

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	PBDS	1
2.4	Random	1
2.5	pragma	1
2.6	set map pq cmp	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
3.9	Dynamic Median	4
3.10	SOS DP	4
4	Flow / Matching	4
4.1	Dinic	4
4.2	MCMF	4
4.3	KM	5
4.4	Hopcroft-Karp	5
4.5	Blossom	6
4.6	Weighted Blossom	6
4.7	Cover / Independent Set	8
4.8	Hungarian Algorithm	8
5	Graph	8
5.1	Heavy-Light Decomposition	8
5.2	Centroid Decomposition	9
5.3	Bellman-Ford + SPFA	9
5.4	BCC - AP	10
5.5	BCC - Bridge	10
5.6	SCC - Tarjan	11
5.7	SCC - Kosaraju	11
5.8	Eulerian Path - Undir	11
5.9	Eulerian Path - Dir	12
5.10	Hamilton Path	12
5.11	Kth Shortest Path	12
5.12	System of Difference Constraints	13

6	String	13
6.1	Aho Corasick	13
6.2	KMP	14
6.3	Z Value	14
6.4	Manacher	14
6.5	Suffix Array	14
6.6	Minimum Rotation	15
6.7	Lyndon Factorization	15
6.8	Rolling Hash	15
6.9	Trie	15
7	Geometry	15
7.1	Basic Operations	15
7.2	Sort by Angle	16
7.3	Intersection	16
7.4	Polygon Area	16
7.5	Convex Hull	16
7.6	Point In Convex	16
7.7	Point Segment Distance	17
7.8	Point in Polygon	17
7.9	Minimum Euclidean Distance	17
7.10	Minkowski Sum	17
7.11	Lower Concave Hull	17
7.12	Pick's Theorem	17
7.13	Rotating SweepLine	17
7.14	Half Plane Intersection	18
7.15	Minimum Enclosing Circle	18
7.16	Union of Circles	18
7.17	Area Of Circle Polygon	18
7.18	3D Point	19
8	Number Theory	19
8.1	FFT	19
8.2	Pollard's rho	20
8.3	Miller Rabin	20
8.4	Fast Power	20
8.5	Extend GCD	20
8.6	Mu + Phi	20
8.7	Discrete Log	20
8.8	sqrt mod	21
8.9	Primitive Root	21
8.10	Other Formulas	21
8.11	Polynomial	22
9	Linear Algebra	23
9.1	Gaussian-Jordan Elimination	23
9.2	Determinant	23
10	Combinatorics	23
10.1	Catalan Number	23
10.2	Burnside's Lemma	23
11	Special Numbers	23
11.1	Fibonacci Series	23
11.2	Prime Numbers	24

2 Basic

2.1 Vimrc

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

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

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

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

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

2.2 Runcpp.sh

```
#!/bin/bash
clear
echo "Start compiling $1..."
echo
g++ -O2 -std=c++20 -Wall -Wextra -Wshadow $2/$1 -o $2/
out
if [ "$?" -ne 0 ]
then
exit 1
fi
echo
echo "Done compiling"
echo "===== "
echo
echo "Input file:"
echo
cat $2/in.txt
echo
echo "===== "
echo
declare startTime=`date +%s%N`
$2/out < $2/in.txt > $2/out.txt
declare endTime=`date +%s%N`
delta=`expr $endTime - $startTime`
delta=`expr $delta / 1000000`
cat $2/out.txt
echo
echo "time: $delta ms"
```

2.3 PBDS

```
#include <bits/extc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

// map
tree<int, int, less<>, rb_tree_tag,
tree_order_statistics_node_update> tr;
tr.order_of_key(element);
tr.find_by_order(rank);

// set
tree<int, null_type, less<>, rb_tree_tag,
tree_order_statistics_node_update> tr;
tr.order_of_key(element);
tr.find_by_order(rank);

// hash table
```

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!

```

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); // join

```

2.4 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);

```

2.5 pragma

```

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

```

2.6 set map pq cmp

```

1 struct edge
2 {
3     int a, b, w;
4     friend istream& operator>>(istream &in, edge &x)
5     { in >> x.a >> x.b >> x.w; }
6     friend ostream& operator<<(ostream &out, const edge
7       &x)
8     { out << "(" << x.a << ", " << x.b << ", " << x.w
9       << ")"; return out; }
10 };
11 struct cmp
12 { bool operator()(const edge &x, const edge &y)
13   const { return x.w < y.w; } };
14 set<edge, cmp> st; //遞增
15 map<edge, long long, cmp> mp; //遞增
16 priority_queue<edge, vector<edge>, cmp> pq; //遞減

```

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

```

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) {

```

```

23 // debug(q1, qr, x); return;
24 if (qr <= 1 || r <= q1) return;
25 if (q1 <= 1 && r <= qr) {
26     arr[i].push_back(x);
27     return;
28 }
29 if (qr <= m)
30     insert(q1, qr, x, i << 1, 1, m);
31 else if (m <= q1)
32     insert(q1, qr, x, i << 1 | 1, m, r);
33 else {
34     insert(q1, qr, x, i << 1, 1, m);
35     insert(q1, qr, x, i << 1 | 1, m, r);
36 }
37 }
38 void traversal(V<int>& ans, int i = 1, int l = 0, int r
39             = q) {
40     int opcnt = 0;
41     // debug(i, l, r);
42     for (auto [a, b] : arr[i])
43         if (merge(a, b))
44             opcnt++, cnt--;
45     if (r - l == 1)
46         ans[l] = cnt;
47     else {
48         traversal(ans, i << 1, l, m);
49         traversal(ans, i << 1 | 1, m, r);
50     }
51     while (opcnt--)
52         undo(), cnt++;
53     arr[i].clear();
54 }
55 #undef m
56 inline void solve() {
57     int n, m;
58     cin >> n >> m >> q, q++;
59     dsu.resize(cnt = n), sz.assign(n, 1);
60     iota(dsu.begin(), dsu.end(), 0);
61     // a, b, time, operation
62     unordered_map<ll, V<int>> s;
63     for (int i = 0; i < m; i++) {
64         int a, b;
65         cin >> a >> b;
66         if (a > b) swap(a, b);
67         s[(((ll)a << 32) | b).emplace_back(0);
68     }
69     for (int i = 1; i < q; i++) {
70         int op, a, b;
71         cin >> op >> a >> b;
72         if (a > b) swap(a, b);
73         switch (op) {
74             case 1:
75                 s[(((ll)a << 32) | b).push_back(i);
76                 break;
77             case 2:
78                 auto tmp = s[(((ll)a << 32) | b).back();
79                 s[(((ll)a << 32) | b).pop_back();
80                 insert(tmp, i, P<int>{a, b});
81         }
82     }
83     for (auto [p, v] : s) {
84         int a = p >> 32, b = p & -1;
85         while (v.size()) {
86             insert(v.back(), q, P<int>{a, b});
87             v.pop_back();
88         }
89     }
90     V<int> ans(q);
91     traversal(ans);
92     for (auto i : ans)
93         cout << i << ' ';
94     cout << endl;
95 }

```

3.9 Dynamic Median

```

1 struct Dynamic_Median {
2     multiset<long long> lo, hi;
3     long long slo = 0, shi = 0;
4     void rebalance() {
5         // keep sz(lo) >= sz(hi) and sz(lo) - sz(hi) <=
6         1

```

```

6         while((int)lo.size() > (int)hi.size() + 1) {
7             auto it = prev(lo.end());
8             long long x = *it;
9             lo.erase(it); slo -= x;
10            hi.insert(x); shi += x;
11        }
12        while((int)lo.size() < (int)hi.size()) {
13            auto it = hi.begin();
14            long long x = *it;
15            hi.erase(it); shi -= x;
16            lo.insert(x); slo += x;
17        }
18    }
19    void add(long long x) {
20        if(lo.empty() || x <= *prev(lo.end())) {
21            lo.insert(x); slo += x;
22        }
23        else {
24            hi.insert(x); shi += x;
25        }
26        rebalance();
27    }
28    void remove_one(long long x) {
29        if(!lo.empty() && x <= *prev(lo.end())) {
30            auto it = lo.find(x);
31            if(it != lo.end()) {
32                lo.erase(it); slo -= x;
33            }
34            else {
35                auto it2 = hi.find(x);
36                hi.erase(it2); shi -= x;
37            }
38        }
39        else {
40            auto it = hi.find(x);
41            if(it != hi.end()) {
42                hi.erase(it); shi -= x;
43            }
44            else {
45                auto it2 = lo.find(x);
46                lo.erase(it2); slo -= x;
47            }
48        }
49        rebalance();
50    }
51 };

```

3.10 SOS DP

```

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

```

4 Flow / Matching

4.1 Dinic

```

1 struct Dinic {
2     struct Edge { int to, cap, rev; };
3     int n, s, t;
4     vector<vector<Edge>> g;
5     vector<int> level, it;
6
7     void init(int _n, int _s, int _t){
8         n=_n; s=_s; t=_t;
9         g.assign(n, {});
10        level.assign(n, 0);
11        it.assign(n, 0);
12    }
13    void add(int a, int b, int c){
14        Edge f{b, c, (int)g[b].size()};
15        Edge r{a, 0, (int)g[a].size()};
16        g[a].push_back(f);
17        g[b].push_back(r);
18    }
19    bool bfs(){
20        fill(level.begin(), level.end(), -1);
21        queue<int> q; level[s]=0; q.push(s);
22        while(!q.empty()){

```

```

23     int u=q.front(); q.pop();
24     for(const auto &e: g[u]){
25         if(e.cap>0 && level[e.to]==-1){
26             level[e.to]=level[u]+1;
27             q.push(e.to);
28         }
29     }
30 }
31 return level[t]!=-1;
32 }
33 int dfs(int u,int f){
34     if(!f || u==t) return f;
35     for(int &i=it[u]; i<(int)g[u].size(); ++i){
36         auto &e=g[u][i];
37         if(e.cap>0 && level[e.to]==level[u]+1){
38             int got=dfs(e.to, min(f, e.cap));
39             if(got){
40                 e.cap-=got;
41                 g[e.to][e.rev].cap+=got;
42                 return got;
43             }
44         }
45     }
46     return 0;
47 }
48 int maxflow(){
49     int flow=0, add;
50     while(bfs()){
51         fill(it.begin(), it.end(), 0);
52         while((add=dfs(s, INF))) flow+=add;
53     }
54     return flow;
55 }
56 };

```

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) {

```

```

44        spfa();
45        if (dis[t] == INF)
46            break;
47        int mn = INF;
48        for (int i = t; i != s; i = par[i])
49            mn = min(mn, path[par[i]][p_i[i]].cap);
50        flow += mn;
51        cost += dis[t] * mn;
52        for (int i = t; i != s; i = par[i]) {
53            edge &now = path[par[i]][p_i[i]];
54            now.cap -= mn;
55            path[i][now.rev].cap += mn;
56        }
57    }
58    return mp(flow, cost);
59 }
60 };

```

4.3 KM

```

1 struct KM {
2     int n, mx[1005], my[1005], pa[1005];
3     int g[1005][1005], lx[1005], ly[1005], sy[1005];
4     bool vx[1005], vy[1005];
5     void init(int _n) {
6         n = _n;
7         FOR(i, 1, n + 1)
8             fill(g[i], g[i] + 1 + n, 0);
9     }
10    void add(int a, int b, int c) { g[a][b] = c; }
11    void augment(int y) {
12        for (int x, z; y; y = z)
13            x = pa[y], z = mx[x], my[y] = x, mx[x] = y;
14    }
15    void bfs(int st) {
16        FOR(i, 1, n + 1)
17            sy[i] = INF,
18            vx[i] = vy[i] = 0;
19        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                }
55                vy[y] = 1;
56                q.push(my[y]);
57            }
58        }
59    }
60    int solve() {

```

```

63     fill(mx, mx + n + 1, 0);
64     fill(my, my + n + 1, 0);
65     fill(ly, ly + n + 1, 0);
66     fill(lx, lx + n + 1, 0);
67     FOR(x, 1, n + 1)
68     FOR(y, 1, n + 1)
69     lx[x] = max(lx[x], g[x][y]);
70     FOR(x, 1, n + 1)
71     bfs(x);
72     int ans = 0;
73     FOR(y, 1, n + 1)
74     ans += g[my[y]][y];
75     return ans;
76 }
77 };

```

4.4 Hopcroft-Karp

```

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

```

```

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) {

```



```

18     if (!slack[x] || e_delta(g[u][x]) < e_delta(g[ 89
19         slack[x]][x])) slack[x] = u;
20 }
21 void set_slack(int x) {
22     slack[x] = 0;
23     for (int u = 1; u <= n; ++u)
24         if (g[u][x].w > 0 && st[u] != x && S[st[u]]
25             == 0)
26             update_slack(u, x);
27 }
28 void q_push(int x) {
29     if (x <= n)
30         q.push(x);
31     else
32         for (size_t i = 0; i < flo[x].size(); i++)
33             q_push(flo[x][i]);
34 }
35 void set_st(int x, int b) {
36     st[x] = b;
37     if (x > n)
38         for (size_t i = 0; i < flo[x].size(); ++i)
39             set_st(flo[x][i], b);
40 }
41 int get_pr(int b, int xr) {
42     int pr = find(flo[b].begin(), flo[b].end(), xr)
43         - flo[b].begin();
44     if (pr % 2 == 1) {
45         reverse(flo[b].begin() + 1, flo[b].end());
46         return (int)flo[b].size() - pr;
47     }
48     return pr;
49 }
50 void set_match(int u, int v) {
51     match[u] = g[u][v].v;
52     if (u <= n) return;
53     edge e = g[u][v];
54     int xr = flo_from[u][e.u], pr = get_pr(u, xr);
55     for (int i = 0; i < pr; ++i) set_match(flo[u][i]
56         , flo[u][i ^ 1]);
57     set_match(xr, v);
58     rotate(flo[u].begin(), flo[u].begin() + pr, flo
59         [u].end());
60 }
61 void augment(int u, int v) {
62     for (;;) {
63         int xnv = st[match[u]];
64         set_match(u, v);
65         if (!xnv) return;
66         set_match(xnv, st[pa[xnv]]);
67         u = st[pa[xnv]], v = xnv;
68     }
69 }
70 int get_lca(int u, int v) {
71     static int t = 0;
72     for (++t; u || v; swap(u, v)) {
73         if (u == 0) continue;
74         if (vis[u] == t) return u;
75         vis[u] = t;
76         u = st[match[u]];
77         if (u) u = st[pa[u]];
78     }
79     return 0;
80 }
81 void add_blossom(int u, int lca, int v) {
82     int b = n + 1;
83     while (b <= n_x && st[b]) ++b;
84     if (b > n_x) ++n_x;
85     lab[b] = 0, S[b] = 0;
86     match[b] = match[lca];
87     flo[b].clear();
88     flo[b].push_back(lca);
89     for (int x = u, y; x != lca; x = st[pa[y]])
90         flo[b].push_back(x), flo[b].push_back(y =
91             st[match[x]]), q_push(y);
92     reverse(flo[b].begin() + 1, flo[b].end());
93     for (int x = v, y; x != lca; x = st[pa[y]])
94         flo[b].push_back(x), flo[b].push_back(y =
95             st[match[x]]), q_push(y);
96     set_st(b, b);
97     for (int x = 1; x <= n_x; ++x) g[b][x].w = g[x
98         ][b].w = 0;
99 }
100
101 for (int x = 1; x <= n; ++x) flo_from[b][x] =
102     0;
103 for (size_t i = 0; i < flo[b].size(); ++i) {
104     int xs = flo[b][i];
105     for (int x = 1; x <= n_x; ++x)
106         if (g[b][x].w == 0 || e_delta(g[xs][x])
107             < e_delta(g[b][x]))
108             g[b][x] = g[xs][x], g[x][b] = g[x][
109                 xs];
110     for (int x = 1; x <= n; ++x)
111         if (flo_from[xs][x]) flo_from[b][x] =
112             xs;
113 }
114 set_slack(b);
115 }
116 void expand_blossom(int b) {
117     for (size_t i = 0; i < flo[b].size(); ++i)
118         set_st(flo[b][i], flo[b][i]);
119     int xr = flo_from[b][g[b][pa[b]].u], pr =
120         get_pr(b, xr);
121     for (int i = 0; i < pr; i += 2) {
122         int xs = flo[b][i], xns = flo[b][i + 1];
123         pa[xs] = g[xns][xs].u;
124         S[xs] = 1, S[xns] = 0;
125         slack[xs] = 0, set_slack(xns);
126         q_push(xns);
127     }
128     S[xr] = 1, pa[xr] = pa[b];
129     for (size_t i = pr + 1; i < flo[b].size(); ++i)
130         {
131             int xs = flo[b][i];
132             S[xs] = -1, set_slack(xs);
133         }
134     st[b] = 0;
135 }
136 bool on_found_edge(const edge &e) {
137     int u = st[e.u], v = st[e.v];
138     if (S[v] == -1) {
139         pa[v] = e.u, S[v] = 1;
140         int nu = st[match[v]];
141         slack[v] = slack[nu] = 0;
142         S[nu] = 0, q_push(nu);
143     } else if (S[v] == 0) {
144         int lca = get_lca(u, v);
145         if (!lca)
146             return augment(u, v), augment(v, u),
147                 true;
148         else
149             add_blossom(u, lca, v);
150     }
151     return false;
152 }
153 bool matching() {
154     memset(S + 1, -1, sizeof(int) * n_x);
155     memset(slack + 1, 0, sizeof(int) * n_x);
156     q = queue<int>();
157     for (int x = 1; x <= n_x; ++x)
158         if (st[x] == x && !match[x]) pa[x] = 0, S[x
159             ] = 0, q_push(x);
160     if (q.empty()) return false;
161     for (;;) {
162         while (q.size()) {
163             int u = q.front();
164             q.pop();
165             if (S[st[u]] == 1) continue;
166             for (int v = 1; v <= n; ++v)
167                 if (g[u][v].w > 0 && st[u] != st[v
168                     ]) {
169                     if (e_delta(g[u][v]) == 0) {
170                         if (on_found_edge(g[u][v]))
171                             return true;
172                     } else
173                         update_slack(u, st[v]);
174                 }
175         }
176         int d = inf;
177         for (int b = n + 1; b <= n_x; ++b)
178             if (st[b] == b && S[b] == 1) d = min(d,
179                 lab[b] / 2);
180         for (int x = 1; x <= n_x; ++x)
181             if (st[x] == x && slack[x]) {
182                 if (S[x] == -1)

```

```

160         d = min(d, e_delta(g[slack[x]](
161             x)));
162         else if (S[x] == 0)
163             d = min(d, e_delta(g[slack[x]](
164                 x)) / 2);
165     }
166     for (int u = 1; u <= n; ++u) {
167         if (S[st[u]] == 0) {
168             if (lab[u] <= d) return 0;
169             lab[u] -= d;
170         } else if (S[st[u]] == 1)
171             lab[u] += d;
172     }
173     for (int b = n + 1; b <= n_x; ++b)
174         if (st[b] == b) {
175             if (S[st[b]] == 0)
176                 lab[b] += d * 2;
177             else if (S[st[b]] == 1)
178                 lab[b] -= d * 2;
179         }
180     q = queue<int>();
181     for (int x = 1; x <= n_x; ++x)
182         if (st[x] == x && slack[x] && st[slack[
183             x]] != x && e_delta(g[slack[x]](x))
184             == 0)
185             if (on_found_edge(g[slack[x]](x)))
186                 return true;
187     for (int b = n + 1; b <= n_x; ++b)
188         if (st[b] == b && S[b] == 1 && lab[b]
189             == 0) expand_blossom(b);
190 }
191 return false;
192 }
193 pair<long long, int> solve() {
194     memset(match + 1, 0, sizeof(int) * n);
195     n_x = n;
196     int n_matches = 0;
197     long long tot_weight = 0;
198     for (int u = 0; u <= n; ++u) st[u] = u, flo[u].
199         clear();
200     int w_max = 0;
201     for (int u = 1; u <= n; ++u)
202         for (int v = 1; v <= n; ++v) {
203             flo_from[u][v] = (u == v ? u : 0);
204             w_max = max(w_max, g[u][v].w);
205         }
206     for (int u = 1; u <= n; ++u) lab[u] = w_max;
207     while (matching()) ++n_matches;
208     for (int u = 1; u <= n; ++u)
209         if (match[u] && match[u] < u)
210             tot_weight += g[u][match[u]].w;
211     return make_pair(tot_weight, n_matches);
212 }
213 void add_edge(int ui, int vi, int wi) { g[ui][vi].w
214     = g[vi][ui].w = wi; }
215 void init(int _n) {
216     n = _n;
217     for (int u = 1; u <= n; ++u)
218         for (int v = 1; v <= n; ++v)
219             g[u][v] = edge(u, v, 0);
220 }
221 };

```

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}

```

4.8 Hungarian Algorithm

```

1 const int N = 2e3;
2 int match[N];
3 bool vis[N];
4 int n;
5 vector<int> ed[N];
6 int match_cnt;
7 bool dfs(int u) {
8     vis[u] = 1;
9     for (int i : ed[u]) {
10         if (match[i] == 0 || !vis[match[i]] && dfs(match
11             [i])) {
12             match[i] = u;
13             return true;
14         }
15     }
16     return false;
17 }
18 void hungary() {
19     memset(match, 0, sizeof(match));
20     match_cnt = 0;
21     for (int i = 1; i <= n; i++) {
22         memset(vis, 0, sizeof(vis));
23         if (dfs(i)) match_cnt++;
24     }
25 }

```

5 Graph

5.1 Heavy-Light Decomposition

```

1 const int N = 2e5 + 5;
2 int n, dfn[N], son[N], top[N], num[N], dep[N], p[N];
3 vector<int> path[N];
4 struct node {
5     int mx, sum;
6 } seg[N << 2];
7 void update(int x, int l, int r, int qx, int val) {
8     if (l == r) {
9         seg[x].mx = seg[x].sum = val;
10        return;
11    }
12    int mid = (l + r) >> 1;
13    if (qx <= mid) update(x << 1, l, mid, qx, val);
14    else update(x << 1 | 1, mid + 1, r, qx, val);
15    seg[x].mx = max(seg[x << 1].mx, seg[x << 1 | 1].mx);
16    seg[x].sum = seg[x << 1].sum + seg[x << 1 | 1].sum;
17 }
18 int big(int x, int l, int r, int ql, int qr) {
19     if (ql <= l && r <= qr) return seg[x].mx;
20     int mid = (l + r) >> 1;
21     int res = -INF;
22     if (ql <= mid) res = max(res, big(x << 1, l, mid,
23         ql, qr));
24     if (mid < qr) res = max(res, big(x << 1 | 1, mid +
25         1, r, ql, qr));
26     return res;
27 }
28 int ask(int x, int l, int r, int ql, int qr) {
29     if (ql <= l && r <= qr) return seg[x].sum;
30     int mid = (l + r) >> 1;
31     int res = 0;
32     if (ql <= mid) res += ask(x << 1, l, mid, ql, qr);
33     if (mid < qr) res += ask(x << 1 | 1, mid + 1, r, ql
34         , qr);
35     return res;
36 }
37 void dfs1(int now) {
38     son[now] = -1;
39     num[now] = 1;
40     for (auto i : path[now]) {
41         if (!dep[i]) {
42             dep[i] = dep[now] + 1;
43             p[i] = now;
44             dfs1(i);
45             num[now] += num[i];
46             if (son[now] == -1 || num[i] > num[son[now]
47                 ]) son[now] = i;
48         }
49     }
50 }

```



```

45     }
46 }
47 int cnt;
48 void dfs2(int now, int t) {
49     top[now] = t;
50     cnt++;
51     dfn[now] = cnt;
52     if (son[now] == -1) return;
53     dfs2(son[now], t);
54     for (auto i : path[now])
55         if (i != p[now] && i != son[now]) dfs2(i, i);
56 }
57 int path_big(int x, int y) {
58     int res = -INF;
59     while (top[x] != top[y]) {
60         if (dep[top[x]] < dep[top[y]]) swap(x, y);
61         res = max(res, big(1, 1, n, dfn[top[x]], dfn[x]));
62         x = p[top[x]];
63     }
64     if (dfn[x] > dfn[y]) swap(x, y);
65     res = max(res, big(1, 1, n, dfn[x], dfn[y]));
66     return res;
67 }
68 int path_sum(int x, int y) {
69     int res = 0;
70     while (top[x] != top[y]) {
71         if (dep[top[x]] < dep[top[y]]) swap(x, y);
72         res += ask(1, 1, n, dfn[top[x]], dfn[x]);
73         x = p[top[x]];
74     }
75     if (dfn[x] > dfn[y]) swap(x, y);
76     res += ask(1, 1, n, dfn[x], dfn[y]);
77     return res;
78 }
79 void buildTree() {
80     FOR(i, 0, n - 1) {
81         int a, b;
82         cin >> a >> b;
83         path[a].pb(b);
84         path[b].pb(a);
85     }
86 }
87 void buildHLD(int root) {
88     dep[root] = 1;
89     dfs1(root);
90     dfs2(root, root);
91     FOR(i, 1, n + 1) {
92         int now;
93         cin >> now;
94         update(1, 1, n, dfn[i], now);
95     }
96 }

```

5.2 Centroid Decomposition

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int N = 1e5 + 5;
4 vector<int> a[N];
5 int sz[N], lv[N];
6 bool used[N];
7 int f_sz(int x, int p) {
8     sz[x] = 1;
9     for (int i : a[x])
10         if (i != p && !used[i])
11             sz[x] += f_sz(i, x);
12     return sz[x];
13 }
14 int f_cen(int x, int p, int total) {
15     for (int i : a[x]) {
16         if (i != p && !used[i] && 2 * sz[i] > total)
17             return f_cen(i, x, total);
18     }
19     return x;
20 }
21 void cd(int x, int p) {
22     int total = f_sz(x, p);
23     int cen = f_cen(x, p, total);
24     lv[cen] = lv[p] + 1;
25     used[cen] = 1;

```

```

26 // cout << "cd: " << x << " " << p << " " << cen <<
27 // "\n";
28 for (int i : a[cen]) {
29     if (!used[i])
30         cd(i, cen);
31 }
32 int main() {
33     ios_base::sync_with_stdio(0);
34     cin.tie(0);
35     int n;
36     cin >> n;
37     for (int i = 0, x, y; i < n - 1; i++) {
38         cin >> x >> y;
39         a[x].push_back(y);
40         a[y].push_back(x);
41     }
42     cd(1, 0);
43     for (int i = 1; i <= n; i++)
44         cout << (char)('A' + lv[i] - 1) << " ";
45     cout << "\n";
46 }

```

5.3 Bellman-Ford + SPFA

```

1 int n, m;
2
3 // Graph
4 vector<vector<pair<int, ll> > > g;
5 vector<ll> dis;
6 vector<bool> negCycle;
7
8 // SPFA
9 vector<int> rlx;
10 queue<int> q;
11 vector<bool> inq;
12 vector<int> pa;
13 void SPFA(vector<int>& src) {
14     dis.assign(n + 1, LINF);
15     negCycle.assign(n + 1, false);
16     rlx.assign(n + 1, 0);
17     while (!q.empty()) q.pop();
18     inq.assign(n + 1, false);
19     pa.assign(n + 1, -1);
20
21     for (auto& s : src) {
22         dis[s] = 0;
23         q.push(s);
24         inq[s] = true;
25     }
26
27     while (!q.empty()) {
28         int u = q.front();
29         q.pop();
30         inq[u] = false;
31         if (rlx[u] >= n) {
32             negCycle[u] = true;
33         } else
34             for (auto& e : g[u]) {
35                 int v = e.first;
36                 ll w = e.second;
37                 if (dis[v] > dis[u] + w) {
38                     dis[v] = dis[u] + w;
39                     rlx[v] = rlx[u] + 1;
40                     pa[v] = u;
41                     if (!inq[v]) {
42                         q.push(v);
43                         inq[v] = true;
44                     }
45                 }
46             }
47     }
48 }
49
50 // Bellman-Ford
51 queue<int> q;
52 vector<int> pa;
53 void BellmanFord(vector<int>& src) {
54     dis.assign(n + 1, LINF);
55     negCycle.assign(n + 1, false);
56     pa.assign(n + 1, -1);
57 }

```

```

58 for (auto& s : src) dis[s] = 0;
59
60 for (int rlx = 1; rlx <= n; rlx++) {
61     for (int u = 1; u <= n; u++) {
62         if (dis[u] == LINF) continue; // Important
63         !!
64         for (auto& e : g[u]) {
65             int v = e.first;
66             ll w = e.second;
67             if (dis[v] > dis[u] + w) {
68                 dis[v] = dis[u] + w;
69                 pa[v] = u;
70                 if (rlx == n) negCycle[v] = true;
71             }
72         }
73     }
74 }
75
76 // Negative Cycle Detection
77 void NegCycleDetect() {
78     /* No Neg Cycle: NO
79     Exist Any Neg Cycle: YES
80     v0 v1 v2 ... vk v0 */
81
82     vector<int> src;
83     for (int i = 1; i <= n; i++)
84         src.emplace_back(i);
85
86     SPFA(src);
87     // BellmanFord(src);
88
89     int ptr = -1;
90     for (int i = 1; i <= n; i++)
91         if (negCycle[i]) {
92             ptr = i;
93             break;
94         }
95
96     if (ptr == -1) {
97         return cout << "NO" << endl, void();
98     }
99
100     cout << "YES\n";
101     vector<int> ans;
102     vector<bool> vis(n + 1, false);
103
104     while (true) {
105         ans.emplace_back(ptr);
106         if (vis[ptr]) break;
107         vis[ptr] = true;
108         ptr = pa[ptr];
109     }
110     reverse(ans.begin(), ans.end());
111
112     vis.assign(n + 1, false);
113     for (auto& x : ans) {
114         cout << x << ' ';
115         if (vis[x]) break;
116         vis[x] = true;
117     }
118     cout << endl;
119 }
120
121 // Distance Calculation
122 void calcDis(int s) {
123     vector<int> src;
124     src.emplace_back(s);
125     SPFA(src);
126     // BellmanFord(src);
127
128     while (!q.empty()) q.pop();
129     for (int i = 1; i <= n; i++)
130         if (negCycle[i]) q.push(i);
131
132     while (!q.empty()) {
133         int u = q.front();
134         q.pop();
135         for (auto& e : g[u]) {
136             int v = e.first;
137             if (!negCycle[v]) {

```

```

139         q.push(v);
140         negCycle[v] = true;
141     }
142 }
143 }
144 }

```

5.4 BCC - AP

```

1 int n, m;
2 int low[maxn], dfn[maxn], instp;
3 vector<int> E, g[maxn];
4 bitset<maxn> isap;
5 bitset<maxm> vis;
6 stack<int> stk;
7 int bccnt;
8 vector<int> bcc[maxn];
9 inline void popout(int u) {
10     bccnt++;
11     bcc[bccnt].emplace_back(u);
12     while (!stk.empty()) {
13         int v = stk.top();
14         if (u == v) break;
15         stk.pop();
16         bcc[bccnt].emplace_back(v);
17     }
18 }
19 void dfs(int u, bool rt = 0) {
20     stk.push(u);
21     low[u] = dfn[u] = ++instp;
22     int kid = 0;
23     Each(e, g[u]) {
24         if (vis[e]) continue;
25         vis[e] = true;
26         int v = E[e] ^ u;
27         if (!dfn[v]) {
28             // tree edge
29             kid++;
30             dfs(v);
31             low[u] = min(low[u], low[v]);
32             if (!rt && low[v] >= dfn[u]) {
33                 // bcc found: u is ap
34                 isap[u] = true;
35                 popout(u);
36             }
37         } else {
38             // back edge
39             low[u] = min(low[u], dfn[v]);
40         }
41     }
42     // special case: root
43     if (rt) {
44         if (kid > 1) isap[u] = true;
45         popout(u);
46     }
47 }
48 void init() {
49     cin >> n >> m;
50     fill(low, low + maxn, INF);
51     REP(i, m) {
52         int u, v;
53         cin >> u >> v;
54         g[u].emplace_back(i);
55         g[v].emplace_back(i);
56         E.emplace_back(u ^ v);
57     }
58 }
59 void solve() {
60     FOR(i, 1, n + 1, 1) {
61         if (!dfn[i]) dfs(i, true);
62     }
63     vector<int> ans;
64     int cnt = 0;
65     FOR(i, 1, n + 1, 1) {
66         if (isap[i]) cnt++, ans.emplace_back(i);
67     }
68     cout << cnt << endl;
69     Each(i, ans) cout << i << ' ';
70     cout << endl;
71 }

```

5.5 BCC - Bridge

```

1 int n, m;
2 vector<int> g[maxn], E;
3 int low[maxn], dfn[maxn], instp;
4 int bccnt, bccid[maxn];
5 stack<int> stk;
6 bitset<maxn> vis, isbrg;
7 void init() {
8     cin >> n >> m;
9     REP(i, m) {
10         int u, v;
11         cin >> u >> v;
12         E.emplace_back(u ^ v);
13         g[u].emplace_back(i);
14         g[v].emplace_back(i);
15     }
16     fill(low, low + maxn, INF);
17 }
18 void popout(int u) {
19     bccnt++;
20     while (!stk.empty()) {
21         int v = stk.top();
22         if (v == u) break;
23         stk.pop();
24         bccid[v] = bccnt;
25     }
26 }
27 void dfs(int u) {
28     stk.push(u);
29     low[u] = dfn[u] = ++instp;
30
31     Each(e, g[u]) {
32         if (vis[e]) continue;
33         vis[e] = true;
34
35         int v = E[e] ^ u;
36         if (dfn[v]) {
37             // back edge
38             low[u] = min(low[u], dfn[v]);
39         } else {
40             // tree edge
41             dfs(v);
42             low[u] = min(low[u], low[v]);
43             if (low[v] == dfn[v]) {
44                 isbrg[e] = true;
45                 popout(u);
46             }
47         }
48     }
49 }
50 void solve() {
51     FOR(i, 1, n + 1, 1) {
52         if (!dfn[i]) dfs(i);
53     }
54     vector<pii> ans;
55     vis.reset();
56     FOR(u, 1, n + 1, 1) {
57         Each(e, g[u]) {
58             if (!isbrg[e] || vis[e]) continue;
59             vis[e] = true;
60             int v = E[e] ^ u;
61             ans.emplace_back(mp(u, v));
62         }
63     }
64     cout << (int)ans.size() << endl;
65     Each(e, ans) cout << e.F << ' ' << e.S << endl;
66 }

```

5.6 SCC - Tarjan

```

1 // 2-SAT
2 vector<int> E, g[maxn]; // 1~n, n+1~2n
3 int low[maxn], in[maxn], instp;
4 int sccnt, sccid[maxn];
5 stack<int> stk;
6 bitset<maxn> ins, vis;
7 int n, m;
8 void init() {
9     cin >> m >> n;
10    E.clear();
11    fill(g, g + maxn, vector<int>());

```

```

12    fill(low, low + maxn, INF);
13    memset(in, 0, sizeof(in));
14    instp = 1;
15    sccnt = 0;
16    memset(sccid, 0, sizeof(sccid));
17    ins.reset();
18    vis.reset();
19 }
20 inline int no(int u) {
21     return (u > n ? u - n : u + n);
22 }
23 int ecnt = 0;
24 inline void clause(int u, int v) {
25     E.eb(no(u) ^ v);
26     g[no(u)].eb(ecnt++);
27     E.eb(no(v) ^ u);
28     g[no(v)].eb(ecnt++);
29 }
30 void dfs(int u) {
31     in[u] = instp++;
32     low[u] = in[u];
33     stk.push(u);
34     ins[u] = true;
35
36     Each(e, g[u]) {
37         if (vis[e]) continue;
38         vis[e] = true;
39
40         int v = E[e] ^ u;
41         if (ins[v])
42             low[u] = min(low[u], in[v]);
43         else if (!in[v]) {
44             dfs(v);
45             low[u] = min(low[u], low[v]);
46         }
47     }
48     if (low[u] == in[u]) {
49         sccnt++;
50         while (!stk.empty()) {
51             int v = stk.top();
52             stk.pop();
53             ins[v] = false;
54             sccid[v] = sccnt;
55             if (u == v) break;
56         }
57     }
58 }
59 int main() {
60     init();
61     REP(i, m) {
62         char su, sv;
63         int u, v;
64         cin >> su >> u >> sv >> v;
65         if (su == '-') u = no(u);
66         if (sv == '-') v = no(v);
67         clause(u, v);
68     }
69     FOR(i, 1, 2 * n + 1, 1) {
70         if (!in[i]) dfs(i);
71     }
72     FOR(u, 1, n + 1, 1) {
73         int du = no(u);
74         if (sccid[u] == sccid[du]) {
75             return cout << "IMPOSSIBLE\n", 0;
76         }
77     }
78     FOR(u, 1, n + 1, 1) {
79         int du = no(u);
80         cout << (sccid[u] < sccid[du] ? '+' : '-') << '
81         ' << endl;
82     }
83 }

```

5.7 SCC - Kosaraju

```

1 const int N = 1e5 + 10;
2 vector<int> ed[N], ed_b[N]; // 反邊
3 vector<int> SCC(N); // 最後SCC的分組
4 bitset<N> vis;
5 int SCC_cnt;
6 int n, m;

```

```

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

```

5.8 Eulerian Path - Undir

```

1 // from 1 to n
2 #define gg return cout << "IMPOSSIBLE\n", void();
3
4 int n, m;
5 vector<int> g[maxn];
6 bitset<maxn> inodd;
7
8 void init() {
9     cin >> n >> m;
10    inodd.reset();
11    for (int i = 0; i < m; i++) {
12        int u, v;
13        cin >> u >> v;
14        inodd[u] = inodd[u] ^ true;
15        inodd[v] = inodd[v] ^ true;
16        g[u].emplace_back(v);
17        g[v].emplace_back(u);
18    }
19 }
20 stack<int> stk;
21 void dfs(int u) {
22     while (!g[u].empty()) {
23         int v = g[u].back();
24         g[u].pop_back();
25         dfs(v);
26     }
27     stk.push(u);
28 }

```

5.9 Eulerian Path - Dir

```

1 // from node 1 to node n
2 #define gg return cout << "IMPOSSIBLE\n", 0
3
4 int n, m;
5 vector<int> g[maxn];
6 stack<int> stk;
7 int in[maxn], out[maxn];
8
9 void init() {
10    cin >> n >> m;
11    for (int i = 0; i < m; i++) {
12        int u, v;
13        cin >> u >> v;

```

```

14        g[u].emplace_back(v);
15        out[u]++, in[v]++;
16    }
17    for (int i = 1; i <= n; i++) {
18        if (i == 1 && out[i] - in[i] != 1) gg;
19        if (i == n && in[i] - out[i] != 1) gg;
20        if (i != 1 && i != n && in[i] != out[i]) gg;
21    }
22 }
23 void dfs(int u) {
24     while (!g[u].empty()) {
25         int v = g[u].back();
26         g[u].pop_back();
27         dfs(v);
28     }
29     stk.push(u);
30 }
31 void solve() {
32     dfs(1) for (int i = 1; i <= n; i++) if ((int)g[i].
33         size()) gg;
34     while (!stk.empty()) {
35         int u = stk.top();
36         stk.pop();
37         cout << u << ' ';
38     }

```

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;
6         ll d;
7         nd(int ui = 0, int vi = 0, ll di = INF) {
8             u = ui;
9             v = vi;
10            d = di;
11        }
12    };
13    struct heap {
14        nd* edge;
15        int dep;
16        heap* chd[4];
17    };
18    static int cmp(heap* a, heap* b) { return a->edge->
19        d > b->edge->d; }
20    struct node {
21        int v;
22        ll d;
23        heap* H;
24        nd* E;
25        node() {}
26        node(ll _d, int _v, nd* _E) {
27            d = _d;
28            v = _v;
29            E = _E;
30        }
31        node(heap* _H, ll _d) {
32            H = _H;
33            d = _d;
34        }
35        friend bool operator<(node a, node b) { return
36            a.d > b.d; }
37    };
38    int n, k, s, t, dst[N];
39    nd* nxt[N];
40    vector<nd*> g[N], rg[N];
41    heap *nullNd, *head[N];
42    void init(int _n, int _k, int _s, int _t) {
43        n = _n;
44        k = _k;
45        s = _s;
46        t = _t;
47        for (int i = 1; i <= n; i++) {
48            g[i].clear();
49            rg[i].clear();
50            nxt[i] = NULL;
51            head[i] = NULL;
52            dst[i] = -1;
53        }
54    }
55    void addEdge(int ui, int vi, ll di) {
56        nd* e = new nd(ui, vi, di);
57        g[ui].push_back(e);
58        rg[vi].push_back(e);
59    }
60    queue<int> dfsQ;
61    void dijkstra() {
62        while (dfsQ.size()) dfsQ.pop();
63        priority_queue<node> Q;
64        Q.push(node(0, t, NULL));
65        while (!Q.empty()) {
66            node p = Q.top();
67            Q.pop();
68            if (dst[p.v] != -1) continue;
69            dst[p.v] = p.d;
70            nxt[p.v] = p.E;
71            dfsQ.push(p.v);
72            for (auto e : rg[p.v]) Q.push(node(p.d + e
73                ->d, e->u, e));
74        }
75    }
76    heap* merge(heap* curNd, heap* newNd) {
77        if (curNd == nullNd) return newNd;
78        heap* root = new heap;
79        memcpy(root, curNd, sizeof(heap));
80        if (newNd->edge->d < curNd->edge->d) {
81            root->edge = newNd->edge;
82            root->chd[2] = newNd->chd[2];
83            root->chd[3] = newNd->chd[3];
84            newNd->edge = curNd->edge;
85            newNd->chd[2] = curNd->chd[2];
86            newNd->chd[3] = curNd->chd[3];
87        }
88        if (root->chd[0]->dep < root->chd[1]->dep)
89            root->chd[0] = merge(root->chd[0], newNd);
90        else
91            root->chd[1] = merge(root->chd[1], newNd);
92        root->dep = max(root->chd[0]->dep,
93            root->chd[1]->dep) +
94            1;
95        return root;
96    }
97    vector<heap*> V;
98    void build() {
99        nullNd = new heap;
100        nullNd->dep = 0;
101        nullNd->edge = new nd;
102        fill(nullNd->chd, nullNd->chd + 4, nullNd);
103        while (not dfsQ.empty()) {
104            int u = dfsQ.front();
105            dfsQ.pop();
106            if (!nxt[u])
107                head[u] = nullNd;
108            else
109                head[u] = head[nxt[u]->v];
110            V.clear();
111            for (auto& e : g[u]) {
112                int v = e->v;
113                if (dst[v] == -1) continue;
114                e->d += dst[v] - dst[u];
115                if (nxt[u] != e) {
116                    heap* p = new heap;
117                    fill(p->chd, p->chd + 4, nullNd);
118                    p->dep = 1;
119                    p->edge = e;
120                    V.push_back(p);
121                }
122            }
123            if (V.empty()) continue;
124            make_heap(V.begin(), V.end(), cmp);
125            #define L(X) ((X << 1) + 1)
126            #define R(X) ((X << 1) + 2)
127            for (size_t i = 0; i < V.size(); i++) {
128                if (L(i) < V.size())
129                    V[i]->chd[2] = V[L(i)];
130                else
131                    V[i]->chd[2] = nullNd;
132                if (R(i) < V.size())
133                    V[i]->chd[3] = V[R(i)];
134                else
135                    V[i]->chd[3] = nullNd;
136            }
137            head[u] = merge(head[u], V.front());
138        }
139    }
140    vector<ll> ans;
141    void first_K() {
142        ans.clear();
143        priority_queue<node> Q;
144        if (dst[s] == -1) return;
145        ans.push_back(dst[s]);
146        if (head[s] != nullNd)
147            Q.push(node(head[s], dst[s] + head[s]->edge
148                ->d));
149        for (int _ = 1; _ < k and not Q.empty(); _++) {
150            node p = Q.top();
151            Q.pop();
152            ans.push_back(p.d);
153            if (head[p.H->edge->v] != nullNd) {
154                q.H = head[p.H->edge->v];
155                q.d = p.d + q.H->edge->d;
156                Q.push(q);
157            }
158        }
159        for (int i = 0; i < 4; i++)
160            if (p.H->chd[i] != nullNd) {
161                q.H = p.H->chd[i];
162                q.d = p.d - p.H->edge->d + p.H->chd
163                    [i]->edge->d;

```

```

158         Q.push(q);
159     }
160 }
161 }
162 void solve() { // ans[i] stores the i-th shortest
163     path
164     dijkstra();
165     build();
166     first_K(); // ans.size() might less than k
167 }
168 } solver;

```

5.12 System of Difference Constraints

```

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

```

- $x_u - x_v \leq c \Rightarrow \text{add}(v, u, c)$
- $x_u - x_v \geq c \Rightarrow \text{add}(u, v, -c)$
- $x_u - x_v = c \Rightarrow \text{add}(v, u, c), \text{add}(u, v, -c)$
- $x_u \geq c \Rightarrow \text{add super vertex } x_0 = 0, \text{ then } x_u - x_0 \geq c \Rightarrow \text{add}(u, 0, -c)$
- Don't forget non-negative constraints for every variable if specified implicitly.
- Interval sum \Rightarrow Use prefix sum to transform into differential constraints. Don't forget $S_{i+1} - S_i \geq 0$ if x_i needs to be non-negative.
- $\frac{x_u}{x_v} \leq c \Rightarrow \log x_u - \log x_v \leq \log c$

6 String

6.1 Aho Corasick

```

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

```

```

37         if (fr->go[i]) {
38             Node *ptr = fr->fail;
39             while (ptr && !ptr->go[i]) ptr = ptr->fail;
40             fr->go[i]->fail = ptr = (ptr ? ptr->go[i] : root);
41             fr->go[i]->dic = (ptr->cnt ? ptr : ptr->dic);
42             que.push(fr->go[i]);
43         }
44     }
45 }
46 } AC;
47

```

6.2 KMP

```

1 vector<int> f;
2 void buildFailFunction(string &s) {
3     f.resize(s.size(), -1);
4     for (int i = 1; i < s.size(); i++) {
5         int now = f[i - 1];
6         while (now != -1 and s[now + 1] != s[i]) now = f[now];
7         if (s[now + 1] == s[i]) f[i] = now + 1;
8     }
9 }
10
11 void KMPmatching(string &a, string &b) {
12     for (int i = 0, now = -1; i < a.size(); i++) {
13         while (a[i] != b[now + 1] and now != -1) now = f[now];
14         if (a[i] == b[now + 1]) now++;
15         if (now + 1 == b.size()) {
16             cout << "found a match start at position "
17                  << i - now << endl;
18             now = f[now];
19         }
20     }
21 }

```

6.3 Z Value

```

1 string is, it, s;
2 int n;
3 vector<int> z;
4 void init() {
5     cin >> is >> it;
6     s = it + '0' + is;
7     n = (int)s.size();
8     z.resize(n, 0);
9 }
10 void solve() {
11     int ans = 0;
12     z[0] = n;
13     for (int i = 1, l = 0, r = 0; i < n; i++) {
14         if (i <= r) z[i] = min(z[i - l], r - i + 1);
15         while (i + z[i] < n && s[z[i]] == s[i + z[i]])
16             z[i]++;
17         if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
18         if (z[i] == (int)it.size()) ans++;
19     }
20     cout << ans << endl;
21 }

```

6.4 Manacher

```

1 int n;
2 string S, s;
3 vector<int> m;
4 void manacher() {
5     s.clear();
6     s.resize(2 * n + 1, '.');
7     for (int i = 0, j = 1; i < n; i++, j += 2) s[j] = S[i];
8     m.clear();
9     m.resize(2 * n + 1, 0);
10    // m[i] := max k such that s[i-k, i+k] is
11    // palindrome
12    int mx = 0, mxk = 0;
13    for (int i = 1; i < 2 * n + 1; i++) {

```



```

13     if (mx - (i - mx) >= 0) m[i] = min(m[mx - (i - 49
        mx)], mx + mxk - i); 50
14     while (0 <= i - m[i] - 1 && i + m[i] + 1 < 2 * 51
        n + 1 && 52
        s[i - m[i] - 1] == s[i + m[i] + 1]) m[i] 53
        ]++;
15     if (i + m[i] > mx + mxk) mx = i, mxk = m[i]; 54
16 } 55
17 void init() { 56
18     cin >> S; 57
19     n = (int)S.size(); 58
20 } 59
21 void solve() {
22     manacher();
23     int mx = 0, ptr = 0;
24     for (int i = 0; i < 2 * n + 1; i++)
25         if (mx < m[i]) {
26             mx = m[i];
27             ptr = i;
28         }
29     for (int i = ptr - mx; i <= ptr + mx; i++)
30         if (s[i] != '.') cout << s[i];
31     cout << endl;
32 }

```

6.5 Suffix Array

```

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

```

```

for (int i = 0; i < n - 1; i++) {
    if (rk[i] == 0) continue;
    int pi = rk[i];
    int j = suf[pi - 1];
    while (i + k < n && j + k < n && s[i + k]
        == s[j + k]) k++;
    lcp[pi] = k;
    k = max(k - 1, 0);
}
};
SuffixArray suffixarray;

```

6.6 Minimum Rotation

```

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

```

6.7 Lyndon Factorization

```

1 vector<string> duval(string const& s) {
2     int n = s.size();
3     int i = 0;
4     vector<string> factorization;
5     while (i < n) {
6         int j = i + 1, k = i;
7         while (j < n && s[k] <= s[j]) {
8             if (s[k] < s[j])
9                 k = i;
10            else
11                k++;
12            j++;
13        }
14        while (i <= k) {
15            factorization.push_back(s.substr(i, j - k));
16            i += j - k;
17        }
18    }
19    return factorization; // O(n)
}

```

6.8 Rolling Hash

```

1 const ll C = 27;
2 inline int id(char c) { return c - 'a' + 1; }
3 struct RollingHash {
4     string s;
5     int n;
6     ll mod;
7     vector<ll> Cexp, hs;
8     RollingHash(string& _s, ll _mod) : s(_s), n((int)_s
        .size()), mod(_mod) {
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    }
}

```

```

22 inline ll query(int l, int r) {
23     ll res = hs[r] - (l ? hs[l - 1] * Cexp[r - 1 +
24         1] : 0);
25     res = (res % mod + mod) % mod;
26     return res;
27 };

```

6.9 Trie

```

1 pii a[N][26];
2
3 void build(string &s) {
4     static int idx = 0;
5     int n = s.size();
6     for (int i = 0, v = 0; i < n; i++) {
7         pii &now = a[v][s[i] - 'a'];
8         if (now.first != -1)
9             v = now.first;
10        else
11            v = now.first = ++idx;
12        if (i == n - 1)
13            now.second++;
14    }
15 }

```

7 Geometry

7.1 Basic Operations

```

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

```

```

48     return sgn(u ^ v) > 0;
49 }
50 int ori(Pt &o, Pt &a, Pt &b) {
51     return sgn((a - o) ^ (b - o));
52 }
53 struct Line {
54     Pt a, b;
55     Pt dir() { return b - a; }
56 };
57 int PtSide(Pt p, Line L) {
58     return sgn(ori(L.a, L.b, p)); // for int
59     return sgn(ori(L.a, L.b, p) / sqrt(len2(L.a - L.b))
60         );
61 }
62 bool PtOnSeg(Pt p, Line L) {
63     return PtSide(p, L) == 0 and sgn((p - L.a) * (p - L
64         .b)) <= 0;
65 }
66 Pt proj(Pt &p, Line &l) {
67     Pt d = l.b - l.a;
68     T d2 = len2(d);
69     if (sgn(d2) == 0) return l.a;
70     T t = ((p - l.a) * d) / d2;
71     return l.a + d * t;
72 }
73 struct Cir {
74     Pt o;
75     T r;
76 };
77 bool disjunct(Cir a, Cir b) {
78     return sgn(sqrtl(len2(a.o - b.o)) - a.r - b.r) >=
79         0;
80 }
81 bool contain(Cir a, Cir b) {
82     return sgn(a.r - b.r - sqrtl(len2(a.o - b.o))) >=
83         0;
84 }

```

7.2 Sort by Angle

```

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

```

7.3 Intersection

```

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

```

```

24 }
25 vector<Pt> CircleLineInter(Cir c, Line l) {
26     Pt H = proj(c.o, l);
27     Pt dir = unit(l.b - l.a);
28     T h = sqrtl(len2(H - c.o));
29     if (sgn(h - c.r) > 0) return {};
30     T d = sqrtl(max((T)0, c.r * c.r - h * h));
31     if (sgn(d) == 0) return {H};
32     return {H - dir * d, H + dir * d};
33 }

```

7.4 Polygon Area

```

1 // 2 * area
2 T dbPoly_area(vector<Pt>& e) {
3     T res = 0;
4     int sz = e.size();
5     for (int i = 0; i < sz; i++) {
6         res += e[i] ^ e[(i + 1) % sz];
7     }
8     return abs(res);
9 }

```

7.5 Convex Hull

```

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

```

7.6 Point In Convex

```

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

```

7.7 Point Segment Distance

```

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

```

```

13 }
14 double dx0 = double(p.x - q0.x), dy0 = double(p.y -
15     q0.y);
16 double dx1 = double(p.x - q1.x), dy1 = double(p.y -
17     q1.y);
18 return min(sqrt(dx0 * dx0 + dy0 * dy0), sqrt(dx1 *
19     dx1 + dy1 * dy1));
20 }

```

7.8 Point in Polygon

```

1 short inPoly(vector<Pt>& pts, Pt p) {
2     // 0=Bound 1=In -1=Out
3     int n = pts.size();
4     for (int i = 0; i < pts.size(); i++) if (onseg(p,
5         pts[i], pts[(i + 1) % n])) return 0;
6     int cnt = 0;
7     for (int i = 0; i < pts.size(); i++) if (
8         line_intersect_check(p, Pt(p.x + 1, p.y + 2e9),
9         pts[i], pts[(i + 1) % n])) cnt ^= 1;
10    return (cnt ? 1 : -1);
11 }

```

7.9 Minimum Euclidean Distance

```

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

```

7.10 Minkowski Sum

```

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

```

7.11 Lower Concave Hull

```

1 struct Line {
2     mutable ll m, b, p;
3     bool operator<(const Line& o) const { return m < o.m;
4         }
5     bool operator<(ll x) const { return p < x; }
6 }

```

```

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

```

7.12 Pick's Theorem

Consider a polygon which vertices are all lattice points.

Let i = number of points inside the polygon.

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

Then we have the following formula:

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

7.13 Rotating SweepLine

```

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

```

7.14 Half Plane Intersection

```

1 bool cover(Line& L, Line& P, Line& Q) {
2     long double u = (Q.a - P.a) ^ Q.dir();
3     long double v = P.dir() ^ Q.dir();
4     long double x = P.dir().x * u + (P.a - L.a).x * v;
5     long double y = P.dir().y * u + (P.a - L.a).y * v;
6     return sgn(x * L.dir().y - y * L.dir().x) * sgn(v)
7     >= 0;
8 }
9 vector<Line> HPI(vector<Line> P) {
10    sort(P.begin(), P.end(), [&](Line& l, Line& m) {
11        if (argcmp(l.dir(), m.dir()) return true;
12        if (argcmp(m.dir(), l.dir()) return false;
13        return ori(m.a, m.b, l.a) > 0;
14    });

```

```

14 int l = 0, r = -1;
15 for (size_t i = 0; i < P.size(); ++i) {
16     if (i && !argcmp(P[i - 1].dir(), P[i].dir()))
17         continue;
18     while (l < r && cover(P[i], P[r - 1], P[r])) --
19         r;
20     while (l < r && cover(P[i], P[l], P[l + 1])) ++
21         l;
22     P[++r] = P[i];
23 }
24 while (l < r && cover(P[l], P[r - 1], P[r])) --r;
25 while (l < r && cover(P[r], P[l], P[l + 1])) ++l;
26 if (r - l <= 1 || !argcmp(P[l].dir(), P[r].dir()))
27     return {};
28 if (cover(P[l + 1], P[l], P[r])) return {};
29 return vector<Line>(P.begin() + l, P.begin() + r +
30     1);

```

7.15 Minimum Enclosing Circle

```

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

```

7.16 Union of Circles

```

1 // Area[i] : area covered by at least i circle
2 vector<T> CircleUnion(const vector<Cir> &C) {
3     const int n = C.size();
4     vector<T> Area(n + 1);
5     auto check = [&](int i, int j) {
6         if (!contain(C[i], C[j]))
7             return false;
8         return sgn(C[i].r - C[j].r) > 0 or (sgn(C[i].r
9             - C[j].r) == 0 and i < j);
10    };
11    struct Teve {
12        double ang; int add; Pt p;
13        bool operator<(const Teve &b) { return ang < b.
14            ang; }
15    };

```

```

14 auto ang = [&](Pt p) { return atan2(p.y, p.x); };
15 for (int i = 0; i < n; i++) {
16     int cov = 1;
17     vector<Teve> event;
18     for (int j = 0; j < n; j++) if (i != j) {
19         if (check(j, i)) cov++;
20         else if (!check(i, j) and !disjunct(C[i], C[j])) {
21             auto I = CircleInter(C[i], C[j]);
22             assert(I.size() == 2);
23             double a1 = ang(I[0] - C[i].o), a2 =
24                 ang(I[1] - C[i].o);
25             event.push_back({a1, 1, I[0]});
26             event.push_back({a2, -1, I[1]});
27             if (a1 > a2) cov++;
28         }
29     }
30     if (event.empty()) {
31         Area[cov] += acos(-1) * C[i].r * C[i].r;
32         continue;
33     }
34     sort(event.begin(), event.end());
35     event.push_back(event[0]);
36     for (int j = 0; j + 1 < event.size(); j++) {
37         cov += event[j].add;
38         Area[cov] += (event[j].p ^ event[j + 1].p)
39             / 2.;
40         double theta = event[j + 1].ang - event[j].ang;
41         if (theta < 0) theta += 2 * acos(-1);
42         Area[cov] += (theta - sin(theta)) * C[i].r
43             * C[i].r / 2.;
44     }
45 }
46 return Area;
47 }

```

7.17 Area Of Circle Polygon

```

1 double AreaOfCirclePoly(Cir C, vector<Pt> &P) {
2     auto arg = [&](Pt p, Pt q) { return atan2(p ^ q, p
3         * q); };
4     double r2 = (double)(C.r * C.r / 2);
5     auto tri = [&](Pt p, Pt q) {
6         Pt d = q - p;
7         T a = (d * p) / (d * d);
8         T b = ((p * p) - C.r * C.r) / (d * d);
9         T det = a * a - b;
10        if (det <= 0) return (double)(arg(p, q) * r2);
11        T s = max((T)0.0L, -a - sqrtl(det));
12        T t = min((T)1.0L, -a + sqrtl(det));
13        if (t < 0 || 1 <= s) return (double)(arg(p, q)
14            * r2);
15        Pt u = p + d * s, v = p + d * t;
16        return (double)(arg(p, u) * r2 + (u ^ v) / 2 +
17            arg(v, q) * r2);
18    };
19    long double sum = 0.0L;
20    for (int i = 0; i < (int)P.size(); i++)
21        sum += tri(P[i] - C.o, P[(i + 1) % P.size()] -
22            C.o);
23    return (double)fabs1(sum);
24 }

```

7.18 3D Point

```

1 struct Pt {
2     double x, y, z;
3     Pt(double _x = 0, double _y = 0, double _z = 0): x(_x
4         ), y(_y), z(_z){}
5     Pt operator + (const Pt &o) const
6     { return Pt(x + o.x, y + o.y, z + o.z); }
7     Pt operator - (const Pt &o) const
8     { return Pt(x - o.x, y - o.y, z - o.z); }
9     Pt operator * (const double &k) const
10    { return Pt(x * k, y * k, z * k); }
11    Pt operator / (const double &k) const
12    { return Pt(x / k, y / k, z / k); }
13    double operator * (const Pt &o) const
14    { return x * o.x + y * o.y + z * o.z; }
15    Pt operator ^ (const Pt &o) const

```

```

16    { return {Pt(y * o.z - z * o.y, z * o.x - x * o.z, x
17        * o.y - y * o.x)}; }
18 };
19 double abs2(Pt o) { return o * o; }
20 double abs(Pt o) { return sqrt(abs2(o)); }
21 Pt cross3(Pt a, Pt b, Pt c)
22 { return (b - a) ^ (c - a); }
23 double area(Pt a, Pt b, Pt c)
24 { return abs(cross3(a, b, c)); }
25 double volume(Pt a, Pt b, Pt c, Pt d)
26 { return cross3(a, b, c) * (d - a); }
27 bool coplaner(Pt a, Pt b, Pt c, Pt d)
28 { return sign(volume(a, b, c, d)) == 0; }
29 Pt proj(Pt o, Pt a, Pt b, Pt c) // o proj to plane abc
30 { Pt n = cross3(a, b, c);
31     return o - n * ((o - a) * (n / abs2(n))); }
32 Pt line_plane_intersect(Pt u, Pt v, Pt a, Pt b, Pt c) {
33     // intersection of line uv and plane abc
34     Pt n = cross3(a, b, c);
35     double s = n * (u - v);
36     if (sign(s) == 0) return {-1, -1, -1}; // not found
37     return v + (u - v) * ((n * (a - v)) / s); }

```

8 Number Theory

8.1 FFT

```

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

```

```

43         cp tmp = xomega[r * i] * a[j + mid
44             + i];
45         a[j + mid + i] = a[j + i] - tmp;
46         a[j + i] = a[j + i] + tmp;
47     }
48 }
49
50 void fft(vector<cp> &a) { transform(a, omega); }
51 void ifft(vector<cp> &a) {
52     transform(a, iomega);
53     for (int i = 0; i < n; i++) a[i] /= n;
54 }
55 } FFT;
56
57 const int MAXN = 262144;
58 // (must be 2^k)
59 // 262144, 524288, 1048576, 2097152, 4194304
60 // before any usage, run pre_fft() first
61 typedef long double ld;
62 typedef complex<ld> cplx; // real(), imag()
63 const ld PI = acos(-1);
64 const cplx I(0, 1);
65 cplx omega[MAXN + 1];
66 void pre_fft() {
67     for (int i = 0; i <= MAXN; i++) {
68         omega[i] = exp(i * 2 * PI / MAXN * I);
69     }
70 }
71 // n must be 2^k
72 void fft(int n, cplx a[], bool inv = false) {
73     int basic = MAXN / n;
74     int theta = basic;
75     for (int m = n; m >= 2; m >>= 1) {
76         int mh = m >> 1;
77         for (int i = 0; i < mh; i++) {
78             cplx w = omega[inv ? MAXN - (i * theta %
79                 MAXN) : i * theta % MAXN];
80             for (int j = i; j < n; j += m) {
81                 int k = j + mh;
82                 cplx x = a[j] - a[k];
83                 a[j] += a[k];
84                 a[k] = w * x;
85             }
86             theta = (theta * 2) % MAXN;
87         }
88         int i = 0;
89         for (int j = 1; j < n - 1; j++) {
90             for (int k = n >> 1; k > (i ^= k); k >>= 1);
91             if (j < i) swap(a[i], a[j]);
92         }
93         if (inv) {
94             for (i = 0; i < n; i++) a[i] /= n;
95         }
96     }
97     cplx arr[MAXN + 1];
98     inline void mul(int _n, long long a[], int _m, long
99         long b[], long long ans[]) {
100         int n = 1, sum = _n + _m - 1;
101         while (n < sum) n <<= 1;
102         for (int i = 0; i < n; i++) {
103             double x = (i < _n ? a[i] : 0), y = (i < _m ? b
104                 [i] : 0);
105             arr[i] = complex<double>(x + y, x - y);
106         }
107         fft(n, arr);
108         for (int i = 0; i < n; i++) arr[i] = arr[i] * arr[i
109             ];
110         fft(n, arr, true);
111         for (int i = 0; i < sum; i++) ans[i] = (long long
112             int)(arr[i].real() / 4 + 0.5);
113     }
114 }
115
116 long long a[MAXN];
117 long long b[MAXN];
118 long long ans[MAXN];
119 int a_length;
120 int b_length;

```

8.2 Pollard's rho

```

1 ll add(ll x, ll y, ll p) {
2     return (x + y) % p;
3 }
4 ll qMul(ll x, ll y, ll mod) {
5     ll ret = x * y - (ll)((long double)x / mod * y) *
6         mod;
7     return ret < 0 ? ret + mod : ret;
8 }
9 ll f(ll x, ll mod) { return add(qMul(x, x, mod), 1, mod
10     ); }
11 ll pollard_rho(ll n) {
12     if (!(n & 1)) return 2;
13     while (true) {
14         ll y = 2, x = rand() % (n - 1) + 1, res = 1;
15         for (int sz = 2; res == 1; sz *= 2) {
16             for (int i = 0; i < sz && res <= 1; i++) {
17                 x = f(x, n);
18                 res = __gcd(llabs(x - y), n);
19             }
20             y = x;
21         }
22         if (res != 0 && res != n) return res;
23     }
24 }
25 vector<ll> ret;
26 void fact(ll x) {
27     if (miller_rabin(x)) {
28         ret.push_back(x);
29         return;
30     }
31     ll f = pollard_rho(x);
32     fact(f);
33     fact(x / f);
34 }

```

8.3 Miller Rabin

```

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

```

8.4 Fast Power

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

8.5 Extend GCD

```

1 ll GCD;
2 pll extgcd(ll a, ll b) {
3     if (b == 0) {
4         GCD = a;
5         return pll{1, 0};
6     }
7     pll ans = extgcd(b, a % b);
8     return pll{ans.S, ans.F - a / b * ans.S};
9 }

```



```

10 pll bezout(ll a, ll b, ll c) {
11     bool negx = (a < 0), negy = (b < 0);
12     pll ans = extgcd(abs(a), abs(b));
13     if (c % GCD != 0) return pll{-LLINF, -LLINF};
14     return pll{ans.F * c / GCD * (negx ? -1 : 1),
15               ans.S * c / GCD * (negy ? -1 : 1)};
16 }
17 ll inv(ll a, ll p) {
18     if (p == 1) return -1;
19     pll ans = bezout(a % p, -p, 1);
20     if (ans == pll{-LLINF, -LLINF}) return -1;
21     return (ans.F % p + p) % p;
22 }

```

8.6 Mu + Phi

```

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

```

8.7 Discrete Log

```

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

```

```

35     gamma = (__int128)gamma * am_inv % p;
36 }
37 return -1;
38 }

```

8.8 sqrt mod

```

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

```

8.9 Primitive Root

```

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

```

8.10 Other Formulas

- Inversion:
 $aa^{-1} \equiv 1 \pmod{m}$. a^{-1} exists iff $\gcd(a, m) = 1$.
- Linear inversion:
 $a^{-1} \equiv (m - \lfloor \frac{m}{a} \rfloor) \times (m \bmod a)^{-1} \pmod{m}$

- Fermat's little theorem:
 $a^p \equiv a \pmod{p}$ if p is prime.
- Euler function:
 $\phi(n) = n \prod_{p|n} \frac{p-1}{p}$
- Euler theorem:
 $a^{\phi(n)} \equiv 1 \pmod{n}$ if $\gcd(a, n) = 1$.
- Extended Euclidean algorithm:
 $ax + by = \gcd(a, b) = \gcd(b, a \bmod b) = \gcd(b, a - \lfloor \frac{a}{b} \rfloor b) = bx_1 + (a - \lfloor \frac{a}{b} \rfloor b)y_1 = ay_1 + b(x_1 - \lfloor \frac{a}{b} \rfloor y_1)$
- Divisor function:
 $\sigma_x(n) = \sum_{d|n} d^x \cdot n = \prod_{i=1}^r p_i^{a_i}$
 $\sigma_x(n) = \prod_{i=1}^r \frac{p_i^{(a_i+1)x} - 1}{p_i^x - 1}$ 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: $(f * g)(n) = \sum_{d|n} f(n)g(n/d)$
- Important Multiplicative Functions + Properties:
 1. $\epsilon(n) = [n = 1]$
 2. $1(n) = 1$
 3. $id(n) = n$
 4. $\mu(n) = 0$ if n has squared prime factor
 5. $\mu(n) = (-1)^k$ if $n = p_1 p_2 \cdots p_k$
 6. $\epsilon = \mu * 1$
 7. $\phi = \mu * id$
 8. $[n = 1] = \sum_{d|n} \mu(d)$
 9. $[gcd = 1] = \sum_{d|gcd} \mu(d)$
- Möbius inversion: $f = g * 1 \Leftrightarrow g = f * \mu$

8.11 Polynomial

```

1 const int maxk = 20;
2 const int maxn = 1<<maxk;
3 const ll LINF = 1e18;
4
5 /* P = r*2^k + 1
6 P          r    k    g
7 998244353  119  23    3
8 1004535809 479  21    3
9
10 P          r    k    g
11 3          1    1    2
12 5          1    2    2
13 17         1    4    3
14 97         3    5    5
15 193        3    6    5
16 257        1    8    3
17 7681       15    9   17
18 12289      3   12   11
19 40961      5   13    3
20 65537      1   16    3
21 786433     3   18   10
22 5767169    11  19    3
23 7340033    7   20    3
24 23068673   11  21    3
25 104857601  25  22    3
26 167772161  5   25    3

```

```

27 469762049   7   26    3
28 1004535809 479  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++) {

```

```

107     if (!(i & (i-1))) hb++;
108     rev[i] = rev[i ^ (1<<hb)] | (1<<(maxk-hb-1));
109 } }
110
111 template<typename T>
112 void NTT(vector<T>& a, bool inv=false) {
113
114     int _n = (int)a.size();
115     int k = __lg(_n) + ((1<<__lg(_n)) != _n);
116     int n = 1<<k;
117     a.resize(n, 0);
118
119     short shift = maxk-k;
120     for (int i = 0; i < n; i++)
121         if (i > (rev[i]>>shift))
122             swap(a[i], a[rev[i]>>shift]);
123
124     for (int len = 2, half = 1, div = maxn>>1; len <= n; len<=1, half<=1, div>=1) {
125         for (int i = 0; i < n; i += len) {
126             for (int j = 0; j < half; j++) {
127                 T u = a[i+j];
128                 T v = a[i+j+half] * (inv ? ix[j*div] :
129                     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         if (inv) {
136             T dn = pw(n, MOD-2);
137             for (auto& x : a) {
138                 x *= dn;
139                 if (x >= MOD) x %= MOD;
140             }
141         }
142     }
143
144     template<typename T>
145     inline void resize(vector<T>& a) {
146         int cnt = (int)a.size();
147         for (; cnt > 0; cnt--) if (a[cnt-1]) break;
148         a.resize(max(cnt, 1));
149     }
150
151     template<typename T>
152     vector<T>& operator*=(vector<T>& a, vector<T> b) {
153         int na = (int)a.size();
154         int nb = (int)b.size();
155         a.resize(na + nb - 1, 0);
156         b.resize(na + nb - 1, 0);
157
158         NTT(a); NTT(b);
159         for (int i = 0; i < (int)a.size(); i++) {
160             a[i] *= b[i];
161             if (a[i] >= MOD) a[i] %= MOD;
162         }
163         NTT(a, true);
164
165         resize(a);
166         return a;
167     }
168
169     template<typename T>
170     void inv(vector<T>& ia, int N) {
171         vector<T> _a(move(ia));
172         ia.resize(1, pw(_a[0], MOD-2));
173         vector<T> a(1, -_a[0] + (-_a[0] < 0 ? MOD : 0));
174
175         for (int n = 1; n < N; n<=1) {
176             // n -> 2*n
177             // ia' = ia(2-a*ia);
178
179             for (int i = n; i < min(siz(_a), (n<<1)); i++)
180                 a.emplace_back(-_a[i] + (-_a[i] < 0 ? MOD : 0));
181
182             vector<T> tmp = ia;
183             ia *= a;
184             ia.resize(n<<1);
185             ia[0] = ia[0] + 2 >= MOD ? ia[0] + 2 - MOD : ia[0] + 2;
186             ia *= tmp;
187             ia.resize(n<<1);
188         }
189     }
190
191     template<typename T>
192     void mod(vector<T>& a, vector<T>& b) {
193         int n = (int)a.size()-1, m = (int)b.size()-1;
194         if (n < m) return;
195
196         vector<T> ra = a, rb = b;
197         reverse(ra.begin(), ra.end()); ra.resize(min(n+1, n-m+1));
198         reverse(rb.begin(), rb.end()); rb.resize(min(m+1, n-m+1));
199
200         inv(rb, n-m+1);
201
202         vector<T> q = move(ra);
203         q *= rb;
204         q.resize(n-m+1);
205         reverse(q.begin(), q.end());
206
207         q *= b;
208         a -= q;
209         resize(a);
210
211         /* Kitamasa Method (Fast Linear Recurrence):
212         Find a[K] (Given a[j] = c[0]a[j-N] + ... + c[N-1]a[j-1])
213         Let B(x) = x^N - c[N-1]x^(N-1) - ... - c[1]x^1 - c[0]
214         Let R(x) = x^K mod B(x) (get x^K using fast pow and use poly mod to get R(x))
215         Let r[i] = the coefficient of x^i in R(x)
216         => a[K] = a[0]r[0] + a[1]r[1] + ... + a[N-1]r[N-1] */
217     }
218
219     9 Linear Algebra
220     9.1 Gaussian-Jordan Elimination
221
222     1 int n;
223     2 vector<vector<ll>> v;
224     3 void gauss(vector<vector<ll>>& v) {
225         4 int r = 0;
226         5 for (int i = 0; i < n; i++) {
227             6 bool ok = false;
228             7 for (int j = r; j < n; j++) {
229                 8 if (v[j][i] == 0) continue;
230                 9 swap(v[j], v[r]);
231                 10 ok = true;
232                 11 break;
233             }
234             12 if (!ok) continue;
235             13 ll div = inv(v[r][i]);
236             14 for (int j = 0; j < n + 1; j++) {
237                 15 v[r][j] *= div;
238                 16 if (v[r][j] >= MOD) v[r][j] %= MOD;
239             }
240             17 for (int j = 0; j < n; j++) {
241                 18 if (j == r) continue;
242                 19 ll t = v[j][i];
243                 20 for (int k = 0; k < n + 1; k++) {
244                     21 v[j][k] -= v[r][k] * t % MOD;
245                     22 if (v[j][k] < 0) v[j][k] += MOD;
246                 }
247             }
248             23 r++;
249         }
250     }
251
252     9.2 Determinant
253
254     1. Use GJ Elimination, if there's any row consists of only 0, then det = 0, otherwise det = product of diagonal elements.
255
256     2. Properties of det:
257
258         • Transpose: Unchanged
259         • 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 $\leq n \approx n/((\ln n) - 1)$

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

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

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

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

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

