

# Contents

<b>1 Misc</b>	<b>2</b>	3.11 Trie . . . . .	8	<b>7 Math</b>	<b>19</b>
1.1 Custom Set PQ Sort . . . . .	2	<b>4 Dynamic-Programming</b>	<b>8</b>	7.1 CRT m Coprime . . . . .	19
1.2 Default Code New . . . . .	2	4.1 Digit DP . . . . .	8	7.2 CRT m Not Coprime . . . . .	20
1.3 Default Code Old . . . . .	2	4.2 SOS DP . . . . .	8	7.3 Fraction . . . . .	20
1.4 Enumerate Subset . . . . .	2	4.3 Integer Partition . . . . .	8	7.4 Josephus Problem . . . . .	20
1.5 Fast Input . . . . .	2	<b>5 Geometry</b>	<b>8</b>	7.5 Lagrange any x . . . . .	20
1.6 Radix Sort . . . . .	2	5.1 Geometry Struct . . . . .	8	7.6 Lagrange continuous x . . . . .	21
1.7 Xor Basis . . . . .	2	5.2 Geometry 卦長 . . . . .	10	7.7 Lucas's Theorem . . . . .	21
1.8 random int . . . . .	3	5.3 Pick's Theorem . . . . .	12	7.8 Matrix . . . . .	21
1.9 hash command . . . . .	3	<b>6 Graph</b>	<b>13</b>	7.9 Matrix 01 . . . . .	22
1.10 run . . . . .	3	6.1 2-SAT . . . . .	13	7.10 Miller Rabin . . . . .	22
1.11 setup . . . . .	3	6.2 Augment Path . . . . .	13	7.11 Pollard Rho . . . . .	22
<b>2 Convolution</b>	<b>3</b>	6.3 Bridge BCC . . . . .	13	7.12 Quick Pow . . . . .	22
2.1 FFT any mod . . . . .	3	6.4 Cut BCC . . . . .	14	7.13 數論分塊 . . . . .	22
2.2 FFT new . . . . .	3	6.5 Dijkstra . . . . .	14	7.14 最大質因數 . . . . .	22
2.3 FFT old . . . . .	4	6.6 Dinic . . . . .	14	7.15 歐拉公式 . . . . .	23
2.4 NTT mod 998244353 . . . . .	4	6.7 Dominator Tree . . . . .	14	7.16 線性篩 . . . . .	23
<b>3 Data-Structure</b>	<b>4</b>	6.8 Find Bridge . . . . .	15	7.17 Burnside's Lemma . . . . .	23
3.1 BIT . . . . .	4	6.9 HLD . . . . .	15	7.18 Catalan Number . . . . .	23
3.2 Disjoint Set Persistent . . . . .	5	6.10 Kosaraju . . . . .	15	7.19 Matrix Tree Theorem . . . . .	23
3.3 PBDS GP Hash Table . . . . .	5	6.11 Kuhn Munkres . . . . .	16	7.20 Stirling's formula . . . . .	23
3.4 PBDS Order Set . . . . .	5	6.12 LCA . . . . .	16	7.21 Theorem . . . . .	23
3.5 Segment Tree Add Set . . . . .	5	6.13 MCMF . . . . .	17	7.22 二元一次方程式 . . . . .	23
3.6 Segment Tree Li Chao Line . . . . .	6	6.14 Tarjan . . . . .	17	7.23 歐拉定理 . . . . .	23
3.7 Segment Tree Li Chao Segment . . . . .	6	6.15 Tarjan Find AP . . . . .	17	7.24 錯排公式 . . . . .	23
3.8 Segment Tree Persistent . . . . .	7	6.16 Tree Isomorphism . . . . .	18	<b>8 String</b>	<b>23</b>
3.9 Sparse Table . . . . .	7	6.17 圓方樹 . . . . .	18	8.1 Hash . . . . .	23
3.10 Treap . . . . .	7	6.18 最大權閉合圖 . . . . .	19	8.2 KMP . . . . .	24
		6.19 Theorem . . . . .	19	8.3 Manacher . . . . .	24
				8.4 Min Rotation . . . . .	24
				8.5 Suffix Array . . . . .	24
				8.6 Z Algorithm . . . . .	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 Radix Sort

```

1 // 值域限制:  $0 \sim 1073741823(2^{30}-1)$ 
2 inline void radix_sort(vector<int> &a, int n){
3     static int cnt[32768] = {0};
4     vector<int> tmpa(n);
5     for(int i = 0; i < n; ++i)
6         ++cnt[a[i] & 32767];
7     for(int i = 1; i < 32768; ++i)
8         cnt[i] += cnt[i-1];
9     static int temp;
10    for(int i = n-1; i >= 0; --i){
11        temp = a[i] & 32767;
12        --cnt[temp];
13        tmpa[cnt[temp]] = a[i];
14    }
15
16    static int cnt2[32768] = {0};
17    for(int i = 0; i < n; ++i)
18        ++cnt2[(tmpa[i]>>15)];
19    for(int i = 1; i < 32768; ++i)
20        cnt2[i] += cnt2[i-1];
21
22    for(int i = n-1; i >= 0; --i){
23        temp = (tmpa[i]>>15);
24        --cnt2[temp];
25        a[cnt2[temp]] = tmpa[i];
26    }
27    return;
28 }

```

## 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  $2^{\{basis\} \text{ 的大小}}$ 
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 hash command

```

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

```

## 1.10 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()

```

## 1.11 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=k} 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(
          outs[i])+0.5);
70         int bv = (int)(imag(outl[i])+0.5) + (int)(real(outs[i]
          ))+0.5);
71         res[i] = ((av%MOD*cut+bv) % MOD*cut+cv) % MOD;
72     }
73
74     return res;
75 }

```

### 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     return;
36 }
37 // 39029d
38 vector<double> PolyMul(const vector<double> a, const vector<
39     double> b){
40     if (a.empty() || b.empty()) return {};
41     vector<double> res(a.size()+b.size()-1);
42     int L = 32 - __builtin_clz(res.size()), n = 1 << L;
43     vector<cd> in(n), out(n);
44
45     copy(a.begin(), a.end(), begin(in));
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 }

```

```

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

```

```

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);
45
46     vector<int> L(a), R(b), out(n);
47     L.resize(n), R.resize(n);
48     NTT(L), NTT(R);
49     for (int i=0 ; i<n ; i++){
50         out[-i&(n-1)] = L[i]*R[i]%MOD*inv%MOD;
51     }
52     NTT(out);
53
54     out.resize(s);
55     return out;
56 }

```

## 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;

```

## 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++){

```

## 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            int sz1 = sz.query_version(a, a+1, arr.version.size()
33                (-1).val;
34            int sz2 = sz.query_version(b, b+1, arr.version.size()
35                (-1).val;
36
37            if (sz1<sz2){
38                arr.update_version(a, b, arr.version.size()
39                    (-1);
40                sz.update_version(b, sz1+sz2, arr.version.size()
41                    (-1);
42            }else{
43                arr.update_version(b, a, arr.version.size()
44                    (-1);
45                sz.update_version(a, sz1+sz2, arr.version.size()
46                    (-1);
47            }
48            return true;
49        }
50    }
51 }

```

```

44     return false;
45 }
46 };

```

## 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    }

```

```

76 void set(int ql, int qr, int val, int idx=0, int ll=0,
77         int rr=n){
78     push(idx, ll, rr);
79     if (rr<=ql || qr<=ll) return;
80     if (ql<=ll && rr<=qr){
81         arr[idx].add_tag = 0;
82         arr[idx].set_tag = val;
83         push(idx, ll, rr);
84         return;
85     }
86     int mid = (ll+rr)/2;
87     set(ql, qr, val, idx*2+1, ll, mid);
88     set(ql, qr, val, idx*2+2, mid, rr);
89     arr[idx] = pull(arr[idx*2+1], arr[idx*2+2]);
90 }
91
92 node query(int ql, int qr, int idx = 0, int ll = 0, int
93           rr = n){
94     push(idx, ll, rr);
95     if (rr<=ql || qr<=ll) return node();
96     if (ql<=ll && rr<=qr) return arr[idx];
97
98     int mid = (ll+rr)/2;
99     return pull(query(ql, qr, idx*2+1, ll, mid), query(ql
100                  , qr, idx*2+2, mid, rr));

```

### 3.6 Segment Tree Li Chao Line

```

1  /*
2  全部都是  $\theta$ -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){

```

```

31         if (val.y(ll)<arr[idx].y(ll)){
32             arr[idx] = val;
33         }
34         return;
35     }
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
57     int mid = (ll+rr)/2;
58     if (x<mid){
59         return min(arr[idx].y(x), query(x, idx*2+1, ll,
60             mid));
61     }else{
62         return min(arr[idx].y(x), query(x, idx*2+2, mid,
63             rr));
64     }
65 }
66
67 };

```

### 3.7 Segment Tree Li Chao Segment

```

1  /*
2  全部都是  $\theta$ -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
24     void update_segment(Node val, int ql, int qr, int idx =
25           0, int ll = 0, int rr = MAX_V){
26         if (rr-ll==0) return;
27         if (rr-ll==1) return;
28         if (ql<=ll && rr<=qr){
29             update(val, idx, ll, rr);
30             return;
31         }
32
33         int mid = (ll+rr)/2;
34         update_segment(val, ql, qr, idx*2+1, ll, mid);
35         update_segment(val, ql, qr, idx*2+2, mid, rr);
36         return;
37     }
38
39     int query(int x, int idx = 0, int ll = 0, int rr = MAX_V)
40     {
41         if (rr-ll==0) return INF;
42         if (rr-ll==1){
43             return arr[idx].y(ll);
44         }
45
46         int mid = (ll+rr)/2;
47         if (x<mid){
48             return min(arr[idx].y(x), query(x, idx*2+1, ll,
49                 mid));
50         }else{
51             return min(arr[idx].y(x), query(x, idx*2+2, mid,
52                 rr));
53         }
54     }
55 }
56
57 };

```

```

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
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     if (rr-ll==1) return;
56     if (ql<=ll && rr<=qr){
57         update(val, idx, ll, rr);
58         return;
59     }
60
61     int mid = (ll+rr)/2;
62     update_segment(val, ql, qr, idx*2+1, ll, mid);
63     update_segment(val, ql, qr, idx*2+2, mid, rr);
64     return;
65 }
66
67 int query(int x, int idx = 0, int ll = 0, int rr = MAX_V)
68 {
69     if (rr-ll==0) return INF;
70     if (rr-ll==1){
71         return arr[idx].y(ll);
72     }
73
74     int mid = (ll+rr)/2;
75     if (x<mid){
76         return min(arr[idx].y(x), query(x, idx*2+1, ll,
77             mid));
78     }else{
79         return min(arr[idx].y(x), query(x, idx*2+2, mid,
80             rr));
81     }
82 }
83
84 };

```

### 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){

```

```

61         arr[node_cnt] = arr[now.lc];
62         now.lc = node_cnt;
63         node_cnt++;
64         update(pos, val, now.lc, ll, mid);
65     }else{
66         arr[node_cnt] = arr[now.rc];
67         now.rc = node_cnt;
68         node_cnt++;
69         update(pos, val, now.rc, mid, rr);
70     }
71     pull(now, arr[now.lc], arr[now.rc]);
72     return;
73 }
74
75 void update_version(int pos, int val, int ver){
76     update(pos, val, version[ver]);
77 }
78
79 Node query(int ql, int qr, int idx, int ll = 0, int rr =
80     n){
81     auto &now = arr[idx];
82
83     if (ql<=ll && rr<=qr) return now;
84     if (rr<=ql || qr<=ll) return Node();
85
86     int mid = (ll+rr)/2;
87
88     Node ret;
89     pull(ret, query(ql, qr, now.lc, ll, mid), query(ql,
90         qr, now.rc, mid, rr));
91     return ret;
92 }
93
94 Node query_version(int ql, int qr, int ver){
95     return query(ql, qr, version[ver]);
96 }
97
98 void clone_version(int ver){
99     version.push_back(node_cnt);
100     arr[node_cnt] = arr[version[ver]];
101     node_cnt++;
102 }
103
104 };

```

### 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
17     // 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 pair<Treap*, Treap*> split(Treap *&t, int k){ // 1-based <前
30     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
45 // functions
46 Treap* build(vector<int> v){
47     Treap* ret;
48     for (int i=0 ; i<SZ(v) ; i++){
49         ret = merge(ret, new Treap(v[i]));
50     }
51     return ret;
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 } tr;
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  $O(n \cdot 2^n)$ 
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
25    // 判斷 ab 到 ac 的方向: {1:逆時鐘,0:重疊,-1:順時鐘}
26    friend int ori(point a, point b, point c) {
27        return sign((b-a)^(c-a));
28    }
29
30    friend bool btw(point a, point b, point c) {
31        return ori(a, b, c) == 0 && sign((a-c)*(b-c)) <= 0;
32    }
33
34    // 判斷線段 ab, cd 是否相交
35    friend bool banana(point a, point b, point c, point d) {
36        int s1 = ori(a, b, c);
37        int s2 = ori(a, b, d);
38        int s3 = ori(c, d, a);
39    }

```



```

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
47     .x}; }
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     int ori(point<T> &p) {
65         return sign((p2-p1) ^ (p-p1));
66     }
67     // 判斷直線斜率是否相同
68     bool parallel(line &l) {
69         return ((p1-p2) ^ (l.p1-l.p2)) == 0;
70     }
71     // 兩直線交點
72     point<long double> line_intersection(line &l) {
73         using P = point<long double>;
74         point<T> a = p2-p1, b = l.p2-l.p1, s = l.p1-p1;
75         return P(p1.x, p1.y) + P(a.x, a.y) * (((long double)(s^b))
76             / (a^b));
77     }
78 };
79
80 template<typename T>
81 struct polygon {
82     vector<point<T>> v;
83     polygon() {}
84     polygon(const vector<point<T>> &u) : v(u) {}
85     // simple 為 true 的時候會回傳任意三點不共線的凸包
86     void make_convex_hull(int simple) {
87         auto cmp = [&](point<T> &p, point<T> &q) {
88             return (p.x == q.x) ? (p.y < q.y) : (p.x < q.x);
89         };
90         simple = (bool)simple;
91         sort(v.begin(), v.end(), cmp);
92         v.resize(unique(v.begin(), v.end()) - v.begin());
93         vector<point<T>> hull;
94         for (int t = 0; t < 2; ++t){
95             int sz = hull.size();
96             for (auto &i:v) {
97                 while (hull.size() >= sz+2 && ori(hull[hull.
98                     size()-2], hull.back(), i) < simple) {
99                     hull.pop_back();
100                 }
101                 hull.push_back(i);
102             }
103             hull.pop_back();
104             reverse(v.begin(), v.end());
105             swap(hull, v);
106         }
107     }
108     // 可以在有 n 個點的簡單多邊形內·用 O(n) 判斷一個點：
109     // {1：在多邊形內，0：在多邊形上，-1：在多邊形外}
110     int in_polygon(point<T> a){
111         const T MAX_POS = 1e9 + 5; // [記得修改] 座標的最大值
112         point<T> pre = v.back(), b(MAX_POS, a.y + 1);
113         int cnt = 0;
114
115         for (auto &i:v) {
116             if (btw(pre, i, a)) return 0;
117             if (banana(a, b, pre, i)) cnt++;
118             pre = i;
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
123     /// 共線的凸包 ///
124     // 可以在有 n 個點的凸包內·用 O(Log n) 判斷一個點：
125     // {1：在凸包內，0：在凸包邊上，-1：在凸包外}
126     int in_convex(point<T> p) {
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) ? 1 : 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
172     /// 定表示線段沒有通過凸包內部 ///
173     int segment_across_convex(line<T> L) {
174         point<T> p(L.a, L.b); // 記得 L 要 build
175         auto gt = [&](int neg) {
176             auto f = [&](int x, int y) {
177                 return sign((v[x] - v[y]) * p) == neg;
178             };
179             return cycle_search(f);
180         };
181         int i = gt(1), j = gt(-1), n = v.size();
182         T x = -(v[i] * p), y = -(v[j] * p);
183         if (L.c < x || y < L.c) return -1;
184         if (L.c == x || L.c == y) return 0;
185
186         if (i > j) swap(i, j);
187         auto g = [&](int x, int lim) {
188             int now = 0, nxt;
189             for (int i = 1 << __lg(lim); i > 0; i /= 2) {
190                 if (now + i > lim) continue;
191                 nxt = (x + i) % n;
192                 if (L.ori(v[x], * L.ori(v[nxt]) >= 0) {
193                     x = nxt;
194                     now += i;
195                 } // ↓ BE CAREFUL
196                 return -(ori(v[x], v[(x + 1) % n], L.p1) * ori(v[
197                     x], v[(x + 1) % n], L.p2));
198             }
199             return max(g(i, j - i), g(j, n - (j - i)));
200         };
201         // 可以在有 n 個點的凸包內·用 O(Log n) 判斷一個線段：
202         // {1：線段上存在某一點位於凸包內部 (邊上不算)，
203         // 0：線段上存在某一點碰到凸包的邊但線段上任一點均不在凸包
204         // 內部，
205         // -1：線段完全在凸包外面}
206         int segment_pass_convex_interior(line<T> L) {
207             if (in_convex(L.p1) == 1 || in_convex(L.p2) == 1)
208                 return 1;
209             point<T> p(L.a, L.b); // 記得 L 要 build
210             auto gt = [&](int neg) {
211                 auto f = [&](int x, int y) {
212                     return sign((v[x] - v[y]) * p) == neg;
213                 };
214                 return cycle_search(f);
215             };
216             int i = gt(1), j = gt(-1), n = v.size();
217             T x = -(v[i] * p), y = -(v[j] * p);
218             if (L.c < x || y < L.c) return -1;
219             if (L.c == x || L.c == y) return 0;
220
221             if (i > j) swap(i, j);
222             auto g = [&](int x, int lim) {
223                 int now = 0, nxt;
224                 for (int i = 1 << __lg(lim); i > 0; i /= 2) {

```

```

220         if (now + i > lim) continue;
221         nxt = (x + i) % n;
222         if (L.ori(v[x]) * L.ori(v[nxt]) > 0) {
223             x = nxt;
224             now += i;
225         }
226     } // ↓ BE CAREFUL
227     return -(ori(v[x], v[(x + 1) % n], L.p1) * ori(v[
228         x], v[(x + 1) % n], L.p2));
229 };
230 int ret = max(g(i, j - i), g(j, n - (j - i)));
231 return (ret == 0) ? (in_convex(L.p1) == 0 &&
232     in_convex(L.p2) == 0) : ret;
233 }
234 // 回傳點過凸包的兩條切線的切點的  $\theta$ -based index (不保證兩條
235 // 切線的順逆時針關係)
236 pair<int,int> convex_tangent_point(point<T> p) {
237     int n = v.size(), z = -1, edg = -1;
238     auto gt = [&](int neg) {
239         auto check = [&](int x) {
240             if (v[x] == p) z = x;
241             if (btw(v[x], v[(x + 1) % n], p)) edg = x;
242             if (btw(v[(x + n - 1) % n], v[x], p)) edg = (
243                 x + n - 1) % n;
244         };
245         auto f = [&](int x, int y) {
246             check(x); check(y);
247             return ori(p, v[x], v[y]) == neg;
248         };
249         return cycle_search(f);
250     };
251     int x = gt(1), y = gt(-1);
252     if (z != -1) {
253         return {(z + n - 1) % n, (z + 1) % n};
254     }
255     else if (edg != -1) {
256         return {edg, (edg + 1) % n};
257     }
258     else {
259         return {x, y};
260     }
261 }
262 friend int halfplane_intersection(vector<line<T>> &s,
263     polygon<T> &P) {
264     #define neg(p) ((p.y == 0 ? p.x : p.y) < 0)
265     auto angle_cmp = [&](line<T> &A, line<T> &B) {
266         point<T> a = A.p2-A.p1, b = B.p2-B.p1;
267         return neg(a) < neg(b) || (neg(a) == neg(b) && (a
268             ^b) > 0);
269     };
270     #undef neg
271     sort(s.begin(), s.end(), angle_cmp); // 線段左側為該
272     // 線段半平面
273     int L, R, n = s.size();
274     vector<point<T>> px(n);
275     vector<line<T>> q(n);
276     q[L = R = 0] = s[0];
277     for(int i = 1; i < n; ++i) {
278         while(L < R && s[i].ori(px[R-1]) <= 0) --R;
279         while(L < R && s[i].ori(px[L]) <= 0) ++L;
280         q[++R] = s[i];
281         if(q[R].parallel(q[R-1])) {
282             --R;
283             if(q[R].ori(s[i].p1) > 0) q[R] = s[i];
284         }
285     }

```

## 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     point operator-(const point &b)const{
10        return point(x-b.x,y-b.y); }
11     point operator*(const T &b)const{
12        return point(x*b,y*b); }
13     point operator/(const T &b)const{
14        return point(x/b,y/b); }
15     bool operator==(const point &b)const{
16        return x==b.x&&y==b.y; }
17     T dot(const point &b)const{
18        return x*b.x+y*b.y; }
19     T cross(const point &b)const{
20        return x*b.y-y*b.x; }
21     point normal()const{//求法向量
22        return point(-y,x); }
23     T abs2()const{//向量長度的平方
24        return dot(*this); }
25     T rad(const point &b)const{//兩向量的弧度
26        return fabs(atan2(fabs(cross(b)),dot(b))); }
27     T getA()const{//對x軸的弧度
28        T A=atan2(y,x);//超過180度會變負的
29        if(A<=-PI/2)A+=PI*2;
30        return A;
31    };
32 };
33 template<typename T>
34 struct line{
35     line(){}
36     point<T> p1,p2;
37     T a,b,c;//ax+by+c=0
38     line(const point<T>&x,const point<T>&y):p1(x),p2(y){}
39     void pton()const{//轉成一般式
40        a=p1.y-p2.y;
41        b=p2.x-p1.x;
42        c=-a*p1.x-b*p1.y;
43    }
44     T ori(const point<T> &p)const{//點和有向直線的關係 · >0左
45        //邊、=0在線上<0右邊
46        return (p2-p1).cross(p-p1);

```

```

47     T btw(const point<T> &p)const{//點投影落在線段上<=0
48         return (p1-p).dot(p2-p);
49     }
50     bool point_on_segment(const point<T>&p)const{//點是否在線段
51         //上
52         return ori(p)==0&&btw(p)<=0;
53     }
54     T dis2(const point<T> &p,bool is_segment=0)const{//點跟直線
55         //線段的距離平方
56         point<T> v=p2-p1,v1=p-p1;
57         if(is_segment){
58             point<T> v2=p-p2;
59             if(v.dot(v1)<=0)return v1.abs2();
60             if(v.dot(v2)>=0)return v2.abs2();
61         }
62         T tmp=v.cross(v1);
63         return tmp*tmp/v.abs2();
64     }
65     T seg_dis2(const line<T> &l)const{//兩線段距離平方
66         return min({dis2(l.p1,1),dis2(l.p2,1),l.dis2(p1,1),l.dis2
67             (p2,1)});
68     }
69     point<T> projection(const point<T> &p)const{//點對直線的投
70         //影
71         point<T> n=(p2-p1).normal();
72         return p-n*(p-p1).dot(n)/n.abs2();
73     }
74     point<T> mirror(const point<T> &p)const{
75         //點對直線的鏡射 · 要先呼叫pton轉成一般式
76         point<T> R;
77         T d=a*p+b;
78         R.x=(b*b*p.x-a*a*p.x-2*a*b*p.y-2*a*c)/d;
79         R.y=(a*a*p.y-b*b*p.y-2*a*b*p.x-2*b*c)/d;
80         return R;
81     }
82     bool equal(const line &l)const{//直線相等
83         return ori(l.p1)==0&&ori(l.p2)==0;
84     }
85     bool parallel(const line &l)const{
86         return (p1-p2).cross(l.p1-l.p2)==0;
87     }
88     bool cross_seg(const line &l)const{
89         return (p2-p1).cross(l.p1-p1)*(p2-p1).cross(l.p2-p1)<=0;
90         //直線是否交線段
91     }
92     int line_intersect(const line &l)const{//直線相交情況 · -1無
93         //限多點、1交於一點、0不相交
94         return parallel(1)?(ori(l.p1)==0?-1:0):1;
95     }
96     int seg_intersect(const line &l)const{
97         T c1=ori(l.p1), c2=ori(l.p2);
98         T c3=l.ori(p1), c4=l.ori(p2);
99         if(c1==0&&c2==0){//共線
100             bool b1=btw(l.p1)>=0,b2=btw(l.p2)>=0;
101             T a3=l.btw(p1),a4=l.btw(p2);
102             if(b1&&b2&&a3==0&&a4>=0) return 2;
103             if(b1&&b2&&a3>=0&&a4==0) return 3;
104             if(b1&&b2&&a3>=0&&a4>=0) return 0;
105             return -1;//無限交點
106         }else if(c1*c2<=0&&c3*c4<=0)return 1;
107         return 0;//不相交
108     }
109     point<T> line_intersection(const line &l)const{//*直線交點*/

```

```

104 point<T> a=p2-p1,b=1.p2-1.p1,s=1.p1-p1;
105 //if(a.cross(b)==0)return INF;
106 return p1+a*(s.cross(b)/a.cross(b));
107 }
108 point<T> seg_intersection(const line &l)const{//線段交點
109 int res=seg_intersect(l);
110 if(res<=0) assert(0);
111 if(res==2) return p1;
112 if(res==3) return p2;
113 return line_intersection(l);
114 }
115 };
116 template<typename T>
117 struct polygon{
118     polygon(){}
119     vector<point<T> > p;//逆時針順序
120     T area()const{//面積
121         T ans=0;
122         for(int i=p.size()-1,j=0;j<(int)p.size();i=j++){
123             ans+=p[i].cross(p[j]);
124         }
125         return ans/2;
126     }
127     point<T> center_of_mass()const{//重心
128         T cx=0,cy=0,w=0;
129         for(int i=p.size()-1,j=0;j<(int)p.size();i=j++){
130             T a=p[i].cross(p[j]);
131             cx+=(p[i].x+p[j].x)*a;
132             cy+=(p[i].y+p[j].y)*a;
133             w+=a;
134         }
135         return point<T>(cx/3/w,cy/3/w);
136     }
137     char ahas(const point<T>& t)const{//點是否在簡單多邊形內。
138         //是的話回傳1、在邊上回傳-1、否則回傳0
139         bool c=0;
140         for(int i=0,j=p.size()-1;i<p.size();j=i++){
141             if(line<T>(p[i],p[j]).point_on_segment(t))return -1;
142             else if((p[i].y>t.y)!=p[j].y>t.y)&&
143                 t.x<(p[j].x-p[i].x)*(t.y-p[i].y)/(p[j].y-p[i].y)+p[i].x
144             )
145                 c=!c;
146             return c;
147         }
148     }
149     char point_in_convex(const point<T>&x)const{
150         int l=1,r=(int)p.size()-2;
151         while(l<r){//點是否在凸多邊形內。是的話回傳1、在邊上回傳
152             -1、否則回傳0
153             int mid=(l+r)/2;
154             T a1=(p[mid]-p[0]).cross(x-p[0]);
155             T a2=(p[mid+1]-p[0]).cross(x-p[0]);
156             if(a1>0&&a2<=0){
157                 T res=(p[mid+1]-p[mid]).cross(x-p[mid]);
158                 return res>0?1:(res>0?-1:0);
159             }else if(a1<0)r=mid-1;
160             else l=mid+1;
161         }
162         return 0;
163     }
164     vector<T> getA()const{//凸包邊對x軸的夾角
165         vector<T>res;//一定是遞增的
166         for(size_t i=0;i<p.size();i++){
167             res.push_back((p[(i+1)%p.size()]-p[i]).getA());
168         }
169         return res;
170     }
171     bool line_intersect(const vector<T>&A,const line<T> &l)
172         const{//0(LogN)
173         int f1=upper_bound(A.begin(),A.end(),(1.p1-1.p2).getA())-
174             A.begin();
175         int f2=upper_bound(A.begin(),A.end(),(1.p2-1.p1).getA())-
176             A.begin();
177         return l.cross_seg(line<T>(p[f1],p[f2]));
178     }
179     polygon cut(const line<T> &l)const{//凸包對直線切割。得到直
180         //線L左側的凸包
181         polygon ans;
182         for(int n=p.size(),i=n-1,j=0;j<n;i=j++){
183             if(l.ori(p[i])>=0){
184                 ans.p.push_back(p[i]);
185                 if(l.ori(p[j])<0)
186                     ans.p.push_back(l.line_intersection(line<T>(p[i],p[
187                         j])));
188             }else if(l.ori(p[j])>0)
189                 ans.p.push_back(l.line_intersection(line<T>(p[i],p[j
190                     ])));
191             }
192         }
193         return ans;
194     }
195     static bool monotone_chain_cmp(const point<T>& a,const
196         point<T>& b){//凸包排序函數
197         return (a.x<b.x)|| (a.x==b.x&&a.y<b.y);
198     }
199     void monotone_chain(vector<point<T> > &s){//凸包
200         sort(s.begin(),s.end(),monotone_chain_cmp);
201         p.resize(s.size()+1);
202         int m=0;
203         for(size_t i=0;i<s.size();i++){
204             while(m>=2&&(p[m-1]-p[m-2]).cross(s[i]-p[m-2])<=0)--m;
205             p[m++]=s[i];
206         }
207         for(int i=s.size()-2,t=m+1;i>=0;--i){
208             while(m>=2&&(p[m-1]-p[m-2]).cross(s[i]-p[m-2])<=0)--m;
209             p[m++]=s[i];
210         }
211         if(s.size()>1)--m;
212         p.resize(m);
213     }
214     T diam()const{//直徑
215         int n=p.size(),t=1;
216         T ans=0;p.push_back(p[0]);
217         for(int i=0;i<n;i++){
218             point<T> now=p[i+1]-p[i];
219             while(now.cross(p[t+1]-p[i])>now.cross(p[t]-p[i]))t=(t
220                 +1)%n;
221             ans=max(ans,(p[i]-p[t]).abs2());
222         }
223         return p.pop_back(),ans;
224     }
225     T min_cover_rectangle()const{//最小覆蓋矩形
226         int n=p.size(),t=1,r=1,l;
227         if(n<3)return 0;//也可以做最小周長矩形
228         T ans=1e99;p.push_back(p[0]);
229         for(int i=0;i<n;i++){
230             point<T> now=p[i+1]-p[i];
231             while(now.cross(p[t+1]-p[i])>now.cross(p[t]-p[i]))t=(t
232                 +1)%n;
233             while(now.dot(p[r+1]-p[i])>now.dot(p[r]-p[i]))r=(r+1)%n
234             ;
235             if(!i)l=r;
236         }
237     }
238     while(now.dot(p[l+1]-p[i])<=now.dot(p[l]-p[i]))l=(l+1)%
239         n;
240     T d=now.abs2();
241     T tmp=now.cross(p[t]-p[i])*(now.dot(p[r]-p[i])-now.dot(
242         p[l]-p[i]))/d;
243     ans=min(ans,tmp);
244 }
245 return p.pop_back(),ans;
246 }
247 T dis2(polygon &p1){//凸包最近距離平方
248     vector<point<T> > &P=p,&Q=p1.p;
249     int n=P.size(),m=Q.size(),l=0,r=0;
250     for(int i=0;i<n;++i)if(P[i].y<P[l].y)l=i;
251     for(int i=0;i<m;++i)if(Q[i].y<Q[r].y)r=i;
252     P.push_back(P[0]),Q.push_back(Q[0]);
253     T ans=1e99;
254     for(int i=0;i<n;++i){
255         while((P[l]-P[l+1]).cross(Q[r+1]-Q[r])<0)r=(r+1)%m;
256         ans=min(ans,line<T>(P[l],P[l+1]).seg_dis2(line<T>(Q[r],
257             Q[r+1])));
258         l=(l+1)%n;
259     }
260     return P.pop_back(),Q.pop_back(),ans;
261 }
262 static char sign(const point<T>&t){
263     return (t.y==0?t.x:t.y)<0;
264 }
265 static bool angle_cmp(const line<T>& A,const line<T>& B){
266     point<T> a=A.p2-A.p1,b=B.p2-B.p1;
267     return sign(a)<sign(b)|| (sign(a)==sign(b)&&a.cross(b)>0);
268 }
269 int halfplane_intersection(vector<line<T> > &s){//半平面交
270     sort(s.begin(),s.end(),angle_cmp);//線段左側為該線段半平
271     面
272     int L,R,n=s.size();
273     vector<point<T> > px(n);
274     vector<line<T> > q(n);
275     q[L=R=0]=s[0];
276     for(int i=1;i<n;++i){
277         while(L<R&&s[i].ori(px[R-1])<=0)--R;
278         while(L<R&&s[i].ori(px[L])<=0)+L;
279         q[++R]=s[i];
280         if(q[R].parallel(q[R-1])){
281             --R;
282             if(q[R].ori(s[i].p1)>0)q[R]=s[i];
283         }
284         if(L<R)px[R-1]=q[R-1].line_intersection(q[R]);
285     }
286     while(L<R&&q[L].ori(px[R-1])<=0)--R;
287     p.clear();
288     if(R-L<=1)return 0;
289     px[R]=q[R].line_intersection(q[L]);
290     for(int i=L;i<R;++i)p.push_back(px[i]);
291     return R-L+1;
292 }
293 };
294 template<typename T>
295 struct triangle{
296     point<T> a,b,c;
297     triangle(){}
298     triangle(const point<T> &a,const point<T> &b,const point<T>
299         &c):a(a),b(b),c(c){}
300     T area()const{
301         T t=(b-a).cross(c-a)/2;
302         return t>0?t:-t;
303     }
304 }

```

```

279 }
280 point<T> barycenter()const{//重心
281     return (a+b+c)/3;
282 }
283 point<T> circumcenter()const{//外心
284     static line<T> u,v;
285     u.p1=(a+b)/2;
286     u.p2=point<T>(u.p1.x-a.y+b.y,u.p1.y+a.x-b.x);
287     v.p1=(a+c)/2;
288     v.p2=point<T>(v.p1.x-a.y+c.y,v.p1.y+a.x-c.x);
289     return u.line_intersection(v);
290 }
291 point<T> incenter()const{//內心
292     T A=sqrt((b-c).abs2()),B=sqrt((a-c).abs2()),C=sqrt((a-b).abs2());
293     return point<T>(A*a.x+B*b.x+C*c.x,A*a.y+B*b.y+C*c.y)/(A+B+C);
294 }
295 point<T> perpercenter()const{//垂心
296     return barycenter()*3-circumcenter()*2;
297 }
298 };
299 template<typename T>
300 struct point3D{
301     T x,y,z;
302     point3D(){}
303     point3D(const T&x,const T&y,const T&z):x(x),y(y),z(z){}
304     point3D operator+(const point3D &b)const{
305         return point3D(x+b.x,y+b.y,z+b.z);
306     }
307     point3D operator-(const point3D &b)const{
308         return point3D(x-b.x,y-b.y,z-b.z);
309     }
310     point3D operator*(const T &b)const{
311         return point3D(x*b,y*b,z*b);
312     }
313     bool operator==(const point3D &b)const{
314         return x==b.x&&y==b.y&&z==b.z;
315     }
316     T dot(const point3D &b)const{
317         return x*b.x+y*b.y+z*b.z;
318     }
319     point3D cross(const point3D &b)const{
320         return point3D(y*b.z-z*b.y,z*b.x-x*b.z,x*b.y-y*b.x);
321     }
322     T abs2()const{//向量長度的平方
323         return dot(*this);
324     }
325     T area2(const point3D &b)const{//和b、原點圍成面積的平方
326         return cross(b).abs2()/4;
327     }
328 };
329 template<typename T>
330 struct line3D{
331     point3D<T> p1,p2;
332     line3D(){}
333     line3D(const point3D<T> &p1,const point3D<T> &p2):p1(p1),p2(p2){}
334     T dis2(const point3D<T> &p,bool is_segment=0)const{//點跟直線/線段的距離平方
335         point3D<T> v=p2-p1,v1=p-p1;
336         if(is_segment){
337             point3D<T> v2=p-p2;
338             if(v.dot(v1)<=0)return v1.abs2();
339             if(v.dot(v2)>=0)return v2.abs2();
340         }
341         point3D<T> tmp=v.cross(v1);
342         return tmp.abs2()/v.abs2();
343     }
344 };
345 pair<point3D<T>,point3D<T>> closest_pair(const line3D<T> & 393
346     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     ;
353     T t1=(G.cross(d2)).dot(D)/D.abs2();
354     T t2=(G.cross(d1)).dot(D)/D.abs2();
355     return make_pair(p1+d1*t1,l.p1+d2*t2);
356 }
357 bool same_side(const point3D<T> &a,const point3D<T> &b)
358     const{
359     return (p2-p1).cross(a-p1).dot((p2-p1).cross(b-p1))>0;
360 }
361 template<typename T>
362 struct plane{
363     point3D<T> p0,n;//平面上的點和法向量
364     plane(){}
365     plane(const point3D<T> &p0,const point3D<T> &n):p0(p0),n(n)
366     {}
367     T dis2(const point3D<T> &p)const{//點到平面距離的平方
368         T tmp=(p-p0).dot(n);
369         return tmp*tmp/n.abs2();
370     }
371     point3D<T> projection(const point3D<T> &p)const{
372         return p-n*(p-p0).dot(n)/n.abs2();
373     }
374     point3D<T> line_intersection(const line3D<T> &l)const{
375         T tmp=n.dot(l.p2-l.p1);//等於0表示平行或重合該平面
376         return l.p1+(l.p2-l.p1)*(n.dot(p0-l.p1)/tmp);
377     }
378     line3D<T> plane_intersection(const plane &p1)const{
379         point3D<T> e=n.cross(p1.n),v=n.cross(e);
380         T tmp=p1.n.dot(v);//等於0表示平行或重合該平面
381         point3D<T> q=p0+(v*(p1.n.dot(p1.p0-p0))/tmp);
382         return line3D<T>(q,q+e);
383     }
384 };
385 template<typename T>
386 struct triangle3D{
387     point3D<T> a,b,c;
388     triangle3D(){}
389     triangle3D(const point3D<T> &a,const point3D<T> &b,const
390         point3D<T> &c):a(a),b(b),c(c){}
391     bool point_in(const point3D<T> &p)const{//點在該平面上的投影在三角形中
392         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);
393     }
394 };
395 template<typename T>
396 struct tetrahedron{//四面體
397     point3D<T> a,b,c,d;
398     tetrahedron(){}
399     tetrahedron(const point3D<T> &a,const point3D<T> &b,const
400         point3D<T> &c,const point3D<T> &d):a(a),b(b),c(c),d(d)
401     {}
402     T volume6()const{//體積的六倍
403         return (d-a).dot((b-a).cross(c-a));
404     }
405     point3D<T> centroid()const{
406         return (a+b+c+d)/4;
407     }
408     bool point_in(const point3D<T> &p)const{
409         return triangle3D<T>(a,b,c).point_in(p)&&triangle3D<T>(c,d,a).point_in(p);
410     }
411 };
412 template<typename T>
413 struct convexhull3D{
414     static const int MAXN=1005;
415     struct face{
416         int a,b,c;
417         face(int a,int b,int c):a(a),b(b),c(c){}
418     };
419     vector<point3D<T>> pt;
420     vector<face> ans;
421     int fid[MAXN][MAXN];
422     void build(){
423         int n=pt.size();
424         ans.clear();
425         memset(fid,0,sizeof(fid));
426         ans.emplace_back(0,1,2);//注意不能共線
427         ans.emplace_back(2,1,0);
428         int ftop = 0;
429         for(int i=3, ftop=1; i<n; ++i,++ftop){
430             vector<face> next;
431             for(auto &f:ans){
432                 T d=(pt[i]-pt[f.a]).dot((pt[f.b]-pt[f.a]).cross(pt[f.c]-pt[f.a]));
433                 if(d<=0) next.push_back(f);
434                 int ff=0;
435                 if(d>0) ff=ftop;
436                 else if(d<0) ff=-ftop;
437                 fid[f.a][f.b]=fid[f.b][f.c]=fid[f.c][f.a]=ff;
438             }
439             for(auto &f:ans){
440                 if(fid[f.a][f.b]>0 && fid[f.a][f.b]!=fid[f.b][f.a])
441                     next.emplace_back(f.a,f.b,i);
442                 if(fid[f.b][f.c]>0 && fid[f.b][f.c]!=fid[f.c][f.b])
443                     next.emplace_back(f.b,f.c,i);
444                 if(fid[f.c][f.a]>0 && fid[f.c][f.a]!=fid[f.a][f.c])
445                     next.emplace_back(f.c,f.a,i);
446             }
447             ans=next;
448         }
449     }
450     point3D<T> centroid()const{
451         point3D<T> res(0,0,0);
452         T vol=0;
453         for(auto &f:ans){
454             T tmp=pt[f.a].dot(pt[f.b].cross(pt[f.c]));
455             res=res+(pt[f.a]+pt[f.b]+pt[f.c])*tmp;
456             vol+=tmp;
457         }
458         return res/(vol*4);
459     }
460 };

```

### 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 cl) {
22        comp[v] = cl;
23        for (int u : rev_G[v]) {
24            if (comp[u] == -1)
25                dfs2(u, cl);
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);

```

```

59        rev_G[a].push_back(neg_b);
60        return;
61    }
62    void get_result(vector<int>& res) {
63        res.clear();
64        for (int i = 0; i < n; i++)
65            res.push_back(assignment[i]);
66    }
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         -1), my(m, -1), visx(n), visy(n){
10        stamp = 0;
11    }
12    void add(int x, int y){
13        G[x].push_back(y);
14    }
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        for (int i=0 ; i<n ; i++){
49            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     return ret;
86 }
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
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 }

```

```

35 // e22e39
36 T flow(int s, int t) {
37     T ans = 0;
38     while (true) {
39         fill(dis.begin(), dis.end(), INF);
40         queue<int> q;
41         q.push(s);
42         dis[s] = 0;
43
44         while (q.size()) {
45             int u = q.front(); q.pop();
46             for (auto [v, rc, rid] : G[u]) {
47                 if (rc <= 0 || dis[v] < INF) continue;
48                 dis[v] = dis[u] + 1;
49                 q.push(v);
50             }
51         }
52         if (dis[t] == INF) break;
53
54         fill(it.begin(), it.end(), 0);
55         while (true) {
56             T df = dfs(s, t, INF);
57             if (df <= 0) break;
58             ans += df;
59         }
60     }
61     return ans;
62 }
63
64 // the code below constructs minimum cut
65 void dfs_mincut(int now, vector<bool> &vis) {
66     vis[now] = true;
67     for (auto &[v, rc, rid] : G[now]) {
68         if (vis[v] == false && rc > 0) {
69             dfs_mincut(v, vis);
70         }
71     }
72 }
73
74 vector<pair<int, int>> construct(int n, int s, vector<
   pair<int, int>> &E) {
75     // E is G without capacity
76     vector<bool> vis(n);
77     dfs_mincut(s, vis);
78     vector<pair<int, int>> ret;
79     for (auto &[u, v] : E) {
80         if (vis[u] == true && vis[v] == false) {
81             ret.emplace_back(u, v);
82         }
83     }
84     return ret;
85 }
86
87 };

```

## 6.7 Dominator Tree

```

1 /*
2 全部都是 0-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 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 }

```

## 6.9 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), mxson(_n + 1), topf(_n + 1), id(_n + 1) {}
11     void dfs1(int v = 1, int p = -1) {
12         pa[v] = p; sz[v] = 1; mxson[v] = 0;
13         depth[v] = (p == -1 ? 0 : depth[p] + 1);
14         for (int u : G[v]) {
15             if (u == p) continue;
16             dfs1(u, v);
17             sz[v] += sz[u];
18             if (sz[u] > sz[mxson[v]]) mxson[v] = u;
19         }
20     }
21     void dfs2(int v = 1, int top = 1) {
22         id[v] = ++idcnt;
23         topf[v] = top;
24         if (mxson[v]) dfs2(mxson[v], top);
25         for (int u : G[v]) {
26             if (u == mxson[v] || u == pa[v]) continue;
27             dfs2(u, u);
28         }
29     }
30     // query 為區間資料結構
31     int path_query(int a, int b) {
32         int res = 0;
33         while (topf[a] != topf[b]) { /// 若不在同一條鍊上
34             if (depth[topf[a]] < depth[topf[b]]) swap(a, b);
35             res = max(res, 011); // query : l = id[topf[a]],
36                 r = id[a]
37             a = pa[topf[a]];
38         }
39         /// 此時已在同一條鍊上
40         if (depth[a] < depth[b]) swap(a, b);
41         res = max(res, 011); // query : l = id[b], r = id[a]
42         return res;
43     }
44 };

```

## 6.10 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
   的結果存在 result 裡
9
10 SCC[i] = 某個 SCC 中的所有點
11 SCC_id[i] = 第 i 個點在第幾個 SCC
12 */
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]].
            push_back(SCC_id[v]);
70     }
71     for (int i=0 ; i<SCC.size() ; i++){
72         sort(result[i].begin(), result[i].end());
73         result[i].resize(unique(result[i].begin(), result
            [i].end())-result[i].begin());
74     }
75 }
76 };

```

## 6.11 Kuhn Munkres

```

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

```

```

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

```

## 6.12 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 }

```

```

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

```

## 6.13 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

```

```

31     for (int i=0 ; i<SZ(G[v]) ; i++){
32         auto [u, rc, k, rv] = G[v][i];
33         if (rc>0 && dis[v]+k<dis[u]){
34             dis[u] = dis[v]+k;
35             par[u] = v;
36             par_eid[u] = i;
37             if (!in_q[u]) Q.push(u);
38             in_q[u] = true;
39         }
40     }
41 }
42
43 return dis[t];
44 }
45
46 // return <max flow, min cost>, 150093
47 pair<int, int> flow(int s, int t){
48     int fl = 0, cost = 0, d;
49     while ((d = spfa(s, t))<INF){
50         int cur = INF;
51         for (int v=t ; v!=s ; v=par[v])
52             cur = min(cur, G[par[v]][par_eid[v]].rc);
53         fl += cur;
54         cost += d*cur;
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         return {fl, cost};
60     }
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 };

```

## 6.14 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 };

```

## 6.15 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        }
14        else if (vis[x]==0){
15            cnt++;
16            dfs(x, now);
17            low[now] = min(low[now], low[x]);
18            if (low[x]>=dep[now]) ap=1;
19        }
20        else{
21            low[now] = min(low[now], dep[x]);
22        }
23    }
24
25    if ((now==pre && cnt>=2) || (now!=pre && ap)){

```

```

24 |         AP.push_back(now);
25 |     }
26 | }

```

## 6.16 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<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<
   | int, int> &rec, int now, int pre){
26 |     s[now]=1;
27 |     w[now]=0;
28 |     for (auto x : g[now]){
29 |         if (x!=pre){
30 |             centroid(g, s, w, rec, x, now);
31 |             s[now]+=s[x];
32 |             w[now]=max(w[now], s[x]);
33 |         }
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 |     }else{

```

```

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

```

## 6.17 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 <<
   | qwe << ' '; cerr << endl;
15 | #define deb1 cerr << "deb1" << endl;
16 | #define deb2 cerr << "deb2" << endl;
17 | #define deb3 cerr << "deb3" << endl;
18 | #define deb4 cerr << "deb4" << endl;
19 | #define deb5 cerr << "deb5" << endl;
20 | #define bye exit(0);
21 | using namespace std;
22 |
23 | const int mxn = (int)(2e5) + 10;
24 | const int mxlg = 17;
25 | int last_special_node = (int)(1e5) + 1;
26 | vector<int> E[mxn], F[mxn];
27 |
28 | struct edg{
29 |     int fr, to;
30 |     edg(int _fr, int _to){
31 |         fr = _fr;
32 |         to = _to;
33 |     }
34 | };
35 | ostream& operator<<(ostream& os, edg x){os << x.fr << "--" <<
   | x.to;}
36 | vector<edg> EV;
37 |
38 | void tarjan(int v, int par, stack<int>& S){
39 |     static vector<int> dfn(mxn), low(mxn);
40 |     static vector<bool> to_add(mxn);
41 |     static int nowT = 0;
42 |
43 |     int child = 0;
44 |     nowT += 1;
45 |     dfn[v] = low[v] = nowT;
46 |     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         childs += 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(edge(u, v));
146         E[v].pb(EV.size());
147         EV.pb(edge(v, u));
148     }
149     E[0].pb(EV.size());
150     EV.pb(edge(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     }

```

## 6.18 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 -> 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   MaxFlow(n, m) (Non-Biparte:O(n^2m) / Bipartite:O(mvn))
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 > Σw_i /
30         G.add(u, v, INF);
31     }
32     int cut = G.flow(S, T);
33     return sum - cut;

```

## 6.19 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;
24     m1=m1*m2/g;
25     a1=(a1%m1+m1)%m1;
26     return {1, {a1, m1}};

```

## 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     }
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){

```

```

61     return a*B.b > B.a*b;
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 };
72 ostream& operator<<(ostream &os, const frac& A){
73     os << A.a << "/" << A.b;
74     return os;
75 }
76 /// Fraction template ends ///
77 void test(frac A, frac B){
78     cout << "A = " << A << endl;
79     cout << "B = " << B << endl;
80     cout << endl;
81     cout << "A + B = " << A + B << endl;
82     cout << "A - B = " << A - B << endl;
83     cout << "A * B = " << A * B << endl;
84     cout << "A / B = " << A / B << endl;
85     cout << endl;
86     cout << "(A < B) = " << (A < B) << endl;
87     cout << "(A <= B) = " << (A <= B) << endl;
88     cout << "(A > B) = " << (A > B) << endl;
89     cout << "(A >= B) = " << (A >= B) << endl;
90     cout << "(A == B) = " << (A == B) << endl;
91     cout << "(A != B) = " << (A != B) << endl;
92     cout << "-----\n";
93     return;
94 }
95 int main(){
96     frac tmp1(-7, 2);
97     frac tmp2(5, 3);
98     test(tmp1, tmp2);
99
100     frac tmp3(-7);
101     frac tmp4(0);
102     test(tmp3, tmp4);
103     return 0;

```

## 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; 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).
24      * This determines a polynomial f(x) whose degree is at most
25      * n.
26      * Then you can call sample(x) and you get the value of f(x)
27      *
28      * Complexity of init() and sample() are both O(n).
29      */
30     int m, shift; // m = n + 1
31     vector<int> v, mul;

```

```

28 // You can use this function if you don't have inv_fac array
29 // already.
30 void construct_inv_fac() {
31     long long fac = 1;
32     for (int i = 2; i < MAX_N; ++i) {
33         fac = fac * i % mod;
34     }
35     inv_fac[MAX_N - 1] = fp(fac, mod - 2);
36     for (int i = MAX_N - 1; i >= 1; --i) {
37         inv_fac[i - 1] = inv_fac[i] * i % mod;
38     }
39 }
40 // You call init() many times without having a second
41 // instance of this struct.
42 void init(int X_0, vector<int> &u) {
43     v = u;
44     shift = ((1 - X_0) % mod + mod) % mod;
45     if (v.size() == 1) v.push_back(v[0]);
46     m = v.size();
47     mul.resize(m);
48 }
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 int main() {
74     int n; cin >> n;
75     vector<int> v(n);
76     for (int i = 0; i < n; ++i) {
77         cin >> v[i];
78     }
79     Lagrange L;
80     L.construct_inv_fac();
81     L.init(0, v);
82     int x; cin >> x;
83     cout << L.sample(x);

```

## 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.resize(n, vector<int>(m));
9     }
10
11     Matrix operator * (Matrix b){
12         Matrix b_t(b.m, b.n);
13         for (int i=0 ; i<b.n ; i++){
14             for (int j=0 ; j<b.m ; j++){
15                 b_t.arr[j][i] = b.arr[i][j];
16             }
17         }
18
19         Matrix ret(n, b.m);
20         for (int i=0 ; i<n ; i++){
21             for (int j=0 ; j<b.m ; j++){
22                 for (int k=0 ; k<m ; k++){
23                     ret.arr[i][j] += arr[i][k]*b_t.arr[j][k];
24                     ret.arr[i][j] %= MOD;
25                 }
26             }
27         }
28         return ret;
29     }
30
31     Matrix pow(int p){
32         Matrix ret(n, n), mul = *this;
33         for (int i=0 ; i<n ; i++){
34             ret.arr[i][i] = 1;
35         }
36
37         for (; p ; p>=1){
38             if (p&1) ret = ret*mul;
39             mul = mul*mul;
40         }
41
42         return ret;
43     }
44
45     int det(){
46         vector<vector<int>> arr = this->arr;
47         bool flag = false;
48         for (int i=0 ; i<n ; i++){
49             int target = -1;
50             for (int j=i ; j<n ; j++){
51                 if (arr[j][i]){
52                     target = j;
53                     break;
54                 }
55             }
56         }

```



```

57     if (target==-1) return 0;
58     if (i!=target){
59         swap(arr[i], arr[target]);
60         flag = !flag;
61     }
62
63     for (int j=i+1 ; j<n ; j++){
64         if (!arr[j][i]) continue;
65         int freq = arr[j][i]*qp(arr[i][i], MOD-2)%MOD
66             ;
67         for (int k=i ; k<n ; k++){
68             arr[j][k] -= freq*arr[i][k];
69             arr[j][k] = (arr[j][k]%MOD+MOD)%MOD;
70         }
71     }
72
73     int ret = !flag ? 1 : MOD-1;
74     for (int i=0 ; i<n ; i++){
75         ret *= arr[i][i];
76         ret %= MOD;
77     }
78     return ret;
79 }
80 };

```

## 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
41         if (x==n-1) continue;
42         return 0;
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
7  // O(n^{1/4}) 回傳 1 或自己的因數、記得先判斷 n 是不是質數
8  // (用 Miller-Rabin)
9  // c1670c
10 int Pollard_Rho(int n){
11     int s = 0, t = 0;
12     int c = rnd(1, n-1);

```

```

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

- $1 \sim x$  質數的數量  $\approx \frac{x}{\ln x}$
- $1 \sim x$  的因數的數量  $\approx x^{\frac{1}{3}}$
- $x$  的質因數的數量  $\approx \log \log x$
- $p$  is a prime number  $\Leftrightarrow (p-1)! \equiv -1 \pmod{p}$
- 每個正整數都可以表示成四個整數的平方和
- 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  $S[1 \dots k] = S[i-k$ 
     $+1 \dots i]$  (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
7                 -1); break;}
8             if (s[a+k] > s[b+k]) {a = b; break;}
9         }
10    }
11    return a;
```

## 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,
11            lim)), rank(n);
12        x.push_back(0);
13        sa = lcp = y;
14        iota(sa.begin(), sa.end(), 0);
15        for (int j=0, p=0 ; p<n ; j=max(1LL, j*2), lim=p) {
16            p = j;
17            iota(y.begin(), y.end(), n-j);
18            for (int i=0 ; i<n ; i++) if (sa[i] >= j) y[p++]
19                = sa[i] - j;
20            fill(ws.begin(), ws.end(), 0);
21            for (int i=0 ; i<n ; i++) ws[x[i]]++;
22            for (int i=1 ; i<lim ; i++) ws[i] += ws[i-1];
23            for (int i = n; i--;) sa[--ws[x[i]]] = y[i];
24            swap(x, y), p = 1, x[sa[0]] = 0;
25            for (int i=1 ; i<n ; i++){
26                a = sa[i-1];
27                b = sa[i];
28                x[b] = (y[a] == y[b] && y[a+j] == y[b+j])
29                    ? p-1 : p++;
30            }
31        }
32
33        for (int i=1 ; i<n ; i++) rank[sa[i]] = i;
34        for (int i=0, j ; i<n-1 ; lcp[rank[i++]]+=k)
35            for (k && k-- , j=sa[rank[i]-1] ; i+k<s.size() &&
36                j+k<s.size() && s[i+k]==s[j+k] ; k++);
37        sa.erase(sa.begin());
38        lcp.erase(lcp.begin(), lcp.begin()+2);
39    }
40
41    // f49583
42    vector<int> pos; // pos[i] = i 這個值在 pos 的哪個地方
43    SparseTable st;
44    void init_lcp(){
45        pos.resize(sa.size());
46        for (int i=0 ; i<sa.size() ; i++){
47            pos[sa[i]] = i;
48        }
49        if (lcp.size()){
50            st.build(lcp);
51        }
52    }
53
54    // 用之前記得 init
55    // 回傳 [l1, r1] 跟 [l2, r2] 的 lcp，0-based
56    int get_lcp(int l1, int r1, int l2, int r2){
57        int pos_1 = pos[l1], len_1 = r1-l1+1;
58        int pos_2 = pos[l2], len_2 = r2-l2+1;
59        if (pos_1>pos_2){
60            swap(pos_1, pos_2);
61            swap(len_1, len_2);
62        }
63
64        if (l1==l2){
65            return min(len_1, len_2);
66        }
67    }
```

```
68 }else{
69     return min({st.query(pos_1, pos_2), len_1, len_2
70         });
71 }
72
73 // 檢查 [l1, r1] 跟 [l2, r2] 的大小關係，0-based
74 // 如果前者小於後者，就回傳 <0，相等就回傳 =0，否則回傳
75 >0
76 // 5b8db0
77 int substring_cmp(int l1, int r1, int l2, int r2){
78     int len_1 = r1-l1+1;
79     int len_2 = r2-l2+1;
80     int res = get_lcp(l1, r1, l2, r2);
81
82     if (res<len_1 && res<len_2){
83         return s[l1+res]-s[l2+res];
84     }else if (len_1==res && len_2==res){
85         // 如果不需要以 index 作為次要排序參數，這裡要回
86         傳 0
87         return l1-l2;
88     }else{
89         return len_1==res ? -1 : 1;
90     }
91 }
92
93 // 對於位置在 <=p 的後綴，找離他左邊/右邊最接近位置 >p 的
94 後綴的 lcp，0-based
95 // pre[i] = s[i] 離他左邊最接近位置 >p 的後綴的 lcp，0-
96 based
97 // suf[i] = s[i] 離他右邊最接近位置 >p 的後綴的 lcp，0-
98 based
99 // da12fa
100 pair<vector<int>, vector<int>> get_left_and_right_lcp(int
101     p){
102     vector<int> pre(p+1);
103     vector<int> suf(p+1);
104
105     { // build pre
106         int now = 0;
107         for (int i=0 ; i<s.size() ; i++){
108             if (sa[i]<=p){
109                 pre[sa[i]] = now;
110                 if (i<lcp.size()) now = min(now, lcp[i]);
111             }else{
112                 if (i<lcp.size()) now = lcp[i];
113             }
114         }
115     }
116     { // build suf
117         int now = 0;
118         for (int i=s.size()-1 ; i>=0 ; i--){
119             if (sa[i]<=p){
120                 suf[sa[i]] = now;
121                 if (i-1>=0) now = min(now, lcp[i-1]);
122             }else{
123                 if (i-1>=0) now = lcp[i-1];
124             }
125         }
126     }
127
128     return {pre, suf};
129 }
```

## 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 }
```