

Contents

1 Misc	2	4 Dynamic-Programming	8	7 Math	20
1.1 Custom Set PQ Sort	2	4.1 Digit DP	8	7.1 CRT m Coprime	20
1.2 Default Code New	2	4.2 SOS DP	8	7.2 CRT m Not Coprime	20
1.3 Default Code Old	2	4.3 Integer Partition	8	7.3 Fraction	20
1.4 Enumerate Subset	2	5 Geometry	8	7.4 Josephus Problem	21
1.5 Fast Input	2	5.1 Geometry Struct	8	7.5 Lagrange any x	21
1.6 OEIS	2	5.2 Geometry 卦長	10	7.6 Lagrange continuous x	21
1.7 Xor Basis	2	5.3 Pick's Theorem	13	7.7 Lucas's Theorem	22
1.8 random int	3	6 Graph	13	7.8 Matrix	22
1.9 Python	3	6.1 2-SAT	13	7.9 Matrix 01	22
1.10 diff	3	6.2 Augment Path	13	7.10 Miller Rabin	22
1.11 hash command	3	6.3 Bridge BCC	14	7.11 Pollard Rho	22
1.12 run	3	6.4 Cut BCC	14	7.12 Quick Pow	23
1.13 setup	3	6.5 Dijkstra	14	7.13 數論分塊	23
2 Convolution	3	6.6 Dinic	14	7.14 最大質因數	23
2.1 FFT any mod	3	6.7 Dominator Tree	15	7.15 歐拉公式	23
2.2 FFT new	4	6.8 Enumerate Triangle	15	7.16 線性篩	23
2.3 FFT old	4	6.9 Find Bridge	15	7.17 Burnside's Lemma	23
2.4 NTT mod 998244353	4	6.10 HLD	15	7.18 Catalan Number	23
3 Data-Structure	5	6.11 Kosaraju	16	7.19 Matrix Tree Theorem	23
3.1 BIT	5	6.12 Kuhn Munkres	16	7.20 Stirling's formula	23
3.2 Disjoint Set Persistent	5	6.13 LCA	17	7.21 Theorem	23
3.3 PBDS GP Hash Table	5	6.14 MCMF	17	7.22 二元一次方程式	24
3.4 PBDS Order Set	5	6.15 Tarjan	17	7.23 歐拉定理	24
3.5 Segment Tree Add Set	5	6.16 Tarjan Find AP	18	7.24 錯排公式	24
3.6 Segment Tree Li Chao Line	6	6.17 Tree Isomorphism	18	8 String	24
3.7 Segment Tree Li Chao Segment	6	6.18 圓方樹	19	8.1 Hash	24
3.8 Segment Tree Persistent	7	6.19 最大權閉合圖	19	8.2 KMP	24
3.9 Sparse Table	7	6.20 Theorem	20	8.3 Manacher	24
				8.4 Min Rotation	24
				8.5 Suffix Array	24
				8.6 Z Algorithm	25
				8.7 k-th Substring1	25

1 Misc

1.1 Custom Set PQ Sort

```

1 // priority_queue · 務必檢查相等的 case · 給所有元素一個排序的
  依據
2 struct cmp{
3     bool operator () (Data a, Data b){
4         return a.x<b.x;
5     }
6 };
7 priority_queue<Data, vector<Data>, cmp> pq;
8
9 // set · 務必檢查相等的 case · 給所有元素一個排序的依據
10 struct Data{
11     int x;
12
13     bool operator < (const Data &b) const {
14         return x<b.x;
15     }
16 };

```

1.2 Default Code New

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define int long long
4
5 const int MAX_N = 5e5 + 10;
6 const int INF = 2e18;
7
8 void solve(){
9
10 }
11
12 signed main(){
13     ios::sync_with_stdio(0), cin.tie(0);
14
15     int t = 1;
16     while (t--){
17         solve();
18     }
19
20     return 0;
21 }

```

1.3 Default Code Old

```

1 #include <bits/stdc++.h>
2 #define int long long
3 #define ALL(x) x.begin(), x.end()
4 #define SZ(x) ((int)x.size())
5 #define fastio ios::sync_with_stdio(0), cin.tie(0);
6 using namespace std;
7
8 #ifdef LOCAL
9 #define cout cout << "\033[0;32m"

```

```

10 #define cerr cerr << "\033[0;31m"
11 #define endl endl << "\033[0m"
12 #else
13 #pragma GCC optimize("O3,unroll-loops")
14 #pragma GCC target("avx,avx2,sse,sse2,sse3,sse4,popcnt")
15 #define endl "\n"
16 #endif
17
18 const int MAX_N = 5e5+10;
19 const int INF = 2e18;
20
21 void solve1(){
22
23     return;
24 }
25
26 signed main(){
27
28     fastio;
29
30     int t = 1;
31     while (t--){
32         solve1();
33     }
34
35     return 0;
36 }

```

1.4 Enumerate Subset

```

1 // 時間複雜度  $O(3^n)$ 
2 // 枚舉每個 mask 的子集
3 for (int mask=0; mask<(1<<n); mask++){
4     for (int s=mask; s>=0; s=(s-1)&m){
5         // s 是 mask 的子集
6         if (s==0) break;
7     }
8 }

```

1.5 Fast Input

```

1 // fast IO
2 // 6f8879
3 inline char readchar(){
4     static char buffer[BUFSIZ], * now = buffer + BUFSIZ, *
      end = buffer + BUFSIZ;
5     if (now == end)
6     {
7         if (end < buffer + BUFSIZ)
8             return EOF;
9         end = (buffer + fread(buffer, 1, BUFSIZ, stdin));
10        now = buffer;
11    }
12    return *now++;
13 }
14 inline int nextint(){
15     int x = 0, c = readchar(), neg = false;
16     while (('0' > c || c > '9') && c != '-' && c != EOF) c =
      readchar();

```

```

17     if (c == '-') neg = true, c = readchar();
18     while ('0' <= c && c <= '9') x = (x<<3) + (x<<1) + (c^'0')
      , c = readchar();
19     if (neg) x = -x;
20     return x; // returns 0 if EOF
21 }

```

1.6 OEIS

```

1 // 若一個線性遞迴有 k 項 · 給他恰好 2*k 個項可以求出線性遞迴
2 // f915c2
3 template <typename T>
4 vector<T> BerlekampMassey(vector<T> a) {
5     auto scalarProduct = [](vector<T> v, T c) {
6         for (T &x: v) x *= c;
7         return v;
8     };
9     vector<T> s, best;
10    int bestPos = 0;
11    for (size_t i = 0; i < a.size(); i++) {
12        T error = a[i];
13        for (size_t j = 0; j < s.size(); j++) error -= s[j] *
          a[i-1-j];
14        if (error == 0) continue;
15        if (s.empty()) {
16            s.resize(i + 1);
17            bestPos = i;
18            best.push_back(1 / error);
19            continue;
20        }
21        vector<T> fix = scalarProduct(best, error);
22        fix.insert(fix.begin(), i - bestPos - 1, 0);
23        if (fix.size() >= s.size()) {
24            best = scalarProduct(s, - 1 / error);
25            best.insert(best.begin(), 1 / error);
26            bestPos = i;
27            s.resize(fix.size());
28        }
29        for (size_t j = 0; j < fix.size(); j++)
30            s[j] += fix[j];
31    }
32    return s;
33 }

```

1.7 Xor Basis

```

1 vector<int> basis;
2 void add_vector(int x){
3     for (auto v : basis){
4         x=min(x, x^v);
5     }
6     if (x) basis.push_back(x);
7 }
8
9 // 給一數字集合 S · 求能不能 XOR 出 x
10 bool check(int x){
11     for (auto v : basis){
12         x=min(x, x^v);
13     }

```

```

14 |     return x;
15 | }
16 |
17 | // 給一數字集合 S · 求能 XOR 出多少數字
18 | // 答案等於 2^{basis 的大小}
19 |
20 | // 給一數字集合 S · 求 XOR 出最大的數字
21 | int get_max(){
22 |     int ans=0;
23 |     for (auto v : basis){
24 |         ans=max(ans, ans^v);
25 |     }
26 |     return ans;
27 | }

```

1.8 random int

```

1 | mt19937 seed(chrono::steady_clock::now().time_since_epoch().
  | count());
2 | int rng(int l, int r){
3 |     return uniform_int_distribution<int>(l, r)(seed);
4 | }

```

1.9 Python

```

1 | sys.setrecursionlimit(100000)
2 |
3 | sys.set_int_max_str_digits(10000)

```

1.10 diff

```

1 | set -e
2 | g++ ac.cpp -o ac
3 | g++ wa.cpp -o wa
4 | for ((i=0;;i++))
5 | do
6 |     echo "$i"
7 |     python3 gen.py > input
8 |     ./ac < input > ac.out
9 |     ./wa < input > wa.out
10 |    diff ac.out wa.out || break
11 | done

```

1.11 hash command

```

1 | cat file.cpp | cpp -dD -P -fpreprocessed | tr -d "[:space:]"
  | | md5sum | cut -c-6

```

1.12 run

```

1 | import os
2 |
3 | f = "pA"
4 |
5 | while 1:
6 |     i = input("input: ")
7 |     p = os.listdir(".")
8 |     if i != "":
9 |         f = i
10 |        print(f"file = {f}")
11 |        if os.system(f"g++ {f}.cpp -std=c++17 -Wall -Wextra -
          | Wshadow -O2 -D LOCAL -g -fsanitize=undefined,address
          | -o {f}"):
12 |            print("CE")
13 |            continue
14 |        os.system("clear")
15 |
16 |        for x in sorted(p):
17 |            if f in x and ".in" in x:
18 |                print(x)
19 |                if os.system(f"./{f} < {x}"):
20 |                    print("RE")
21 |                    print()

```

1.13 setup

```

1 | se nu rnu bs=2 sw=4 ts=4 hls ls=2 si acd bo=all mouse=a
2 |
3 | :inoremap " ""<Esc>i
4 | :inoremap {<CR> {<CR><Esc>ko
5 | :inoremap [{ {<Esc>i
6 |
7 | function! F(...)
8 |     execute '!./%:r < ./' . a:1
9 | endfunction
10 | command! -nargs=* R call F(<f-args>)
11 |
12 | map <F7> :w<bar>!g++ "%" -o %:r -std=c++17 -Wall -Wextra -
          | Wshadow -O2 -DLOCAL -g -fsanitize=undefined,address<CR>
13 | map <F8> :!./%:r<CR>
14 | map <F9> :!./%:r < ./%:r.in<CR>
15 |
16 | ca hash w !cpp -dD -P -fpreprocessed \\\ tr -d "[:space:]" \\\
          | md5sum \\\ cut -c-6
17 |
18 | " i+<esc>25A---<esc>
19 | " o|<esc>25A |<esc>
20 | " ggVGyG35pGdd

```

2 Convolution

2.1 FFT any mod

```

1 | /*
2 | 修改 const int MOD = 998244353 更改要取餘的數字

```

```

3 | PolyMul(a, b) 回傳多項式乘法的結果 ( c_k = \sum_{i+j} a_i b_j
  | mod MOD )
4 |
5 | 大約可以支援 5e5 · a_i, b_i 皆在 MOD 以下的非負整數
6 | */
7 | const int MOD = 998244353;
8 | typedef complex<double> cd;
9 |
10 | // b9c90a
11 | void FFT(vector<cd> &a) {
12 |     int n = a.size(), L = 31-__builtin_clz(n);
13 |     vector<complex<long double>> R(2, 1);
14 |     vector<cd> rt(2, 1);
15 |     for (int k=2; k<n; k*=2){
16 |         R.resize(n);
17 |         rt.resize(n);
18 |         auto x = polar(1.0L, acos(-1.0L) / k);
19 |         for (int i=k; i<2*k; i++){
20 |             rt[i] = R[i] = (i&1 ? R[i/2]*x : R[i/2]);
21 |         }
22 |     }
23 |
24 |     vector<int> rev(n);
25 |     for (int i=0; i<n; i++){
26 |         rev[i] = (rev[i/2] | (i&1)<<L)/2;
27 |     }
28 |     for (int i=0; i<n; i++){
29 |         if (i<rev[i]) swap(a[i], a[rev[i]]);
30 |     }
31 |     for (int k=1; k<n; k*=2){
32 |         for (int i=0; i<n; i+=2*k){
33 |             for (int j=0; j<k; j++){
34 |                 auto x = (double *)&rt[j+k];
35 |                 auto y = (double *)&a[i+j+k];
36 |                 cd z(x[0]*y[0] - x[1]*y[1], x[0]*y[1] + x[1]*
          | y[0]);
37 |                 a[i+j+k] = a[i+j]-z;
38 |                 a[i+j] += z;
39 |             }
40 |         }
41 |     }
42 |     return;
43 | }
44 |
45 | // d3c65e
46 | vector<int> PolyMul(vector<int> a, vector<int> b){
47 |     if (a.empty() || b.empty()) return {};
48 |
49 |     vector<int> res(a.size()+b.size()-1);
50 |     int B = 32-__builtin_clz(res.size()), n = (1<<B), cut =
          | int(sqrt(MOD));
51 |     vector<cd> L(n), R(n), outs(n), outl(n);
52 |
53 |     for (int i=0; i<a.size(); i++){
54 |         L[i] = cd((int) a[i]/cut, (int)a[i]%cut);
55 |     }
56 |     for (int i=0; i<b.size(); i++){
57 |         R[i] = cd((int) b[i]/cut, (int)b[i]%cut);
58 |     }
59 |     FFT(L);
60 |     FFT(R);
61 |     for (int i=0; i<n; i++){
62 |         int j = -i&(n-1);
63 |         outl[j] = (L[i]+conj(L[j])) * R[i]/(2.0*n);
64 |         outs[j] = (L[i]-conj(L[j])) * R[i]/(2.0*n)/1i;

```

```

65 }
66 FFT(outl);
67 FFT(outs);
68 for (int i=0 ; i<res.size() ; i++){
69     int av = (int)(real(outl[i])+0.5), cv = (int)(imag(
70         outs[i])+0.5);
71     int bv = (int)(imag(outl[i])+0.5) + (int)(real(outs[i]
72         )+0.5);
73     res[i] = ((av%MOD*cut+bv) % MOD*cut+cv) % MOD;
74 }
75 return res;
76 }

```

```

46 for (int i=0 ; i<b.size() ; i++){
47     in[i].imag(b[i]);
48 }
49 FFT(in);
50 for (cd& x : in) x *= x;
51 for (int i=0 ; i<n ; i++){
52     out[i] = in[-i & (n - 1)] - conj(in[i]);
53 }
54 FFT(out);
55
56 for (int i=0 ; i<res.size() ; i++){
57     res[i] = imag(out[i]) / (4 * n);
58 }
59
60 return res;
61 }

```

```

45 while (n<sa+sb-1) n *= 2;
46 a.resize(n);
47 b.resize(n);
48 vector<cd> c(n);
49
50 FFT(a, 0);
51 FFT(b, 0);
52 for (int i=0 ; i<n ; i++) c[i] = a[i]*b[i];
53 FFT(c, 1);
54
55 c.resize(sa+sb-1);
56
57 return c;
58 }
59 }

```

2.2 FFT new

```

1 typedef complex<double> cd;
2
3 // b9c90a
4 void FFT(vector<cd> &a) {
5     int n = a.size(), L = 31-__builtin_clz(n);
6     vector<complex<long double>> R(2, 1);
7     vector<cd> rt(2, 1);
8     for (int k=2 ; k<n ; k*=2){
9         R.resize(n);
10        rt.resize(n);
11        auto x = polar(1.0L, acos(-1.0L) / k);
12        for (int i=k ; i<2*k ; i++){
13            rt[i] = R[i] = (i&1 ? R[i/2]*x : R[i/2]);
14        }
15    }
16
17    vector<int> rev(n);
18    for (int i=0 ; i<n ; i++){
19        rev[i] = (rev[i/2] | (i&1)<<L)/2;
20    }
21    for (int i=0 ; i<n ; i++){
22        if (i<rev[i]) swap(a[i], a[rev[i]]);
23    }
24    for (int k=1 ; k<n ; k*=2){
25        for (int i=0 ; i<n ; i+=2*k){
26            for (int j=0 ; j<k ; j++){
27                auto x = (double *)&rt[j+k];
28                auto y = (double *)&a[i+j+k];
29                cd z(x[0]*y[0] - x[1]*y[1], x[0]*y[1] + x[1]*
30                    y[0]);
31                a[i+j+k] = a[i+j]-z;
32                a[i+j] += z;
33            }
34        }
35    }
36    return;
37 }
38 // 39029d
39 vector<double> PolyMul(const vector<double> a, const vector<
40     double> b){
41     if (a.empty() || b.empty()) return {};
42     vector<double> res(a.size()+b.size()-1);
43     int L = 32 - __builtin_clz(res.size()), n = 1 << L;
44     vector<cd> in(n), out(n);
45     copy(a.begin(), a.end(), begin(in));

```

2.3 FFT old

```

1 typedef complex<double> cd;
2 const double PI = acos(-1);
3
4 void FFT(vector<cd> &a, bool inv){
5
6     int n = a.size();
7
8     for (int i=1, j=0 ; i<n ; i++){
9         int bit = (n>>1);
10        for ( ; j&bit ; bit>>=1){
11            j ^= bit;
12        }
13        j ^= bit;
14        if (i<j){
15            swap(a[i], a[j]);
16        }
17    }
18
19    for (int len=2 ; len<=n ; len<=1){
20        cd wlen = polar(1.0, (inv ? 2 : -2)*PI/len);
21
22        for (int i=0 ; i<n ; i+=len){
23            cd w(1);
24            for (int j=0 ; j<len/2 ; j++){
25                cd u = a[i+j];
26                cd v = a[i+j+len/2]*w;
27                a[i+j] = u+v;
28                a[i+j+len/2] = u-v;
29                w *= wlen;
30            }
31        }
32    }
33
34    if (inv){
35        for (auto &x : a){
36            x /= n;
37        }
38    }
39
40    return;
41 }
42
43 vector<cd> polyMul(vector<cd> a, vector<cd> b){
44     int sa = a.size(), sb = b.size(), n = 1;

```

2.4 NTT mod 998244353

```

1 const int MOD = (119 << 23) + 1, ROOT = 62; // = 998244353
2 // For p < 2^30 there is also e.g. 5 << 25, 7 << 26, 479 <<
3 // 21
4 // and 483 << 21 (same root). The last two are > 10^9.
5 // 9cd58a
6 void NTT(vector<int> &a) {
7     int n = a.size();
8     int L = 31-__builtin_clz(n);
9     vector<int> rt(2, 1);
10    for (int k=2, s=2 ; k<n ; k*=2, s++){
11        rt.resize(n);
12        int z[] = {1, qp(ROOT, MOD>>s)};
13        for (int i=k ; i<2*k ; i++){
14            rt[i] = rt[i/2]*z[i&1]%MOD;
15        }
16    }
17
18    vector<int> rev(n);
19    for (int i=0 ; i<n ; i++){
20        rev[i] = (rev[i/2] | (i&1)<<L)/2;
21    }
22    for (int i=0 ; i<n ; i++){
23        if (i<rev[i]){
24            swap(a[i], a[rev[i]]);
25        }
26    }
27
28    for (int k=1 ; k<n ; k*=2){
29        for (int i=0 ; i<n ; i+=2*k){
30            for (int j=0 ; j<k ; j++){
31                int z = rt[j+k]*a[i+j+k]%MOD, &ai = a[i+j];
32                a[i+j+k] = ai-z+(z>ai ? MOD : 0);
33                ai += (ai+z>MOD ? z-MOD : z);
34            }
35        }
36    }
37 }
38
39 // 0b0e99
40 vector<int> polyMul(vector<int> &a, vector<int> &b){
41     if (a.empty() || b.empty()) return {};
42     int s = a.size()+b.size()-1, B = 32-__builtin_clz(s), n =
43         1<<B;
44     int inv = qp(n, MOD-2);

```

```

44 |
45 | vector<int> L(a), R(b), out(n);
46 | L.resize(n), R.resize(n);
47 | NTT(L), NTT(R);
48 | for (int i=0 ; i<n ; i++){
49 |     out[-i&(n-1)] = L[i]*R[i]%MOD*inv%MOD;
50 | }
51 | NTT(out);
52 |
53 | out.resize(s);
54 | return out;
55 | }

```

3 Data-Structure

3.1 BIT

```

1 | vector<int> BIT(MAX_SIZE);
2 | void update(int pos, int val){
3 |     for (int i=pos ; i<MAX_SIZE ; i+=i&-i){
4 |         BIT[i]+=val;
5 |     }
6 | }
7 |
8 | int query(int pos){
9 |     int ret=0;
10 |    for (int i=pos ; i>0 ; i-=i&-i){
11 |        ret+=BIT[i];
12 |    }
13 |    return ret;
14 | }
15 |
16 | // const int MAX_N = (1<<20)
17 | int k_th(int k){ // 回傳 BIT 中第 k 小的元素 (based-1)
18 |     int res = 0;
19 |     for (int i=MAX_N>>1 ; i>=1 ; i>>=1)
20 |         if (bit[res+i]<k)
21 |             k -= bit[res+i];
22 |     return res+1;
23 | }

```

3.2 Disjoint Set Persistent

```

1 | struct Persistent_Disjoint_Set{
2 |     Persistent_Segment_Tree arr, sz;
3 |
4 |     void init(int n){
5 |         arr.init(n);
6 |         vector<int> v1;
7 |         for (int i=0 ; i<n ; i++){
8 |             v1.push_back(i);
9 |         }
10 |        arr.build(v1, 0);
11 |
12 |        sz.init(n);
13 |        vector<int> v2;
14 |        for (int i=0 ; i<n ; i++){
15 |            v2.push_back(1);

```

```

16 |     }
17 |     sz.build(v2, 0);
18 | }
19 |
20 | int find(int a){
21 |     int res = arr.query_version(a, a+1, arr.version.size()
22 |         (-1).val);
23 |     if (res==a) return a;
24 |     return find(res);
25 | }
26 |
27 | bool unite(int a, int b){
28 |     a = find(a);
29 |     b = find(b);
30 |
31 |     if (a!=b){
32 |
33 |         int sz1 = sz.query_version(a, a+1, arr.version.
34 |             size()-1).val;
35 |         int sz2 = sz.query_version(b, b+1, arr.version.
36 |             size()-1).val;
37 |
38 |         if (sz1<sz2){
39 |             arr.update_version(a, b, arr.version.size()
40 |                 -1);
41 |             sz.update_version(b, sz1+sz2, arr.version.
42 |                 size()-1);
43 |         }else{
44 |             arr.update_version(b, a, arr.version.size()
45 |                 -1);
46 |             sz.update_version(a, sz1+sz2, arr.version.
47 |                 size()-1);
48 |         }
49 |         return true;
50 |     }
51 |     return false;
52 | }
53 | }
54 | }
55 | }

```

3.3 PBDS GP Hash Table

```

1 | #include <ext/pb_ds/assoc_container.hpp>
2 | using namespace __gnu_pbds;
3 | typedef tree<int, null_type, less<int>, rb_tree_tag,
4 |     tree_order_statistics_node_update> order_set;
5 | struct custom_hash {
6 |     static uint64_t splitmix64(uint64_t x) {
7 |         // http://xorshift.di.unimi.it/splitmix64.c
8 |         x += 0x9e3779b97f4a7c15;
9 |         x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
10 |        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
11 |        return x ^ (x >> 31);
12 |    }
13 |
14 |    size_t operator()(uint64_t x) const {
15 |        static const uint64_t FIXED_RANDOM = chrono::
16 |            steady_clock::now().time_since_epoch().count();
17 |        return splitmix64(x + FIXED_RANDOM);
18 |    }
19 | };
20 | gp_hash_table<int, int, custom_hash> ss;

```

3.4 PBDS Order Set

```

1 | /*
2 | .find_by_order(k) 回傳第 k 小的值 (based-0)
3 | .order_of_key(k) 回傳有多少元素比 k 小
4 | 不能在 #define int long long 後 #include 檔案
5 | */
6 |
7 | #include <ext/pb_ds/assoc_container.hpp>
8 | #include <ext/pb_ds/tree_policy.hpp>
9 | using namespace __gnu_pbds;
10 | typedef tree<int, null_type, less<int>, rb_tree_tag,
11 |     tree_order_statistics_node_update> order_set;

```

3.5 Segment Tree Add Set

```

1 | // [ll, rr), based-0
2 | // 使用前記得 init(陣列大小), build(陣列名稱)
3 | // add(ll, rr): 區間修改
4 | // set(ll, rr): 區間賦值
5 | // query(ll, rr): 區間求和 / 求最大值
6 | struct SegmentTree{
7 |     struct node{
8 |         int add_tag = 0;
9 |         int set_tag = 0;
10 |        int sum = 0;
11 |        int ma = 0;
12 |    };
13 |
14 |    vector<node> arr;
15 |
16 |    SegmentTree(int n){
17 |        arr.resize(n<<2);
18 |    }
19 |
20 |    node pull(node A, node B){
21 |        node C;
22 |        C.sum = A.sum+B.sum;
23 |        C.ma = max(A.ma, B.ma);
24 |        return C;
25 |    }
26 |
27 |    // cce0c8
28 |    void push(int idx, int ll, int rr){
29 |        if (arr[idx].set_tag!=0){
30 |            arr[idx].sum = (rr-ll)*arr[idx].set_tag;
31 |            arr[idx].ma = arr[idx].set_tag;
32 |            if (rr-ll>1){
33 |                arr[idx*2+1].add_tag = 0;
34 |                arr[idx*2+1].set_tag = arr[idx].set_tag;
35 |                arr[idx*2+2].add_tag = 0;
36 |                arr[idx*2+2].set_tag = arr[idx].set_tag;
37 |            }
38 |            arr[idx].set_tag = 0;
39 |        }
40 |        if (arr[idx].add_tag!=0){
41 |            arr[idx].sum += (rr-ll)*arr[idx].add_tag;
42 |            arr[idx].ma += arr[idx].add_tag;
43 |            if (rr-ll>1){
44 |                arr[idx*2+1].add_tag += arr[idx].add_tag;
45 |                arr[idx*2+2].add_tag += arr[idx].add_tag;

```

```

46     }
47     arr[idx].add_tag = 0;
48 }
49 }
50
51 void build(vector<int> &v, int idx = 0, int ll = 0, int
52         rr = n){
53     if (rr-ll==1){
54         arr[idx].sum = v[ll];
55         arr[idx].ma = v[ll];
56     }else{
57         int mid = (ll+rr)/2;
58         build(v, idx*2+1, ll, mid);
59         build(v, idx*2+2, mid, rr);
60         arr[idx] = pull(arr[idx*2+1], arr[idx*2+2]);
61     }
62 }
63
64 void add(int ql, int qr, int val, int idx = 0, int ll =
65         0, int rr = n){
66     push(idx, ll, rr);
67     if (rr<=ql || qr<=ll) return;
68     if (ql<=ll && rr<=qr){
69         arr[idx].add_tag += val;
70         push(idx, ll, rr);
71         return;
72     }
73     int mid = (ll+rr)/2;
74     add(ql, qr, val, idx*2+1, ll, mid);
75     add(ql, qr, val, idx*2+2, mid, rr);
76     arr[idx]=pull(arr[idx*2+1], arr[idx*2+2]);
77 }
78
79 void set(int ql, int qr, int val, int idx=0, int ll=0,
80         int rr=n){
81     push(idx, ll, rr);
82     if (rr<=ql || qr<=ll) return;
83     if (ql<=ll && rr<=qr){
84         arr[idx].add_tag = 0;
85         arr[idx].set_tag = val;
86         push(idx, ll, rr);
87         return;
88     }
89     int mid = (ll+rr)/2;
90     set(ql, qr, val, idx*2+1, ll, mid);
91     set(ql, qr, val, idx*2+2, mid, rr);
92     arr[idx] = pull(arr[idx*2+1], arr[idx*2+2]);
93 }
94
95 node query(int ql, int qr, int idx = 0, int ll = 0, int
96         rr = n){
97     push(idx, ll, rr);
98     if (rr<=ql || qr<=ll) return node();
99     if (ql<=ll && rr<=qr) return arr[idx];
100 }

```

3.6 Segment Tree Li Chao Line

```

1  /*
2  全部都是 0-based
3
4  宣告
5  LC_Segment_Tree st(n);
6
7  函式：
8  update({a, b})：插入一條  $y=ax+b$  的全域直線
9  query(x)：查詢所有直線在位置 x 的最小值
10 */
11 const int MAX_V = 1e6+10; // 值域最大值
12
13 struct LC_Segment_Tree{
14     struct Node{ //  $y = ax+b$ 
15         int a = 0;
16         int b = INF;
17
18         int y(int x){
19             return a*x+b;
20         }
21     };
22     vector<Node> arr;
23
24     LC_Segment_Tree(int n = 0){
25         arr.resize(4*n);
26     }
27
28     void update(Node val, int idx = 0, int ll = 0, int rr =
29         MAX_V){
30         if (rr-ll==0) return;
31         if (rr-ll==1){
32             if (val.y(ll)<arr[idx].y(ll)){
33                 arr[idx] = val;
34             }
35             return;
36         }
37         int mid = (ll+rr)/2;
38         if (arr[idx].a > val.a) swap(arr[idx], val); // 原本
39         // 的線斜率要比較小
40         if (arr[idx].y(mid) < val.y(mid)){ // 交點在左邊
41             update(val, idx*2+1, ll, mid);
42         }else{ // 交點在右邊
43             swap(arr[idx], val); // 在左子樹中，新線比舊線還
44             // 要好
45             update(val, idx*2+2, mid, rr);
46         }
47         return;
48     }
49
50     int query(int x, int idx = 0, int ll = 0, int rr = MAX_V)
51     {
52         if (rr-ll==0) return INF;
53         if (rr-ll==1){
54             return arr[idx].y(ll);
55         }
56         int mid = (ll+rr)/2;
57         if (x<mid){
58             return min(arr[idx].y(x), query(x, idx*2+1, ll,
59                 mid));
60         }else{
61             return min(arr[idx].y(x), query(x, idx*2+2, mid,
62                 rr));
63         }
64     }
65 }

```

```

59     }
60 }
61 };
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

3.7 Segment Tree Li Chao Segment

```

1  /*
2  全部都是 0-based
3
4  宣告
5  LC_Segment_Tree st(n);
6
7  函式：
8  update_segment({a, b}, ql, qr)：在 [ql, qr) 插入一條  $y=ax+b$ 
9  的線段
10 query(x)：查詢所有直線在位置 x 的最小值
11 */
12 const int MAX_V = 1e6+10; // 值域最大值
13
14 struct LC_Segment_Tree{
15     struct Node{ //  $y = ax+b$ 
16         int a = 0;
17         int b = INF;
18
19         int y(int x){
20             return a*x+b;
21         }
22     };
23     vector<Node> arr;
24
25     LC_Segment_Tree(int n = 0){
26         arr.resize(4*n);
27     }
28
29     void update(Node val, int idx = 0, int ll = 0, int rr =
30         MAX_V){
31         if (rr-ll==0) return;
32         if (rr-ll==1){
33             if (val.y(ll)<arr[idx].y(ll)){
34                 arr[idx] = val;
35             }
36             return;
37         }
38         int mid = (ll+rr)/2;
39         if (arr[idx].a > val.a) swap(arr[idx], val); // 原本
40         // 的線斜率要比較小
41         if (arr[idx].y(mid) < val.y(mid)){ // 交點在左邊
42             update(val, idx*2+1, ll, mid);
43         }else{ // 交點在右邊
44             swap(arr[idx], val); // 在左子樹中，新線比舊線還
45             // 要好
46             update(val, idx*2+2, mid, rr);
47         }
48         return;
49     }
50
51     // 在 [ql, qr) 加上一條 val 的線段
52     void update_segment(Node val, int ql, int qr, int idx =
53         0, int ll = 0, int rr = MAX_V){
54         if (rr-ll==0) return;
55     }
56 }

```

```

51     if (rr<=ql || qr<=ll) return;
52     if (ql<=ll && rr<=qr){
53         update(val, idx, ll, rr);
54         return;
55     }
56
57     int mid = (ll+rr)/2;
58     update_segment(val, ql, qr, idx*2+1, ll, mid);
59     update_segment(val, ql, qr, idx*2+2, mid, rr);
60     return;
61 }
62
63 int query(int x, int idx = 0, int ll = 0, int rr = MAX_V)
64 {
65     if (rr-ll==0) return INF;
66     if (rr-ll==1){
67         return arr[idx].y(ll);
68     }
69
70     int mid = (ll+rr)/2;
71     if (x<mid){
72         return min(arr[idx].y(x), query(x, idx*2+1, ll,
73         mid));
74     }else{
75         return min(arr[idx].y(x), query(x, idx*2+2, mid,
76         rr));
77     }
78 }
79 };

```

3.8 Segment Tree Persistent

```

1  /*
2  全部都是 0-based
3
4  宣告
5  Persistent_Segment_Tree st(n+q);
6  st.build(v, 0);
7
8  函式：
9  update_version(pos, val, ver)：對版本 ver 的 pos 位置改成 val
10 query_version(ql, qr, ver)：對版本 ver 查詢 [ql, qr) 的區間和
11 clone_version(ver)：複製版本 ver 到最新的版本
12 */
13 struct Persistent_Segment_Tree{
14     int node_cnt = 0;
15     struct Node{
16         int lc = -1;
17         int rc = -1;
18         int val = 0;
19     };
20     vector<Node> arr;
21     vector<int> version;
22
23     Persistent_Segment_Tree(int sz){
24         arr.resize(32*sz);
25         version.push_back(node_cnt++);
26         return;
27     }
28
29     void pull(Node &c, Node a, Node b){
30         c.val = a.val+b.val;

```

```

31     return;
32 }
33
34 void build(vector<int> &v, int idx, int ll = 0, int rr =
35     n){
36     auto &now = arr[idx];
37
38     if (rr-ll==1){
39         now.val = v[ll];
40         return;
41     }
42
43     int mid = (ll+rr)/2;
44     now.lc = node_cnt++;
45     now.rc = node_cnt++;
46     build(v, now.lc, ll, mid);
47     build(v, now.rc, mid, rr);
48     pull(now, arr[now.lc], arr[now.rc]);
49     return;
50 }
51
52 void update(int pos, int val, int idx, int ll = 0, int rr
53     = n){
54     auto &now = arr[idx];
55
56     if (rr-ll==1){
57         now.val = val;
58         return;
59     }
60
61     int mid = (ll+rr)/2;
62     if (pos<mid){
63         arr[node_cnt] = arr[now.lc];
64         now.lc = node_cnt;
65         node_cnt++;
66         update(pos, val, now.lc, ll, mid);
67     }else{
68         arr[node_cnt] = arr[now.rc];
69         now.rc = node_cnt;
70         node_cnt++;
71         update(pos, val, now.rc, mid, rr);
72     }
73     pull(now, arr[now.lc], arr[now.rc]);
74     return;
75 }
76
77 void update_version(int pos, int val, int ver){
78     update(pos, val, version[ver]);
79 }
80
81 Node query(int ql, int qr, int idx, int ll = 0, int rr =
82     n){
83     auto &now = arr[idx];
84
85     if (ql<=ll && rr<=qr) return now;
86     if (rr<=ql || qr<=ll) return Node();
87
88     int mid = (ll+rr)/2;
89
90     Node ret;
91     pull(ret, query(ql, qr, now.lc, ll, mid), query(ql,
92     qr, now.rc, mid, rr));
93     return ret;
94 }
95
96 Node query_version(int ql, int qr, int ver){

```

```

93     return query(ql, qr, version[ver]);
94 }
95
96 void clone_version(int ver){
97     version.push_back(node_cnt);
98     arr[node_cnt] = arr[version[ver]];
99     node_cnt++;
100 }
101 };

```

3.9 Sparse Table

```

1 struct SparseTable{
2     vector<vector<int>> st;
3     void build(vector<int> v){
4         int h = __lg(v.size());
5         st.resize(h+1);
6         st[0] = v;
7
8         for (int i=1; i<=h; i++){
9             int gap = (1<<(i-1));
10            for (int j=0; j+gap<st[i-1].size(); j++){
11                st[i].push_back(min(st[i-1][j], st[i-1][j+gap]
12                ));
13            }
14        }
15
16        // 回傳 [LL, rr) 的最小值
17        int query(int ll, int rr){
18            int h = __lg(rr-ll);
19            return min(st[h][ll], st[h][rr-(1<<h)]);
20        }
21    };

```

3.10 Treap

```

1 struct Treap{
2     Treap *l = nullptr, *r = nullptr;
3     int pri = rand(), val = 0, sz = 1;
4
5     Treap(int _val){
6         val = _val;
7     }
8 };
9
10 int size(Treap *t){return t ? t->sz : 0;}
11 void pull(Treap *t){
12     t->sz = size(t->l)+size(t->r)+1;
13 }
14
15 Treap* merge(Treap *a, Treap *b){
16     if (!a || !b) return a ? a : b;
17
18     if (a->pri>b->pri){
19         a->r = merge(a->r, b);
20         pull(a);
21         return a;
22     }else{

```



```

23     b->l = merge(a, b->l);
24     pull(b);
25     return b;
26 }
27 }
28 }
29 }
30 pair<Treap*, Treap*> split(Treap *t, int k){ // 1-based <前
    k 個元素, 其他元素>
31     if (!t) return {};
32     if (size(t->l)>=k){
33         auto pa = split(t->l, k);
34         t->l = pa.second;
35         pull(t);
36         return {pa.first, t};
37     }else{
38         auto pa = split(t->r, k-size(t->l)-1);
39         t->r = pa.first;
40         pull(t);
41         return {t, pa.second};
42     }
43 }
44 // functions
45 Treap* build(vector<int> v){
46     Treap* ret = nullptr;
47     for (int i=0 ; i<v.size() ; i++){
48         ret = merge(ret, new Treap(v[i]));
49     }
50     return ret;
51 }
52 }
53 }
54 array<Treap*, 3> cut(Treap *t, int l, int r){ // 1-based <前
    1~l-1 個元素, l~r 個元素, r+1 個元素>
55     array<Treap*, 3> ret;
56     tie(ret[1], ret[2]) = split(t, r);
57     tie(ret[0], ret[1]) = split(ret[1], l-1);
58     return ret;
59 }
60 }
61 void print(Treap *t, bool flag = true){
62     if (t->l!=0) print(t->l, false);
63     cout << t->val;
64     if (t->r!=0) print(t->r, false);
65     if (flag) cout << endl;
66 }

```

3.11 Trie

```

1 struct Trie{
2     struct Data{
3         int nxt[2]={0, 0};
4     };
5
6     int sz=0;
7     vector<Data> arr;
8
9     void init(int n){
10         arr.resize(n);
11     }
12
13     void insert(int n){

```

```

14     int now=0;
15     for (int i=N ; i>=0 ; i--){
16         int v=(n>>i)&1;
17         if (!arr[now].nxt[v]){
18             arr[now].nxt[v]=++sz;
19         }
20         now=arr[now].nxt[v];
21     }
22 }
23
24 int query(int n){
25     int now=0, ret=0;
26     for (int i=N ; i>=0 ; i--){
27         int v=(n>>i)&1;
28         if (arr[now].nxt[1-v]){
29             ret+=(1<<i);
30             now=arr[now].nxt[1-v];
31         }else if (arr[now].nxt[v]){
32             now=arr[now].nxt[v];
33         }else{
34             return ret;
35         }
36     }
37     return ret;
38 }
39 }
40 } tr;

```

4 Dynamic-Programming

4.1 Digit DP

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 long long l, r;
5 long long dp[20][10][2][2]; // dp[pos][pre][limit] = 後 pos
    位 · pos 前一位是 pre · (是/否) 有上界 · (是/否) 有前綴零
    的答案數量
6
7 long long memorize_search(string &s, int pos, int pre, bool
    limit, bool lead){
8
9     // 已經被找過了 · 直接回傳值
10    if (dp[pos][pre][limit][lead]!=-1) return dp[pos][pre][
    limit][lead];
11
12    // 已經搜尋完畢 · 紀錄答案並回傳
13    if (pos==(int)s.size()){
14        return dp[pos][pre][limit][lead] = 1;
15    }
16
17    // 枚舉目前的位數數字是多少
18    long long ans = 0;
19    for (int now=0 ; now<=(limit ? s[pos]-'0' : 9) ; now++){
20        if (now==pre){
21
22            // 1~9 絕對不能連續出現
23            if (pre!=0) continue;
24

```

```

25        // 如果已經不在前綴零的範圍內 · 0 不能連續出現
26        if (lead==false) continue;
27    }
28
29    ans += memorize_search(s, pos+1, now, limit&(now==(s[
    pos]-'0')), lead&(now==0));
30 }
31
32 // 已經搜尋完畢 · 紀錄答案並回傳
33 return dp[pos][pre][limit][lead] = ans;
34 }
35
36 // 回傳 [0, n] 有多少數字符合條件
37 long long find_answer(long long n){
38     memset(dp, -1, sizeof(dp));
39     string tmp = to_string(n);
40
41     return memorize_search(tmp, 0, 0, true, true);
42 }
43
44 int main(){
45
46     // input
47     cin >> l >> r;
48
49     // output - 計算 [l, r] 有多少數字任意兩個位數都不相同
50     cout << find_answer(r)-find_answer(l-1) << "\n";
51
52     return 0;
53 }

```

4.2 SOS DP

```

1 // 總時間複雜度為 O(n 2^n)
2 // 計算 dp[i] = i 所有 bit mask 子集的和
3 for (int i=0 ; i<n ; i++){
4     for (int mask=0 ; mask<(1<<n) ; mask++){
5         if ((mask>>i)&1){
6             dp[mask] += dp[mask^(1<<i)];
7         }
8     }
9 }

```

4.3 Integer Partition

$dp[i][x]$ = 要將整數 x 拆成 i 堆的「組合數」

$dp[i+1][x+1] += dp[i][x]$ (創造新的一堆)
 $dp[i][x+i] += dp[i][x]$ (把每一堆都增加 1)

5 Geometry

5.1 Geometry Struct


```

1 // 判斷數值正負：{1:正數,0:零,-1:負數}
2 int sign(long long x) {return (x >= 0) ? ((bool)x) : -1; }
3 int sign(double x) {
4     return (abs(x) < 1e-9) ? 0 : (x > 0 ? 1 : -1);
5 }
6
7 template<typename T>
8 struct point {
9     T x, y;
10    point() {}
11    point(const T &x, const T &y) : x(x), y(y) {}
12
13    point operator+(point b) {return {x+b.x, y+b.y}; }
14    point operator-(point b) {return {x-b.x, y-b.y}; }
15    point operator*(T b) {return {x*b, y*b}; }
16    point operator/(T b) {return {x/b, y/b}; }
17    bool operator==(point b) {return x==b.x && y==b.y; }
18    // 逆時針極角排序
19    bool operator<(point &b) {return (x*b.y > b.x*y); }
20    friend ostream& operator<<(ostream& os, point p) {
21        os << "(" << p.x << ", " << p.y << ")";
22        return os;
23    }
24    // 判斷 ab 到 ac 的方向：{1:逆時鐘,0:重疊,-1:順時鐘}
25    friend int ori(point a, point b, point c) {
26        return sign((b-a)^(c-a));
27    }
28    friend bool btw(point a, point b, point c) {
29        return ori(a, b, c) == 0 && sign((a-c)*(b-c)) <= 0;
30    }
31    // 判斷線段 ab, cd 是否相交
32    friend bool banana(point a, point b, point c, point d) {
33        int s1 = ori(a, b, c);
34        int s2 = ori(a, b, d);
35        int s3 = ori(c, d, a);
36        int s4 = ori(c, d, b);
37        if (btw(a, b, c) || btw(a, b, d) || btw(c, d, a) ||
38            btw(c, d, b)) return 1;
39        return (s1 * s2 < 0) && (s3 * s4 < 0);
40    }
41    T operator*(point b) {return x * b.x + y * b.y; }
42    T operator^(point b) {return x * b.y - y * b.x; }
43    T abs2() {return (*this) * (*this); }
44
45    // 旋轉 Arg(b) 的角度 (小心溢位)
46    point rotate(point b) {return {x*b.x - y*b.y, x*b.y + y*b.x}; }
47 };
48
49 template<typename T>
50 struct line {
51     point<T> p1, p2;
52     // ax + by + c = 0
53     T a, b, c; // |a|, |b| ≤ 2C, |c| ≤ 8C^2
54     line() {}
55     line(const point<T> &x, const point<T> &y) : p1(x), p2(y) {
56         build();
57     }
58     void build() {
59         a = p1.y - p2.y;
60         b = p2.x - p1.x;
61         c = (-a*p1.x) - b*p1.y;
62     }
63     // 判斷點和有向直線的關係：{1:左邊,0:在線上,-1:右邊}
64
65     int ori(point<T> &p) {
66         return sign((p2-p1) ^ (p-p1));
67     }
68     // 判斷直線斜率是否相同
69     bool parallel(line &l) {
70         return ((p1-p2) ^ (l.p1-l.p2)) == 0;
71     }
72     // 兩直線交點
73     point<long double> line_intersection(line &l) {
74         using P = point<long double>;
75         point<T> a = p2-p1, b = l.p2-l.p1, s = l.p1-p1;
76         return P(p1.x, p1.y) + P(a.x, a.y) * (((long double)(s^b)) / (a^b));
77     };
78
79     template<typename T>
80     struct polygon {
81         vector<point<T>> v;
82         polygon() {}
83         polygon(const vector<point<T>> &u) : v(u) {}
84         // simple 為 true 的時候會回傳任意三點不共線的凸包
85         void make_convex_hull(int simple) {
86             auto cmp = [&](point<T> &p, point<T> &q) {
87                 return (p.x == q.x) ? (p.y < q.y) : (p.x < q.x);
88             };
89             simple = (bool)simple;
90             sort(v.begin(), v.end(), cmp);
91             v.resize(unique(v.begin(), v.end()) - v.begin());
92             vector<point<T>> hull;
93             for (int t = 0; t < 2; ++t) {
94                 int sz = hull.size();
95                 for (auto &i:v) {
96                     while (hull.size() >= sz+2 && ori(hull[hull.size()-2], hull.back(), i) < simple) {
97                         hull.pop_back();
98                     }
99                     hull.push_back(i);
100                 }
101                 hull.pop_back();
102                 reverse(v.begin(), v.end());
103             }
104             swap(hull, v);
105         }
106     };
107     // 可以在有 n 個點的簡單多邊形內，用 O(n) 判斷一個點：
108     // {1: 在多邊形內, 0: 在多邊形上, -1: 在多邊形外}
109     int in_polygon(point<T> a) {
110         const T MAX_POS = 1e9 + 5; // [記得修改] 座標的最大值
111         point<T> pre = v.back(), b(MAX_POS, a.y + 1);
112         int cnt = 0;
113
114         for (auto &i:v) {
115             if (btw(pre, i, a)) return 0;
116             if (banana(a, b, pre, i)) cnt++;
117             pre = i;
118         }
119
120         return cnt%2 ? 1 : -1;
121     }
122     /// 警告：以下所有凸包專用的函式都只接受逆時針排序且任三點不共線的凸包 ///
123     // 可以在有 n 個點的凸包內，用 O(Log n) 判斷一個點：
124     // {1: 在凸包內, 0: 在凸包邊上, -1: 在凸包外}
125     int in_convex(point<T> p) {
126
127         int n = v.size();
128         int a = ori(v[0], v[1], p), b = ori(v[0], v[n-1], p);
129         if (a < 0 || b > 0) return -1;
130         if (btw(v[0], v[1], p)) return 0;
131         if (btw(v[0], v[n-1], p)) return 0;
132         int l = 1, r = n - 1, mid;
133         while (l + 1 < r) {
134             mid = (l + r) >> 1;
135             if (ori(v[0], v[mid], p) >= 0) l = mid;
136             else r = mid;
137         }
138         int k = ori(v[l], v[r], p);
139         if (k <= 0) return k;
140         return 1;
141     }
142     // 凸包專用的環狀二分搜，回傳 0-based index
143     int cycle_search(auto &f) {
144         int n = v.size(), l = 0, r = n;
145         bool rv = f(1, 0);
146         while (r - l > 1) {
147             int m = (l + r) / 2;
148             if (f(0, m) ? rv : f(m, (m + 1) % n)) r = m;
149             else l = m;
150         }
151         return f(1, r % n) ? l : r % n;
152     }
153     // 可以在有 n 個點的凸包內，用 O(Log n) 判斷一條直線：
154     // {1: 穿過凸包, 0: 剛好切過凸包, -1: 沒碰到凸包}
155     int line_cut_convex(line<T> L) {
156         point<T> p(L.a, L.b); // 記得 L 要 build
157         auto gt = [&](int neg) {
158             auto f = [&](int x, int y) {
159                 return sign((v[x] - v[y]) * p) == neg;
160             };
161             return -(v[cycle_search(f)] * p);
162         };
163         T x = gt(1), y = gt(-1);
164         if (L.c < x || y < L.c) return -1;
165         return not (L.c == x || L.c == y);
166     }
167     // 可以在有 n 個點的凸包內，用 O(Log n) 判斷一個線段：
168     // {1: 存在一個凸包上的邊可以把這個線段切成兩半,
169     // 0: 有碰到凸包但沒有任何凸包上的邊可以把它切成兩半,
170     // -1: 沒碰到凸包}
171     /// 除非線段兩端點都不在凸包邊上，否則此函數回傳 0 的時候不一定表示線段沒有通過凸包內部 ///
172     int segment_across_convex(line<T> L) {
173         point<T> p(L.a, L.b); // 記得 L 要 build
174         auto gt = [&](int neg) {
175             auto f = [&](int x, int y) {
176                 return sign((v[x] - v[y]) * p) == neg;
177             };
178             return cycle_search(f);
179         };
180         int i = gt(1), j = gt(-1), n = v.size();
181         T x = -(v[i] * p), y = -(v[j] * p);
182         if (L.c < x || y < L.c) return -1;
183         if (L.c == x || L.c == y) return 0;
184
185         if (i > j) swap(i, j);
186         auto g = [&](int x, int lim) {
187             int now = 0, nxt;
188             for (int i = 1 << __lg(lim); i > 0; i /= 2) {
189                 if (now + i > lim) continue;
190             }
191         };
192     }

```

```

188     nzt = (x + i) % n;
189     if (L.ori(v[x]) * L.ori(v[nzt]) >= 0) {
190         x = nzt;
191         now += i;
192     }
193     } // ↓ BE CAREFUL
194     return -(ori(v[x], v[(x + 1) % n], L.p1) * ori(v[
195         x], v[(x + 1) % n], L.p2));
196 };
197 return max(g(i, j - i), g(j, n - (j - i)));
198 }
199 // 可以在有 n 個點的凸包內，用 O(Log n) 判斷一個線段：
200 // {1 : 線段上存在某一端位於凸包內部（邊上不算），
201 // 0 : 線段上存在某一端碰到凸包的邊但線段上任一點均不在凸包
202 // 內部，
203 // -1 : 線段完全在凸包外面}
204 int segment_pass_convex_interior(line<T> L) {
205     if (in_convex(L.p1) == 1 || in_convex(L.p2) == 1)
206         return 1;
207     point<T> p(L.a, L.b); // 記得 L 要 build
208     auto gt = [&](int neg) {
209         auto f = [&](int x, int y) {
210             return sign((v[x] - v[y]) * p) == neg;
211         };
212         return cycle_search(f);
213     };
214     int i = gt(1), j = gt(-1), n = v.size();
215     T x = -(v[i] * p), y = -(v[j] * p);
216     if (L.c < x || y < L.c) return -1;
217     if (L.c == x || L.c == y) return 0;
218     if (i > j) swap(i, j);
219     auto g = [&](int x, int lim) {
220         int now = 0, nzt;
221         for (int i = 1 <= __lg(lim); i > 0; i /= 2) {
222             if (now + i > lim) continue;
223             nzt = (x + i) % n;
224             if (L.ori(v[x]) * L.ori(v[nzt]) > 0) {
225                 x = nzt;
226                 now += i;
227             }
228         } // ↓ BE CAREFUL
229         return -(ori(v[x], v[(x + 1) % n], L.p1) * ori(v[
230             x], v[(x + 1) % n], L.p2));
231     };
232     int ret = max(g(i, j - i), g(j, n - (j - i)));
233     return (ret == 0) ? (in_convex(L.p1) == 0 &&
234         in_convex(L.p2) == 0) : ret;
235 }
236 // 回傳點過凸包的兩條切線的切點的 0-based index (不保證兩條
237 // 切線的順逆時針關係)
238 pair<int, int> convex_tangent_point(point<T> p) {
239     int n = v.size(), z = -1, edg = -1;
240     auto gt = [&](int neg) {
241         auto check = [&](int x) {
242             if (v[x] == p) z = x;
243             if (btw(v[x], v[(x + 1) % n], p)) edg = x;
244             if (btw(v[(x + n - 1) % n], v[x], p)) edg =
245                 x + n - 1 % n;
246         };
247         auto f = [&](int x, int y) {
248             check(x); check(y);
249             return ori(p, v[x], v[y]) == neg;
250         };
251         return cycle_search(f);
252     };
253     int i = gt(1), j = gt(-1), n = v.size();
254     T x = -(v[i] * p), y = -(v[j] * p);
255     if (L.c < x || y < L.c) return -1;
256     if (L.c == x || L.c == y) return 0;
257     if (i > j) swap(i, j);
258     auto g = [&](int x, int lim) {
259         int now = 0, nzt;
260         for (int i = 1 <= __lg(lim); i > 0; i /= 2) {
261             if (now + i > lim) continue;
262             nzt = (x + i) % n;
263             if (L.ori(v[x]) * L.ori(v[nzt]) > 0) {
264                 x = nzt;
265                 now += i;
266             }
267         } // ↓ BE CAREFUL
268         return -(ori(v[x], v[(x + 1) % n], L.p1) * ori(v[
269             x], v[(x + 1) % n], L.p2));
270     };
271     int ret = max(g(i, j - i), g(j, n - (j - i)));
272     return (ret == 0) ? (in_convex(L.p1) == 0 &&
273         in_convex(L.p2) == 0) : ret;
274 }
275 // 回傳點過凸包的兩條切線的切點的 0-based index (不保證兩條
276 // 切線的順逆時針關係)
277 pair<int, int> convex_tangent_point(point<T> p) {
278     int n = v.size(), z = -1, edg = -1;
279     auto gt = [&](int neg) {
280         auto check = [&](int x) {
281             if (v[x] == p) z = x;
282             if (btw(v[x], v[(x + 1) % n], p)) edg = x;
283             if (btw(v[(x + n - 1) % n], v[x], p)) edg =
284                 x + n - 1 % n;
285         };
286         auto f = [&](int x, int y) {
287             check(x); check(y);
288             return ori(p, v[x], v[y]) == neg;
289         };
290         return cycle_search(f);
291     };
292     int i = gt(1), j = gt(-1), n = v.size();
293     T x = -(v[i] * p), y = -(v[j] * p);
294     if (L.c < x || y < L.c) return -1;
295     if (L.c == x || L.c == y) return 0;
296     if (i > j) swap(i, j);
297     auto g = [&](int x, int lim) {
298         int now = 0, nzt;
299         for (int i = 1 <= __lg(lim); i > 0; i /= 2) {
300             if (now + i > lim) continue;
301             nzt = (x + i) % n;
302             if (L.ori(v[x]) * L.ori(v[nzt]) > 0) {
303                 x = nzt;
304                 now += i;
305             }
306         } // ↓ BE CAREFUL
307         return -(ori(v[x], v[(x + 1) % n], L.p1) * ori(v[
308             x], v[(x + 1) % n], L.p2));
309     };
310     int ret = max(g(i, j - i), g(j, n - (j - i)));
311     return (ret == 0) ? (in_convex(L.p1) == 0 &&
312         in_convex(L.p2) == 0) : ret;
313 }

```

```

245     return cycle_search(f);
246 };
247 int x = gt(1), y = gt(-1);
248 if (z != -1) {
249     return {(z + n - 1) % n, (z + 1) % n};
250 }
251 else if (edg != -1) {
252     return {edg, (edg + 1) % n};
253 }
254 else {
255     return {x, y};
256 }
257 }
258 friend int halfplane_intersection(vector<line<T>> &s,
259     polygon<T> &p) {
260     #define neg(p) ((p.y == 0 ? p.x : p.y) < 0)
261     auto angle_cmp = [&](line<T> &A, line<T> &B) {
262         point<T> a = A.p2-A.p1, b = B.p2-B.p1;
263         return neg(a) < neg(b) || (neg(a) == neg(b) && (a
264             ^ b) > 0);
265     };
266     #undef neg
267     sort(s.begin(), s.end(), angle_cmp); // 線段左側為該
268     // 線段半平面
269     int L, R, n = s.size();
270     vector<point<T>> px(n);
271     vector<line<T>> q(n);
272     q[L = R = 0] = s[0];
273     for (int i = 1; i < n; ++i) {
274         while (L < R && s[i].ori(px[R-1]) <= 0) --R;
275         while (L < R && s[i].ori(px[L]) <= 0) ++L;
276         q[++R] = s[i];
277         if (q[R].parallel(q[R-1])) {
278             --R;
279             if (q[R].ori(s[i].p1) > 0) q[R] = s[i];
280         }
281         if (L < R) px[R-1] = q[R-1].line_intersection(q[R]
282             );
283     }
284     while (L < R && q[L].ori(px[R-1]) <= 0) --R;
285     P.v.clear();
286     if (R - L <= 1) return 0;
287     px[R] = q[R].line_intersection(q[L]);
288     for (int i = L; i <= R; ++i) P.v.push_back(px[i]);
289     return R - L + 1;
290 }
291 }

```

5.2 Geometry 卦長

```

1 const double PI=atan2(0.0,-1.0);
2 template<typename T>
3 struct point{
4     T x,y;
5     point(){}
6     point(const T&x,const T&y):x(x),y(y){}
7     point operator+(const point &b)const{
8         return point(x+b.x,y+b.y);
9     }
10    point operator-(const point &b)const{
11        return point(x-b.x,y-b.y);
12    }
13    point operator*(const T &b)const{
14        return point(x*b,y*b);
15    }

```

```

13    point operator/(const T &b)const{
14        return point(x/b,y/b);
15    }
16    bool operator==(const point &b)const{
17        return x==b.x&&y==b.y;
18    }
19    T dot(const point &b)const{
20        return x*b.x+y*b.y;
21    }
22    T cross(const point &b)const{
23        return x*b.y-y*b.x;
24    }
25    point normal()const{//求法向量
26        return point(-y,x);
27    }
28    T abs2()const{//向量長度的平方
29        return dot(*this);
30    }
31    T rad(const point &b)const{//兩向量的弧度
32        return fabs(atan2(fabs(cross(b)),dot(b)));
33    }
34    T getA()const{//對x軸的弧度
35        T A=atan2(y,x);
36        if(A<=-PI/2)A+=PI*2;
37        return A;
38    }
39    };
40    template<typename T>
41    struct line{
42        line(){}
43        point<T> p1,p2;
44        T a,b,c;//ax+by+c=0
45        line(const point<T>&x,const point<T>&y):p1(x),p2(y){}
46        void pton()const{//轉成一般式
47            a=p1.y-p2.y;
48            b=p2.x-p1.x;
49            c=-a*p1.x-b*p1.y;
50        }
51        T ori(const point<T> &p)const{//點和有向直線的關係，>0左
52            //邊，=0在線上<0右邊
53            return (p2-p1).cross(p-p1);
54        }
55        T btw(const point<T> &p)const{//點投影落在線段上<=0
56            return (p1-p).dot(p2-p);
57        }
58        bool point_on_segment(const point<T>&p)const{//點是否在線段
59            //上
60            return ori(p)==0&&btw(p)<=0;
61        }
62        T dis2(const point<T> &p,bool is_segment=0)const{//點跟直線
63            //線段的距離平方
64            point<T> v=p2-p1,v1=p-p1;
65            if(is_segment){
66                point<T> v2=p-p2;
67                if(v.dot(v1)<=0)return v1.abs2();
68                if(v.dot(v2)>=0)return v2.abs2();
69            }
70            T tmp=v.cross(v1);
71            return tmp*tmp/v.abs2();
72        }
73        T seg_dis2(const line<T> &l)const{//兩線段距離平方
74            return min({dis2(l.p1,1),dis2(l.p2,1),l.dis2(p1,1),l.dis2
75                (p2,1)});
76        }
77        point<T> projection(const point<T> &p)const{//點對直線的投
78            //影
79            point<T> n=(p2-p1).normal();
80            return p-n*(p-p1).dot(n)/n.abs2();
81        }
82        point<T> mirror(const point<T> &p)const{

```

```

71 //點對直線的鏡射，要先呼叫 $pton$ 轉成一般式
72 point<T> R;
73 T d=a*a+b*b;
74 R.x=(b*b*p.x-a*a*p.x-2*a*b*p.y-2*a*c)/d;
75 R.y=(a*a*p.y-b*b*p.y-2*a*b*p.x-2*b*c)/d;
76 return R;
77 }
78 bool equal(const line &l) const { //直線相等
79     return ori(l.p1)==0&&ori(l.p2)==0;
80 }
81 bool parallel(const line &l) const {
82     return (p1-p2).cross(l.p1-l.p2)==0;
83 }
84 bool cross_seg(const line &l) const {
85     return (p2-p1).cross(l.p1-p1)*(p2-p1).cross(l.p2-p1)<=0;
86     //直線是否交線段
87 }
88 int line_intersect(const line &l) const { //直線相交情況，-1無
89     //限多點、1交於一點、0不相交
90     return parallel(l)?(ori(l.p1)==0?-1:0):1;
91 }
92 int seg_intersect(const line &l) const {
93     T c1=ori(l.p1), c2=ori(l.p2);
94     T c3=l.ori(p1), c4=l.ori(p2);
95     if(c1==0&&c2==0){ //共線
96         bool b1=btw(l.p1)>=0, b2=btw(l.p2)>=0;
97         T a3=l.btw(p1), a4=l.btw(p2);
98         if(b1&&b2&&a3==0&&a4>=0) return 2;
99         if(b1&&b2&&a3>=0&&a4==0) return 3;
100         if(b1&&b2&&a3>=0&&a4>=0) return 0;
101         return -1; //無限交點
102     } else if(c1*c2<=0&&c3*c4<=0) return 1;
103     return 0; //不相交
104 }
105 point<T> line_intersection(const line &l) const { //直線交點
106     point<T> a=p2-p1, b=l.p2-l.p1, s=l.p1-p1;
107     //if(a.cross(b)==0) return INF;
108     return p1+a*(s.cross(b)/a.cross(b));
109 }
110 point<T> seg_intersection(const line &l) const { //線段交點
111     int res=seg_intersect(l);
112     if(res<=0) assert(0);
113     if(res==2) return p1;
114     if(res==3) return p2;
115     return line_intersection(l);
116 }
117 };
118 template<typename T>
119 struct polygon {
120     polygon() {}
121     vector<point<T>> p; //逆時針順序
122     T area() const { //面積
123         T ans=0;
124         for(int i=p.size()-1, j=0; j<(int)p.size(); i=j++)
125             ans+=p[i].cross(p[j]);
126         return ans/2;
127     }
128     point<T> center_of_mass() const { //重心
129         T cx=0, cy=0, w=0;
130         for(int i=p.size()-1, j=0; j<(int)p.size(); i=j++){
131             T a=p[i].cross(p[j]);
132             cx+=(p[i].x+p[j].x)*a;
133             cy+=(p[i].y+p[j].y)*a;
134             w+=a;
135         }
136         return point<T>(cx/3/w, cy/3/w);
137     }
138 }
139 char ahas(const point<T>& t) const { //點是否在簡單多邊形內，
140     //是的話回傳1，在邊上回傳-1，否則回傳0
141     bool c=0;
142     for(int i=0, j=p.size()-1; i<p.size(); j=i++){
143         if(line<T>(p[i], p[j]).point_on_segment(t)) return -1;
144         else if((p[i].y>t.y) != (p[j].y>t.y) &&
145             t.x<(p[j].x-p[i].x)*(t.y-p[i].y)/(p[j].y-p[i].y)+p[i].x)
146             )
147             c=!c;
148     }
149     return c;
150 }
151 char point_in_convex(const point<T>&x) const {
152     int l=1, r=(int)p.size()-2;
153     while(l<r) { //點是否在凸多邊形內，是的話回傳1，在邊上回傳
154         // -1，否則回傳0
155         int mid=(l+r)/2;
156         T a1=(p[mid]-p[0]).cross(x-p[0]);
157         T a2=(p[mid+1]-p[0]).cross(x-p[0]);
158         if(a1>=0&&a2<=0) {
159             T res=(p[mid+1]-p[mid]).cross(x-p[mid]);
160             return res>0?1:(res>=0?-1:0);
161         } else if(a1<0) r=mid-1;
162         else l=mid+1;
163     }
164     return 0;
165 }
166 vector<T> getA() const { //凸包邊對x軸的夾角
167     vector<T> res; //一定是遞增的
168     for(size_t i=0; i<p.size(); ++i)
169         res.push_back((p[(i+1)%p.size()]-p[i]).getA());
170     return res;
171 }
172 bool line_intersect(const vector<T>&A, const line<T> &l)
173     const { //O(LogN)
174     int f1=upper_bound(A.begin(), A.end(), (l.p1-l.p2).getA())-
175         A.begin();
176     int f2=upper_bound(A.begin(), A.end(), (l.p2-l.p1).getA())-
177         A.begin();
178     return l.cross_seg(line<T>(p[f1], p[f2]));
179 }
180 polygon cut(const line<T> &l) const { //凸包對直線切割，得到直
181     //線L左側的凸包
182     polygon ans;
183     for(int n=p.size(), i=n-1, j=0; j<n; i=j++){
184         if(l.ori(p[i])>=0) {
185             ans.p.push_back(p[i]);
186             if(l.ori(p[j])<0)
187                 ans.p.push_back(l.line_intersection(line<T>(p[i], p[
188                     j])));
189             } else if(l.ori(p[j])>0)
190                 ans.p.push_back(l.line_intersection(line<T>(p[i], p[j
191                     ])));
192         }
193     }
194     return ans;
195 }
196 static bool monotone_chain_cmp(const point<T>& a, const
197     point<T>& b) { //凸包排序函數
198     return (a.x<b.x) || (a.x==b.x && a.y<b.y);
199 }
200 void monotone_chain(vector<point<T>> &s) { //凸包
201     sort(s.begin(), s.end(), monotone_chain_cmp);
202     p.resize(s.size()+1);
203     int m=0;
204     for(size_t i=0; i<s.size(); ++i) {
205         while(m>=2 && (p[m-1]-p[m-2]).cross(s[i]-p[m-2])<=0) --m;
206         p[m++]=s[i];
207     }
208     for(int i=s.size()-2, t=m+1; i>=0; --i) {
209         while(m>=2 && (p[m-1]-p[m-2]).cross(s[i]-p[m-2])<=0) --m;
210         p[m++]=s[i];
211     }
212     if(s.size()>1) --m;
213     p.resize(m);
214 }
215 T diam() { //直徑
216     int n=p.size(), t=1;
217     T ans=0; p.push_back(p[0]);
218     for(int i=0; i<n; i++){
219         point<T> now=p[i+1]-p[i];
220         while(now.cross(p[t+1]-p[i])>now.cross(p[t]-p[i])) t=(t
221             +1)%n;
222         ans=max(ans, (p[i]-p[t]).abs2());
223     }
224     return p.pop_back(), ans;
225 }
226 T min_cover_rectangle() { //最小覆蓋矩形
227     int n=p.size(), t=1, r=1, l;
228     if(n<3) return 0; //也可以做最小周長矩形
229     T ans=1e99; p.push_back(p[0]);
230     for(int i=0; i<n; i++){
231         point<T> now=p[i+1]-p[i];
232         while(now.cross(p[t+1]-p[i])>now.cross(p[t]-p[i])) t=(t
233             +1)%n;
234         while(now.dot(p[r+1]-p[i])>now.dot(p[r]-p[i])) r=(r+1)%n;
235         if(!i) l=r;
236         while(now.dot(p[l+1]-p[i])<=now.dot(p[l]-p[i])) l=(l+1)%
237             n;
238         T d=now.abs2();
239         T tmp=now.cross(p[t]-p[i])*(now.dot(p[r]-p[i])-now.dot(
240             p[l]-p[i]))/d;
241         ans=min(ans, tmp);
242     }
243     return p.pop_back(), ans;
244 }
245 T dis2(polygon &p1) { //凸包最近距離平方
246     vector<point<T>> &P=p, &Q=p1.p;
247     int n=P.size(), m=Q.size(), l=0, r=0;
248     for(int i=0; i<n; ++i) if(P[i].y<P[l].y) l=i;
249     for(int i=0; i<m; ++i) if(Q[i].y<Q[r].y) r=i;
250     P.push_back(P[0]), Q.push_back(Q[0]);
251     T ans=1e99;
252     for(int i=0; i<n; ++i) {
253         while((P[l]-P[l+1]).cross(Q[r+1]-Q[r])<0) r=(r+1)%m;
254         ans=min(ans, line<T>(P[l], P[l+1]).seg_dis2(line<T>(Q[r],
255             Q[r+1])));
256         l=(l+1)%n;
257     }
258     return P.pop_back(), Q.pop_back(), ans;
259 }
260 static char sign(const point<T>&t) {
261     return (t.y==0?t.x:t.y)<0;
262 }
263 static bool angle_cmp(const line<T>& A, const line<T>& B) {
264     point<T> a=A.p2-A.p1, b=B.p2-B.p1;
265     return sign(a)<sign(b) || (sign(a)==sign(b) && a.cross(b)>0);
266 }

```

```

246 }
247 int halfplane_intersection(vector<line<T> > &s){//半平面交
248     sort(s.begin(),s.end(),angle_cmp);//線段左側為該線段半平面
249     int L,R,n=s.size();
250     vector<point<T> > px(n);
251     vector<line<T> > q(n);
252     q[L=R=0]=s[0];
253     for(int i=1;i<n;++i){
254         while(L<R&&s[i].ori(px[R-1])<=0)--R;
255         while(L<R&&s[i].ori(px[L])<=0)++L;
256         q[++R]=s[i];
257         if(q[R].parallel(q[R-1])){
258             --R;
259             if(q[R].ori(s[i].p1)>0)q[R]=s[i];
260         }
261         if(L<R)px[R-1]=q[R-1].line_intersection(q[R]);
262     }
263     while(L<R&&q[L].ori(px[R-1])<=0)--R;
264     p.clear();
265     if(R-L<=1)return 0;
266     px[R]=q[R].line_intersection(q[L]);
267     for(int i=L;i<=R;++i)p.push_back(px[i]);
268     return R-L+1;
269 }
270 };
271 template<typename T>
272 struct triangle{
273     point<T> a,b,c;
274     triangle(){
275         triangle(const point<T> &a,const point<T> &b,const point<T> &c):a(a),b(b),c(c){}
276     }
277     T area()const{
278         T t=(b-a).cross(c-a)/2;
279         return t>0?t:-t;
280     }
281     point<T> barycenter()const{//重心
282         return (a+b+c)/3;
283     }
284     point<T> circumcenter()const{//外心
285         static line<T> u,v;
286         u.p1=(a+b)/2;
287         u.p2=point<T>(u.p1.x-a.y+b.y,u.p1.y+a.x-b.x);
288         v.p1=(a+c)/2;
289         v.p2=point<T>(v.p1.x-a.y+c.y,v.p1.y+a.x-c.x);
290         return u.line_intersection(v);
291     }
292     point<T> incenter()const{//內心
293         T A=sqrt((b-c).abs2()),B=sqrt((a-c).abs2()),C=sqrt((a-b).abs2());
294         return point<T>(A*a.x+B*b.x+C*c.x,A*a.y+B*b.y+C*c.y)/(A+B+C);
295     }
296     point<T> perpcenter()const{//垂心
297         return barycenter()*3-circumcenter()*2;
298     }
299 };
300 template<typename T>
301 struct point3D{
302     T x,y,z;
303     point3D(){
304         point3D(const T&x,const T&y,const T&z):x(x),y(y),z(z){}
305         point3D operator+(const point3D &b)const{
306             return point3D(x+b.x,y+b.y,z+b.z);
307         }
308         point3D operator*(const T &b)const{
309             return point3D(x*b,y*b,z*b);
310         }
311         point3D operator/(const T &b)const{
312             return point3D(x/b,y/b,z/b);
313         }
314         bool operator==(const point3D &b)const{
315             return x==b.x&&y==b.y&&z==b.z;
316         }
317         T dot(const point3D &b)const{
318             return x*b.x+y*b.y+z*b.z;
319         }
320         point3D cross(const point3D &b)const{
321             return point3D(y*b.z-z*b.y,z*b.x-x*b.z,x*b.y-y*b.x);
322         }
323         T abs2()const{//向量長度的平方
324             return dot(*this);
325         }
326         T area2(const point3D &b)const{//和b、原點圍成面積的平方
327             return cross(b).abs2()/4;
328         }
329 };
330 template<typename T>
331 struct line3D{
332     point3D<T> p1,p2;
333     line3D(){
334         line3D(const point3D<T> &p1,const point3D<T> &p2):p1(p1),p2(p2){}
335     }
336     T dis2(const point3D<T> &p,bool is_segment=0)const{//點跟直線/線段的距離平方
337         point3D<T> v=p2-p1,v1=p-p1;
338         if(is_segment){
339             point3D<T> v2=p-p2;
340             if(v.dot(v1)<=0)return v1.abs2();
341             if(v.dot(v2)>=0)return v2.abs2();
342         }
343         point3D<T> tmp=v.cross(v1);
344         return tmp.abs2()/v.abs2();
345     }
346     pair<point3D<T>,point3D<T> > closest_pair(const line3D<T> &l)const{
347         point3D<T> v1=(p1-p2),v2=(l.p1-l.p2);
348         point3D<T> N=v1.cross(v2),ab(p1-l.p1);
349         //if(N.abs2()==0)return NULL;平行或重合
350         T tmp=N.dot(ab),ans=tmp*tmp/N.abs2();//最近點對距離
351         point3D<T> d1=p2-p1,d2=l.p2-l.p1,D=d1.cross(d2),G=l.p1-p1;
352         T t1=(G.cross(d2)).dot(D)/D.abs2();
353         T t2=(G.cross(d1)).dot(D)/D.abs2();
354         return make_pair(p1+d1*t1,l.p1+d2*t2);
355     }
356     bool same_side(const point3D<T> &a,const point3D<T> &b)const{
357         return (p2-p1).cross(a-p1).dot((p2-p1).cross(b-p1))>0;
358     }
359 };
360 template<typename T>
361 struct plane{
362     point3D<T> p0,n;//平面上的點和法向量
363     plane(){
364         plane(const point3D<T> &p0,const point3D<T> &n):p0(p0),n(n){}
365     }
366     T dis2(const point3D<T> &p)const{//點到平面距離的平方
367         T tmp=(p-p0).dot(n);
368         return tmp*tmp/n.abs2();
369     }
370     point3D<T> projection(const point3D<T> &p)const{
371         return p-n*(p-p0).dot(n)/n.abs2();
372     }
373     point3D<T> line_intersection(const line3D<T> &l)const{
374         T tmp=n.dot(l.p2-l.p1);//等於0表示平行或重合該平面
375         return l.p1+(l.p2-l.p1)*(n.dot(p0-l.p1)/tmp);
376     }
377     line3D<T> plane_intersection(const plane &p1)const{
378         point3D<T> e=n.cross(p1.n),v=n.cross(e);
379         T tmp=p1.n.dot(v);//等於0表示平行或重合該平面
380         point3D<T> q=p0+(v*(p1.n.dot(p1.p0-p0))/tmp);
381         return line3D<T>(q,q+e);
382     }
383 };
384 template<typename T>
385 struct triangle3D{
386     point3D<T> a,b,c;
387     triangle3D(){
388         triangle3D(const point3D<T> &a,const point3D<T> &b,const point3D<T> &c):a(a),b(b),c(c){}
389     }
390     bool point_in(const point3D<T> &p)const{//點在該平面上的投影在三角形中
391         return line3D<T>(b,c).same_side(p,a)&&line3D<T>(a,c).same_side(p,b)&&line3D<T>(a,b).same_side(p,c);
392     }
393 };
394 template<typename T>
395 struct tetrahedron{//四面體
396     point3D<T> a,b,c,d;
397     tetrahedron(){
398         tetrahedron(const point3D<T> &a,const point3D<T> &b,const point3D<T> &c,const point3D<T> &d):a(a),b(b),c(c),d(d){}
399     }
400     T volume6()const{//體積的六倍
401         return (d-a).dot((b-a).cross(c-a));
402     }
403     point3D<T> centroid()const{
404         return (a+b+c+d)/4;
405     }
406     bool point_in(const point3D<T> &p)const{
407         return triangle3D<T>(a,b,c).point_in(p)&&triangle3D<T>(c,d,a).point_in(p);
408     }
409 };
410 template<typename T>
411 struct convexhull3D{
412     static const int MAXN=1005;
413     struct face{
414         int a,b,c;
415         face(int a,int b,int c):a(a),b(b),c(c){}
416     };
417     vector<point3D<T>> pt;
418     vector<face> ans;
419     int fid[MAXN][MAXN];
420     void build(){
421         int n=pt.size();
422         ans.clear();
423         memset(fid,0,sizeof(fid));
424         ans.emplace_back(0,1,2);//注意不能共線
425         ans.emplace_back(2,1,0);
426         int ftop = 0;
427         for(int i=3, ftop=1; i<n; ++i,++ftop){
428             vector<face> next;
429             for(auto &f:ans){
430                 T d=(pt[i]-pt[f.a]).dot((pt[f.b]-pt[f.a]).cross(pt[f.c]-pt[f.a]));
431                 if(d<=0) next.push_back(f);
432                 int ff=0;
433                 if(d>0) ff=ftop;
434             }
435         }
436     }
437 };

```



```

423     else if(d<0) ff=-ftop;
424     fid[f.a][f.b]=fid[f.b][f.c]=fid[f.c][f.a]=ff;
425 }
426 for(auto &f:ans){
427     if(fid[f.a][f.b]>0 && fid[f.a][f.b]!=fid[f.b][f.a])
428         next.emplace_back(f.a,f.b,i);
429     if(fid[f.b][f.c]>0 && fid[f.b][f.c]!=fid[f.c][f.b])
430         next.emplace_back(f.b,f.c,i);
431     if(fid[f.c][f.a]>0 && fid[f.c][f.a]!=fid[f.a][f.c])
432         next.emplace_back(f.c,f.a,i);
433 }
434 ans=next;
435 }
436 }
437 point3D<T> centroid()const{
438     point3D<T> res(0,0,0);
439     T vol=0;
440     for(auto &f:ans){
441         T tmp=pt[f.a].dot(pt[f.b].cross(pt[f.c]));
442         res=res+(pt[f.a]+pt[f.b]+pt[f.c])*tmp;
443         vol+=tmp;
444     }
445     return res/(vol*4);
446 }
447 };

```

5.3 Pick's Theorem

給定頂點坐標均是整點的簡單多邊形，面積 = 內部格點數 + 邊上格點數/2 - 1

6 Graph

6.1 2-SAT

```

1 struct TWO_SAT {
2     int n, N;
3     vector<vector<int>> G, rev_G;
4     deque<bool> used;
5     vector<int> order, comp;
6     deque<bool> assignment;
7     void init(int _n) {
8         n = _n;
9         N = _n * 2;
10        G.resize(N + 5);
11        rev_G.resize(N + 5);
12    }
13    void dfs1(int v) {
14        used[v] = true;
15        for (int u : G[v]) {
16            if (!used[u])
17                dfs1(u);
18        }
19        order.push_back(v);
20    }
21    void dfs2(int v, int c1) {
22        comp[v] = c1;
23        for (int u : rev_G[v]) {
24            if (comp[u] == -1)
25                dfs2(u, c1);

```

```

26    }
27 }
28 bool solve() {
29     order.clear();
30     used.assign(N, false);
31     for (int i = 0; i < N; ++i) {
32         if (!used[i])
33             dfs1(i);
34     }
35     comp.assign(N, -1);
36     for (int i = 0, j = 0; i < N; ++i) {
37         int v = order[N - i - 1];
38         if (comp[v] == -1)
39             dfs2(v, j++);
40     }
41     assignment.assign(n, false);
42     for (int i = 0; i < N; i += 2) {
43         if (comp[i] == comp[i + 1])
44             return false;
45         assignment[i / 2] = (comp[i] > comp[i + 1]);
46     }
47     return true;
48 }
49 void add_disjunction(int a, bool na, int b, bool nb) { //
50     A or B 都是 0-based
51     // na means whether a is negative or not
52     // nb means whether b is negative or not
53     a = 2 * a ^ na;
54     b = 2 * b ^ nb;
55     int neg_a = a ^ 1;
56     int neg_b = b ^ 1;
57     G[neg_a].push_back(b);
58     G[neg_b].push_back(a);
59     rev_G[b].push_back(neg_a);
60     rev_G[a].push_back(neg_b);
61     return;
62 }
63 void get_result(vector<int>& res) {
64     res.clear();
65     for (int i = 0; i < n; i++)
66         res.push_back(assignment[i]);
67 };

```

6.2 Augment Path

```

1 struct AugmentPath{
2     int n, m;
3     vector<vector<int>> G;
4     vector<int> mx, my;
5     vector<int> visx, visy;
6     int stamp;
7
8     AugmentPath(int _n, int _m) : n(_n), m(_m), G(n), mx(n),
9         my(m, -1), visx(n), visy(n){
10         stamp = 0;
11     }
12
13     void add(int x, int y){
14         G[x].push_back(y);
15     }
16     // bb03e2

```

```

17 bool dfs1(int now){
18     visx[now] = stamp;
19
20     for (auto x : G[now]){
21         if (my[x]==-1){
22             mx[now] = x;
23             my[x] = now;
24             return true;
25         }
26     }
27     for (auto x : G[now]){
28         if (visx[my[x]]!=stamp && dfs1(my[x])){
29             mx[now] = x;
30             my[x] = now;
31             return true;
32         }
33     }
34     return false;
35 }
36
37 vector<pair<int, int>> find_max_matching(){
38     vector<pair<int, int>> ret;
39
40     while (true){
41         stamp++;
42         int tmp = 0;
43         for (int i=0 ; i<n ; i++){
44             if (mx[i]==-1 && dfs1(i)) tmp++;
45         }
46         if (tmp==0) break;
47     }
48
49     for (int i=0 ; i<n ; i++){
50         if (mx[i]!=-1){
51             ret.push_back({i, mx[i]});
52         }
53     }
54     return ret;
55 }
56
57 // 645577
58 void dfs2(int now){
59     visx[now] = true;
60
61     for (auto x : G[now]){
62         if (my[x]!=-1 && visy[x]==false){
63             visy[x] = true;
64             dfs2(my[x]);
65         }
66     }
67 }
68
69 // 要先執行 find_max_matching 一次
70 vector<pair<int, int>> find_min_vertex_cover(){
71     fill(visx.begin(), visx.end(), false);
72     fill(visy.begin(), visy.end(), false);
73
74     vector<pair<int, int>> ret;
75     for (int i=0 ; i<n ; i++){
76         if (mx[i]==-1) dfs2(i);
77     }
78
79     for (int i=0 ; i<n ; i++){
80         if (visx[i]==false) ret.push_back({1, i});
81     }

```

```

82     for (int i=0 ; i<m ; i++){
83         if (visy[i]==true) ret.push_back({2, i});
84     }
85
86     return ret;
87 }
88 };

```

6.3 Bridge BCC

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 const int N = 200005;
5 vector<int> G[N];
6 int low[N], depth[N];
7 bool vis[N];
8 vector<vector<int>> bcc;
9 stack<int> stk;
10
11 void dfs(int v, int p) {
12     stk.push(v);
13     vis[v] = true;
14     low[v] = depth[v] = (p == -1 ? 1 : depth[p] + 1);
15     for (int u : G[v]) {
16         if (u == p) continue;
17         if (!vis[u]) {
18             /// (v, u) 是樹邊
19             dfs(u, v);
20             low[v] = min(low[v], low[u]);
21         } else {
22             /// (v, u) 是回邊
23             low[v] = min(low[v], depth[u]);
24         }
25     }
26     /// v 在不依靠父邊的情況下永遠沒辦法走到它的祖先
27     if (low[v] == depth[v]) {
28         bcc.emplace_back();
29         while (stk.top() != v) {
30             bcc.back().push_back(stk.top());
31             stk.pop();
32         }
33         bcc.back().push_back(stk.top());
34         stk.pop();
35     }
36 }

```

6.4 Cut BCC

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 const int N = 200005;
5 vector<int> G[N];
6 int low[N], depth[N];
7 bool vis[N];
8 vector<vector<int>> bcc;
9 stack<int> stk;
10

```

```

11 void dfs(int v, int p) {
12     stk.push(v);
13     vis[v] = true;
14     low[v] = depth[v] = (p == -1 ? 1 : depth[p] + 1);
15     for (int u : G[v]) {
16         if (u == p) continue;
17         if (!vis[u]) {
18             /// (v, u) 是樹邊
19             dfs(u, v);
20             low[v] = min(low[v], low[u]);
21             /// u 無法在不經過父邊的情況走到 v 的祖先
22             if (low[u] >= depth[v]) {
23                 bcc.emplace_back();
24                 while (stk.top() != u) {
25                     bcc.back().push_back(stk.top());
26                     stk.pop();
27                 }
28                 bcc.back().push_back(stk.top());
29                 stk.pop();
30                 bcc.back().push_back(v);
31             }
32         } else {
33             /// (v, u) 是回邊
34             low[v] = min(low[v], depth[u]);
35         }
36     }
37 }

```

6.5 Dijkstra

```

1 // 可以在  $O(E \log E)$  的時間複雜度解決在無負權有向圖單點源最短
   路
2 const int INF = 2e18; // 要確保 INF 開的足夠大
3
4 vector<vector<pair<int, int>>> G(n); // G[i] = <節點, 權重>
5 vector<int> dis(n, INF);
6 priority_queue<pair<int, int>, vector<pair<int, int>>,
   greater<pair<int, int>>> pq;
7 dis[s] = 0;
8 pq.push({0, s});
9
10 while (pq.size()){
11     int now_dis = pq.top().first;
12     int now_node = pq.top().second;
13     pq.pop();
14
15     if (now_dis > dis[now_node]) continue;
16
17     for (auto x : G[now_node]){
18         if (now_dis + x.second < dis[x.first]){
19             dis[x.first] = now_dis + x.second;
20             pq.push({dis[x.first], x.first});
21         }
22     }
23 }

```

6.6 Dinic

```

1 // 一般圖： $O(EV^2)$ 
2 // 二分圖： $O(EV)$ 
3 struct Flow{
4     using T = int; // 可以換成別的类型別
5     struct Edge{
6         int v; T rc; int rid;
7     };
8     vector<vector<Edge>> G;
9     void add(int u, int v, T c){
10         G[u].push_back({v, c, G[v].size()});
11         G[v].push_back({u, 0, G[u].size()-1});
12     }
13     vector<int> dis, it;
14
15     Flow(int n){
16         G.resize(n);
17         dis.resize(n);
18         it.resize(n);
19     }
20
21     /// ce56d6
22     T dfs(int u, int t, T f){
23         if (u == t || f == 0) return f;
24         for (int &i=it[u] ; i<G[u].size() ; i++){
25             auto &[v, rc, rid] = G[u][i];
26             if (dis[v]!=dis[u]+1) continue;
27             T df = dfs(v, t, min(f, rc));
28             if (df <= 0) continue;
29             rc -= df;
30             G[v][rid].rc += df;
31             return df;
32         }
33         return 0;
34     }
35
36     /// e22e39
37     T flow(int s, int t){
38         T ans = 0;
39         while (true){
40             fill(dis.begin(), dis.end(), INF);
41             queue<int> q;
42             q.push(s);
43             dis[s] = 0;
44
45             while (q.size()){
46                 int u = q.front(); q.pop();
47                 for (auto [v, rc, rid] : G[u]){
48                     if (rc <= 0 || dis[v] < INF) continue;
49                     dis[v] = dis[u] + 1;
50                     q.push(v);
51                 }
52             }
53             if (dis[t]==INF) break;
54
55             fill(it.begin(), it.end(), 0);
56             while (true){
57                 T df = dfs(s, t, INF);
58                 if (df <= 0) break;
59                 ans += df;
60             }
61         }
62         return ans;
63     }
64 }
65
66 // the code below constructs minimum cut

```

```

66 void dfs_mincut(int now, vector<bool> &vis){
67     vis[now] = true;
68     for (auto &[v, rc, rid] : G[now]){
69         if (vis[v] == false && rc > 0){
70             dfs_mincut(v, vis);
71         }
72     }
73 }
74
75 vector<pair<int, int>> construct(int n, int s, vector<
    pair<int, int>> &E){
76     // E is G without capacity
77     vector<bool> vis(n);
78     dfs_mincut(s, vis);
79     vector<pair<int, int>> ret;
80     for (auto &[u, v] : E){
81         if (vis[u] == true && vis[v] == false){
82             ret.emplace_back(u, v);
83         }
84     }
85     return ret;
86 }
87 };

```

6.7 Dominator Tree

```

1  /*
2  全部都是  $\theta$ -based
3  G 要是有向無權圖
4  一開始要初始化  $G(N, root)$  · 代表有  $N$  個節點 · 根是  $root$ 
5  用完之後要 build
6   $G[i] = i$  的  $idom$  · 也就是從  $root$  走到  $i$  時 · 一定要走到的點且離
   i 最近
7  */
8  struct DominatorTree{
9      int N;
10     vector<vector<int>> G;
11     vector<vector<int>> buckets, rg;
12     // dfn[x] = the DFS order of x
13     // rev[x] = the vertex with DFS order x
14     // par[x] = the parent of x
15     vector<int> dfn, rev, par;
16     vector<int> sdom, dom, idom;
17     vector<int> fa, val;
18     int stamp;
19     int root;
20
21     int operator [] (int x){
22         return idom[x];
23     }
24
25     DominatorTree(int _N, int _root) :
26         N(_N),
27         G(N), buckets(N), rg(N),
28         dfn(N, -1), rev(N, -1), par(N, -1),
29         sdom(N, -1), dom(N, -1), idom(N, -1),
30         fa(N, -1), val(N, -1)
31     {
32         stamp = 0;
33         root = _root;
34     }
35 }

```

```

36 void add_edge(int u, int v){
37     G[u].push_back(v);
38 }
39
40 void dfs(int x){
41     rev[dfn[x] = stamp] = x;
42     fa[stamp] = sdom[stamp] = val[stamp] = stamp;
43     stamp++;
44
45     for (int u : G[x]){
46         if (dfn[u]==-1){
47             dfs(u);
48             par[dfn[u]] = dfn[x];
49         }
50         rg[dfn[u]].push_back(dfn[x]);
51     }
52 }
53
54 int eval(int x, bool first){
55     if (fa[x]==x) return !first ? -1 : x;
56     int p = eval(fa[x], false);
57
58     if (p==-1) return x;
59     if (sdom[val[x]]>sdom[val[fa[x]]]) val[x] = val[fa[x]];
60     fa[x] = p;
61
62     return !first ? p : val[x];
63 }
64
65 void link(int x, int y){
66     fa[x] = y;
67 }
68
69 void build(){
70     dfs(root);
71
72     for (int x=stamp-1 ; x>=0 ; x--){
73         for (int y : rg[x]){
74             sdom[x] = min(sdom[x], sdom[eval(y, true)]);
75         }
76         if (x>0) buckets[sdom[x]].push_back(x);
77         for (int u : buckets[x]){
78             int p = eval(u, true);
79             if (sdom[p]==x) dom[u] = x;
80             else dom[u] = p;
81         }
82         if (x>0) link(x, par[x]);
83     }
84
85     idom[root] = root;
86     for (int x=1 ; x<stamp ; x++){
87         if (sdom[x]!=dom[x]) dom[x] = dom[dom[x]];
88     }
89     for (int i=1 ; i<stamp ; i++) idom[rev[i]] = rev[dom[i]];
90 }
91 };

```

6.8 Enumerate Triangle

1 // $O(m \log m)$ 枚舉無向圖所有三角形 · θ -based

```

2 void Enumerate_Triangle(vector<pair<int, int>> &edge, vector<
    int> &deg){
3     int n = deg.size();
4     int m = edge.size();
5     vector<vector<int>> G(n);
6
7     for (int i=0 ; i<m ; i++){
8         if (deg[edge[i].first] > deg[edge[i].second]) swap(
9             edge[i].first, edge[i].second);
10        if (deg[edge[i].first] == deg[edge[i].second] && edge
11            [i].first > edge[i].second) swap(edge[i].first,
12            edge[i].second);
13        G[edge[i].first].push_back(edge[i].second);
14    }
15
16    vector<int> vis(n, false);
17    for (int i=0 ; i<n ; i++){
18        for (auto j : G[i]) vis[j] = true;
19        for (auto j : G[i]){
20            for (auto k : G[j]){
21                if (vis[k]){
22                    // i, j, k is a triangle
23                }
24            }
25        }
26        for (auto j : G[i]) vis[j] = false;
27    }
28 }

```

6.9 Find Bridge

```

1 vector<int> dep(MAX_N), low(MAX_N);
2 vector<pair<int, int>> bridge;
3 bitset<MAX_N> vis;
4
5 void dfs(int now, int pre){
6     vis[now] = 1;
7     low[now] = dep[now] = (now==1 ? 0 : dep[pre]+1);
8
9     for (auto x : G[now]){
10        if (x==pre){
11            continue;
12        }else if (vis[x]==0){
13            // 沒有走過的節點
14            dfs(x, now);
15            low[now] = min(low[now], low[x]);
16        }else if (vis[x]==1){
17            low[now] = min(low[now], dep[x]);
18        }
19    }
20
21    if (now!=1 && low[now]==dep[now]){
22        bridge.push_back({now, pre});
23    }
24    return;
25 }

```

6.10 HLD


```

1 #include <bits/stdc++.h>
2 #define int long long
3 using namespace std;
4
5 const int N = 100005;
6 vector<int> G[N];
7 struct HLD {
8     vector<int> pa, sz, depth, mxson, topf, id;
9     int n, idcnt = 0;
10    HLD(int n) : n(n), pa(n+1), sz(n+1), depth(n+1),
11            mxson(n+1), topf(n+1), id(n+1) {}
12    void dfs1(int v = 1, int p = -1) {
13        pa[v] = p; sz[v] = 1; mxson[v] = 0;
14        depth[v] = (p == -1 ? 0 : depth[p] + 1);
15        for (int u : G[v]) {
16            if (u == p) continue;
17            dfs1(u, v);
18            sz[v] += sz[u];
19            if (sz[u] > sz[mxson[v]]) mxson[v] = u;
20        }
21    }
22    void dfs2(int v = 1, int top = 1) {
23        id[v] = ++idcnt;
24        topf[v] = top;
25        if (mxson[v]) dfs2(mxson[v], top);
26        for (int u : G[v]) {
27            if (u == mxson[v] || u == pa[v]) continue;
28            dfs2(u, u);
29        }
30    }
31    // query 為區間資料結構
32    int path_query(int a, int b) {
33        int res = 0;
34        while (topf[a] != topf[b]) { /// 若不在同一條鍊上
35            if (depth[topf[a]] < depth[topf[b]]) swap(a, b);
36            res = max(res, 0ll); // query : L = id[topf[a]],
37                                // r = id[a]
38            a = pa[topf[a]];
39        }
40        /// 此時已在同一條鍊上
41        if (depth[a] < depth[b]) swap(a, b);
42        res = max(res, 0ll); // query : L = id[b], r = id[a]
43        return res;
44    }
45 };

```

6.11 Kosaraju

```

1 /*
2 給定一個有向圖，迴傳傳縮點後的圖、SCC 的資訊
3 所有點都以 based-0 編號
4
5 函式：
6 SCC_compress G(n): 宣告一個有 n 個點的圖
7 .add_edge(u, v): 加上一條邊 u -> v
8 .compress: O(n log n) 計算 G3、SCC、SCC_id 的資訊，並把縮點後
9             的結果存在 result 裡
10
11 SCC[i] = 某個 SCC 中的所有點
12 SCC_id[i] = 第 i 個點在第幾個 SCC
13 */

```

```

13 struct SCC_compress{
14     int N, M, sz;
15     vector<vector<int>> G, inv_G, result;
16     vector<pair<int, int>> edges;
17     vector<bool> vis;
18     vector<int> order;
19
20     vector<vector<int>> SCC;
21     vector<int> SCC_id;
22
23     SCC_compress(int N) :
24         N(N), M(0), sz(0),
25         G(N), inv_G(N),
26         vis(N), SCC_id(N)
27     {}
28
29     vector<int> operator [] (int x){
30         return result[x];
31     }
32
33     void add_edge(int u, int v){
34         G[u].push_back(v);
35         inv_G[v].push_back(u);
36         edges.push_back({u, v});
37         M++;
38     }
39
40     void dfs1(vector<vector<int>> &G, int now){
41         vis[now] = 1;
42         for (auto x : G[now]) if (!vis[x]) dfs1(G, x);
43         order.push_back(now);
44     }
45
46     void dfs2(vector<vector<int>> &G, int now){
47         SCC_id[now] = SCC.size()-1;
48         SCC.back().push_back(now);
49         vis[now] = 1;
50         for (auto x : G[now]) if (!vis[x]) dfs2(G, x);
51     }
52
53     void compress(){
54         fill(vis.begin(), vis.end(), 0);
55         for (int i=0 ; i<N ; i++) if (!vis[i]) dfs1(G, i);
56
57         fill(vis.begin(), vis.end(), 0);
58         reverse(order.begin(), order.end());
59         for (int i=0 ; i<N ; i++){
60             if (!vis[order[i]]){
61                 SCC.push_back(vector<int>());
62                 dfs2(inv_G, order[i]);
63             }
64         }
65
66         result.resize(SCC.size());
67         sz = SCC.size();
68         for (auto [u, v] : edges){
69             if (SCC_id[u]!=SCC_id[v]) result[SCC_id[u]].
70                 push_back(SCC_id[v]);
71         }
72         for (int i=0 ; i<SCC.size() ; i++){
73             sort(result[i].begin(), result[i].end());
74             result[i].resize(unique(result[i].begin(), result
75                 [i].end())-result[i].begin());
76         }
77     }
78 };

```

6.12 Kuhn Munkres

```

1 // O(n^2) 找到最大權匹配
2 struct KuhnMunkres{
3     int n; // max(n, m)
4     vector<vector<int>> G;
5     vector<int> match, lx, ly, visx, visy;
6     vector<int> slack;
7     int stamp = 0;
8
9     KuhnMunkres(int n) : n(n), G(n, vector<int>(n)), lx(n),
10        ly(n), slack(n), match(n), visx(n), visy(n) {}
11
12     void add(int x, int y, int w){
13         G[x][y] = max(G[x][y], w);
14     }
15
16     bool dfs(int i, bool aug){ // aug = true 表示要更新 match
17         if (visx[i]==stamp) return false;
18         visx[i] = stamp;
19
20         for (int j=0 ; j<n ; j++){
21             if (visy[j]==stamp) continue;
22             int d = lx[i]+ly[j]-G[i][j];
23
24             if (d==0){
25                 visy[j] = stamp;
26                 if (match[j]==-1 || dfs(match[j], aug)){
27                     if (aug){
28                         match[j] = i;
29                     }
30                     return true;
31                 }
32             }
33             else{
34                 slack[j] = min(slack[j], d);
35             }
36         }
37         return false;
38     }
39
40     bool augment(){
41         for (int j=0 ; j<n ; j++){
42             if (visy[j]!=stamp && slack[j]==0){
43                 visy[j] = stamp;
44                 if (match[j]==-1 || dfs(match[j], false)){
45                     return true;
46                 }
47             }
48         }
49         return false;
50     }
51
52     void relabel(){
53         int delta = INF;
54         for (int j=0 ; j<n ; j++){
55             if (visy[j]!=stamp) delta = min(delta, slack[j]);
56         }
57         for (int i=0 ; i<n ; i++){
58             if (visx[i]==stamp) lx[i] -= delta;
59         }
60         for (int j=0 ; j<n ; j++){
61             if (visy[j]==stamp) ly[j] += delta;
62             else slack[j] -= delta;
63         }
64     }
65 };

```

```

63 int solve(){
64     for (int i=0; i<n; i++){
65         lx[i] = 0;
66         for (int j=0; j<n; j++){
67             lx[i] = max(lx[i], G[i][j]);
68         }
69     }
70     fill(ly.begin(), ly.end(), 0);
71     fill(match.begin(), match.end(), -1);
72
73     for(int i = 0; i < n; i++) {
74         fill(slack.begin(), slack.end(), INF);
75         stamp++;
76         if(dfs(i, true)) continue;
77
78         while(augment()==false) relabel();
79         stamp++;
80         dfs(i, true);
81     }
82
83     int ans = 0;
84     for (int j=0; j<n; j++){
85         if (match[j]!=-1){
86             ans += G[match[j]][j];
87         }
88     }
89     return ans;
90 }
91 }
92 }
93 }
94 };

```

6.13 LCA

```

1 struct Tree{
2     int N, M = 0, H;
3     vector<vector<int>> G;
4     vector<vector<int>> LCA;
5     vector<int> parent;
6     vector<int> dep;
7
8     Tree(int _N) : N(_N), H(__lg(_N)+1){
9         G.resize(N);
10        parent.resize(N, -1);
11        dep.resize(N, 0);
12        LCA.resize(H, vector<int>(N, 0));
13    }
14
15    void add_edge(int u, int v){
16        M++;
17        G[u].push_back(v);
18        G[v].push_back(u);
19    }
20
21    void dfs(int now, int pre){ // root 的 pre 是自己
22        dep[now] = dep[pre]+1;
23        parent[now] = pre;
24        for (auto x : G[now]){
25            if (x==pre) continue;
26            dfs(x, now);
27        }
28    }

```

```

29 void build_LCA(int root = 0){
30     dfs(root, root);
31     for (int i=0; i<N; i++) LCA[0][i] = parent[i];
32     for (int i=1; i<H; i++){
33         for (int j=0; j<N; j++){
34             LCA[i][j] = LCA[i-1][LCA[i-1][j]];
35         }
36     }
37 }
38
39 int jump(int u, int step){
40     for (int i=0; i<H; i++){
41         if (step&(1<<i)) u = LCA[i][u];
42     }
43     return u;
44 }
45
46 int get_LCA(int u, int v){
47     if (dep[u]<dep[v]) swap(u, v);
48     u = jump(u, dep[u]-dep[v]);
49     if (u==v) return u;
50     for (int i=H-1; i>=0; i--){
51         if (LCA[i][u]!=LCA[i][v]){
52             u = LCA[i][u];
53             v = LCA[i][v];
54         }
55     }
56     return parent[u];
57 }
58 }
59 };

```

6.14 MCMF

```

1 struct Flow {
2     struct Edge {
3         int u, rc, k, rv;
4     };
5
6     vector<vector<Edge>> G;
7     vector<int> par, par_eid;
8     Flow(int n) : G(n+1), par(n+1), par_eid(n+1) {}
9
10    // v->u, capacity: c, cost: k
11    void add(int v, int u, int c, int k){
12        G[v].push_back({u, c, k, SZ(G[u])});
13        G[u].push_back({v, 0, -k, SZ(G[v])-1});
14    }
15
16    // 3701d6
17    int spfa(int s, int t){
18        fill(ALL(par), -1);
19        vector<int> dis(SZ(par), INF);
20        vector<bool> in_q(SZ(par), false);
21        queue<int> Q;
22        dis[s] = 0;
23        in_q[s] = true;
24        Q.push(s);
25
26        while (!Q.empty()){
27            int v = Q.front();
28            Q.pop();
29            in_q[v] = false;

```

```

30     for (int i=0; i<SZ(G[v]); i++){
31         auto [u, rc, k, rv] = G[v][i];
32         if (rc>0 && dis[v]+k<dis[u]){
33             dis[u] = dis[v]+k;
34             par[u] = v;
35             par_eid[u] = i;
36             if (!in_q[u]) Q.push(u);
37             in_q[u] = true;
38         }
39     }
40 }
41
42 return dis[t];
43 }
44
45 // return <max flow, min cost>, 150093
46 pair<int, int> flow(int s, int t){
47     int fl = 0, cost = 0, d;
48     while ((d = spfa(s, t))<INF){
49         int cur = INF;
50         for (int v=t; v!=s; v=par[v]){
51             cur = min(cur, G[par[v]][par_eid[v]].rc);
52             fl += cur;
53             cost += d*cur;
54         }
55         for (int v=t; v!=s; v=par[v]){
56             G[par[v]][par_eid[v]].rc -= cur;
57             G[v][G[par[v]][par_eid[v]].rv].rc += cur;
58         }
59     }
60     return {fl, cost};
61 }
62
63 vector<pair<int, int>> construct(){
64     vector<pair<int, int>> ret;
65     for (int i=0; i<n; i++){
66         for (auto x : G[i]){
67             if (x.rc==0){
68                 ret.push_back({i+1, x.u-n+1});
69                 break;
70             }
71         }
72     }
73     return ret;
74 }
75 };

```

6.15 Tarjan

```

1 struct tarjan_SCC {
2     int now_T, now_SCCs;
3     vector<int> dfn, low, SCC;
4     stack<int> S;
5     vector<vector<int>> E;
6     vector<bool> vis, in_stack;
7
8     tarjan_SCC(int n) {
9         init(n);
10    }
11    void init(int n) {
12        now_T = now_SCCs = 0;
13        dfn = low = SCC = vector<int>(n);
14        E = vector<vector<int>>(n);

```

```

15     S = stack<int>();
16     vis = in_stack = vector<bool>(n);
17 }
18 void add(int u, int v) {
19     E[u].push_back(v);
20 }
21 void build() {
22     for (int i = 0; i < dfn.size(); ++i) {
23         if (!dfn[i]) dfs(i);
24     }
25 }
26 void dfs(int v) {
27     now_T++;
28     vis[v] = in_stack[v] = true;
29     dfn[v] = low[v] = now_T;
30     S.push(v);
31     for (auto &i:E[v]) {
32         if (!vis[i]) {
33             vis[i] = true;
34             dfs(i);
35             low[v] = min(low[v], low[i]);
36         }
37         else if (in_stack[i]) {
38             low[v] = min(low[v], dfn[i]);
39         }
40     }
41     if (low[v] == dfn[v]) {
42         int tmp;
43         do {
44             tmp = S.top();
45             S.pop();
46             SCC[tmp] = now_SCCs;
47             in_stack[tmp] = false;
48         } while (tmp != v);
49         now_SCCs += 1;
50     }
51 }
52 };

```

6.16 Tarjan Find AP

```

1 vector<int> dep(MAX_N), low(MAX_N), AP;
2 bitset<MAX_N> vis;
3
4 void dfs(int now, int pre){
5     int cnt = 0;
6     bool ap = 0;
7     vis[now] = 1;
8     low[now] = dep[now] = (now==1 ? 0 : dep[pre]+1);
9
10    for (auto x : G[now]){
11        if (x==pre){
12            continue;
13        }else if (vis[x]==0){
14            cnt++;
15            dfs(x, now);
16            low[now] = min(low[now], low[x]);
17            if (low[x]>=dep[now]) ap=1;
18        }else{
19            low[now] = min(low[now], dep[x]);
20        }
21    }
22 }

```

```

23     if ((now==pre && cnt>=2) || (now!=pre && ap)){
24         AP.push_back(now);
25     }
26 }

```

6.17 Tree Isomorphism

```

1 #include <bits/stdc++.h>
2 #pragma GCC optimize("O3,unroll-loops")
3 #define fastio ios::sync_with_stdio(0), cin.tie(0), cout.tie(0)
4 #define dbg(x) cerr << #x << " = " << x << endl
5 #define int long long
6 using namespace std;
7
8 // declare
9 const int MAX_SIZE = 2e5+5;
10 const int INF = 9e18;
11 const int MOD = 1e9+7;
12 const double EPS = 1e-6;
13 typedef vector<vector<int>> Graph;
14 typedef map<vector<int>, int> Hash;
15
16 int n, a, b;
17 int id1, id2;
18 pair<int, int> c1, c2;
19 vector<int> sz1(MAX_SIZE), sz2(MAX_SIZE);
20 vector<int> we1(MAX_SIZE), we2(MAX_SIZE);
21 Graph g1(MAX_SIZE), g2(MAX_SIZE);
22 Hash m1, m2;
23 int testcase=0;
24
25 void centroid(Graph &g, vector<int> &s, vector<int> &w, pair<
26     int, int> &rec, int now, int pre){
27     s[now]=1;
28     w[now]=0;
29     for (auto x : g[now]){
30         if (x!=pre){
31             centroid(g, s, w, rec, x, now);
32             s[now]+=s[x];
33             w[now]=max(w[now], s[x]);
34         }
35     }
36     w[now]=max(w[now], n-s[now]);
37     if (w[now]<=n/2){
38         if (rec.first==0) rec.first=now;
39         else rec.second=now;
40     }
41 }
42
43 int dfs(Graph &g, Hash &m, int &id, int now, int pre){
44     vector<int> v;
45     for (auto x : g[now]){
46         if (x!=pre){
47             int add=dfs(g, m, id, x, now);
48             v.push_back(add);
49         }
50     }
51     sort(v.begin(), v.end());
52
53     if (m.find(v)!=m.end()){
54         return m[v];
55     }
56 }

```

```

55 }else{
56     m[v]++;id;
57     return id;
58 }
59 }
60
61 void solve1(){
62     // init
63     id1=0;
64     id2=0;
65     c1={0, 0};
66     c2={0, 0};
67     fill(sz1.begin(), sz1.begin()+n+1, 0);
68     fill(sz2.begin(), sz2.begin()+n+1, 0);
69     fill(we1.begin(), we1.begin()+n+1, 0);
70     fill(we2.begin(), we2.begin()+n+1, 0);
71     for (int i=1; i<=n; i++){
72         g1[i].clear();
73         g2[i].clear();
74     }
75     m1.clear();
76     m2.clear();
77
78     // input
79     cin >> n;
80     for (int i=0; i<=n-1; i++){
81         cin >> a >> b;
82         g1[a].push_back(b);
83         g1[b].push_back(a);
84     }
85     for (int i=0; i<=n-1; i++){
86         cin >> a >> b;
87         g2[a].push_back(b);
88         g2[b].push_back(a);
89     }
90
91     // get tree centroid
92     centroid(g1, sz1, we1, c1, 1, 0);
93     centroid(g2, sz2, we2, c2, 1, 0);
94
95     // process
96     int res1=0, res2=0, res3=0;
97     if (c2.second!=0){
98         res1=dfs(g1, m1, id1, c1.first, 0);
99         m2=m1;
100         id2=id1;
101         res2=dfs(g2, m1, id1, c2.first, 0);
102         res3=dfs(g2, m2, id2, c2.second, 0);
103     }else if (c1.second!=0){
104         res1=dfs(g2, m1, id1, c2.first, 0);
105         m2=m1;
106         id2=id1;
107         res2=dfs(g1, m1, id1, c1.first, 0);
108         res3=dfs(g1, m2, id2, c1.second, 0);
109     }else{
110         res1=dfs(g1, m1, id1, c1.first, 0);
111         res2=dfs(g2, m1, id1, c2.first, 0);
112     }
113
114     // output
115     cout << (res1==res2 || res1==res3 ? "YES" : "NO") << endl;
116
117     return;
118 }
119

```

```

120 }
121
122 signed main(void){
123     fastio;
124
125     int t=1;
126     cin >> t;
127     while (t--){
128         solve1();
129     }
130     return 0;
131 }

```

6.18 圓方樹

```

1 #include <bits/stdc++.h>
2 #define lp(i,a,b) for(int i=(a);i<(b);i++)
3 #define pii pair<int,int>
4 #define pb push_back
5 #define ins insert
6 #define ff first
7 #define ss second
8 #define opa(x) cerr << #x << " = " << x << ", ";
9 #define op(x) cerr << #x << " = " << x << endl;
10 #define ops(x) cerr << x;
11 #define etr cerr << endl;
12 #define spc cerr << ' ';
13 #define BAE(x) (x).begin(), (x).end()
14 #define STL(x) cerr << #x << " : "; for(auto &qwe:x) cerr <<
15     qwe << ' '; cerr << endl;
16 #define deb1 cerr << "deb1" << endl;
17 #define deb2 cerr << "deb2" << endl;
18 #define deb3 cerr << "deb3" << endl;
19 #define deb4 cerr << "deb4" << endl;
20 #define deb5 cerr << "deb5" << endl;
21 #define bye exit(0);
22 using namespace std;
23
24 const int mxn = (int)(2e5) + 10;
25 const int mxlg = 17;
26 int last_special_node = (int)(1e5) + 1;
27 vector<int> E[mxn], F[mxn];
28
29 struct edg{
30     int fr, to;
31     edg(int _fr, int _to){
32         fr = _fr;
33         to = _to;
34     }
35 };
36 ostream& operator<<(ostream& os, edg x){os << x.fr << "--" <<
37     x.to;}
38 vector<edg> EV;
39
40 void tarjan(int v, int par, stack<int>& S){
41     static vector<int> dfn(mxn), low(mxn);
42     static vector<bool> to_add(mxn);
43     static int nowT = 0;
44
45     int child = 0;
46     nowT += 1;
47     dfn[v] = low[v] = nowT;
48     for(auto &ne:E[v]){

```

```

47     int i = EV[ne].to;
48     if(i == par) continue;
49     if(!dfn[i]){
50         S.push(ne);
51         tarjan(i, v, S);
52         child += 1;
53         low[v] = min(low[v], low[i]);
54
55         if(par >= 0 && low[i] >= dfn[v]){
56             vector<int> bcc;
57             int tmp;
58             do{
59                 tmp = S.top(); S.pop();
60                 if(!to_add[EV[tmp].fr]){
61                     to_add[EV[tmp].fr] = true;
62                     bcc.pb(EV[tmp].fr);
63                 }
64                 if(!to_add[EV[tmp].to]){
65                     to_add[EV[tmp].to] = true;
66                     bcc.pb(EV[tmp].to);
67                 }
68             }while(tmp != ne);
69             for(auto &j:bcc){
70                 to_add[j] = false;
71                 F[last_special_node].pb(j);
72                 F[j].pb(last_special_node);
73             }
74             last_special_node += 1;
75         }
76     }
77     else{
78         low[v] = min(low[v], dfn[i]);
79         if(dfn[i] < dfn[v]){ // edge i--v will be visited
80             // twice at here, but we only need one.
81             S.push(ne);
82         }
83     }
84 }
85
86 int dep[mxn], jmp[mxn][mxlg];
87 void dfs_lca(int v, int par, int depth){
88     dep[v] = depth;
89     for(auto &i:F[v]){
90         if(i == par) continue;
91         jmp[i][0] = v;
92         dfs_lca(i, v, depth + 1);
93     }
94 }
95
96 inline void build_lca(){
97     jmp[1][0] = 1;
98     dfs_lca(1, -1, 1);
99     lp(j,1,mxlg){
100         lp(i,1,mxn){
101             jmp[i][j] = jmp[jmp[i][j-1]][j-1];
102         }
103     }
104 }
105
106 inline int lca(int x, int y){
107     if(dep[x] < dep[y]){ swap(x, y); }
108
109     int diff = dep[x] - dep[y];
110     lp(j,0,mxlg){
111         if((diff >> j) & 1){

```

```

112         x = jmp[x][j];
113     }
114 }
115 if(x == y) return x;
116
117 for(int j = mxlg - 1; j >= 0; j--){
118     if(jmp[x][j] != jmp[y][j]){
119         x = jmp[x][j];
120         y = jmp[y][j];
121     }
122 }
123 return jmp[x][0];
124 }
125
126 inline bool can_reach(int fr, int to){
127     if(dep[to] > dep[fr]) return false;
128
129     int diff = dep[fr] - dep[to];
130     lp(j,0,mxlg){
131         if((diff >> j) & 1){
132             fr = jmp[fr][j];
133         }
134     }
135     return fr == to;
136 }
137
138 int main(){
139     ios::sync_with_stdio(false); cin.tie(0);
140     // freopen("test_input.txt", "r", stdin);
141     int n, m, q; cin >> n >> m >> q;
142     lp(i,0,m){
143         int u, v; cin >> u >> v;
144         E[u].pb(EV.size());
145         EV.pb(edg(u, v));
146         E[v].pb(EV.size());
147         EV.pb(edg(v, u));
148     }
149     E[0].pb(EV.size());
150     EV.pb(edg(0, 1));
151     stack<int> S;
152     tarjan(0, -1, S);
153     build_lca();
154
155     lp(queries,0,q){
156         int fr, to, relay; cin >> fr >> to >> relay;
157         if(fr == relay || to == relay){
158             cout << "NO\n";
159             continue;
160         }
161         if((can_reach(fr, relay) || can_reach(to, relay)) &&
162             dep[relay] >= dep[lca(fr, to)]){
163             cout << "NO\n";
164             continue;
165         }
166         cout << "YES\n";
167     }
168 }

```

6.19 最大權閉合圖

```

1 /*
2 Problem:
3 Given w = [w_0, w_1, ..., w_{n-1}] (which can be

```

```

4   either positive or negative or 0), you can choose
5   to take  $w_i$  ( $0 < i < n$ ) or not, but if edge  $u \rightarrow v$ 
6   exists, you must take  $w_v$  if you want to take  $w_u$ 
7   (in other words, you can't take  $w_u$  without taking
8    $w_v$ ), this function returns the maximum value( $> 0$ )
9   you can get. If you need a construction, you can
10  output the minimum cut of the  $S$ (source) side.
11  Complexity:
12   $\text{MaxFlow}(n, m)$  (Non-Biparte: $O(n^2m)$  / Bipartite: $O(m\sqrt{n})$ )
13  */
14  int maximum_closure(vector<int> w, vector<pair<int,int>> EV)
15  {
16      int n = w.size(), S = n + 1, T = n + 2;
17      Flow G(T + 5); // Graph/Dinic.cpp
18      int sum = 0;
19      for (int i = 0; i < n; ++i) {
20          if (w[i] > 0) {
21              G.add(S, i, w[i]);
22              sum += w[i];
23          }
24          else if (w[i] < 0) {
25              G.add(i, T, abs(w[i]));
26          }
27      }
28      for (auto &[u, v] : EV) { // You should make sure that
29          INF >  $\sum w_i$ 
30          G.add(u, v, INF);
31      }
32      int cut = G.flow(S, T);
33      return sum - cut;

```

6.20 Theorem

- 任意圖
 - 不能有孤點 · 最大匹配 + 最小邊覆蓋 = n - 點覆蓋的補集是獨立集。
最小點覆蓋 + 最大獨立集 = n
- 二分圖
 - 最小點覆蓋 = 最大匹配 = n - 最大獨立集
- 只有邊帶權的二分圖
 - w-vertex-cover (帶權點覆蓋): 每條邊的兩個連接點被選中的次數總和至少要是 w_e 。
 - w-weight matching (帶權匹配)
 - minimum vertex count of w-vertex-cover = maximum weight count of w-weight matching (一個點可以被選很多次 · 但邊不行)
- 點、邊都帶權的二分圖的定理
 - b-matching: 假設 v 的點權是 b_v · 那所有 v 的匹配邊 e 的權重都要滿足 $\sum w_e \leq b_v$ 。
 - The maximum w-weight of a b-matching equals the minimum b-weight of vertices in a w-vertex-cover.

7 Math

7.1 CRT m Coprime

```

1  vector<int> a, m;
2
3  int extgcd(int a, int b, int &x, int &y){
4      if (b==0){
5          x=1, y=0;
6          return a;
7      }
8
9      int ret=extgcd(b, a%b, y, x);
10     y-=a/b*x;
11     return ret;
12 }
13
14 // n = 有幾個式子 · 求解  $x \equiv a_i \pmod{m_i}$ 
15 int CRT(int n, vector<int> &a, vector<int> &m){
16     int p=1, ans=0;
17
18     vector<int> M(n), inv_M(n);
19
20     for (int i=0; i<n; i++) p*=m[i];
21     for (int i=0; i<n; i++){
22         M[i]=p/m[i];
23         int tmp;
24         extgcd(M[i], m[i], inv_M[i], tmp);
25         ans+=a[i]*inv_M[i]*M[i];
26         ans%=p;
27     }
28
29     return (ans%p+p)%p;
30 }

```

7.2 CRT m Not Coprime

```

1  int extgcd(int a, int b, int &x, int &y){
2      if (b==0){
3          x=1, y=0;
4          return a;
5      }
6
7      int ret=extgcd(b, a%b, y, x);
8      y-=a/b*x;
9      return ret;
10 }
11
12 // 對於方程組的式子兩兩求解
13 // {是否有解, {a, m}}
14 pair<bool, pair<int, int>> CRT(int a1, int m1, int a2, int m2)
15 {
16     int g=__gcd(m1, m2);
17     if ((a2-a1)%g!=0) return {0, {-1, -1}};
18
19     int x, y;
20     extgcd(m1, m2, x, y);
21
22     x=(a2-a1)*x/g; // 兩者不能相反
23     a1=x*m1+a1;

```

```

23     m1=m1*m2/g;
24     a1=(a1*m1+m1)%m1;
25     return {1, {a1, m1}};
26 }

```

7.3 Fraction

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  /// Fraction template starts ///
5  #define fraction_template_bonus_check
6  const long long ll_overflow_warning_value = (long long)(3e9);
7
8  long long gcd(long long a, long long b){
9      if(a == 0) return 0;
10     if(b == 0) return a;
11     if(a < b) return gcd(b,a);
12     return gcd(b, a%b);
13 }
14 struct frac{
15     long long a, b;
16     frac(long long _a = 0, long long _b = 1){
17         a = _a; b = _b;
18         if(b == 0){
19             cerr << "Error: division by zero\n";
20             cerr << "Called : Constructor(" << _a << ", " <<
                _b << ")\n";
21             return;
22         }
23         if(a == 0){b = 1; return;}
24         if(b < 0){a = -a; b = -b;}
25         long long gcd_ab = gcd(std::abs(a), b);
26         if(gcd_ab != 1){a /= gcd_ab; b /= gcd_ab;}
27
28         #ifdef fraction_template_bonus_check
29         if(std::abs(a) > ll_overflow_warning_value || b >
            ll_overflow_warning_value){
30             cerr << "Overflow warning : " << a << "/" << b <<
                "\n";
31         }
32         #endif // fraction_template_bonus_check
33     }
34     frac operator+(frac const &B){
35         return frac(a*(B.b)+(B.a)*b, b*(B.b));
36     }
37     frac operator-(frac const &B){
38         return frac(a*(B.b)-(B.a)*b, b*(B.b));
39     }
40     frac operator*(frac const &B){
41         return frac(a*(B.a), b*(B.b));
42     }
43     frac operator/(frac const &B){
44         return frac(a*(B.b), b*(B.a));
45     }
46
47     frac operator+=(frac const &B){
48         *this = frac(a*(B.b)+(B.a)*b, b*(B.b));
49     }
50     frac operator-=(frac const &B){
51         *this = frac(a*(B.b)-(B.a)*b, b*(B.b));
52     }
53     frac operator*=(frac const &B){
54         *this = frac(a*(B.a), b*(B.b));
55     }
56     frac operator/=(frac const &B){
57         *this = frac(a*(B.b), b*(B.a));
58     }
59
60     frac abs(){
61         a = std::abs(a);
62         return *this;
63     }

```

```

54 }
55
56 bool operator<(frac const &B){
57     return a*B.b < B.a*b;
58 }
59 bool operator<=(frac const &B){
60     return a*B.b <= B.a*b;
61 }
62 bool operator>(frac const &B){
63     return a*B.b > B.a*b;
64 }
65 bool operator>=(frac const &B){
66     return a*B.b >= B.a*b;
67 }
68 bool operator==(frac const &B){
69     return a * B.b == B.a * b;
70 }
71 bool operator!=(frac const &B){
72     return a * B.b != B.a * b;
73 };
74 ostream& operator<<(ostream &os, const frac& A){
75     os << A.a << "/" << A.b;
76     return os;
77 }
78 /// Fraction template ends ///
79
80 void test(frac A, frac B){
81     cout << "A = " << A << endl;
82     cout << "B = " << B << endl;
83     cout << endl;
84     cout << "A + B = " << A + B << endl;
85     cout << "A - B = " << A - B << endl;
86     cout << "A * B = " << A * B << endl;
87     cout << "A / B = " << A / B << endl;
88     cout << endl;
89     cout << "(A < B) = " << (A < B) << endl;
90     cout << "(A <= B) = " << (A <= B) << endl;
91     cout << "(A > B) = " << (A > B) << endl;
92     cout << "(A >= B) = " << (A >= B) << endl;
93     cout << "(A == B) = " << (A == B) << endl;
94     cout << "(A != B) = " << (A != B) << endl;
95     cout << "-----\n";
96     return;
97 }
98
99 int main(){
100     frac tmp1(-7, 2);
101     frac tmp2(5, 3);
102     test(tmp1, tmp2);
103
104     frac tmp3(-7);
105     frac tmp4(0);
106     test(tmp3, tmp4);
107     return 0;
108 }

```

7.4 Josephus Problem

```

1 // 有 n 個人，第偶數個報數的人被刪掉，問第 k 個被踢掉的是誰
2 int solve(int n, int k){
3     if (n==1) return 1;
4     if (k<=(n+1)/2){
5         if (2*k>n) return 2*k%n;
6         else return 2*k;
7     }else{
8         int res=solve(n/2, k-(n+1)/2);
9         if (n&1) return 2*res+1;
10        else return 2*res-1;

```

```

11 }
12 }

```

7.5 Lagrange any x

```

1 // init: (x1, y1), (x2, y2) in a vector
2 struct Lagrange{
3     int n;
4     vector<pair<int, int>> v;
5
6     Lagrange(vector<pair<int, int>> &_v){
7         n = _v.size();
8         v = _v;
9     }
10
11     // O(n^2 log MAX_A)
12     int solve(int x){
13         int ret = 0;
14         for (int i=0 ; i<n ; i++){
15             int now = v[i].second;
16             for (int j=0 ; j<n ; j++){
17                 if (i==j) continue;
18                 now *= ((x-v[j].first+MOD)%MOD);
19                 now %= MOD;
20                 now *= (qp((v[i].first-v[j].first+MOD)%MOD,
21                     MOD-2)+MOD)%MOD;
22                 now %= MOD;
23             }
24             ret = (ret+now)%MOD;
25         }
26         return ret;
27     }
28 };

```

7.6 Lagrange continuous x

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 const int MAX_N = 5e5 + 10;
5 const int mod = 1e9 + 7;
6
7 long long inv_fac[MAX_N];
8
9 inline int fp(long long x, int y) {
10     int ret = 1;
11     for (; y >= 1) {
12         ret = (y & 1) ? (ret * x % mod) : ret;
13         x = x * x % mod;
14     }
15     return ret;
16 }
17
18 // TO USE THIS TEMPLATE, YOU MUST MAKE SURE THAT THE MOD
19 // NUMBER IS A PRIME.
20 struct Lagrange {
21     /*
22      * Initialize a polynomial with f(x_0), f(x_0 + 1), ..., f(
23      * x_0 + n).

```

```

22     This determines a polynomial f(x) whose degree is at most
23     n.
24     Then you can call sample(x) and you get the value of f(x)
25     .
26     Complexity of init() and sample() are both O(n).
27 */
28 int m, shift; // m = n + 1
29 vector<int> v, mul;
30 // You can use this function if you don't have inv_fac array
31 // already.
32 void construct_inv_fac() {
33     long long fac = 1;
34     for (int i = 2; i < MAX_N; ++i) {
35         fac = fac * i % mod;
36     }
37     inv_fac[MAX_N - 1] = fp(fac, mod - 2);
38     for (int i = MAX_N - 1; i >= 1; --i) {
39         inv_fac[i - 1] = inv_fac[i] * i % mod;
40     }
41 // You call init() many times without having a second
42 // instance of this struct.
43 void init(int X_0, vector<int> &u) {
44     v = u;
45     shift = ((1 - X_0) % mod + mod) % mod;
46     if (v.size() == 1) v.push_back(v[0]);
47     m = v.size();
48     mul.resize(m);
49 // You can use sample(x) instead of sample(x % mod).
50 int sample(int x) {
51     x = ((long long)x + shift) % mod;
52     x = (x < 0) ? (x + mod) : x;
53     long long now = 1;
54     for (int i = m; i >= 1; --i) {
55         mul[i - 1] = now;
56         now = now * (x - i) % mod;
57     }
58     int ret = 0;
59     bool neg = (m - 1) & 1;
60     now = 1;
61     for (int i = 1; i <= m; ++i) {
62         int up = now * mul[i - 1] % mod;
63         int down = inv_fac[m - i] * inv_fac[i - 1] % mod;
64         int tmp = ((long long)v[i - 1] * up % mod) * down
65             % mod;
66         ret += (neg && tmp) ? (mod - tmp) : (tmp);
67         ret = (ret >= mod) ? (ret - mod) : ret;
68         now = now * (x - i) % mod;
69         neg ^= 1;
70     }
71     return ret;
72 }
73 };
74
75 int main() {
76     int n; cin >> n;
77     vector<int> v(n);
78     for (int i = 0; i < n; ++i) {
79         cin >> v[i];
80     }
81     Lagrange L;
82     L.construct_inv_fac();
83     L.init(0, v);
84     int x; cin >> x;
85     cout << L.sample(x);

```


83 }

7.7 Lucas's Theorem

```

1 // 對於很大的  $C^n_m$  對質數  $p$  取模，只要  $p$  不大就可以用。
2 int Lucas(int n, int m, int p){
3     if (m==0) return 1;
4     return (C(n%p, m%p, p)*Lucas(n/p, m/p, p)%p);
5 }

```

7.8 Matrix

```

1 struct Matrix{
2     int n, m;
3     vector<vector<int>> arr;
4
5     Matrix(int _n, int _m){
6         n = _n;
7         m = _m;
8         arr.assign(n, vector<int>(m));
9     }
10
11     vector<int> operator [] (int i){
12         return arr[i];
13     }
14
15     Matrix operator * (Matrix b){
16         Matrix ret(n, b.m);
17         for (int i=0 ; i<n ; i++){
18             for (int j=0 ; j<b.m ; j++){
19                 for (int k=0 ; k<m ; k++){
20                     ret.arr[i][j] += arr[i][k]*b.arr[k][j]%
21                         MOD;
22                     ret.arr[i][j] %= MOD;
23                 }
24             }
25             return ret;
26         }
27
28         Matrix pow(int p){
29             Matrix ret(n, n), mul = *this;
30             for (int i=0 ; i<n ; i++){
31                 ret.arr[i][i] = 1;
32             }
33
34             for ( ; p ; p>>=1){
35                 if (p&1) ret = ret*mul;
36                 mul = mul*mul;
37             }
38
39             return ret;
40         }
41
42         int det(){
43             vector<vector<int>> arr = this->arr;
44             bool flag = false;
45             for (int i=0 ; i<n ; i++){
46                 int target = -1;

```

```

48         for (int j=i ; j<n ; j++){
49             if (arr[j][i]){
50                 target = j;
51                 break;
52             }
53         }
54         if (target==-1) return 0;
55         if (i!=target){
56             swap(arr[i], arr[target]);
57             flag = !flag;
58         }
59
60         for (int j=i+1 ; j<n ; j++){
61             if (!arr[j][i]) continue;
62             int freq = arr[j][i]*qp(arr[i][i], MOD-2)%MOD
63                 ;
64             for (int k=i ; k<n ; k++){
65                 arr[j][k] -= freq*arr[i][k];
66                 arr[j][k] = (arr[j][k]%MOD+MOD)%MOD;
67             }
68         }
69
70         int ret = !flag ? 1 : MOD-1;
71         for (int i=0 ; i<n ; i++){
72             ret *= arr[i][i];
73             ret %= MOD;
74         }
75         return ret;
76     }
77 };

```

7.9 Matrix 01

```

1 const int MAX_N = (1LL<<12);
2 struct Matrix{
3     int n, m;
4     vector<bitset<MAX_N>> arr;
5
6     Matrix(int _n, int _m){
7         n = _n;
8         m = _m;
9         arr.resize(n);
10    }
11
12    Matrix operator * (Matrix b){
13        Matrix b_t(b.m, b.n);
14        for (int i=0 ; i<b.n ; i++){
15            for (int j=0 ; j<b.m ; j++){
16                b_t.arr[j][i] = b.arr[i][j];
17            }
18        }
19
20        Matrix ret(n, b.m);
21        for (int i=0 ; i<n ; i++){
22            for (int j=0 ; j<b.m ; j++){
23                ret.arr[i][j] = ((arr[i]&b_t.arr[j]).count()
24                    &1);
25            }
26        }
27        return ret;
28    }
};

```

7.10 Miller Rabin

```

1 // O(log n)
2 typedef Uint unsigned long long
3 Uint modmul(Uint a, Uint b, Uint m) {
4     int ret = a*b - m*(Uint)((long double)a*b/m);
5     return ret + m*(ret < 0) - m*(ret>=(int)m);
6 }
7
8 int qp(int b, int p, int m){
9     int ret = 1;
10    for ( ; p ; p>>=1){
11        if (p&1){
12            ret = modmul(ret, b, m);
13        }
14        b = modmul(b, b, m);
15    }
16    return ret;
17 }
18
19 // ed23aa
20 vector<int> llsprp = {2, 325, 9375, 28178, 450775, 9780504,
21     1795265022};
22 bool isprime(int n, vector<int> sprp = llsprp){
23     if (n==2) return 1;
24     if (n<2 || n%2==0) return 0;
25
26     int t = 0;
27     int u = n-1;
28     for ( ; u%2==0 ; t++) u>>=1;
29
30     for (int i=0 ; i<sprp.size() ; i++){
31         int a = sprp[i]%n;
32         if (a==0 || a==1 || a==n-1) continue;
33         int x = qp(a, u, n);
34         if (x==1 || x==n-1) continue;
35         for (int j=0 ; j<t ; j++){
36             x = modmul(x, x, n);
37             if (x==1) return 0;
38             if (x==n-1) break;
39         }
40         if (x==n-1) continue;
41         return 0;
42     }
43
44     return 1;
45 }

```

7.11 Pollard Rho

```

1 mt19937 seed(chrono::steady_clock::now().time_since_epoch().
2     count());
3 int rnd(int l, int r){
4     return uniform_int_distribution<int>(l, r)(seed);
5 }
6 // O(n^{1/4}) 回傳 1 或自己的因數、記得先判斷 n 是不是質數
7 // (用 Miller-Rabin)
8 // c1670c
9 int Pollard_Rho(int n){

```



```

9 | int s = 0, t = 0;
10 | int c = rnd(1, n-1);
11 |
12 | int step = 0, goal = 1;
13 | int val = 1;
14 |
15 | for (goal=1 ; ; goal<=1, s=t, val=1){
16 |     for (step=1 ; step<=goal ; step++){
17 |
18 |         t = ((__int128)t*t+c)%n;
19 |         val = ((__int128)val*abs(t-s)%n;
20 |
21 |         if ((step % 127) == 0){
22 |             int d = __gcd(val, n);
23 |             if (d>1) return d;
24 |         }
25 |     }
26 |
27 |     int d = __gcd(val, n);
28 |     if (d>1) return d;
29 | }
30 | }

```

7.12 Quick Pow

```

1 | int qp(int b, int p, int m = MOD){
2 |     int ret = 1;
3 |     for ( ; p ; p>>=1){
4 |         if (p&1) ret = ret*b%m;
5 |         b = b*b%m;
6 |     }
7 |     return ret;
8 | }

```

7.13 數論分塊

```

1 | /*
2 | 時間複雜度為 O(sqrt(n))
3 | 區間為 [L, r]
4 | */
5 | for(int i=1 ; i<=n ; i++){
6 |     int l = i, r = n/(n/i);
7 |     i = r;
8 |     ans.push_back(r);
9 | }

```

7.14 最大質因數

```

1 | void max_fac(int n, int &ret){
2 |     if (n<=ret || n<2) return;
3 |     if (isprime(n)){
4 |         ret = max(ret, n);
5 |         return;
6 |     }
7 |
8 |     int p = Pollard_Rho(n);

```

```

9 |     max_fac(p, ret), max_fac(n/p, ret);
10 | }

```

7.15 歐拉公式

```

1 | // phi(n) = 小於 n 並與 n 互質的正整數數量。
2 | // O(sqrt(n)) · 回傳 phi(n)
3 | int phi(int n){
4 |     int ret = n;
5 |
6 |     for (int i=2 ; i*i<=n ; i++){
7 |         if (n%i==0){
8 |             while (n%i==0) n /= i;
9 |             ret = ret*(i-1)/i;
10 |        }
11 |    }
12 |    if (n>1) ret = ret*(n-1)/n;
13 |
14 |    return ret;
15 | }
16 |
17 | // O(n Log n) · 回傳 1~n 的 phi 值
18 | vector<int> phi_1_to_n(int n){
19 |     vector<int> phi(n+1);
20 |     phi[0]=0;
21 |     phi[1]=1;
22 |
23 |     for (int i=2 ; i<=n ; i++){
24 |         phi[i]=i-1;
25 |     }
26 |
27 |     for (int i=2 ; i<=n ; i++){
28 |         for (int j=2*i ; j<=n ; j+=i){ // 枚舉所有倍數
29 |             phi[j]-=phi[i];
30 |         }
31 |     }
32 |
33 |     return phi;
34 | }

```

7.16 線性篩

```

1 | const int MAX_N = 5e5;
2 |
3 | // lpf[i] = i 的最小質因數
4 | vector<int> prime, lpf(MAX_N);
5 |
6 | void prime_init(){
7 |     for (int i=2 ; i<MAX_N ; i++){
8 |         if (lpf[i]==0){
9 |             lpf[i] = i;
10 |            prime.push_back(i);
11 |        }
12 |
13 |        for (int j : prime){
14 |            if (i*j>=MAX_N) break;
15 |            lpf[i*j] = j;
16 |            if (i%j==0) break;
17 |        }

```

```

18 |     }
19 | }

```

7.17 Burnside's Lemma

$$\sum_{k=1}^n \frac{c(k)}{n}$$

- n : 有多少種置換方式 (例如: 旋轉方式)
- $c(k)$: 所有可能中, 經過 k 次旋轉後, 仍不會和別人相同的方式的數量

7.18 Catalan Number

任意括號序列: $C_n = \frac{1}{n+1} \binom{2n}{n}$

7.19 Matrix Tree Theorem

目標: 給定一張無向圖, 問他的生成樹數量。
方法: 先把所有自環刪掉, 定義 Q 為以下矩陣

$$Q_{i,j} = \begin{cases} \deg(v_i) & \text{if } i = j \\ -(邊v_i v_j \text{ 的數量}) & \text{otherwise} \end{cases}$$

接著刪掉 Q 的第一個 row 跟 column, 它的 determinant 就是答案。
目標: 給定一張有向圖, 問他的以 r 為根, 可以走到所有點生成樹數量。

方法: 先把所有自環刪掉, 定義 Q 為以下矩陣

$$Q_{i,j} = \begin{cases} \deg_{in}(v_i) & \text{if } i = j \\ -(邊v_i v_j \text{ 的數量}) & \text{otherwise} \end{cases}$$

接著刪掉 Q 的第 r 個 row 跟 column, 它的 determinant 就是答案。

7.20 Stirling's formula

$$n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$$

7.21 Theorem

1. $1 \sim x$ 質數的數量 $\approx \frac{x}{\ln x}$
2. x 的因數的數量 $\approx x^{\frac{1}{3}}$
3. x 的質因數的數量 $\approx \log \log x$
4. p is a prime number $\Leftrightarrow (p-1)! \equiv -1 \pmod{p}$
5. 每個正整數都可以表示成四個整數的平方和
6. 任何大於 2 的整數都可以表示成兩個質數的和

7.22 二元一次方程式

$$\begin{cases} ax + by = c \\ dx + ey = f \end{cases} = \begin{cases} x = \frac{ed-bf}{ad-bc} \\ y = \frac{af-ec}{ad-bc} \end{cases}$$

若 $x = \frac{0}{0}$ 且 $y = \frac{0}{0}$ ，則代表無限多組解。若 $x = \frac{*}{0}$ 且 $y = \frac{*}{0}$ ，則代表無解。

7.23 歐拉定理

若 a, m 互質，則：

$$a^n \equiv a^{n \bmod \varphi(m)} \pmod{m}$$

若 a, m 不互質，則：

$$a^n \equiv a^{\varphi(m) + [n \bmod \varphi(m)]} \pmod{m}$$

7.24 錯排公式

錯排公式：(n 個人中，每個人皆不再原來位置的組合數)

$$dp_i = \begin{cases} 1 & i = 0 \\ 0 & i = 1 \\ (i-1)(dp_{i-1} + dp_{i-2}) & \text{otherwise} \end{cases}$$

8 String

8.1 Hash

```
1 mt19937 seed(chrono::steady_clock::now().time_since_epoch().
  count());
2 int rng(int l, int r){
3     return uniform_int_distribution<int>(l, r)(seed);
4 }
5 int A = rng(1e5, 8e8);
6 const int B = 1e9+7;
7
8 // 2f6192
9 struct RollingHash{
10     vector<int> Pow, Pre;
11     RollingHash(string s = ""){
12         Pow.resize(s.size());
13         Pre.resize(s.size());
14
15         for (int i=0 ; i<s.size() ; i++){
16             if (i==0){
17                 Pow[i] = 1;
18                 Pre[i] = s[i];
19             }else{
20                 Pow[i] = Pow[i-1]*A%B;
21                 Pre[i] = (Pre[i-1]*A+s[i])%B;
22             }
23         }
24 }
```

```
25     return;
26 }
27
28 int get(int l, int r){ // 取得 [l, r] 的數值
29     if (l==0) return Pre[r];
30     int res = (Pre[r]-Pre[l-1]*Pow[r-l+1])%B;
31     if (res<0) res += B;
32     return res;
33 }
34 };
```

8.2 KMP

```
1 // 給一個字串 S，定義函數 \pi(i) = k 代表 S[1 ... k] = S[i-k
  +1 ... i] (最長真前後綴)
2 // e5b7ce
3 vector<int> KMP(string &s){
4     int n = s.size();
5     vector<int> ret(n);
6     for (int i=1 ; i<n ; i++){
7         int j = ret[i-1];
8         while (j>0 && s[i]!=s[j]) j = ret[j-1];
9         j += (s[i]==s[j]);
10        ret[i] = j;
11    }
12    return ret;
13 }
```

8.3 Manacher

```
1 string Manacher(string str) {
2     string tmp = "$#";
3     for(char i : str) {
4         tmp += i;
5         tmp += '#';
6     }
7
8     vector<int> p(tmp.size(), 0);
9     int mx = 0, id = 0, len = 0, center = 0;
10    for(int i=1 ; i<(int)tmp.size() ; i++) {
11        p[i] = mx > i ? min(p[id*2-i], mx-i) : 1;
12
13        while(tmp[i+p[i]] == tmp[i-p[i]]) p[i]++;
14        if(mx<i+p[i]) mx = i+p[i], id = i;
15        if(len<p[i]) len = p[i], center = i;
16    }
17    return str.substr((center-len)/2, len-1);
18 }
```

8.4 Min Rotation

```
1 // 9d296f
2 int minRotation(string s) {
3     int a=0, N=SZ(s); s += s;
4     for (int b=0 ; b<N ; b++){
5         for (int k=0 ; k<N ; k++){
```

```
6         if (a+k == b || s[a+k] < s[b+k]) {b += max(0LL, k
          -1); break;}
7         if (s[a+k] > s[b+k]) {a = b; break;}
8     }
9 }
10 return a;
11 }
```

8.5 Suffix Array

```
1 // 注意，當 |s|=1 時，lcp 不會有值，務必測試 |s|=1 的 case
2 struct SuffixArray {
3     string s;
4     vector<int> sa, lcp;
5
6     // 69ced9
7     SuffixArray(string _s, int lim = 256) {
8         s = _s;
9         int n = s.size()+1, k = 0, a, b;
10        vector<int> x(s.begin(), s.end()), y(n), ws(max(n,
          lim)), rank(n);
11        x.push_back(0);
12        sa = lcp = y;
13        iota(sa.begin(), sa.end(), 0);
14        for (int j=0, p=0 ; p<n ; j=max(1LL, j*2), lim=p) {
15            p = j;
16            iota(y.begin(), y.end(), n-j);
17            for (int i=0 ; i<n ; i++) if (sa[i] >= j) y[p++]
              = sa[i] - j;
18            fill(ws.begin(), ws.end(), 0);
19            for (int i=0 ; i<n ; i++) ws[x[i]]++;
20            for (int i=1 ; i<lim ; i++) ws[i] += ws[i-1];
21            for (int i = n; i--;) sa[--ws[x[i]]] = y[i];
22            swap(x, y), p = 1, x[sa[0]] = 0;
23            for (int i=1 ; i<n ; i++){
24                a = sa[i-1];
25                b = sa[i];
26                x[b] = (y[a] == y[b] && y[a+j] == y[b+j])
                  ? p-1 : p++;
27            }
28        }
29
30        for (int i=1 ; i<n ; i++) rank[sa[i]] = i;
31        for (int i=0, j ; i<n-1 ; lcp[rank[i++]] = k)
32            for (k && k--, j=sa[rank[i]-1] ; i+k<s.size() &&
              j+k<s.size() && s[i+k]==s[j+k] ; k++);
33        sa.erase(sa.begin());
34        lcp.erase(lcp.begin(), lcp.begin()+2);
35    }
36
37    // f49583
38    vector<int> pos; // pos[i] = i 這個值在 pos 的哪個地方
39    SparseTable st;
40    void init_lcp(){
41        pos.resize(sa.size());
42        for (int i=0 ; i<sa.size() ; i++){
43            pos[sa[i]] = i;
44        }
45        if (lcp.size()){
46            st.build(lcp);
47        }
48    }
49 }
```

```

50 // 用之前記得 init
51 // 回傳 [l1, r1] 跟 [l2, r2] 的 lcp · 0-based
52 int get_lcp(int l1, int r1, int l2, int r2){
53     int pos_1 = pos[l1], len_1 = r1-l1+1;
54     int pos_2 = pos[l2], len_2 = r2-l2+1;
55     if (pos_1>pos_2){
56         swap(pos_1, pos_2);
57         swap(len_1, len_2);
58     }
59
60     if (l1==l2){
61         return min(len_1, len_2);
62     }else{
63         return min({st.query(pos_1, pos_2), len_1, len_2});
64     }
65 }
66
67 // 檢查 [l1, r1] 跟 [l2, r2] 的大小關係 · 0-based
68 // 如果前者小於後者 · 就回傳 <0 · 相等就回傳 =0 · 否則回傳 >0
69 // 5b8db0
70 int substring_cmp(int l1, int r1, int l2, int r2){
71     int len_1 = r1-l1+1;
72     int len_2 = r2-l2+1;
73     int res = get_lcp(l1, r1, l2, r2);
74
75     if (res<len_1 && res<len_2){
76         return s[l1+res]-s[l2+res];
77     }else if (len_1==res && len_2==res){
78         // 如果不需要以 index 作為次要排序參數 · 這裡要回傳 0
79         return l1-l2;
80     }else{
81         return len_1==res ? -1 : 1;
82     }
83 }
84
85 // 對於位置在 <=p 的後綴 · 找離他左邊/右邊最接近位置 >p 的後綴的 lcp · 0-based
86 // pre[i] = s[i] 離他左邊最接近位置 >p 的後綴的 lcp · 0-based
87 // suf[i] = s[i] 離他右邊最接近位置 >p 的後綴的 lcp · 0-based
88 // da12fa
89 pair<vector<int>, vector<int>> get_left_and_right_lcp(int p){
90     vector<int> pre(p+1);
91     vector<int> suf(p+1);
92
93     { // build pre
94         int now = 0;
95         for (int i=0 ; i<s.size() ; i++){
96             if (sa[i]<=p){
97                 pre[sa[i]] = now;
98                 if (i<lcp.size()) now = min(now, lcp[i]);
99             }else{
100                 if (i<lcp.size()) now = lcp[i];
101             }
102         }
103     }
104     { // build suf
105         int now = 0;
106         for (int i=s.size()-1 ; i>=0 ; i--){

```

```

107             if (sa[i]<=p){
108                 suf[sa[i]] = now;
109                 if (i-1>=0) now = min(now, lcp[i-1]);
110             }else{
111                 if (i-1>=0) now = lcp[i-1];
112             }
113         }
114     }
115
116     return {pre, suf};
117 }
118 };

```

8.6 Z Algorithm

```

1 // 定義一個長度為 n 的文本為 T · 則陣列 Z 的 Z[i] 代表 T[0:n]
   // 和 T[i:n] 最長共同前綴
2 // bcfbd6
3 vector<int> z_function(string s){
4     vector<int> ret(s.size());
5     int ll = 0, rr = 0;
6
7     for (int i=1 ; i<s.size() ; i++){
8         int j = 0;
9
10        if (i<rr) j = min(ret[i-ll], rr-i);
11        while (s[j]==s[i+j]) j++;
12        ret[i] = j;
13
14        if (i+j>rr){
15            ll = i;
16            rr = i+j;
17        }
18    }
19
20    ret[0] = s.size();
21    return ret;
22 }

```

8.7 k-th Substring1

```

1 // 回傳 s 所有子字串 (完全不同) 中 · 第 k 大的
2 string k_th_substring(string &s, int k){
3     int n = s.size();
4     SuffixArray sa(s);
5     sa.init_lcp();
6
7     int prePrefix = 0, nowRank = 0;
8     for (int i=0 ; i<n ; i++){
9         int len = n-sa[i];
10        int add = len-prePrefix;
11
12        if (nowRank+add>=k){
13            return s.substr(sa[i], prePrefix+k-nowRank);
14        }
15
16        prePrefix = sa.lcp[i];
17        nowRank += add;
18    }
19 }

```