

Contents

1	Ubuntu	1
1.1	Terminal Title	1
1.2	GDB 參數	1
1.3	GDB 指令	1
2	字串	2
2.1	最長迴文子字串	2
2.2	Manacher	2
2.3	KMP	2
2.4	Z Algorithm	2
2.5	Suffix Array	2
3	math	3
3.1	公式	3
3.2	Rational	3
3.3	乘法逆元、組合數	3
3.4	歐拉函數	3
3.5	質數與因數	4
3.6	高斯消去	4
3.7	Extended GCD	4
3.8	大步小步	4
3.9	Pisano Period	5
3.10	矩陣快速幂	5
4	algorithm	6
4.1	greedy	6
4.2	Josephus Problem	7
4.3	二分搜	7
4.4	三分搜	7
4.5	dinic	7
4.6	dijkstra	8
4.7	SPFA	8
4.8	SCC Kosaraju	8
4.9	SCC Tarjan	8
4.10	BCC 邊	9
4.11	BCC 點	9
4.12	Articulation Points Tarjan	9
4.13	最小樹狀圖	10
4.14	KM	10
4.15	二分圖最大匹配	10
4.16	差分	10
4.17	莫隊	11
4.18	MCMF	11
4.19	Blossom Algorithm	12
4.20	Dancing Links	12
4.21	Astar	12
4.22	LCA 倍增法	13
4.23	LCA 樹壓平 RMQ	13
4.24	LCA 樹鍊剖分	14
5	DataStructure	15
5.1	BIT	15
5.2	帶權併查集	15
5.3	Trie	15
5.4	AC Trie	15
5.5	線段樹 1D	16
5.6	線段樹 2D	16
5.7	權值線段樹	17
5.8	ChthollyTree	17
5.9	單調隊列	17
6	Geometry	18
6.1	公式	18
6.2	Template	18
6.3	旋轉卡尺	18
6.4	半平面相交	18
6.5	Polygon	19
6.6	凸包	19
6.7	最小圓覆蓋	19
6.8	交點、距離	19
7	DP	20
7.1	背包	20
7.2	Range DP	21
7.3	Deque 最大差距	21
7.4	string DP	21
7.5	Barcode	21
7.6	LCS 和 LIS	21
7.7	樹 DP 有幾個 path 長度為 k	21
7.8	抽屜	21
7.9	TreeDP reroot	22
7.10	Weighted LIS	22

1 Ubuntu

1.1 Terminal Title

方法：

```
1 | PS1='\\e;\\a'
```

舉例：

```
1 | PS1='\\e;\\W\\a\\w$ '
```

```
2 | // 可改 \\W 為想要的 Title
```

- [\\a] - ASCII Bell
- [\\d] - 日期
- [\\e] - 跳脫字元
- [\\h] - 主機
- [\\H] - 主機名
- [\\t] - 時間
- [\\u] - 使用者
- [\\w] - 當前路徑
- [\\W] - 當前資料夾名稱

1.2 GDB 參數

```
1 | g++ main.cpp -g -o main
```

```
2 | gdb -tui -q ./main
```

- [-tui] 在終端機顯示文字檔案
- [-q] 在初始設定不顯示版本資訊

1.3 GDB 指令

command	功能
[break] [b]	在當前這行放中斷點
[b fn]	在函式 fn 的開頭放中斷點
[b N]	在第 N 行放中斷點
[clear N] [cl N]	刪除第 N 行的中斷點
[command N] [comm N]	設定編號 N 的中斷點的指令
[cond N i==3]	編號 N 的中斷點 i=3 再停
[delete] [d]	刪除所有中斷點
[d N]	刪除編號為 N 的中斷點
[disable] [dis]	使所有中斷點無效
[dis N]	使編號為 N 的中斷點無效
[dp a, "%d\\n", c]	碰到第 a 行時印出 c
[enable] [en]	使所有中斷點有效
[en N]	使編號為 N 的中斷點有效
[tbreak N] [tb N]	只停一次的 [b N]
[watch x==3] [wa x==3]	執行到符合條件時停止

command	功能
[call fn]	呼叫函式 fn
[display x] [disp x]	每執行一步都印出 x
[print var] [p var]	印出 var
[set print] [set p]	設定 print
[set p array]	array 印出漂亮格式
[set p array off]	取消 array 印出漂亮格式
[set p array-i]	array 印出索引
[set p array-i off]	取消 array 印出索引
[set p el N]	array 最多印 N 個元素
[set p pre]	struct 印出漂亮格式
[set p pre off]	取消 struct 印出漂亮格式
[set var N=3]	將 N 設為 3
[undisp x]	取消編號為 x 的 disp

command	功能
[list] [l]	印出 10 行程式碼
[l N]	印出包含第 N 行的程式碼
[l fn]	印出包含函式 fn 的程式碼
[l var]	印出包含變數 var 的程式碼

command	功能
[record] [rec]	開始記錄
[rec s]	停止記錄

command	功能
[continue] [c]	執行到下一個中斷點或錯誤
[finish] [fin]	執行到跳出堆疊框
[kill] [k]	終止程式
[next] [n]	執行下一行（不進入函式）
[n N]	執行 [n] 一共 N 次
[reverse-continue] [rc]	反向的 [c]
[rn]	反向的 [n]
[rs]	反向的 [s]
[run] [r]	執行程式
[r < FILE]	像 [r]，輸入為 FILE
[start]	開始執行，停在第一步
[start < FILE]	像 [start]，輸入為 FILE
[step] [s]	執行下一步（進入函式）
[s N]	執行 [s] 一共 N 次
[until N] [u N]	執行到第 N 行停下來

command	功能
[backtrace] [bt]	印出所有堆疊
[down] [do]	印出下一層堆疊
[frame] [f]	印出當前堆疊
[f N]	印出往上第 N 層堆疊
[Return] [ret]	從當前函式 return
[up]	印出上一層堆疊

command	功能
[info] [i]	顯示資訊
[i b]	列出所有中斷點資訊
[i disp]	列出所有監看變數資訊
[i local] [i lo]	列出所有區域變數資訊
[i var]	列出所有變數

command	功能
[help] [h]	協助
[b N if i==2]	當 i=2 時停在第 N 行
[quit] [q]	結束 gdb

command	功能
[refresh] [ref]	刷新終端機佈置
[tui d]	取消使用 TUI
[tui e]	使用 TUI
[update] [upd]	更新視窗以顯示當前程式碼

- 按 <enter> 鍵可以執行上一動
- 執行 reverse (像是 rc, rn, rs) 前，要先執行 record (rec)，但存不了幾步就沒空間了
- 輸入命令 command (comm) 後，接下來每一行輸入一個命令，以 end 作結，之後執行到這裡都會執行所有命令。同一個中斷點有變動其 command 時會完全按照新的輸入。

2 字串

2.1 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=0;
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 }

```

2.2 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 }

```

2.3 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 failureFunction() { // 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 }

```

2.4 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() && s[z[i]]==s[i+z[i]]) z[i]++;
12        }
13        if(i+z[i]-1 > r) l = i, r = i+z[i]-1;
14    }
15 }

```

2.5 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 }

```

3 math

3.1 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. Catalan 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

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

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

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，

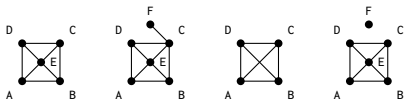
其面積 A 和內部格點數目 i 、邊上格點數目 b 的關係為

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

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)、(2) \bigcirc ；(3) \times ， \overline{AC} 與 \overline{BD} 相交；(4) \times ，非連通圖

7. Simpson Integral

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

3.2 Rational

```

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 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] = (ll)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 = (ll)invFact[b] * invFact[a-b] % mod;
39     return x * y % mod;
40 }

```

3.4 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<=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 }

```

3.3 乘法逆元、組合數

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

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

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;
```

```

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 }

```

3.5 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 problem :
18 給定整數 N，求N最少可以拆成多少個質數的和。
19 如果N是質數，則答案為 1。
20 如果N是偶數(N!=2)，則答案為2(強歌德巴赫猜想)。
21 如果N是奇數且N-2是質數，則答案為2(2+質數)。
22 其他狀況答案為 3 (弱歌德巴赫猜想)。
23 bool isPrime(int n){
24     for(int i=2;i<n;++i){
25         if(i*i>n) return true;
26         if(n%i==0) return false;
27     }
28     return true;
29 }
30 int main(){
31     int n;
32     cin>>n;
33     if(isPrime(n)) cout<<"1\n";
34     else if(n%2==0||isPrime(n-2)) cout<<"2\n";
35     else cout<<"3\n";
36 }

```

3.6 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 }

```

3.7 Extended GCD

```

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 }

```

3.8 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
5    $B^0, B^1, B^2, \dots, B^{P-1}$ ，...，
6 也就是說如果有解則  $L < N$ ，枚舉 0,1,2,L-1
7 能得到結果，但會超時。
8 將 L 拆成  $mx+y$ ，只要分別枚舉 x,y 就能得到答案，
9 設  $m=\sqrt{P}$  能保證最多枚舉  $\sqrt{2P}$  次。
10  $B^*(mx+y) \equiv N \pmod{P}$ 
11  $B^y \equiv N(B^*-m)^x \pmod{P}$ 
12 先求出  $B^0, B^1, B^2, \dots, B^{(m-1)}$ ，
13 再枚舉  $N(B^*(-m)), N(B^*(-m))^2, \dots$  查看是否有對應的
14  $B^y$ 。
15 這種算法稱為大步小步演算法，
16 大步指的是枚舉 x (一次跨 m 步)，
17 小步指的是枚舉 y (一次跨 1 步)。
18 複雜度分析
19 利用 map/unorder_map 存放
20    $B^0, B^1, B^2, \dots, B^{(m-1)}$ ，
21 枚舉 x 查詢 map/unorder_map 是否有對應的  $B^y$ ，
22 存放和查詢最多  $\sqrt{2P}$  次，時間複雜度為
23    $O(\sqrt{P} \log P) / O(\sqrt{P})$ 。
24
25 using LL = long long;
26 LL B, N, P;
27 LL fpow(LL a, LL b, LL c){
28     LL res=1;
29     for(;b>=1;){
30         if(b&1) res=(res*a)%c;
31         a=(a*a)%c;
32         b>>=1;
33     }
34     return res;
35 }
36 LL BSGS(LL a,LL b,LL p){
37     a%=p,b%=p;
38     if(a==0) return b==0?1:-1;
39     if(b==1) return 0;
40     map<LL, LL> tb;
41     LL sq=ceil(sqrt(p-1));
42     LL inv=fpow(a,p-sq-1,p);
43     tb[1]=sq;
44     for(LL i=1,tmp=1;i<sq;++i){
45         tmp=(tmp*a)%p;
46         if(!tb.count(tmp)) tb[tmp]=i;
47     }
48     for(LL i=0;i<sq;++i){
49         if(tb.count(b)){
50             LL res=tb[b];
51             return i*sq+(res==sq?0:res);
52         }
53         b=(b*inv)%p;
54     }
55     return -1;
56 }
57 int main(){
58     IOS; //輸入優化
59     while(cin>>P>>B>>N){
60         LL ans=BSGS(B,N,P);
61         if(ans!=-1) cout<<"no solution\n";
62         else cout<<ans<<"\n";
63     }
64 }

```

3.9 Pisano Period

```

1  /*Pisano Period:
2     費氏數列在 $\text{mod } n$ 的情況下會有循環週期，
3     且週期的結束判斷會在
4      $\text{fib}[i - 1] == 0 \ \&\& \ \text{fib}[i] == 1$ 時，
5     此時循環週期長度是  $i - 1$ 
6     Pisano period可證一個週期的長度會在  $[n, n^2]$ 之間
7      $\text{mod } 1$ 都等於0，沒有週期*/

```

3.10 矩陣快速冪

```

1  using ll = long long;
2  using mat = vector<vector<ll>>;
3  const int mod = 1e9 + 7;
4  mat operator*(mat A, mat B) {
5      mat res(A.size(), vector<ll>(B[0].size()));
6      for(int i=0; i<A.size(); i++) {
7          for(int j=0; j<B[0].size(); j++) {
8              for(int k=0; k<B.size(); k++) {
9                  res[i][j] += A[i][k] * B[k][j] % mod;
10                 res[i][j] %= mod;
11             }
12         }
13     }
14     return res;
15 }
16 // 迴圈版本
17 mat mpow(mat M, int n) {
18     mat res(M.size(), vector<ll>(M[0].size()));
19     for(int i=0; i<res.size(); i++)
20         res[i][i] = 1;
21     for(; n; n>>=1) {
22         if(n & 1) res = res * M;
23         M = M * M;
24     }
25     return res;
26 }

```

4 algorithm

4.1 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
  位數，
7 扣除高位數的影響較扣除低位數的大。
8 //code
9 int main(){
10     string s;
11     int k;
12     cin>>s>>k;
13     for(int i=0;i<k;++i){
14         if((int)s.size()==0) break;
15         int pos =(int)s.size()-1;
16         for(int j=0;j<(int)s.size()-1;++j){
17             if(s[j]>s[j+1]){
18                 pos=j;
19                 break;
20             }
21         }
22         s.erase(pos,1);
23     }
24     while((int)s.size()>0&&s[0]=='0')
25         s.erase(0,1);
26     if((int)s.size()) cout<<s<<'\n';
27     else cout<<0<<'\n';
28 }
29 最小區間覆蓋長度
30 //problem
31 給定 n 條線段區間為 [Li,Ri]，
32 請問最少要選幾個區間才能完全覆蓋 [0,S]?
33 //solution
34 先將所有區間依照左界由小到大排序，
35 對於當前區間 [Li,Ri]，要從左界 >Ri 的所有區間中，
36 找到有著最大的右界的區間，連接當前區間。
37 //problem
38 長度 n 的直線中有數個加熱器，
39 在 x 的加熱器可以讓 [x-r,x+r] 內的物品加熱，
40 問最少要幾個加熱器可以把 [0,n] 的範圍加熱。
41 //solution
42 對於最左邊沒加熱的點a，選擇最遠可以加熱a的加熱器，
43 更新已加熱範圍，重複上述動作繼續尋找加熱器。
44 //code
45 int main(){
46     int n, r;
47     int a[1005];
48     cin>>n>>r;
49     for(int i=1;i<=n;++i) cin>>a[i];
50     int i=1,ans=0;
51     while(i<=n){
52         int R=min(i+r-1,n),L=max(i-r+1,0)
53         int nextR=-1;
54         for(int j=R;j>=L;--j){
55             if(a[j]){
56                 nextR=j;
57                 break;
58             }
59         }
60         if(nextR!=-1){
61             ans=-1;
62             break;
63         }
64         ++ans;
65         i=nextR+r;
66     }
67     cout<<ans<<'\n';
68 }
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 //problem
105 最小化最大延遲問題
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 maxLi 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 //problem
135 最少延遲數量問題
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 → 烏龜重量
149 工作期限 → 烏龜可承受重量
150 多少工作不延期 → 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 }
178 任務調度問題
179 //problem
180 給定 N 項工作，每項工作的需要處理時長為 Ti，
181 期限是 Di，如果第 i 項工作延遲需要受到 pi
182 單位懲罰，
183 請問最少會受到多少單位懲罰。
184 //solution
185 依照懲罰由大到小排序，
186 每項工作依序嘗試可不可以放在
187     Di-Ti+1,Di-Ti,...,1,0，
188 如果有空間就放進去，否則延後執行。
189 //problem
190 給定 N 項工作，每項工作的需要處理時長為 Ti，
191 期限是 Di，如果第 i 項工作在期限內完成會獲得 ai
192 單位獎勵，
193 請問最多會獲得多少單位獎勵。
194 //solution
195 和上題相似，這題變成依照獎勵由大到小排序。
196 //code
197 struct Work{
198     int d,p;
199     bool operator<(const Work &rhs)const{
200         return p>rhs.p;
201     }
202 };
203 int main(){
204     int n;
205     Work a[100005];
206     bitset<100005> ok;
207     while(cin>>n){
208         ok.reset();
209         for(int i=0;i<n;++i)
210             cin>>a[i].d>>a[i].p;
211         sort(a,a+n);
212         int ans=0;
213         for(int i=0;i<n;++i){
214             int j=a[i].d;
215             while(j--){
216                 if(!ok[j]){
217                     ans+=a[i].p;
218                     ok[j]=true;
219                     break;
220                 }
221             }
222         }
223     }
224     cout<<ans<<'\n';
225 }

```


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

```

4.3 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     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 }

```

4.4 三分搜

```

1 題意：給定兩射線方向和速度，問兩射線最近距離。
2 題解
3 假設 F(t) 為兩射線在時間 t 的距離，
4 F(t) 為二次函數，可用三分搜找二次函數最小值。
5 struct Point{
6     double x, y, z;
7     Point() {}
8     Point(double _x, double _y, double _z):
9         x(_x),y(_y),z(_z){}
10    friend istream& operator>>(istream& is,
11        Point& p) {
12        is >> p.x >> p.y >> p.z;
13        return is;
14    }
15    Point operator+(const Point &rhs) const
16    {return Point(x+rhs.x,y+rhs.y,z+rhs.z);}
17    Point operator-(const Point &rhs) const
18    {return Point(x-rhs.x,y-rhs.y,z-rhs.z);}
19    Point operator*(const double &d) const
20    { return Point(x*d, y*d, z*d); }
21    Point operator/(const double &d) const
22    { return Point(x/d, y/d, z/d); }
23    double dist(const Point &rhs) const {
24        double res = 0;
25        res+=(x-rhs.x)*(x-rhs.x);
26        res+=(y-rhs.y)*(y-rhs.y);
27        res+=(z-rhs.z)*(z-rhs.z);
28        return res;
29    }
30 }
31 int main(){
32     IOS; //輸入優化
33     int T;
34     cin>>T;
35     for(int ti=1;ti<=T;++ti){
36         double time;
37         Point x1,y1,d1,x2,y2,d2;
38         cin>>time>>x1>>y1>>x2>>y2;
39         d1=(y1-x1)/time;
40         d2=(y2-x2)/time;
41         double L=0,R=1e8,m1,m2,f1,f2;
42         double ans = x1.dist(x2);
43         while(abs(L-R)>1e-10){
44             m1=(L+R)/2;
45             m2=(m1+R)/2;
46             f1=((d1*m1)+x1).dist((d2*m1)+x2);
47             f2=((d1*m2)+x1).dist((d2*m2)+x2);
48             ans = min(ans,min(f1,f2));
49             if(f1<f2) R=m2;
50             else L=m1;
51         }
52         cout<<"Case "<<ti<<": ";
53         cout << fixed << setprecision(4) <<
54             sqrt(ans) << '\n';
55     }
56 }
57
58 //oi wiki模板, [l, r]
59 //只要是單峰函數，三分可找最大或最小，以下為最小化
60 //計算lmid以及rmid時要避免數字溢出
61 while (r - l > eps) {
62     mid = (l + r) / 2;
63     lmid = mid - eps;
64     rmid = mid + eps;
65     if (f(lmid) < f(rmid)) r = mid;
66     else l = mid;
67 }

```

4.5 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 }

```

4.6 dijkstra

```

1 struct edge{
2     int v,w;
3 };
4
5 struct Item{
6     int u,dis;
7     bool operator<(const Item &rhs)const{
8         return dis>rhs.dis;
9     }
10 };
11
12 vector<edge> G[maxn];
13 int dist[maxn];
14
15 void dijkstra(int s){ // O((V + E)log(E))
16     memset(dist,INF,sizeof(dist));
17     dist[s]=0;
18     priority_queue<Item> pq;
19     pq.push({s,0});
20     while(!pq.empty()){
21         Item now=pq.top();
22         pq.pop();
23         if(now.dis>dist[now.u]) continue;
24         for(edge e:G[now.u]){
25             if(dist[e.v]>dist[now.u]+e.w){
26                 dist[e.v]=dist[now.u]+e.w;
27                 pq.push({e.v,dist[e.v]});
28             }
29         }
30     }
31 }
32
33 int main(){
34     int t,cas=1;
35     cin>>t;
36     while(t--){
37         int n,m,s,t;
38         cin>>n>>m>>s>>t;
39         for(int i=0;i<=n;i++) G[i].clear();
40         int u,v,w;
41         for(int i=0;i<m;i++){
42             cin>>u>>v>>w;
43             G[u].push_back({v,w});
44             G[v].push_back({u,w});
45         }
46         dijkstra(s);
47         cout<<"Case #"<<cas++<<"<<" ";
48         if(dist[t]==INF)
49             cout<<"unreachable\n";
50         else cout<<dist[t]<<endl;
51     }
52 }

```

4.7 SPFA

```

1 struct Edge{
2     int t;
3     long long w;
4     Edge(){};
5     Edge(int _t, long long _w) : t(_t),
6         w(_w) {}
7 };
8
9 bool SPFA(int st) // 平均O(V + E) 最糟O(VE)
10 {
11     vector<int> cnt(n, 0);
12     bitset<MXV> inq(0);
13     queue<int> q;
14     q.push(st);
15     dis[st] = 0;
16     inq[st] = true;
17     while (!q.empty()){
18         int cur = q.front();
19         q.pop();
20         inq[cur] = false;
21         for (auto &e : G[cur]){
22             if (dis[e.t] <= dis[cur] + e.w)
23                 continue;
24             dis[e.t] = dis[cur] + e.w;
25             if (inq[e.t]) continue;
26             ++cnt[e.t];
27             if (cnt[e.t] > n)
28                 return false; // negative cycle
29             inq[e.t] = true;
30             q.push(e.t);
31         }
32     }
33     return true;
34 }

```

4.8 SCC Kosaraju

```

1 //做兩次dfs, O(V + E)
2 //g 是原圖, g2 是反圖
3 //s是dfs離開的節點
4 void dfs1(int u) {
5     vis[u] = true;
6     for (int v : g[u])
7         if (!vis[v]) dfs1(v);
8     s.push_back(u);
9 }
10
11 void dfs2(int u) {
12     group[u] = sccCnt;
13     for (int v : g2[u])
14         if (!group[v]) dfs2(v);
15 }
16
17 void kosaraju() {
18     sccCnt = 0;
19     for (int i = 1; i <= n; ++i)
20         if (!vis[i]) dfs1(i);
21     for (int i = n; i >= 1; --i)
22         if (!group[s[i]]) {
23             ++sccCnt;
24             dfs2(s[i]);
25         }
26 }

```

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

```


4.10 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,
    e[tot].nt = hd[u], hd[u] = tot; }
10 void uadd(int u, int v) { add(u, v), add(v, u); }
11 bool bz[M << 1];
12 int bcc_cnt, dfn[N], low[N], vis_bcc[N];
13 vector<vector<int>> bcc;
14 void tarjan(int x, int in) {
15     dfn[x] = low[x] = ++bcc_cnt;
16     for (int i = hd[x]; i; i = e[i].nt) {
17         int v = e[i].to;
18         if (dfn[v] == 0) {
19             tarjan(v, i);
20             if (dfn[x] < low[v])
21                 bz[i] = bz[i ^ 1] = true;
22             low[x] = min(low[x], low[v]);
23         } else if (i != (in ^ 1))
24             low[x] = min(low[x], dfn[v]);
25     }
26 }
27 void dfs(int x, int id) {
28     vis_bcc[x] = id, bcc[id - 1].push_back(x);
29     for (int i = hd[x]; i; i = e[i].nt) {
30         int v = e[i].to;
31         if (vis_bcc[v] || bz[i]) continue;
32         dfs(v, id);
33     }
34 }

```

4.11 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,
    hd[u]}, hd[u] = tot; }
13
14 void uadd(int u, int v) { add(u, v), add(v, u); }
15
16 int ans;
17 int dfn[N], low[N], bcc_cnt;
18 int sta[N], top, cnt;
19 bool cut[N];
20 vector<int> dcc[N];
21 int root;
22
23 void tarjan(int u) {
24     dfn[u] = low[u] = ++bcc_cnt, sta[++top] = u;
25     if (u == root && hd[u] == 0) {
26         dcc[++cnt].push_back(u);
27         return;
28     }
29     int f = 0;
30     for (int i = hd[u]; i; i = e[i].nt) {
31         int v = e[i].to;
32         if (!dfn[v]) {
33             tarjan(v);
34             low[u] = min(low[u], low[v]);
35             if (low[v] >= dfn[u]) {
36                 if (++f > 1 || u != root)
37                     cut[u] = true;
38                 cnt++;
39                 dcc[cnt].push_back(sta[top--]);
40                 while (sta[top + 1] != v);
41                 dcc[cnt].push_back(u);
42             }
43         } else
44             low[u] = min(low[u], dfn[v]);
45     }
46 }

```

4.12 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 }

```

4.13 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                 if(cur == root) break;
37                 vis[cur] = i;
38                 cur = pre[cur];
39             }
40             if(cur != root && idx[cur] == -1) {
41                 for(int j=pre[cur]; j!=cur; j=pre[j])
42                     idx[j] = seq;
43                 idx[cur] = seq++;
44             }
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 }

```

4.14 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             if (i != 0) printf("%d", Lx[i]);
72             else printf("%d", Lx[i]);
73             res += Lx[i];
74         }
75     }
76 }

```

```

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

```

4.15 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 }

```

4.16 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 }

```

4.17 莫隊

```

1  /*利用prefix前綴XOR和
2   如果要求[x, y]的XOR和只要回答prefix[y] ^
   prefix[x - 1]即可在O(1)回答
3   同時維護cnt[i]代表[x, y]XOR和 == i的個數
4   如此我們知道[l, r]可以快速知道[l - 1, r], [l
   + 1, r], [l, r - 1], [l, r + 1]的答案
5   就符合Mo's algorithm的思維O(N * sqrt(n))
6   每次轉移為O(1), 具體轉移方法在下面*/
7  #define maxn 100005
8  //在此prefix[i]是[1, i]的XOR和
9  int prefix[maxn];
10 //log_2(1000000) =
   19.931568569324174087221916576937...
11 //所以開到1 << 20
12 //cnt[i]代表的是有符合nums[x, y] such that
   nums[x] ^ nums[x + 1] ^ .. ^ nums[y] ==
   i
13 //的個數
14 long long cnt[1 << 20];
15 //塊大小 -> sqrt(n)
16 int sqrtQ;
17 struct Query {
18     int l, r, id;
19     bool operator < (const Query& other)
       const {
20         if (this->l / sqrtQ != other.l /
           sqrtQ)
21             return this->l < other.l;
22         //奇偶排序(優化)
23         if (this->l / sqrtQ & 1)
24             return this->r < other.r;
25         return this->r > other.r;
26     }
27 };
28 Query queries[maxn];
29 long long ans[maxn];
30 long long res = 0;
31 int k;
32 void add(int x) {
33     res += cnt[k ^ prefix[x]];
34     ++cnt[prefix[x]];
35 }
36 void sub(int x) {
37     --cnt[prefix[x]];
38     res -= cnt[k ^ prefix[x]];
39 }
40 int main() {
41     int n, m;
42     scanf("%d %d %d", &n, &m, &k);
43     sqrtQ = sqrt(n);
44     for (int i = 1; i <= n; ++i) {
45         scanf("%d", &prefix[i]);
46         prefix[i] ^= prefix[i - 1];
47     }
48     for (int i = 1; i <= m; ++i) {
49         scanf("%d %d", &queries[i].l,
           &queries[i].r);
50         //減1是因為prefix[i]是[1,
           i]的前綴XOR和, 所以題目問[l,
           r]我們要回答[l - 1, r]的答案
51         --queries[i].l;
52         queries[i].id = i;
53     }
54     sort(queries + 1, queries + m + 1);
55     int l = 1, r = 0;
56     for (int i = 1; i <= m; ++i) {
57         while (l < queries[i].l) {
58             sub(l);
59             ++l;
60         }
61         while (l > queries[i].l) {
62             --l;
63             add(l);
64         }
65         while (r < queries[i].r) {
66             ++r;

```

```

67         add(r);
68     }
69     while (r > queries[i].r) {
70         sub(r);
71         --r;
72     }
73     ans[queries[i].id] = res;
74 }
75 for (int i = 1; i <= m; ++i){
76     printf("%lld\n", ans[i]);
77 }
78 return 0;
79 }

```

4.18 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) &&
               (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 }
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 = edges[parent[curr]].u;

```

```

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,
               sellPrice, sellMax, inventoryMonth;
84             scanf("%d %d %d %d", &produceCost,
               &produceMax, &sellPrice,
               &sellMax, &inventoryMonth);
85             addEdge(s, i, produceMax, produceCost);
86             addEdge(M + i, t, sellMax, -sellPrice);
87             for (int j = 0; j <= inventoryMonth; ++j) {
88                 if (i + j <= M)
89                     addEdge(i, M + i + j, INF, I * j);
90             }
91         }
92         printf("Case %d: %lld\n", Case, -MCMF());
93     }
94 }
95 return 0;
96 }

```

4.19 Blossom Algorithm

```

1 const int maxn = 500 + 10;
2
3 struct Edge { int s, t; };
4
5 int n;
6 int base[maxn], match[maxn], p[maxn], inq[maxn];
7 bool vis[maxn], flower[maxn];
8 vector<Edge> G[maxn];
9 queue<int> q;
10
11 int lca(int a, int b) {
12     memset(vis, 0, sizeof(vis));
13     while(1) {
14         a = base[a];
15         vis[a] = true;
16         if(match[a] == -1) break;
17         a = p[match[a]];
18     }
19     while(1) {
20         b = base[b];
21         if(vis[b]) return b;
22         b = p[match[b]];
23     }
24     return -1;
25 }
26
27 void set_path(int x, int father) {
28     int tmp;
29     while(x != father) {
30         tmp = match[x];
31         flower[base[x]] = flower[base[tmp]] = 1;
32         tmp = p[tmp];
33         if(base[tmp] != father) p[tmp] = match[x];
34         x = tmp;
35     }
36 }
37
38 void blossom(int x, int y) {
39     memset(flower, 0, sizeof(flower));
40     int father = lca(x, y);
41     set_path(x, father);
42     set_path(y, father);
43     if(base[x] != father) p[x] = y;
44     if(base[y] != father) p[y] = x;
45     for(int i=1; i<=n; i++) {
46         if(!flower[base[i]]) continue;
47         base[i] = father;
48         if(!inq[i]) {
49             q.push(i);
50             inq[i] = true;
51         }
52     }
53 }
54
55 bool bfs(int root) {
56     int cur, y, nxt;
57     q = queue<int>();
58     q.push(root);
59     memset(inq, 0, sizeof(inq));
60     memset(p, -1, sizeof(p));
61     for(int i=1; i<=n; i++) base[i] = i;
62
63     while(!q.empty()) {
64         cur = q.front();
65         q.pop();
66         inq[cur] = false;
67
68         for(auto e : G[cur]) {
69             if(base[e.s] == base[e.t]) continue;
70             if(match[e.s] == e.t) continue;
71             if(e.t == root ||
72                (~match[e.t] && ~p[match[e.t]])) {
73                 blossom(cur, e.t);
74             } else if(p[e.t] == -1) {
75                 p[e.t] = cur;
76                 if(match[e.t] == -1) {

```

```

77         cur = e.t;
78         while(cur != -1) {
79             y = p[cur];
80             nxt = match[y];
81             match[cur] = y;
82             match[y] = cur;
83             cur = nxt;
84         }
85         return true;
86     } else {
87         q.push(match[e.t]);
88         inq[match[e.t]] = true;
89     }
90 }
91 }
92 }
93 return false;
94 }
95
96 int maxMatch() {
97     int res = 0;
98     memset(match, -1, sizeof(match));
99     for(int i=1; i<=n; i++) {
100         if(match[i] == -1 && bfs(i)) res++;
101     }
102     return res;
103 }

```

4.20 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]=0]=0;
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,
19        ++colSize[c];
20        U[seq]=c, D[seq]=D[c], U[D[c]]=seq,
21        D[c]=seq;
22        if(rowHead[r]) {
23            L[seq]=rowHead[r],
24            R[seq]=R[rowHead[r]];
25            L[R[rowHead[r]]]=seq,
26            R[rowHead[r]]=seq;
27        } else {
28            rowHead[r] = L[seq] = R[seq] =
29            seq;
30        }
31    }
32    void remove(int c) {
33        L[R[c]] = L[c], R[L[c]] = R[c];
34        for(int i=D[c]; i!=c; i=D[i]) {
35            for(int j=R[i]; j!=i; j=R[j]) {
36                U[D[j]] = U[j];
37                D[U[j]] = D[j];
38                --colSize[col[j]];
39            }
40        }
41    }
42    void recover(int c) {
43        for(int i=U[c]; i!=c; i=U[i]) {
44            for(int j=L[i]; j!=i; j=L[j]) {
45                U[D[j]] = D[U[j]] = j;
46                ++colSize[col[j]];
47            }
48        }
49    }
50 }

```

```

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 }

```

4.21 Astar

```

1 /*A*求k短路
2  $f(x) = g(x) + h(x)$ 
3  $g(x)$  是實際cost,  $h(x)$  是估計cost
4 在此 $h(x)$ 用所有點到終點的最短距離, 則當用Astar找點
5 當該點 $cnt[u] == k$ 時即得到該點的第k短路
6 */
7 #define maxn 105
8 struct Edge { int u, v, w; };
9 struct Item_pqH {
10     int u, w;
11     bool operator < (const Item_pqH& other)
12     const {
13         return this->w > other.w;
14     }
15 };
16 struct Item_aster {
17     int u, g, f;
18     bool operator < (const Item_aster& other)
19     const {
20         return this->f > other.f;
21     }
22 };
23 vector<vector<Edge>> G;
24 //反向圖, 用於建 $h(u)$ 
25 vector<vector<Edge>> invertG;
26 int h[maxn];
27 bool visited[maxn];
28 int cnt[maxn];

```

4.22 LCA 倍增法

4.23 LCA 樹壓平 RMQ

```

27 //用反向圖去求出每一點到終點的最短距離，並以此當作h(u)
28 void dijkstra(int s, int t) {
29     memset(visited, 0, sizeof(visited));
30     priority_queue<Item_pqh> pq;
31     pq.push({s, 0});
32     h[s] = 0;
33     while (!pq.empty()) {
34         Item_pqh curr = pq.top();
35         pq.pop();
36         visited[curr.u] = true;
37         for (Edge& edge: invertG[curr.u]) {
38             if (!visited[edge.v]) {
39                 if (h[edge.v] > h[curr.u] +
40                     edge.w) {
41                     h[edge.v] = h[curr.u] +
42                         edge.w;
43                     pq.push({edge.v,
44                             h[edge.v]});
45                 }
46             }
47         }
48     }
49     int Astar(int s, int t, int k) {
50         memset(cnt, 0, sizeof(cnt));
51         priority_queue<Item_astar> pq;
52         pq.push({s, 0, h[s]});
53         while (!pq.empty()) {
54             Item_astar curr = pq.top();
55             pq.pop();
56             ++cnt[curr.u];
57             //終點出現k次，此時即可得k短路
58             if (cnt[t] == k)
59                 return curr.g;
60             for (Edge& edge: G[curr.u]) {
61                 if (cnt[edge.v] < k) {
62                     pq.push({edge.v, curr.g +
63                             edge.w + edge.w
64                             + h[edge.v]});
65                 }
66             }
67         }
68         return -1;
69     }
70     int main() {
71         int n, m;
72         while (scanf("%d %d", &n, &m) && (n != 0
73             && m != 0)) {
74             G.assign(n + 5, vector<Edge>());
75             invertG.assign(n + 5, vector<Edge>());
76             int s, t, k;
77             scanf("%d %d %d", &s, &t, &k);
78             int u, v, w;
79             for (int i = 0; i < m; ++i) {
80                 scanf("%d %d %d", &u, &v, &w);
81                 G[u].emplace_back(Edge{u, v, w});
82                 invertG[v].emplace_back(Edge{v,
83                     u, w});
84             }
85             memset(h, 0x3f, sizeof(h));
86             dijkstra(t, s);
87             printf("%d\n", Astar(s, t, k));
88         }
89         return 0;
90     }

```

```

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]) swap(x, y);
31         int deltaDep = dep[y] - dep[x];
32         long long res = 0;
33         //讓y與x在同一個深度
34         for (int i = 0; deltaDep != 0; ++i,
35             deltaDep >>= 1)
36             if (deltaDep & 1)
37                 res += dis[y][i], y = fa[y][i];
38         if (y == x) //x = y -> x、y彼此是彼此的祖先
39             return res;
40         //往上找，一起跳，但x、y不能重疊
41         for (int i = 30; i >= 0 && y != x; --i) {
42             if (fa[x][i] != fa[y][i]) {
43                 res += dis[x][i] + dis[y][i];
44                 x = fa[x][i];
45                 y = fa[y][i];
46             }
47         }
48         //最後發現不能跳了，此時x的第2^0 = 1個祖先
49         // (或說y的第2^0 = 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+1),
69                     (q ? ' ': '\n'));
70             }
71         }
72     }

```

```

1 //樹壓平求LCA RMQ(sparse table
2 //O(nlogn)建立，O(1)查詢)，求任意兩點距離，
3 //如果用笛卡兒樹可以壓到O(n)建立，O(1)查詢
4 //理論上可以過，但遇到直鏈的case dfs深度會stack
5 //overflow
6 #define maxn 100005
7 struct Edge {
8     int u, v, w;
9 };
10 int dep[maxn], pos[maxn];
11 long long dis[maxn];
12 int st[maxn * 2][32]; //sparse table
13 int realLCA[maxn * 2][32];
14 //最小深度對應的節點，及真正的LCA
15 int Log[maxn]; //取代std::log2
16 int tp; // timestamp
17 vector<vector<Edge>> G; // tree
18 void calLog() {
19     Log[1] = 0;
20     Log[2] = 1;
21     for (int i = 3; i < maxn; ++i)
22         Log[i] = Log[i / 2] + 1;
23 }
24 void buildST() {
25     for (int j = 0; Log[tp]; ++j) {
26         for (int i = 0; i + (1 << j) - 1 < tp;
27             ++i) {
28             if (st[i-1][j] < st[i-1][j] + (1 <<
29                 i-1)) {
30                 st[i][j] = st[i-1][j];
31                 realLCA[i][j] = realLCA[i-1][j];
32             }
33             else {
34                 st[i][j] = st[i-1][j] + (1 << i-1);
35                 realLCA[i][j] = realLCA[i-1][j] + (1
36                     << i-1);
37             }
38         }
39     } // O(nlogn)
40     int query(int l, int r) { // [l, r] min
41         //depth即為lca的深度
42         int k = Log[r - l + 1];
43         if (st[l][k] < st[r - (1 << k) + 1][k])
44             return realLCA[l][k];
45         else
46             return realLCA[r - (1 << k) + 1][k];
47     }
48     void dfs(int u, int p) { //euler tour
49         pos[u] = tp;
50         st[tp][0] = dep[u];
51         realLCA[tp][0] = dep[u];
52         ++tp;
53         for (int i = 0; i < G[u].size(); ++i) {
54             Edge& edge = G[u][i];
55             if (edge.v == p) continue;
56             dep[edge.v] = dep[u] + 1;
57             dis[edge.v] = dis[edge.u] + edge.w;
58             dfs(edge.v, u);
59             st[tp++][0] = dep[u];
60         }
61     }
62     long long getDis(int u, int v) {
63         if (pos[u] > pos[v])
64             swap(u, v);
65         int lca = query(pos[u], pos[v]);
66         return dis[u] + dis[v] - 2 *
67             dis[query(pos[u], pos[v])];
68     }
69     int main() {
70         int n, q;
71         calLog();
72         while (~scanf("%d", &n) && n) {
73             int v, w;
74             G.assign(n + 5, vector<Edge>());

```



```

68     tp = 0;
69     for (int i = 1; i <= n - 1; ++i) {
70         scanf("%d %d", &v, &w);
71         G[i].push_back({i, v, w});
72         G[v].push_back({v, i, w});
73     }
74     dfs(0, -1);
75     buildST();
76     scanf("%d", &q);
77     int u;
78     while (q--) {
79         scanf("%d %d", &u, &v);
80         printf("%lld%c", getDis(u, v),
81             (q) ? ' ' : '\n');
82     }
83     return 0;
84 }

```

4.24 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(經過的邊)
82             int dis = depth[u] + depth[v] - 2 *
83                 depth[lca];
84             //讓v比u深或等於
85             if (depth[u] > depth[v]) swap(u, v);
86             if (u == v) {
87                 printf("The fleas meet at %d.\n", u);
88             }
89             else if (dis % 2 == 0) {
90                 //路徑長是even -> 有中點
91                 printf("The fleas meet at %d.\n",
92                     getK_parent(v, dis / 2));
93             }
94             else {
95                 //路徑長是odd -> 沒有中點
96                 if (depth[u] == depth[v]) {
97                     int x = getK_parent(u, dis / 2);
98                     int y = getK_parent(v, dis / 2);
99                     if (x > y) swap(x, y);
100                     printf("The fleas jump forever
101                         between %d and %d.\n", x, y);
102                 }
103                 else {
104                     //技巧: 讓深的點v往上dis / 2步 = y,
105                     //這個點的parent設為x
106                     //此時的x、y就是答案要的中點兩點
107                     //主要是往下不好找, 所以改用深的點用parent往上
108                     int y = getK_parent(v, dis / 2);
109                     int x = getK_parent(y, 1);
110                     if (x > y) swap(x, y);
111                     printf("The fleas jump forever
112                         between %d and %d.\n", x, y);
113                 }
114             }
115         }
116     }
117 }

```


5 DataStructure

5.1 BIT

```

1 template <class T> class BIT {
2 private:
3     int size;
4     vector<T> bit;
5     vector<T> arr;
6
7 public:
8     BIT(int sz=0):
9         size(sz), bit(sz+1), arr(sz) {}
10
11     /** Sets the value at index idx to val. */
12     void set(int idx, T val) {
13         add(idx, val - arr[idx]);
14     }
15
16     /** Adds val to the element at index idx. */
17     void add(int idx, T val) {
18         arr[idx] += val;
19         for(++idx; idx<=size; idx+=(idx & -idx))
20             bit[idx] += val;
21     }
22
23     /** The sum of all values in [0, idx]. */
24     T pre_sum(int idx) {
25         T total = 0;
26         for(++idx; idx>0; idx--=(idx & -idx))
27             total += bit[idx];
28         return total;
29     }
30 };

```

5.2 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 }

```

5.3 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 };

```

5.4 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     }
49     // 要存 wordId 才要 vec
50     // 同單字重複match要把所有vis取消掉
51     int match(char* s, vector<int>& vec) {
52         int res = 0;
53         memset(vis, 0, sizeof(vis));
54         for(int p=root; *s; s++) {
55             p = trie[p][*s-minc];
56             for(int k=p; k && !vis[k]; k=fail[k]) {
57                 vis[k] = true;
58                 res += cnt[k];
59                 if(cnt[k]) vec.push_back(val[k]);
60             }
61         }
62         return res; // 匹配到的單字量
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)

```

5.5 線段樹 1D

5.6 線段樹 2D

```

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 }
67 //如果是直接改值而不是加值，query與update中的tag與st
68 //改值從+=改成=

```

```

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 N;
4 void modifyY(int index, int l, int r, int val,
5     int yPos, int xIndex, bool xIsLeaf) {
6     if (l == r) {
7         if (xIsLeaf) {
8             maxST[xIndex][index] =
9                 minST[xIndex][index] = val;
10            return;
11        }
12        maxST[xIndex][index] =
13            max(maxST[xIndex * 2][index],
14                maxST[xIndex * 2 + 1][index]);
15        minST[xIndex][index] =
16            min(minST[xIndex * 2][index],
17                minST[xIndex * 2 + 1][index]);
18    }
19    else {
20        int mid = (l + r) / 2;
21        if (yPos <= mid)
22            modifyY(index * 2, l, mid, val, yPos,
23                xIndex, xIsLeaf);
24        else
25            modifyY(index * 2 + 1, mid + 1, r, val,
26                yPos, xIndex, xIsLeaf);
27        maxST[xIndex][index] =
28            max(maxST[xIndex * 2][index],
29                maxST[xIndex * 2 + 1][index]);
30        minST[xIndex][index] =
31            min(minST[xIndex * 2][index],
32                minST[xIndex * 2 + 1][index]);
33    }
34 }
35 void modifyX(int index, int l, int r, int
36     val, int xPos, int yPos) {
37     if (l == r) {
38         modifyY(1, 1, N, val, yPos, index, true);
39     }
40     else {
41         int mid = (l + r) / 2;
42         if (xPos <= mid)
43             modifyX(index * 2, l, mid, val, xPos, yPos);
44         else
45             modifyX(index * 2 + 1, mid + 1, r, val,
46                 xPos, yPos);
47         modifyY(1, 1, N, val, yPos, index, false);
48     }
49 }
50 void queryY(int index, int l, int r, int yql,
51     int yqr, int xIndex, int& vmax, int& vmin) {
52     if (yql <= l && r <= yqr) {
53         vmax = max(vmax, maxST[xIndex][index]);
54         vmin = min(vmin, minST[xIndex][index]);
55     }
56     else {
57         int mid = (l + r) / 2;
58         if (yql <= mid)
59             queryY(index * 2, l, mid, yql, yqr,
60                 xIndex, vmax, vmin);
61         if (mid < yqr)
62             queryY(index * 2 + 1, mid + 1, r, yql,
63                 yqr, xIndex, vmax, vmin);
64     }
65 }
66 void queryX(int index, int l, int r, int
67     xql, int xqr, int yql, int yqr, int&
68     vmax, int& vmin) {
69     if (xql <= l && r <= xqr) {
70         queryY(1, 1, N, yql, yqr, index, vmax, vmin);
71     }
72     else {
73         int mid = (l + r) / 2;
74         if (xql <= mid)

```

```

58     queryX(index * 2, l, mid, xql, xqr, yql,
59         yqr, vmax, vmin);
60     if (mid < xqr)
61         queryX(index * 2 + 1, mid + 1, r, xql,
62             xqr, yql, yqr, vmax, vmin);
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,
83                     &xqr, &yqr);
84                 vmax = -0x3f3f3f3f;
85                 vmin = 0x3f3f3f3f;
86                 queryX(1, 1, N, xql, xqr, yql, yqr,
87                     vmax, vmin);
88                 printf("%d %d\n", vmax, vmin);
89             }
90             else {
91                 scanf("%d %d %d", &xql, &yql, &val);
92                 modifyX(1, 1, N, val, xql, yql);
93             }
94         }
95     }
96     return 0;
97 }

```

5.7 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
28             - st[index*2]);
29 }
30 int main() {
31     int t;
32     cin >> t;
33     bool first = true;
34     while (t--) {
35         if (first) first = false;
36         else puts("");
37         memset(st, 0, sizeof(st));
38         int m, n;
39         cin >> m >> n;
40         for (int i = 1; i <= m; ++i) {
41             cin >> nums[i];
42             id[i] = nums[i];
43         }
44         for (int i = 0; i < n; ++i)
45             cin >> getArr[i];
46         //離散化
47         //防止 m == 0
48         if (m) sort(id + 1, id + m + 1);
49         int stSize = unique(id + 1, id + m + 1)
50             - (id + 1);
51         for (int i = 1; i <= m; ++i) {
52             nums[i] = lower_bound(id + 1, id +
53                 stSize + 1, nums[i]) - id;
54         }
55         int addCount = 0, getCount = 0;
56         int k = 1;
57         while (getCount < n) {
58             if (getArr[getCount] == addCount) {
59                 printf("%d\n", query(1, 1, stSize, k));
60                 ++k;
61                 ++getCount;
62             }
63             else {
64                 update(1, 1, stSize, nums[addCount+1]);
65                 ++addCount;
66             }
67         }
68     }
69 }

```

5.8 ChthollyTree

```

1 //重點：要求輸入資料隨機，否則可能被卡時間
2 struct Node {
3     long long l, r;
4     mutable long long val;
5     Node(long long l, long long r, long long
6         val)
7         : l(l), r(r), val(val){}
8     bool operator<(const Node& other) const {
9         return this->l < other.l;
10    };
11    set<Node> chthollyTree;
12    //將[l, r]拆成 [l, pos - 1], [pos, r]
13    set<Node>::iterator split(long long pos) {
14        //找第一個左端點大於等於pos的區間
15        set<Node>::iterator it =
16            chthollyTree.lower_bound(Node(pos,
17                0, 0));
18        //運氣很好直接找到左端點是pos的區間
19        if (it != chthollyTree.end() && it->l ==
20            pos)
21            return it;
22        //到這邊代表找到的是第一個左端點大於pos的區間
23        //it - 1即可找到左端點等於pos的區間
24        //(不會是別的，因為沒有重疊的區間)
25        --it;
26        long long l = it->l, r = it->r;
27        long long val = it->val;
28        chthollyTree.erase(it);
29        chthollyTree.insert(Node(l, pos-1, val));
30        //回傳左端點是pos的區間iterator
31        return chthollyTree.insert(Node(pos, r,
32            val)).first;
33    }
34 }
35 //區間賦值
36 void assign(long long l, long long r, long
37     long val) {
38     //<注意>
39     //end與begin的順序不能調換，因為end的split可能會改
40     //因為end可以在原本begin的區間中
41     set<Node>::iterator end = split(r + 1),
42         begin = split(l);
43     //begin到end全部刪掉
44     chthollyTree.erase(begin, end);
45     //填回去 [l, r]的區間
46     chthollyTree.insert(Node(l, r, val));
47 }
48 //區間加值(直接一個個區間去加)
49 void add(long long l, long long r, long long
50     val) {
51     set<Node>::iterator end = split(r + 1);
52     set<Node>::iterator begin = split(l);
53     for (set<Node>::iterator it = begin; it
54         != end; ++it) {
55         it->val += val;
56     }
57     //查詢區間第k小 -> 直接把每個區間丟去vector排序
58     long long getKthSmallest(long long l, long
59         long r, long long k) {
60         set<Node>::iterator end = split(r + 1);
61         set<Node>::iterator begin = split(l);
62         //pair -> first: val, second: 區間長度
63         vector<pair<long long, long long>> vec;
64         for (set<Node>::iterator it = begin; it
65             != end; ++it) {
66             vec.push_back({it->val, it->r - it->l
67                 + 1});
68         }
69         sort(vec.begin(), vec.end());
70         for (const pair<long long, long long>&
71             p: vec) {
72             k -= p.second;
73             if (k <= 0) return p.first;
74         }
75         //不應該跑到這
76         return -1;
77     }
78 }

```

```

64 }
65 //快速幂
66 long long qpow(long long x, long long n,
67     long long mod) {
68     long long res = 1;
69     x %= mod;
70     while (n) {
71         if (n & 1) res = res * x % mod;
72         n >>= 1;
73         x = x * x % mod;
74     }
75     return res;
76 }
77 //區間n次方和
78 long long sumOfPow(long long l, long long r,
79     long long n, long long mod) {
80     long long total = 0;
81     set<Node>::iterator end = split(r + 1);
82     set<Node>::iterator begin = split(l);
83     for (set<Node>::iterator it = begin; it
84         != end; ++it) {
85         total = (total + qpow(it->val, n, mod) *
86             (it->r - it->l + 1)) % mod;
87     }
88     return total;
89 }

```

5.9 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 }

```

6 Geometry

6.1 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)

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

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

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

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

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

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

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

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

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

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。

6.2 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};
```

6.3 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}
```

6.4 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

6.6 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 }
```

6.7 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 }
```

6.8 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 }
```

6.5 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 }
```


7 DP

7.1 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 c

```
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             c[j] = c[j-w[i]];
6         } else if(dp[j] == dp[j-w[i]]+v[i]) {
7             c[j] += c[j-w[i]];
8         }
9     }
10 }
```

2. 求最大價值的一組方案 p

```
1 memset(p, 0, sizeof(p));
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             p[i] = 1;
7         } else {
8             p[i] = 0;
9         }
10     }
11 }
```

3. 求最大價值的字典序最小的一組方案 p

```
1 // reverse(item), 要把物品順序倒過來
2 memset(p, 0, sizeof(p));
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             p[i] = 1;
8         } else {
9             p[i] = 0;
10         }
11     }
12 }
```


7.2 Range DP

```

1 //區間dp
2 int dp[55][55];
3 // dp[i][j] -> [i,j] 切割區間中最小的cost
4 int cuts[55];
5 int solve(int i, int j) {
6     if (dp[i][j] != -1)
7         return dp[i][j];
8     //代表沒有其他切法，只能是cuts[j] - cuts[i]
9     if (i == j - 1)
10        return dp[i][j] = 0;
11    int cost = 0x3f3f3f3f;
12    for (int m = i + 1; m < j; ++m) {
13        //枚舉區間中間切點
14        cost = min(cost, solve(i, m) +
15            solve(m, j) + cuts[j] - cuts[i]);
16    }
17    return dp[i][j] = cost;
18 }
19 int main() {
20     int l, n;
21     while (scanf("%d", &l) != EOF && l){
22         scanf("%d", &n);
23         for (int i = 1; i <= n; ++i)
24             scanf("%d", &cuts[i]);
25         cuts[0] = 0;
26         cuts[n + 1] = 1;
27         memset(dp, -1, sizeof(dp));
28         printf("ans = %d.\n", solve(0, n+1));
29     }
30     return 0;
31 }

```

7.3 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 }

```

7.4 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

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

7.5 Barcode

```

1 int N, K, M;
2 long long dp[55][55];
3 // n -> 目前剩多少units
4 // k -> 目前剩多少bars
5 // m -> 1 bar最多多少units
6 long long dfs(int n, int k) {
7     if (k == 1) {
8         return (n <= M);
9     }
10    if (dp[n][k] != -1)
11        return dp[n][k];
12    long long result = 0;
13    for (int i = 1; i < min(M + 1, n); ++i)
14        { // < min(M + 1, n)是因為n不能==0
15            result += dfs(n - i, k - 1);
16        }
17    return dp[n][k] = result;
18 }
19 int main() {
20     while (scanf("%d %d %d", &N, &K, &M) !=
21         EOF) {
22         memset(dp, -1, sizeof(dp));
23         printf("%lld\n", dfs(N, K));
24     }
25 }

```

7.6 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

```

7.7 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 }

```

7.8 抽屜

```

1 long long dp[70][70][2];
2 // 初始條件
3 dp[1][0][0] = dp[1][1][1] = 1;
4 for (int i = 2; i <= 66; ++i){
5     // i個抽屜0個安全且上方0 =
6     // (底下i - 1個抽屜且1個安全且最上面L) +
7     // (底下n - 1個抽屜0個安全且最上方為0)
8     dp[i][0][0]=dp[i-1][1][1]+dp[i-1][0][0];
9     for (int j = 1; j <= i; ++j) {
10        dp[i][j][0] =
11            dp[i-1][j+1][1]+dp[i-1][j][0];
12        dp[i][j][1] =
13            dp[i-1][j-1][1]+dp[i-1][j-1][0];
14    }
15 } //答案在 dp[n][s][0] + dp[n][s][1];

```

7.9 TreeDP reroot

7.10 WeightedLIS

```

1  /*re-root dp on tree  $O(n + n + n) \rightarrow O(n)*$ 
2  class Solution {
3  public:
4      vector<int> sumOfDistancesInTree(int n,
5          vector<vector<int>>& edges) {
6          this->res.assign(n, 0);
7          G.assign(n + 5, vector<int>());
8          for (vector<int>& edge: edges) {
9              G[edge[0]].emplace_back(edge[1]);
10             G[edge[1]].emplace_back(edge[0]);
11         }
12         memset(this->visited, 0,
13             sizeof(this->visited));
14         this->dfs(0);
15         memset(this->visited, 0,
16             sizeof(this->visited));
17         this->dfs2(0, 0);
18         return this->res;
19     private:
20         vector<vector<int>> G;
21         bool visited[30005];
22         int subtreeSize[30005];
23         vector<int> res;
24         //求subtreeSize
25         int dfs(int u) {
26             this->visited[u] = true;
27             for (int v: this->G[u])
28                 if (!this->visited[v])
29                     this->subtreeSize[u] +=
30                         this->dfs(v);
31             //自己
32             this->subtreeSize[u] += 1;
33             return this->subtreeSize[u];
34         }
35         //求res[0], 0到所有點的距離
36         int dfs2(int u, int dis) {
37             this->visited[u] = true;
38             int sum = 0;
39             for (int v: this->G[u])
40                 if (!visited[v])
41                     sum += this->dfs2(v, dis + 1);
42             //要加上自己的距離
43             return sum + dis;
44         }
45         //算出所有的res
46         void dfs3(int u, int n) {
47             this->visited[u] = true;
48             for (int v: this->G[u]) {
49                 if (!visited[v]) {
50                     this->res[v] = this->res[u] +
51                         n - 2 *
52                         this->subtreeSize[v];
53                     this->dfs3(v, n);
54                 }
55             }
56         }
57     };

```

```

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 update(p, (index << 1) + 1, mid + 1, r, v);
16     st[index] =
17         max(st[index << 1], st[(index << 1) + 1]);
18 }
19 long long query(int index, int l, int r, int
20     ql, int qr) {
21     if (ql <= l && r <= qr) return st[index];
22     int mid = (l + r) >> 1;
23     long long res = -1;
24     if (ql <= mid)
25         res = max(res, query(index << 1, l, mid, ql, qr));
26     if (mid < qr) res =
27         max(res, query((index << 1) + 1, mid + 1, r, ql, qr));
28     return res;
29 }
30 int main() {
31     int n;
32     scanf("%d", &n);
33     for (int i = 1; i <= n; ++i)
34         scanf("%lld", &height[i]);
35     for (int i = 1; i <= n; ++i)
36         scanf("%lld", &B[i]);
37     long long res = B[1];
38     update(height[1], 1, 1, n, B[1]);
39     for (int i = 2; i <= n; ++i) {
40         long long temp;
41         if (height[i] - 1 >= 1)
42             temp =
43                 B[i] + query(1, 1, n, 1, height[i] - 1);
44         else
45             temp = B[i];
46         update(height[i], 1, 1, n, temp);
47         res = max(res, temp);
48     }
49     printf("%lld\n", res);
50     return 0;
51 }

```