

## 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	Pisano Period	4
2.9	矩陣快速幂	4
3	algorithm	4
3.1	JosephusProblem	4
3.2	二分搜	4
3.3	三分搜	4
3.4	dinic	4
3.5	SCC Tarjan	5
3.6	BCC 邊	5
3.7	BCC 點	5
3.8	ArticulationPoints Tarjan	5
3.9	最小樹狀圖	6
3.10	KM	6
3.11	二分圖最大匹配	6
3.12	差分	6
3.13	MCMF	6
3.14	LCA 倍增法	7
3.15	LCA 樹鍊剖分	7
4	DataStructure	8
4.1	帶權併查集	8
4.2	Trie	8
4.3	AC Trie	8
4.4	線段樹 1D	8
4.5	線段樹 2D	9
4.6	單調隊列	9
5	Geometry	9
5.1	公式	9
5.2	Template	10
5.3	旋轉卡尺	10
5.4	半平面相交	10
5.5	Polygon	11
5.6	凸包	11
5.7	最小圓覆蓋	11
5.8	交點、距離	11
6	DP	11
6.1	背包	11
6.2	Deque 最大差距	12
6.3	string DP	13
6.4	LCS 和 LIS	13
6.5	樹 DP 有幾個 path 長度為 k	13

## 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: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

$$\begin{bmatrix} f_{n-1} & f_n \\ f_n & f_{n+1} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} f_n & f_{n+1} \\ f_{n+1} & f_{n+2} \end{bmatrix}$$
$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^p = \begin{bmatrix} f_{n+p} & f_{n+p+1} \end{bmatrix}, p \in \mathbb{N}$$

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

#### 5. Pick's Theorem

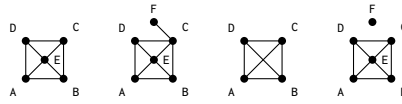
給定頂點座標均是整點（或正方形格子點）的簡單多邊形，其面積  $A$  和內部格點數目  $i$ 、邊上格點數目  $b$  的關係為

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

#### 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)  $\times$ , (2)  $\circ$ ; (3)  $\times$ ,  $\overline{AC}$  與  $\overline{BD}$  相交; (4)  $\times$ , 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

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

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

```

1 #include <stdio>
2 #include <vector>
3 using namespace std;
4 using ull = unsigned long long;
5 #define maxn 1005
6
7 /*
8  Pisano Period + 快速幂 + mod
9  Pisano Period:
10  費氏數列在mod n的情況下會有循環週期，
11  且週期的結束判斷會在
12  fib[i - 1] == 0 && fib[i] == 1時，
13  此時循環週期長度是i - 1
14

```

所以這題是在找出循環週期後，  
用快速幂並 $\text{mod}(\text{循環週期長度})$ 即可AC(快速幂記得 $\text{mod}$ )，  
此外fib要 $\text{mod } n$ ，也要找週期，所以用預處理的方式列表

```

18  Pisano period可證一個週期的長度會在 $[n, n^2]$ 之間
19  很可惜，會爆 int fib[maxn][maxn * maxn];
20  改用vector
21  */
22  vector<int> fib[maxn];
23  int period[maxn];
24
25  int qpow(int a, ull b, int mod) {
26      if (b == 0) return a;
27      long long res = 1;
28      while (b) {
29          if (b & 1)
30              res = ((a % mod) * (res % mod)) % mod;
31          a = ((a % mod) * (a % mod)) % mod;
32          b >>= 1;
33      }
34      return res;
35  }
36
37
38  int main() {
39      int t, n;
40      ull a, b;
41
42      //注意：這裡沒算mod 1的循環長度，
43      //因為mod 1都等於0，沒有週期
44      for (int i = 2; i < maxn; ++i) {
45          fib[i].emplace_back(0);
46          fib[i].emplace_back(1);

```

```

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

```

## 2.9 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>=1; n>>=1) {
33         if(n & 1) res = res * M;
34         M = M * M;
35     }
36     return res;
37 }

```

## 3 algorithm

### 3.1 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) res += n; // mod n
35     else res += res / (k - 1); // 还原位置
36     return res;
37 }

```

## 3.2 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     while (l < r) {
6         int mid = (l + r) >> 1;
7         if (check(mid)) r = mid;
8         else l = mid + 1;
9     }
10    return l;
11 }
12
13 //ooooov..... 即答案右邊界，符合條件最大的
14 int bsearch(int l, int r) {
15     while (l < r) {
16         int mid = (l + r + 1) >> 1;
17         if (check(mid)) l = mid;
18         else r = mid - 1;
19     }
20    return l;
21 }

```

## 3.3 三分搜

```

1 //只要是單峰函數，三分可找最大或最小，以下為最小化
2 //計算lmid以及rmid時要避免數字溢出
3 while (r - l > eps) { // [l, r]
4     mid = (l + r) / 2;
5     lmid = mid - eps;
6     rmid = mid + eps;
7     if (f(lmid) < f(rmid)) r = mid;
8     else l = mid;
9 }

```

## 3.4 dinic

```

1 const int maxn = 1e5 + 10;
2 const int inf = 0x3f3f3f3f;
3 struct Edge { int s, t, cap, flow; };
4 int n, m, S, T;
5 int level[maxn], dfs_idx[maxn];
6 vector<Edge> E;
7 vector<vector<int>> G;
8 void init() {
9     S = 0;
10    T = n + m;
11    E.clear();
12    G.assign(maxn, vector<int>());
13 }
14 void addEdge(int s, int t, int cap) {
15     E.push_back({s, t, cap, 0});
16     E.push_back({t, s, 0, 0});
17     G[s].push_back(E.size()-2);
18     G[t].push_back(E.size()-1);
19 }
20 bool bfs() {
21     queue<int> q({S});
22     memset(level, -1, sizeof(level));
23     level[S] = 0;
24     while(!q.empty()) {
25         int cur = q.front();
26         q.pop();
27         for(int i : G[cur]) {
28             Edge e = E[i];
29             if(level[e.t]==-1 &&
30                e.cap>e.flow) {
31                 level[e.t] = level[e.s] + 1;
32                 q.push(e.t);
33             }
34         }
35     }
36     return level[T];
37 }
38 int dfs(int cur, int lim) {
39     if(cur==T || lim==0) return lim;
40     int result = 0;
41     for(int& i=dfs_idx[cur]; i<G[cur].size()
42        && lim>0; i++) {
43         Edge& e = E[G[cur][i]];
44         if(level[e.s]+1 != level[e.t]) continue;
45         int flow = dfs(e.t, min(lim,
46            e.cap-e.flow));
47         if(flow <= 0) continue;
48         e.flow += flow;
49         result += flow;
50         E[G[cur][i]^1].flow -= flow;
51         lim -= flow;
52     }
53     return result;
54 }
55 int dinic() { // O((V^2)E)
56     int result = 0;
57     while(bfs()) {
58         memset(dfs_idx, 0, sizeof(dfs_idx));
59         result += dfs(S, inf);
60     }
61     return result;
62 }

```

### 3.5 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
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) break;
49         }
50         totalCost += minCost;
51         ways = (ways * currWays) % MOD;
52     }
53 }

```

### 3.6 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 struct edge {int to, nt;} e[M << 1];
9 void add(int u, int v) {e[++tot].to = v,
10    e[tot].nt = hd[u], hd[u] = tot;}
11 void uadd(int u, int v) {add(u,v),add(v,u);}
12 bool bz[M << 1];
13 int bcc_cnt, dfn[N], low[N], vis_bcc[N];
14 vector<vector<int>> bcc;
15 void tarjan(int x, int in) {
16     dfn[x] = low[x] = ++bcc_cnt;
17     for (int i = hd[x]; i; i = e[i].nt) {
18         int v = e[i].to;
19         if (dfn[v] == 0) {
20             tarjan(v, i);
21             if (dfn[x] < low[v])
22                 bz[i] = bz[i ^ 1] = true;
23             low[x] = min(low[x], low[v]);
24         } else if (i != (in ^ 1))
25             low[x] = min(low[x], dfn[v]);
26     }
27 }
28 void dfs(int x, int id) {
29     vis_bcc[x] = id, bcc[id - 1].push_back(x);
30     for (int i = hd[x]; i; i = e[i].nt) {
31         int v = e[i].to;
32         if (vis_bcc[v] || bz[i]) continue;
33         dfs(v, id);
34     }
35 }

```

### 3.7 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 { int to, nt; } e[M << 1];
9
10 int hd[N], tot = 1;
11
12 void add(int u, int v) { e[++tot] = edge{v,
13    hd[u]}, hd[u] = tot; }
14
15 void uadd(int u, int v) {add(u,v),add(v,u);}
16
17 int ans;
18 int dfn[N], low[N], bcc_cnt;
19 int sta[N], top, cnt;
20 bool cut[N];
21 vector<int> dcc[N];
22 int root;
23
24 void tarjan(int u) {
25     dfn[u]=low[u] = ++bcc_cnt, sta[++top] = u;
26     if (u == root && hd[u] == 0) {
27         dcc[++cnt].push_back(u);
28         return;
29     }
30     int f = 0;
31     for (int i = hd[u]; i; i = e[i].nt) {
32         int v = e[i].to;
33         if (!dfn[v]) {
34             tarjan(v);
35             low[u] = min(low[u], low[v]);
36             if (low[v] >= dfn[u]) {
37                 if (++f > 1 || u != root)
38                     cut[u] = true;
39                 cnt++;
40                 do dcc[cnt].push_back(sta[top--]);
41                 while (sta[top + 1] != v);
42                 dcc[cnt].push_back(u);
43             }
44         } else
45             low[u] = min(low[u], dfn[v]);
46     }
47 }

```

### 3.8 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 }

```

### 3.9 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],
8     vis[maxn];
9 // 對於每個點，選擇對它入度最小的那條邊
10 // 找環，如果沒有則 return;
11 // 進行縮環並更新其他點到環的距離。
12 int dirMST(vector<Edge> edges, int low) {
13     int result = 0, root = 0, N = n;
14     while(true) {
15         memset(inEdge, 0x3f, sizeof(inEdge));
16         // 找所有點的 in edge 放進 inEdge
17         // optional: low 為最小 cap 限制
18         for(const Edge& e : edges) {
19             if(e.cap < low) continue;
20             if(e.s!=e.t && e.cost<inEdge[e.t]) {
21                 inEdge[e.t] = e.cost;
22                 pre[e.t] = e.s;
23             }
24         }
25         for(int i=0; i<N; i++) {
26             if(i!=root && inEdge[i]==inf)
27                 return -1; //除了root 還有點沒有 in edge
28         }
29         int seq = inEdge[root] = 0;
30         memset(idx, -1, sizeof(idx));
31         memset(vis, -1, sizeof(vis));
32         // 找所有的 cycle，一起編號為 seq
33         for(int i=0; i<N; i++) {
34             result += inEdge[i];
35             int cur = i;
36             while(vis[cur]!=i && idx[cur]==-1) {
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         }
47         if(seq == 0) return result; // 沒有 cycle
48         for(int i=0; i<N; i++)
49             // 沒有被縮點的點
50             if(idx[i] == -1) idx[i] = seq++;
51         // 縮點並重新編號
52         for(Edge& e : edges) {
53             if(idx[e.s] != idx[e.t])
54                 e.cost -= inEdge[e.t];
55             e.s = idx[e.s];
56             e.t = idx[e.t];
57         }
58         N = seq;
59         root = idx[root];
60     }

```

### 3.10 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]) diff = min(diff, Lx[i] +
36                                         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
58                             //labeling以增加增廣路徑
59         }
60     }
61 }
62 int main() {
63     while (scanf("%d", &n) != EOF) {
64         for (int i = 0; i < n; ++i)
65             for (int j = 0; j < n; ++j)
66                 scanf("%d", &W[i][j]);
67         KM();
68         int res = 0;
69         for (int i = 0; i < n; ++i) {
70             if (i != 0) printf(" ");
71             else printf("%d", Lx[i]);
72             res += Lx[i];
73         }
74         puts("");
75         for (int i = 0; i < n; ++i) {
76             if (i != 0) printf(" ");
77             else printf("%d", Ly[i]);
78         }
79         puts("");
80     }

```

```

75     else printf("%d", Ly[i]);
76     res += Ly[i];
77 }
78 puts("");
79 printf("%d\n", res);
80 }
81 return 0;
82 }

```

### 3.11 二分圖最大匹配

```

1 /* 核心：最大點獨立集 = |V| -
2    /最大匹配數 /，用匈牙利演算法找出最大匹配數 */
3 vector<Student> boys;
4 vector<Student> girls;
5 vector<vector<int>> G;
6 bool used[505];
7 int p[505];
8 bool match(int i) {
9     for (int j: G[i]) {
10         if (!used[j]) {
11             used[j] = true;
12             if (p[j] == -1 || match(p[j])) {
13                 p[j] = i;
14                 return true;
15             }
16         }
17     }
18     return false;
19 }
20 void maxMatch(int n) {
21     memset(p, -1, sizeof(p));
22     int res = 0;
23     for (int i = 0; i < boys.size(); ++i) {
24         memset(used, false, sizeof(used));
25         if (match(i)) ++res;
26     }
27     cout << n - res << '\n';

```

### 3.12 差分

```

1 用途：在區間 [l, r] 加上一個數字v。
2 b[l] += v; (b[0~l] 加上v)
3 b[r+1] -= v; (b[r+1~n] 減去v (b[r] 仍保留v) )
4 給的 a[] 是前綴和數列，建構 b[]，
5 因為 a[i] = b[0] + b[1] + b[2] + ... + b[i]。
6 所以 b[i] = a[i] - a[i-1]。
7 在 b[l] 加上 v，b[r+1] 減去 v，
8 最後再從 0 跑到 n 使 b[i] += b[i-1]。
9 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 }

```

### 3.13 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 //一邊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) &&
38                 (dis[edge.v] > dis[u] + edge.cost)) {
39                 dis[edge.v] = dis[u] + edge.cost;
40                 parent[edge.v] = edgeIndex;
41                 outFlow[edge.v] = min(outFlow[u],
42                     (long long)(edge.cap - edge.flow));
43                 if (!inqueue[edge.v]) {
44                     q.push(edge.v);
45                     inqueue[edge.v] = true;
46                 }
47             }
48         }
49     }
50     //如果dis[t] > 0代表根本不賺還倒賠
51     if (dis[t] > 0) return false;
52     maxFlow += outFlow[t];
53     minCost += dis[t] * outFlow[t];
54     //一路更新回去這次最短路流完後要維護的
55     //MaxFlow演算法相關(如反向邊等)
56     int curr = t;
57     while (curr != s) {
58         edges[parent[curr]].flow += outFlow[t];
59         edges[parent[curr]^1].flow -= outFlow[t];
60         curr = parent[curr];
61     }
62     return true;
63 }
64 long long MCMF() {
65     long long maxFlow = 0, minCost = 0;
66     while (SPFA(maxFlow, minCost));
67     return minCost;
68 }
69 int main() {
70     int T;
71     scanf("%d", &T);
72     for (int Case = 1; Case <= T; ++Case) {
73         //總共幾個月, 囤貨成本
74         int M, I;
75         scanf("%d %d", &M, &I);
76         //node size
77         n = M + M + 2;

```

```

78         G.assign(n + 5, vector<int>());
79         edges.clear();
80         s = 0;
81         t = M + M + 1;
82         for (int i = 1; i <= M; ++i) {
83             int produceCost, produceMax,
84                 sellPrice, sellMax, inventoryMonth;
85             scanf("%d %d %d %d %d", &produceCost,
86                 &produceMax, &sellPrice,
87                 &sellMax, &inventoryMonth);
88             addEdge(s, i, produceMax, produceCost);
89             addEdge(M + i, t, sellMax, -sellPrice);
90             for (int j = 0; j <= inventoryMonth; ++j) {
91                 if (i + j <= M)
92                     addEdge(i, M + i + j, INF, I * j);
93             }
94             printf("Case %d: %lld\n", Case, -MCMF());
95         }
96         return 0;
97     }

```

### 3.14 LCA 倍增法

```

1 //倍增法預處理 $O(n\log n)$ , 查詢 $O(\log n)$ ,
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 }
27 long long lca(int x, int y) {
28     //此函數是找lca同時計算x、y的距離 -> dis(x,
29         lca) + dis(lca, y)
30     //讓y比x深
31     if (dep[x] > dep[y]) 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];
46             y = fa[y][i];
47         }
48     }
49     //最後發現不能跳了, 此時x的第2^0 = 1個祖先
50     // (或說y的第2^0 = 1的祖先)即為x、y的lca
51     res += dis[x][0] + dis[y][0];
52     return res;
53 }

```

```

51 int main() {
52     int n, q;
53     while (~scanf("%d", &n) && n) {
54         int v, w;
55         G.assign(n + 5, vector<Edge>());
56         for (int i = 1; i <= n - 1; ++i) {
57             scanf("%d %d", &v, &w);
58             G[i + 1].push_back({i + 1, v + 1, w});
59             G[v + 1].push_back({v + 1, i + 1, w});
60         }
61         dfs(1, 0);
62         scanf("%d", &q);
63         int u;
64         while (q--) {
65             scanf("%d %d", &u, &v);
66             printf("%lld%c", lca(u + 1, v + 1),
67                 (q ? ' ': '\n'));
68         }
69     }

```

### 3.15 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 int top[maxn]; //樹鍊的頂點
10 int dfnToNode[maxn]; //將dfn轉成node編碼
11 int hson[maxn]; //重兒子
12 int dfsTime = 1;
13 vector<vector<int>> G; //tree
14 //處理parent、depth、subtreeSize、dfnToNode
15 void dfs1(int u, int p) {
16     parent[u] = p;
17     hson[u] = -1;
18     subtreeSize[u] = 1;
19     for (int v: G[u]) {
20         if (v != p) {
21             depth[v] = depth[u] + 1;
22             dfs1(v, u);
23             subtreeSize[u] += subtreeSize[v];
24             if (hson[u] == -1 ||
25                 subtreeSize[hson[u]] < subtreeSize[v]) {
26                 hson[u] = v;
27             }
28         }
29     }
30 }
31 //實際剖分 <- 參數t是top的意思
32 //t初始應為root本身
33 void dfs2(int u, int t) {
34     top[u] = t;
35     dfn[u] = dfsTime;
36     dfnToNode[dfsTime] = u;
37     ++dfsTime;
38     //葉子點 -> 沒有重兒子
39     if (hson[u] == -1) return;
40     //優先對重兒子dfs, 才能保證同一重鍊dfn連續
41     dfs2(hson[u], t);
42     for (int v: G[u]) {
43         if (v != parent[u] && v != hson[u])
44             dfs2(v, v);
45     }
46 }
47 //不斷跳鍊, 當跳到同一條鍊時, 深度小的即為LCA
48 //跳鍊時優先鍊頂深度大的跳
49 int LCA(int u, int v) {
50     while (top[u] != top[v]) {
51         if (depth[top[u]] > depth[top[v]])
52             u = parent[top[u]];
53         else
54             v = parent[top[v]];

```

```

55 }
56 return (depth[u] > depth[v]) ? v : u;
57 }
58 int getK_parent(int u, int k) {
59     while (k-- && (u != -1)) u = parent[u];
60     return u;
61 }
62 int main() {
63     int n;
64     while (scanf("%d", &n) && n) {
65         dfsTime = 1;
66         G.assign(n + 5, vector<int>());
67         int u, v;
68         for (int i = 1; i < n; ++i) {
69             scanf("%d %d", &u, &v);
70             G[u].emplace_back(v);
71             G[v].emplace_back(u);
72         }
73         dfs1(1, -1);
74         dfs2(1, 1);
75         int q;
76         scanf("%d", &q);
77         for (int i = 0; i < q; ++i) {
78             scanf("%d %d", &u, &v);
79             //先得到LCA
80             int lca = LCA(u, v);
81             //計算路徑長(經過的中點)
82             int dis = depth[u] + depth[v] - 2 *
                depth[lca];
83             //讓v比u深或等於
84             if (depth[u] > depth[v]) swap(u, v);
85             if (u == v) {
86                 printf("The fleas meet at %d.\n", u);
87             }
88             else if (dis % 2 == 0) {
89                 //路徑長是even -> 有中點
90                 printf("The fleas meet at %d.\n",
                    getK_parent(v, dis / 2));
91             }
92             else {
93                 //路徑長是odd -> 沒有中點
94                 if (depth[u] == depth[v]) {
95                     int x = getK_parent(u, dis / 2);
96                     int y = getK_parent(v, dis / 2);
97                     if (x > y) swap(x, y);
98                     printf("The fleas jump forever
                        between %d and %d.\n", x, y);
99                 }
100             }
101             //技巧：讓深的點v往上dis / 2步 = y，
102             //這個點的parent設為x
103             //此時的x、y就是答案要的中點兩點
104             //主要是往下不好找，所以改用深的點用parent
105             int y = getK_parent(v, dis / 2);
106             int x = getK_parent(y, 1);
107             if (x > y) swap(x, y);
108             printf("The fleas jump forever
                between %d and %d.\n", x, y);
109         }
110     }
111 }
112 }
113 }

```

## 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,  $val[v] - val[u] \neq w$  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 struct TrieNode {
4     int cnt;
5     TrieNode* child[maxc];
6     TrieNode() {
7         cnt = 0;
8         for(auto& node : child)
9             node = nullptr;
10     }
11 };
12 struct Trie {
13     TrieNode* root;
14     Trie() { root = new TrieNode(); }
15     void insert(string word) {
16         TrieNode* cur = root;
17         for(auto& ch : word) {
18             int c = ch - minc;
19             if(!cur->child[c])
20                 cur->child[c] = new TrieNode();
21             cur = cur->child[c];
22         }
23         cur->cnt++;
24     }
25     void remove(string word) {
26         TrieNode* cur = root;
27         for(auto& ch : word) {
28             int c = ch - minc;
29             if(!cur->child[c]) return;
30             cur = cur->child[c];
31         }
32         cur->cnt--;
33     }
34     // 字典裡有出現 word
35     bool search(string word, bool prefix=0) {
36         TrieNode* cur = root;
37         for(auto& ch : word) {
38             int c = ch - minc;
39             if(!cur->child[c]) return false;
40         }
41         return cur->cnt || prefix;
42     }
43     // 字典裡有 word 的前綴為 prefix
44     bool startsWith(string prefix) {
45         return search(prefix, true);
46     }
47 };

```

### 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     ACTrie() {
15         seq = 0;
16         root = newNode();
17     }
18     int newNode() {
19         for(int i=0; i<maxc; trie[seq][i]=0);
20         val[seq] = cnt[seq] = fail[seq] = 0;
21         return seq++;
22     }
23     void insert(char* s, int wordId=0) {
24         int p = root;
25         for(; *s; s++) {
26             int c = *s - minc;
27             if(!trie[p][c]) trie[p][c] = newNode();
28             p = trie[p][c];
29         }
30         val[p] = wordId;
31         cnt[p]++;
32     }
33     void build() {
34         queue<int> q({root});
35         while(!q.empty()) {
36             int p = q.front();
37             q.pop();
38             for(int i=0; i<maxc; i++) {
39                 int& t = trie[p][i];
40                 if(t) {
41                     fail[t] = p?trie[fail[p]][i]:root;
42                     q.push(t);
43                 } else {
44                     t = trie[fail[p]][i];
45                 }
46             }
47         }
48         // 要存 wordId 才要 vec
49         // 同單字重複match要把所有vis取消掉
50         int match(char* s, vector<int>& vec) {
51             int res = 0;
52             memset(vis, 0, sizeof(vis));
53             for(int p=root; *s; s++) {
54                 p = trie[p][*s-minc];
55                 for(int k=p; k && !vis[k]; k=fail[k]) {
56                     vis[k] = true;
57                     res += cnt[k];
58                     if(cnt[k]) vec.push_back(val[k]);
59                 }
60             }
61             return res; // 匹配到的單字量
62         }
63     };
64 };
65
66 ACTrie ac; // 建構, 初始化
67 ac.insert(s); // 加字典單字
68 // 加完字典後
69 ac.build(); // !!! 建 failure link !!!
70 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
6 inline int pull(int l, int r) {
7 // 隨題目改變sum~max~min
8 // l、r是左右樹的index
9 return st[l] + st[r];
10 }
11 void build(int l, int r, int i) {
12 // 在[l, r]區間建樹，目前根的index為i
13 if (l == r) {
14 st[i] = data[l];
15 return;
16 }
17 int mid = l + ((r - l) >> 1);
18 build(l, mid, i * 2);
19 build(mid + 1, r, i * 2 + 1);
20 st[i] = pull(i * 2, i * 2 + 1);
21 }
22 int qry(int ql, int qr, int l, int r, int i) {
23 // [ql,qr]是查詢區間，[l,r]是當前節點包含的區間
24 if (ql <= l && r <= qr)
25 return st[i];
26 int mid = l + ((r - l) >> 1);
27 if (tag[i]) {
28 //如果當前懶標有值則更新左右節點
29 st[i * 2] += tag[i] * (mid - l + 1);
30 st[i * 2 + 1] += tag[i] * (r - mid);
31 tag[i * 2] += tag[i];
32 tag[i * 2 + 1] += tag[i];
33 tag[i] = 0;
34 }
35 int sum = 0;
36 if (ql <= mid)
37 sum += query(ql, qr, l, mid, i * 2);
38 if (qr > mid)
39 sum += query(ql, qr, mid + 1, r, i * 2 + 1);
40 return sum;
41 }
42 void update(
43 int ql, int qr, int l, int r, int i, int c) {
44 // [ql,qr]是查詢區間，[l,r]是當前節點包含的區間
45 // c是變化量
46 if (ql <= l && r <= qr) {
47 st[i] += (r - l + 1) * c;
48 //求和,此需乘上區間長度
49 tag[i] += c;
50 return;
51 }
52 int mid = l + ((r - l) >> 1);
53 if (tag[i] && l != r) {
54 //如果當前懶標有值則更新左右節點
55 st[i * 2] += tag[i] * (mid - l + 1);
56 st[i * 2 + 1] += tag[i] * (r - mid);
57 tag[i * 2] += tag[i]; //下傳懶標至左節點
58 tag[i * 2 + 1] += tag[i]; //下傳懶標至右節點
59 tag[i] = 0;
60 }
61 if (ql <= mid)
62 update(ql, qr, l, mid, i * 2, c);
63 if (qr > mid)
64 update(ql, qr, mid + 1, r, i * 2 + 1, c);
65 st[i] = pull(i * 2, i * 2 + 1);
66 //如果是直接改值而不是加值，query與update中的tag與st
67 //改值從+=改成=

```

## 4.5 線段樹 2D

```

1 #define maxn 2005 //500 * 4 + 5 //純2D
2 //segment tree 區間查詢單點修改最大最小值
3 int maxST[maxn][maxn], minST[maxn][maxn];
4 int N;
5 void modifyY(int index, int l, int r, int val,
6 int yPos, int xIndex, bool xIsLeaf) {
7 if (l == r) {
8 if (xIsLeaf) {
9 maxST[xIndex][index] =
10 minST[xIndex][index] = val;

```

```

9 return;
10 }
11 maxST[xIndex][index] =
12 max(maxST[xIndex * 2][index],
13 maxST[xIndex * 2 + 1][index]);
14 minST[xIndex][index] =
15 min(minST[xIndex * 2][index],
16 minST[xIndex * 2 + 1][index]);
17 }
18 else {
19 int mid = (l + r) / 2;
20 if (yPos <= mid)
21 modifyY(index * 2, l, mid, val, yPos,
22 xIndex, xIsLeaf);
23 else
24 modifyY(index * 2 + 1, mid + 1, r, val,
25 yPos, xIndex, xIsLeaf);
26 maxST[xIndex][index] =
27 max(maxST[xIndex * 2][index * 2],
28 maxST[xIndex][index * 2 + 1]);
29 minST[xIndex][index] =
30 min(minST[xIndex * 2][index * 2],
31 minST[xIndex][index * 2 + 1]);
32 }
33 }
34 void modifyX(int index, int l, int r, int
35 val, int xPos, int yPos) {
36 if (l == r) {
37 modifyY(1, 1, N, val, yPos, index, true);
38 }
39 else {
40 int mid = (l + r) / 2;
41 if (xPos <= mid)
42 modifyX(index * 2, l, mid, val, xPos, yPos);
43 else
44 modifyX(index * 2 + 1, mid + 1, r, val,
45 xPos, yPos);
46 modifyY(1, 1, N, val, yPos, index, false);
47 }
48 }
49 void queryY(int index, int l, int r, int yql,
50 int yqr, int xIndex, int& vmax, int& vmin) {
51 if (yql <= l && r <= yqr) {
52 vmax = max(vmax, maxST[xIndex][index]);
53 vmin = min(vmin, minST[xIndex][index]);
54 }
55 else {
56 int mid = (l + r) / 2;
57 if (yql <= mid)
58 queryY(index * 2, l, mid, yql, yqr,
59 xIndex, vmax, vmin);
60 if (mid < yqr)
61 queryY(index * 2 + 1, mid + 1, r, yql,
62 yqr, xIndex, vmax, vmin);
63 }
64 }
65 void queryX(int index, int l, int r, int
66 xql, int xqr, int yql, int yqr, int&
67 vmax, int& vmin) {
68 if (xql <= l && r <= xqr) {
69 queryY(1, 1, N, yql, yqr, index, vmax, vmin);
70 }
71 else {
72 int mid = (l + r) / 2;
73 if (xql <= mid)
74 queryX(index * 2, l, mid, xql, xqr, yql,
75 yqr, vmax, vmin);
76 if (mid < xqr)
77 queryX(index * 2 + 1, mid + 1, r, xql,
78 xqr, yql, yqr, vmax, vmin);
79 }
80 }
81 }
82 }
83 int main() {
84 while (scanf("%d", &N) != EOF) {
85 int val;
86 for (int i = 1; i <= N; ++i) {
87 for (int j = 1; j <= N; ++j) {
88 scanf("%d", &val);

```

```

69 modifyX(1, 1, N, val, i, j);
70 }
71 }
72 int q;
73 int vmax, vmin;
74 int xql, xqr, yql, yqr;
75 char op;
76 scanf("%d", &q);
77 while (q--) {
78 getchar(); //for \n
79 scanf("%c", &op);
80 if (op == 'q') {
81 scanf("%d %d %d %d", &xql, &yql,
82 &xqr, &yqr);
83 vmax = -0x3f3f3f3f;
84 vmin = 0x3f3f3f3f;
85 queryX(1, 1, N, xql, xqr, yql, yqr,
86 vmax, vmin);
87 printf("%d %d\n", vmax, vmin);
88 }
89 else {
90 scanf("%d %d %d", &xql, &yql, &val);
91 modifyX(1, 1, N, val, xql, yql);
92 }
93 }
94 return 0;
95 }

```

## 4.6 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 example:
4 給出一個長度為 n 的數組，
5 輸出每 k 個連續的數中的最大值和最小值。
6 #define maxn 1000100
7 int q[maxn], a[maxn];
8 int n, k;
9 //得到這個隊列裡的最小值，直接找到最後的就行了
10 void getmin() {
11 int head=0, tail=0;
12 for(int i=1;i<k;i++) {
13 while(head<=tail&& a[q[tail]]>=a[i]) tail--;
14 q[++tail]=i;
15 }
16 for(int i=k;i<=n;i++) {
17 while(head<=tail&& a[q[tail]]>=a[i]) tail--;
18 q[++tail]=i;
19 while(q[head]<=i-k) head++;
20 cout<<a[q[head]]<<" ";
21 }
22 cout<<endl;
23 }
24 //和上面同理
25 void getmax() {
26 int head=0, tail=0;
27 for(int i=1;i<k;i++) {
28 while(head<=tail&& a[q[tail]]<=a[i]) tail--;
29 q[++tail]=i;
30 }
31 for(int i=k;i<=n;i++) {
32 while(head<=tail&& a[q[tail]]<=a[i]) tail--;
33 q[++tail]=i;
34 while(q[head]<=i-k) head++;
35 cout<<a[q[head]]<<" ";
36 }
37 cout<<endl;
38 }

```

## 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  $\Delta$

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  $(H_x, H_y)$

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

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

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

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

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

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

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

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

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

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 Vector operator-(Vector a, Vector b)
13 {return Vector(a.x-b.x, a.y-b.y);}
14 Vector operator*(Vector a, DBL b)
15 {return Vector(a.x*b, a.y*b);}
16 Vector operator/(Vector a, DBL b)
17 {return Vector(a.x/b, a.y/b);}
18 Tp dot(Vector a, Vector b)
19 {return a.x*b.x + a.y*b.y;}
20 Tp cross(Vector a, Vector b)
21 {return a.x*b.y - a.y*b.x;}
22 DBL Vector::length()
23 {return sqrt(dot(*this, *this));}
24 Vector unit_normal_vector(Vector v) {
25     DBL len = v.length();
26     return Vector(-v.y/len, v.x/len);
27 }
```

### 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 };
```

### 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) {return dis(o,p) <= r;}
18 };
```

## 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    }
```

```

20 if(q1 < q2)
21     P[qr-1] = intersection(L[qr-1], L[q1]);
22 }
23 for(; q1<q2 && !lft(P[qr-1], L[q1]); qr--);
24 if(qr-q1 <= 1) return {};
25 P[qr] = intersection(L[qr], L[q1]);
26 return Polygon(P+q1, 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)<sup>2</sup>
- 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 bool hasIntersection(Segment a, Segment b) {
12     // 判斷在 X 軸 Y 軸的投影是否相交
13     auto intr1D=[&](DBL w, DBL x, DBL y, DBL z){
14         if(w > x) swap(w, x);
15         if(y > z) swap(y, z);
16         return dcmp(max(w, y), min(x, z)) <= 0;
17     };
18
19     DBL a1 = cross(a.v, b.s-a.s);
20     DBL a2 = cross(a.v, b.e-a.s);
21     DBL b1 = cross(b.v, a.s-b.s);
22     DBL b2 = cross(b.v, a.e-b.s);
23
24     return intr1D(a.s.x, a.e.x, b.s.x, b.e.x)
25         && intr1D(a.s.y, a.e.y, b.s.y, b.e.y)
26         && dcmp(a1) * dcmp(a2) <= 0
27         && dcmp(b1) * dcmp(b2) <= 0;
28 }
29 Point intersection(Segment a, Segment b) {
30     Vector v = b.s - a.s;
31     DBL c1 = cross(a.v, b.v);
32     DBL c2 = cross(v, b.v);
33     DBL c3 = cross(v, a.v);
34
35     if(dcmp(c1) < 0) c1=-c1, c2=-c2, c3=-c3;
36     if(dcmp(c1) && dcmp(c2)>=0 && dcmp(c3)>=0)

```

```

37     && dcmp(c1, c2)>=0 && dcmp(c1, c3)>=0)
38     return a.s + (a.v * (c2 / c1));
39     return Point(inf, inf); // a 和 b 共線
40 }
41 Point intersection(Line a, Line b) {
42     // cross(a.v, b.v) == 0 時平行
43     Vector u = a.p - b.p;
44     DBL t = 1.0*cross(b.v, u)/cross(a.v, b.v);
45     return a.p + a.v*t;
46 }
47 DBL dis(Point a, Point b)
48 {return sqrt(dot(a-b, a-b));}
49 DBL dis(Point p, Line l)
50 {return abs(cross(p-l.p, l.v))/l.v.length();}
51 DBL dis(Point p, Segment s) {
52     Vector u = p - s.s, v = p - s.e;
53     if(dcmp(dot(s.v, u))<=0) return u.length();
54     if(dcmp(dot(s.v, v))>=0) return v.length();
55     return abs(cross(s.v, u)) / s.length();
56 }
57 DBL dis(Segment a, Segment b) {
58     if(hasIntersection(a, b)) return 0;
59     return min({
60         dis(a.s, b), dis(a.e, b),
61         dis(b.s, a), dis(b.e, a)
62     });
63 }
64 DBL dis(Line a, Line b) {
65     if(dcmp(cross(a.v, b.v)) == 0) return 0;
66     return dis(a.p, b);
67 }
68 Point getPedal(Line l, Point p) {
69     // 返回 p 在 l 上的垂足(投影點)
70     DBL len = dot(p-l.p, l.v) / dot(l.v, l.v);
71     return l.p + l.v * len;
72 }

```

## 6 DP

### 6.1 背包

#### 0-1 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

```

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; 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

```

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]);

```

### 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]);

```

### 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 }

```

### 二維費用背包

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

```

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][j], dp[i-c[k]][j-t[k]] + 1);

```

### 分組背包

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];
8 long long dp[maxn][maxn];
9 long long a[maxn];
10 long long solve(int l, int r) {
11     if (l > r) return 0;
12     if (vis[l][r]) return dp[l][r];
13     vis[l][r] = true;
14     long long res = a[l] - solve(l + 1, r);
15     res = max(res, a[r] - solve(l, r - 1));
16     return dp[l][r] = res;
17 }
18 int main() {
19     ...
20     printf("%lld\n", solve(1, n));
21 }

```

## 6.3 string DP

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

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

Longest Palindromic Subsequence

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

## 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的child且距離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    //統計在u子樹中距離u為k的數量
20    res += dp[u][k];
21    long long cnt = 0;
22    for (int v: G[u]) {
23        if (v == p) continue; //重點算法
24        for (int x = 0; x <= k-2; ++x) {
25            cnt +=
26                dp[v][x]*(dp[u][k-x-1]-dp[v][k-x-2]);
27        }
28    }
29    res += cnt / 2;
30 }
31 int main() {
32     ...
33     dfs(1, -1);
34     printf("%lld\n", res);
35     return 0;
36 }

```