge(heap* curNd, heap* newNd){
43 if(curNd==nullNd) return newNd;
44 heap* root=new heap; memcpy(root, curNd, sizeof(heap));
45 if(newNd->edge->d<curNd->edge->d){
46 root->edge=newNd->edge;
47 root->chd[2]=newNd->chd[2];
48 root->chd[3]=newNd->chd[3];
49 newNd->edge=curNd->edge;
50 newNd->chd[2]=curNd->chd[2];
51 newNd->chd[3]=curNd->chd[3];
52 }
53 if(root->chd[0]->dep<root->chd[1]->dep)
54 root->chd[0]=merge(root->chd[0], newNd);
55 else root->chd[1]=merge(root->chd[1], newNd);
56 root->dep=max(root->chd[0]->dep,
57 root->chd[1]->dep)+1;
58 return root;
59 }
60 vector<heap*> V;
61 void build(){
62 nullNd=new heap; nullNd->dep=0; nullNd->edge=new nd
63 ;
64 fill(nullNd->chd, nullNd->chd+4, nullNd);
65 while(not dfsQ.empty()){
66 int u=dfsQ.front(); dfsQ.pop();
67 if(!nxt[u]) head[u]=nullNd;
68 else head[u]=head[nxt[u]->v];
69 V.clear();
70 for(auto&& e: g[u]){
71 int v=e->v;
72 if(dst[v]==-1) continue;
73 e->d+=dst[v]-dst[u];
74 if(nxt[u]!=e){
75 heap* p=new heap; fill(p->chd, p->chd+4, nullNd);
76 p->dep=1; p->edge=e; V.push_back(p);
77 }
78 if(V.empty()) continue;
79 make_heap(V.begin(), V.end(), cmp);
80 #define L(X) ((X<<1)+1)
81 #define R(X) ((X<<1)+2)
82 for(size_t i=0; i<V.size(); i++){
83 if(L(i)<V.size()) V[i]->chd[2]=V[L(i)];
84 else V[i]->chd[2]=nullNd;
85 if(R(i)<V.size()) V[i]->chd[3]=V[R(i)];
86 else V[i]->chd[3]=nullNd;
87 }
88 head[u]=merge(head[u], V.front());

```

```

89 }
90 }
91 vector<ll> ans;
92 void first_K(){
93 ans.clear(); priority_queue<node> Q;
94 if(dst[s]==-1) return;
95 ans.push_back(dst[s]);
96 if(head[s]!=nullNd)
97 Q.push(node(head[s], dst[s]+head[s]->edge->d));
98 for(int _=1; _<k and not Q.empty(); _++){
99 node p=Q.top(); Q.pop(); ans.push_back(p.d);
100 if(head[p.H->edge->v]!=nullNd){
101 q.H=head[p.H->edge->v]; q.d=p.d+q.H->edge->d;
102 Q.push(q);
103 }
104 for(int i=0; i<4; i++){
105 if(p.H->chd[i]!=nullNd){
106 q.H=p.H->chd[i];
107 q.d=p.d-p.H->edge->d+p.H->chd[i]->edge->d;
108 Q.push(q);
109 } } }
110 void solve(){ // ans[i] stores the i-th shortest path
111 dijkstra(); build();
112 first_K(); // ans.size() might less than k
113 }
114 } 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 Rolling Hash

```

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

```

```

22     ll res = hs[r] - (1 ? hs[l-1] * Cexp[r-l+1] :
23         0);
24     res = (res % mod + mod) % mod;
25     return res; }
};

```

6.2 Trie

```

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

```

6.3 KMP

```

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

```

6.4 Z Value

```

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

```

6.5 Manacher

```

1 int n; string S, s;
2 vector<int> m;

```

```

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

```

6.6 Suffix Array

```

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

```

```

50     }}
51 };
52 SuffixArray suffixarray;

```

6.7 SA-IS

```

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

```

6.8 Minimum Rotation

```

1  //rotate(begin(s), begin(s)+minRotation(s), end(s))
2  int minRotation(string s) {
3      int a = 0, n = s.size(); s += s;
4      for(int b = 0; b < n; b++) for(int k = 0; k < n; k++) {
5          if(a + k == b ||| s[a + k] < s[b + k]) {
6              b += max(0, k - 1);
7              break; }
8          if(s[a + k] > s[b + k]) {
9              a = b;
10             break; }

```

```

11     } }
12     return a; }

```

6.9 Aho Corasick

```

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

```

7 Geometry

7.1 Basic Operations

```

1  typedef long long T;
2  // typedef long double T;
3  const long double eps = 1e-8;
4
5  short sgn(T x) {
6      if (abs(x) < eps) return 0;
7      return x < 0 ? -1 : 1;
8  }
9
10 struct Pt {
11     T x, y;
12     Pt(T _x=0, T _y=0):x(_x), y(_y) {}
13     Pt operator+(Pt a) { return Pt(x+a.x, y+a.y); }
14     Pt operator-(Pt a) { return Pt(x-a.x, y-a.y); }
15     Pt operator*(T a) { return Pt(x*a, y*a); }
16     Pt operator/(T a) { return Pt(x/a, y/a); }
17     T operator*(Pt a) { return x*a.x + y*a.y; }
18     T operator^(Pt a) { return x*a.y - y*a.x; }
19     bool operator<(Pt a)
20         { return x < a.x || (x == a.x && y < a.y); }
21     //return sgn(x-a.x) < 0 || (sgn(x-a.x) == 0 && sgn(y-a.y) < 0); }
22     bool operator==(Pt a)
23         { return sgn(x-a.x) == 0 && sgn(y-a.y) == 0; }
24 };
25
26 Pt mv(Pt a, Pt b) { return b-a; }
27 T len2(Pt a) { return a*a; }
28 T dis2(Pt a, Pt b) { return len2(b-a); }
29
30 short ori(Pt a, Pt b) { return ((a^b)>0) - ((a^b)<0); }

```



```

31 bool onseg(Pt p, Pt l1, Pt l2) {
32     Pt a = mv(p, l1), b = mv(p, l2);
33     return ((a^b) == 0) && ((a*b) <= 0);
34 }

```

7.2 InPoly

```

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

```

7.3 Sort by Angle

```

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

```

7.4 Line Intersect Check

```

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

```

7.5 Line Intersection

```

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

```

7.6 Convex Hull

```

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

```

7.7 Lower Concave Hull

```

1 struct Line {
2     mutable ll m, b, p;

```

```

3     bool operator<(const Line& o) const { return m < o.m;
4     }
5     bool operator<(ll x) const { return p < x; }
6 };
7 struct LineContainer : multiset<Line, less<>> {
8     // (for doubles, use inf = 1/.0, div(a,b) = a/b)
9     const ll inf = LLONG_MAX;
10    ll div(ll a, ll b) { // floored division
11        return a / b - ((a ^ b) < 0 && a % b); }
12    bool isect(iterator x, iterator y) {
13        if (y == end()) { x->p = inf; return false; }
14        if (x->m == y->m) x->p = x->b > y->b ? inf : -inf;
15        else x->p = div(y->b - x->b, x->m - y->m);
16        return x->p >= y->p;
17    }
18    void add(ll m, ll b) {
19        auto z = insert({m, b, 0}), y = z++, x = y;
20        while (isect(y, z)) z = erase(z);
21        if (x != begin() && isect(--x, y)) isect(x, y =
22            erase(y));
23        while ((y = x) != begin() && (--x)->p >= y->p)
24            isect(x, erase(y));
25    }
26    ll query(ll x) {
27        assert(!empty());
28        auto l = *lower_bound(x);
29        return l.m * x + l.b;
30    }
31 };

```

7.8 Polygon Area

```

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

```

7.9 Pick's Theorem

Consider a polygon which vertices are all lattice points.
 Let i = number of points inside the polygon.
 Let b = number of points on the boundary of the polygon.

Then we have the following formula:

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

7.10 Minimum Enclosing Circle

```

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

```

```

27     if (dis2(center, E[k]) <= r2) continue;
28     center = circumcenter(E[i], E[j], E[k]);
29     r2 = dis2(center, E[i]);
30 }
31 }
32 } }

```

7.11 PolyUnion

```

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

```

7.12 Minkowski Sum

```

1 /* convex hull Minkowski Sum*/
2 #define INF 10000000000000LL
3 int pos(const Pt& tp){
4     if(tp.Y == 0) return tp.X > 0 ? 0 : 1;
5     return tp.Y > 0 ? 0 : 1;
6 }
7 #define N 300030
8 Pt pt[N], qt[N], rt[N];
9 LL Lx,Rx;
10 int dn,un;

```

```

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

```

```

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

```

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

```

```

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

```

8.2 Pollard's rho

```

1  ll add(ll x, ll y, ll p) {
2      return (x + y) % p;
3  }
4  ll qMul(ll x, ll y, ll mod){
5      ll ret = x * y - (ll)((long double)x / mod * y) * mod;
6      return ret<0?ret+mod:ret;
7  }
8  ll f(ll x, ll mod) { return add(qMul(x,x,mod),1,mod); }
9  ll pollard_rho(ll n) {
10     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); fact(x / f);
31 }

```

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

```

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

```

8.7 Other Formulas

- Inversion: $aa^{-1} \equiv 1 \pmod{m}$. a^{-1} exists iff $\gcd(a, m) = 1$.
- Linear inversion: $a^{-1} \equiv (m - \lfloor \frac{m}{a} \rfloor) \times (m \bmod a)^{-1} \pmod{m}$
- Fermat's little theorem: $a^p \equiv a \pmod{p}$ if p is prime.
- Euler function: $\phi(n) = n \prod_{p|n} \frac{p-1}{p}$
- Euler theorem: $a^{\phi(n)} \equiv 1 \pmod{n}$ if $\gcd(a, n) = 1$.
- Extended Euclidean algorithm: $ax + by = \gcd(a, b) = \gcd(b, a \bmod b) = \gcd(b, a - \lfloor \frac{a}{b} \rfloor b) = bx_1 + (a - \lfloor \frac{a}{b} \rfloor b)y_1 = ay_1 + b(x_1 - \lfloor \frac{a}{b} \rfloor y_1)$
- Divisor function: $\sigma_x(n) = \sum_{d|n} d^x$. $n = \prod_{i=1}^r p_i^{a_i}$.
 $\sigma_x(n) = \prod_{i=1}^r \frac{p_i^{(a_i+1)x} - 1}{p_i^x - 1}$ if $x \neq 0$. $\sigma_0(n) = \prod_{i=1}^r (a_i + 1)$.
- Chinese remainder theorem (Coprime Moduli):
 $x \equiv a_i \pmod{m_i}$.
 $M = \prod m_i$. $M_i = M/m_i$. $t_i = M_i^{-1}$.
 $x = kM + \sum a_i t_i M_i$, $k \in \mathbb{Z}$.
- Chinese remainder theorem:
 $x \equiv a_1 \pmod{m_1}, x \equiv a_2 \pmod{m_2} \Rightarrow x = m_1 p + a_1 = m_2 q + a_2 \Rightarrow m_1 p - m_2 q = a_2 - a_1$
Solve for (p, q) using ExtGCD.
 $x \equiv m_1 p + a_1 \equiv m_2 q + a_2 \pmod{\text{lcm}(m_1, m_2)}$
- Avoiding Overflow: $ca \bmod cb = c(a \bmod b)$
- Dirichlet Convolution: $(f * g)(n) = \sum_{d|n} f(d)g(n/d)$
- Important Multiplicative Functions + Properties:
 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: $f = g * 1 \Leftrightarrow g = f * \mu$

8.8 Polynomial

```

1  const int maxk = 20;
2  const int maxn = 1<<maxk;
3  const ll LINF = 1e18;
4
5  /* P = r*2^k + 1
6  P          r    k    g
7  998244353    119 23   3
8  1004535809   479 21   3
9
10 P          r    k    g
11 3           1    1    2
12 5           1    2    2
13 17          1    4    3
14 97          3    5    5
15 193         3    6    5
16 257         1    8    3
17 7681        15   9   17
18 12289       3   12   11
19 40961       5   13   3
20 65537       1   16   3
21 786433      3   18   10
22 5767169     11  19   3
23 7340033     7   20   3
24 23068673    11  21   3
25 104857601   25  22   3
26 167772161   5   25   3
27 469762049   7   26   3
28 1004535809  479 21   3
29 2013265921  15  27   31
30 2281701377  17  27   3
31 3221225473  3   30   5
32 75161927681 35  31   3
33 77309411329 9   33   7
34 206158430209 3  36   22
35 2061584302081 15 37   7
36 2748779069441 5  39   3
37 6597069766657 3  41   5
38 39582418599937 9  42   5
39 79164837199873 9  43   5
40 263882790666241 15 44   7
41 1231453023109121 35 45   3
42 1337006139375617 19 46   3
43 3799912185593857 27 47   5
44 4222124650659841 15 48   19
45 7881299347898369 7  50   6
46 31525197391593473 7  52   3
47 180143985094819841 5  55   6
48 1945555039024054273 27 56   5
49 4179340454199820289 29 57   3
50 9097271247288401921 505 54   6 */
51
52 const int g = 3;
53 const ll MOD = 998244353;
54
55 ll pw(ll a, ll n) { /* fast pow */ }
56
57 #define siz(x) (int)x.size()
58
59 template<typename T>
60 vector<T>& operator+=(vector<T>& a, const vector<T>& b)
61 {
62     if (siz(a) < siz(b)) a.resize(siz(b));
63     for (int i = 0; i < min(siz(a), siz(b)); i++) {
64         a[i] += b[i];
65         a[i] -= a[i] >= MOD ? MOD : 0;
66     }
67     return a;
68 }
69
70 template<typename T>
71 vector<T>& operator-=(vector<T>& a, const vector<T>& b)
72 {
73     if (siz(a) < siz(b)) a.resize(siz(b));
74     for (int i = 0; i < min(siz(a), siz(b)); i++) {
75         a[i] -= b[i];
76         a[i] += a[i] < 0 ? MOD : 0;
77     }
78     return a;
79 }
80
81 template<typename T>
82 vector<T> operator-(const vector<T>& a) {
83     vector<T> ret(siz(a));
84     for (int i = 0; i < siz(a); i++) {
85         ret[i] = -a[i] < 0 ? -a[i] + MOD : -a[i];
86     }
87     return ret;
88 }
89
90 vector<ll> X, iX;
91 vector<int> rev;
92
93 void init_ntt() {
94     X.clear(); X.resize(maxn, 1); // x1 = g^((p-1)/n)
95     iX.clear(); iX.resize(maxn, 1);
96
97     ll u = pw(g, (MOD-1)/maxn);
98     ll iu = pw(u, MOD-2);
99
100     for (int i = 1; i < maxn; i++) {
101         X[i] = X[i-1] * u;
102         iX[i] = iX[i-1] * iu;
103         if (X[i] >= MOD) X[i] %= MOD;
104         if (iX[i] >= MOD) iX[i] %= MOD;
105     }
106
107     rev.clear(); rev.resize(maxn, 0);
108     for (int i = 1, hb = -1; i < maxn; i++) {
109         if (!(i & (i-1))) hb++;
110         rev[i] = rev[i ^ (1<<hb)] | (1<<(maxk-hb-1));
111     }
112 }
113
114 template<typename T>
115 void NTT(vector<T>& a, bool inv=false) {
116
117     int _n = (int)a.size();
118     int k = __lg(_n) + ((1<<__lg(_n)) != _n);
119     int n = 1<<k;
120     a.resize(n, 0);
121
122     short shift = maxk-k;
123     for (int i = 0; i < n; i++)
124         if (i > (rev[i]>>shift))
125             swap(a[i], a[rev[i]>>shift]);
126
127     for (int len = 2, half = 1, div = maxn>>1; len <= n; len<=1, half<=1, div>=1) {
128         for (int i = 0; i < n; i += len) {
129             for (int j = 0; j < half; j++) {
130                 T u = a[i+j];
131                 T v = a[i+j+half] * (inv ? iX[j*div] : X[j*div]) % MOD;
132                 a[i+j] = (u+v >= MOD ? u+v-MOD : u+v);
133                 a[i+j+half] = (u-v < 0 ? u-v+MOD : u-v);
134             }
135         }
136     }
137
138     if (inv) {
139         T dn = pw(n, MOD-2);
140         for (auto& x : a) {
141             x *= dn;
142             if (x >= MOD) x %= MOD;
143         }
144     }
145 }
146
147 template<typename T>
148 inline void resize(vector<T>& a) {
149     int cnt = (int)a.size();
150     for (; cnt > 0; cnt--) if (a[cnt-1]) break;
151     a.resize(max(cnt, 1));
152 }
153
154 template<typename T>
155 vector<T>& operator*=(vector<T>& a, vector<T> b) {
156     int na = (int)a.size();
157     int nb = (int)b.size();
158     a.resize(na + nb - 1, 0);
159     b.resize(na + nb - 1, 0);
160 }

```

```

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

```

```

153 NTT(a); NTT(b);
154 for (int i = 0; i < (int)a.size(); i++) {
155     a[i] *= b[i];
156     if (a[i] >= MOD) a[i] %= MOD;
157 }
158 NTT(a, true);
159
160 resize(a);
161 return a;
162 }
163
164 template<typename T>
165 void inv(vector<T>& ia, int N) {
166     vector<T> _a(move(ia));
167     ia.resize(1, pw(_a[0], MOD-2));
168     vector<T> a(1, -_a[0] + (-_a[0] < 0 ? MOD : 0));
169
170     for (int n = 1; n < N; n<=1) {
171         // n -> 2*n
172         // ia' = ia(2-a*ia);
173
174         for (int i = n; i < min(siz(_a), (n<<1)); i++)
175             a.emplace_back(-_a[i] + (-_a[i] < 0 ? MOD : 0));
176
177         vector<T> tmp = ia;
178         ia *= a;
179         ia.resize(n<<1);
180         ia[0] = ia[0] + 2 >= MOD ? ia[0] + 2 - MOD : ia[0] + 2;
181         ia *= tmp;
182         ia.resize(n<<1);
183     }
184     ia.resize(N);
185 }
186
187 template<typename T>
188 void mod(vector<T>& a, vector<T>& b) {
189     int n = (int)a.size()-1, m = (int)b.size()-1;
190     if (n < m) return;
191
192     vector<T> ra = a, rb = b;
193     reverse(ra.begin(), ra.end()); ra.resize(min(n+1, n-m+1));
194     reverse(rb.begin(), rb.end()); rb.resize(min(m+1, n-m+1));
195
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-1])
210
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 use poly mod to get R(x))
213 Let r[i] = the coefficient of x^i in R(x)
214 => 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; vector<vector<ll>> > v;
2 void gauss(vector<vector<ll>>& v) {
3     int r = 0;
4     for (int i = 0; i < n; i++) {
5         bool ok = false;
6         for (int j = r; j < n; j++) {
7             if (v[j][i] == 0) continue;

```

```

8             swap(v[j], v[r]);
9             ok = true; break;
10         }
11         if (!ok) continue;
12         ll div = inv(v[r][i]);
13         for (int j = 0; j < n+1; j++) {
14             v[r][j] *= div;
15             if (v[r][j] >= MOD) v[r][j] %= MOD;
16         }
17         for (int j = 0; j < n; j++) {
18             if (j == r) continue;
19             ll t = v[j][i];
20             for (int k = 0; k < n+1; k++) {
21                 v[j][k] -= v[r][k] * t % MOD;
22                 if (v[j][k] < 0) v[j][k] += MOD;
23             }
24             r++;
25         }

```

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

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

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

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

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

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

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

