

Contents

1	字串	1
1.1	最長迴文子字串	1
1.2	Manacher	1
1.3	KMP	1
1.4	Z Algorithm	1
1.5	Suffix Array	1
2	math	2
2.1	公式	2
2.2	Rational	2
2.3	乘法逆元、組合數	2
2.4	歐拉函數	3
2.5	質數與因數	3
2.6	高斯消去	3
2.7	Extended GCD	3
2.8	大步小步	4
2.9	Pisano Period	4
2.10	矩陣快速幂	4
3	algorithm	5
3.1	greedy	5
3.2	JosephusProblem	6
3.3	二分搜	6
3.4	三分搜	6
3.5	dinic	6
3.6	SCC Tarjan	6
3.7	BCC 邊	7
3.8	BCC 點	7
3.9	ArticulationPoints Tarjan	7
3.10	最小樹狀圖	8
3.11	KM	8
3.12	二分圖最大匹配	8
3.13	差分	9
3.14	MCMF	9
3.15	Dancing Links	9
3.16	LCA 倍增法	10
3.17	LCA 樹鍊剖分	10
4	DataStructure	11
4.1	帶權併查集	11
4.2	Trie	11
4.3	AC Trie	11
4.4	線段樹 1D	11
4.5	線段樹 2D	12
4.6	權值線段樹	12
4.7	單調隊列	13
5	Geometry	13
5.1	公式	13
5.2	Template	13
5.3	旋轉卡尺	14
5.4	半平面相交	14
5.5	Polygon	14
5.6	凸包	14
5.7	最小圓覆蓋	14
5.8	交點、距離	14
6	DP	15
6.1	背包	15
6.2	Deque 最大差距	16
6.3	string DP	16
6.4	LCS 和 LIS	16
6.5	樹 DP 有幾個 path 長度為 k	16
6.6	WeightedLIS	16

1 字串

1.1 最長迴文子字串

```

1 #include<bits/stdc++.h>
2 #define T(x) ((x)%2 ? s[(x)/2] : '.')
3 using namespace std;
4
5 string s;
6 int n;
7
8 int ex(int l,int r){
9     int i=l;
10    while(l-i>=0&&r+i<n&&T(l-i)==T(r+i)) i++;
11    return i;
12 }
13
14 int main(){
15     cin>>s;
16     n=2*s.size()+1;
17     int mx=0;
18     int center=0;
19     vector<int> r(n);
20     int ans=1;
21     r[0]=1;
22     for(int i=1;i<n;i++){
23         int ii=center-(i-center);
24         int len=mx-i+1;
25         if(i>mx){
26             r[i]=ex(i,i);
27             center=i;
28             mx=i+r[i]-1;
29         }
30         else if(r[ii]==len){
31             r[i]=len+ex(i-len,i+len);
32             center=i;
33             mx=i+r[i]-1;
34         }
35         else r[i]=min(r[ii],len);
36         ans=max(ans,r[i]);
37     }
38     cout<<ans-1<<"\n";
39     return 0;
40 }

```

1.2 Manacher

s: 增長為兩倍的字串, 以 '@' 為首, 以 '\$' 為間隔, 以 '\0' 節尾

p: 以 s[i] 為中心, 半徑為 p[i] 是迴文

return: 最長的迴文長度

```

1 const int maxn = 1e5 + 10;
2
3 char s[maxn<<1] = "@$";
4 int p[maxn<<1];
5
6 int manacher(char* str, int n) {
7     for(int i=1; i<=n; i++) {
8         s[i<<1] = str[i-1];
9         s[i<<1|1] = '$';
10    }
11
12    int cur = 0, r = 0, res = 0;
13    s[n] = (n+1) << 1;
14    for(int i=1; i<n; i++) {
15        p[i] = (i>r) ? 1 : min(p[cur*2-i], r-i);
16        for(; s[i-p[i]]==s[i+p[i]]; p[i]++);
17        if(i+p[i] > r) {
18            r = i + p[i];
19            cur = i;
20        }
21        res = max(res, p[i]);
22    }
23    return res - 1;
24 }

```

1.3 KMP

```

1 const int maxn = 1e6 + 10;
2
3 int n, m; // len(a), len(b)
4 int f[maxn]; // failure function
5 char a[maxn], b[maxn];
6
7 void failureFuntion() { // f[0] = 0
8     for(int i=1, j=0; i<m; ) {
9         if(b[i] == b[j]) f[i++] = ++j;
10        else if(j) j = f[j-1];
11        else f[i++] = 0;
12    }
13 }
14
15 int kmp() {
16     int i = 0, j = 0, res = 0;
17     while(i < n) {
18         if(a[i] == b[j]) i++, j++;
19         else if(j) j = f[j-1];
20         else i++;
21         if(j == m) {
22             res++; // 找到答案
23             j = 0; // non-overlapping
24         }
25     }
26     return res;
27 }
28
29 // Problem: 所有在b裡, 前後綴相同的長度
30 // b = ababcababababababab
31 // f = 001201234123456789
32 // 前9 = 後9
33 // 前4 = 前9的後4 = 後4
34 // 前2 = 前4的後2 = 前9的後2 = 後2
35 for(int j=m; j; j=f[j-1]) {
36     // j 是答案
37 }

```

1.4 Z Algorithm

```

1 const int maxn = 1e6 + 10;
2
3 int z[maxn]; // s[0:z[i]] = s[i:z[i]]
4 string s;
5
6 void makeZ() { // z[0] = 0
7     for(int i=1, l=0, r=0; i<s.length(); i++) {
8         if(i<=r && z[i-l]<r-i+1) z[i] = z[i-l];
9         else {
10            z[i] = max(0, r-i+1);
11            while(i+z[i]<s.length() &&
12                s[z[i]]==s[i+z[i]]) z[i]++;
13        }
14        if(i+z[i]-1 > r) l = i, r = i+z[i]-1;
15    }
16 }

```

1.5 Suffix Array

- $O(n \log(n))$
- SA: 後綴數組
- HA: 相鄰後綴的共同前綴長度 (Longest Common Prefix)
- maxc: 可用字元的最大 ASCII 值
- maxn >= maxc
- 記得先取 n 的值 (strlen(s))

```
1 const int maxn = 2e5 + 10;
2 const int maxc = 256 + 10;
3
4 int n;
5 int SA[maxn], HA[maxn];
6 int rk[maxn], cnt[maxn], tmp[maxn];
7 char s[maxn];
8
9 void getSA() {
10     int mx = maxc;
11     for(int i=0; i<mx; cnt[i++]=0);
12
13     // 第一次 stable counting sort, 編 rank 和 sa
14     for(int i=0; i<n; i++) cnt[rk[i]=s[i]]++;
15     for(int i=1; i<mx; i++) cnt[i] += cnt[i-1];
16     for(int i=n-1; i>=0; i--) SA[--cnt[s[i]]]=i;
17
18     // 倍增法運算
19     for(int k=1, r=0; k<n; k<=<=1, r=0) {
20         for(int i=0; i<mx; cnt[i++]=0);
21         for(int i=0; i<n; i++) cnt[rk[i]]++;
22         for(int i=1; i<mx; i++) cnt[i] += cnt[i-1];
23         for(int i=n-k; i<n; i++) tmp[r++] = i;
24         for(int i=0; i<n; i++) {
25             if(SA[i] >= k) tmp[r++] = SA[i] - k;
26         }
27
28         // 計算本回 SA
29         for(int i=n-1; i>=0; i--) {
30             SA[--cnt[rk[tmp[i]]]] = tmp[i];
31         }
32
33         // 計算本回 rank
34         tmp[SA[0]] = r = 0;
35         for(int i=1; i<n; i++) {
36             if((SA[i-1]+k >= n) ||
37                 (rk[SA[i-1]] != rk[SA[i]]) ||
38                 (rk[SA[i-1]+k] != rk[SA[i]+k])) r++;
39             tmp[SA[i]] = r;
40         }
41         for(int i=0; i<n; i++) rk[i] = tmp[i];
42         if((mx=r+1) == n) break;
43     }
44 }
45
46 void getHA() { // HA[0] = 0
47     for(int i=0; i<n; i++) rk[SA[i]] = i;
48     for(int i=0, k=0; i<n; i++) {
49         if(!rk[i]) continue;
50         if(k) k--;
51         while(s[i+k] == s[SA[rk[i]-1]+k]) k++;
52         HA[rk[i]] = k;
53     }
54 }
```

2 math

2.1 公式

1. Most Divisor Number

Range	最多因數數	因數個數
10^9	735134400	1344
2^{31}	2095133040	1600
10^{18}	897612484786617600	103680
2^{64}	9200527969062830400	161280

2. Catlan Number

$$C_n = \frac{1}{n} \binom{2n}{n}, C_{n+1} = \frac{2(2n+1)}{n+2} C_n$$
$$C = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, \dots$$

3. Lagrange Polynomial

拉格朗日插值法：找出 n 次多項函數 $f(x)$ 的點
 $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$

$$L(x) = \sum_{j=0}^n y_j l_j(x)$$

$$l_j(x) = \prod_{i=0, i \neq j}^n \frac{x - x_i}{x_j - x_i}$$

4. Fibonacci

$$\begin{bmatrix} f_{n-1} & f_n \\ f_n & f_{n+1} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} f_n & f_{n+1} \\ f_{n+p} & f_{n+p+1} \end{bmatrix}, p \in \mathbb{N}$$

$$F_n = \frac{1}{\sqrt{5}} \left[\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right]$$

5. Pick's Theorem

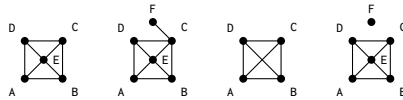
給定頂點座標均是整點（或正方形格子點）的簡單多邊形，
其面積 A 和內部格點數目 i 、邊上格點數目 b 的關係為

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

6. Euler's Formula

對於有 V 個點、 E 條邊、 F 個面（含外部）的連通平面圖

$$F + V - E = 2$$



(1) × (2) ○ (3) × (4) ×, 非連通圖

7. Simpson Integral

$$\int_a^b f(x) dx \approx \frac{b-a}{6} \left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b) \right]$$

2.2 Rational

```
1 const char sep = '/'; // 分數的分隔符
2 bool div0; // 要記得適時歸零
3 using ll = long long;
4
5 struct Rational {
6     ll p, q;
7
8     Rational(ll a=0, ll b=1) {
9         p = a, q = b;
10        reduce();
11    }
12
13    Rational(string s) {
14        if(s.find(sep) == string::npos) {
15            p = stoll(s);
16            q = 1;
17        } else {
18            p = stoll(s.substr(0, s.find(sep)));
19            q = stoll(s.substr(s.find(sep)+1));
20        }
21        reduce();
22    }
23
24    void reduce() {
25        ll t = abs(__gcd(p, q));
26        if(t == 0) {
27            div0 = true;
28            return;
29        }
30        p /= t, q /= t;
31        if(q < 0) p = -p, q = -q;
32        return;
33    }
34
35    string toString() {
36        if(q == 0) {
37            div0 = true;
```

```
38        return "INVALID";
39    }
40    if(p%q == 0) return to_string(p/q);
41    return to_string(p) + sep + to_string(q);
42 }
43
44 friend istream& operator>>(
45     istream& i, Rational& r) {
46     string s;
47     i >> s;
48     r = Rational(s);
49     return i;
50 }
51
52 friend ostream& operator<<(
53     ostream& o, Rational r) {
54     o << r.toString();
55     return o;
56 }
57 };
58
59 Rational operator+(Rational x, Rational y) {
60     ll t = abs(__gcd(x.q, y.q));
61     if(t == 0) return Rational(0, 0);
62     return Rational(
63         y.q/t*x.p + x.q/t*y.p, x.q/t*y.q);
64 }
65
66 Rational operator-(Rational x, Rational y) {
67     return x + Rational(-y.p, y.q);
68 }
69
70 Rational operator*(Rational x, Rational y) {
71     return Rational(x.p*y.p, x.q*y.q);
72 }
73
74 Rational operator/(Rational x, Rational y) {
75     return x * Rational(y.q, y.p);
76 }
```

2.3 乘法逆元、組合數

$$x^{-1} \bmod m = \begin{cases} 1, & \text{if } x = 1 \\ -\lfloor \frac{m}{x} \rfloor (m \bmod x)^{-1}, & \text{otherwise} \end{cases} \quad (m \bmod m)$$
$$= \begin{cases} 1, & \text{if } x = 1 \\ (m - \lfloor \frac{m}{x} \rfloor)(m \bmod x)^{-1}, & \text{otherwise} \end{cases} \quad (m \bmod m)$$

若 $p \in \text{prime}$, 根據費馬小定理, 則

$$\begin{aligned} \therefore ax &\equiv 1 \pmod{p} \\ \therefore ax &\equiv a^{p-1} \pmod{p} \\ \therefore x &\equiv a^{p-2} \pmod{p} \end{aligned}$$

```
1 using ll = long long;
2 const int maxn = 2e5 + 10;
3 const int mod = 1e9 + 7;
4
5 int fact[maxn] = {1, 1}; // x! % mod
6 int inv[maxn] = {1, 1}; // x^(-1) % mod
7 int invFact[maxn] = {1, 1}; // (x!)^(-1) % mod
8
9 void build() {
10     for(int x=2; x<maxn; x++) {
11         fact[x] = (ll)x * fact[x-1] % mod;
12         inv[x] = (ll)(mod-mod/x)*inv[mod%x]%mod;
13         invFact[x] = (ll)invFact[x-1]*inv[x]%mod;
14     }
15 }
16
17 // 前提: mod 為質數
18 void build() {
19     auto qpow = [&](ll a, int b) {
20         ll res = 1;
21         for(; b; b>>=1) {
22             if(b & 1) res = res * a % mod;
23             a = a * a % mod;
24         }
25         return res;
26     };
```

```

27
28     for(int x=2; x<maxn; x++) {
29         fact[x] = (1ll)x * fact[x-1] % mod;
30         invFact[x] = qpow(fact[x], mod-2);
31     }
32 }
33
34 // C(a, b) % mod
35 int comb(int a, int b) {
36     if(a < b) return 0;
37     ll x = fact[a];
38     ll y = (1ll)invFact[b] * invFact[a-b] % mod;
39     return x * y % mod;
40 }

```

2.4 歐拉函數

```

1 //計算閉區間 [1,n] 中有幾個正整數與 n 互質
2
3 int phi(){
4     int ans=n;
5     for(int i=2;i*i<=n;i++)
6         if(n%i==0){
7             ans=ans-ans/i;
8             while(n%i==0) n/=i;
9         }
10    if(n>1) ans=ans-ans/n;
11    return ans;
12 }

```

2.5 質數與因數

```

1 歐拉篩O(n)
2 #define MAXN 47000 //sqrt(2^31)=46,340...
3 bool isPrime[MAXN];
4 int p[MAXN];
5 int pSize=0;
6 void getPrimes(){
7     memset(isPrime,true,sizeof(isPrime));
8     isPrime[0]=isPrime[1]=false;
9     for(int i=2;i<MAXN;i++){
10        if(isPrime[i]) p[pSize++]=i;
11        for(int j=0;j<pSize&&i*p[j]<=MAXN;j++){
12            isPrime[i*p[j]]=false;
13            if(i%p[j]==0) break;
14        }
15    }
16 }
17
18 最大公因數 O(log(min(a,b)))
19 int GCD(int a, int b){
20     if(b == 0) return a;
21     return GCD(b, a%b);
22 }

```

```

23 質因數分解
24 void primeFactorization(int n){
25     for(int i=0; i<p.size(); ++i) {
26         if(p[i]*p[i] > n) break;
27         if(n % p[i]) continue;
28         cout << p[i] << ' ';
29         while(n%p[i] == 0) n /= p[i];
30     }
31     if(n != 1) cout << n << ' ';
32     cout << '\n';
33 }
34
35 擴展歐幾里得算法 ax + by = GCD(a, b)
36 int ext_euc(int a, int b, int &x, int &y) {
37     if(b == 0){
38         x = 1, y = 0;
39         return a;
40     }
41     int d = ext_euc(b, a%b, y, x);

```

```

43     y -= a/b*x;
44     return d;
45 }
46 int main(){
47     int a, b, x, y;
48     cin >> a >> b;
49     ext_euc(a, b, x, y);
50     cout << x << ' ' << y << endl;
51     return 0;
52 }
53
54 歌德巴赫猜想
55 解：把偶數 N (6≤N≤10^6) 寫成兩個質數的 和。
56 #define N 20000000
57 int ox[N], p[N], pr;
58 void PrimeTable(){
59     ox[0] = ox[1] = 1;
60     pr = 0;
61     for(int i=2;i<N;i++){
62         if(!ox[i]) p[pr++] = i;
63         for(int j=0; i*p[j]<N&&j<pr; j++)
64             ox[i*p[j]] = 1;
65     }
66 }
67 int main(){
68     PrimeTable();
69     int n;
70     while(cin>>n, n){
71         int x;
72         for(x=1;; x+=2)
73             if(!ox[x] && !ox[n-x]) break;
74         printf("%d = %d + %d\n", n, x, n-x);
75     }
76 }

```

```

77
78 problem :
79 給定整數 N，求N最少可以拆成多少個質數的和。
80 如果N是質數，則答案為 1。
81 如果N是偶數(N!=2)，則答案為2(強歌德巴赫猜想)。
82 如果N是奇數且N-2是質數，則答案為2(2+質數)。
83 其他狀況答案為 3 (弱歌德巴赫猜想)。
84
85 bool isPrime(int n){
86     for(int i=2;i<n;++i){
87         if(i*i>n) return true;
88         if(n%i==0) return false;
89     }
90     return true;
91 }
92 int main(){
93     int n;
94     cin>>n;
95     if(isPrime(n)) cout<<"1\n";
96     else if(n%2==0||isPrime(n-2)) cout<<"2\n";
97     else cout<<"3\n";
98 }

```

2.6 高斯消去

計算 $AX = B$

傳入：

M = 增廣矩陣 $[A|B]$
 equ = 有幾個 equation
 var = 有幾個 variable

回傳： $X = (x_0, \dots, x_{n-1})$ 的解集

>>無法判斷無解或無限多組解<<

```

1 using DBL = double;
2 using mat = vector<vector<DBL>>;
3
4 vector<DBL> Gauss(mat& M, int equ, int var) {
5     auto dcmp = [](DBL a, DBL b=0.0) {
6         return (a > b) - (a < b);
7     };
8

```

```

9     for(int r=0, c=0; r<equ && c<var; ) {
10        int mx = r; // 找絕對值最大的 M[i][c]
11        for(int i=r+1; i<equ; i++) {
12            if(dcmp(abs(M[i][c]),abs(M[mx][c]))==1)
13                mx = i;
14        }
15        if(mx != r) swap(M[mx], M[r]);
16
17        if(dcmp(M[r][c]) == 0) {
18            c++;
19            continue;
20        }
21
22        for(int i=r+1; i<equ; i++) {
23            if(dcmp(M[i][c]) == 0) continue;
24            DBL t = M[i][c] / M[r][c];
25            for(int j=c; j<M[c].size(); j++) {
26                M[i][j] -= t * M[r][j];
27            }
28        }
29        r++, c++;
30    }
31
32    vector<DBL> X(var);
33    for(int i=var-1; i>=0; i--) {
34        X[i] = M[i][var];
35        for(int j=var-1; j>i; j--) {
36            X[i] -= M[i][j] * X[j];
37        }
38        X[i] /= M[i][i];
39    }
40    return X;
41 }

```

2.7 Extended GCD

題目要求：解 $ax + by = n, a, b \in \mathbb{Z}^{0+}$
 已知題幹 $ax + by = n$ 滿足丟番圖方程式
 同時利用貝祖等式 $ax_1 + by_1 = \gcd(a, b)$
 觀察兩式可知將 $ax_1 + by_1 = \gcd(a, b)$ 兩邊乘上 $\frac{n}{\gcd(a, b)}$
 得 $a \frac{nx_1}{\gcd(a, b)} + b \frac{ny_1}{\gcd(a, b)} = n$
 此時可看成 $x = \frac{nx_1}{\gcd(a, b)}, y = \frac{ny_1}{\gcd(a, b)}$
 可以找出一通解

$$x = \frac{nx_1}{\gcd(a, b)} + k \times \frac{b}{\gcd(a, b)}$$

$$y = \frac{ny_1}{\gcd(a, b)} - k \times \frac{a}{\gcd(a, b)}$$

$$k \in \mathbb{Z}$$

(以上通解帶回 $ax + by = n$ 會發現 k 會被消除)
 由於 $x \geq 0, y \geq 0$ 所以

$$x = \frac{nx_1}{\gcd(a, b)} + k \times \frac{b}{\gcd(a, b)}$$

$$y = \frac{ny_1}{\gcd(a, b)} - k \times \frac{a}{\gcd(a, b)}$$

經過移項運算可得

$$-\frac{nx_1}{b} \leq k \leq \frac{ny_1}{a}$$

```

1 ll exgcd(ll a, ll b, ll& x, ll& y) {
2     if (b == 0) {
3         x = 1, y = 0;
4         return a;
5     }
6     ll gcd = exgcd(b, a % b, x, y);
7     ll y1 = y;
8     y = x - (a / b) * y;
9     x = y1;
10    return gcd;
11 }
12 int main() {
13     ll n;
14     ll x, y;
15     ll c1, c2, a, b;

```

```

16 while (~scanf("%lld", &n) && n) {
17     scanf("%lld %lld", &c1, &a);
18     scanf("%lld %lld", &c2, &b);
19     ll gcd = exgcd(a, b, x, y);
20     if (n % gcd != 0) {
21         printf("failed\n");
22         continue;
23     }
24     ll l = ceil((double)(-n) * x / b);
25     ll r = floor((double)(n) * y / a);
26     if (l > r) {
27         printf("failed\n");
28         continue;
29     }
30     if (c1 * b < c2 * a) { //斜率正or負
31         //斜率負，帶入k的上界
32         x = n * x / gcd + b / gcd * r;
33         y = n * y / gcd - a / gcd * r;
34     }
35     else {
36         //斜率正，帶入k的下界
37         x = n * x / gcd + b / gcd * l;
38         y = n * y / gcd - a / gcd * l;
39     }
40     printf("%lld %lld\n", x, y);
41 }
42 return 0;
43 }

```

2.8 大步小步

1 題意
2 給定 B, N, P ，求出 L 滿足 $B^L \equiv N \pmod{P}$ 。
3 題解
4 餘數的循環節長度必定為 P 的因數，因此
 $B^0, B^1, B^2, \dots, B^{P-1}$ ，
5 也就是說如果有解則 $L < N$ ，枚舉 $0, 1, 2, \dots, L-1$
能得到結果，但會超時。
6 將 L 拆成 $mx+y$ ，只要分別枚舉 x, y 就能得到答案，
7 設 $m=\sqrt{P}$ 能保證最多枚舉 $2\sqrt{P}$ 次。
8 $B^{mx+y} \equiv N \pmod{P}$
9 $B^{mx} B^y \equiv N \pmod{P}$
10 $B^y \equiv N(B^{-m})^x \pmod{P}$
11 先求出 $B^0, B^1, B^2, \dots, B^{m-1}$ ，
12 再枚舉 $N(B^{-m})^x, N(B^{-m})^2, \dots$ 查看是否有對應的
 B^y 。
13 這種算法稱為大步小步演算法，
14 大步指的是枚舉 x （一次跨 m 步），
15 小步指的是枚舉 y （一次跨 1 步）。
16 複雜度分析
17 利用 map/unorder_map 存放
 $B^0, B^1, B^2, \dots, B^{m-1}$ ，
18 枚舉 x 查詢 map/unorder_map 是否有對應的 B^y ，
19 存放和查詢最多 $2\sqrt{P}$ 次，時間複雜度為
 $O(\sqrt{P} \log \sqrt{P}) / O(\sqrt{P})$ 。

```

20 using LL = long long;
21 LL B, N, P;
22 LL fpow(LL a, LL b, LL c){
23     LL res=1;
24     for(; b >= 1; b >>= 1){
25         if(b&1)
26             res=(res*a)%c;
27         a=(a*a)%c;
28     }
29     return res;
30 }
31 LL BSGS(LL a, LL b, LL p){
32     a%=p, b%=p;
33     if(a==0)
34         return b==0?1:-1;
35     if(b==1)
36         return 0;
37     map<LL, LL> tb;
38     LL sq=ceil(sqrt(p-1));
39     LL inv=fpow(a, p-sq-1, p);
40     tb[1]=sq;

```

```

42     for(LL i=1, tmp=1; i<sq; ++i){
43         tmp=(tmp*a)%p;
44         if(!tb.count(tmp))
45             tb[tmp]=i;
46     }
47     for(LL i=0; i<sq; ++i){
48         if(tb.count(b)){
49             LL res=tb[b];
50             return i*sq+(res==sq?0:res);
51         }
52         b=(b*inv)%p;
53     }
54     return -1;
55 }
56 int main(){
57     IOS; //輸入優化
58     while(cin>>P>>B>>N){
59         LL ans=BSGS(B, N, P);
60         if(ans!=-1)
61             cout<<"no solution\n";
62         else
63             cout<<ans<<"\n";
64     }
65 }

```

2.9 Pisano Period

```

1 #include <cstdio>
2 #include <vector>
3 using namespace std;
4
5 /*
6  Pisano Period + 快速幂 + mod
7  Pisano Period:
8      費氏數列在mod n的情況下會有循環週期，
9      且週期的結束判斷會在fib[i - 1] == 0 &&
10         fib[i] == 1時，
11         此時循環週期長度是i - 1
12  所以這題是在找出循環週期後，
13  用快速幂並mod(循環週期長度)即可AC(快速幂記得mod)，
14  此外fib要mod n，也要找週期，所以用預處理的方式列表
15  */
16 #define maxn 1005
17
18 /*
19  Pisano period可證一個週期的長度會在[n, n ^ n]之間
20  */
21 //很可惜，會爆
22 // int fib[maxn][maxn * maxn];
23 //改用vector
24 vector<int> fib[maxn];
25 int period[maxn];
26
27 int qpow(int a, unsigned long long b, int mod)
28 {
29     if (b == 0) return a;
30     long long res = 1;
31     while (b) {
32         if (b & 1)
33             res = ((a % mod) * (res % mod)) % mod;
34         a = ((a % mod) * (a % mod)) % mod;
35         b >>= 1;
36     }
37     return res;
38 }
39
40 int main()
41 {
42     int t;
43     unsigned long long a, b;
44     int n;
45
46     //注意：這裡沒算mod 1的循環長度，
47     //因為mod 1都等於0，沒有週期

```

```

49     for (int i = 2; i < maxn; ++i)
50     {
51         fib[i].emplace_back(0);
52         fib[i].emplace_back(1);
53         for (int j = 2; j < maxn * maxn; ++j)
54         {
55             fib[i].emplace_back(
56                 (fib[i][j-1]*fib[i][j-2])%i
57             );
58             if (fib[i][j-1]==0&&fib[i][j]==1)
59             {
60                 period[i] = j - 1;
61                 break;
62             }
63         }
64     }
65
66     scanf("%d", &t);
67
68     while (t--)
69     {
70         scanf("%llu %llu %d", &a, &b, &n);
71         if (a == 0)
72             puts("0");
73         else if (n == 1) //當mod 1時任何數都是0，
74             puts("0");
75         //所以直接輸出0，避免我們沒算
76         //fib[1][i]的問題(Runtime
77         error)
78         printf("%d\n",
79             fib[n][qpow(a % period[n], b,
80                 period[n])]);
81     }
82 }

```

2.10 矩陣快速幂

```

1 using ll = long long;
2 using mat = vector<vector<ll>>>;
3 const int mod = 1e9 + 7;
4
5 mat operator*(mat A, mat B) {
6     mat res(A.size(), vector<ll>(B[0].size()));
7     for(int i=0; i<A.size(); i++) {
8         for(int j=0; j<B[0].size(); j++) {
9             for(int k=0; k<B.size(); k++) {
10                 res[i][j] += A[i][k] * B[k][j] % mod;
11                 res[i][j] %= mod;
12             }
13         }
14     }
15     return res;
16 }
17
18 mat I = ;
19 // compute matrix M^n
20 // 需先 init I 矩陣
21 mat mpow(mat& M, int n) {
22     if(n <= 1) return n ? M : I;
23     mat v = mpow(M, n>>1);
24     return (n & 1) ? v*v*M : v*v;
25 }
26
27 // 迴圈版本
28 mat mpow(mat M, int n) {
29     mat res(M.size(), vector<ll>(M[0].size()));
30     for(int i=0; i<res.size(); i++)
31         res[i][i] = 1;
32     for(; n; n>>=1) {
33         if(n & 1) res = res * M;
34         M = M * M;
35     }
36     return res;
37 }

```

3 algorithm

3.1 greedy

```

1 刪數字問題
2 //problem
3 給定一個數字  $N(\leq 10^{100})$ ，需要刪除  $K$  個數字，
4 請問刪除  $K$  個數字後最小的數字為何？
5 //solution
6 刪除滿足第  $i$  位數大於第  $i+1$  位數的最左邊第  $i$ 
7 位數，
8 扣除高位數的影響較扣除低位數的大。
9 //code
10 int main(){
11     string s;
12     int k;
13     cin>>s>>k;
14     for(int i=0;i<k;++i){
15         if((int)s.size()==0) break;
16         int pos =(int)s.size()-1;
17         for(int j=0;j<(int)s.size()-1;++j){
18             if(s[j]>s[j+1]){
19                 pos=j;
20                 break;
21             }
22         }
23         s.erase(pos,1);
24     }
25     while((int)s.size()>0&&s[0]=='0')
26         s.erase(0,1);
27     if((int)s.size()) cout<<s<<'\n';
28     else cout<<0<<'\n';
29 }
30 最小區間覆蓋長度
31 //problem
32 給定  $n$  條線段區間為  $[Li,Ri]$ ，
33 請問最少要選幾個區間才能完全覆蓋  $[0,S]$ ？
34 //solution
35 先將所有區間依照左界由小到大排序，
36 對於當前區間  $[Li,Ri]$ ，要從左界  $>Ri$  的所有區間中，
37 找到有著最大的右界的區間，連接當前區間。
38 //problem
39 長度  $n$  的直線中有數個加熱器，
40 在  $x$  的加熱器可以讓  $[x-r,x+r]$  內的物品加熱，
41 問最少要幾個加熱器可以把  $[0,n]$  的範圍加熱。
42 //solution
43 對於最左邊沒加熱的點 $a$ ，選擇最遠可以加熱 $a$ 的加熱器，
44 更新已加熱範圍，重複上述動作繼續尋找加熱器。
45 //code
46 int main(){
47     int n, r;
48     int a[1005];
49     cin>>n>>r;
50     for(int i=1;i<=n;++i) cin>>a[i];
51     int i=1,ans=0;
52     while(i<=n){
53         int R=min(i+r-1,n),L=max(i-r+1,0)
54         int nextR=-1;
55         for(int j=R;j>=L;--j){
56             if(a[j]){
57                 nextR=j;
58                 break;
59             }
60         }
61         if(nextR!=-1){
62             ans++;
63             break;
64         }
65         ++ans;
66         i=nextR+r;
67     }
68     cout<<ans<<'\n';
69 }
70 最多不重疊區間
71 //problem
72 給你  $n$  條線段區間為  $[Li,Ri]$ ，
73 請問最多可以選擇幾條不重疊的線段(頭尾可相連)？

```

```

74 //solution
75 依照右界由小到大排序，
76 每次取到一個不重疊的線段，答案 +1。
77 //code
78 struct Line{
79     int L,R;
80     bool operator<(const Line &rhs)const{
81         return R<rhs.R;
82     }
83 };
84 int main(){
85     int t;
86     cin>>t;
87     Line a[30];
88     while(t--){
89         int n=0;
90         while(cin>>a[n].L>a[n].R,a[n].L||a[n].R){
91             ++n;
92         }
93         sort(a,a+n);
94         int ans=1,R=a[0].R;
95         for(int i=1;i<n;i++){
96             if(a[i].L>R){
97                 ++ans;
98                 R=a[i].R;
99             }
100         }
101         cout<<ans<<'\n';
102     }
103 }
104 最小化最大延遲問題
105 //problem
106 給定  $N$  項工作，每項工作的需要處理時長為  $Ti$ ，
107 期限是  $Di$ ，第  $i$  項工作延遲的時間為
108  $Li=\max(0,Fi-Di)$ ，
109 原本 $Fi$  為第  $i$  項工作的完成時間，
110 求一種工作排序使  $\max Li$  最小。
111 //solution
112 按照到期時間從早到晚處理。
113 //code
114 struct Work{
115     int t, d;
116     bool operator<(const Work &rhs)const{
117         return d<rhs.d;
118     }
119 };
120 int main(){
121     int n;
122     Work a[10000];
123     cin>>n;
124     for(int i=0;i<n;++i)
125         cin>>a[i].t>>a[i].d;
126     sort(a,a+n);
127     int maxL=0,sumT=0;
128     for(int i=0;i<n;++i){
129         sumT+=a[i].t;
130         maxL=max(maxL,sumT-a[i].d);
131     }
132     cout<<maxL<<'\n';
133 }
134 最少延遲數量問題
135 //problem
136 給定  $N$  個工作，每個工作的需要處理時長為  $Ti$ ，
137 期限是  $Di$ ，求一種工作排序使得逾期工作數量最小。
138 //solution
139 期限越早到期的工作越先做。
140 將工作依照到期時間從早到晚排序，
141 依序放入工作列表中，如果發現有工作預期，
142 就從目前選擇的工作中，移除耗時最長的工作。
143 上述方法為 Moore-Hodgson's Algorithm。
144 //problem
145 給定烏龜的重量和可承受重量，問最多可以疊幾隻烏龜？
146 //solution
147 和最少延遲數量問題是相同的問題，只要將題敘做轉換。
148 工作處理時長  $\rightarrow$  烏龜重量
149 工作期限  $\rightarrow$  烏龜可承受重量
150 多少工作不延期  $\rightarrow$  可以疊幾隻烏龜
151 //code

```

```

151 struct Work{
152     int t, d;
153     bool operator<(const Work &rhs)const{
154         return d<rhs.d;
155     }
156 };
157 int main(){
158     int n=0;
159     Work a[10000];
160     priority_queue<int> pq;
161     while(cin>>a[n].t>>a[n].d)
162         ++n;
163     sort(a,a+n);
164     int sumT=0,ans=n;
165     for(int i=0;i<n;++i){
166         pq.push(a[i].t);
167         sumT+=a[i].t;
168         if(a[i].d<sumT){
169             int x=pq.top();
170             pq.pop();
171             sumT-=x;
172             --ans;
173         }
174     }
175     cout<<ans<<'\n';
176 }
177 任務調度問題
178 //problem
179 給定  $N$  項工作，每項工作的需要處理時長為  $Ti$ ，
180 期限是  $Di$ ，如果第  $i$  項工作延遲需要受到  $pi$ 
181 單位懲罰，
182 請問最少會受到多少單位懲罰。
183 //solution
184 依照懲罰由大到小排序，
185 每項工作依序嘗試可不可以放在
186  $Di-Ti+1, Di-Ti, \dots, 1, 0$ ，
187 如果有空間就放進去，否則延後執行。
188 //problem
189 給定  $N$  項工作，每項工作的需要處理時長為  $Ti$ ，
190 期限是  $Di$ ，如果第  $i$  項工作在期限內完成會獲得  $ai$ 
191 單位獎勵，
192 請問最多會獲得多少單位獎勵。
193 //solution
194 和上題相似，這題變成依照獎勵由大到小排序。
195 //code
196 struct Work{
197     int d,p;
198     bool operator<(const Work &rhs)const{
199         return p>rhs.p;
200     }
201 };
202 int main(){
203     int n;
204     Work a[100005];
205     bitset<100005> ok;
206     while(cin>>n){
207         ok.reset();
208         for(int i=0;i<n;++i)
209             cin>>a[i].d>>a[i].p;
210         sort(a,a+n);
211         int ans=0;
212         for(int i=0;i<n;++i){
213             int j=a[i].d;
214             while(j--){
215                 if(!ok[j]){
216                     ans+=a[i].p;
217                     ok[j]=true;
218                     break;
219                 }
220             }
221         }
222     }
223     cout<<ans<<'\n';
224 }

```


3.2 JosephusProblem

```

1 //JosephusProblem, 只是規定要先砍1號
2 //所以當作有 n - 1個人, 目標的13順移成12
3 //再者從0開始比較好算, 所以目標12順移成11
4
5 // O(n)
6 int getWinner(int n, int k) {
7     int winner = 0;
8     for (int i = 1; i <= n; ++i)
9         winner = (winner + k) % i;
10    return winner;
11 }
12
13 int main() {
14     int n;
15     while (scanf("%d", &n) != EOF && n){
16         --n;
17         for (int k = 1; k <= n; ++k){
18             if (getWinner(n, k) == 11){
19                 printf("%d\n", k);
20                 break;
21             }
22         }
23     }
24     return 0;
25 }
26
27 // O(k log(n))
28 int josephus(int n, int k) {
29     if (n == 1) return 0;
30     if (k == 1) return n - 1;
31     if (k > n) return (josephus(n-1,k)+k)%n;
32     int res = josephus(n - n / k, k);
33     res -= n % k;
34     if (res < 0)
35         res += n; // mod n
36     else
37         res += res / (k - 1); // 还原位置
38     return res;
39 }

```

3.3 二分搜

```

1 // 以下經過check()後 . 為false, o 為true
2 //皆為[l, r]區間
3 //.....voooooo 即答案左邊界, 符合條件最小的
4 int bsearch(int l, int r)
5 {
6     while (l < r)
7     {
8         int mid = (l + r) >> 1;
9         if (check(mid)) r = mid;
10        else l = mid + 1;
11    }
12    return l;
13 }
14
15 //ooooov..... 即答案右邊界, 符合條件最大的
16 int bsearch(int l, int r)
17 {
18     while (l < r)
19     {
20         int mid = (l + r + 1) >> 1;
21         if (check(mid)) l = mid;
22         else r = mid - 1;
23     }
24     return l;
25 }

```

3.4 三分搜

```

1 題意
2 給定兩射線方向和速度, 問兩射線最近距離。

```

```

3 題解
4 假設 F(t) 為兩射線在時間 t 的距離, F(t)
5 為二次函數,
6 可用三分搜找二次函數最小值。
7 struct Point{
8     double x, y, z;
9     Point() {}
10    Point(double _x, double _y, double _z):
11        x(_x), y(_y), z(_z){}
12    friend istream& operator>>(istream& is,
13        Point& p) {
14        is >> p.x >> p.y >> p.z;
15        return is;
16    }
17    Point operator+(const Point &rhs) const{
18        return Point(x+rhs.x, y+rhs.y, z+rhs.z);
19    }
20    Point operator-(const Point &rhs) const{
21        return Point(x-rhs.x, y-rhs.y, z-rhs.z);
22    }
23    Point operator*(const double &d) const{
24        return Point(x*d, y*d, z*d);
25    }
26    Point operator/(const double &d) const{
27        return Point(x/d, y/d, z/d);
28    }
29    double dist(const Point &rhs) const{
30        double res = 0;
31        res += (x-rhs.x)*(x-rhs.x);
32        res += (y-rhs.y)*(y-rhs.y);
33        res += (z-rhs.z)*(z-rhs.z);
34        return res;
35    }
36 };
37 int main(){
38     IOS; //輸入優化
39     int T;
40     cin>>T;
41     for(int ti=1; ti<=T; ++ti){
42         double time;
43         Point x1, y1, d1, x2, y2, d2;
44         cin>>time>>x1>>y1>>x2>>y2;
45         d1=(y1-x1)/time;
46         d2=(y2-x2)/time;
47         double L=0, R=1e8, m1, m2, f1, f2;
48         double ans = x1.dist(x2);
49         while(abs(L-R)>1e-10){
50             m1=(L+R)/2;
51             m2=(m1+R)/2;
52             f1=((d1*m1)+x1).dist((d2*m1)+x2);
53             f2=((d1*m2)+x1).dist((d2*m2)+x2);
54             ans = min(ans, min(f1, f2));
55             if(f1<f2) R=m2;
56             else L=m1;
57         }
58         cout<<"Case "<<ti<<" ";
59         cout << fixed << setprecision(4) <<
60             sqrt(ans) << '\n';
61     }
62 }
63
64 //oi wiki模板, [l, r]
65 //只要是單峰函數, 三分可找最大或最小, 以下為最小化
66 //計算lmid以及rmid時要避免數字溢出
67 while (r - l > eps) {
68     mid = (l + r) / 2;
69     lmid = mid - eps;
70     rmid = mid + eps;
71     if (f(lmid) < f(rmid)) r = mid;
72     else l = mid;
73 }

```

3.5 dinic

```

1 const int maxn = 1e5 + 10;
2 const int inf = 0x3f3f3f3f;
3 struct Edge {

```

```

4     int s, t, cap, flow;
5 };
6 int n, m, S, T;
7 int level[maxn], dfs_idx[maxn];
8 vector<Edge> E;
9 vector<vector<int>> G;
10 void init() {
11     S = 0;
12     T = n + m;
13     E.clear();
14     G.assign(maxn, vector<int>());
15 }
16 void addEdge(int s, int t, int cap) {
17     E.push_back({s, t, cap, 0});
18     E.push_back({t, s, 0, 0});
19     G[s].push_back(E.size()-2);
20     G[t].push_back(E.size()-1);
21 }
22 bool bfs() {
23     queue<int> q({S});
24     memset(level, -1, sizeof(level));
25     level[S] = 0;
26     while(!q.empty()) {
27         int cur = q.front();
28         q.pop();
29         for(int i : G[cur]) {
30             Edge e = E[i];
31             if(level[e.t]==-1 &&
32                 e.cap-e.flow) {
33                 level[e.t] = level[e.s] + 1;
34                 q.push(e.t);
35             }
36         }
37     }
38     return level[T];
39 }
40 int dfs(int cur, int lim) {
41     if(cur==T || lim==0) return lim;
42     int result = 0;
43     for(int& i=dfs_idx[cur]; i<G[cur].size()
44         && lim>0; i++) {
45         Edge& e = E[G[cur][i]];
46         if(level[e.s]+1 != level[e.t])
47             continue;
48         int flow = dfs(e.t, min(lim,
49             e.cap-e.flow));
50         if(flow <= 0) continue;
51         e.flow += flow;
52         result += flow;
53         E[G[cur][i]^1].flow -= flow;
54         lim -= flow;
55     }
56     return result;
57 }
58 int dinic() { // O(V^2)E
59     int result = 0;
60     while(bfs()) {
61         memset(dfs_idx, 0, sizeof(dfs_idx));
62         result += dfs(S, inf);
63     }
64     return result;
65 }

```

3.6 SCC Tarjan

```

1 //單純考SCC, 每個SCC中找成本最小的蓋, 如果有多個一樣小
2 //的要數出來, 因為題目要方法數
3 //注意以下程式有縮點, 但沒存起來,
4 //存法就是開一個array -> ID[u] = SCCID
5 #define maxn 100005
6 #define MOD 1000000007
7 long long cost[maxn];
8 vector<vector<int>> G;
9 int SCC = 0;
10 stack<int> sk;
11 int dfn[maxn];
12 int low[maxn];

```

```

13 bool inStack[maxn];
14 int dfsTime = 1;
15 long long totalCost = 0;
16 long long ways = 1;
17 void dfs(int u) {
18     dfn[u] = low[u] = dfsTime;
19     ++dfsTime;
20     sk.push(u);
21     inStack[u] = true;
22     for (int v: G[u]) {
23         if (dfn[v] == 0) {
24             dfs(v);
25             low[u] = min(low[u], low[v]);
26         }
27         else if (inStack[v]) {
28             //屬於同個SCC且是我的back edge
29             low[u] = min(low[u], dfn[v]);
30         }
31     }
32     //如果是SCC
33     if (dfn[u] == low[u]) {
34         long long minCost = 0x3f3f3f3f;
35         int currWays = 0;
36         ++SCC;
37         while (1) {
38             int v = sk.top();
39             inStack[v] = 0;
40             sk.pop();
41             if (minCost > cost[v]) {
42                 minCost = cost[v];
43                 currWays = 1;
44             }
45             else if (minCost == cost[v]) {
46                 ++currWays;
47             }
48             if (v == u)
49                 break;
50         }
51         totalCost += minCost;
52         ways = (ways * currWays) % MOD;
53     }
54 }
55 int main() {
56     int n;
57     scanf("%d", &n);
58     for (int i = 1; i <= n; ++i)
59         scanf("%lld", &cost[i]);
60     G.assign(n + 5, vector<int>());
61     int m;
62     scanf("%d", &m);
63     int u, v;
64     for (int i = 0; i < m; ++i) {
65         scanf("%d %d", &u, &v);
66         G[u].emplace_back(v);
67     }
68     for (int i = 1; i <= n; ++i) {
69         if (dfn[i] == 0)
70             dfs(i);
71     }
72     printf("%lld %lld\n", totalCost, ways % MOD);
73     return 0;
74 }

```

3.7 BCC 邊

```

1 //oi-wiki, 找無向圖的邊雙連通分量個數,
2 //並輸出每個邊雙連通分量
3 //對於任意u、v, 刪去哪個邊都不會不連通
4 //-> 邊雙連通(V + E)
5 constexpr int N = 5e5 + 5, M = 2e6 + 5;
6 int n, m, ans;
7 int tot = 1, hd[N];
8
9 struct edge {
10     int to, nt;
11 } e[M << 1];

```

```

12
13 void add(int u, int v) { e[++tot].to = v,
14     e[tot].nt = hd[u], hd[u] = tot; }
15
16 void uadd(int u, int v) { add(u, v), add(v,
17     u); }
18
19 bool bz[M << 1];
20 int bcc_cnt, dfn[N], low[N], vis_bcc[N];
21 vector<vector<int>> bcc;
22
23 void tarjan(int x, int in) {
24     dfn[x] = low[x] = ++bcc_cnt;
25     for (int i = hd[x]; i; i = e[i].nt) {
26         int v = e[i].to;
27         if (dfn[v] == 0) {
28             tarjan(v, i);
29             if (dfn[x] < low[v]) bz[i] = bz[i ^ 1]
30                 = true;
31             low[x] = min(low[x], low[v]);
32         } else if (i != (in ^ 1))
33             low[x] = min(low[x], dfn[v]);
34     }
35 }
36 void dfs(int x, int id) {
37     vis_bcc[x] = id, bcc[id - 1].push_back(x);
38     for (int i = hd[x]; i; i = e[i].nt) {
39         int v = e[i].to;
40         if (vis_bcc[v] || bz[i]) continue;
41         dfs(v, id);
42     }
43 }
44 int main() {
45     cin.tie(nullptr)->sync_with_stdio(false);
46     cin >> n >> m;
47     int u, v;
48     for (int i = 1; i <= m; i++) {
49         cin >> u >> v;
50         if (u == v) continue;
51         uadd(u, v);
52     }
53     for (int i = 1; i <= n; i++)
54         if (dfn[i] == 0) tarjan(i, 0);
55     for (int i = 1; i <= n; i++)
56         if (vis_bcc[i] == 0) {
57             bcc.push_back(vector<int>());
58             dfs(i, ++ans);
59         }
60     cout << ans << '\n';
61     for (int i = 0; i < ans; i++) {
62         cout << bcc[i].size();
63         for (int j = 0; j < bcc[i].size(); j++)
64             cout << ' ' << bcc[i][j];
65         cout << '\n';
66     }
67     return 0;
68 }

```

3.8 BCC 點

```

1 //oi-wiki, 找無向圖的點雙連通分量個數,
2 //並輸出每個點雙連通分量
3 //對於任意u、v, 刪去哪個點(只能刪一個)都不會不連通
4 //-> 點雙連通(V + E)
5 constexpr int N = 5e5 + 5, M = 2e6 + 5;
6 int n, m;
7
8 struct edge {
9     int to, nt;
10 } e[M << 1];
11
12 int hd[N], tot = 1;
13
14 void add(int u, int v) { e[++tot] = edge{v,
15     hd[u]}, hd[u] = tot; }

```

```

16 void uadd(int u, int v) { add(u, v), add(v,
17     u); }
18
19 int ans;
20 int dfn[N], low[N], bcc_cnt;
21 int sta[N], top, cnt;
22 bool cut[N];
23 vector<int> dcc[N];
24 int root;
25
26 void tarjan(int u) {
27     dfn[u] = low[u] = ++bcc_cnt, sta[++top] =
28     u;
29     if (u == root && hd[u] == 0) {
30         dcc[++cnt].push_back(u);
31         return;
32     }
33     int f = 0;
34     for (int i = hd[u]; i; i = e[i].nt) {
35         int v = e[i].to;
36         if (!dfn[v]) {
37             tarjan(v);
38             low[u] = min(low[u], low[v]);
39             if (low[v] >= dfn[u]) {
40                 if (++f > 1 || u != root) cut[u] =
41                     true;
42                 cnt++;
43                 do dcc[cnt].push_back(sta[top--]);
44                 while (sta[top + 1] != v);
45                 dcc[cnt].push_back(u);
46             }
47         } else if (i != f)
48             low[u] = min(low[u], dfn[v]);
49     }
50 }
51
52 int main() {
53     cin.tie(nullptr)->sync_with_stdio(false);
54     cin >> n >> m;
55     int u, v;
56     for (int i = 1; i <= m; i++) {
57         cin >> u >> v;
58         if (u != v) uadd(u, v);
59     }
60     for (int i = 1; i <= n; i++)
61         if (!dfn[i]) root = i, tarjan(i);
62     cout << cnt << '\n';
63     for (int i = 1; i <= cnt; i++) {
64         cout << dcc[i].size() << ' ';
65         for (int j = 0; j < dcc[i].size(); j++)
66             cout << dcc[i][j] << ' ';
67         cout << '\n';
68     }
69     return 0;
70 }

```

3.9 ArticulationPoints Tarjan

```

1 vector<vector<int>> G;
2 int N, timer;
3 bool visited[105];
4 int dfn[105]; // 第一次visit的時間
5 int low[105];
6 //最小能回到的父節點
7 //(不能是自己的parent)的visTime
8 int res;
9 //求割點數量
10 void tarjan(int u, int parent) {
11     int child = 0;
12     bool isCut = false;
13     visited[u] = true;
14     dfn[u] = low[u] = ++timer;
15     for (int v: G[u]) {
16         if (!visited[v]) {
17             ++child;
18             tarjan(v, u);

```

```

19     low[u] = min(low[u], low[v]);
20     if (parent != -1 && low[v] >=
        dfn[u])
21         isCut = true;
22 }
23 else if (v != parent)
24     low[u] = min(low[u], dfn[v]);
25 }
26 //If u is root of DFS
    tree->有兩個以上的children
27 if (parent == -1 && child >= 2)
28     isCut = true;
29 if (isCut) ++res;
30 }
31 int main() {
32     char input[105];
33     char* token;
34     while (scanf("%d", &N) != EOF && N) {
35         G.assign(105, vector<int>());
36         memset(visited, false,
            sizeof(visited));
37         memset(low, 0, sizeof(low));
38         memset(dfn, 0, sizeof(visited));
39         timer = 0;
40         res = 0;
41         getchar(); // for \n
42         while (fgets(input, 105, stdin)) {
43             if (input[0] == '\0')
44                 break;
45             int size = strlen(input);
46             input[size - 1] = '\0';
47             --size;
48             token = strtok(input, " ");
49             int u = atoi(token);
50             int v;
51             while (token = strtok(NULL, " "))
52                 v = atoi(token);
53             G[u].emplace_back(v);
54             G[v].emplace_back(u);
55         }
56         tarjan(1, -1);
57         printf("%d\n", res);
58     }
59 }
60 return 0;
61 }

```

3.10 最小樹狀圖

```

1 const int maxn = 60 + 10;
2 const int inf = 0x3f3f3f3f;
3 struct Edge {
4     int s, t, cap, cost;
5 }; // cap 為頻寬 (optional)
6 int n, m, c;
7 int inEdge[maxn], idx[maxn], pre[maxn],
    vis[maxn];
8 // 對於每個點，選擇對它入度最小的那條邊
9 // 找環，如果沒有則 return;
10 // 進行縮環並更新其他點到環的距離。
11 int dirMST(vector<Edge> edges, int low) {
12     int result = 0, root = 0, N = n;
13     while(true) {
14         memset(inEdge, 0x3f, sizeof(inEdge));
15         // 找所有點的 in edge 放進 inEdge
16         // optional: low 為最小 cap 限制
17         for(const Edge& e : edges) {
18             if(e.cap < low) continue;
19             if(e.s!=e.t &&
                e.cost<inEdge[e.t]) {
20                 inEdge[e.t] = e.cost;
21                 pre[e.t] = e.s;
22             }
23         }
24         for(int i=0; i<N; i++) {
25             if(i!=root && inEdge[i]==inf)

```

```

26         return -1; //除了root 還有點沒有 in
            edge
27     }
28     int seq = inEdge[root] = 0;
29     memset(idx, -1, sizeof(idx));
30     memset(vis, -1, sizeof(vis));
31     // 找所有的 cycle，一起編號為 seq
32     for(int i=0; i<N; i++) {
33         result += inEdge[i];
34         int cur = i;
35         while(vis[cur]!=i &&
            idx[cur]==-1) {
36             idx[cur]=i;
37             if(cur == root) break;
38             vis[cur] = i;
39             cur = pre[cur];
40         }
41         if(cur!=root && idx[cur]==-1) {
42             for(int j=pre[cur]; j!=cur;
                j=pre[j])
43                 idx[j] = seq;
44             idx[cur] = seq++;
45         }
46         if(seq == 0) return result; // 沒有
            cycle
47         for(int i=0; i<N; i++)
48             // 沒有被縮點的點
49             if(idx[i] == -1) idx[i] = seq++;
50         // 縮點並重新編號
51         for(Edge& e : edges) {
52             if(idx[e.s] != idx[e.t])
53                 e.cost -= inEdge[e.t];
54             e.s = idx[e.s];
55             e.t = idx[e.t];
56         }
57         N = seq;
58         root = idx[root];
59     }
60 }

```

3.11 KM

```

1 #define maxn 505
2 int W[maxn][maxn];
3 int Lx[maxn], Ly[maxn];
4 bool S[maxn], T[maxn];
5 //L[i] = j -> S_i配給T_j, -1 for 還沒匹配
6 int L[maxn];
7 int n;
8 bool match(int i) {
9     S[i] = true;
10    for (int j = 0; j < n; ++j) {
11        // KM重點
12        // Lx + Ly >= selected_edge(x, y)
13        // 要想辦法降低 Lx + Ly
14        // 所以選 Lx + Ly == selected_edge(x, y)
15        if (Lx[i] + Ly[j] == W[i][j] &&
            !T[j]) {
16            T[j] = true;
17            if ((L[j] == -1) || match(L[j])) {
18                L[j] = i;
19                return true;
20            }
21        }
22    }
23    return false;
24 }
25 //修改二分圖上的交錯路徑上點的權重
26 //此舉是在通過調整vertex labeling看看
27 //能不能產生出新的增廣路
28 //(KM的增廣路要求 Lx[i] + Ly[j] == W[i][j])
29 //在這裡優先從最小的 diff 調看，才能保證最大權重匹配
30 void update() {
31     int diff = 0x3f3f3f3f;
32     for (int i = 0; i < n; ++i) {
33         if (S[i]) {
34             for (int j = 0; j < n; ++j) {

```

```

35         if (!T[j])
36             diff = min(diff, Lx[i] +
                Ly[j] - W[i][j]);
37     }
38 }
39 }
40 for (int i = 0; i < n; ++i) {
41     if (S[i]) Lx[i] -= diff;
42     if (T[i]) Ly[i] += diff;
43 }
44 }
45 void KM() {
46     for (int i = 0; i < n; ++i) {
47         L[i] = -1;
48         Lx[i] = Ly[i] = 0;
49         for (int j = 0; j < n; ++j)
50             Lx[i] = max(Lx[i], W[i][j]);
51     }
52     for (int i = 0; i < n; ++i) {
53         while(1) {
54             memset(S, false, sizeof(S));
55             memset(T, false, sizeof(T));
56             if (match(i)) break;
57             else update(); //去調整vertex
                labeling以增加增廣路徑
58         }
59     }
60 }
61 int main() {
62     while (scanf("%d", &n) != EOF) {
63         for (int i = 0; i < n; ++i)
64             for (int j = 0; j < n; ++j)
65                 scanf("%d", &W[i][j]);
66         KM();
67         int res = 0;
68         for (int i = 0; i < n; ++i) {
69             if (i != 0)
70                 printf("%d", Lx[i]);
71             else
72                 printf("%d", Lx[i]);
73             res += Lx[i];
74         }
75         puts("");
76         for (int i = 0; i < n; ++i) {
77             if (i != 0)
78                 printf("%d", Ly[i]);
79             else
80                 printf("%d", Ly[i]);
81             res += Ly[i];
82         }
83         puts("");
84         printf("%d\n", res);
85     }
86     return 0;
87 }

```

3.12 二分圖最大匹配

```

1 /* 核心：最大點獨立集 = |V| -
    /最大匹配數/，用匈牙利演算法找出最大匹配數 */
2 vector<Student> boys;
3 vector<Student> girls;
4 vector<vector<int>> G;
5 bool used[505];
6 int p[505];
7 bool match(int i) {
8     for (int j: G[i]) {
9         if (!used[j]) {
10             used[j] = true;
11             if (p[j] == -1 || match(p[j])) {
12                 p[j] = i;
13                 return true;
14             }
15         }
16     }
17     return false;
18 }

```



```

19 void maxMatch(int n) {
20     memset(p, -1, sizeof(p));
21     int res = 0;
22     for (int i = 0; i < boys.size(); ++i) {
23         memset(used, false, sizeof(used));
24         if (match(i)) ++res;
25     }
26     cout << n - res << '\n';
27 }

```

3.13 差分

用途：在區間 [l, r] 加上一個數字 v。

b[l] += v; (b[0~l] 加上v)

b[r+1] -= v; (b[r+1~n] 減去v (b[r] 仍保留v))

給的 a[] 是前綴和數列，建構 b[]，

因為 a[i] = b[0] + b[1] + b[2] + ... + b[i]，

所以 b[i] = a[i] - a[i-1]。

在 b[l] 加上 v，b[r+1] 減去 v，

最後再從 0 跑到 n 使 b[i] += b[i-1]。

這樣一來，b[] 是一個在某區間加上v的前綴和。

```

10 int a[1000], b[1000];
11 // a: 前綴和數列, b: 差分數列
12 int main(){
13     int n, l, r, v;
14     cin >> n;
15     for(int i=1; i<=n; i++){
16         cin >> a[i];
17         b[i] = a[i] - a[i-1]; //建構差分數列
18     }
19     cin >> l >> r >> v;
20     b[l] += v;
21     b[r+1] -= v;
22     for(int i=1; i<=n; i++){
23         b[i] += b[i-1];
24         cout << b[i] << ' ';
25     }
26 }

```

3.14 MCMF

```

1 #define maxn 225
2 #define INF 0x3f3f3f3f
3 struct Edge {
4     int u, v, cap, flow, cost;
5 };
6 //node size, edge size, source, target
7 int n, m, s, t;
8 vector<vector<int>> G;
9 vector<Edge> edges;
10 bool inqueue[maxn];
11 long long dis[maxn];
12 int parent[maxn];
13 long long outFlow[maxn];
14 void addEdge(int u, int v, int cap, int cost) {
15     edges.emplace_back(Edge{u, v, cap, 0, cost});
16     edges.emplace_back(Edge{v, u, 0, 0, -cost});
17     m = edges.size();
18     G[u].emplace_back(m - 2);
19     G[v].emplace_back(m - 1);
20 }
21 //一求最短路的同時一邊MaxFlow
22 bool SPFA(long long& maxFlow, long long& minCost) {
23     // memset(outFlow, 0x3f, sizeof(outFlow));
24     memset(dis, 0x3f, sizeof(dis));
25     memset(inqueue, false, sizeof(inqueue));
26     queue<int> q;
27     q.push(s);
28     dis[s] = 0;

```

```

29     inqueue[s] = true;
30     outFlow[s] = INF;
31     while (!q.empty()) {
32         int u = q.front();
33         q.pop();
34         inqueue[u] = false;
35         for (const int edgeIndex: G[u]) {
36             const Edge& edge = edges[edgeIndex];
37             if ((edge.cap > edge.flow) && (dis[edge.v] > dis[u] + edge.cost)) {
38                 dis[edge.v] = dis[u] + edge.cost;
39                 parent[edge.v] = edgeIndex;
40                 outFlow[edge.v] = min(outFlow[u], (long long)(edge.cap - edge.flow));
41                 if (!inqueue[edge.v]) {
42                     q.push(edge.v);
43                     inqueue[edge.v] = true;
44                 }
45             }
46         }
47     }
48     //如果dis[t] > 0代表根本不賺還倒賠
49     if (dis[t] > 0)
50         return false;
51     maxFlow += outFlow[t];
52     minCost += dis[t] * outFlow[t];
53     //一路更新回去這次最短路流完後要維護的
54     //MaxFlow演算法相關(如反向邊等)
55     int curr = t;
56     while (curr != s) {
57         edges[parent[curr]].flow += outFlow[t];
58         edges[parent[curr] ^ 1].flow -= outFlow[t];
59         curr = edges[parent[curr]].u;
60     }
61     return true;
62 }
63 long long MCMF() {
64     long long maxFlow = 0;
65     long long minCost = 0;
66     while (SPFA(maxFlow, minCost))
67         ;
68     return minCost;
69 }
70 int main() {
71     int T;
72     scanf("%d", &T);
73     for (int Case = 1; Case <= T; ++Case){
74         //總共幾個月，囤貨成本
75         int M, I;
76         scanf("%d %d", &M, &I);
77         //node size
78         n = M + M + 2;
79         G.assign(n + 5, vector<int>());
80         edges.clear();
81         s = 0;
82         t = M + M + 1;
83         for (int i = 1; i <= M; ++i) {
84             int produceCost, produceMax, sellPrice, sellMax, inventoryMonth;
85             scanf("%d %d %d %d %d", &produceCost, &produceMax, &sellPrice, &sellMax, &inventoryMonth);
86             addEdge(s, i, produceMax, produceCost);
87             addEdge(M + i, t, sellMax, -sellPrice);
88             for (int j = 0; j <= inventoryMonth; ++j) {
89                 if (i + j <= M)

```

```

90                 addEdge(i, M + i + j, INF, I * j);
91             }
92         }
93         printf("Case %d: %lld\n", Case, -MCMF());
94     }
95     return 0;
96 }

```

3.15 Dancing Links

```

1 struct DLX {
2     int seq, resSize;
3     int col[maxn], row[maxn];
4     int U[maxn], D[maxn], R[maxn], L[maxn];
5     int rowHead[maxn], colSize[maxn];
6     int result[maxn];
7     DLX(int r, int c) {
8         for(int i=0; i<=c; i++) {
9             L[i] = i-1, R[i] = i+1;
10            U[i] = D[i] = i;
11        }
12        L[R[seq=c]=0]=c;
13        resSize = -1;
14        memset(rowHead, 0, sizeof(rowHead));
15        memset(colSize, 0, sizeof(colSize));
16    }
17    void insert(int r, int c) {
18        row[++seq]=r, col[seq]=c, ++colSize[c];
19        U[seq]=c, D[seq]=D[c], U[D[c]]=seq, D[c]=seq;
20        if(rowHead[r]) {
21            L[seq]=rowHead[r], R[seq]=R[rowHead[r]];
22            L[R[rowHead[r]]]=seq, R[rowHead[r]]=seq;
23        } else {
24            rowHead[r] = L[seq] = R[seq] = seq;
25        }
26    }
27    void remove(int c) {
28        L[R[c]] = L[c], R[L[c]] = R[c];
29        for(int i=D[c]; i!=c; i=D[i]) {
30            for(int j=R[i]; j!=i; j=R[j]) {
31                U[D[j]] = U[j];
32                D[U[j]] = D[j];
33                --colSize[col[j]];
34            }
35        }
36    }
37    void recover(int c) {
38        for(int i=U[c]; i!=c; i=U[i]) {
39            for(int j=L[i]; j!=i; j=L[j]) {
40                U[D[j]] = D[U[j]] = j;
41                ++colSize[col[j]];
42            }
43        }
44        L[R[c]] = R[L[c]] = c;
45    }
46    bool dfs(int idx=0) { // 判斷其中一解版
47        if(R[0] == 0) {
48            resSize = idx;
49            return true;
50        }
51        int c = R[0];
52        for(int i=R[0]; i; i=R[i]) {
53            if(colSize[i] < colSize[c]) c = i;
54        }
55        remove(c);
56        for(int i=D[c]; i!=c; i=D[i]) {
57            result[idx] = row[i];
58            for(int j=R[i]; j!=i; j=R[j])
59                remove(col[j]);
60            if(dfs(idx+1)) return true;

```

```

61     for(int j=L[i]; j!=i; j=L[j])
62         recover(col[j]);
63     }
64     recover(c);
65     return false;
66 }
67 void dfs(int idx=0) { // 判斷最小 dfs
68     depth 版
69     if(R[0] == 0) {
70         resSize = min(resSize, idx); //
71         注意init值
72         return;
73     }
74     int c = R[0];
75     for(int i=R[0]; i; i=R[i]) {
76         if(colSize[i] < colSize[c]) c = i;
77     }
78     remove(c);
79     for(int i=D[c]; i!=c; i=D[i]) {
80         for(int j=R[i]; j!=i; j=R[j])
81             remove(col[j]);
82         dfs(idx+1);
83         for(int j=L[i]; j!=i; j=L[j])
84             recover(col[j]);
85     }
86     recover(c);
87 }
88 };

```

3.16 LCA 倍增法

```

1 //倍增法預處理O(nlogn), 查詢O(logn),
2 //利用lca找樹上任兩點距離
3 #define maxn 100005
4 struct Edge { int u, v, w; };
5 vector<vector<Edge>> G; // tree
6 int fa[maxn][31]; //fa[u][i] -> u的第2^i個祖先
7 long long dis[maxn][31];
8 int dep[maxn]; //深度
9 void dfs(int u, int p) { //預處理fa
10     fa[u][0] = p; //因為u的第2^0 = 1的祖先就是p
11     dep[u] = dep[p] + 1;
12     //第2^i的祖先是(第2^(i-1)個祖先)的
13     //第2^(i-1)的祖先
14     //ex: 第8個祖先是 (第4個祖先)的第4個祖先
15     for (int i = 1; i < 31; ++i) {
16         fa[u][i] = fa[fa[u][i-1]][i-1];
17         dis[u][i] = dis[fa[u][i-1]][i-1]
18             + dis[u][i-1];
19     }
20     //遍歷子節點
21     for (Edge& edge: G[u]) {
22         if (edge.v == p) continue;
23         dis[edge.v][0] = edge.w;
24         dfs(edge.v, u);
25     }
26 long long lca(int x, int y) {
27     //此函數是找lca同時計算x、y的距離 -> dis(x,
28     lca) + dis(lca, y)
29     //讓y比x深
30     if (dep[x] > dep[y])
31         swap(x, y);
32     int deltaDep = dep[y] - dep[x];
33     long long res = 0;
34     //讓y與x在同一個深度
35     for (int i = 0; deltaDep != 0; ++i,
36         deltaDep >>= 1)
37         if (deltaDep & 1)
38             res += dis[y][i], y = fa[y][i];
39     if (y == x) //x = y -> x、y彼此是彼此的祖先
40         return res;
41     //往上找, 一起跳, 但x、y不能重疊
42     for (int i = 30; i >= 0 && y != x; --i) {
43         if (fa[x][i] != fa[y][i]) {
44             res += dis[x][i] + dis[y][i];
45             x = fa[x][i];

```

```

44         y = fa[y][i];
45     }
46 }
47 //最後發現不能跳了, 此時x的第2^0 =
48 1個祖先(或說y的第2^0 =
49 1的祖先)即為x、y的lca
50 res += dis[x][0] + dis[y][0];
51 return res;
52 }
53 int main() {
54     int n, q;
55     while (~scanf("%d", &n) && n) {
56         int v, w;
57         G.assign(n + 5, vector<Edge>());
58         for (int i = 1; i <= n - 1; ++i) {
59             scanf("%d %d", &v, &w);
60             G[i + 1].push_back({i + 1, v + 1, w});
61             G[v + 1].push_back({v + 1, i + 1, w});
62         }
63         dfs(1, 0);
64         scanf("%d", &q);
65         int u;
66         while (q--) {
67             scanf("%d %d", &u, &v);
68             printf("%lld%c", lca(u + 1, v +
69                 1), (q) ? ' ' : '\n');
70         }
71     }
72 }

```

3.17 LCA 樹鍊剖分

```

1 #define maxn 5005
2 //LCA, 用來練習樹鍊剖分
3 //題意: 給定樹, 找任兩點的中點,
4 //若中點不存在(路徑為even), 就是中間的兩個點
5 int dfn[maxn];
6 int parent[maxn];
7 int depth[maxn];
8 int subtreeSize[maxn];
9 //樹鍊的頂點
10 int top[maxn];
11 //將dfn轉成node編碼
12 int dfnToNode[maxn];
13 //重兒子
14 int hson[maxn];
15 int dfsTime = 1;
16 //tree
17 vector<vector<int>> G;
18 //處理parent、depth、subtreeSize、dfnToNode
19 void dfs1(int u, int p) {
20     parent[u] = p;
21     hson[u] = -1;
22     subtreeSize[u] = 1;
23     for (int v: G[u]) {
24         if (v != p) {
25             depth[v] = depth[u] + 1;
26             dfs1(v, u);
27             subtreeSize[u] += subtreeSize[v];
28             if (hson[u] == -1 ||
29                 subtreeSize[hson[u]] <
30                 subtreeSize[v]) {
31                 hson[u] = v;
32             }
33         }
34     }
35     //實際剖分 <- 參數t是top的意思
36     //t初始應為root本身
37 void dfs2(int u, int t) {
38     top[u] = t;
39     dfn[u] = dfsTime;
40     dfnToNode[dfsTime] = u;
41     ++dfsTime;
42     //葉子點 -> 沒有重兒子
43     if (hson[u] == -1)
44         return;

```

```

44     //優先對重兒子dfs, 才能保證同一重鍊dfn連續
45     dfs2(hson[u], t);
46     for (int v: G[u]) {
47         if (v != parent[u] && v != hson[u])
48             dfs2(v, v);
49     }
50 }
51 //不斷跳鍊, 當跳到同一條鍊時, 深度小的即為LCA
52 //跳鍊時優先鍊頂深度大的跳
53 int LCA(int u, int v) {
54     while (top[u] != top[v]) {
55         if (depth[top[u]] > depth[top[v]])
56             u = parent[top[u]];
57         else
58             v = parent[top[v]];
59     }
60     return (depth[u] > depth[v]) ? v : u;
61 }
62 int getK_parent(int u, int k) {
63     while (k-- && (u != -1))
64         u = parent[u];
65     return u;
66 }
67 int main() {
68     int n;
69     while (scanf("%d", &n) && n) {
70         dfsTime = 1;
71         G.assign(n + 5, vector<int>());
72         int u, v;
73         for (int i = 1; i < n; ++i) {
74             scanf("%d %d", &u, &v);
75             G[u].emplace_back(v);
76             G[v].emplace_back(u);
77         }
78         dfs1(1, -1);
79         dfs2(1, 1);
80         int q;
81         scanf("%d", &q);
82         for (int i = 0; i < q; ++i) {
83             scanf("%d %d", &u, &v);
84             //先得LCA
85             int lca = LCA(u, v);
86             //計算路徑長(經過的邊)
87             int dis = depth[u] + depth[v] - 2
88                 * depth[lca];
89             //讓v比u深或等於
90             if (depth[u] > depth[v])
91                 swap(u, v);
92             if (u == v) {
93                 printf("The fleas meet at
94                     %d.\n", u);
95             }
96             else if (dis % 2 == 0) {
97                 //路徑長是even -> 有中點
98                 printf("The fleas meet at
99                     %d.\n", getK_parent(v,
100                         dis / 2));
101             }
102             else {
103                 //路徑長是odd -> 沒有中點
104                 if (depth[u] == depth[v]) {
105                     int x = getK_parent(u, dis
106                         / 2);
107                     int y = getK_parent(v, dis
108                         / 2);
109                     if (x > y) swap(x, y);
110                     printf("The fleas jump
111                         forever between %d
112                         and %d.\n", x, y);
113                 }
114                 else {
115                     //技巧: 讓深的點v往上dis /
116                     2步 = y,
117                     //這個點的parent設為x
118                     //此時的x、y就是答案要的中點兩點
119                     //主要是往下不好找, 所以改用深的點u
120                     int y = getK_parent(v, dis
121                         / 2);

```

```

112         int x = getK_parent(y, 1);
113         if (x > y) swap(x, y);
114         printf("The fleas jump
           forever between %d
           and %d.\n", x, y);
115     }
116 }
117 }
118 }
119 return 0;
120 }

```

4 DataStructure

4.1 帶權併查集

val[x] 為 x 到 p[x] 的距離 (隨題目變化更改)

merge(u, v, w)
 $u \xrightarrow{w} v$
 pu = pv 時, val[v] - val[u] ≠ w 代表有誤

若 [l, r] 的總和為 w, 則應呼叫 merge(l-1, r, w)

```

1 const int maxn = 2e5 + 10;
2
3 int p[maxn], val[maxn];
4
5 int findP(int x) {
6     if(p[x] == -1) return x;
7     int par = findP(p[x]);
8     val[x] += val[p[x]]; //依題目更新val[x]
9     return p[x] = par;
10 }
11
12 void merge(int u, int v, int w) {
13     int pu = findP(u);
14     int pv = findP(v);
15     if(pu == pv) {
16         // 理論上 val[v]-val[u] == w
17         // 依題目判斷 error 的條件
18         return;
19     }
20     val[pv] = val[u] - val[v] + w;
21     p[pv] = pu;
22 }

```

4.2 Trie

```

1 const int maxc = 26; // 單字字符數
2 const char minc = 'a'; // 首個 ASCII
3
4 struct TrieNode {
5     int cnt;
6     TrieNode* child[maxc];
7
8     TrieNode() {
9         cnt = 0;
10        for(auto& node : child) {
11            node = nullptr;
12        }
13    }
14 };
15
16 struct Trie {
17     TrieNode* root;
18
19     Trie() { root = new TrieNode(); }
20
21     void insert(string word) {
22         TrieNode* cur = root;
23         for(auto& ch : word) {
24             int c = ch - minc;
25             if(!cur->child[c])
26                 cur->child[c] = new TrieNode();

```

```

27         cur = cur->child[c];
28     }
29     cur->cnt++;
30 }
31
32 void remove(string word) {
33     TrieNode* cur = root;
34     for(auto& ch : word) {
35         int c = ch - minc;
36         if(!cur->child[c]) return;
37         cur = cur->child[c];
38     }
39     cur->cnt--;
40 }
41
42 // 字典裡有出現 word
43 bool search(string word, bool prefix=0) {
44     TrieNode* cur = root;
45     for(auto& ch : word) {
46         int c = ch - minc;
47         if(!cur->child[c]) return false;
48     }
49     return cur->cnt || prefix;
50 }
51
52 // 字典裡有 word 的前綴為 prefix
53 bool startsWith(string prefix) {
54     return search(prefix, true);
55 }
56 };

```

4.3 AC Trie

```

1 const int maxn = 1e4 + 10; // 單字字數
2 const int maxl = 50 + 10; // 單字字長
3 const int maxc = 128; // 單字字符數
4 const char minc = ' '; // 首個 ASCII
5
6 int trie[maxn*maxl][maxc]; // 原字典樹
7 int val[maxn*maxl]; // 結尾(單字編號)
8 int cnt[maxn*maxl]; // 結尾(重複個數)
9 int fail[maxn*maxl]; // failure link
10 bool vis[maxn*maxl]; // 同單字不重複
11
12 struct ACTrie {
13     int seq, root;
14
15     ACTrie() {
16         seq = 0;
17         root = newNode();
18     }
19
20     int newNode() {
21         for(int i=0; i<maxc; trie[seq][i++]=0;
22         val[seq] = cnt[seq] = fail[seq] = 0;
23         return seq++;
24     }
25
26     void insert(char* s, int wordId=0) {
27         int p = root;
28         for(; *s; s++) {
29             int c = *s - minc;
30             if(!trie[p][c]) trie[p][c] = newNode();
31             p = trie[p][c];
32         }
33         val[p] = wordId;
34         cnt[p]++;
35     }
36
37     void build() {
38         queue<int> q({root});
39         while(!q.empty()) {
40             int p = q.front();
41             q.pop();
42             for(int i=0; i<maxc; i++) {
43                 int& t = trie[p][i];
44                 if(t) {

```

```

45                 fail[t] = p?trie[fail[p]][i]:root;
46                 q.push(t);
47             } else {
48                 t = trie[fail[p]][i];
49             }
50         }
51     }
52 }
53
54 // 要存 wordId 才要 vec
55 // 同單字重複match要把所有vis取消掉
56 int match(char* s, vector<int>& vec) {
57     int res = 0;
58     memset(vis, 0, sizeof(vis));
59     for(int p=root; *s; s++) {
60         p = trie[p][*s-minc];
61         for(int k=p; k && !vis[k]; k=fail[k]) {
62             vis[k] = true;
63             res += cnt[k];
64             if(cnt[k]) vec.push_back(val[k]);
65         }
66     }
67     return res; // 匹配到的單字量
68 }
69 };
70
71 ACTrie ac; // 建構, 初始化
72 ac.insert(s); // 加字典單字
73 // 加完字典後
74 ac.build(); // !!! 建 failure link !!!
75 ac.match(s); // 多模式匹配(傳入vec可以存編號)

```

4.4 線段樹 1D

```

1 #define MAXN 1000
2 int data[MAXN]; //原數據
3 int st[4 * MAXN]; //線段樹
4 int tag[4 * MAXN]; //懶標
5 inline int pull(int l, int r) {
6     // 隨題目改變sum、max、min
7     // l、r是左右樹的index
8     return st[l] + st[r];
9 }
10 void build(int l, int r, int i) {
11     // 在[l, r]區間建樹, 目前根的index為i
12     if (l == r) {
13         st[i] = data[l];
14         return;
15     }
16     int mid = l + ((r - l) >> 1);
17     build(l, mid, i * 2);
18     build(mid + 1, r, i * 2 + 1);
19     st[i] = pull(i * 2, i * 2 + 1);
20 }
21 int qry(int ql, int qr, int l, int r, int i){
22     // [ql,qr]是查詢區間, [l,r]是當前節點包含的區間
23     if (ql <= l && r <= qr)
24         return st[i];
25     int mid = l + ((r - l) >> 1);
26     if (tag[i]) {
27         //如果當前懶標有值則更新左右節點
28         st[i * 2] += tag[i] * (mid - l + 1);
29         st[i * 2 + 1] += tag[i] * (r - mid);
30         tag[i * 2] += tag[i];
31         tag[i * 2 + 1] += tag[i];
32         tag[i] = 0;
33     }
34     int sum = 0;
35     if (ql <= mid)
36         sum+=query(ql, qr, l, mid, i * 2);
37     if (qr > mid)
38         sum+=query(ql, qr, mid+1, r, i*2+1);
39     return sum;
40 }
41 void update(
42     int ql,int qr,int l,int r,int i,int c) {
43     // [ql,qr]是查詢區間, [l,r]是當前節點包含的區間

```

```

44 // c是變化量
45 if (ql <= l && r <= qr) {
46     st[i] += (r - l + 1) * c;
47     //求和,此需乘上區間長度
48     tag[i] += c;
49     return;
50 }
51 int mid = l + ((r - l) >> 1);
52 if (tag[i] && l != r) {
53     //如果當前懶標有值則更新左右節點
54     st[i * 2] += tag[i] * (mid - l + 1);
55     st[i * 2 + 1] += tag[i] * (r - mid);
56     tag[i * 2] += tag[i]; //下傳懶標至左節點
57     tag[i * 2 + 1] += tag[i]; //下傳懶標至右節點
58     tag[i] = 0;
59 }
60 if (ql <= mid) update(ql, qr, l, mid, i
    * 2, c);
61 if (qr > mid) update(ql, qr, mid + 1, r,
    i * 2 + 1, c);
62 st[i] = pull(i * 2, i * 2 + 1);
63 }
64 //如果是直接改值而不是加值, query與update中的tag與st
65 //改值從 += 改成 =

```

4.5 線段樹 2D

```

1 //純2D segment tree 區間查詢單點修改最大最小值
2 #define maxn 2005 //500 * 4 + 5
3 int maxST[maxn][maxn], minST[maxn][maxn];
4 int N;
5 void modifyY(int index, int l, int r, int
    val, int yPos, int xIndex, bool
    xIsLeaf) {
6     if (l == r) {
7         if (xIsLeaf) {
8             maxST[xIndex][index] =
                minST[xIndex][index] = val;
9             return;
10        }
11        maxST[xIndex][index] =
            max(maxST[xIndex * 2][index],
                maxST[xIndex * 2 + 1][index]);
12        minST[xIndex][index] =
            min(minST[xIndex * 2][index],
                minST[xIndex * 2 + 1][index]);
13    }
14    else {
15        int mid = (l + r) / 2;
16        if (yPos <= mid)
17            modifyY(index * 2, l, mid, val,
                yPos, xIndex, xIsLeaf);
18        else
19            modifyY(index * 2 + 1, mid + 1,
                r, val, yPos, xIndex,
                xIsLeaf);
20    }
21    maxST[xIndex][index] =
        max(maxST[xIndex][index * 2],
            maxST[xIndex][index * 2 + 1]);
22    minST[xIndex][index] =
        min(minST[xIndex][index * 2],
            minST[xIndex][index * 2 + 1]);
23    }
24 }
25 void modifyX(int index, int l, int r, int
    val, int xPos, int yPos) {
26     if (l == r) {
27         modifyY(1, 1, N, val, yPos, index,
            true);
28     }
29     else {
30         int mid = (l + r) / 2;
31         if (xPos <= mid)
32             modifyX(index * 2, l, mid, val,
                xPos, yPos);
33         else

```

```

34         modifyX(index * 2 + 1, mid + 1,
            r, val, xPos, yPos);
35         modifyY(1, 1, N, val, yPos, index,
            false);
36     }
37 }
38 void queryY(int index, int l, int r, int
    yql, int yqr, int xIndex, int& vmax,
    int& vmin) {
39     if (yql <= l && r <= yqr) {
40         vmax = max(vmax,
            maxST[xIndex][index]);
41         vmin = min(vmin,
            minST[xIndex][index]);
42     }
43     else
44     {
45         int mid = (l + r) / 2;
46         if (yql <= mid)
47             queryY(index * 2, l, mid, yql,
                yqr, xIndex, vmax, vmin);
48         if (mid < yqr)
49             queryY(index * 2 + 1, mid + 1, r,
                yql, yqr, xIndex, vmax,
                vmin);
50     }
51 }
52 void queryX(int index, int l, int r, int
    xql, int xqr, int yql, int yqr, int&
    vmax, int& vmin) {
53     if (xql <= l && r <= xqr) {
54         queryY(1, 1, N, yql, yqr, index,
            vmax, vmin);
55     }
56     else {
57         int mid = (l + r) / 2;
58         if (xql <= mid)
59             queryX(index * 2, l, mid, xql,
                xqr, yql, yqr, vmax, vmin);
60         if (mid < xqr)
61             queryX(index * 2 + 1, mid + 1, r,
                xql, xqr, yql, yqr, vmax,
                vmin);
62     }
63 }
64 int main() {
65     while (scanf("%d", &N) != EOF) {
66         int val;
67         for (int i = 1; i <= N; ++i) {
68             for (int j = 1; j <= N; ++j) {
69                 scanf("%d", &val);
70                 modifyX(1, 1, N, val, i, j);
71             }
72         }
73         int q;
74         int vmax, vmin;
75         int xql, xqr, yql, yqr;
76         char op;
77         scanf("%d", &q);
78         while (q--) {
79             getchar(); //for \n
80             scanf("%c", &op);
81             if (op == 'q') {
82                 scanf("%d %d %d %d", &xql,
                    &yql, &xqr, &yqr);
83                 vmax = -0x3f3f3f3f;
84                 vmin = 0x3f3f3f3f;
85                 queryX(1, 1, N, xql, xqr,
                    yql, yqr, vmax, vmin);
86                 printf("%d %d\n", vmax, vmin);
87             }
88             else {
89                 scanf("%d %d %d", &xql, &yql,
                    &val);
90                 modifyX(1, 1, N, val, xql,
                    yql);
91             }
92         }

```

4.6 權值線段樹

```

1 //權值線段樹 + 離散化 解決區間第k小問題
2 #define maxn 30005
3 int nums[maxn];
4 int getArr[maxn];
5 int id[maxn];
6 int st[maxn << 2];
7 void update(int index, int l, int r, int qx){
8     if (l == r) {
9         ++st[index];
10        return;
11    }
12    int mid = (l + r) / 2;
13    if (qx <= mid)
14        update(index * 2, l, mid, qx);
15    else
16        update(index * 2 + 1, mid + 1, r, qx);
17    st[index] = st[index * 2] + st[index * 2
        + 1];
18 }
19 //找區間第k個小的
20 int query(int index, int l, int r, int k) {
21     if (l == r) return id[l];
22     int mid = (l + r) / 2;
23     //k比左子樹小
24     if (k <= st[index * 2])
25         return query(index * 2, l, mid, k);
26     else
27         return query(index * 2 + 1, mid + 1,
            r, k - st[index * 2]);
28 }
29 int main() {
30     int t;
31     cin >> t;
32     bool first = true;
33     while (t--) {
34         if (first) first = false;
35         else puts("");
36         memset(st, 0, sizeof(st));
37         int m, n;
38         cin >> m >> n;
39         for (int i = 1; i <= m; ++i) {
40             cin >> nums[i];
41             id[i] = nums[i];
42         }
43         for (int i = 0; i < n; ++i)
44             cin >> getArr[i];
45         //離散化
46         //防止m == 0
47         if (m) sort(id + 1, id + m + 1);
48         int stSize = unique(id + 1, id + m +
            1) - (id + 1);
49         for (int i = 1; i <= m; ++i) {
50             nums[i] = lower_bound(id + 1, id
                + stSize + 1, nums[i]) - id;
51         }
52         int addCount = 0;
53         int getCount = 0;
54         int k = 1;
55         while (getCount < n) {
56             if (getArr[getCount] == addCount)
57                 printf("%d\n", query(1, 1,
                    stSize, k));
58                 ++k;
59                 ++getCount;
60         }
61         else {
62             update(1, 1, stSize,
                nums[addCount + 1]);
63             ++addCount;
64         }

```

```

65 }
66 }
67 }

```

4.7 單調隊列

```

1 //單調隊列
2 "如果一個選手比你小還比你強，你就可以退役了。"
3
4 example:
5 給出一個長度為 n 的數組，
6 輸出每 k 個連續的數中的最大值和最小值。
7
8 #define maxn 1000100
9 int q[maxn], a[maxn];
10 int n, k;
11 //得到這個隊列裡的最小值，直接找到最後的就行了
12 void getmin() {
13     int head=0,tail=0;
14     for(int i=1;i<k;i++) {
15         while(head<=tail&&[q[tail]]>=a[i])
16             tail--;
17         q[++tail]=i;
18     }
19     for(int i=k;i<=n;i++) {
20         while(head<=tail&&[q[tail]]>=a[i])
21             tail--;
22         q[++tail]=i;
23         while(q[head]<=i-k) head++;
24         cout<<a[q[head]]<<" ";
25     }
26     cout<<endl;
27 }
28 //和上面同理
29 void getmax() {
30     int head=0,tail=0;
31     for(int i=1;i<k;i++) {
32         while(head<=tail&&[q[tail]]<=a[i])tail--;
33         q[++tail]=i;
34     }
35     for(int i=k;i<=n;i++) {
36         while(head<=tail&&[q[tail]]<=a[i])tail--;
37         q[++tail]=i;
38         while(q[head]<=i-k) head++;
39         cout<<a[q[head]]<<" ";
40     }
41     cout<<endl;
42 }
43 int main(){
44     cin>>n>>k; //每k個連續的數
45     for(int i=1;i<=n;i++) cin>>a[i];
46     getmin();
47     getmax();
48 }

```

5 Geometry

5.1 公式

1. Circle and Line

點 $P(x_0, y_0)$

到直線 $L: ax + by + c = 0$ 的距離

$$d(P, L) = \frac{|ax_0 + by_0 + c|}{\sqrt{a^2 + b^2}}$$

兩平行直線 $L_1: ax + by + c_1 = 0$

與 $L_2: ax + by + c_2 = 0$ 的距離

$$d(L_1, L_2) = \frac{|c_1 - c_2|}{\sqrt{a^2 + b^2}}$$

2. Triangle

設三角形頂點為 $A(x_1, y_1), B(x_2, y_2), C(x_3, y_3)$

點 A, B, C 的對邊長分別為 a, b, c

三角形面積為 Δ

重心為 (G_x, G_y) ，內心為 (I_x, I_y) ，

外心為 (O_x, O_y) 和垂心為 (H_x, H_y)

$$\Delta = \frac{1}{2} \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}$$

$$G_x = \frac{1}{3}(x_1 + x_2 + x_3)$$

$$G_y = \frac{1}{3}(y_1 + y_2 + y_3)$$

$$I_x = \frac{ax_1 + bx_2 + cx_3}{a + b + c}$$

$$I_y = \frac{ay_1 + by_2 + cy_3}{a + b + c}$$

$$O_x = \frac{1}{4\Delta} \begin{vmatrix} x_1^2 + y_1^2 & x_2^2 + y_2^2 & x_3^2 + y_3^2 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{vmatrix}$$

$$O_y = \frac{1}{4\Delta} \begin{vmatrix} x_1 & x_1^2 + y_1^2 & 1 \\ x_2 & x_2^2 + y_2^2 & 1 \\ x_3 & x_3^2 + y_3^2 & 1 \end{vmatrix}$$

$$H_x = -\frac{1}{2\Delta} \begin{vmatrix} x_2x_3 + y_2y_3 & y_1 & 1 \\ x_1x_3 + y_1y_3 & y_2 & 1 \\ x_1x_2 + y_1y_2 & y_3 & 1 \end{vmatrix}$$

$$H_y = -\frac{1}{2\Delta} \begin{vmatrix} x_1 & x_2x_3 + y_2y_3 & 1 \\ x_2 & x_1x_3 + y_1y_3 & 1 \\ x_3 & x_1x_2 + y_1y_2 & 1 \end{vmatrix}$$

任意三角形，重心、外心、垂心共線

$$G_x = \frac{2}{3}O_x + \frac{1}{3}H_x$$

$$G_y = \frac{2}{3}O_y + \frac{1}{3}H_y$$

3. Quadrilateral

任意凸四邊形 $ABCD$ 的四邊長分別為 a, b, c, d

且已知 $\angle A + \angle C$ ，則四邊形 $ABCD$ 的面積為

$$\sqrt{(s-a)(s-b)(s-c)(s-d)} - \Delta$$

where

$$s = \frac{a + b + c + d}{2}$$

$$\Delta = abcd \cos^2 \left(\frac{A + C}{2} \right)$$

特例：若 $ABCD$ 為圓內接四邊形，則 $\Delta = 0$

若只知道其中一角，則可用餘弦定理

$$c^2 = a^2 + b^2 - 2ab \cos(\angle C)$$

求出對角線長，再用海龍計算兩個三角形面積即可。

5.2 Template

Predefined Variables

```

1 using DBL = double;
2 using Tp = DBL; // 存點的型態
3
4 const DBL pi = acos(-1);
5 const DBL eps = 1e-9;
6 const Tp inf = 1e30;
7 const int maxn = 5e4 + 10;

```

Vector、Point

```

1 struct Vector {
2     Tp x, y;
3     Vector(Tp x=0, Tp y=0): x(x), y(y) {}
4     DBL length();
5 };
6
7 using Point = Vector;
8 using Polygon = vector<Point>;
9
10 Vector operator+(Vector a, Vector b) {
11     return Vector(a.x+b.x, a.y+b.y);
12 }
13
14 Vector operator-(Vector a, Vector b) {
15     return Vector(a.x-b.x, a.y-b.y);
16 }
17
18 Vector operator*(Vector a, DBL b) {
19     return Vector(a.x*b, a.y*b);
20 }
21
22 Vector operator/(Vector a, DBL b) {
23     return Vector(a.x/b, a.y/b);
24 }
25
26 Tp dot(Vector a, Vector b) {
27     return a.x*b.x + a.y*b.y;
28 }
29
30 Tp cross(Vector a, Vector b) {
31     return a.x*b.y - a.y*b.x;
32 }
33
34 DBL Vector::length() {
35     return sqrt(dot(*this, *this));
36 }
37
38 Vector unit_normal_vector(Vector v) {
39     DBL len = v.length();
40     return Vector(-v.y/len, v.x/len);
41 }

```

Line

```

1 struct Line {
2     Point p;
3     Vector v;
4     DBL ang;
5     Line(Point _p={}, Vector _v={}) {
6         p = _p;
7         v = _v;
8         ang = atan2(v.y, v.x);
9     }
10     bool operator<(const Line& l) const {
11         return ang < l.ang;
12     }
13 };

```

Segment

```

1 struct Segment {
2     Point s, e;
3     Vector v;
4     Segment(): s(0, 0), e(0, 0), v(0, 0) {}

```



```

5 Segment(Point s, Point e): s(s), e(e) {
6     v = e - s;
7 }
8 DBL length() { return v.length(); }
9 };

```

Circle

```

1 struct Circle {
2     Point o;
3     DBL r;
4     Circle(): o({0, 0}), r(0) {}
5     Circle(Point o, DBL r=0): o(o), r(r) {}
6     Circle(Point a, Point b) { // ab 直徑
7         o = (a + b) / 2;
8         r = dis(o, a);
9     }
10    Circle(Point a, Point b, Point c) {
11        Vector u = b-a, v = c-a;
12        DBL c1=dot(u, a+b)/2, c2=dot(v, a+c)/2;
13        DBL dx=c1*v.y-c2*u.y, dy=u.x*c2-v.x*c1;
14        o = Point(dx, dy) / cross(u, v);
15        r = dis(o, a);
16    }
17    bool cover(Point p) {
18        return dis(o, p) <= r;
19    }
20 };

```

5.3 旋轉卡尺

```

1 // 回傳凸包內最遠兩點的距離 ^2
2 int longest_distance(Polygon& p) {
3     auto test = [&](Line l, Point a, Point b) {
4         return cross(l.v, a-l.p) <= cross(l.v, b-l.p);
5     };
6     if(p.size() <= 2) {
7         return cross(p[0]-p[1], p[0]-p[1]);
8     }
9     int mx = 0, n = p.size();
10    for(int i=0, j=1; i<n; i++) {
11        Line l(p[i], p[(i+1)%n] - p[i]);
12        for(; test(l, p[j], p[(j+1)%n]); j=(j+1)%n);
13        mx = max({
14            mx,
15            dot(p[(i+1)%n]-p[j], p[(i+1)%n]-p[j]),
16            dot(p[i]-p[j], p[i]-p[j])
17        });
18    }
19    return mx;
20 }

```

5.4 半平面相交

Template

```

1 using DBL = double;
2 using Tp = DBL; // 存點的型態
3 const int maxn = 5e4 + 10;
4 const DBL eps = 1e-9;
5 struct Vector;
6 using Point = Vector;
7 using Polygon = vector<Point>;
8 Vector operator+(Vector, Vector);
9 Vector operator-(Vector, Vector);
10 Vector operator*(Vector, DBL);
11 Tp cross(Vector, Vector);
12 struct Line;
13 Point intersection(Line, Line);
14 int dcmp(DBL, DBL); // 不見得會用到

```

Halfplane Intersection

```

1 // Return: 能形成半平面交的凸包邊界點
2 Polygon halfplaneIntersect(vector<Line>& nar) {
3     sort(nar.begin(), nar.end());
4     // p 是否在 l 的左半平面
5     auto lft = [&](Point p, Line l) {
6         return dcmp(cross(l.v, p-l.p)) > 0;
7     };
8
9     int ql = 0, qr = 0;
10    Line L[maxn] = {nar[0]};
11    Point P[maxn];
12
13    for(int i=1; i<nar.size(); i++) {
14        for(; ql<qr&&!lft(P[qr-1], nar[i]); qr--);
15        for(; ql<qr&&!lft(P[ql], nar[i]); ql++);
16        L[++qr] = nar[i];
17        if(dcmp(cross(L[qr].v, L[qr-1].v))==0) {
18            if(lft(nar[i].p, L[-qr])) L[qr]=nar[i];
19        }
20        if(ql < qr)
21            P[qr-1] = intersection(L[qr-1], L[qr]);
22    }
23    for(; ql<qr && !lft(P[qr-1], L[ql]); qr--);
24    if(qr-ql <= 1) return {};
25    P[qr] = intersection(L[qr], L[ql]);
26    return Polygon(P+ql, P+qr+1);
27 }

```

5.5 Polygon

```

1 // 判斷點 (point) 是否在凸包 (p) 內
2 bool pointInConvex(Polygon& p, Point point) {
3     // 根據 Tp 型態來寫，沒浮點數不用 dblcmp
4     auto dblcmp = [&](DBL v) { return (v>0)-(v<0); };
5     // 不包含線上，改 '>=' 為 '>'
6     auto test = [&](Point& p0, Point& p1) {
7         return dblcmp(cross(p1-p0, point-p0))>=0;
8     };
9     p.push_back(p[0]);
10    for(int i=1; i<p.size(); i++) {
11        if(!test(p[i-1], p[i])) {
12            p.pop_back();
13            return false;
14        }
15    }
16    p.pop_back();
17    return true;
18 }
19
20 // 計算簡單多邊形的面積
21 // ! p 為排序過的點 !
22 DBL polygonArea(Polygon& p) {
23     DBL sum = 0;
24     for(int i=0, n=p.size(); i<n; i++)
25         sum += cross(p[i], p[(i+1)%n]);
26     return abs(sum) / 2.0;
27 }

```

5.6 凸包

- Tp 為 Point 裡 x 和 y 的型態
- struct Point 需要加入並另外計算的 variables:
 1. ang, 該點與基準點的 atan2 值
 2. d2, 該點與基準點的 (距離)²
- 注意計算 d2 的型態範圍限制

Template

```

1 using DBL = double;
2 using Tp = long long; // 存點的型態
3 const DBL eps = 1e-9;
4 const Tp inf = 1e9; // 座標極大值
5 struct Vector;
6 using Point = Vector;
7 using Polygon = vector<Point>;

```

```

8 Vector operator-(Vector, Vector);
9 Tp cross(Vector, Vector);
10 int dcmp(DBL, DBL);

Convex Hull

1 Polygon convex_hull(Point* p, int n) {
2     auto rmv = [&](Point a, Point b, Point c) {
3         return cross(b-a, c-b) <= 0; // 非浮點數
4         return dcmp(cross(b-a, c-b)) <= 0;
5     };
6
7     // 選最下裡最左的當基準點，可在輸入時計算
8     Tp lx = inf, ly = inf;
9     for(int i=0; i<n; i++) {
10        if(p[i].y<ly || (p[i].y==ly&&p[i].x<lx)){
11            lx = p[i].x, ly = p[i].y;
12        }
13    }
14
15    for(int i=0; i<n; i++) {
16        p[i].ang=atan2(p[i].y-ly, p[i].x-lx);
17        p[i].d2 = (p[i].x-lx)*(p[i].x-lx) +
18                (p[i].y-ly)*(p[i].y-ly);
19    }
20    sort(p, p+n, [&](Point& a, Point& b) {
21        if(dcmp(a.ang, b.ang))
22            return a.ang < b.ang;
23        return a.d2 < b.d2;
24    });
25
26    int m = 1; // stack size
27    Point st[n] = {p[n] = p[0]};
28    for(int i=1; i<n; i++) {
29        for(; m>1&&rmv(st[m-2], st[m-1], p[i]); m--);
30        st[m++] = p[i];
31    }
32    return Polygon(st, st+m-1);
33 }

```

5.7 最小圓覆蓋

```

1 vector<Point> p(3); // 在圖上的點
2 Circle MEC(vector<Point>& v, int n, int d=0) {
3     Circle mec;
4     if(d == 1) mec = Circle(p[0]);
5     if(d == 2) mec = Circle(p[0], p[1]);
6     if(d == 3) return Circle(p[0], p[1], p[2]);
7     for(int i=0; i<n; i++) {
8         if(mec.cover(v[i])) continue;
9         p[d] = v[i];
10        mec = MEC(v, i, d+1);
11    }
12    return mec;
13 }

```

5.8 交點、距離

```

1 int dcmp(DBL a, DBL b=0.0) {
2     if(abs(a-b) < eps) return 0;
3     return a<b ? -1 : 1;
4 }
5 bool hasIntersection(Point p, Segment s) {
6     if(dcmp(cross(s.s-p, s.e-p))) return false;
7     return dcmp(dot(s.s-p, s.e-p)) <= 0;
8 }
9 bool hasIntersection(Point p, Line l) {
10    return dcmp(cross(p-l.p, l.v)) == 0;
11 }
12 bool hasIntersection(Segment a, Segment b) {
13     // 判斷在 X 軸 Y 軸的投影是否相交
14     auto intr1D = [&](DBL w, DBL x, DBL y, DBL z) {
15         if(w > x) swap(w, x);
16         if(y > z) swap(y, z);
17         return dcmp(max(w, y), min(x, z)) <= 0;
18     };
19 }

```

```
18 };
19
20 DBL a1 = cross(a.v, b.s-a.s);
21 DBL a2 = cross(a.v, b.e-a.s);
22 DBL b1 = cross(b.v, a.s-b.s);
23 DBL b2 = cross(b.v, a.e-b.s);
24
25 return intr1D(a.s.x, a.e.x, b.s.x, b.e.x)
26     && intr1D(a.s.y, a.e.y, b.s.y, b.e.y)
27     && dcmp(a1) * dcmp(a2) <= 0
28     && dcmp(b1) * dcmp(b2) <= 0;
29 }
30 Point intersection(Segment a, Segment b) {
31     Vector v = b.s - a.s;
32     DBL c1 = cross(a.v, b.v);
33     DBL c2 = cross(v, b.v);
34     DBL c3 = cross(v, a.v);
35
36     if(dcmp(c1) < 0) c1=-c1, c2=-c2, c3=-c3;
37     if(dcmp(c1) && dcmp(c2)>=0 && dcmp(c3)>=0
38         && dcmp(c1, c2)>=0 && dcmp(c1, c3)>=0)
39         return a.s + (a.v * (c2 / c1));
40     return Point(inf, inf); // a 和 b 共線
41 }
42 Point intersection(Line a, Line b) {
43     // cross(a.v, b.v) == 0 時平行
44     Vector u = a.p - b.p;
45     DBL t = 1.0*cross(b.v, u)/cross(a.v, b.v);
46     return a.p + a.v*t;
47 }
48 DBL dis(Point a, Point b) {
49     return sqrt(dot(a-b, a-b));
50 }
51 DBL dis(Point p, Line l) {
52     return abs(cross(p-l.p, l.v))/l.v.length();
53 }
54 DBL dis(Point p, Segment s) {
55     Vector u = p - s.s, v = p - s.e;
56     if(dcmp(dot(s.v, u))<=0) return u.length();
57     if(dcmp(dot(s.v, v))>=0) return v.length();
58     return abs(cross(s.v, u)) / s.length();
59 }
60 DBL dis(Segment a, Segment b) {
61     if(hasIntersection(a, b)) return 0;
62     return min({
63         dis(a.s, b), dis(a.e, b),
64         dis(b.s, a), dis(b.e, a)
65     });
66 }
67 DBL dis(Line a, Line b) {
68     if(dcmp(cross(a.v, b.v)) == 0) return 0;
69     return dis(a.p, b);
70 }
71 Point getPedal(Line l, Point p) {
72     // 返回 p 在 l 上的垂足(投影點)
73     DBL len = dot(p-l.p, l.v) / dot(l.v, l.v);
74     return l.p + l.v * len;
75 }
```

6 DP

6.1 背包

<u>0-1 背包</u>
複雜度： $O(NW)$
已知：第 i 個物品重量為 w_i ，價值 v_i ；背包總容量 W
意義：dp[前 i 個物品][重量] = 最高價值
maxn: 物品數量
maxw: 背包最大容量
<pre>1 int W; 2 int w[maxn], v[maxn]; 3 int dp[maxw];</pre>

```
4
5 memset(dp, 0, sizeof(dp));
6 for(int i=1; i<=n; i++) {
7     for(int j=W; j>=w[i]; j--) {
8         dp[j] = max(dp[j], dp[j-w[i]]+v[i]);
9     }
10 }
```

價值為主的 0-1 背包

複雜度： $O(NV)$

已知：第 i 個物品重量為 w_i ，價值 v_i ；物品最大總價值 V

意義：dp[前 i 個物品][價值] = 最小重量

maxn: 物品數量

maxv: 物品最大總價值

$V = \sum v_i$

```
1 int w[maxn], v[maxn];
2 int dp[maxv];
3
4 memset(dp, 0x3f, sizeof(dp));
5 dp[0] = 0;
6 for(int i=0; i<n; i++) {
7     for(int j=V; j>=v[i]; j--) {
8         dp[j] = min(dp[j], dp[j-v[i]]+w[i]);
9     }
10 }
11
12 int res = 0;
13 for(int val=V; val>=0; val--) {
14     if(dp[val] <= w) {
15         res = val;
16         break;
17     }
18 }
```

完全背包（無限背包）

複雜度： $O(NW)$
已知：第 i 個物品重量為 w_i ，價值 v_i ；背包總容量 W
意義：dp[前 i 個物品][重量] = 最高價值
maxn: 物品數量
maxw: 背包最大容量
<pre>1 int W; 2 int w[maxn], v[maxn]; 3 int dp[maxw]; 4 5 memset(dp, 0, sizeof(dp)); 6 for(int i=1; i<=n; i++) { 7 for(int j=w[i]; j<=W; j++) { 8 dp[j] = max(dp[j], dp[j-w[i]]+v[i]); 9 } 10 }</pre>

多重背包

複雜度： $O(W \sum cnt_i)$
已知：第 i 個物品重量為 w_i ，價值 v_i ，有 cnt_i 個； 背包總容量 W
意義：dp[前 i 個物品][重量] = 最高價值
maxn: 物品數量
maxw: 背包最大容量

```
1 int W;
2 int w[maxn], v[maxn], cnt[maxn];
3 int dp[maxw];
4
5 memset(dp, 0, sizeof(dp));
6 for(int i=1; i<=n; i++)
7     for(int j=W; j>=w[i]; j--)
8         for(int k=1; k*w[i]<=j&&k<=cnt[i]; k++)
9             dp[j] = max(dp[j], dp[j-k*w[i]]+k*v[i]);
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

混合背包（0-1/完全/多重）

複雜度： $O(W \sum cnt_i)$

已知：第 i 個物品重量為 w_i ，價值 v_i ，有 cnt_i 個；
背包總容量 W

意義：dp[前 i 個物品][重量] = 最高價值

maxn: 物品數量

maxw: 背包最大容量

$cnt_i = 0$ 代表無限

```
1 int W;
2 int w[maxn], v[maxn], cnt[maxn];
3 int dp[maxw];
4
5 memset(dp, 0, sizeof(dp));
6 for(int i=1; i<=n; i++) {
7     if(cnt[i]) {
8         for(int j=W; j>=w[i]; j--) {
9             for(int k=1; k*w[i]<=j&&k<=cnt[i]; k++) {
10                 dp[j]=max(dp[j], dp[j-k*w[i]]+k*v[i]);
11             }
12         }
13     } else {
14         for(int j=w[i]; j<=W; j++) {
15             dp[j] = max(dp[j], dp[j-w[i]] + v[i]);
16         }
17     }
18 }
```

<u>二維費用背包</u>
複雜度： $O(NCT)$
已知：第 k 個任務需要花費 c_k 元，耗時 t_k 分鐘； 總經費 C ，總耗時 T
意義：dp[前 k 個任務][花費][耗時] = 最多任務數
maxc: 最大花費
maxt: 最大耗時
<pre>1 int C, T; 2 int c[maxn], t[maxn]; 3 int dp[maxc][maxt]; 4 5 memset(dp, 0, sizeof(dp)); 6 for(int k=1; k<=n; k++) 7 for(int i=C; i>=c[k]; i--) 8 for(int j=T; j>=t[k]; j--) 9 dp[i][j] = max(10 dp[i], dp[i-c[k]][j-t[k]] + 1);</pre>

<u>分組背包</u>
複雜度： $O(W \sum M)$
已知：第 i 組第 j 個物品重量為 w_{ij} ，價值 v_{ij} ； 背包總容量 W ；每組只能取一個
意義：dp[前 i 組物品][重量] = 最高價值
maxn: 物品組數
maxm: 每組物品數
maxw: 背包最大容量

```

1 int W;
2 int dp[maxw];
3 vector<vector<int>> w, v;
4
5 memset(dp, 0, sizeof(dp));
6 for(int i=0; i<n; i++)
7     for(int j=W; j>=0; j--)
8         for(int k=0; k<w[i].size(); k++)
9             if(j >= w[i][k])
10                 dp[j] = max(
11                     dp[j], dp[j-w[i][k]] + v[i][k]);

```

依賴背包

已知： 第 j 個物品在第 i 個物品沒選的情況下不能選

做法： 樹 DP，有爸爸才有小孩。轉化為分組背包。

意義： dp [選物品 i 為根][重量] = 最高價值

過程： 對所有 $u \rightarrow v$, dfs 計算完 v 後更新 u

背包變化

1. 求最大價值的方法總數 cnt

```

1 for(int i=1; i<=n; i++) {
2     for(int j=W; j>=w[i]; j--) {
3         if(dp[j] < dp[j-w[i]]+v[i]) {
4             dp[j] = dp[j-w[i]] + v[i];
5             cnt[j] = cnt[j-w[i]];
6         } else if(dp[j] == dp[j-w[i]]+v[i]) {
7             cnt[j] += cnt[j-w[i]];
8         }
9     }
10 }

```

2. 求最大價值的一組方案 pick

```

1 memset(pick, 0, sizeof(pick));
2 for(int i=1; i<=n; i++) {
3     for(int j=W; j>=w[i]; j--) {
4         if(dp[i][j] < dp[i-1][j-w[i]]+v[i]) {
5             dp[i][j] = dp[i-1][j-w[i]] + v[i];
6             pick[i] = 1;
7         } else {
8             pick[i] = 0;
9         }
10     }
11 }

```

3. 求最大價值的字典序最小的一組方案 pick

```

1 // reverse(item), 要把物品順序倒過來
2 memset(pick, 0, sizeof(pick));
3 for(int i=1; i<=n; i++) {
4     for(int j=W; j>=w[i]; j--) {
5         if(dp[i][j] <= dp[i-1][j-w[i]]+v[i]) {
6             dp[i][j] = dp[i-1][j-w[i]] + v[i];
7             pick[i] = 1;
8         } else {
9             pick[i] = 0;
10        }
11    }
12 }

```

6.2 Deque 最大差距

```

1 /*定義 dp[l][r] 是 l ~ r 時與先手最大差異值
2 轉移式: dp[l][r] = max{a[l] - solve(l + 1,
3     r), a[r] - solve(l, r - 1)}
4 裡面用減的主要是因為求的是相減且會一直換手，
5 所以正負正負...*/
6 #define maxn 3005
7 bool vis[maxn][maxn];

```

```

7 long long dp[maxn][maxn];
8 long long a[maxn];
9 long long solve(int l, int r) {
10     if (l > r) return 0;
11     if (vis[l][r]) return dp[l][r];
12     vis[l][r] = true;
13     long long res = a[l] - solve(l + 1, r);
14     res = max(res, a[r] - solve(l, r - 1));
15     return dp[l][r] = res;
16 }
17 int main() {
18     ...
19     printf("%lld\n", solve(1, n));
20 }

```

6.3 string DP

Edit distance S_1 最少需要經過幾次增、刪或換字變成 S_2

$$dp[i, j] = \begin{cases} i+1, & \text{if } j = -1 \\ j+1, & \text{if } i = -1 \\ dp[i-1, j-1], & \text{if } S_1[i] = S_2[j] \\ \min \begin{cases} dp[i, j-1] \\ dp[i-1, j] \\ dp[i-1, j-1] \end{cases} + 1, & \text{if } S_1[i] \neq S_2[j] \end{cases}$$

Longest Palindromic Subsequence

$$dp[l, r] = \begin{cases} 1 & \text{if } l = r \\ \max\{dp[l+1, r-1], dp[l, r-1]\} & \text{if } S[l] = S[r] \\ \max\{dp[l+1, r], dp[l, r-1]\} & \text{if } S[l] \neq S[r] \end{cases}$$

6.4 LCS 和 LIS

```

1 //LCS 和 LIS 題目轉換
2 LIS 轉成 LCS
3     1. A 為原序列, B=sort(A)
4     2. 對 A,B 做 LCS
5 LCS 轉成 LIS
6     1. A, B 為原本的兩序列
7     2. 最 A 序列作編號轉換, 將轉換規則套用在 B
8     3. 對 B 做 LIS
9     4. 重複的數字在編號轉換時後要變成不同的數字,
10        越早出現的數字要越小
11     5. 如果有數字在 B 裡面而不在 A 裡面,
12        直接忽略這個數字不做轉換即可

```

6.5 樹 DP 有幾個 path 長度為 k

```

1 #define maxn 50005
2 #define maxk 505
3 //dp[u][u.child] 距距離 u 長度 k 的數量
4 long long dp[maxn][maxk];
5 vector<vector<int>> G;
6 int n, k;
7 long long res = 0;
8 void dfs(int u, int p) {
9     //u 自己
10    dp[u][0] = 1;
11    for (int v: G[u]) {
12        if (v == p) continue;
13        dfs(v, u);
14        for (int i = 1; i <= k; ++i) {
15            //子樹 v 距離 i - 1 的等於對於 u 來說距離 i 的
16            dp[u][i] += dp[v][i - 1];
17        }
18    }
19 }
20 //統計在 u 子樹中距離 u 為 k 的數量
21 res += dp[u][k];
22 long long cnt = 0;
23 for (int v: G[u]) {
24     if (v == p) continue; //重點算法
25 }

```

```

26 for (int x = 0; x <= k - 2; ++x) {
27     cnt +=
28         dp[v][x] * (dp[u][k-x-1] - dp[v][k-x-2]);
29 }
30 }
31 res += cnt / 2;
32 }
33 int main() {
34     ...
35     dfs(1, -1);
36     printf("%lld\n", res);
37     return 0;
38 }

```

6.6 Weighted LIS

```

1 #define maxn 200005
2 long long dp[maxn];
3 long long height[maxn];
4 long long B[maxn];
5 long long st[maxn << 2];
6 void update(int p, int index, int l, int r,
7     long long v) {
8     if (l == r) {
9         st[index] = v;
10        return;
11    }
12    int mid = (l + r) >> 1;
13    if (p <= mid)
14        update(p, (index << 1), l, mid, v);
15    else
16        update(p, (index << 1) + 1, mid + 1, r, v);
17    st[index] =
18        max(st[index << 1], st[(index << 1) + 1]);
19 }
20 long long query(int index, int l, int r, int
21     ql, int qr) {
22     if (ql <= l && r <= qr)
23         return st[index];
24     int mid = (l + r) >> 1;
25     long long res = -1;
26     if (ql <= mid)
27         res =
28             max(res, query(index << 1, l, mid, ql, qr));
29     if (mid < qr)
30         res =
31             max(res, query((index << 1) + 1, mid + 1, r, ql, qr));
32 }
33 int main() {
34     int n;
35     scanf("%d", &n);
36     for (int i = 1; i <= n; ++i)
37         scanf("%lld", &height[i]);
38     for (int i = 1; i <= n; ++i)
39         scanf("%lld", &B[i]);
40     long long res = B[1];
41     update(height[1], 1, 1, n, B[1]);
42     for (int i = 2; i <= n; ++i) {
43         long long temp;
44         if (height[i] - 1 >= 1)
45             temp =
46                 B[i] + query(1, 1, n, 1, height[i] - 1);
47         else
48             temp = B[i];
49         update(height[i], 1, 1, n, temp);
50         res = max(res, temp);
51     }
52     printf("%lld\n", res);
53     return 0;
54 }

```