

Contents					
1 Misc	2	3.9 Sparse Table	7	7 Math	18
1.1 Default Code	2	3.10 Treap2	7	7.1 CRT	18
1.2 Run	2	3.11 Trie	8	7.2 Josephus Problem	18
1.3 Custom Set PQ Sort	2	4 Dynamic-Programming	8	7.3 Lagrange any x	18
1.4 Dynamic Bitset	2	4.1 Digit DP	8	7.4 Lagrange continuous x	18
1.5 Enumerate Subset	2	4.2 Knapsack On Tree	8	7.5 Linear Mod Inverse	19
1.6 Fast Input	2	4.3 SOS DP	8	7.6 Lucas's Theorem	19
1.7 OEIS	2	4.4 Integer Partition	8	7.7 Matrix	19
1.8 Pragma	2	5 Geometry	9	7.8 Matrix 01	19
1.9 Xor Basis	2	5.1 Geometry Struct	9	7.9 Miller Rabin	20
1.10 random int	3	5.2 Geometry Struct 3D	10	7.10 Pollard Rho	20
1.11 OEIS	3	5.3 Pick's Theorem	11	7.11 Polynomial	20
1.12 Python	3	6 Graph	11	7.12 josephus	21
1.13 diff	3	6.1 2-SAT	11	7.13 數論分塊	21
1.14 disable ASLR	3	6.2 Augment Path	11	7.14 最大質因數	21
1.15 hash command	3	6.3 C3C4	12	7.15 歐拉公式	21
1.16 hash windows	3	6.4 Cut BCC	12	7.16 Burnside's Lemma	21
2 Convolution	3	6.5 Dinic	12	7.17 Catalan Number	21
2.1 FFT	3	6.6 Dominator Tree	13	7.18 Matrix Tree Theorem	21
2.2 FFT any mod	4	6.7 EdgeBCC	13	7.19 Stirling's formula	21
2.3 FWT	4	6.8 EnumeratePlanarFace	13	7.20 Theorem	22
2.4 Min Convolution Concave Concave	4	6.9 HLD	14	7.21 二元一次方程式	22
2.5 NTT mod 998244353	4	6.10 Kosaraju	14	7.22 歐拉定理	22
3 Data-Structure	5	6.11 Kuhn Munkres	14	7.23 錯排公式	22
3.1 BIT	5	6.12 LCA	15	8 String	22
3.2 Disjoint Set Persistent	5	6.13 MCMF	15	8.1 AC automation	22
3.3 PBDS GP Hash Table	5	6.14 Tarjan	16	8.2 Enumerate Runs	22
3.4 PBDS Order Set	5	6.15 Tarjan Find AP	16	8.3 Hash	22
3.5 Segment Tree Add Set	5	6.16 Tree Isomorphism	16	8.4 KMP	22
3.6 Segment Tree Li Chao Line	6	6.17 圓方樹	17	8.5 Manacher	23
3.7 Segment Tree Li Chao Segment	6	6.18 最大權閉合圖	18	8.6 Min Rotation	23
3.8 Segment Tree Persistent	6	6.19 Theorem	18	8.7 Suffix Array	23
				8.8 Wildcard Matching	23
				8.9 Z Algorithm	24
				8.10 k-th Substring1	24

1 Misc

1.1 Default Code [24a798]

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 #define int long long
4 #define debug(a...) cerr << #a << " = ", dout(a)
5 void dout() { cerr << "\n"; }
6 template <typename A, typename... B>
7 void dout(A a, B... b) { cerr << a << ' ', dout(b...); }
8
9 void solve(){
10
11 }
12
13 signed main(){
14     ios::sync_with_stdio(0), cin.tie(0);
15
16     int t = 1;
17     while (t--){
18         solve();
19     }
20
21     return 0;
22 }
```

1.2 Run

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

1.3 Custom Set PQ Sort [d4df55]

```
1 // 所有自訂的結構體，務必檢查相等的 case，給所有元素一個排序
2 // 的依據
3 struct my_struct{
4     int val;
5     my_struct(int _val) : val(_val) {}
6 };
7
8 auto cmp = [](my_struct a, my_struct b) {
9     return a.val > b.val;
10 };
```

```
10
11 set<my_struct, decltype(cmp)> ss({1, 2, 3}, cmp);
12 priority_queue<my_struct, vector<my_struct>, decltype(cmp)>
13     pq(cmp, {1, 2, 3});
14 map<my_struct, my_struct, decltype(cmp)> mp({{1, 4}, {2, 5},
15     {3, 6}}, cmp);
```

1.4 Dynamic Bitset [c78aa8]

```
1 const int MAXN = 2e5 + 5;
2 template <int len = 1>
3 void solve(int n) {
4     if (n > len) {
5         solve<min(len*2, MAXN)>(n);
6         return;
7     }
8     bitset<len> a;
9 }
```

1.5 Enumerate Subset [a13e46]

```
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 }
```

1.6 Fast Input [6f8879]

```
1 // fast IO
2 // 6f8879
3 inline char readchar(){
4     static char buffer[BUFSIZ], *now = buffer + BUFSIZ, *
5     end = buffer + BUFSIZ;
6     if (now == end)
7     {
8         if (end < buffer + BUFSIZ)
9             return EOF;
10        end = (buffer + fread(buffer, 1, BUFSIZ, stdin));
11        now = buffer;
12    }
13    return *now++;
14 }
15 inline int nextint(){
16     int x = 0, c = readchar(), neg = false;
17     while(('0' > c || c > '9') && c!='-' && c!=EOF) c =
18         readchar();
19     if(c == '-') neg = true, c = readchar();
20     while('0' <= c && c <= '9') x = (x<<3) + (x<<1) + (c^'0')
21     , c = readchar();
22     if(neg) x = -x;
23     return x; // returns 0 if EOF
24 }
```

1.7 OEIS [ec45dc]

```
1 // 若一個線性遞迴有 k 項，給他恰好 2*k 個項可以求出線性遞迴
2 // f915c2
3 template <typename T>
4 vector<T> BerlekampMassey(vector<T> a) {
```

```
5     auto scalarProduct = [](vector<T> v, T c) {
6         for (T &x: v) x *= c;
7         return v;
8     };
9     vector<T> s, best;
10    int bestPos = 0;
11    for (int i=0; i<a.size(); i++){
12        T error = a[i];
13        for (int j=0; j<s.size(); j++) error -= s[j] * a[i
14            -1-j];
15        if (error == 0) continue;
16        if (s.empty()) {
17            s.resize(i + 1);
18            bestPos = i;
19            best.push_back(1 / error);
20            continue;
21        }
22        vector<T> fix = scalarProduct(best, error);
23        fix.insert(fix.begin(), i - bestPos - 1, 0);
24        if (fix.size() >= s.size()) {
25            best = scalarProduct(s, - 1 / error);
26            best.insert(best.begin(), 1 / error);
27            bestPos = i;
28            s.resize(fix.size());
29        }
30        for (int j = 0; j < fix.size(); j++) s[j] += fix[j];
31    }
32    reverse(s.begin(), s.end());
33    return s;
34 }
```

1.8 Pragma [09d13e]

```
1 #pragma GCC optimize("O3,unroll-loops")
2 #pragma GCC target("avx,avx2,sse,sse2,sse3,sse4,popcnt")
```

1.9 Xor Basis [840136]

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

27 | }

1.10 random int [9ccc603]

```
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 }
```

1.11 OEIS

```
1 from fractions import Fraction
2
3 def BerlekampMassey(a: list[Fraction]) -> list[Fraction]:
4     def scale(v: list[Fraction], c: Fraction) -> list[
5         Fraction]:
6         return [x * c for x in v]
7
8     s: list[Fraction] = []
9     best: list[Fraction] = []
10    bestPos = 0
11
12    for i in range(len(a)):
13        error: Fraction = a[i]
14        for j in range(len(s)):
15            error -= s[j] * a[i - 1 - j]
16        if error == 0:
17            continue
18
19        if not s:
20            s = [Fraction(0)] * (i + 1)
21            bestPos = i
22            best = [Fraction(1, error)]
23            continue
24
25        fix = scale(best, error)
26        fix = [Fraction(0)] * (i - bestPos - 1) + fix
27
28        if len(fix) >= len(s):
29            best = scale(s, Fraction(-1, error))
30            best.insert(0, Fraction(1, error))
31            bestPos = i
32            if len(s) < len(fix):
33                s += [Fraction(0)] * (len(fix) - len(s))
34
35        for j in range(len(fix)):
36            s[j] += fix[j]
37
38    return list(reversed(s))
39
40 n = int(input())
41 l = list(map(Fraction, input().split()))
42 for i in range(len(l)):
43     coeffs = BerlekampMassey(l[:i+1])
44     for x in coeffs:
45         print(x, end=" ")
46     print()
```

1.12 Python

```
1 # Decimal
2 from decimal import *
3 getcontext().prec = 6
```

```
4
5 # system setting
6 sys.setrecursionlimit(100000)
7 sys.set_int_max_str_digits(10000)
8
9 # turtle
10 from turtle import *
11
12 N = 3000000010
13 setworldcoordinates(-N, -N, N, N)
14 hideturtle()
15 speed(100)
16
17 def draw_line(a, b, c, d):
18     teleport(a, b)
19     goto(c, d)
20
21 def write_dot(x, y, text, diff=1): # diff = 文字的偏移
22     teleport(x, y)
23     dot(5, "red")
24
25     teleport(x+N/100*diff, y+N/100*diff)
26     write(text, font=("Arial", 5, "bold"))
27
28 # usage
29 draw_line(*a[i], *(a[i-1]))
30 write_dot(*a[i], str(a[i]))
31 # 以自己左方 100 單位為圓心向前畫一個 90 度的弧 / 正五邊形
32     ( 參數可以是負的 )
33 circle(100, extent = 90)
34 circle(100, steps = 5)
35 left(90) # 左轉 90 度
36 fd(-100) # 倒退 100 單位
37
38 # OOP
39 class Point:
40     def __init__(self, x, y):
41         self.x = x
42         self.y = y
43
44     def __add__(self, o): # use dir(int) to know operator
45         name
46         return Point(self.x+o.x, self.y+o.y)
47
48     @property
49     def distance(self):
50         return (self.x**2 + self.y**2)**(0.5)
51
52 a = Point(3, 4)
53 print(a.distance)
54
55 # Fraction
56 from fractions import Fraction
57 a = Fraction(Decimal(1.1))
58 a.numerator # 分子
59 a.denominator # 分母
```

1.13 diff

```
1 set -e
2 g++ ac.cpp -o ac
3 g++ wa.cpp -o wa
4 for ((i=0;;i++))
5 do
```

```
6     echo "$i"
7     python3 gen.py > input
8     ./ac < input > ac.out
9     ./wa < input > wa.out
10    diff ac.out wa.out || break
11 done
```

1.14 disable ASLR

```
1 # Disable randomization of memory addresses
2 setarch `uname -m` -R ./yourProgram
3 setarch $(uname -m) -R
```

1.15 hash command

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

1.16 hash windows

```
1 def get_hash(path):
2     from subprocess import run, PIPE
3     from hashlib import md5
4
5     p = run(
6         ["cpp", "-dD", "-P", "-fpreprocessed", path],
7         stdout = PIPE,
8         stderr = PIPE,
9         text = True
10    )
11
12    if p.returncode != 0:
13        raise RuntimeError(p.stderr)
14
15    s = ''.join(p.stdout.split())
16    ret = md5(s.encode()).hexdigest()
17    return ret[:6]
18
19 print(get_hash("Suffix_Array.cpp"))
```

2 Convolution

2.1 FFT [16591a]

```
1 typedef complex<double> cd;
2
3 // 778272
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), rt.resize(n);
10        auto x = polar(1.0L, acos(-1.0L) / k);
11        for (int i=k; i<2*k; i++) rt[i] = R[i] = (i&1 ? R[i
12            /2]*x : R[i/2]);
13    }
14
15    vector<int> rev(n);
16    for (int i=0; i<n; i++) rev[i] = (rev[i/2] | (i&1)<<L)
17        /2;
18    for (int i=0; i<n; i++) if (i<rev[i]) swap(a[i], a[rev[
19        i]]);
20    for (int k=1; k<n; k*=2){
```

```

18     for (int i=0 ; i<n ; i+=2*k){
19         for (int j=0 ; j<k ; j++){
20             // cd z = rt[j+k] * a[i+j+k];
21             auto x = (double *)&rt[j+k], y = (double *)&a
                [i+j+k];
22             cd z(x[0]*y[0] - x[1]*y[1], x[0]*y[1] + x[1]*
                y[0]);
23             a[i+j+k] = a[i+j]-z;
24             a[i+j] += z;
25         }
26     }
27 }
28 // 192528
29 vector<double> PolyMul(const vector<double> a, const vector<
    double> b){
30     if (a.empty() || b.empty()) return {};
31     vector<double> res(a.size()+b.size()-1);
32     int L = 32 - __builtin_clz(res.size()), n = 1<<L;
33     vector<cd> in(n), out(n);
34     copy(a.begin(), a.end(), in.begin());
35     for (int i=0 ; i<b.size() ; i++) in[i].imag(b[i]);
36     FFT(in);
37     for (cd& x : in) x *= x;
38     for (int i=0 ; i<n ; i++) out[i] = in[-i & (n - 1)] -
        conj(in[i]);
39     FFT(out);
40     for (int i=0 ; i<res.size() ; i++) res[i] = imag(out[i])
        / (4 * n);
41     return res;
42 }

```

2.2 FFT any mod [412000]

```

1 // n=524288, ~400ms
2 const int MOD = 998244353;
3 typedef complex<double> cd;
4 // bb6d94
5 void FFT(vector<cd> &a) {
6     int n = a.size(), L = 31-__builtin_clz(n);
7     vector<complex<long double>> R(2, 1);
8     vector<cd> rt(2, 1);
9     for (int k=2 ; k<n ; k*=2){
10         R.resize(n);
11         rt.resize(n);
12         auto x = polar(1.0/L, acos(-1.0/L) / k);
13         for (int i=k ; i<2*k ; i++){
14             rt[i] = R[i] = (i&1 ? R[i/2]*x : R[i/2]);
15         }
16     }
17 }
18 vector<int> rev(n);
19 for (int i=0 ; i<n ; i++) rev[i] = (rev[i/2] | (i&1)<<L)
    /2;
20 for (int i=0 ; i<n ; i++) if (i<rev[i]) swap(a[i], a[rev[
    i]]);
21 for (int k=1 ; k<n ; k*=2){
22     for (int i=0 ; i<n ; i+=2*k){
23         for (int j=0 ; j<k ; j++){
24             auto x = (double *)&rt[j+k];
25             auto y = (double *)&a[i+j+k];

```

```

28         cd z(x[0]*y[0] - x[1]*y[1], x[0]*y[1] + x[1]*
            y[0]);
29         a[i+j+k] = a[i+j]-z;
30         a[i+j] += z;
31     }
32 }
33 }
34 }
35 }
36 // c3dcf6
37 vector<int> PolyMul(vector<int> a, vector<int> b){
38     if (a.empty() || b.empty()) return {};
39     vector<int> res(a.size()+b.size()-1);
40     int B = 32-__builtin_clz(res.size()), n = (1<<B), cut = (
        int)(sqrt(MOD));
41     vector<cd> L(n), R(n), outs(n), outl(n);
42     for (int i=0 ; i<a.size() ; i++) L[i] = cd((int) a[i]/cut
        , (int)a[i]%cut);
43     for (int i=0 ; i<b.size() ; i++) R[i] = cd((int) b[i]/cut
        , (int)b[i]%cut);
44     FFT(L), FFT(R);
45     for (int i=0 ; i<n ; i++){
46         int j = -i&(n-1);
47         outl[j] = (L[i]+conj(L[j])) * R[i]/(2.0*n);
48         outs[j] = (L[i]-conj(L[j])) * R[i]/(2.0*n)/1i;
49     }
50     FFT(outl), FFT(outs);
51     for (int i=0 ; i<res.size() ; i++){
52         int av = (int)(real(outl[i])+0.5), cv = (int)(imag(
            outs[i])+0.5);
53         int bv = (int)(imag(outl[i])+0.5) + (int)(real(outs[
            i])+0.5);
54         res[i] = ((av%MOD*cut+bv) % MOD*cut+cv) % MOD;
55     }
56     return res;
57 }
58 }

```

2.3 FWT [e50788]

```

1 // 已經把 mint 刪掉 · 需要增加註解
2 vector<int> xor_convolution(vector<int> a, vector<int> b, int
    k) {
3     if (k==0) return vector<int>{a[0]*b[0]};
4     vector<int> aa(1 << (k - 1)), bb(1 << (k - 1));
5     for (int i=0 ; i<(1<<(k-1)) ; i++){
6         aa[i] = a[i] + a[i + (1 << (k - 1))];
7         bb[i] = b[i] + b[i + (1 << (k - 1))];
8     }
9     vector<int> X = xor_convolution(aa, bb, k - 1);
10    for (int i=0 ; i<(1<<(k-1)) ; i++){
11        aa[i] = a[i] - a[i + (1 << (k - 1))];
12        bb[i] = b[i] - b[i + (1 << (k - 1))];
13    }
14    vector<int> Y = xor_convolution(aa, bb, k - 1);
15    vector<int> c(1 << k);
16    for (int i=0 ; i<(1<<(k-1)) ; i++){
17        c[i] = (X[i] + Y[i]) / 2;
18        c[i + (1 << (k - 1))] = (X[i] - Y[i]) / 2;
19    }
20    return c;
21 }

```

2.4 Min Convolution Concave Concave [ffb28d]

```

1 // 需要增加註解
2 // min convolution
3 vector<int> mkk(vector<int> a, vector<int> b) {
4     vector<int> slope;
5     FOR (i, 1, ssize(a) - 1) slope.pb(a[i] - a[i - 1]);
6     FOR (i, 1, ssize(b) - 1) slope.pb(b[i] - b[i - 1]);
7     sort(all(slope));
8     slope.insert(begin(slope), a[0] + b[0]);
9     partial_sum(all(slope), begin(slope));
10    return slope;
11 }
12 // 2.5 NTT mod 998244353 [f9fed4]
13 const int MOD = (119 << 23) + 1, ROOT = 62; // = 998244353
14 // For p < 2^30 there is also e.g. 5 << 25, 7 << 26, 479 <<
    21
15 // and 483 << 21 (same root). The last two are > 10^9.
16 // e169db
17 void NTT(vector<int> &a) {
18     int n = a.size();
19     int L = 31-__builtin_clz(n);
20     vector<int> rt(2, 1);
21     for (int k=2, s=2 ; k<n ; k*=2, s++){
22         rt.resize(n);
23         int z[] = {1, qp(ROOT, MOD>>s)};
24         for (int i=k ; i<2*k ; i++) rt[i] = rt[i/2]*z[i&1%
            MOD];
25     }
26     vector<int> rev(n);
27     for (int i=0 ; i<n ; i++) rev[i] = (rev[i/2] | (i&1)<<L)/2;
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                 int z = rt[j+k]*a[i+j+k]%MOD, &ai = a[i+j];
35                 a[i+j+k] = ai-z+(z>ai ? MOD : 0);
36                 ai += (ai+z>MOD ? z-MOD : z);
37             }
38         }
39     }
40     // 290957
41     vector<int> polyMul(vector<int> &a, vector<int> &b){
42         if (a.empty() || b.empty()) return {};
43         int s = a.size()+b.size()-1, B = 32-__builtin_clz(s), n =
            1<<B;
44         int inv = qp(n, MOD-2);
45         vector<int> L(a), R(b), out(n);
46         L.resize(n), R.resize(n);
47         NTT(L), NTT(R);
48         for (int i=0 ; i<n ; i++) out[-i&(n-1)] = L[i]*R[i]%MOD*
            inv%MOD;
49         NTT(out);
50         out.resize(s);
51         return out;
52 }

```

3 Data-Structure

3.1 BIT [7ef3a9]

```

1 vector<int> BIT(MAX_SIZE);
2
3 // const int MAX_N = (1<<20)
4 int k_th(int k){ // 回傳 BIT 中第 k 小的元素 (based-1)
5     int res = 0;
6     for (int i=MAX_N>>1; i>=1; i>>=1)
7         if (BIT[res+i]<k)
8             k -= BIT[res+=i];
9     return res+1;
10 }

```

3.2 Disjoint Set Persistent [447002]

```

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        }
49    }

```

```

42         return true;
43     }
44     return false;
45 }
46 };

```

3.3 PBDS GP Hash Table [866cf6]

```

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 [231774]

```

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

```

3.5 Segment Tree Add Set [bb1898]

```

1 // [ll, rr), based-0
2 // 使用前記得 init(陣列大小), build(陣列名稱)
3 // add(LL, rr): 區間修改
4 // set(LL, rr): 區間賦值
5 // query(LL, rr): 區間求和 / 求最大值
6 struct SegmentTree{
7     struct node{
8         int add_tag = 0;
9         int set_tag = 0;
10        int sum = 0;
11        int ma = 0;
12    };
13
14    vector<node> arr;
15
16    SegmentTree(int n){
17        arr.resize(n<<2);
18    }
19
20    node pull(node A, node B){

```

```

21     node C;
22     C.sum = A.sum+B.sum;
23     C.ma = max(A.ma, B.ma);
24     return C;
25 }
26
27 // cce0c8
28 void push(int idx, int ll, int rr){
29     if (arr[idx].set_tag!=0){
30         arr[idx].sum = (rr-ll)*arr[idx].set_tag;
31         arr[idx].ma = arr[idx].set_tag;
32         if (rr-ll>1){
33             arr[idx*2+1].add_tag = 0;
34             arr[idx*2+1].set_tag = arr[idx].set_tag;
35             arr[idx*2+2].add_tag = 0;
36             arr[idx*2+2].set_tag = arr[idx].set_tag;
37         }
38         arr[idx].set_tag = 0;
39     }
40     if (arr[idx].add_tag!=0){
41         arr[idx].sum += (rr-ll)*arr[idx].add_tag;
42         arr[idx].ma += arr[idx].add_tag;
43         if (rr-ll>1){
44             arr[idx*2+1].add_tag += arr[idx].add_tag;
45             arr[idx*2+2].add_tag += arr[idx].add_tag;
46         }
47         arr[idx].add_tag = 0;
48     }
49 }
50
51 void build(vector<int> &v, int idx = 0, int ll = 0, int
52     rr = n){
53     if (rr-ll==1){
54         arr[idx].sum = v[ll];
55         arr[idx].ma = v[ll];
56     }else{
57         int mid = (ll+rr)/2;
58         build(v, idx*2+1, ll, mid);
59         build(v, idx*2+2, mid, rr);
60         arr[idx] = pull(arr[idx*2+1], arr[idx*2+2]);
61     }
62 }
63
64 void add(int ql, int qr, int val, int idx = 0, int ll =
65     0, int rr = n){
66     push(idx, ll, rr);
67     if (rr<=ql || qr<=ll) return;
68     if (ql<=ll && rr<=qr){
69         arr[idx].add_tag += val;
70         push(idx, ll, rr);
71         return;
72     }
73     int mid = (ll+rr)/2;
74     add(ql, qr, val, idx*2+1, ll, mid);
75     add(ql, qr, val, idx*2+2, mid, rr);
76     arr[idx]=pull(arr[idx*2+1], arr[idx*2+2]);
77 }
78
79 void set(int ql, int qr, int val, int idx=0, int ll=0,
80     int rr=n){
81     push(idx, ll, rr);
82     if (rr<=ql || qr<=ll) return;
83     if (ql<=ll && rr<=qr){
84         arr[idx].add_tag = 0;
85         arr[idx].set_tag = val;
86         push(idx, ll, rr);
87     }

```

```

84         return;
85     }
86     int mid = (ll+rr)/2;
87     set(q1, qr, val, idx*2+1, ll, mid);
88     set(q1, 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 q1, int qr, int idx = 0, int ll = 0, int
93           rr = n){
94     push(idx, ll, rr);
95     if (rr<=q1 || qr<=ll) return node();
96     if (q1<=ll && rr<=qr) return arr[idx];
97
98     int mid = (ll+rr)/2;
99     return pull(query(q1, qr, idx*2+1, ll, mid), query(q1
100   , qr, idx*2+2, mid, rr));

```

3.6 Segment Tree Li Chao Line [45b8ba]

```

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

```

```

40         return;
41     }
42
43     int query(int x, int idx = 0, int ll = 0, int rr = MAX_V)
44     {
45         if (rr-ll==0) return INF;
46         if (rr-ll==1){
47             return arr[idx].y(ll);
48         }
49
50         int mid = (ll+rr)/2;
51         if (x<mid){
52             return min(arr[idx].y(x), query(x, idx*2+1, ll,
53               mid));
54         }else{
55             return min(arr[idx].y(x), query(x, idx*2+2, mid,
56               rr));
57         }
58     }
59 }
60 };

```

3.7 Segment Tree Li Chao Segment [2cb0a4]

```

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

```

```

38         update(val, idx*2+2, mid, rr);
39     }
40     return;
41 }
42
43 // 在 [ql, qr) 加上一條 val 的線段
44 void update_segment(Node val, int ql, int qr, int idx =
45   0, int ll = 0, int rr = MAX_V){
46     if (rr-ll==0) return;
47     if (rr<=ql || qr<=ll) return;
48     if (ql<=ll && rr<=qr){
49         update(val, idx, ll, rr);
50         return;
51     }
52
53     int mid = (ll+rr)/2;
54     update_segment(val, ql, qr, idx*2+1, ll, mid);
55     update_segment(val, ql, qr, idx*2+2, mid, rr);
56     return;
57 }
58
59 int query(int x, int idx = 0, int ll = 0, int rr = MAX_V)
60 {
61     if (rr-ll==0) return INF;
62     if (rr-ll==1){
63         return arr[idx].y(ll);
64     }
65
66     int mid = (ll+rr)/2;
67     if (x<mid){
68         return min(arr[idx].y(x), query(x, idx*2+1, ll,
69           mid));
70     }else{
71         return min(arr[idx].y(x), query(x, idx*2+2, mid,
72           rr));
73     }
74 }
75 };

```

3.8 Segment Tree Persistent [3b5aa9]

```

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

```



```

22 }
23
24 void pull(Node &c, Node a, Node b){
25     c.val = a.val+b.val;
26     return;
27 }
28
29 void build(vector<int> &v, int idx, int ll = 0, int rr =
30     n){
31     auto &now = arr[idx];
32
33     if (rr-ll==1){
34         now.val = v[ll];
35         return;
36     }
37
38     int mid = (ll+rr)/2;
39     now.lc = node_cnt++;
40     now.rc = node_cnt++;
41     build(v, now.lc, ll, mid);
42     build(v, now.rc, mid, rr);
43     pull(now, arr[now.lc], arr[now.rc]);
44     return;
45 }
46
47 void update(int pos, int val, int idx, int ll = 0, int rr
48     = n){
49     auto &now = arr[idx];
50
51     if (rr-ll==1){
52         now.val = val;
53         return;
54     }
55
56     int mid = (ll+rr)/2;
57     if (pos<mid){
58         arr[node_cnt] = arr[now.lc];
59         now.lc = node_cnt;
60         node_cnt++;
61         update(pos, val, now.lc, ll, mid);
62     }else{
63         arr[node_cnt] = arr[now.rc];
64         now.rc = node_cnt;
65         node_cnt++;
66         update(pos, val, now.rc, mid, rr);
67     }
68
69     pull(now, arr[now.lc], arr[now.rc]);
70     return;
71 }
72
73 void update_version(int pos, int val, int ver){
74     update(pos, val, version[ver]);
75 }
76
77 Node query(int ql, int qr, int idx, int ll = 0, int rr =
78     n){
79     auto &now = arr[idx];
80
81     if (ql<=ll && rr<=qr) return now;
82     if (rr<=ql || qr<=ll) return Node();
83
84     int mid = (ll+rr)/2;
85
86     Node ret;
87     pull(ret, query(ql, qr, now.lc, ll, mid), query(ql,
88         qr, now.rc, mid, rr));

```

```

84     return ret;
85 }
86
87 Node query_version(int ql, int qr, int ver){
88     return query(ql, qr, version[ver]);
89 }
90
91 void clone_version(int ver){
92     version.push_back(node_cnt);
93     arr[node_cnt] = arr[version[ver]];
94     node_cnt++;
95 }
96 };

```

3.9 Sparse Table [31f22a]

```

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

```

3.10 Treap2 [3b0cca]

```

1 // 1-based · 請注意 MAX_N 是否足夠大
2 int root = 0;
3 int lc[MAX_N], rc[MAX_N];
4 int pri[MAX_N], val[MAX_N];
5 int sz[MAX_N], tag[MAX_N], fa[MAX_N], total[MAX_N];
6 // tag 為不包含自己 (僅要給子樹) 的資訊
7 int nodeCnt = 0;
8 int& new_node(int v){
9     nodeCnt++;
10    val[nodeCnt] = v;
11    total[nodeCnt] = v;
12    sz[nodeCnt] = 1;
13    pri[nodeCnt] = rand();
14    return nodeCnt;
15 }
16
17 void apply(int x, int V){
18     val[x] += V;
19     tag[x] += V;
20     total[x] += V*sz[x];
21 }
22
23 void push(int x){

```

```

24     if (tag[x]){
25         if (lc[x]) apply(lc[x], tag[x]);
26         if (rc[x]) apply(rc[x], tag[x]);
27     }
28     tag[x] = 0;
29 }
30 int pull(int x){
31     if (x){
32         fa[x] = 0;
33         sz[x] = 1+sz[lc[x]]+sz[rc[x]];
34         total[x] = val[x]+total[lc[x]]+total[rc[x]];
35         if (lc[x]) fa[lc[x]] = x;
36         if (rc[x]) fa[rc[x]] = x;
37     }
38     return x;
39 }
40
41 int merge(int a, int b){
42     if (!a || !b) return a|b;
43     push(a), push(b);
44
45     if (pri[a]>pri[b]){
46         rc[a] = merge(rc[a], b);
47         return pull(a);
48     }else{
49         lc[b] = merge(a, lc[b]);
50         return pull(b);
51     }
52 }
53
54 // [1, k] [k+1, n]
55 void split(int x, int k, int &a, int &b) {
56     if (!x) return a = b = 0, void();
57     push(x);
58     if (sz[lc[x]] >= k) {
59         split(lc[x], k, a, lc[x]);
60         b = x;
61         pull(a); pull(b);
62     }else{
63         split(rc[x], k - sz[lc[x]] - 1, rc[x], b);
64         a = x;
65         pull(a); pull(b);
66     }
67 }
68
69 // functions
70 // 回傳 x 在 Treap 中的位置
71 int get_pos(int x){
72     vector<int> sta;
73     while (fa[x]){
74         sta.push_back(x);
75         x = fa[x];
76     }
77     while (sta.size()){
78         push(x);
79         x = sta.back();
80         sta.pop_back();
81     }
82     push(x);
83
84     int res = sz[x] - sz[rc[x]];
85     while (fa[x]){
86         if (rc[fa[x]]==x){
87             res += sz[fa[x]]-sz[x];
88         }

```

```

89     x = fa[x];
90 }
91 return res;
92 }
93
94 // 1-based <前 [1, l-1] 個元素, [l, r] 個元素, [r+1, n] 個元素>
95 array<int, 3> cut(int x, int l, int r){
96     array<int, 3> ret;
97     split(x, r, ret[1], ret[2]);
98     split(ret[1], l-1, ret[0], ret[1]);
99     return ret;
100 }
101
102 void print(int x){
103     push(x);
104     if (lc[x]) print(lc[x]);
105     cerr << val[x] << " ";
106     if (rc[x]) print(rc[x]);
107 }

```

3.11 Trie [b6475c]

```

1 struct Trie{
2     struct Data{
3         int nxt[2]={0, 0};
4     };
5
6     int sz=0;
7     vector<Data> arr;
8
9     void init(int n){
10         arr.resize(n);
11     }
12
13     void insert(int n){
14         int now=0;
15         for (int i=N; i>=0; i--){
16             int v=(n>>i)&1;
17             if (!arr[now].nxt[v]){
18                 arr[now].nxt[v]=++sz;
19             }
20             now=arr[now].nxt[v];
21         }
22     }
23
24     int query(int n){
25         int now=0, ret=0;
26         for (int i=N; i>=0; i--){
27             int v=(n>>i)&1;
28             if (arr[now].nxt[1-v]){
29                 ret+=(1<<i);
30                 now=arr[now].nxt[1-v];
31             }else if (arr[now].nxt[v]){
32                 now=arr[now].nxt[v];
33             }else{
34                 return ret;
35             }
36         }
37         return ret;
38     }
39 }
40 } tr;

```

4 Dynamic-Programming

4.1 Digit DP [133f00]

```

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
6 // 位 · pos 前一位是 pre · (是/否) 有上界 · (是/否) 有前綴零
7 // 的答案數量
8
9 long long memorize_search(string &s, int pos, int pre, bool
10 limit, bool lead){
11
12     // 已經被找過了 · 直接回傳值
13     if (dp[pos][pre][limit][lead]!=-1) return dp[pos][pre][
14 limit][lead];
15
16     // 已經搜尋完畢 · 紀錄答案並回傳
17     if (pos==(int)s.size()){
18         return dp[pos][pre][limit][lead] = 1;
19     }
20
21     // 枚舉目前的位數數字是多少
22     if (pre!=0) continue;
23
24     // 如果已經不在前綴零的範圍內 · 0 不能連續出現
25     if (lead==false) continue;
26
27 }
28
29 ans += memorize_search(s, pos+1, now, limit&(now==(s[
30 pos]-'0')), lead&(now==0));
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 Knapsack On Tree [df69b1]

```

1 // 需要重構、需要增加註解
2 #include <bits/stdc++.h>
3 #define F first
4 #define S second
5 #define all(x) begin(x), end(x)
6 using namespace std;
7
8 #define chmax(a, b) (a) = (a) < (b) ? (b) : (a)
9 #define chmin(a, b) (a) = (a) < (b) ? (a) : (b)
10
11 #define ll long long
12
13 #define FOR(i, a, b) for (int i = a; i <= b; i++)
14
15 int N, W, cur;
16 vector<int> w, v, sz;
17 vector<vector<int>> adj, dp;
18
19 void dfs(int x) {
20     sz[x] = 1;
21     for (int i : adj[x]) dfs(i), sz[x] += sz[i];
22     cur++;
23     // choose x
24     for (int i=w[x]; i<=W; i++){
25         dp[cur][i] = dp[cur - 1][i - w[x]] + v[x];
26     }
27     // not choose x
28     for (int i=0; i<=W; i++){
29         chmax(dp[cur][i], dp[cur - sz[x]][i]);
30     }
31 }
32
33 signed main() {
34     cin >> N >> W;
35     adj.resize(N + 1);
36     w.assign(N + 1, 0);
37     v.assign(N + 1, 0);
38     sz.assign(N + 1, 0);
39     dp.assign(N + 2, vector<int>(W + 1, 0));
40     for (int i=1; i<=N; i++){
41         int p; cin >> p;
42         adj[p].push_back(i);
43     }
44
45     for (int i=1; i<=N; i++) cin >> w[i];
46     for (int i=1; i<=N; i++) cin >> v[i];
47     dfs(0);
48     cout << dp[N + 1][W] << "\n";
49 }

```

4.3 SOS DP [8dfa8b]

```

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 }

```

4.4 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 [d9966f]

```
1 using ld = double;
2
3 // 判斷數值正負：{1:正數,0:零,-1:負數}
4 int sign(long long x) {return (x >= 0) ? ((bool)x) : -1; }
5 int sign(ld x) {return (abs(x) < 1e-9) ? 0 : (x>0 ? 1 : -1); }
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     explicit operator point<ld>() {return point<ld>(x, y); }
13     // A [6357c4], Line 9 ~ 13
14     point operator+(point b) {return {x+b.x, y+b.y}; }
15     point operator-(point b) {return {x-b.x, y-b.y}; }
16     point operator*(T b) {return {x*b, y*b}; }
17     point operator/(T b) {return {x/b, y/b}; }
18     bool operator==(point b) {return x==b.x && y==b.y; }
19
20     T operator*(point b) {return x * b.x + y * b.y; }
21     T operator^(point b) {return x * b.y - y * b.x; }
22     // B [c415da], Line 14 ~ 22
23     // 逆時針極角排序
24     bool side() const{return (y == 0) ? (x > 0) : (y < 0); }
25     bool operator<(const point &b) const {
26         return side() == b.side() ?
27             (x*b.y > b.x*y) : side() < b.side();
28     }
29     friend ostream& operator<<(ostream& os, point p) {
30         return os << "(" << p.x << ", " << p.y << ")";
31     }
32     // 判斷 ab 到 ac 的方向：{1:逆時鐘,0:重疊,-1:順時鐘}
33     friend int ori(point a, point b, point c) {
34         return sign((b-a)^(c-a));
35     }
36     friend bool btw(point a, point b, point c) {
37         return ori(a, b, c) == 0 && sign((a-c)*(b-c)) <= 0;
38     }
39     // 判斷線段 ab, cd 是否相交
40     friend bool banana(point a, point b, point c, point d) {
41         if (btw(a, b, c) || btw(a, b, d)
42             || btw(c, d, a) || btw(c, d, b)) return true;
43         int u = ori(a, b, c) * ori(a, b, d);
44         int v = ori(c, d, a) * ori(c, d, b);
45         return u < 0 && v < 0;
46     } // C [09fd7c], only this function
47     // 判斷 "射線 ab" 與 "線段 cd" 是否相交
48     friend bool rayHitSeg(point a, point b, point c, point d) {
49         if (a == b) return btw(c, d, a); // Special case
50         if (((a - b) ^ (c - d)) == 0) {
51             return btw(a, c, b) || btw(a, d, b) || banana(a,
52                 b, c, d);
53         }
54         point u = b - a, v = d - c, s = c - a;
55         return sign(s ^ u) * sign(u ^ v) >= 0 && sign(s ^ u)
56             * sign(u ^ v) >= 0 && abs(s ^ u) <= abs(u ^ v);
57     } // D [db541a], only this function
58     // 旋轉 Arg(b) 的角度 (小心溢位)
```

```
58 point rotate(point b){return {x*b.x-y*b.y, x*b.y+y*b.x}; }
59 // 回傳極座標角度·值域：[-π, +π]
60 friend ld Arg(point b) {
61     return (b.x != 0 || b.y != 0) ? atan2(b.y, b.x) : 0;
62 }
63 friend T abs2(point b) {return b * b; }
64 };
65
66 template<typename T>
67 struct line {
68     point<T> p1, p2;
69     // ax + by + c = 0
70     T a, b, c; // |a|, |b| ≤ 2C, |c| ≤ 8C²
71     line() {}
72     line(const point<T> &x, const point<T> &y) : p1(x), p2(y){
73         build();
74     }
75     void build() {
76         a = p1.y - p2.y;
77         b = p2.x - p1.x;
78         c = (-a*p1.x)-b*p1.y;
79     } // E [683239], Line 68 ~ 79
80     // 判斷點和有向直線的關係：{1:左邊,0:在線上,-1:右邊}
81     int ori(point<T> &p) {
82         return sign((p2-p1) ^ (p-p1));
83     }
84     // 判斷直線斜率是否相同
85     bool parallel(line &l) {
86         return ((p1-p2) ^ (l.p1-l.p2)) == 0;
87     }
88     // 兩直線交點
89     point<ld> line_intersection(line &l) {
90         using P = point<ld>;
91         point<T> u = p2-p1, v = l.p2-l.p1, s = l.p1-p1;
92         return P(p1) + P(u) * ((ld(s^v)) / (u^v));
93     }
94 };
95
96 template<typename T>
97 struct polygon {
98     vector<point<T>> v;
99     polygon() {}
100     polygon(const vector<point<T>> &u) : v(u) {}
101     // simple 為 true 的時候會回傳任意三點不共線的凸包
102     void make_convex_hull(int simple) {
103         auto cmp = [&](point<T> &p, point<T> &q) {
104             return (p.x == q.x) ? (p.y < q.y) : (p.x < q.x);
105         };
106         simple = (bool)simple;
107         sort(v.begin(), v.end(), cmp);
108         v.resize(unique(v.begin(), v.end()) - v.begin());
109         if (v.size() <= 1) return;
110         vector<point<T>> hull;
111         for (int t = 0; t < 2; ++t){
112             int sz = hull.size();
113             for (auto &i:v) {
114                 while (hull.size() >= sz+2 && ori(hull[hull.
115                     size()-2], hull.back(), i) < simple) {
116                     hull.pop_back();
117                 }
118                 hull.push_back(i);
119             }
120             hull.pop_back();
121             reverse(v.begin(), v.end());
122         }
```

```
123 swap(hull, v);
124 } // F [2bb3ef], only this function
125 // 可以在有 n 個點的簡單多邊形內·用 O(n) 判斷一個點：
126 // {1: 在多邊形內, 0: 在多邊形上, -1: 在多邊形外}
127 int in_polygon(point<T> a){
128     const T MAX_POS = 1e9 + 5; // [記得修改] 座標的最大值
129     point<T> pre = v.back(), b(MAX_POS, a.y + 1);
130     int cnt = 0;
131
132     for (auto &i:v) {
133         if (btw(pre, i, a)) return 0;
134         if (banana(a, b, pre, i)) cnt++;
135         pre = i;
136     }
137     return cnt%2 ? 1 : -1;
138 } // G [f11340], only this function
139 /// 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
140 /// 共線的凸包 ///
141 // 可以在有 n 個點的凸包內·用 O(Log n) 判斷一個點：
142 // {1: 在凸包內, 0: 在凸包邊上, -1: 在凸包外}
143 int in_convex(point<T> p) {
144     int n = v.size();
145     int a = ori(v[0], v[1], p), b = ori(v[0], v[n-1], p);
146     if (a < 0 || b > 0) return -1;
147     if (btw(v[0], v[1], p)) return 0;
148     if (btw(v[0], v[n-1], p)) return 0;
149     int l = 1, r = n - 1, mid;
150     while (l + 1 < r) {
151         mid = (l + r) >> 1;
152         if (ori(v[0], v[mid], p) >= 0) l = mid;
153         else r = mid;
154     }
155     int k = ori(v[l], v[r], p);
156     if (k <= 0) return k;
157     return 1;
158 } // H [e64f1e], only this function
159 // 凸包專用的環狀二分搜·回傳 0-based index
160 int cycle_search(auto &f) {
161     int n = v.size(), l = 0, r = n;
162     if (n == 1) return 0;
163     bool rv = f(l, 0);
164     while (r - l > 1) {
165         int m = (l + r) / 2;
166         if (f(0, m) ? rv : f(m, (m + 1) % n)) r = m;
167         else l = m;
168     }
169     return f(l, r % n) ? l : r % n;
170 } // I [fe2f51], only this function
171 // 可以在有 n 個點的凸包內·用 O(Log n) 判斷一條直線：
172 // {1: 穿過凸包, 0: 剛好切過凸包, -1: 沒碰到凸包}
173 int line_cut_convex(line<T> L) {
174     L.build();
175     point<T> p(L.a, L.b);
176     auto gt = [&](int neg) {
177         auto f = [&](int x, int y) {
178             return sign((v[x] - v[y]) * p) == neg;
179         };
180         return -(v[cycle_search(f)] * p);
181     };
182     T x = gt(1), y = gt(-1);
183     if (L.c < x || y < L.c) return -1;
184     return not (L.c == x || L.c == y);
185 } // J [b6a4c8], only this function
```

```

186 // 可以在有  $n$  個點的凸包內，用  $O(\log n)$  判斷一個線段：
187 // {1 : 存在一個凸包上的邊可以把這個線段切成兩半，
188 // 0 : 有碰到凸包但沒有任何凸包上的邊可以把它切成兩半，
189 // -1 : 沒碰到凸包}
190 /// 除非線段兩端點都不在凸包邊上，否則此函數回傳 0 的時候不一
    定表示線段沒有通過凸包內部 ///
191 int segment_across_convex(line<T> L) {
192     L.build();
193     point<T> p(L.a, L.b);
194     auto gt = [&](int neg) {
195         auto f = [&](int x, int y) {
196             return sign((v[x] - v[y]) * p) == neg;
197         };
198         return cycle_search(f);
199     };
200     int i = gt(1), j = gt(-1), n = v.size();
201     T x = -(v[i] * p), y = -(v[j] * p);
202     if (L.c < x || y < L.c) return -1;
203     if (L.c == x || L.c == y) return 0;
204
205     if (i > j) swap(i, j);
206     auto g = [&](int x, int lim) {
207         int now = 0, nxt;
208         for (int i = 1 << __lg(lim); i > 0; i /= 2) {
209             if (now + i > lim) continue;
210             nxt = (x + i) % n;
211             if (L.ori(v[x]) * L.ori(v[nxt]) >= 0) {
212                 x = nxt;
213                 now += i;
214             }
215         } // ↓ BE CAREFUL
216         return -(ori(v[x], v[(x + 1) % n], L.p1) * ori(v[
217             x], v[(x + 1) % n], L.p2));
218     };
219     return max(g(i, j - i), g(j, n - (j - i)));
220 } // K [b4f073], only this function
221 // 可以在有  $n$  個點的凸包內，用  $O(\log n)$  判斷一個線段：
222 // {1 : 線段上存在某一點位於凸包內部（邊上不算），
223 // 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
    內部，
224 // -1 : 線段完全在凸包外面}
225 int segment_pass_convex_interior(line<T> L) {
226     if (in_convex(L.p1) == 1 || in_convex(L.p2) == 1)
227         return 1;
228     L.build();
229     point<T> p(L.a, L.b);
230     auto gt = [&](int neg) {
231         auto f = [&](int x, int y) {
232             return sign((v[x] - v[y]) * p) == neg;
233         };
234         return cycle_search(f);
235     };
236     int i = gt(1), j = gt(-1), n = v.size();
237     T x = -(v[i] * p), y = -(v[j] * p);
238     if (L.c < x || y < L.c) return -1;
239     if (L.c == x || L.c == y) return 0;
240
241     if (i > j) swap(i, j);
242     auto g = [&](int x, int lim) {
243         int now = 0, nxt;
244         for (int i = 1 << __lg(lim); i > 0; i /= 2) {
245             if (now + i > lim) continue;
246             nxt = (x + i) % n;
247             if (L.ori(v[x]) * L.ori(v[nxt]) > 0) {

```

```

248                 x = nxt;
249                 now += i;
250             }
251         } // ↓ BE CAREFUL
252         return -(ori(v[x], v[(x + 1) % n], L.p1) * ori(v[
253             x], v[(x + 1) % n], L.p2));
254     };
255     return max(g(i, j - i), g(j, n - (j - i)));
256 } // L [5f45ca], only this function
257 /// 回傳點過凸包的兩條切線的切點的  $\theta$ -based index (不保證兩條
    切線的順逆時針關係)
258 pair<int, int> convex_tangent_point(point<T> p) {
259     int n = v.size(), z = -1, edg = -1;
260     auto gt = [&](int neg) {
261         auto check = [&](int x) {
262             if (v[x] == p) z = x;
263             if (btw(v[x], v[(x + 1) % n], p)) edg = x;
264             if (btw(v[(x + n - 1) % n], v[x], p)) edg = (
265                 x + n - 1) % n;
266         };
267         auto f = [&](int x, int y) {
268             check(x); check(y);
269             return ori(p, v[x], v[y]) == neg;
270         };
271         return cycle_search(f);
272     };
273     int x = gt(1), y = gt(-1);
274     if (z != -1) {
275         return {(z + n - 1) % n, (z + 1) % n};
276     }
277     else if (edg != -1) {
278         return {edg, (edg + 1) % n};
279     }
280     else {
281         return {x, y};
282     }
283 } // M [a6f66b], only this function
284 friend int halfplane_intersection(vector<line<T>> &s,
285     polygon<T> &P) {
286     auto angle_cmp = [&](line<T> &A, line<T> &B) {
287         point<T> a = A.p2 - A.p1, b = B.p2 - B.p1;
288         return (a < b);
289     };
290     sort(s.begin(), s.end(), angle_cmp); // 線段左側為該
    線段半平面
291     int L, R, n = s.size();
292     vector<point<T>> px(n);
293     vector<line<T>> q(n);
294     q[L = R = 0] = s[0];
295     for (int i = 1; i < n; ++i) {
296         while (L < R && s[i].ori(px[R-1]) <= 0) --R;
297         while (L < R && s[i].ori(px[L]) <= 0) ++L;
298         q[++R] = s[i];
299         if (q[R].parallel(q[R-1])) {
300             --R;
301             if (q[R].ori(s[i].p1) > 0) q[R] = s[i];
302         }
303         if (L < R) px[R-1] = q[R-1].line_intersection(q[R]);
304     }
305     while (L < R && q[L].ori(px[R-1]) <= 0) --R;
306     P.v.clear();
307     if (R - L <= 1) return 0;
308     px[R] = q[R].line_intersection(q[L]);

```

```

309     for (int i = L; i <= R; ++i) P.v.push_back(px[i]);
310     return R - L + 1;
311 } // N [102d48], only this function
312 };
313
314 struct Cir {
315     point<ld> o; ld r;
316     friend ostream& operator<<(ostream& os, Cir c) {
317         return os << "(x" << "+"-[c.o.x >= 0] << abs(c.o.x)
318             << ")^2 + (y" << "+"-[c.o.y >= 0] << abs(c.o.y)
319             << ")^2 = " << c.r * c.r;
320     }
321     bool covers(Cir b) {
322         return sqrt((ld)abs2(o - b.o)) + b.r <= r;
323     }
324     vector<point<ld>> Cir_intersect(Cir c) {
325         ld d2 = abs2(o - c.o), d = sqrt(d2);
326         if (d < max(r, c.r) - min(r, c.r) || d > r + c.r)
327             return {};
328         auto sqdf = [&](ld x, ld y) { return x*x - y*y; };
329         point<ld> u = (o + c.o) / 2 + (o - c.o) * (sqdf(c.r,
330             r) / (2 * d2));
331         ld A = sqrt(sqdf(r + d, c.r) * sqdf(c.r, d - r));
332         point<ld> v = (c.o - o).rotate({0, 1}) * A / (2 * d2);
333         if (sign(v.x) == 0 && sign(v.y) == 0) return {u};
334         return {u - v, u + v};
335     } // O [330a1c], only this function
336     auto point_tangent(point<ld> p) {
337         vector<point<ld>> res;
338         ld d_sq = abs2(p - o);
339         if (sign(d_sq - r * r) <= 0) {
340             res.pb(p + (p - o).rotate({0, 1}));
341         } else if (d_sq > r * r) {
342             ld s = d_sq - r * r;
343             point<ld> v = p + (o - p) * s / d_sq;
344             point<ld> u = (o - p).rotate({0, 1}) * sqrt(s) *
345                 r / d_sq;
346             res.pb(v + u);
347             res.pb(v - u);
348         }
349         return res;
350     } // P [0067e6], only this function
351 };

```

5.2 Geometry Struct 3D [4a50c9]

```

1 using ld = long double;
2
3 // 判斷數值正負：{1:正數,0:零,-1:負數}
4 int sign(long long x) {return (x >= 0) ? ((bool)x) : -1; }
5 int sign(ld x) {return (abs(x) < 1e-9) ? 0 : (x>0 ? 1 : -1); }
6
7 template<typename T>
8 struct pt3 {
9     T x, y, z;
10     pt3(){}
11     pt3(const T &x, const T &y, const T &z):x(x),y(y),z(z){}
12     explicit operator pt3<ld>() {return pt3<ld>(x, y, z); }
13
14     pt3 operator+(pt3 b) {return {x+b.x, y+b.y, z+b.z}; }
15     pt3 operator-(pt3 b) {return {x-b.x, y-b.y, z-b.z}; }
16     pt3 operator*(T b) {return {x * b, y * b, z * b}; }
17     pt3 operator/(T b) {return {x / b, y / b, z / b}; }
18     bool operator==(pt3 b){return x==b.x&&y==b.y&&z==b.z; }
19

```

```

20 T operator*(pt3 b) {return x*b.x+y*b.y+z*b.z; }
21 pt3 operator^(pt3 b) {
22     return pt3(y * b.z - z * b.y,
23               z * b.x - x * b.z,
24               x * b.y - y * b.x);
25 }
26 friend T abs2(pt3 b) {return b * b; }
27 friend T len (pt3 b) {return sqrt(abs2(b)); }
28 friend ostream& operator<<(ostream& os, pt3 p) {
29     return os << "(" << p.x << ", " <<
30               p.y << ", " << p.z << ")";
31 }
32 };
33
34 template<typename T>
35 struct face {
36     int a, b, c; // 三角形在 vector 裡面的 index
37     pt3<T> q;    // 面積向量 ( 朝外 )
38 };
39
40 /// 警告 ; v 在過程中可能被修改 · 回傳的 face 以修改後的為準
41 ///  $O(n^2)$  · 最多只有  $2n-4$  個面
42 /// 當凸包退化時會回傳空的凸包 · 否則回傳凸包上的每個面
43 template<typename T>
44 vector<face<T>> hull3(vector<pt3<T>> &v) {
45     int n = v.size();
46     if (n < 3) return {};
47     // don't use "==" when you use Ld
48     sort(all(v), [&](pt3<T> &p, pt3<T> &q) {
49         return sign(p.x - q.x) ? (p.x < q.x) :
50                (sign(p.y - q.y) ? p.y < q.y : p.z < q.z);
51     });
52     v.resize(unique(v.begin(), v.end()) - v.begin());
53     for (int i = 2; i <= n; ++i) {
54         if (i == n) return {};
55         if (sign(len((v[1] - v[0]) ^ (v[i] - v[0])))) {
56             swap(v[2], v[i]);
57             break;
58         }
59     }
60     pt3<T> tmp_q = (v[1] - v[0]) ^ (v[2] - v[0]);
61     for (int i = 3; i <= n; ++i) {
62         if (i == n) return {};
63         if (sign((v[i] - v[0]) * tmp_q)) {
64             swap(v[3], v[i]);
65             break;
66         }
67     }
68 }
69
70 vector<face<T>> f;
71 vector<vector<int>> dead(n, vector<int>(n, true));
72 auto add_face = [&](int a, int b, int c) {
73     f.emplace_back(a, b, c, (v[b] - v[a]) ^ (v[c] - v[a]));
74     dead[a][b] = dead[b][c] = dead[c][a] = false;
75 };
76 add_face(0, 1, 2);
77 add_face(0, 2, 1);
78
79 for (int i = 3; i < n; ++i) {
80     vector<face<T>> f2;
81     for (auto &[a, b, c, q] : f) {
82         if (sign((v[i] - v[a]) * q) > 0)
83             dead[a][b] = dead[b][c] = dead[c][a] = true;
84         else f2.emplace_back(a, b, c, q);
85     }
86 }

```

```

84 }
85 f.clear();
86 for (face<T> &f : f2) {
87     int arr[3] = {F.a, F.b, F.c};
88     for (int j = 0; j < 3; ++j) {
89         int a = arr[j], b = arr[(j + 1) % 3];
90         if (dead[b][a]) add_face(b, a, i);
91     }
92 }
93 f.insert(f.end(), all(f2));
94 }
95 return f;
96 } // 15ef50

```

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 [5a6317]

```

1 struct TWO_SAT {
2     int n, N;
3     vector<vector<int>> G, rev_G;
4     deque<bool> used;
5     vector<int> order, comp;
6     deque<bool> assignment;
7     void init(int _n) {
8         n = _n;
9         N = _n * 2;
10        G.resize(N + 5);
11        rev_G.resize(N + 5);
12    }
13    void dfs1(int v) {
14        used[v] = true;
15        for (int u : G[v]) {
16            if (!used[u])
17                dfs1(u);
18        }
19        order.push_back(v);
20    }
21    void dfs2(int v, int c1) {
22        comp[v] = c1;
23        for (int u : rev_G[v]) {
24            if (comp[u] == -1)
25                dfs2(u, c1);
26        }
27    }
28    bool solve() {
29        order.clear();
30        used.assign(N, false);
31        for (int i = 0; i < N; ++i) {
32            if (!used[i])
33                dfs1(i);
34        }
35        comp.assign(N, -1);
36        for (int i = 0, j = 0; i < N; ++i) {
37            int v = order[N - i - 1];
38            if (comp[v] == -1)
39                dfs2(v, j++);
40        }
41        assignment.assign(n, false);
42        for (int i = 0; i < N; i += 2) {
43            if (comp[i] == comp[i + 1])
44                return false;
45        }
46    }
47 }

```

```

45         assignment[i / 2] = (comp[i] > comp[i + 1]);
46     }
47     return true;
48 }
49 // A or B 都是 0-based
50 void add_disjunction(int a, bool na, int b, bool nb) {
51     // na is true => ~a, na is false => a
52     // nb is true => ~b, nb is false => b
53     a = 2 * a ^ na;
54     b = 2 * b ^ nb;
55     int neg_a = a ^ 1;
56     int neg_b = b ^ 1;
57     G[neg_a].push_back(b);
58     G[neg_b].push_back(a);
59     rev_G[b].push_back(neg_a);
60     rev_G[a].push_back(neg_b);
61     return;
62 }
63 void get_result(vector<int>& res) {
64     res.clear();
65     for (int i = 0; i < n; i++)
66         res.push_back(assignment[i]);
67 }
68 };

```

6.2 Augment Path [f8a5dd]

```

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
13     void add(int x, int y){
14         G[x].push_back(y);
15     }
16
17     // bb03e2
18     bool dfs1(int now){
19         visx[now] = stamp;
20
21         for (auto x : G[now]){
22             if (my[x]==-1){
23                 mx[now] = x;
24                 my[x] = now;
25                 return true;
26             }
27         }
28         for (auto x : G[now]){
29             if (visx[my[x]]!=stamp && dfs1(my[x])){
30                 mx[now] = x;
31                 my[x] = now;
32                 return true;
33             }
34         }
35         return false;
36     }
37
38     vector<pair<int, int>> find_max_matching(){
39
40     }
41 }

```

```

38     vector<pair<int, int>> ret;
39
40     while (true){
41         stamp++;
42         int tmp = 0;
43         for (int i=0 ; i<n ; i++){
44             if (mx[i]==-1 && dfs1(i)) tmp++;
45         }
46         if (tmp==0) break;
47     }
48
49     for (int i=0 ; i<n ; i++){
50         if (mx[i]!=-1){
51             ret.push_back({i, mx[i]});
52         }
53     }
54     return ret;
55 }
56
57 // 645577
58 void dfs2(int now){
59     visx[now] = true;
60
61     for (auto x : G[now]){
62         if (my[x]!=-1 && visy[x]==false){
63             visy[x] = true;
64             dfs2(my[x]);
65         }
66     }
67 }
68
69 // 要先執行 find_max_matching 一次
70 vector<pair<int, int>> find_min_vertex_cover(){
71     fill(visx.begin(), visx.end(), false);
72     fill(visy.begin(), visy.end(), false);
73
74     vector<pair<int, int>> ret;
75     for (int i=0 ; i<n ; i++){
76         if (mx[i]==-1) dfs2(i);
77     }
78
79     for (int i=0 ; i<n ; i++){
80         if (visx[i]==false) ret.push_back({1, i});
81     }
82     for (int i=0 ; i<m ; i++){
83         if (visy[i]==true) ret.push_back({2, i});
84     }
85
86     return ret;
87 }
88 };

```

6.3 C3C4 [d00465]

```

1 // 0-based
2 void C3C4(vector<int> deg, vector<array<int, 2>> edges){
3     int N = deg.size();
4     int M = deges.size();
5
6     vector<int> ord(N), rk(N);
7     iota(ord.begin(), ord.end(), 0);
8     sort(ord.begin(), ord.end(), [&](int x, int y) { return
9         deg[x] > deg[y]; });
10    for (int i=0 ; i<N ; i++) rk[ord[i]] = i;

```

```

11    vector<vector<int>> D(N), adj(N);
12    for (auto [u, v] : e) {
13        if (rk[u] > rk[v]) swap(u, v);
14        D[u].emplace_back(v);
15        adj[u].emplace_back(v);
16        adj[v].emplace_back(u);
17    }
18
19    vector<int> vis(N);
20
21    int c3 = 0, c4 = 0;
22    for (int x : ord) { // c3
23        for (int y : D[x]) vis[y] = 1;
24        for (int y : D[x]) for (int z : D[y]){
25            c3 += vis[z]; // xyz is C3
26        }
27        for (int y : D[x]) vis[y] = 0;
28    }
29    for (int x : ord) { // c4
30        for (int y : D[x]) for (int z : adj[y])
31            if (rk[z] > rk[x]) c4 += vis[z]++;
32        for (int y : D[x]) for (int z : adj[y])
33            if (rk[z] > rk[x]) --vis[z];
34    } // both are O(M*sqrt(M)), test @ 2022 CCPC guangzhou
35    cout << c4 << "\n";
36 }

```

6.4 Cut BCC [2af809]

```

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 Dinic [961b34]

```

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
36     // e22e39
37     T flow(int s, int t){
38         T ans = 0;
39         while (true){
40             fill(dis.begin(), dis.end(), INF);
41             queue<int> q;
42             q.push(s);
43             dis[s] = 0;
44
45             while (q.size()){
46                 int u = q.front(); q.pop();
47                 for (auto [v, rc, rid] : G[u]){
48                     if (rc <= 0 || dis[v] < INF) continue;
49                     dis[v] = dis[u] + 1;
50                     q.push(v);
51                 }
52             }
53             if (dis[t]==INF) break;
54
55             fill(it.begin(), it.end(), 0);
56             while (true){
57                 T df = dfs(s, t, INF);
58                 if (df <= 0) break;
59                 ans += df;
60             }
61         }

```

```

62     return ans;
63 }
64
65 // the code below constructs minimum cut
66 void dfs_mincut(int now, vector<bool> &vis){
67     vis[now] = true;
68     for (auto &[v, rc, rid] : G[now]){
69         if (vis[v] == false && rc > 0){
70             dfs_mincut(v, vis);
71         }
72     }
73 }
74
75 vector<pair<int, int>> construct(int n, int s, vector<
76     pair<int, int>> &E){
77     // E is G without capacity
78     vector<bool> vis(n);
79     dfs_mincut(s, vis);
80     vector<pair<int, int>> ret;
81     for (auto &[u, v] : E){
82         if (vis[u] == true && vis[v] == false){
83             ret.emplace_back(u, v);
84         }
85     }
86     return ret;
87 };

```

6.6 Dominator Tree [52b249]

```

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

```

```

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

```

6.7 EdgeBCC [d09eb1]

```

1 // d09eb1
2 // 0-based · 支援重邊
3 struct EdgeBCC{
4     int n, m, dep, sz;
5     vector<vector<pair<int, int>>> G;
6     vector<vector<int>> bcc;

```

```

7     vector<int> dfn, low, stk, isBridge, bccId;
8     vector<pair<int, int>> edge, bridge;
9
10    EdgeBCC(int _n) : n(_n), m(0), sz(0), dfn(n), low(n), G(n
11        ), bcc(n), bccId(n) {}
12
13    void add_edge(int u, int v) {
14        edge.push_back({u, v});
15        G[u].push_back({v, m});
16        G[v].push_back({u, m++});
17    }
18
19    void dfs(int now, int pre) {
20        dfn[now] = low[now] = ++dep;
21        stk.push_back(now);
22
23        for (auto [x, id] : G[now]){
24            if (!dfn[x]){
25                dfs(x, id);
26                low[now] = min(low[now], low[x]);
27            }else if (id!=pre){
28                low[now] = min(low[now], dfn[x]);
29            }
30        }
31
32        if (low[now]==dfn[now]){
33            if (pre!=-1) isBridge[pre] = true;
34            int u;
35            do{
36                u = stk.back();
37                stk.pop_back();
38                bcc[sz].push_back(u);
39                bccId[u] = sz;
40            } while (u!=now);
41            sz++;
42        }
43    }
44
45    void get_bcc() {
46        isBridge.assign(m, 0);
47        dep = 0;
48        for (int i=0 ; i<n ; i++){
49            if (!dfn[i]) dfs(i, -1);
50        }
51
52        for (int i=0 ; i<m ; i++){
53            if (isBridge[i]){
54                bridge.push_back({edge[i].first , edge[i].
55                    second});
56            }
57        }
58    }
59 };

```

6.8 EnumeratePlanarFace [e70ee1]

```

1 // 0-based
2 struct PlanarGraph{
3     int n, m, id;
4     vector<point<int>> v;
5     vector<vector<pair<int, int>>> G;
6     vector<int> conv, nxt, vis;
7
8     PlanarGraph(int n, int m, vector<point<int>> _v) :
9         n(n), m(m), id(0),

```



```

10 v(_v), G(n),
11 conv(2*m), nxt(2*m), vis(2*m) {}
12
13 void add_edge(int x, int y){
14     G[x].push_back({y, 2*id});
15     G[y].push_back({x, 2*id+1});
16     conv[2*id] = x;
17     conv[2*id+1] = y;
18     id++;
19 }
20
21 vector<int> enumerate_face(){
22     for (int i=0; i<n; i++){
23         sort(G[i].begin(), G[i].end(), [&](pair<int, int>
24             a, pair<int, int> b){
25             return (v[a.first]-v[i])<(v[b.first]-v[i]);
26             });
27
28         int sz = G[i].size(), pre = sz-1;
29         for (int j=0; j<sz; j++){
30             nxt[G[i][pre].second] = G[i][j].second^1;
31             pre = j;
32         }
33     }
34
35     vector<int> ret;
36     for (int i=0; i<2*m; i++){
37         if (vis[i]==false){
38             int area = 0, now = i;
39             vector<int> pt;
40
41             while (!vis[now]){
42                 vis[now] = true;
43                 pt.push_back(conv[now]);
44                 now = nxt[now];
45             }
46
47             pt.push_back(pt.front());
48             for (int i=0; i+1<pt.size(); i++){
49                 area += (v[pt[i]]^v[pt[i+1]]);
50             }
51
52             // pt = face boundary
53             if (area>0){
54                 ret.push_back(area);
55             }else{
56                 // pt is outer face
57             }
58         }
59     }
60     return ret;
61 };

```

6.9 HLD [f57ec6]

```

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

```

6.10 Kosaraju [c7d5aa]

```

1 // 給定一個有向圖，迴傳傳縮點後的圖、SCC 的資訊
2 // 所有點都以 based-0 編號
3 // 函式：
4 // .compress: O(n log n) 計算 G3、SCC、SCC_id 的資訊，並把縮
5 // 點後的結果存在 result 裡
6 // SCC[i] = 某個 SCC 中的所有點
7 // SCC_id[i] = 第 i 個點在第幾個 SCC
8 struct SCC_compress{
9     int N, M, sz;
10    vector<vector<int>> G, inv_G, result;
11    vector<pair<int, int>> edges;
12    vector<bool> vis;
13    vector<int> order;
14
15    vector<vector<int>> SCC;
16    vector<int> SCC_id;
17
18    SCC_compress(int _N) :
19        N(_N), M(0), sz(0),
20        G(N), inv_G(N),
21        vis(N), SCC_id(N) {}
22
23    vector<int> operator [] (int x){
24        return result[x];

```

```

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

```

6.11 Kuhn Munkres [e66c35]

```

1 // O(n^3) 找到最大權匹配
2 struct KuhnMunkres{
3     int n; // max(n, m)
4     vector<vector<int>> G;
5     vector<int> match, lx, ly, visx, visy;
6     vector<int> slack;
7     int stamp = 0;
8
9     KuhnMunkres(int n) : n(n), G(n, vector<int>(n)), lx(n),
10        ly(n), slack(n), match(n), visx(n), visy(n) {}
11
12     void add(int x, int y, int w){
13         G[x][y] = max(G[x][y], w);

```



```

14 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                     return true;
28                 }
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     for (int i=0 ; i<n ; i++){
65         lx[i] = 0;
66         for (int j=0 ; j<n ; j++){
67             lx[i] = max(lx[i], G[i][j]);
68         }
69     }
70
71     fill(ly.begin(), ly.end(), 0);
72     fill(match.begin(), match.end(), -1);
73
74     for(int i = 0; i < n; i++) {
75         fill(slack.begin(), slack.end(), INF);
76         stamp++;

```

```

79     if(dfs(i, true)) continue;
80
81     while(augment()==false) relabel();
82     stamp++;
83     dfs(i, true);
84 }
85
86 int ans = 0;
87 for (int j=0 ; j<n ; j++){
88     if (match[j]!=-1){
89         ans += G[match[j]][j];
90     }
91 }
92 return ans;
93 }
94 };

```

6.12 LCA [5b6a5b]

```

1 // 1-based · 可以支援森林 · 0 是超級源點 · 所有樹都要跟他建邊
2 struct Tree{
3     int N, M = 0, H;
4     vector<int> parent, dep;
5     vector<vector<int>> G, LCA;
6
7     Tree(int _N) : N(_N+1), H(__lg(N)+1), parent(N, -1), dep(
8         N), G(N){
9         LCA.resize(H, vector<int>(N, 0));
10    }
11
12    void add_edge(int u, int v){
13        M++;
14        G[u].push_back(v);
15        G[v].push_back(u);
16    }
17
18    void dfs(int now = 0, int pre = 0){
19        dep[now] = dep[pre]+1;
20        parent[now] = pre;
21        for (auto x : G[now]){
22            if (x==pre) continue;
23            dfs(x, now);
24        }
25    }
26
27    void build_LCA(int root = 0){
28        dfs();
29        for (int i=0 ; i<N ; i++) LCA[0][i] = parent[i];
30        for (int i=1 ; i<H ; i++){
31            for (int j=0 ; j<N ; j++){
32                LCA[i][j] = LCA[i-1][LCA[i-1][j]];
33            }
34        }
35    }
36
37    int jump(int u, int step){
38        for (int i=0 ; i<H ; i++){
39            if (step&(1<<i)) u = LCA[i][u];
40        }
41        return u;
42    }
43
44    int get_LCA(int u, int v){
45        if (dep[u]<dep[v]) swap(u, v);
46        u = jump(u, dep[u]-dep[v]);

```

```

46     if (u==v) return u;
47     for (int i=H-1 ; i>=0 ; i--){
48         if (LCA[i][u]!=LCA[i][v]){
49             u = LCA[i][u];
50             v = LCA[i][v];
51         }
52     }
53     return parent[u];
54 }
55 };

```

6.13 MCMF [0d524d]

```

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, G[u].size()});
13        G[u].push_back({v, 0, -k, G[v].size()-1});
14    }
15
16    // 6d1140
17    int spfa(int s, int t){
18        fill(par.begin(), par.end(), -1);
19        vector<int> dis(par.size(), INF);
20        vector<bool> in_q(par.size(), 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<G[v].size() ; 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>, d7e7ad
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 }
60 return {fl, cost};
61 }
62 };

```

6.14 Tarjan [8b2350]

```

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++;
50        }
51    }
52 };

```

6.15 Tarjan Find AP [1daed6]

```

1 vector<int> dep(MAX_N), low(MAX_N), AP;
2 bitset<MAX_N> vis;
3
4 void dfs(int now, int pre){
5     int cnt = 0;
6     bool ap = 0;
7     vis[now] = 1;
8     low[now] = dep[now] = (now==1 ? 0 : dep[pre]+1);
9
10    for (auto x : G[now]){
11        if (x==pre){
12            continue;
13        }else if (vis[x]==0){
14            cnt++;
15            dfs(x, now);
16            low[now] = min(low[now], low[x]);
17            if (low[x]>=dep[now]) ap=1;
18        }else{
19            low[now] = min(low[now], dep[x]);
20        }
21    }
22    if ((now==pre && cnt>=2) || (now!=pre && ap)){
23        AP.push_back(now);
24    }
25 }
26 }

```

6.16 Tree Isomorphism [cd2bbc]

```

1 #include <bits/stdc++.h>
2 #pragma GCC optimize("O3,unroll-loops")
3 #define fastio ios::sync_with_stdio(0), cin.tie(0), cout.tie(0)
4 #define dbg(x) cerr << #x << " = " << x << endl
5 #define int long long
6 using namespace std;
7
8 // declare
9 const int MAX_SIZE = 2e5+5;
10 const int INF = 9e18;
11 const int MOD = 1e9+7;
12 const double EPS = 1e-6;
13 typedef vector<vector<int>> Graph;
14 typedef map<vector<int>, int> Hash;
15
16 int n, a, b;
17 int id1, id2;
18 pair<int, int> c1, c2;
19 vector<int> sz1(MAX_SIZE), sz2(MAX_SIZE);
20 vector<int> we1(MAX_SIZE), we2(MAX_SIZE);
21 Graph g1(MAX_SIZE), g2(MAX_SIZE);
22 Hash m1, m2;
23 int testcase=0;
24
25 void centroid(Graph &g, vector<int> &s, vector<int> &w, pair<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     w[now]=max(w[now], n-s[now]);
36     if (w[now]<=n/2){
37         if (rec.first==0) rec.first=now;
38         else rec.second=now;
39     }
40 }
41
42 int dfs(Graph &g, Hash &m, int &id, int now, int pre){
43     vector<int> v;
44     for (auto x : g[now]){
45         if (x!=pre){
46             int add=dfs(g, m, id, x, now);
47             v.push_back(add);
48         }
49     }
50     sort(v.begin(), v.end());
51     if (m.find(v)!=m.end()){
52         return m[v];
53     }else{
54         m[v]=++id;
55         return id;
56     }
57 }
58
59 void solve1(){
60
61     // init
62     id1=0;
63     id2=0;
64     c1={0, 0};
65     c2={0, 0};
66     fill(sz1.begin(), sz1.begin()+n+1, 0);
67     fill(sz2.begin(), sz2.begin()+n+1, 0);
68     fill(we1.begin(), we1.begin()+n+1, 0);
69     fill(we2.begin(), we2.begin()+n+1, 0);
70     for (int i=1; i<=n; i++){
71         g1[i].clear();
72         g2[i].clear();
73     }
74     m1.clear();
75     m2.clear();
76
77     // input
78     cin >> n;
79     for (int i=0; i<=n-1; i++){
80         cin >> a >> b;
81         g1[a].push_back(b);
82         g1[b].push_back(a);
83     }
84     for (int i=0; i<=n-1; i++){
85         cin >> a >> b;
86         g2[a].push_back(b);
87         g2[b].push_back(a);
88     }
89
90     // get tree centroid
91     centroid(g1, sz1, we1, c1, 1, 0);
92     centroid(g2, sz2, we2, c2, 1, 0);
93
94     // process
95     int res1=0, res2=0, res3=0;
96
97 }

```

```

99  if (c2.second!=0){
100      res1=dfs(g1, m1, id1, c1.first, 0);
101      m2=m1;
102      id2=id1;
103      res2=dfs(g2, m1, id1, c2.first, 0);
104      res3=dfs(g2, m2, id2, c2.second, 0);
105  }else if (c1.second!=0){
106      res1=dfs(g2, m1, id1, c2.first, 0);
107      m2=m1;
108      id2=id1;
109      res2=dfs(g1, m1, id1, c1.first, 0);
110      res3=dfs(g1, m2, id2, c1.second, 0);
111  }else{
112      res1=dfs(g1, m1, id1, c1.first, 0);
113      res2=dfs(g2, m1, id1, c2.first, 0);
114  }
115
116  // output
117  cout << (res1==res2 || res1==res3 ? "YES" : "NO") << endl
118      ;
119
120  return;
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 圖方樹 [675acc]

```

1  #include <bits/stdc++.h>
2  #define lp(i,a,b) for(int i=a;i<(b);i++)
3  #define pii pair<int,int>
4  #define pb push_back
5  #define ins insert
6  #define ff first
7  #define ss second
8  #define opa(x) cerr << #x << " = " << x << ", ";
9  #define op(x) cerr << #x << " = " << x << endl;
10 #define ops(x) cerr << x;
11 #define etr cerr << endl;
12 #define spc cerr << ' ';
13 #define BAE(x) (x).begin(), (x).end()
14 #define STL(x) cerr << #x << " : "; for(auto &qwe:x) cerr <<
15   qwe << ' '; cerr << endl;
16 #define deb1 cerr << "deb1" << endl;
17 #define deb2 cerr << "deb2" << endl;
18 #define deb3 cerr << "deb3" << endl;
19 #define deb4 cerr << "deb4" << endl;
20 #define deb5 cerr << "deb5" << endl;
21 #define bye exit(0);
22 using namespace std;
23
24 const int mxn = (int)(2e5) + 10;
25 const int mxlg = 17;
26 int last_special_node = (int)(1e5) + 1;
27 vector<int> E[mxn], F[mxn];

```

```

28 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 << "--" <<
36   x.to;}
37 vector<edg> EV;
38
39 void tarjan(int v, int par, stack<int>& S){
40     static vector<int> dfn(mxn), low(mxn);
41     static vector<bool> to_add(mxn);
42     static int nowT = 0;
43
44     int childs = 0;
45     nowT += 1;
46     dfn[v] = low[v] = nowT;
47     for(auto &ne:E[v]){
48         int i = EV[ne].to;
49         if(i == par) continue;
50         if(!dfn[i]){
51             S.push(ne);
52             tarjan(i, v, S);
53             childs += 1;
54             low[v] = min(low[v], low[i]);
55
56             if(par >= 0 && low[i] >= dfn[v]){
57                 vector<int> bcc;
58                 int tmp;
59                 do{
60                     tmp = S.top(); S.pop();
61                     if(!to_add[EV[tmp].fr]){
62                         to_add[EV[tmp].fr] = true;
63                         bcc.pb(EV[tmp].fr);
64                     }
65                     if(!to_add[EV[tmp].to]){
66                         to_add[EV[tmp].to] = true;
67                         bcc.pb(EV[tmp].to);
68                     }
69                 }while(tmp != ne);
70                 for(auto &j:bcc){
71                     to_add[j] = false;
72                     F[last_special_node].pb(j);
73                     F[j].pb(last_special_node);
74                 }
75                 last_special_node += 1;
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
87 int dep[mxn], jmp[mxn][mxlg];
88 void dfs_lca(int v, int par, int depth){
89     dep[v] = depth;
90     for(auto &i:F[v]){
91         if(i == par) continue;
92         jmp[i][0] = v;

```

```

92         dfs_lca(i, v, depth + 1);
93     }
94 }
95
96 inline void build_lca(){
97     jmp[1][0] = 1;
98     dfs_lca(1, -1, 1);
99     lp(j,1,mxlg){
100         lp(i,1,mxn){
101             jmp[i][j] = jmp[jmp[i][j-1]][j-1];
102         }
103     }
104 }
105
106 inline int lca(int x, int y){
107     if(dep[x] < dep[y]){ swap(x, y); }
108
109     int diff = dep[x] - dep[y];
110     lp(j,0,mxlg){
111         if((diff >> j) & 1){
112             x = jmp[x][j];
113         }
114     }
115     if(x == y) return x;
116
117     for(int j = mxlg - 1; j >= 0; j--){
118         if(jmp[x][j] != jmp[y][j]){
119             x = jmp[x][j];
120             y = jmp[y][j];
121         }
122     }
123     return jmp[x][0];
124 }
125
126 inline bool can_reach(int fr, int to){
127     if(dep[to] > dep[fr]) return false;
128
129     int diff = dep[fr] - dep[to];
130     lp(j,0,mxlg){
131         if((diff >> j) & 1){
132             fr = jmp[fr][j];
133         }
134     }
135     return fr == to;
136 }
137
138 int main(){
139     ios::sync_with_stdio(false); cin.tie(0);
140     // freopen("test_input.txt", "r", stdin);
141     int n, m, q; cin >> n >> m >> q;
142     lp(i,0,m){
143         int u, v; cin >> u >> v;
144         E[u].pb(EV.size());
145         EV.pb(edg(u, v));
146         E[v].pb(EV.size());
147         EV.pb(edg(v, u));
148     }
149     E[0].pb(EV.size());
150     EV.pb(edg(0, 1));
151     stack<int> S;
152     tarjan(0, -1, S);
153     build_lca();
154
155     lp(queries,0,q){
156         int fr, to, relay; cin >> fr >> to >> relay;
157         if(fr == relay || to == relay){

```

```

158     cout << "NO\n";
159     continue;
160 }
161 if((can_reach(fr, relay) || can_reach(to, relay)) &&
162    dep[relay] >= dep[lca(fr, to)]){
163     cout << "NO\n";
164     continue;
165 }
166 cout << "YES\n";
167 }

```

6.18 最大權閉合圖 [6ca663]

```

1 // 邊  $u \rightarrow v$  表示選  $u$  就要選  $v$  ( $\theta$ -based)
2 // 保證回傳值非負
3 // 構造：從  $S$  開始 dfs · 不走最小割的邊 ·
4 // 所有經過的點就是要選的那些點 ·
5 // 一般圖： $O(n^2m)$  / 二分圖： $O(m/n)$ 
6 template<typename U>
7 U maximum_closure(vector<U> w, vector<pair<int,int>> EV) {
8     int n = w.size(), S = n + 1, T = n + 2;
9     Flow G(T + 5); // Graph/Dinic.cpp
10    U sum = 0;
11    for (int i = 0; i < n; ++i) {
12        if (w[i] > 0) {
13            G.add(S, i, w[i]);
14            sum += w[i];
15        }
16        else if (w[i] < 0) {
17            G.add(i, T, abs(w[i]));
18        }
19    }
20    for (auto &[u, v] : EV) { // 請務必確保  $INF > \sum w_i$ 
21        G.add(u, v, INF);
22    }
23    U cut = G.flow(S, T);
24    return sum - cut;
25 }

```

6.19 Theorem

- 任意圖
 - 最大匹配 + 最小邊覆蓋 = n (不能有孤點)
 - 點覆蓋的補集是獨立集。最小點覆蓋 + 最大獨立集 = n
 - w (最小權點覆蓋) + w (最大權獨立集) = $\sum w_v$
 - (帶點權的二分圖可以用最小割解 · 構造請參考 Augment Path.cpp)
- 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 [682ac6]

```

1 // 求出  $d = \gcd(a, b)$  · 並找出  $x, y$  使  $ax + by = d$ 
2 tuple<int, int, int> extgcd(int a, int b){
3     if (!b) return {a, 1, 0};
4     auto [d, x, y] = extgcd(b, a%b);
5     return {d, y, x-a/b*y};
6 }
7
8 // CRT maybe need use int128
9 int CRT_m_coprime(vector<int> &a, vector<int> &m) {
10    int n = a.size(), p = 1, ans = 0;
11    vector<int> M(n), invM(n);
12
13    for (int i=0; i<n; i++) p *= m[i];
14    for (int i=0; i<n; i++){
15        M[i] = p/m[i];
16        auto [d, x, y] = extgcd(M[i], m[i]);
17        invM[i] = x;
18        ans += a[i]*invM[i]*M[i];
19        ans %= p;
20    }
21    return (ans%p+p)%p;
22 }
23
24 // CRT maybe need use int128
25 int CRT_m_not_coprime(vector<int> &a, vector<int> &m) {
26    int n = a.size();
27
28    for (int i=1; i<n; i++){
29        int g = __gcd(m[0], m[i]);
30        if ((a[i]-a[0])%g!=0) return -1;
31
32        auto [d, x, y] = extgcd(m[0], m[i]);
33        x = (a[i]-a[0])*x/g;
34
35        a[0] = x*m[0]+a[0];
36        m[0] = m[0]*m[i]/g;
37        a[0] = (a[0]%m[0]+m[0])%m[0];
38    }
39
40    if (a[0]<0) return a[0]+m[0];
41    return a[0];
42 }
43
44 // ans = a / b (mod m)
45 // ans = ret.F + k * ret.S, k is integer
46 pair<int, int> div(int a, int b, int m) {
47    int flag = 1;
48    if (a < 0) { a = -a; flag *= -1; }
49    if (b < 0) { b = -b; flag *= -1; }
50    int t = -1, k = -1;
51    int res = extgcd_abc(b, m, a, t, k);
52    if (res == INF) return {INF, INF};
53    m = abs(m / res);
54    t = t * flag;
55    t = (t % m + m) % m;
56    return {t, m};
57 }

```

7.2 Josephus Problem [e0ed50]

```

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 }

```

7.3 Lagrange any x [1f2c26]

```

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 };

```

7.4 Lagrange continuous x [57536a]

```

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 // TO USE THIS TEMPLATE, YOU MUST MAKE SURE THAT THE MOD
18 // NUMBER IS A PRIME.
19 struct Lagrange {
20 /*
21  Initialize a polynomial with  $f(x_0)$ ,  $f(x_0 + 1)$ , ...,  $f(x_0 + n)$ .
22  This determines a polynomial  $f(x)$  whose degree is at most  $n$ .
23  Then you can call sample(x) and you get the value of  $f(x)$ .
24  Complexity of init() and sample() are both  $O(n)$ .
25 */
26 int m, shift; //  $m = n + 1$ 
27 vector<int> v, mul;
28 // You can use this function if you don't have inv_fac array already.
29 void construct_inv_fac() {
30     long long fac = 1;
31     for (int i = 2; i < MAX_N; ++i) {
32         fac = fac * i % mod;
33     }
34     inv_fac[MAX_N - 1] = fp(fac, mod - 2);
35     for (int i = MAX_N - 1; i >= 1; --i) {
36         inv_fac[i - 1] = inv_fac[i] * i % mod;
37     }
38 }
39 // You call init() many times without having a second instance of this struct.
40 void init(int X_0, vector<int> &u) {
41     v = u;
42     shift = ((1 - X_0) % mod + mod) % mod;
43     if (v.size() == 1) v.push_back(v[0]);
44     m = v.size();
45     mul.resize(m);
46 }
47 // You can use sample(x) instead of sample(x % mod).
48 int sample(int x) {
49     x = ((long long)x + shift) % mod;
50     x = (x < 0) ? (x + mod) : x;
51     long long now = 1;
52     for (int i = m; i >= 1; --i) {
53         mul[i - 1] = now;
54         now = now * (x - i) % mod;
55     }
56     int ret = 0;
57     bool neg = (m - 1) & 1;
58     now = 1;
59     for (int i = 1; i <= m; ++i) {
60         int up = now * mul[i - 1] % mod;
61         int down = inv_fac[m - i] * inv_fac[i - 1] % mod;
62         int tmp = ((long long)v[i - 1] * up % mod) * down % mod;
63         ret += (neg && tmp) ? (mod - tmp) : (tmp);
64         ret = (ret >= mod) ? (ret - mod) : ret;
65         now = now * (x - i) % mod;
66         neg ^= 1;
67     }
68     return ret;
69 }
70 };
71
72 int main() {
73     int n; cin >> n;
74     vector<int> v(n);
75     for (int i = 0; i < n; ++i) {

```

```

76     cin >> v[i];
77 }
78 Lagrange L;
79 L.construct_inv_fac();
80 L.init(0, v);
81 int x; cin >> x;
82 cout << L.sample(x);
83 }

```

7.5 Linear Mod Inverse [ecf71e]

```

1 // 線性求 1-based  $a[i]$  對  $p$  的乘法反元素
2 vector<int> s(n+1, 1), invS(n+1), invA(n+1);
3 for (int i=1; i<=n; i++) s[i] = s[i-1]*a[i]%p;
4 invS[n] = qp(s[n], p-2, p);
5 for (int i=n; i>=1; i--) invS[i-1] = invS[i]*a[i]%p;
6 for (int i=1; i<=n; i++) invA[i] = invS[i]*s[i-1]%p;

```

7.6 Lucas's Theorem [b37dcf]

```

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 }

```

7.7 Matrix [8d1a23]

```

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

```

```

36         mul = mul*mul;
37     }
38
39     return ret;
40 }
41
42
43 int det(){
44     vector<vector<int>> arr = this->arr;
45     bool flag = false;
46     for (int i=0; i<n; i++){
47         int target = -1;
48         for (int j=i; j<n; j++){
49             if (arr[j][i]){
50                 target = j;
51                 break;
52             }
53         }
54         if (target==-1) return 0;
55         if (i!=target){
56             swap(arr[i], arr[target]);
57             flag = !flag;
58         }
59
60         for (int j=i+1; j<n; j++){
61             if (!arr[j][i]) continue;
62             int freq = arr[j][i]*qp(arr[i][i], MOD-2)%MOD;
63
64             for (int k=i; k<n; k++){
65                 arr[j][k] -= freq*arr[i][k];
66                 arr[j][k] = (arr[j][k]%MOD+MOD)%MOD;
67             }
68         }
69
70         int ret = !flag ? 1 : MOD-1;
71         for (int i=0; i<n; i++){
72             ret *= arr[i][i];
73             ret %= MOD;
74         }
75         return ret;
76     }
77 };

```

7.8 Matrix 01 [8d542a]

```

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 };

```

7.9 Miller Rabin [24bd0d]

```

1 // O(k Log^3 n), k = llsprp.size()
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) ret = modmul(ret, b, m);
12        b = modmul(b, b, m);
13    }
14    return ret;
15 }
16
17 // ed23aa
18 vector<int> llsprp = {2, 325, 9375, 28178, 450775, 9780504,
19     1795265022};
20 bool is_prime(int n, vector<int> sprp = llsprp){
21     if (n==2) return 1;
22     if (n<2 || n%2==0) return 0;
23
24     int t = 0;
25     int u = n-1;
26     for ( ; u%2==0 ; t++) u>>=1;
27
28     for (int i=0 ; i<sprp.size() ; i++){
29         int a = sprp[i]%n;
30         if (a==0 || a==1 || a==n-1) continue;
31         int x = qp(a, u, n);
32         if (x==1 || x==n-1) continue;
33         for (int j=0 ; j<t ; j++){
34             x = modmul(x, x, n);
35             if (x==1) return 0;
36             if (x==n-1) break;
37         }
38         if (x==n-1) continue;
39         return false;
40     }
41
42     return true;
43 }

```

7.10 Pollard Rho [a5daef]

```

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 }

```

```

5 // O(n^{1/4}) 回傳 1 或自己的因數、記得先判斷 n 是不是質數
6 // (用 Miller-Rabin)
7 // c1670c
8 int Pollard_Rho(int n){
9     int s = 0, t = 0;
10    int c = rnd(1, n-1);
11
12    int step = 0, goal = 1;
13    int val = 1;
14
15    for (goal=1 ; ; goal<=1, s=t, val=1){
16        for (step=1 ; step<=goal ; step++){
17
18            t = ((__int128)t*t+c)%n;
19            val = ((__int128)val*abs(t-s)%n;
20
21            if ((step % 127) == 0){
22                int d = __gcd(val, n);
23                if (d>1) return d;
24            }
25        }
26
27        int d = __gcd(val, n);
28        if (d>1) return d;
29    }
30 }

```

7.11 Polynomial [51ca3b]

```

1 struct Poly {
2     int len, deg;
3     int *a;
4     // Len = 2^k >= the original length
5     Poly(): len(0), deg(0), a(nullptr) {}
6     Poly(int _n) {
7         len = 1;
8         deg = _n - 1;
9         while (len < _n) len <= 1;
10        a = (ll*) calloc(len, sizeof(ll));
11    }
12    Poly(int l, int d, int *b) {
13        len = l;
14        deg = d;
15        a = b;
16    }
17    void resize(int _n) {
18        int len1 = 1;
19        while (len1 < _n) len1 <= 1;
20        int *res = (ll*) calloc(len1, sizeof(ll));
21        for (int i = 0; i < min(len, _n); i++) {
22            res[i] = a[i];
23        }
24        len = len1;
25        deg = _n - 1;
26        free(a);
27        a = res;
28    }
29    Poly& operator=(const Poly rhs) {
30        this->len = rhs.len;
31        this->deg = rhs.deg;
32        this->a = (ll*)realloc(this->a, sizeof(ll) * len);
33        copy(rhs.a, rhs.a + len, this->a);
34        return *this;
35    }

```

```

36 Poly operator*(Poly rhs) {
37     int l1 = this->len, l2 = rhs.len;
38     int d1 = this->deg, d2 = rhs.deg;
39     while (l1 > 0 and this->a[l1 - 1] == 0) l1--;
40     while (l2 > 0 and rhs.a[l2 - 1] == 0) l2--;
41     int l = 1;
42     while (l < max(l1 + l2 - 1, d1 + d2 + 1)) l <= 1;
43     int *x, *y, *res;
44     x = (ll*) calloc(l, sizeof(ll));
45     y = (ll*) calloc(l, sizeof(ll));
46     res = (ll*) calloc(l, sizeof(ll));
47     copy(this->a, this->a + l1, x);
48     copy(rhs.a, rhs.a + l2, y);
49     ntt.tran(l, x); ntt.tran(l, y);
50     FOR (i, 0, l - 1)
51         res[i] = x[i] * y[i] % mod;
52     ntt.tran(l, res, true);
53     free(x); free(y);
54     return Poly(l, d1 + d2, res);
55 }
56 Poly operator+(Poly rhs) {
57     int l1 = this->len, l2 = rhs.len;
58     int l = max(l1, l2);
59     Poly res;
60     res.len = l;
61     res.deg = max(this->deg, rhs.deg);
62     res.a = (ll*) calloc(l, sizeof(ll));
63     FOR (i, 0, l1 - 1) {
64         res.a[i] += this->a[i];
65         if (res.a[i] >= mod) res.a[i] -= mod;
66     }
67     FOR (i, 0, l2 - 1) {
68         res.a[i] += rhs.a[i];
69         if (res.a[i] >= mod) res.a[i] -= mod;
70     }
71     return res;
72 }
73 Poly operator-(Poly rhs) {
74     int l1 = this->len, l2 = rhs.len;
75     int l = max(l1, l2);
76     Poly res;
77     res.len = l;
78     res.deg = max(this->deg, rhs.deg);
79     res.a = (ll*) calloc(l, sizeof(ll));
80     FOR (i, 0, l1 - 1) {
81         res.a[i] += this->a[i];
82         if (res.a[i] >= mod) res.a[i] -= mod;
83     }
84     FOR (i, 0, l2 - 1) {
85         res.a[i] -= rhs.a[i];
86         if (res.a[i] < 0) res.a[i] += mod;
87     }
88     return res;
89 }
90 Poly operator*(const int rhs) {
91     Poly res;
92     res = *this;
93     FOR (i, 0, res.len - 1) {
94         res.a[i] = res.a[i] * rhs % mod;
95         if (res.a[i] < 0) res.a[i] += mod;
96     }
97     return res;
98 }
99 Poly(vector<int> f) {
100    int _n = f.size();
101    len = 1;

```



```

102     deg = _n - 1;
103     while (len < _n) len <= 1;
104     a = (ll*) calloc(len, sizeof(ll));
105     FOR (i, 0, deg) a[i] = f[i];
106 }
107 Poly derivative() {
108     Poly g(this->deg);
109     FOR (i, 1, this->deg) {
110         g.a[i - 1] = this->a[i] * i % mod;
111     }
112     return g;
113 }
114 Poly integral() {
115     Poly g(this->deg + 2);
116     FOR (i, 0, this->deg) {
117         g.a[i + 1] = this->a[i] * ::inv(i + 1) % mod;
118     }
119     return g;
120 }
121 Poly inv(int len1 = -1) {
122     if (len1 == -1) len1 = this->len;
123     Poly g(1); g.a[0] = ::inv(a[0]);
124     for (int l = 1; l < len1; l <= 1) {
125         Poly t; t = *this;
126         t.resize(l < 1);
127         t = g * g * t;
128         t.resize(l < 1);
129         Poly g1 = g * 2 - t;
130         swap(g, g1);
131     }
132     return g;
133 }
134 Poly ln(int len1 = -1) {
135     if (len1 == -1) len1 = this->len;
136     auto g = *this;
137     auto x = g.derivative() * g.inv(len1);
138     x.resize(len1);
139     x = x.integral();
140     x.resize(len1);
141     return x;
142 }
143 Poly exp() {
144     Poly g(1);
145     g.a[0] = 1;
146     for (int l = 1; l < len; l <= 1) {
147         Poly t, g1; t = *this;
148         t.resize(l < 1); t.a[0]++;
149         g1 = (t - g.ln(l < 1)) * g;
150         g1.resize(l < 1);
151         swap(g, g1);
152     }
153     return g;
154 }
155 Poly pow(ll n) {
156     Poly &a = *this;
157     int i = 0;
158     while (i <= a.deg and a.a[i] == 0) i++;
159     if (i and (n > a.deg or n * i > a.deg)) return Poly(a
        .deg + 1);
160     if (i == a.deg + 1) {
161         Poly res(a.deg + 1);
162         res.a[0] = 1;
163         return res;
164     }
165     Poly b(a.deg - i + 1);
166     int inv1 = ::inv(a.a[i]);

```

```

167     FOR (j, 0, b.deg)
168         b.a[j] = a.a[j + i] * inv1 % mod;
169     Poly res1 = (b.ln() * (n % mod)).exp() * (::power(a.a
170         [i], n));
171     Poly res2(a.deg + 1);
172     FOR (j, 0, min((ll)(res1.deg), (ll)(a.deg - n * i)))
173         res2.a[j + n * i] = res1.a[j];
174     return res2;
175 }
176 }
177 };

```

7.12 josephus [0be067]

```

1 // n 個人，每 k 個人就刪除的約瑟夫遊戲
2 int josephus(int n, int k) {
3     if (n == 1)
4         return 0;
5     if (k == 1)
6         return n-1;
7     if (k > n)
8         return (josephus(n-1, k) + k) % n;
9     int cnt = n / k;
10    int res = josephus(n - cnt, k);
11    res -= n % k;
12    if (res < 0)
13        res += n;
14    else
15        res += res / (k - 1);
16    return res;
17 }

```

7.13 數論分塊 [8ccab5]

```

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

```

7.14 最大質因數 [ca5e52]

```

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 [85f3b1]

```

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    return ret;
14 }
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 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

7.17 Catalan Number

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

7.18 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。

7.19 Stirling's formula

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

7.20 Theorem

- $1 \sim x$ 質數的數量 $\approx \frac{x}{\ln x}$
- 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
- $n^{k-2} \cdot \prod_{i=1}^k s_i$ n 個點、 k 的連通塊，加上 $k-1$ 條邊使得變成一個連通圖的方法數，其中每個連通塊有 s_i 個點

7.21 二元一次方程式

$$\begin{cases} ax + by = e \\ cx + dy = f \end{cases} = \begin{cases} x = \frac{ed-bf}{ad-bc} \\ y = \frac{af-ec}{ad-bc} \end{cases}$$

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。

7.22 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：

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

7.23 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 AC automation [018290]

```
1 struct ACAutomation{
2     vector<vector<int>> go;
3     vector<int> fail, match, pos;
4     int sz = 0; // 有效節點為 [0, sz]，開陣列的時候要小心!!!
5
6     ACAutomation(int n) : go(n, vector<int>(26)), fail(n), match(n) {}
7
8     void add(string s){
9         int now = 0;
10        for (char c : s){
11            if (!go[now][c-'a']) go[now][c-'a'] = ++sz;
12            now = go[now][c-'a'];
13        }
14        pos.push_back(now);
15    }
16 }
```

```
17 void build(){
18     queue<int> que;
19     for (int i=0 ; i<26 ; i++){
20         if (go[0][i]) que.push(go[0][i]);
21     }
22     while (que.size()){
23         int u = que.front();
24         que.pop();
25         for (int i=0 ; i<26 ; i++){
26             if (go[u][i]){
27                 fail[go[u][i]] = go[fail[u]][i];
28                 que.push(go[u][i]);
29             }else go[u][i] = go[fail[u]][i];
30         }
31     }
32 }
33
34 // counting pattern
35 void buildMatch(string &s){
36     int now = 0;
37     for (char c : s){
38         now = go[now][c-'a'];
39         match[now]++;
40     }
41
42     vector<int> in(sz+1), que;
43     for (int i=1 ; i<=sz ; i++) in[fail[i]]++;
44     for (int i=1 ; i<=sz ; i++) if (in[i]==0) que.push_back(i);
45     for (int i=0 ; i<que.size() ; i++){
46         int now = que[i];
47         match[fail[now]] += match[now];
48         if (--in[fail[now]]==0) que.push_back(fail[now]);
49     }
50 }
51 };
```

8.2 Enumerate Runs [94ca46]

```
1 vector<array<int, 3>> enumerate_run(string s){
2
3
4     int n = s.size();
5     SuffixArray sa(s), saBar(string(s.rbegin(), s.rend()));
6     sa.init_lcp(), saBar.init_lcp();
7
8     set<pair<int, int>> ss;
9     vector<array<int, 3>> runs;
10
11     for (int len=1 ; len<=n ; len++){
12         vector<int> lcp;
13         for (int i=0 ; i+len<n ; i+=len){
14             int pos1 = sa.pos[i];
15             int pos2 = sa.pos[i+len];
16             lcp.push_back(sa.get_lcp(pos1, pos2));
17         }
18
19         for (int ll=0, rr=0 ; ll<lcp.size() ; rr++, ll=rr){
20             while (rr<lcp.size() && lcp[rr]>=len) rr++;
21
22             int preLen = 0;
23             if (ll!=0){
24                 int p = n-1;
25                 int pos1 = saBar.pos[p-(ll*len-1)];
26                 int pos2 = saBar.pos[p-((ll+1)*len-1)];
```

```
27         preLen = saBar.get_lcp(pos1, pos2);
28     }
29     int sufLen = rr<lcp.size() ? lcp[rr] : 0;
30
31     int ansL = ll*len-preLen, ansR = (rr+1)*len-1+sufLen;
32     if (ansL!=ansR && ansR-ansL+1>=2*len && ss.find({ansL, ansR+1})==ss.end()){
33         ss.insert({ansL, ansR+1});
34         runs.push_back({len, ansL, ansR+1});
35     }
36 }
37 }
38
39 return runs;
40 }
```

8.3 Hash [942f42]

```
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 };
```

8.4 KMP [7b95d6]

```
1 // KMP[i] = s[0...i] 的最長共同前後綴長度，KMP[KMP[i]-1] 可以跳 fail link
2 // e5b7ce
3 vector<int> KMP(string &s){
4     vector<int> ret(n);
5     for (int i=1 ; i<s.size() ; i++){
6         int j = ret[i-1];
```

```

7   while (j && s[i]!=s[j]) j = ret[j-1];
8   ret[i] = j + (s[i]==s[j]);
9 }
10 return ret;
11 }

```

8.5 Manacher [9a4b4d]

```

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 }

```

8.6 Min Rotation [b24786]

```

1 int minRotation(string s) {
2     int a = 0, n = s.size();
3     s += s;
4
5     for (int b=0 ; b<n ; b++){
6         for (int k=0 ; k<n ; k++){
7             if (a+k==b || s[a+k]<s[b+k]){
8                 b += max(0LL, k-1);
9                 break;
10            }
11            if (s[a+k]>s[b+k]){
12                a = b;
13                break;
14            }
15        }
16    }
17    return a;
18 }

```

8.7 Suffix Array [f66629]

```

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     // lim 要調整成字元集大小·_s 不可以有 0
8     SuffixArray(string _s, int lim = 256) {
9         s = _s;
10        int n = s.size()+1, k = 0, a, b;
11        vector<int> x(s.begin(), s.end()), y(n), ws(max(n,
12            lim)), rank(n);
13        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        for (int i=1 ; i<n ; i++) rank[sa[i]] = i;
33        for (int i=0, j ; i<n-1 ; lcp[rank[i++]] = k)
34            for (k && k--, j=sa[rank[i]-1] ; i+k<s.size() &&
35                j+k<s.size() && s[i+k]==s[j+k] ; k++);
36        sa.erase(sa.begin());
37        lcp.erase(lcp.begin(), lcp.begin()+2);
38    }
39
40    // f49583
41    vector<int> pos; // pos[i] = i 這個值在 pos 的哪個地方
42    SparseTable st;
43    void init_lcp(){
44        pos.resize(sa.size());
45        for (int i=0 ; i<sa.size() ; i++){
46            pos[sa[i]] = i;
47        }
48        if (lcp.size()){
49            st.build(lcp);
50        }
51
52    // 用之前記得 init
53    // 查詢「sa 上的位置」的 x 跟 y 的 lcp
54    int get_lcp(int x, int y){
55        if (x==y) return s.size()-x;
56        if (x>y) swap(x, y);
57        return st.query(x, y);
58    }
59
60    // 回傳 [l1, r1] 跟 [l2, r2] 的 lcp·0-based
61    int get_lcp(int l1, int r1, int l2, int r2){
62        int pos_1 = pos[l1], len_1 = r1-l1+1;
63        int pos_2 = pos[l2], len_2 = r2-l2+1;
64        if (pos_1>pos_2){
65            swap(pos_1, pos_2);
66            swap(len_1, len_2);
67        }
68
69        if (l1==l2){
70            return min(len_1, len_2);
71        }else{
72            return min({st.query(pos_1, pos_2), len_1, len_2
73                });
74        }
75    }

```

```

75    // 檢查 [l1, r1] 跟 [l2, r2] 的大小關係·0-based
76    // 如果前者小於後者·就回傳 <0·相等就回傳 =0·否則回傳
77    >0
78    // 5b8db0
79    int substring_cmp(int l1, int r1, int l2, int r2){
80        int len_1 = r1-l1+1;
81        int len_2 = r2-l2+1;
82        int res = get_lcp(l1, r1, l2, r2);
83
84        if (res<len_1 && res<len_2){
85            return s[l1+res]-s[l2+res];
86        }else if (len_1==res && len_2==res){
87            return 0;
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        return {pre, suf};
128    }

```

8.8 Wildcard Matching [e93074]

```

1 const int B = 27;
2 string p;
3 for (int i=0 ; i<B ; i++) p += ('a'+i);

```

```

5 | p += '*';
6 | vector<vector<double>> res(B);
7 | for (int i=0 ; i<B ; i++){
8 |     vector<double> ss, tt;
9 |     for (auto x : s) ss.push_back(x==p[i] || x=='*');
10 |    for (auto x : t) tt.push_back(x==p[i] || x=='*');
11 |    reverse(tt.begin(), tt.end());
12 |    res[i] = PolyMul(ss, tt);
13 | }
14 | for (int i=t.size()-1 ; i+t.size()-1<res[0].size() ; i++){
15 |     int total = 0;
16 |     for (int j=0 ; j<B-1 ; j++) total += (int)abs(round(res[j
17 |         ][i]));
18 |     total -= 25*(int)abs(round(res[26][i]));
19 |     cout << (total==t.size());

```

8.9 Z Algorithm [9d559a]

```

1 | // z[i] 回傳 s[0...] 跟 s[i...] 的 lcp, z[0] = 0
2 | vector<int> z_function(string s){
3 |     vector<int> z(s.size());
4 |     int l = -1, r = -1;
5 |     for (int i=1 ; i<s.size() ; i++){
6 |         z[i] = i>=r ? 0 : min(r-i, z[i-l]);
7 |         while (i+z[i]<s.size() && s[i+z[i]]==s[z[i]]) z[i]++;
8 |         if (i+z[i]>r) l=i, r=i+z[i];
9 |     }
10 |    return z;
11 | }

```

8.10 k-th Substring [61f66b]

```

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

```