

Contents

1 Reminder	1	6 String	12
1.1 Bug List	1	6.1 Aho Corasick	12
1.2 OwO	1	6.2 KMP	12
2 Basic	1	6.3 Z Value	12
2.1 Vimrc	1	6.4 Manacher	13
2.2 Stress	1	6.5 Suffix Array	13
2.3 Run Sample	1	6.6 Suffix Automaton	13
2.4 Runcpp.sh	1	6.7 Minimum Rotation	14
2.5 Others	1	6.8 Lyndon Factorization	14
3 Data Structure	2	6.9 Rolling Hash	14
3.1 BIT	2	6.10 Trie	14
3.2 Lazy Propagation Segment Tree	2	7 Geometry	14
3.3 Treap	2	7.1 Basic Operations	14
3.4 Li Chao Tree	3	7.2 Sort by Angle	15
3.5 LineContainer	3	7.3 Intersection	15
3.6 Sparse Table	3	7.4 Polygon Area	15
3.7 Dynamic Median	4	7.5 Convex Hull	15
3.8 SOS DP	4	7.6 Point In Convex	16
4 Flow / Matching	4	7.7 Point Segment Distance	16
4.1 Dinic	4	7.8 Point in Polygon	16
4.2 MCMF	4	7.9 Minimum Euclidean Distance	16
4.3 KM	5	7.10 Minkowski Sum	16
4.4 Hopcroft-Karp	5	7.11 Lower Concave Hull	16
4.5 Blossom	6	7.12 Pick's Theorem	16
4.6 Cover / Independent Set	6	7.13 Rotating SweepLine	17
4.7 Hungarian Algorithm	6	7.14 Half Plane Intersection	17
5 Graph	6	7.15 Minimum Enclosing Circle	17
5.1 Heavy-Light Decomposition	6	7.16 Union of Circles	17
5.2 Centroid Decomposition	7	7.17 Area Of Circle Polygon	17
5.3 Bellman-Ford + SPFA	7	7.18 3D Point	18
5.4 BCC - AP	8	8 Number Theory	18
5.5 BCC - Bridge	9	8.1 FFT	18
5.6 SCC - Tarjan	9	8.2 Pollard's rho	19
5.7 SCC - Kosaraju	10	8.3 Miller Rabin	19
5.8 Eulerian Path - Undir	10	8.4 Fast Power	19
5.9 Eulerian Path - Dir	10	8.5 Extend GCD	19
5.10 Hamilton Path	10	8.6 Mu + Phi	19
5.11 Kth Shortest Path	11	8.7 Discrete Log	20
5.12 System of Difference Constraints	12	8.8 sqrt mod	20

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 Stress

```

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

```

1 Reminder

1.1 Bug List

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

1.2 OwO

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

2.3 Run Sample

```

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

```

2.4 Runcpp.sh

```

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

```

```

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

```

2.5 Others

```

1 #pragma GCC optimize("Ofast,unroll-loops,no-stack-
2   protector,fast-math")
3 #pragma GCC target("see,see2,see3,see4,avx2,bmi,bmi2,
4   lzcnt,popcnt,tune=native")
5 #pragma GCC optimize("trapv")
6 mt19937 gen(chrono::steady_clock::now().
7   time_since_epoch().count());
8 uniform_int_distribution<int> dis(1, 100);
9 cout << dis(gen) << endl;
10 shuffle(v.begin(), v.end(), gen);

11 struct edge {
12   int a, b, w;
13   friend istream& operator>>(istream& in, edge& x) {
14     in >> x.a >> x.b >> x.w; }
15   friend ostream& operator<<(ostream& out, const edge& x) {
16     out << "(" << x.a << "," << x.b << "," << x.w
17       << ")";
18     return out;
19   }
20 };
21 struct cmp {
22   bool operator()(const edge& x, const edge& y) const
23   { return x.w < y.w; }
24 };
25 set<edge, cmp> st; // 遞增
26 map<edge, long long, cmp> mp; // 遞增
27 priority_queue<edge, vector<edge>, cmp> pq; // 遞減

28 #include <bits/extc++.h>
29 #include <ext/pb_ds/assoc_container.hpp>
30 #include <ext/pb_ds/tree_policy.hpp>
31 using namespace __gnu_pbds;

32 // map
33 tree<int, int, less<>, rb_tree_tag,
34   tree_order_statistics_node_update> tr;
35 tr.order_of_key(element);
36 tr.find_by_order(rank);

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

42 gp_hash_table<int, int> ht;
43 ht.find(element);
44 ht.insert({key, value});
45 ht.erase(element);

46 // priority queue Big First
47 __gnu_pbds::priority_queue<int, less<int>> big_q;
48 __gnu_pbds::priority_queue<int, greater<int>> small_q;
49   // Small First
50 q1.join(q2); // join

```

3 Data Structure

3.1 BIT

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

```

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

```

3.2 Lazy Propagation Segment Tree

```

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

3.3 Treap

```

1 mt19937 rng(random_device{}());
2 struct Treap {
3     Treap *l, *r;
4     int val, sum, real, tag, num, pri, rev;
5     Treap(int k) {
6         l = r = NULL;
7         val = sum = k;
8         num = 1;
9         real = -1;
10        tag = 0;
11        rev = 0;
12        pri = rng();
13    }
14 };
15 int siz(Treap *now) { return now ? now->num : 0ll; }
16 int sum(Treap *now) {
17     if (!now) return 0;
18     if (now->real != -1) return (now->real + now->tag)
19         * now->num;
20     return now->sum + now->tag * now->num;
21 }
22 void pull(Treap *&now) {
23     now->num = siz(now->l) + siz(now->r) + 1ll;
24     now->sum = sum(now->l) + sum(now->r) + now->val +
25         now->tag;
26 }
27 void push(Treap *&now) {
28     if (now->rev) {
29         swap(now->l, now->r);
30         now->l->rev ^= 1;
31         now->r->rev ^= 1;
32         now->rev = 0;
33     }
34     if (now->real != -1) {
35         now->real += now->tag;
36         if (now->l) {
37             now->l->tag = 0;
38             now->l->real = now->real;
39             now->l->val = now->real;
40         }
41         if (now->r) {
42             now->r->tag = 0;
43             now->r->real = now->real;
44             now->r->val = now->real;
45         }
46         now->val = now->real;
47         now->sum = now->real * now->num;
48         now->real = -1;
49         now->tag = 0;
50     } else {
51         if (now->l) now->l->tag += now->tag;
52         if (now->r) now->r->tag += now->tag;
53         now->sum += sum(now);
54         now->val += now->tag;
55         now->tag = 0;
56     }
57     Treap *merge(Treap *a, Treap *b) {
58         if (!a || !b) return a ? a : b;
59         else if (a->pri > b->pri) {
60             push(a);
61             a->r = merge(a->r, b);
62             pull(a);
63             return a;
64         } else {
65             push(b);
66             b->l = merge(a, b->l);
67             pull(b);
68             return b;
69         }
70     }
71     void split_size(Treap *rt, Treap *&a, Treap *&b, int
72         val) {
73         if (!rt) {
74             a = b = NULL;
75             return;
76         }
77         push(rt);
78         if (siz(rt->l) + 1 > val) {
79             split_size(rt->l, a, b->l, val);
80             pull(b);
81         } else {
82             a = rt;
83             split_size(rt->r, a->r, b, val - siz(a->l) - 1)
84                 ;
85             pull(a);
86         }
87     }
88     void split_val(Treap *rt, Treap *&a, Treap *&b, int val
89 ) {
90         if (!rt) {
91             a = b = NULL;
92             return;
93         }
94         push(rt);
95         if (rt->val <= val) {
96             a = rt;
97             split_val(rt->r, a->r, b, val);
98             pull(a);
99         } else {
100            b = rt;
101            split_val(rt->l, a, b->l, val);
102            pull(b);
103        }
104    }
105 }
```

3.4 Li Chao Tree

```

1 const int eps=1e-9;
2 struct line
3 {
4     double m,k;
5 }p[N];
6 int seg[N<<2],cnt;
7 double cal(int id,int x){return 1.0*p[id].m*x+p[id].k;}
8 void add(int x0,int y0,int x1,int y1)
9 {
10     cnt++;
11     if(x0==x1)p[cnt].m=0,p[cnt].k=max(y0,y1);
12     else p[cnt].m=1.0*(y1-y0)/(x1-x0),p[cnt].k=y0-p[
13         cnt].m*x0;
14 }
15 void update(int x,int l,int r,int ql,int qr,int u)
16 {
17     int v=seg[x],mid=(l+r)>>1;
18     double resu=cal(u,mid),resv=cal(v,mid);
19     if(qr<1||r<ql) return;
20     if(ql<=l&&r<=qr)
21     {
22         if(l==r)
23         {
24             if(resu>resv)seg[x]=u;
25             return;
26         }
27         if(p[u].m-p[v].m>eps)
28         {
29             if(resu-resv>eps)
30             {
31                 seg[x]=u;
32                 update(x<<1,l,mid,ql,qr,v);
33             }
34             else update(x<<1|1,mid+1,r,ql,qr,u);
35         }
36         else if(p[v].m-p[u].m>eps)
37         {
38             if(resu-resv>eps)
39             {
40                 seg[x]=u;
41                 update(x<<1|1,mid+1,r,ql,qr,v);
42             }
43             else update(x<<1,l,mid,ql,qr,u);
44         }
45         else if(p[u].k-p[v].k>eps)seg[x]=u;
46         return;
47     }
48     update(x<<1,l,mid,ql,qr,u);
49     update(x<<1|1,mid+1,r,ql,qr,u);
50 }
51 double ask(int x,int l,int r,int qx)
52 {
53     if(r<qx||qx<l) return{0,0};
54 }
```

```

53     int mid=(l+r)>>1;
54     double res=cal(seg[x],qx);
55     if(l==r) return res;
56     return max({res,seg[x],ask(x<<1,l,mid,qx),ask(x
57         <<1|1,mid+1,r,qx)});
```

3.5 LineContainer

```

1 struct Line {
2     mutable ll k, m, p;
3     bool operator<(const Line& o) const { return k < o.
4         k; }
5     bool operator<(ll x) const { return p < x; }
6 };
7
8 struct LineContainer : multiset<Line, less<>> {
9     // (for doubles, use inf = 1/.0, div(a,b) = a/b)
10    static const ll inf = LLONG_MAX;
11    ll div(ll a, ll b) { // floored division
12        return a / b - ((a ^ b) < 0 && a % b); }
13    bool isect(iterator x, iterator y) {
14        if (y == end()) return x->p = inf, 0;
15        if (x->k == y->k) x->p = x->m > y->m ? inf : -
16            inf;
17        else x->p = div(y->m - x->m, x->k - y->k);
18        return x->p >= y->p;
19    }
20    void add(ll k, ll m) {
21        auto z = insert({k, m, 0}), y = z++, x = y;
22        while (isect(y, z)) z = erase(z);
23        if (x != begin() && isect(--x, y)) isect(x, y =
24            erase(y));
25        while ((y = x) != begin() && (--x)->p >= y->p)
26            isect(x, erase(y));
27    }
28    ll query(ll x) {
29        assert(!empty());
30        auto l = *lower_bound(x);
31        return l.k * x + l.m;
32    }
33 }
```

3.6 Sparse Table

```

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

3.7 Dynamic Median

```

1 struct Dynamic_Median {
2     multiset<long long> lo, hi;
3     long long slo = 0, shi = 0;
4     void rebalance() {
5         // keep sz(lo) >= sz(hi) and sz(lo) - sz(hi) <= 1
6         while((int)lo.size() > (int)hi.size() + 1) {
7             auto it = prev(lo.end());
8             long long x = *it;
9             lo.erase(it); slo -= x;
10            hi.insert(x); shi += x;
11        }
12        while((int)lo.size() < (int)hi.size()) {
13            auto it = hi.begin();
14            long long x = *it;
15            hi.erase(it); shi -= x;
16            lo.insert(x); slo += x;
17        }
18    }
19 }
```

```

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

3.8 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;
5         }
6     }
7 }
```

4 Flow / Matching

4.1 Dinic

```

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

```

32         }
33     }
34     return 0;
35 }
36 int dinic() {
37     int res = 0;
38     while (true) {
39         bfs();
40         if (level[t] == -1) break;
41         memset(iter, 0, sizeof(iter));
42         int now = 0;
43         while ((now = dfs(s, INF)) > 0) res += now;
44     }
45     return res;
46 }
47 }
```

4.2 MCMF

```

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

4.3 KM

```

1 struct KM {
2     int n, mx[1005], my[1005], pa[1005];
3     int g[1005][1005], lx[1005], ly[1005], sy[1005];
4     bool vx[1005], vy[1005];
5     void init(int _n) {
6         n = _n;
7         FOR(i, 1, n + 1)
8             fill(g[i], g[i] + 1 + n, 0);
9     }
10    void add(int a, int b, int c) { g[a][b] = c; }
11    void augment(int y) {
12        for (int x, z; y; y = z)
13            x = pa[y], z = mx[x], my[y] = x, mx[x] = y;
14    }
15    void bfs(int st) {
16        FOR(i, 1, n + 1)
17            sy[i] = INF,
18            vx[i] = vy[i] = 0;
19        queue<int> q;
20        q.push(st);
21        for (;;) {
22            while (!q.empty()) {
23                int x = q.front();
24                q.pop();
25                vx[x] = 1;
26                FOR(y, 1, n + 1)
27                    if (!vy[y]) {
28                        int t = lx[x] + ly[y] - g[x][y];
29                        if (t == 0) {
30                            pa[y] = x;
31                            if (!my[y])
32                                augment(y);
33                            return;
34                        }
35                        vy[y] = 1, q.push(my[y]);
36                    } else if (sy[y] > t)
37                        pa[y] = x, sy[y] = t;
38            }
39            int cut = INF;
40            FOR(y, 1, n + 1)
41                if (!vy[y] && cut > sy[y]) cut = sy[y];
42            FOR(j, 1, n + 1) {
43                if (vx[j]) lx[j] -= cut;
44                if (vy[j])
45                    ly[j] += cut;
46                else
47                    sy[j] -= cut;
48            }
49            FOR(y, 1, n + 1) {
50                if (!vy[y] && sy[y] == 0) {
51                    if (!my[y])
52                        augment(y);
53                    return;
54                }
55                vy[y] = 1;
56                q.push(my[y]);
57            }
58        }
59    }
60    int solve() {
61        fill(mx, mx + n + 1, 0);
62        fill(my, my + n + 1, 0);
63        fill(ly, ly + n + 1, 0);
64        fill(lx, lx + n + 1, 0);
65        FOR(x, 1, n + 1)
66            FOR(y, 1, n + 1)
67                lx[x] = max(lx[x], g[x][y]);
68        FOR(x, 1, n + 1)
69            bfs(x);
70        int ans = 0;
71        FOR(y, 1, n + 1)
72            ans += g[my[y]][y];
73        return ans;
74    }
75 }
```

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 Cover / Independent Set

```

1 V(E) Cover: choose some V(E) to cover all E(V)
2 V(E) Independ: set of V(E) not adj to each other
3
4 M = Max Matching
5 Cv = Min V Cover
6 Ce = Min E Cover
7 Iv = Max V Ind
8 Ie = Max E Ind (equiv to M)
9
10 M = Cv (Konig Theorem)
11 Iv = V \ Cv
12 Ce = V - M
13
14 Construct Cv:
15 1. Run Dinic
16 2. Find s-t min cut
17 3. Cv = {X in T} + {Y in S}

```

4.7 Hungarian Algorithm

```

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

```

```

19    match_cnt = 0;
20    for(int i = 1; i <= n; i++) {
21        memset(vis, 0, sizeof(vis));
22        if(dfs(i)) match_cnt++;
23    }

```

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    ;
17    seg[x].sum = seg[x << 1].sum + seg[x << 1 | 1].sum;
18 }
19 int big(int x, int l, int r, int ql, int qr) {
20     if (ql <= l && r <= qr) return seg[x].mx;
21     int mid = (l + r) >> 1;
22     int res = -INF;
23     if (ql <= mid) res = max(res, big(x << 1, l, mid,
24         ql, qr));
25     if (mid < qr) res = max(res, big(x << 1 | 1, mid +
26         1, r, ql, qr));
27     return res;
28 }
29 int ask(int x, int l, int r, int ql, int qr) {
30     if (ql <= l && r <= qr) return seg[x].sum;
31     int mid = (l + r) >> 1;
32     int res = 0;
33     if (ql <= mid) res += ask(x << 1, l, mid, ql, qr);
34     if (mid < qr) res += ask(x << 1 | 1, mid + 1, r, ql
35         , qr);
36     return res;
37 }
38 void dfs1(int now) {
39     son[now] = -1;
40     num[now] = 1;
41     for (auto i : path[now]) {
42         if (!dep[i]) {
43             dep[i] = dep[now] + 1;
44             p[i] = now;
45             dfs1(i);
46             num[now] += num[i];
47             if (son[now] == -1 || num[i] > num[son[now]
48                 ]]) son[now] = i;
49         }
50     }
51 }
52 int cnt;
53 void dfs2(int now, int t) {
54     top[now] = t;
55     cnt++;
56     dfn[now] = cnt;
57     if (son[now] == -1) return;
58     dfs2(son[now], t);
59     for (auto i : path[now])
60         if (i != p[now] && i != son[now]) dfs2(i, i);
61 }
62 int path_big(int x, int y) {
63     int res = -INF;
64     while (top[x] != top[y]) {
65         if (dep[top[x]] < dep[top[y]]) swap(x, y);
66         res = max(res, big(1, 1, n, dfn[top[x]], dfn[x
67             ]));
68         x = p[top[x]];
69     }
70     if (dfn[x] > dfn[y]) swap(x, y);

```

```

65     res = max(res, big(1, 1, n, dfn[x], dfn[y]));
66     return res;
67 }
68 int path_sum(int x, int y) {
69     int res = 0;
70     while (top[x] != top[y]) {
71         if (dep[top[x]] < dep[top[y]]) swap(x, y);
72         res += ask(1, 1, n, dfn[top[x]], dfn[x]);
73         x = p[top[x]];
74     }
75     if (dfn[x] > dfn[y]) swap(x, y);
76     res += ask(1, 1, n, dfn[x], dfn[y]);
77     return res;
78 }
79 void buildTree() {
80     FOR(i, 0, n - 1) {
81         int a, b;
82         cin >> a >> b;
83         path[a].pb(b);
84         path[b].pb(a);
85     }
86 }
87 void buildHLD(int root) {
88     dep[root] = 1;
89     dfs1(root);
90     dfs2(root, root);
91     FOR(i, 1, n + 1) {
92         int now;
93         cin >> now;
94         update(1, 1, n, dfn[i], now);
95     }
96 }

```

5.2 Centroid Decomposition

```

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

```

5.3 Bellman-Ford + SPFA

```

1 int n, m;
2
3 // Graph
4 vector<vector<pair<int, ll>> > g;
5 vector<ll> dis;
6 vector<bool> negCycle;
7
8 // SPFA
9 vector<int> rlx;
10 queue<int> q;
11 vector<bool> inq;
12 vector<int> pa;
13 void SPFA(vector<int>& src) {
14     dis.assign(n + 1, LINF);
15     negCycle.assign(n + 1, false);
16     rlx.assign(n + 1, 0);
17     while (!q.empty()) q.pop();
18     inq.assign(n + 1, false);
19     pa.assign(n + 1, -1);
20
21     for (auto& s : src) {
22         dis[s] = 0;
23         q.push(s);
24         inq[s] = true;
25     }
26
27     while (!q.empty()) {
28         int u = q.front();
29         q.pop();
30         inq[u] = false;
31         if (rlx[u] >= n) {
32             negCycle[u] = true;
33         } else {
34             for (auto& e : g[u]) {
35                 int v = e.first;
36                 ll w = e.second;
37                 if (dis[v] > dis[u] + w) {
38                     dis[v] = dis[u] + w;
39                     rlx[v] = rlx[u] + 1;
40                     pa[v] = u;
41                     if (!inq[v]) {
42                         q.push(v);
43                         inq[v] = true;
44                     }
45                 }
46             }
47         }
48     }
49
50 // Bellman-Ford
51 queue<int> q;
52 vector<int> pa;
53 void BellmanFord(vector<int>& src) {
54     dis.assign(n + 1, LINF);
55     negCycle.assign(n + 1, false);
56     pa.assign(n + 1, -1);
57
58     for (auto& s : src) dis[s] = 0;
59
60     for (int rlx = 1; rlx <= n; rlx++) {
61         for (int u = 1; u <= n; u++) {
62             if (dis[u] == LINF) continue; // Important
63             !
64             for (auto& e : g[u]) {
65                 int v = e.first;
66                 ll w = e.second;
67                 if (dis[v] > dis[u] + w) {
68                     dis[v] = dis[u] + w;
69                     pa[v] = u;
70                     if (rlx == n) negCycle[v] = true;
71                 }
72             }
73         }
74     }
75
76 // Negative Cycle Detection
77 void NegCycleDetect() {
78     /* No Neg Cycle: NO
79     Exist Any Neg Cycle:

```

```

80     YES
81     v0 v1 v2 ... vk v0 */
82
83     vector<int> src;
84     for (int i = 1; i <= n; i++)
85         src.emplace_back(i);
86
87     SPFA(src);
88     // BellmanFord(src);
89
90     int ptr = -1;
91     for (int i = 1; i <= n; i++) {
92         if (negCycle[i]) {
93             ptr = i;
94             break;
95         }
96
97         if (ptr == -1) {
98             return cout << "NO" << endl, void();
99         }
100
101        cout << "YES\n";
102        vector<int> ans;
103        vector<bool> vis(n + 1, false);
104
105        while (true) {
106            ans.emplace_back(ptr);
107            if (vis[ptr]) break;
108            vis[ptr] = true;
109            ptr = pa[ptr];
110        }
111        reverse(ans.begin(), ans.end());
112
113        vis.assign(n + 1, false);
114        for (auto& x : ans) {
115            cout << x << ' ';
116            if (vis[x]) break;
117            vis[x] = true;
118        }
119        cout << endl;
120    }
121
122 // Distance Calculation
123 void calcDis(int s) {
124     vector<int> src;
125     src.emplace_back(s);
126     SPFA(src);
127     // BellmanFord(src);
128
129     while (!q.empty()) q.pop();
130     for (int i = 1; i <= n; i++) {
131         if (negCycle[i]) q.push(i);
132
133         while (!q.empty()) {
134             int u = q.front();
135             q.pop();
136             for (auto& e : g[u]) {
137                 int v = e.first;
138                 if (!negCycle[v]) {
139                     q.push(v);
140                     negCycle[v] = true;
141                 }
142             }
143         }
144     }

```

5.4 BCC - AP

```

1 int n, m;
2 int low[maxn], dfn[maxn], instp;
3 vector<int> E, g[maxn];
4 bitset<maxn> isap;
5 bitset<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<maxm> vis, isbrcg;
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     Each(e, g[u]) {
31         if (vis[e]) continue;
32         vis[e] = true;
33         int v = E[e] ^ u;
34         if (dfn[v]) {
35             // back edge
36             low[u] = min(low[u], dfn[v]);
37         } else {
38             // tree edge
39             dfs(v);
40             low[u] = min(low[u], low[v]);
41             if (low[v] == dfn[v]) {
42                 isbrcg[e] = true;
43                 popout(u);
44             }
45         }
46     }
47 }
48 void solve() {
49     FOR(i, 1, n + 1, 1) {
50         if (!dfn[i]) dfs(i);
51     }
52     vector<pii> ans;
53     vis.reset();
54     FOR(u, 1, n + 1, 1) {
55         Each(e, g[u]) {
56             if (!isbrcg[e] || vis[e]) continue;
57             vis[e] = true;
58             int v = E[e] ^ u;
59             ans.emplace_back(mp(u, v));
60         }
61     }
62     cout << (int)ans.size() << endl;
63     Each(e, ans) cout << e.F << ' ' << e.S << endl;
64 }
65 }
66 }

```

```

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     Each(e, g[u]) {
31         if (vis[e]) continue;
32         vis[e] = true;
33         int v = E[e] ^ u;
34         if (dfn[v]) {
35             // back edge
36             low[u] = min(low[u], dfn[v]);
37         } else {
38             // tree edge
39             dfs(v);
40             low[u] = min(low[u], low[v]);
41             if (low[v] == dfn[v]) {
42                 isbrcg[e] = true;
43                 popout(u);
44             }
45         }
46     }
47 }
48 void solve() {
49     FOR(i, 1, n + 1, 1) {
50         if (!dfn[i]) dfs(i);
51     }
52     vector<pii> ans;
53     vis.reset();
54     FOR(u, 1, n + 1, 1) {
55         Each(e, g[u]) {
56             if (!isbrcg[e] || vis[e]) continue;
57             vis[e] = true;
58             int v = E[e] ^ u;
59             ans.emplace_back(mp(u, v));
60         }
61     }
62     cout << (int)ans.size() << endl;
63     Each(e, ans) cout << e.F << ' ' << e.S << endl;
64 }
65 }
66 }

```

5.6 SCC - Tarjan

```

1 // 2-SAT
2 vector<int> E, g[maxn]; // 1~n, n+1~2n
3 int low[maxn], in[maxn], instp;
4 int sccnt, sccid[maxn];
5 stack<int> stk;
6 bitset<maxn> ins, vis;
7 int n, m;
8 void init() {
9     cin >> m >> n;
10    E.clear();
11    fill(g, g + maxn, vector<int>());
12    fill(low, low + maxn, INF);
13    memset(in, 0, sizeof(in));
14    instp = 1;
15    sccnt = 0;
16    memset(sccid, 0, sizeof(sccid));
17    ins.reset();
18    vis.reset();
19 }
20 inline int no(int u) {
21     return (u > n ? u - n : u + n);
22 }
23 int ecnt = 0;
24 inline void clause(int u, int v) {
25     E.eb(no(u) ^ v);
26     g[no(u)].eb(ecnt++);
27     E.eb(no(v) ^ u);
28     g[no(v)].eb(ecnt++);
29 }
30 void dfs(int u) {
31     in[u] = instp++;
32     low[u] = in[u];
33     stk.push(u);
34     ins[u] = true;
35 }

```

```

36     Each(e, g[u]) {
37         if (vis[e]) continue;
38         vis[e] = true;
39
40         int v = E[e] ^ u;
41         if (ins[v])
42             low[u] = min(low[u], in[v]);
43         else if (!in[v]) {
44             dfs(v);
45             low[u] = min(low[u], low[v]);
46         }
47     }
48     if (low[u] == in[u]) {
49         scnt++;
50         while (!stk.empty()) {
51             int v = stk.top();
52             stk.pop();
53             ins[v] = false;
54             sccid[v] = scnt;
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         ;
82     }
83 }
```

5.7 SCC - Kosaraju

```

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

```

30             dfs(i);
31         }
32     }
33     SCC_cnt = 0;
34     vis = 0;
35     for (int i = n - 1; i >= 0; i--) {
36         if (!vis[pre[i]]) {
37             SCC_cnt++;
38             dfs2(pre[i]);
39         }
40     }
41 }
```

5.8 Eulerian Path - Undir

```

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

5.9 Eulerian Path - Dir

```

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

```

36     cout << u << ' ';
37 }
38 }
```

5.10 Hamilton Path

```

1 // top down DP
2 // Be Aware Of Multiple Edges
3 int n, m;
4 ll dp[maxn][1<<maxn];
5 int adj[maxn][maxn];
6
7 void init() {
8     cin >> n >> m;
9     fill(dp[0], dp[maxn-1]+(1<<maxn), -1);
10 }
11
12 void DP(int i, int msk) {
13     if (dp[i][msk] != -1) return;
14     dp[i][msk] = 0;
15     REP(j, n) if (j != i && (msk & (1<<j)) && adj[j][i]
16         ]) {
17         int sub = msk ^ (1<<i);
18         if (dp[j][sub] == -1) DP(j, sub);
19         dp[i][msk] += dp[j][sub] * adj[j][i];
20         if (dp[i][msk] >= MOD) dp[i][msk] %= MOD;
21     }
22
23
24 int main() {
25     WiwiHorz
26     init();
27
28     REP(i, m) {
29         int u, v;
30         cin >> u >> v;
31         if (u == v) continue;
32         adj[--u][--v]++;
33     }
34
35     dp[0][1] = 1;
36     FOR(i, 1, n, 1) {
37         dp[i][1] = 0;
38         dp[i][1|(1<<i)] = adj[0][i];
39     }
40     FOR(msk, 1, (1<<n), 1) {
41         if (msk == 1) continue;
42         dp[0][msk] = 0;
43     }
44
45     DP(n-1, (1<<n)-1);
46     cout << dp[n-1][(1<<n)-1] << endl;
47
48     return 0;
49 }
50 }
```

5.11 Kth Shortest Path

```

1 // time: O(|E| \lg |E| + |V| \lg |V| + K)
2 // memory: O(|E| \lg |E| + |V| |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->d
19        > b->edge->d; }
20    struct node {
21        int v;
22        ll d;
```

```

22        heap* H;
23        nd* E;
24        node() {}
25        node(ll _d, int _v, nd* _E) {
26            d = _d;
27            v = _v;
28            E = _E;
29        }
30        node(heap* _H, ll _d) {
31            H = _H;
32            d = _d;
33        }
34        friend bool operator<(node a, node b) { return
35            a.d > b.d; }
36    };
37    int n, k, s, t, dst[N];
38    nd* nxt[N];
39    vector<nd*> g[N], rg[N];
40    heap *nullNd, *head[N];
41    void init(int _n, int _k, int _s, int _t) {
42        n = _n;
43        k = _k;
44        s = _s;
45        t = _t;
46        for (int i = 1; i <= n; i++) {
47            g[i].clear();
48            rg[i].clear();
49            nxt[i] = NULL;
50            head[i] = NULL;
51            dst[i] = -1;
52        }
53        void addEdge(int ui, int vi, ll di) {
54            nd* e = new nd(ui, vi, di);
55            g[ui].push_back(e);
56            rg[vi].push_back(e);
57        }
58        queue<int> dfsQ;
59        void dijkstra() {
60            while (dfsQ.size()) dfsQ.pop();
61            priority_queue<node> Q;
62            Q.push(node(0, t, NULL));
63            while (!Q.empty()) {
64                node p = Q.top();
65                Q.pop();
66                if (dst[p.v] != -1) continue;
67                dst[p.v] = p.d;
68                nxt[p.v] = p.E;
69                dfsQ.push(p.v);
70                for (auto e : rg[p.v]) Q.push(node(p.d + e
71                    ->d, e->u, e));
72            }
73        }
74        heap* merge(heap* curNd, heap* newNd) {
75            if (curNd == nullNd) return newNd;
76            heap* root = new heap;
77            memcpy(root, curNd, sizeof(heap));
78            if (newNd->edge->d < curNd->edge->d) {
79                root->edge = newNd->edge;
80                root->chd[2] = newNd->chd[2];
81                root->chd[3] = newNd->chd[3];
82                newNd->edge = curNd->edge;
83                newNd->chd[2] = curNd->chd[2];
84                newNd->chd[3] = curNd->chd[3];
85            }
86            if (root->chd[0]->dep < root->chd[1]->dep)
87                root->chd[0] = merge(root->chd[0], newNd);
88            else
89                root->chd[1] = merge(root->chd[1], newNd);
90            root->dep = max(root->chd[0]->dep,
91                             root->chd[1]->dep) +
92                1;
93        }
94        vector<heap*> V;
95        void build() {
96            nullNd = new heap;
97            nullNd->dep = 0;
98            nullNd->edge = new nd;
99            fill(nullNd->chd, nullNd->chd + 4, nullNd);
100           while (not dfsQ.empty()) {
101               int u = dfsQ.front();
```

```

102    dfsQ.pop();
103    if (!nxt[u])
104        head[u] = nullNd;
105    else
106        head[u] = head[nxt[u]->v];
107    V.clear();
108    for (auto&& e : g[u]) {
109        int v = e->v;
110        if (dst[v] == -1) continue;
111        e->d += dst[v] - dst[u];
112        if (nxt[u] != e) {
113            heap* p = new heap;
114            fill(p->chd, p->chd + 4, nullNd);
115            p->dep = 1;
116            p->edge = e;
117            V.push_back(p);
118        }
119    }
120    if (V.empty()) continue;
121    make_heap(V.begin(), V.end(), cmp);
122 #define L(X) ((X << 1) + 1)
123 #define R(X) ((X << 1) + 2)
124     for (size_t i = 0; i < V.size(); i++) {
125         if (L(i) < V.size())
126             V[i]->chd[2] = V[L(i)];
127         else
128             V[i]->chd[2] = nullNd;
129         if (R(i) < V.size())
130             V[i]->chd[3] = V[R(i)];
131         else
132             V[i]->chd[3] = nullNd;
133     }
134     head[u] = merge(head[u], V.front());
135 }
136 vector<ll> ans;
137 void first_K() {
138     ans.clear();
139     priority_queue<node> Q;
140     if (dst[s] == -1) return;
141     ans.push_back(dst[s]);
142     if (head[s] != nullNd)
143         Q.push(node(head[s], dst[s] + head[s]->edge22
144             ->d));
145     for (int _ = 1; _ < k and not Q.empty(); _++) {23
146         node p = Q.top(), q;
147         Q.pop();
148         ans.push_back(p.d);
149         if (head[p.H->edge->v] != nullNd) {25
150             q.H = head[p.H->edge->v];
151             q.d = p.d + q.H->edge->d;
152             Q.push(q);
153         }
154         for (int i = 0; i < 4; i++) {26
155             if (p.H->chd[i] != nullNd) {27
156                 q.H = p.H->chd[i];
157                 q.d = p.d - p.H->edge->d + p.H->chd34
158                     [i]->edge->d;
159                 Q.push(q);
160             }
161         }
162     }
163     void solve() { // ans[i] stores the i-th shortest
164         path
165         dijkstra();
166         build();
167         first_K(); // ans.size() might less than k
168     }
169 } 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 to get $S_{i+1} - S_i \geq 0$ if x_i needs to be non-negative.
- $\frac{x_u}{x_v} \leq c \Rightarrow \log x_u - \log x_v \leq \log c$

6 String

6.1 Aho Corasick

```

1 struct ACautomata {
2     struct Node {
3         int cnt; // 停在此節點的數量
4         Node *go[26], *fail, *dic;
5         // 子節點 fail 指標 最近的模式結尾
6         Node() {
7             cnt = 0;
8             fail = 0;
9             dic = 0;
10            memset(go, 0, sizeof(go));
11        }
12    } pool[1048576], *root;
13    int nMem;
14    Node *new_Node() {
15        pool[nMem] = Node();
16        return &pool[nMem++];
17    }
18    void init() {
19        nMem = 0;
20        root = new_Node();
21    }
22    void add(const string &str) { insert(root, str, 0); }
23    void insert(Node *cur, const string &str, int pos) {
24        for (int i = pos; i < str.size(); i++) {
25            if (!cur->go[str[i] - 'a'])
26                cur->go[str[i] - 'a'] = new_Node();
27            cur = cur->go[str[i] - 'a'];
28        }
29        cur->cnt++;
30    }
31    void make_fail() { // 全部 add 完做
32        queue<Node *> que;
33        que.push(root);
34        while (!que.empty()) {
35            Node *fr = que.front();
36            que.pop();
37            for (int i = 0; i < 26; i++) {
38                if (fr->go[i]) {
39                    Node *ptr = fr->fail;
40                    while (ptr && !ptr->go[i]) ptr =
41                        ptr->fail;
42                    fr->go[i]->fail = ptr = (ptr ? ptr
43                        ->go[i] : root);
44                    fr->go[i]->dic = (ptr->cnt ? ptr :
45                        ptr->dic);
46                    que.push(fr->go[i]);
47                }
48            }
49        }
50    }
51    // 出現過不同 string 的總數
52    int query_unique(const string& text) {
53        Node* p = root;
54        int ans = 0;
55        for(char ch : text) {
56            int i = ch - 'a';
57            while(p && !p->go[i]) p = p->fail;
58            p = p ? p->go[i] : root;
59            if(p->cnt) {ans += p->cnt, p->cnt = 0;}
60        }
61    }

```

```

57     for(Node* t = p->dic; t; t = t->dic) if(t->cnt) {
58         cnt += t->cnt; t->cnt = 0;
59     }
60     return ans;
61 }
62 } AC;
63 }

while (0 <= i - m[i] - 1 && i + m[i] + 1 < 2 * n + 1 &&
       s[i - m[i] - 1] == s[i + m[i] + 1]) m[i]]++;
if (i + m[i] > mx + mxk) mx = i, mxk = m[i];
}

void init() {
    cin >> S;
    n = (int)S.size();
}

void solve() {
    manacher();
    int mx = 0, ptr = 0;
    for (int i = 0; i < 2 * n + 1; i++) {
        if (mx < m[i]) {
            mx = m[i];
            ptr = i;
        }
        for (int j = ptr - mx; j <= ptr + mx; j++) {
            if (s[j] != '.') cout << s[j];
        }
        cout << endl;
    }
}

```

6.2 KMP

```

1 vector<int> f;
2 // 沒匹配到可以退回哪裡
3 void buildFailFunction(string &s) {
4     f.resize(s.size(), -1);
5     for (int i = 1; i < s.size(); i++) {
6         int now = f[i - 1];
7         while (now != -1 and s[now + 1] != s[i]) now =
8             f[now];
9         if (s[now + 1] == s[i]) f[i] = now + 1;
10    }
11 }
12 void KMPmatching(string &a, string &b) {
13     for (int i = 0, now = -1; i < a.size(); i++) {
14         while (a[i] != b[now + 1] and now != -1) now =
15             f[now];
16         if (a[i] == b[now + 1]) now++;
17         if (now + 1 == b.size()) {
18             cout << "found a match start at position "
19                 << i - now << endl;
20             now = f[now];
21         }
22     }
}

```

6.3 Z Value

```

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

```

6.4 Manacher

```

1 // 找最長回文
2 int n;
3 string S, s;
4 vector<int> m;
5 void manacher() {
6     s.clear();
7     s.resize(2 * n + 1, '.');
8     for (int i = 0, j = 1; i < n; i++, j += 2) s[j] = S[i];
9     m.clear();
10    m.resize(2 * n + 1, 0);
11    // m[i] := max k such that s[i-k, i+k] is
12    // palindrome
13    int mx = 0, mxk = 0;
14    for (int i = 1; i < 2 * n + 1; i++) {
15        if (mx - (i - mx) >= 0) m[i] = min(m[mx - (i -
16            mx)], mx + mxk - i);
17    }
}

```

6.5 Suffix Array

```

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

```

```

50   for (int k = 0; (1 << k) < n; k++) {
51     for (int i = 0; i < n; i++)
52       buc[0][i] = make_pair(make_pair(rk[i],      54
53                             rk[(i + (1 << k)) % n]), i);
54     radix_sort();
55     if (fill_suf()) return;
56   }
57   void LCP() {
58     int k = 0;
59     for (int i = 0; i < n - 1; i++) {
60       if (rk[i] == 0) continue;
61       int pi = rk[i];
62       int j = suf[pi - 1];
63       while (i + k < n && j + k < n && s[i + k]
64             == s[j + k]) k++;
65       lcp[pi] = k;
66       k = max(k - 1, 0);
67     }
68 }
69 SuffixArray suffixarray;

```

6.6 Suffix Automaton

```

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

```

```

54   vector<int> order(m);
55   iota(order.begin(), order.end(), 0);
56   sort(order.begin(), order.end(), [&](int a, int
57     b){ return st[a].len > st[b].len; });
58   for (int v : order) {
59     int p = st[v].link;
60     if (p != -1) occ[p] += occ[v];
61   }
62 }

```

6.7 Minimum Rotation

```

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

```

6.8 Lyndon Factorization

```

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

```

6.9 Rolling Hash

```

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

```

```

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

```

6.10 Trie

```

1 ppi 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         ppi &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 && y < a.y); }
20     // return sgn(x-a.x) < 0 || (sgn(x-a.x) == 0 && sgn(y-a.y) < 0); }
21     bool operator==(Pt a) { return sgn(x - a.x) == 0 && sgn(y - a.y) == 0; }
22 }
23
24 Pt mv(Pt a, Pt b) { return b - a; }
25 T len2(Pt a) { return a * a; }
26 T dis2(Pt a, Pt b) { return len2(b - a); }
27 Pt rotate(Pt u) { return {-u.y, u.x}; }
28 Pt unit(Pt x) { return x / sqrtl(x * x); }
29 short ori(Pt a, Pt b) { return ((a ^ b) > 0) - ((a ^ b) < 0); }
30 bool onseg(Pt p, Pt l1, Pt l2) {
31     Pt a = mv(p, l1), b = mv(p, l2);
32     return ((a ^ b) == 0) && ((a * b) <= 0);
33 }
34 inline T cross(const Pt &a, const Pt &b, const Pt &c) {
35     return (b.x - a.x) * (c.y - a.y)
36         - (b.y - a.y) * (c.x - a.x);
37 }
38
39 long double polar_angle(Pt ori, Pt pt){
40     return atan2(pt.y - ori.y, pt.x - ori.x);
41 }
42 // slope to degree atan(Slope) * 180.0 / acos(-1.0);
43 bool argcmp(Pt u, Pt v) {
44     auto half = [](const Pt& p) {
45         return p.y > 0 || (p.y == 0 && p.x >= 0);
46     };
47     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.b)) <= 0;
64     }
65     Pt proj(Pt& p, Line& l) {
66         Pt d = l.b - l.a;
67         T d2 = len2(d);
68         if (sgn(d2) == 0) return l.a;
69         T t = ((p - l.a) * d) / d2;
70         return l.a + d * t;
71     }
72     struct Cir {
73         Pt o;
74         T r;
75     };
76     bool disjunct(Cir a, Cir b) {
77         return sgn(sqrtl(len2(a.o - b.o)) - a.r - b.r) >=
78             0;
79     }
80     bool contain(Cir a, Cir b) {
81         return sgn(a.r - b.r - sqrtl(len2(a.o - b.o))) >=
82             0;
83 }

```

7.2 Sort by Angle

```

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

```

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(q1, p1, p2) ||
3         onseg(q2, p1, p2)) return true;
4     Pt p = mv(p1, p2), q = mv(q1, q2);
5     return (ori(p, mv(p1, q1)) * ori(p, mv(p1, q2)) <
6             0) && (ori(q, mv(q1, p1)) * ori(q, mv(q1, p2)) <
7             0);
8 }
9 // long double
10 Pt line_intersect(Pt a1, Pt a2, Pt b1, Pt b2) {
11     Pt da = mv(a1, a2), db = mv(b1, b2);
12     T det = da ^ db;
13     if (sgn(det) == 0) { // parallel
14         // return Pt(NAN, NAN);
15     }
16     T t = ((b1 - a1) ^ db) / det;
17     return a1 + da * t;
18 }
19 vector<Pt> CircleInter(Cir a, Cir b) {
20     double d2 = len2(a.o - b.o), d = sqrt(d2);
21     if (d < max(a.r, b.r) - min(a.r, b.r) || d > a.r +
22         b.r) return {};
23     Pt u = (a.o + b.o) / 2 + (a.o - b.o) * ((b.r * b.r -
24         a.r * a.r) / (2 * d2));
25     double A = sqrt((a.r + b.r + d) * (a.r - b.r + d) *
26         (a.r + b.r - d) * (-a.r + b.r + d));
27     Pt v = rotate(b.o - a.o) * A / (2 * d2);
28     if (sgn(v.x) == 0 and sgn(v.y) == 0) return {u};
29     return {u - v, u + v}; // counter clockwise of a
30 }

```

```

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[7
8                 hull.size() - 2], hull.back()), mv(hull[8
9                 [hull.size() - 2], ei)) == -1) {
10                 hull.pop_back();
11             }
12             hull.emplace_back(ei);
13         }
14         hull.pop_back();
15         reverse(pts.begin(), pts.end());
16     }
17     return hull;
18 }

```

7.6 Point In Convex

```

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

```

7.7 Point Segment Distance

```

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

```

```

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 * dx1 +
dy1 * dy1));
19 }

```

7.8 Point in Polygon

```

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

```

7.9 Minimum Euclidean Distance

```

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

```

7.10 Minkowski Sum

```

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

```

7.11 Lower Concave Hull

```

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

```

```

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    }

```

7.12 Pick's Theorem

Consider a polygon which vertices are all lattice points.

Let i = number of points inside the polygon.

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

Then we have the following formula:

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

7.13 Rotating SweepLine

```

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

```

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
15     int l = 0, r = -1;
16     for (size_t i = 0; i < P.size(); ++i) {
17         if (i && !argcmp(P[i - 1].dir(), P[i].dir()))
18             continue;
19         while (l < r && cover(P[i], P[r - 1], P[r])) --
20             r;
21         while (l < r && cover(P[i], P[l], P[l + 1])) ++
22             l;
23         P[++r] = P[i];
24     }
25     while (l < r && cover(P[l], P[r - 1], P[r])) --r;
26     while (l < r && cover(P[r], P[l], P[l + 1])) ++l;
27
28     if (r - 1 <= 1 || !argcmp(P[1].dir(), P[r].dir()))
29         return {};
30     if (cover(P[1 + 1], P[1], P[r])) return {};
31
32     return vector<Line>(P.begin() + 1, P.begin() + r + 1);
33 }

```

7.15 Minimum Enclosing Circle

```

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

```

7.16 Union of Circles

```

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

```

```

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 atan2l(p ^ q, p
3         * q); };
4     double r2 = (double)(C.r * C.r / 2);
5     auto tri = [&](Pt p, Pt q) {
6         Pt d = q - p;
7         T a = (d * p) / (d * d);
8         T b = ((p * p) - C.r * C.r) / (d * d);
9         T det = a * a - b;
10        if (det <= 0) return (double)(arg(p, q) * r2);
11        T s = max((T)0.0L, -a - sqrtl(det));
12        T t = min((T)1.0L, -a + sqrtl(det));
13        if (t < 0 || 1 <= s) return (double)(arg(p, q)
14            * r2);
15        Pt u = p + d * s, v = p + d * t;
16        return (double)(arg(p, u) * r2 + (u ^ v) / 2 +
17            arg(v, q) * r2);
18    };
19    long double sum = 0.0L;
20    for (int i = 0; i < (int)P.size(); i++)
21        sum += tri(P[i] - C.o, P[(i + 1) % P.size()] -
22            C.o);
23    return (double)fabsl(sum);
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    }
33    Pt line_plane_intersect(Pt u, Pt v, Pt a, Pt b, Pt c) {
34        // intersection of line uv and plane abc
35        Pt n = cross3(a, b, c);
36        double s = n * (u - v);
37        if (sign(s) == 0) return {-1, -1, -1}; // not found
38        return v + (u - v) * ((n * (a - v)) / s);
39    }
40    Pt rotateAroundAxis(Pt v, Pt axis, double theta) {
41        axis = axis / abs(axis); // axis must be unit
42        vector
43        double cosT = cos(theta);
44        double sinT = sin(theta);
45        Pt term1 = v * cosT;
46        Pt term2 = (axis ^ v) * sinT;
47        Pt term3 = axis * ((axis * v) * (1 - cosT));
48        return term1 + term2 + term3;
49    }
50 }
```

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 }
```

```

34
35     void transform(vector<cp> &a, cp *xomega) {
36         for (int i = 0; i < n; i++)
37             if (i < rev[i]) swap(a[i], a[rev[i]]);
38         for (int len = 2; len <= n; len <= 1) {
39             int mid = len >> 1;
40             int r = n / len;
41             for (int j = 0; j < n; j += len)
42                 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
51     void fft(vector<cp> &a) { transform(a, omega); }
52     void ifft(vector<cp> &a) {
53         transform(a, iomega);
54         for (int i = 0; i < n; i++) a[i] /= n;
55     }
56 } FFT;
57
58 const int MAXN = 262144;
59 // (must be 2^k)
60 // 262144, 524288, 1048576, 2097152, 4194304
61 // before any usage, run pre_fft() first
62 typedef long double ld;
63 typedef complex<ld> cplx; // real(), imag()
64 const ld PI = acosl(-1);
65 const cplx I(0, 1);
66 cplx omega[MAXN + 1];
67 void pre_fft() {
68     for (int i = 0; i <= MAXN; i++) {
69         omega[i] = exp(i * 2 * PI / MAXN * I);
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    }

```

```
10  
11 long long a[MAXN];  
12 long long b[MAXN];  
13 long long ans[MAXN];  
14 int a_length;  
15 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 }
12 ll pollard_rho(ll n) {
13     if (!(n & 1)) return 2;
14     while (true) {
15         ll y = 2, x = rand() % (n - 1) + 1, res = 1;
16         for (int sz = 2; res == 1; sz *= 2) {
17             for (int i = 0; i < sz && res <= 1; i++) {
18                 x = f(x, n);
19                 res = __gcd(llabs(x - y), n);
20             }
21             y = x;
22         }
23         if (res != 0 && res != n) return res;
24     }
25 }
26 vector<ll> ret;
27 void fact(ll x) {
28     if (miller_rabin(x)) {
29         ret.push_back(x);
30         return;
31     }
32     ll f = pollard_rho(x);
33     fact(f);
34     fact(x / f);
35 }

```

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 : pirmes <= 13
4 // n < 2^64                     7 :
5 // 2, 325, 9375, 28178, 450775, 9780504, 1795265022
6 bool witness(ll a, ll n, ll u, int t) {
7     if (!(a % n)) return 0;
8     ll x = mypow(a, u, n);
9     for (int i = 0; i < t; i++) {
10         ll nx = mul(x, x, n);
11         if (nx == 1 && x != 1 && x != n - 1) return 1;
12         x = nx;
13     }
14     return x != 1;
15 }
16 bool miller_rabin(ll n, int s = 100) {
17     // iterate s times of witness on n
18     // return 1 if prime, 0 otherwise
19     if (n < 2) return 0;
20     if (!(n & 1)) return n == 2;
21     ll u = n - 1;
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         }
23         if (j >= lpf[i]) break;
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     }

```

```

26     long long am = mod_pow(a, m, p);
27     long long am_inv = mod_inv(am, p);
28     long long gamma = y % p;
29     for(long long i=0;i<=m;++i){
30         auto it = table.find(gamma);
31         if(it != table.end()){
32             long long x = i*m + it->second;
33             return x;
34         }
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 }

```

15 }

8.10 LinearSieve

```

1 const int C = 1e7 + 2;
2 int mo[C], lp[C], phi[C], isp[C];
3 vector<int> prime;
4 void sieve() {
5     mo[1] = phi[1] = 1;
6     for(int i = 1; i < C; i++) lp[i] = 1;
7     for(int i = 2; i < C; i++) {
8         if(lp[i] == 1) {
9             lp[i] = i;
10            prime.push_back(i);
11            isp[i] = 1;
12            mo[i] = -1;
13            phi[i] = i - 1;
14        }
15        for(int p : prime) {
16            if(i * p >= C) break;
17            lp[i * p] = p;
18            if(i % p == 0) {
19                phi[p * i] = phi[i] * p;
20                break;
21            }
22            phi[i * p] = phi[i] * (p - 1);
23            mo[i * p] = mo[i] * mo[p];
24        }
25    }
26 }
```

8.11 Other Formulas

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

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

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

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

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

- Extended Euclidean algorithm:

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

- Divisor function:

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

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

- 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:

- $\epsilon(n) = [n = 1]$

- $1(n) = 1$

- $id(n) = n$

4. $\mu(n) = 0$ if n has squared prime factor

- $\mu(n) = (-1)^k$ if $n = p_1 p_2 \cdots p_k$

- $\epsilon = \mu * id$

- $[n = 1] = \sum_{d|n} \mu(d)$

- $[gcd = 1] = \sum_{d|gcd} \mu(d)$

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

8.12 Polynomial

```

1 const int maxk = 20;
2 const int maxn = 1<<maxk;
3 const ll LINF = 1e18;

4 /* P = r*2^k + 1
5   P           r   k   g
6   998244353  119 23  3
7   1004535809 479 21  3
8
9   P           r   k   g
10  3            1   1   2
11  5            1   2   2
12  17           1   4   3
13  97           3   5   5
14  193          3   6   5
15  257          1   8   3
16  7681          15  9   17
17  12289         3   12  11
18  40961         5   13  3
19  65537         1   16  3
20  786433        3   18  10
21  5767169        11  19  3
22  7340033        7   20  3
23  23068673        11  21  3
24  104857601       25  22  3
25  167772161       5   25  3
26  469762049       7   26  3
27  1004535809      479 21  3
28  2013265921      15  27  31
29  2281701377      17  27  3
30  3221225473      3   30  5
31  75161927681      35  31  3
32  77309411329      9   33  7
33  206158430209      3   36  22
34  2061584302081      15  37  7
35  2748779069441      5   39  3
36  6597069766657      3   41  5
37  39582418599937      9   42  5
38  79164837199873      9   43  5
39  263882790666241      15  44  7
40  1231453023109121      35  45  3
41  1337006139375617      19  46  3
42  3799912185593857      27  47  5
43  4222124650659841      15  48  19
44  7881299347898369      7   50  6
45  31525197391593473      7   52  3
46  180143985094819841      5   55  6
47  194555039024054273      27  56  5
48  4179340454199820289      29  57  3
49  9097271247288401921      505 54  6 */
50
51 const int g = 3;
52 const ll MOD = 998244353;
53
54 ll pw(ll a, ll n) { /* fast pow */ }
55
56 #define siz(x) (int)x.size()
57
58 template<typename T>
59 vector<T>& operator+=(vector<T>& a, const vector<T>& b)
60 {
61     if (siz(a) < siz(b)) a.resize(siz(b));
62     for (int i = 0; i < min(siz(a), siz(b)); i++) {
63         a[i] += b[i];
64         a[i] -= a[i] >= MOD ? MOD : 0;
65     }
66     return a;
67 }
```

9 Linear Algebra

9.1 Gaussian-Jordan Elimination

```

2 vector<vector<ll>> v;
3 void gauss(vector<vector<ll>>& v) {
4     int r = 0;
5     for (int i = 0; i < n; i++) {
6         bool ok = false;
7         for (int j = r; j < n; j++) {
8             if (v[j][i] == 0) continue;
9             swap(v[j], v[r]);
10            ok = true;
11            break;
12        }
13        if (!ok) continue;
14        ll div = inv(v[r][i]);
15        for (int j = 0; j < n + 1; j++) {
16            v[r][j] *= div;
17            if (v[r][j] >= MOD) v[r][j] %= MOD;
18        }
19        for (int j = 0; j < n; j++) {
20            if (j == r) continue;
21            ll t = v[j][i];
22            for (int k = 0; k < n + 1; k++) {
23                v[j][k] -= v[r][k] * t % MOD;
24                if (v[j][k] < 0) v[j][k] += MOD;
25            }
26        }
27    r++;
28 }
29 }
```

9.2 Determinant

1. Use GJ Elimination, if there's any row consists of only 0, then $\det = 0$, otherwise $\det = \text{product of diagonal elements}$.
2. Properties of \det :
 - Transpose: Unchanged
 - Row Operation 1 - Swap 2 rows: $-\det$
 - Row Operation 2 - $k\vec{r}_i$: $k \times \det$
 - Row Operation 3 - $k\vec{r}_i$ add to \vec{r}_j : Unchaged

10 Combinatorics

10.1 Catalan Number

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

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

10.2 Burnside's Lemma

Let X be the original set.

Let G be the group of operations acting on X .

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

Let X/G be the set of orbits.

Then the following equation holds:

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



