

Contents

1	math	1
1.1	公式	1
1.2	矩陣快速冪	1
1.3	質數與因數	1
1.4	歐拉函數	2
1.5	乘法逆元、組合數	2
1.6	大步小步	2
2	字串	3
2.1	最長迴文子字串	3
2.2	KMP	3
2.3	Z Algorithm	3
3	algorithm	3
3.1	三分搜	3
3.2	差分	3
3.3	greedy	4
3.4	dinic	5
3.5	SCC Tarjan	5
3.6	SCC Kosaraju	5
3.7	ArticulationPoints Tarjan	5
3.8	最小樹狀圖	6
3.9	二分圖最大匹配	6
3.10	Astar	6
3.11	JosephusProblem	7
3.12	KM	7
3.13	LCA 倍增法	7
3.14	LCA 樹壓平 RMQ	7
3.15	LCA 樹鍊剖分	8
3.16	MCMF	9
3.17	莫隊	9
3.18	Dancing Links	9
4	DataStructure	10
4.1	BIT	10
4.2	ChthollyTree	10
4.3	線段樹 1D	10
4.4	線段樹 2D	11
4.5	權值線段樹	11
4.6	Trie	12
4.7	AC Trie	12
4.8	單調隊列	12
5	Geometry	13
5.1	Template	13
5.2	Polygon	13
5.3	Intersection	13
5.4	最小圓覆蓋	13
5.5	旋轉卡尺	14
5.6	凸包	14
5.7	半平面相交	14
6	DP	14
6.1	以價值為主的背包	14
6.2	抽屜	14
6.3	Barcode	14
6.4	Deque 最大差距	15
6.5	LCS 和 LIS	15
6.6	RangeDP	15
6.7	stringDP	15
6.8	樹 DP 有幾個 path 長度為 k	15
6.9	TreeDP reroot	15
6.10	WeightedLIS	16

1 math

1.1 公式

1. Most Divisor Number

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

2. Catlan Number

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

3. Faulhaber’s formula

$$\sum_{k=1}^n k^p = \frac{1}{p+1} \sum_{r=0}^p \binom{p+1}{r} B_r n^{p-r+1}$$

where $B_0 = 1, B_r = 1 - \sum_{i=0}^{r-1} \binom{r}{i} \frac{B_i}{r-i+1}$

也可用高斯消去法找 $deg(p+1)$ 的多項式，例：

$$\sum_{k=1}^n k^2 = a_3 n^3 + a_2 n^2 + a_1 n + a_0$$

$$\begin{bmatrix} 0^3 & 0^2 & 0^1 & 0^0 \\ 1^3 & 1^2 & 1^1 & 1^0 \\ 2^3 & 2^2 & 2^1 & 2^0 \\ 3^3 & 3^2 & 3^1 & 3^0 \end{bmatrix} \begin{bmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} 0^2 + 1^2 \\ 0^2 + 1^2 + 2^2 \\ 0^2 + 1^2 + 2^2 + 3^2 \end{bmatrix}$$
$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 8 & 4 & 2 & 1 & 5 \\ 27 & 9 & 3 & 1 & 14 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 4 & 6 & 7 & 3 \\ 0 & 0 & 6 & 11 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$
$$A = \begin{bmatrix} 1/3 \\ 1/2 \\ 1/6 \\ 0 \end{bmatrix}, \sum_{k=1}^n k^2 = \frac{1}{3} n^3 + \frac{1}{2} n^2 + \frac{1}{6} n$$

4. SG Function

$$SG(x) = mex\{SG(y) | x \rightarrow y\}$$
$$mex(S) = \min\{n | n \in \mathbb{N}, n \notin S\}$$

5. Fibonacci

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

6. Pick’s Theorem

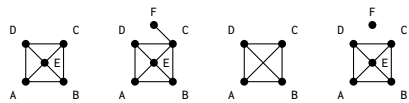
給定頂點座標均是整點（或正方形格子點）的簡單多邊形，其面積 A 和內部格點數目 i 、邊上格點數目 b 的關係為

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

7. Euler’s Formula

對於有 V 個點、 E 條邊、 F 個面（含外部）的連通平面圖

$$F + V - E = 2$$



(1)、(2)○；(3)×， \overline{AC} 與 \overline{BD} 相交；(4)×，非連通圖

1.2 矩陣快速冪

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

1.3 質數與因數

```
1 歐拉篩 O(n)
2 #define MAXN 47000 //sqrt(2^31)=46,340...
3 bool isPrime[MAXN];
4 int p[MAXN];
5 int pSize=0;
6 void getPrimes(){
7     memset(isPrime,true,sizeof(isPrime));
8     isPrime[0]=isPrime[1]=false;
9     for(int i=2;i<MAXN;i++){
10         if(isPrime[i]) p[pSize++]=i;
11         for(int j=0;j<pSize&&i*p[j]<=MAXN;j++){
12             isPrime[i*p[j]]=false;
13             if(i%p[j]==0) break;
14         }
15     }
16 }
17
18 最大公因數 O(log(min(a,b)))
19 int GCD(int a, int b){
20     if(b == 0) return a;
21     return GCD(b, a%b);
22 }
23
24 質因數分解
25 void primeFactorization(int n){
26     for(int i=0; i<p.size(); ++i) {
27         if(p[i]*p[i] > n) break;
28         if(n % p[i]) continue;
29         cout << p[i] << ' ';
30         while(n%p[i] == 0) n /= p[i];
31     }
32     if(n != 1) cout << n << ' ';
33     cout << '\n';
34 }
```

```

35 }
36 擴展歐幾里得算法 ax + by = GCD(a, b)
37 int ext_euc(int a, int b, int &x, int &y) {
38     if(b == 0){
39         x = 1, y = 0;
40         return a;
41     }
42     int d = ext_euc(b, a%b, y, x);
43     y -= a/b*x;
44     return d;
45 }
46 int main(){
47     int a, b, x, y;
48     cin >> a >> b;
49     ext_euc(a, b, x, y);
50     cout << x << ' ' << y << endl;
51     return 0;
52 }
53
54 歌德巴赫猜想
55 解：把偶數 N (6≤N≤10^6) 寫成兩個質數的和。
56 #define N 2000000
57 int ox[N], p[N], pr;
58 void PrimeTable(){
59     ox[0] = ox[1] = 1;
60     pr = 0;
61     for(int i=2;i<N;i++){
62         if(!ox[i]) p[pr++] = i;
63         for(int j=0; i*p[j]<N&&j<pr; j++)
64             ox[i*p[j]] = 1;
65     }
66 }
67
68 int main(){
69     PrimeTable();
70     int n;
71     while(cin>>n, n){
72         int x;
73         for(x=1;; x+=2)
74             if(!ox[x] && !ox[n-x]) break;
75         printf("%d = %d + %d\n", n, x, n-x);
76     }
77 }
78
79 problem :
80 給定整數 N，求N最少可以拆成多少個質數的和。
81 如果N是質數，則答案為 1。
82 如果N是偶數(N!=2)，則答案為2(強歌德巴赫猜想)。
83 如果N是奇數且N-2是質數，則答案為2(2+質數)。
84 其他狀況答案為 3 (弱歌德巴赫猜想)。
85
86 bool isPrime(int n){
87     for(int i=2;i<n;++i){
88         if(i*i>n) return true;
89         if(n%i==0) return false;
90     }
91     return true;
92 }
93
94 int main(){
95     int n;
96     cin>>n;
97     if(isPrime(n)) cout<<"1\n";
98     else if(n%2==0||isPrime(n-2)) cout<<"2\n";
99     else cout<<"3\n";
100 }

```

1.4 歐拉函數

```

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

```

```

9         }
10        if(n>1) ans=ans-ans/n;
11        return ans;
12    }

```

1.5 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;
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 }

```

1.6 大步小步

```

1 題意
2 給定 B,N,P，求出 L 滿足 B^L ≡ N(mod 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^(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=√P 能保證最多枚舉 2√P 次。
10 B^(mx+y) ≡ N(mod P)
11 B^y ≡ N(B^(-m))^x (mod P)
12 先求出 B^0, B^1, B^2, ..., B^(m-1)，
13 再枚舉 N(B^(-m)), N(B^(-m))^2, ... 查看是否有對應的
    B^y。
14 這種算法稱為大步小步演算法，

```

```

14 大步指的是枚舉 x (一次跨 m 步)，
15 小步指的是枚舉 y (一次跨 1 步)。
16 複雜度分析
17 利用 map/unorder_map 存放
18     B^0, B^1, B^2, ..., B^(m-1)，
19 枚舉 x 查詢 map/unorder_map 是否有對應的 B^y，
20 存放和查詢最多 2√P 次，時間複雜度為
21     O(√Plog√P)/O(√P)。
22
23 using LL = long long;
24 LL B, N, P;
25 LL fpow(LL a, LL b, LL c){
26     LL res=1;
27     for(;b;b>>=1){
28         if(b&1)
29             res=(res*a)%c;
30         a=(a*a)%c;
31     }
32     return res;
33 }
34 LL BSGS(LL a, LL b, LL p){
35     a%=p, b%=p;
36     if(a==0)
37         return b==0?-1:-1;
38     if(b==1)
39         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))
47             tb[tmp]=i;
48     }
49     for(LL i=0; i<sq; ++i){
50         if(tb.count(b)){
51             LL res=tb[b];
52             return i*sq+(res==sq?0:res);
53         }
54         b=(b*inv)%p;
55     }
56     return -1;
57 }
58 int main(){
59     IOS; //輸入優化
60     while(cin>>P>>B>>N){
61         LL ans=BSGS(B,N,P);
62         if(ans!=-1)
63             cout<<"no solution\n";
64         else
65             cout<<ans<<"\n";
66     }
67 }

```

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 KMP

```

1 const int maxn = 1e6 + 10;
2
3 int n, m;           // len(a), len(b)
4 int f[maxn];        // failure function
5 char a[maxn], b[maxn];
6
7 void failureFuntion() { // f[0] = 0
8     for(int i=1, j=0; i<m; ) {
9         if(b[i] == b[j]) f[i++] = ++j;
10        else if(j) j = f[j-1];
11        else f[i++] = 0;
12    }
13 }
14
15 int kmp() {
16     int i = 0, j = 0, res = 0;
17     while(i < n) {
18         if(a[i] == b[j]) i++, j++;
19         else if(j) j = f[j-1];
20         else i++;
21         if(j == m) {
22             res++; // 找到答案
23             j = 0; // non-overlapping
24         }
25     }
26     return res;
27 }
28

```

```

29 // Problem: 所有在b裡，前後綴相同的長度
30 // b = ababcbabababababab
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 }

```

2.3 Z Algorithm

```

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

```

3 algorithm

3.1 三分搜

```

1 題意
2 給定兩射線方向和速度，問兩射線最近距離。
3 題解
4 假設 F(t) 為兩射線在時間 t 的距離，F(t)
   為二次函數，
5 可用三分查找二次函數最小值。
6 struct Point{
7     double x, y, z;
8     Point() {}
9     Point(double _x,double _y,double _z):
10         x(_x),y(_y),z(_z){}
11     friend istream& operator>>(istream& is,
12         Point& p) {
13         is >> p.x >> p.y >> p.z;
14         return is;
15     }
16     Point operator+(const Point &rhs) const{
17         return Point(x+rhs.x,y+rhs.y,z+rhs.z);
18     }
19     Point operator-(const Point &rhs) const{
20         return Point(x-rhs.x,y-rhs.y,z-rhs.z);
21     }
22     Point operator*(const double &d) const{
23         return Point(x*d,y*d,z*d);
24     }
25     Point operator/(const double &d) const{
26         return Point(x/d,y/d,z/d);
27     }
28     double dist(const Point &rhs) const{
29         double res = 0;
30         res+=(x-rhs.x)*(x-rhs.x);
31         res+=(y-rhs.y)*(y-rhs.y);
32         res+=(z-rhs.z)*(z-rhs.z);
33         return res;
34     }
35 };
36 int main(){
37     IOS; //輸入優化
38     int T;
39     cin>>T;
40     for(int ti=1;ti<=T;++ti){
41         double time;
42         Point x1,y1,d1,x2,y2,d2;
43         cin>>time>>x1>>y1>>x2>>y2;
44         d1=(y1-x1)/time;
45         d2=(y2-x2)/time;
46         double L=0,R=1e8,m1,m2,f1,f2;
47         double ans = x1.dist(x2);
48         while(abs(L-R)>1e-10){
49             m1=(L+R)/2;
50             m2=(m1+R)/2;
51             f1=((d1*m1)+x1).dist((d2*m1)+x2);
52             f2=((d1*m2)+x1).dist((d2*m2)+x2);
53             ans = min(ans,min(f1,f2));
54             if(f1<f2) R=m2;
55             else L=m1;
56         }
57         cout<<"Case "<<ti<<": ";
58         cout << fixed << setprecision(4) <<
59             sqrt(ans) << '\n';
60     }
61 }

```

3.2 差分

```

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

```

```

7 在 b[l] 加上 v，b[r+1] 減去 v，
8 最後再從 0 跑到 n 使 b[i] += b[i-1]。
9 這樣一來，b[] 是一個在某區間加上v的前綴和。
10 int a[1000], b[1000];
11 // a: 前綴和數列, b: 差分數列
12 int main(){
13     int n, l, r, v;
14     cin >> n;
15     for(int i=1; i<=n; i++){
16         cin >> a[i];
17         b[i] = a[i] - a[i-1]; //建構差分數列
18     }
19     cin >> l >> r >> v;
20     b[l] += v;
21     b[r+1] -= v;
22     for(int i=1; i<=n; i++){
23         b[i] += b[i-1];
24         cout << b[i] << ' ';
25     }
26 }

```

3.3 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 最多不重疊區間
70 //problem
71 給你 n 條線段區間為 [Li, Ri]，
72 請問最多可以選擇幾條不重疊的線段(頭尾可相連)？
73 //solution
74 依照右界由小到大排序，
75 每次取到一個不重疊的線段，答案 +1。
76 //code
77 struct Line{
78     int L, R;
79     bool operator<(const Line &rhs) const{
80         return R<rhs.R;
81     }
82 };
83 int main(){
84     int t;
85     cin >> t;
86     Line a[30];
87     while(t--){
88         int n=0;
89         while(cin >> a[n].L >> a[n].R, a[n].L || a[n].R){
90             ++n;
91         }
92         sort(a, a+n);
93         int ans=1, R=a[0].R;
94         for(int i=1; i<n; ++i){
95             if(a[i].L>=R){
96                 ++ans;
97                 R=a[i].R;
98             }
99         }
100         cout << ans << '\n';
101     }
102 }
103 最小化最大延遲問題
104 //problem
105 給定 N 項工作，每項工作的需要處理時長為 Ti，
106 期限是 Di，第 i 項工作延遲的時間為
    Li=max(0, Fi-Di)，
107 原本 Fi 為第 i 項工作的完成時間，
108 求一種工作排序使 maxLi 最小。
109 //solution
110 按照到期時間從早到晚處理。
111 //code
112 struct Work{
113     int t, d;
114     bool operator<(const Work &rhs) const{
115         return d<rhs.d;
116     }
117 };
118 int main(){
119     int n;
120     Work a[10000];
121     cin >> n;
122     for(int i=0; i<n; ++i)
123         cin >> a[i].t >> a[i].d;
124     sort(a, a+n);
125     int maxL=0, sumT=0;
126     for(int i=0; i<n; ++i){
127         sumT+=a[i].t;
128         maxL=max(maxL, sumT-a[i].d);
129     }
130     cout << maxL << '\n';

```

```

131 }
132 最少延遲數量問題
133 //problem
134 給定 N 個工作，每個工作的需要處理時長為 Ti，
135 期限是 Di，求一種工作排序使得逾期工作數量最小。
136 //solution
137 期限越早到期的工作越先做。
138 將工作依照到期時間從早到晚排序，
139 依序放入工作列表中，如果發現有工作預期，
140 就從目前選擇的工作中，移除耗時最長的工作。
141 上述方法為 Moore-Hodgson's Algorithm。
142 //problem
143 給定烏龜的重量和可承受重量，問最多可以疊幾隻烏龜？
144 //solution
145 和最少延遲數量問題是相同的問題，只要將題敘做轉換。
146 工作處理時長 → 烏龜重量
147 工作期限 → 烏龜可承受重量
148 多少工作不延期 → 可以疊幾隻烏龜
149 //code
150 struct Work{
151     int t, d;
152     bool operator<(const Work &rhs) const{
153         return d<rhs.d;
154     }
155 };
156 int main(){
157     int n=0;
158     Work a[10000];
159     priority_queue<int> pq;
160     while(cin >> a[n].t >> a[n].d)
161         ++n;
162     sort(a, a+n);
163     int sumT=0, ans=n;
164     for(int i=0; i<n; ++i){
165         pq.push(a[i].t);
166         sumT+=a[i].t;
167         if(a[i].d<sumT){
168             int x=pq.top();
169             pq.pop();
170             sumT-=x;
171             --ans;
172         }
173     }
174     cout << ans << '\n';
175 }
176 任務調度問題
177 //problem
178 給定 N 項工作，每項工作的需要處理時長為 Ti，
179 期限是 Di，如果第 i 項工作延遲需要受到 pi
    單位懲罰，
180 請問最少會受到多少單位懲罰。
181 //solution
182 依照懲罰由大到小排序，
183 每項工作依序嘗試可不可以放在
    Di-Ti+1, Di-Ti, ..., 1, 0，
184 如果有空閒就放進去，否則延後執行。
185 //problem
186 給定 N 項工作，每項工作的需要處理時長為 Ti，
187 期限是 Di，如果第 i 項工作在期限內完成會獲得 ai
    單位獎勵，
188 請問最多會獲得多少單位獎勵。
189 //solution
190 和上題相似，這題變成依照獎勵由大到小排序。
191 //code
192 struct Work{
193     int d, p;
194     bool operator<(const Work &rhs) const{
195         return p>rhs.p;
196     }
197 };
198 int main(){
199     int n;
200     Work a[100005];
201     bitset<100005> ok;
202     while(cin >> n){

```

```

206 ok.reset();
207 for(int i=0;i<n;++i)
208     cin>>a[i].d>>a[i].p;
209 sort(a,a+n);
210 int ans=0;
211 for(int i=0;i<n;++i){
212     int j=a[i].d;
213     while(j--){
214         if(!ok[j]){
215             ans+=a[i].p;
216             ok[j]=true;
217             break;
218         }
219     }
220     cout<<ans<<"\n";
221 }
222 }

```

3.4 dinic

```

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

```

```

54 int dinic() { // O((V^2)E)
55     int result = 0;
56     while(bfs()) {
57         memset(dfs_idx, 0, sizeof(dfs_idx));
58         result += dfs(S, inf);
59     }
60     return result;
61 }

```

3.5 SCC Tarjan

```

1 //單純考SCC，每個SCC中找成本最小的蓋，如果有多個一樣小
2 //的要數出來，因為題目要方法數
3 //注意以下程式有縮點，但沒存起來，
4 //存法就是開一個array -> ID[u] = SCCID
5 #define maxn 100005
6 #define MOD 1000000007
7 long long cost[maxn];
8 vector<vector<int>> G;
9 int SCC = 0;
10 stack<int> sk;
11 int dfn[maxn];
12 int low[maxn];
13 bool inStack[maxn];
14 int dfsTime = 1;
15 long long totalCost = 0;
16 long long ways = 1;
17 void dfs(int u) {
18     dfn[u] = low[u] = dfsTime;
19     ++dfsTime;
20     sk.push(u);
21     inStack[u] = true;
22     for (int v: G[u]) {
23         if (dfn[v] == 0) {
24             dfs(v);
25             low[u] = min(low[u], low[v]);
26         }
27         else if (inStack[v]) {
28             //屬於同個SCC且是我的back edge
29             low[u] = min(low[u], dfn[v]);
30         }
31     }
32     //如果是SCC
33     if (dfn[u] == low[u]) {
34         long long minCost = 0x3f3f3f3f;
35         int currWays = 0;
36         ++SCC;
37         while (1) {
38             int v = sk.top();
39             inStack[v] = 0;
40             sk.pop();
41             if (minCost > cost[v]) {
42                 minCost = cost[v];
43                 currWays = 1;
44             }
45             else if (minCost == cost[v]) {
46                 ++currWays;
47             }
48             if (v == u)
49                 break;
50         }
51         totalCost += minCost;
52         ways = (ways * currWays) % MOD;
53     }
54 }
55 int main() {
56     int n;
57     scanf("%d", &n);
58     for (int i = 1; i <= n; ++i)
59         scanf("%lld", &cost[i]);
60     G.assign(n + 5, vector<int>());
61     int m;
62     scanf("%d", &m);
63     int u, v;
64     for (int i = 0; i < m; ++i) {
65         scanf("%d %d", &u, &v);
66         G[u].emplace_back(v);

```

```

67     }
68     for (int i = 1; i <= n; ++i) {
69         if (dfn[i] == 0)
70             dfs(i);
71     }
72     printf("%lld %lld\n", totalCost, ways %
73         MOD);
74     return 0;
75 }

```

3.6 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 void dfs2(int u) {
11     group[u] = sccCnt;
12     for (int v : g2[u])
13         if (!group[v]) dfs2(v);
14 }
15 void kosaraju() {
16     sccCnt = 0;
17     for (int i = 1; i <= n; ++i)
18         if (!vis[i]) dfs1(i);
19     for (int i = n; i >= 1; --i)
20         if (!group[s[i]]) {
21             ++sccCnt;
22             dfs2(s[i]);
23         }
24 }
25 }
26 }

```

3.7 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] >=
21                 dfn[u])
22                 isCut = true;
23         }
24         else if (v != parent)
25             low[u] = min(low[u], dfn[v]);
26     }
27     //If u is root of DFS
28     //tree->有兩個以上的children
29     if (parent == -1 && child >= 2)
30         isCut = true;
31     if (isCut) ++res;
32 }
33 int main() {

```



```

32 char input[105];
33 char* token;
34 while (scanf("%d", &N) != EOF && N) {
35     G.assign(105, vector<int>());
36     memset(visited, false,
37           sizeof(visited));
38     memset(low, 0, sizeof(low));
39     memset(dfn, 0, sizeof(visited));
40     timer = 0;
41     res = 0;
42     getchar(); // for \n
43     while (fgets(input, 105, stdin)) {
44         if (input[0] == '0')
45             break;
46         int size = strlen(input);
47         input[size - 1] = '\0';
48         --size;
49         token = strtok(input, " ");
50         int u = atoi(token);
51         int v;
52         while (token = strtok(NULL, " ")) {
53             v = atoi(token);
54             G[u].emplace_back(v);
55             G[v].emplace_back(u);
56         }
57         tarjan(1, -1);
58         printf("%d\n", res);
59     }
60     return 0;
61 }

```

3.8 最小樹狀圖

```

1 const int maxn = 60 + 10;
2 const int inf = 0x3f3f3f3f;
3 struct Edge {
4     int s, t, cap, cost;
5 }; // cap 為頻寬 (optional)
6 int n, m, c;
7 int inEdge[maxn], idx[maxn], pre[maxn],
8   vis[maxn];
9 // 對於每個點，選擇對它入度最小的那條邊
10 // 找環，如果沒有則 return;
11 // 進行縮環並更新其他點到環的距離。
12 int dirMST(vector<Edge> edges, int low) {
13     int result = 0, root = 0, N = n;
14     while(true) {
15         memset(inEdge, 0x3f, sizeof(inEdge));
16         // 找所有點的 in edge 放進 inEdge
17         // optional: low 為最小 cap 限制
18         for(const Edge& e : edges) {
19             if(e.cap < low) continue;
20             if(e.s!=e.t &&
21                e.cost<inEdge[e.t]) {
22                 inEdge[e.t] = e.cost;
23                 pre[e.t] = e.s;
24             }
25         }
26         for(int i=0; i<N; i++) {
27             if(i!=root && inEdge[i]==inf)
28                 return -1; //除了root 還有點沒有 in
29                 edge
30         }
31         int seq = inEdge[root] = 0;
32         memset(idx, -1, sizeof(idx));
33         memset(vis, -1, sizeof(vis));
34         // 找所有的 cycle，一起編號為 seq
35         for(int i=0; i<N; i++) {
36             result += inEdge[i];
37             int cur = i;
38             while(vis[cur]!=i &&
39                   idx[cur]==-1) {
40                 if(cur == root) break;
41                 vis[cur] = i;
42                 cur = pre[cur];
43             }
44         }
45     }
46 }

```

```

39 }
40 if(cur!=root && idx[cur]==-1) {
41     for(int j=pre[cur]; j!=cur;
42         j=pre[j])
43         idx[j] = seq;
44     idx[cur] = seq++;
45 }
46 if(seq == 0) return result; // 沒有
47 cycle
48 for(int i=0; i<N; i++)
49     // 沒有被縮點的點
50     if(idx[i] == -1) idx[i] = seq++;
51 // 縮點並重新編號
52 for(Edge& e : edges) {
53     if(idx[e.s] != idx[e.t])
54         e.cost -= inEdge[e.t];
55     e.s = idx[e.s];
56     e.t = idx[e.t];
57 }
58 N = seq;
59 root = idx[root];
60 }

```

3.9 二分圖最大匹配

```

1 /* 核心：最大點獨立集 = |V| -
2    /最大匹配數 /，用匈牙利演算法找出最大匹配數 */
3 vector<Student> boys;
4 vector<Student> girls;
5 vector<vector<int>> G;
6 bool used[505];
7 int p[505];
8 bool match(int i) {
9     for (int j: G[i]) {
10         if (!used[j]) {
11             used[j] = true;
12             if (p[j] == -1 || match(p[j])) {
13                 p[j] = i;
14                 return true;
15             }
16         }
17     }
18     return false;
19 }
20 void maxMatch(int n) {
21     memset(p, -1, sizeof(p));
22     int res = 0;
23     for (int i = 0; i < boys.size(); ++i) {
24         memset(used, false, sizeof(used));
25         if (match(i))
26             ++res;
27     }
28     cout << n - res << '\n';
29 }

```

3.10 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 {
9     int u, v, w;
10 };
11 struct Item_pqH {
12     int u, w;
13     bool operator <(const Item_pqH& other)
14         const {
15         return this->w > other.w;
16     }
17 }

```

```

15 }
16 };
17 struct Item_astar {
18     int u, g, f;
19     bool operator <(const Item_astar& other)
20         const {
21         return this->f > other.f;
22     }
23 };
24 vector<vector<Edge>> G;
25 //反向圖，用於建h(u)
26 vector<vector<Edge>> invertG;
27 int h[maxn];
28 bool visited[maxn];
29 int cnt[maxn];
30 //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)
31 void dijkstra(int s, int t) {
32     memset(visited, 0, sizeof(visited));
33     priority_queue<Item_pqH> pq;
34     pq.push({s, 0});
35     h[s] = 0;
36     while (!pq.empty()) {
37         Item_pqH curr = pq.top();
38         pq.pop();
39         visited[curr.u] = true;
40         for (Edge& edge: invertG[curr.u]) {
41             if (!visited[edge.v]) {
42                 if (h[edge.v] > h[curr.u] +
43                     edge.w) {
44                     h[edge.v] = h[curr.u] +
45                         edge.w;
46                     pq.push({edge.v,
47                             h[edge.v]});
48                 }
49             }
50         }
51     }
52 }
53 int Astar(int s, int t, int k) {
54     memset(cnt, 0, sizeof(cnt));
55     priority_queue<Item_astar> pq;
56     pq.push({s, 0, h[s]});
57     while (!pq.empty()) {
58         Item_astar curr = pq.top();
59         pq.pop();
60         ++cnt[curr.u];
61         //終點出現k次，此時即可得k短路
62         if (cnt[t] == k)
63             return curr.g;
64         for (Edge& edge: G[curr.u]) {
65             if (cnt[edge.v] < k) {
66                 pq.push({edge.v, curr.g +
67                         edge.w, curr.g + edge.w
68                         + h[edge.v]});
69             }
70         }
71     }
72     return -1;
73 }
74 int main() {
75     int n, m;
76     while (scanf("%d %d", &n, &m) && (n != 0
77         && m != 0)) {
78         G.assign(n + 5, vector<Edge>());
79         invertG.assign(n + 5, vector<Edge>());
80         int s, t, k;
81         scanf("%d %d %d", &s, &t, &k);
82         int u, v, w;
83         for (int i = 0; i < m; ++i) {
84             scanf("%d %d %d", &u, &v, &w);
85             G[u].emplace_back(Edge{u, v, w});
86             invertG[v].emplace_back(Edge{v,
87                                     u, w});
88         }
89         memset(h, 0x3f, sizeof(h));
90         dijkstra(t, s);
91         printf("%d\n", Astar(s, t, k));
92     }
93 }

```

```

85     return 0;
86 }

```

3.11 JosephusProblem

```

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

```

3.12 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] &&
16            !T[j]) {
17            T[j] = true;
18            if ((L[j] == -1) || match(L[j])) {
19                L[j] = i;
20                return true;
21            }
22        }
23    }
24    return false;
25 }
26 //修改二分圖上的交錯路徑上點的權重
27 //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調調看, 才能保證最大權重匹配
30 void update()
31 {
32     int diff = 0x3f3f3f3f;
33     for (int i = 0; i < n; ++i) {
34         if (S[i]) {
35             for (int j = 0; j < n; ++j) {
36                 if (!T[j])
37                     diff = min(diff, Lx[i] +
38                                 Ly[j] - W[i][j]);
39             }
40         }
41         for (int i = 0; i < n; ++i) {
42             if (S[i]) Lx[i] -= diff;
43             if (T[i]) Ly[i] += diff;
44         }
45     }
46 void KM()
47 {
48     for (int i = 0; i < n; ++i) {
49         L[i] = -1;
50         Lx[i] = Ly[i] = 0;
51         for (int j = 0; j < n; ++j)
52             Lx[i] = max(Lx[i], W[i][j]);
53     }
54     for (int i = 0; i < n; ++i) {
55         while(1) {
56             memset(S, false, sizeof(S));
57             memset(T, false, sizeof(T));
58             if (match(i))
59                 break;
60             else
61                 update(); //去調整vertex
62                             //labeling以增加增廣路徑
63         }
64     }
65 int main() {
66     while (scanf("%d", &n) != EOF) {
67         for (int i = 0; i < n; ++i)
68             for (int j = 0; j < n; ++j)
69                 scanf("%d", &W[i][j]);
70         KM();
71         int res = 0;
72         for (int i = 0; i < n; ++i) {
73             if (i != 0)
74                 printf(" %d", Lx[i]);
75             else
76                 printf("%d", Lx[i]);
77             res += Lx[i];
78         }
79         puts("");
80         for (int i = 0; i < n; ++i) {
81             if (i != 0)
82                 printf(" %d", Ly[i]);
83             else
84                 printf("%d", Ly[i]);
85             res += Ly[i];
86         }
87         puts("");
88         printf("%d\n", res);
89     }
90     return 0;
91 }

```

3.13 LCA 倍增法

```

1 //倍增法預處理O(nlogn), 查詢O(logn),
2 //利用Lca找樹上任兩點距離
3 #define maxn 100005
4 struct Edge {
5     int u, v, w;
6 };
7 vector<vector<Edge>> G; // tree

```

```

8 int fa[maxn][31]; //fa[u][i] -> u的第2^i個祖先
9 long long dis[maxn][31];
10 int dep[maxn]; //深度
11 void dfs(int u, int p) { //預處理fa
12     fa[u][0] = p; //因為u的第2^0 = 1的祖先就是p
13     dep[u] = dep[p] + 1;
14     //第2^i的祖先是(第2^(i-1)個祖先)的
15     //第2^(i-1)的祖先
16     //ex: 第8個祖先是 (第4個祖先)的第4個祖先
17     for (int i = 1; i < 31; ++i) {
18         fa[u][i] = fa[fa[u][i-1]][i-1];
19         dis[u][i] = dis[fa[u][i-1]][i-1]
20             + dis[u][i-1];
21     }
22     //遍歷子節點
23     for (Edge& edge: G[u]) {
24         if (edge.v == p)
25             continue;
26         dis[edge.v][0] = edge.w;
27         dfs(edge.v, u);
28     }
29 long long lca(int x, int y) {
30     //此函數是找lca同時計算x、y的距離 -> dis(x,
31     //lca) + dis(lca, y)
32     //讓y比x深
33     if (dep[x] > dep[y])
34         swap(x, y);
35     int deltaDep = dep[y] - dep[x];
36     long long res = 0;
37     //讓y與x在同一個深度
38     for (int i = 0; deltaDep != 0; ++i,
39         deltaDep >>= 1)
40         if (deltaDep & 1)
41             res += dis[y][i], y = fa[y][i];
42     if (y == x) //x = y -> x、y彼此是彼此的祖先
43         return res;
44     //往上找, 一起跳, 但x、y不能重疊
45     for (int i = 30; i >= 0 && y != x; --i) {
46         if (fa[x][i] != fa[y][i]) {
47             res += dis[x][i] + dis[y][i];
48             x = fa[x][i];
49             y = fa[y][i];
50         }
51     }
52     //最後發現不能跳了, 此時x的第2^0 =
53     //1個祖先(或說y的第2^0 =
54     //1的祖先)即為x、y的lca
55     res += dis[x][0] + dis[y][0];
56     return res;
57 }
58 int main() {
59     int n, q;
60     while (~scanf("%d", &n) && n) {
61         int v, w;
62         G.assign(n + 5, vector<Edge>());
63         for (int i = 1; i <= n - 1; ++i) {
64             scanf("%d %d", &v, &w);
65             G[i + 1].push_back({i + 1, v + 1, w});
66             G[v + 1].push_back({v + 1, i + 1, w});
67         }
68         dfs(1, 0);
69         scanf("%d", &q);
70         int u;
71         while (q--) {
72             scanf("%d %d", &u, &v);
73             printf("%lld%c", lca(u + 1, v +
74                 1), (q ? ' ' : '\n'));
75         }
76     }
77     return 0;
78 }

```

3.14 LCA 樹壓平 RMQ

```

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,

```

```

2 //如果用笛卡兒樹可以壓到 $O(n)$ 建立,  $O(1)$ 查詢
3 //理論上可以過, 但遇到直鏈的case dfs深度會stack
  overflow
4 #define maxn 100005
5 struct Edge {
6     int u, v, w;
7 };
8 int dep[maxn], pos[maxn];
9 long long dis[maxn];
10 int st[maxn * 2][32]; //sparse table
11 int realLCA[maxn * 2][32];
    //最小深度對應的節點, 及真正的LCA
12 int Log[maxn]; //取代std::log2
13 int tp; // timestamp
14 vector<vector<Edge>> G; // tree
15 void calLog() {
16     Log[1] = 0;
17     Log[2] = 1;
18     for (int i = 3; i < maxn; ++i)
19         Log[i] = Log[i / 2] + 1;
20 }
21 void buildST() {
22     for (int j = 0; Log[tp]; ++j) {
23         for (int i = 0; i + (1 << j) - 1 < tp;
24             ++i) {
25             if (st[i - 1][j] < st[i - 1][j + (1 <<
26                 i - 1)]) {
27                 st[i][j] = st[i - 1][j];
28                 realLCA[i][j] = realLCA[i - 1][j];
29             }
30             else {
31                 st[i][j] = st[i - 1][j + (1 << i -
32                     1)];
33                 realLCA[i][j] = realLCA[i - 1][j + (1
34                     << i - 1)];
35             }
36         }
37     } //  $O(n \log n)$ 
38 }
39 int query(int l, int r) { // [l, r] min
    depth即為lca的深度
40     int k = Log[r - l + 1];
41     if (st[l][k] < st[r - (1 << k) + 1][k])
42         return realLCA[l][k];
43     else
44         return realLCA[r - (1 << k) + 1][k];
45 }
46 void dfs(int u, int p) { //euler tour
47     pos[u] = tp;
48     st[tp][0] = dep[u];
49     realLCA[tp][0] = dep[u];
50     ++tp;
51     for (int i = 0; i < G[u].size(); ++i) {
52         Edge& edge = G[u][i];
53         if (edge.v == p) continue;
54         dep[edge.v] = dep[u] + 1;
55         dis[edge.v] = dis[edge.u] + edge.w;
56         dfs(edge.v, u);
57         st[tp++][0] = dep[u];
58     }
59 }
60 long long getDis(int u, int v) {
61     if (pos[u] > pos[v])
62         swap(u, v);
63     int lca = query(pos[u], pos[v]);
64     return dis[u] + dis[v] - 2 *
        dis[query(pos[u], pos[v])];
65 }
66 int main() {
67     int n, q;
68     calLog();
69     while (~scanf("%d", &n) && n) {
70         int v, w;
71         G.assign(n + 5, vector<Edge>());
72         tp = 0;
73         for (int i = 1; i <= n - 1; ++i) {
74             scanf("%d %d", &v, &w);
75             G[i].push_back({i, v, w});
76         }
77     }
78 }

```

```

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("%lldc", getDis(u, v),
81             (q) ? ' ' : '\n');
82     }
83     return 0;
84 }

```

3.15 LCA 樹鍊剖分

```

1 #define maxn 5005
2 //LCA, 用來練習樹鍊剖分
3 //題意:
    給定樹, 找任兩點的中點, 若中點不存在(路徑為even)
4 int dfn[maxn];
5 int parent[maxn];
6 int depth[maxn];
7 int subtreeSize[maxn];
8 //樹鍊的頂點
9 int top[maxn];
10 //將dfn轉成node編碼
11 int dfnToNode[maxn];
12 //重兒子
13 int hson[maxn];
14 int dfsTime = 1;
15 //tree
16 vector<vector<int>> G;
17 //處理parent、depth、subtreeSize、dfnToNode
18 void dfs1(int u, int p) {
19     parent[u] = p;
20     hson[u] = -1;
21     subtreeSize[u] = 1;
22     for (int v: G[u]) {
23         if (v != p) {
24             depth[v] = depth[u] + 1;
25             dfs1(v, u);
26             subtreeSize[u] += subtreeSize[v];
27             if (hson[u] == -1 ||
                subtreeSize[hson[u]] <
                subtreeSize[v]) {
28                 hson[u] = v;
29             }
30         }
31     }
32 }
33 //實際剖分 <- 參數t是top的意思
34 //t初始應為root本身
35 void dfs2(int u, int t) {
36     top[u] = t;
37     dfn[u] = dfsTime;
38     dfnToNode[dfsTime] = u;
39     ++dfsTime;
40     //葉子點 -> 沒有重兒子
41     if (hson[u] == -1)
42         return;
43     //優先對重兒子dfs, 才能保證同一重鍊dfn連續
44     dfs2(hson[u], t);
45     for (int v: G[u]) {
46         if (v != parent[u] && v != hson[u])
47             dfs2(v, v);
48     }
49 }
50 //不斷跳鍊, 當跳到同一條鍊時, 深度小的即為LCA
51 //跳鍊時優先鍊頂深度大的跳
52 int LCA(int u, int v) {
53     while (top[u] != top[v]) {
54         if (depth[top[u]] > depth[top[v]])
55             u = parent[top[u]];
56         else
57             v = parent[top[v]];
58     }
59 }

```

```

58 }
59     return (depth[u] > depth[v]) ? v : u;
60 }
61 int getK_parent(int u, int k) {
62     while (k-- && (u != -1))
63         u = parent[u];
64     return u;
65 }
66 int main() {
67     int n;
68     while (scanf("%d", &n) && n) {
69         dfsTime = 1;
70         G.assign(n + 5, vector<int>());
71         int u, v;
72         for (int i = 1; i < n; ++i) {
73             scanf("%d %d", &u, &v);
74             G[u].emplace_back(v);
75             G[v].emplace_back(u);
76         }
77         dfs1(1, -1);
78         dfs2(1, 1);
79         int q;
80         scanf("%d", &q);
81         for (int i = 0; i < q; ++i) {
82             scanf("%d %d", &u, &v);
83             //先得到LCA
84             int lca = LCA(u, v);
85             //計算路徑長(經過的邊)
86             int dis = depth[u] + depth[v] - 2
                * depth[lca];
87             //讓v比u深或等於
88             if (depth[u] > depth[v])
89                 swap(u, v);
90             if (u == v) {
91                 printf("The fleas meet at
                    %d.\n", u);
92             }
93             else if (dis % 2 == 0) {
94                 //路徑長是even -> 有中點
95                 printf("The fleas meet at
                    %d.\n", getK_parent(v,
                        dis / 2));
96             }
97             else {
98                 //路徑長是odd -> 沒有中點
99                 if (depth[u] == depth[v]) {
100                     int x = getK_parent(u, dis
                        / 2);
101                     int y = getK_parent(v, dis
                        / 2);
102                     if (x > y)
103                         swap(x, y);
104                     printf("The fleas jump
                        forever between %d
                        and %d.\n", x, y);
105                 }
106                 else {
107                     //技巧: 讓深的點v往上dis /
                        2步 = y,
108                     //這個點的parent設為x
109                     //此時的x、y就是答案要的中點兩點
110                     //主要是往下不好找, 所以改用深的點u
111                     int y = getK_parent(v, dis
                        / 2);
112                     int x = getK_parent(y, 1);
113                     if (x > y)
114                         swap(x, y);
115                     printf("The fleas jump
                        forever between %d
                        and %d.\n", x, y);
116                 }
117             }
118         }
119     }
120     return 0;
121 }

```


3.16 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     //如果dis[t] > 0代表根本不賺還倒賠
49     if (dis[t] > 0)
50         return false;
51     maxFlow += outFlow[t];
52     minCost += dis[t] * outFlow[t];
53     //一路更新回去這次最短路流完後要維護的
54     //MaxFlow演算法相關(如反向邊等)
55     int curr = t;
56     while (curr != s) {
57         edges[parent[curr]].flow += outFlow[t];
58         edges[parent[curr] ^ 1].flow -= outFlow[t];
59         curr = edges[parent[curr]].u;
60     }
61     return true;
62 }

```

```

63 long long MCMF() {
64     long long maxFlow = 0;
65     long long minCost = 0;
66     while (SPFA(maxFlow, minCost))
67         ;
68     return minCost;
69 }
70 int main() {
71     int T;
72     scanf("%d", &T);
73     for (int Case = 1; Case <= T; ++Case) {
74         //總共幾個月，囤貨成本
75         int M, I;
76         scanf("%d %d", &M, &I);
77         //node size
78         n = M + M + 2;
79         G.assign(n + 5, vector<int>());
80         edges.clear();
81         s = 0;
82         t = M + M + 1;
83         for (int i = 1; i <= M; ++i) {
84             int produceCost, produceMax, sellPrice, sellMax, inventoryMonth;
85             scanf("%d %d %d %d %d", &produceCost, &produceMax, &sellPrice, &sellMax, &inventoryMonth);
86             addEdge(s, i, produceMax, produceCost);
87             addEdge(M + i, t, sellMax, -sellPrice);
88             for (int j = 0; j <= inventoryMonth; ++j) {
89                 if (i + j <= M)
90                     addEdge(i, M + i + j, INF, I * j);
91             }
92         }
93         printf("Case %d: %lld\n", Case, -MCMF());
94     }
95     return 0;
96 }

```

3.17 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     while (r < queries[i].r)
65         ++r;
66     add(r);
67     while (r > queries[i].r)
68         --r;
69     ans[queries[i].id] = res;
70 }
71 }
72 }
73 }
74 }
75 }
76 }
77 }
78 }
79 }

```

3.18 Dancing Links

```

1 struct DLX {
2     int seq, resSize;
3     int col[maxn], row[maxn];
4     int U[maxn], D[maxn], R[maxn], L[maxn];
5     int rowHead[maxn], colSize[maxn];
6     int result[maxn];
7     DLX(int r, int c) {
8         for (int i = 0; i <= c; ++i) {
9             L[i] = i - 1, R[i] = i + 1;
10            U[i] = D[i] = i;
11        }
12        L[R[seq = c] = 0] = c;
13        resSize = -1;
14        memset(rowHead, 0, sizeof(rowHead));
15        memset(colSize, 0, sizeof(colSize));

```

4 DataStructure

4.1 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): size(sz), bit(sz+1),
9         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     /** @return The sum of all values in [0,
24         idx]. */
25     T pre_sum(int idx) {
26         T total = 0;
27         for (++idx; idx>0; idx--=(idx & -idx))
28             total += bit[idx];
29         return total;
30     }
31 };

```

4.2 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)
9         const{
10         return this->l < other.l;
11     }
12 };
13 set<Node> chthollyTree;
14 //將[l, r] 拆成 [l, pos - 1], [pos, r]
15 set<Node>::iterator split(long long pos) {
16     //找第一個左端點大於等於pos的區間
17     set<Node>::iterator it =
18         chthollyTree.lower_bound(Node(pos,
19             0, 0));
20     //運氣很好直接找到左端點是pos的區間
21     if (it != chthollyTree.end() && it->l ==
22         pos)
23         return it;
24     //到這邊代表找到的是第一個左端點大於pos的區間
25     //it - 1即可找到左端點等於pos的區間
26     // (不會是別的, 因為沒有重疊的區間)
27     --it;
28     long long l = it->l, r = it->r;
29     long long val = it->val;
30     chthollyTree.erase(it);
31     chthollyTree.insert(Node(l, pos - 1,
32         val));
33     //回傳左端點是pos的區間iterator
34     return chthollyTree.insert(Node(pos, r,
35         val)).first;
36 }
37 //區間賦值

```

```

31 void assign(long long l, long long r, long
32     long val) {
33     //<注意>
34     //end與begin的順序不能調換, 因為end的split可能會改變
35     //因為end可以在原本begin的區間中
36     set<Node>::iterator end = split(r + 1),
37         begin = split(l);
38     //begin到end全部刪掉
39     chthollyTree.erase(begin, end);
40     //填回去[l, r]的區間
41     chthollyTree.insert(Node(l, r, val));
42 }
43 //區間加值(直接一個個區間去加)
44 void add(long long l, long long r, long long
45     val) {
46     set<Node>::iterator end = split(r + 1);
47     set<Node>::iterator begin = split(l);
48     for (set<Node>::iterator it = begin; it
49         != end; ++it)
50         it->val += val;
51 }
52 //查詢區間第k小 -> 直接把每個區間丟去vector排序
53 long long getKthSmallest(long long l, long
54     long r, long long k) {
55     set<Node>::iterator end = split(r + 1);
56     set<Node>::iterator begin = split(l);
57     //pair -> first: val, second: 區間長度
58     vector<pair<long long, long long>> vec;
59     for (set<Node>::iterator it = begin; it
60         != end; ++it) {
61         vec.push_back({it->val, it->r - it->l
62             + 1});
63     }
64     sort(vec.begin(), vec.end());
65     for (const pair<long long, long long>&
66         p: vec) {
67         k -= p.second;
68         if (k <= 0)
69             return p.first;
70     }
71 }
72 //不應該跑到這
73 return -1;
74 }
75 //快速幂
76 long long qpow(long long x, long long n,
77     long long mod) {
78     long long res = 1;
79     x %= mod;
80     while (n)
81     {
82         if (n & 1)
83             res = res * x % mod;
84         n >>= 1;
85         x = x * x % mod;
86     }
87     return res;
88 }
89 //區間n次方和
90 long long sumOfPow(long long l, long long r,
91     long long mod) {
92     long long total = 0;
93     set<Node>::iterator end = split(r + 1);
94     set<Node>::iterator begin = split(l);
95     for (set<Node>::iterator it = begin; it
96         != end; ++it)
97     {
98         total = (total + qpow(it->val, n,
99             mod) * (it->r - it->l + 1)) %
100             mod;
101     }
102     return total;
103 }
104 #define MAXN 1000
105 int data[MAXN]; //原數據

```

4.3 線段樹 1D

```

3 int st[4 * MAXN]; //線段樹
4 int tag[4 * MAXN]; //懶標
5 inline int pull(int l, int r) {
6 // 隨題目改變sum、max、min
7 // l、r是左右樹的index
8 return st[l] + st[r];
9 }
10 void build(int l, int r, int i) {
11 // 在[l, r]區間建樹，目前根的index為i
12 if (l == r) {
13 st[i] = data[l];
14 return;
15 }
16 int mid = l + ((r - l) >> 1);
17 build(l, mid, i * 2);
18 build(mid + 1, r, i * 2 + 1);
19 st[i] = pull(i * 2, i * 2 + 1);
20 }
21 int qry(int ql, int qr, int l, int r, int i) {
22 // [ql,qr]是查詢區間，[l,r]是當前節點包含的區間
23 if (ql <= l && r <= qr)
24 return st[i];
25 int mid = l + ((r - l) >> 1);
26 if (tag[i]) {
27 //如果當前懶標有值則更新左右節點
28 st[i * 2] += tag[i] * (mid - l + 1);
29 st[i * 2 + 1] += tag[i] * (r - mid);
30 tag[i * 2] += tag[i];
31 tag[i * 2 + 1] += tag[i];
32 tag[i] = 0;
33 }
34 int sum = 0;
35 if (ql <= mid)
36 sum += query(ql, qr, l, mid, i * 2);
37 if (qr > mid)
38 sum += query(ql, qr, mid + 1, r, i * 2 + 1);
39 return sum;
40 }
41 void update(
42 int ql, int qr, int l, int r, int i, int c) {
43 // [ql,qr]是查詢區間，[l,r]是當前節點包含的區間
44 // c是變化量
45 if (ql <= l && r <= qr) {
46 st[i] += (r - l + 1) * c;
47 //求和，此需乘上區間長度
48 tag[i] += c;
49 return;
50 }
51 int mid = l + ((r - l) >> 1);
52 if (tag[i] && l != r) {
53 //如果當前懶標有值則更新左右節點
54 st[i * 2] += tag[i] * (mid - l + 1);
55 st[i * 2 + 1] += tag[i] * (r - mid);
56 tag[i * 2] += tag[i]; //下傳懶標至左節點
57 tag[i * 2 + 1] += tag[i]; //下傳懶標至右節點
58 tag[i] = 0;
59 }
60 if (ql <= mid) update(ql, qr, l, mid, i * 2, c);
61 if (qr > mid) update(ql, qr, mid + 1, r, i * 2 + 1, c);
62 st[i] = pull(i * 2, i * 2 + 1);
63 //如果是直接改值而不是加值，query與update中的tag與st
64 //改值從+=改成=

```

4.4 線段樹 2D

```

1 //純2D segment tree 區間查詢單點修改最大最小值
2 #define maxn 2005 //500 * 4 + 5
3 int maxST[maxn][maxn], minST[maxn][maxn];
4 int N;
5 void modifyY(int index, int l, int r, int
6 val, int yPos, int xIndex, bool
7 xIsLeaf) {
8 if (l == r) {
9 if (xIsLeaf) {
10 maxST[xIndex][index] =
11 minST[xIndex][index] = val;
12 return;
13 }
14 maxST[xIndex][index] =
15 max(maxST[xIndex * 2][index],
16 maxST[xIndex * 2 + 1][index]);
17 minST[xIndex][index] =
18 min(minST[xIndex * 2][index],
19 minST[xIndex * 2 + 1][index]);
20 }
21 else {
22 int mid = (l + r) / 2;
23 if (yPos <= mid)
24 modifyY(index * 2, l, mid, val,
25 yPos, xIndex, xIsLeaf);
26 else
27 modifyY(index * 2 + 1, mid + 1,
28 r, val, yPos, xIndex,
29 xIsLeaf);
30 }
31 }
32 void modifyX(int index, int l, int r, int
33 val, int xPos, int yPos) {
34 if (l == r) {
35 modifyY(1, 1, N, val, yPos, index,
36 true);
37 }
38 else {
39 int mid = (l + r) / 2;
40 if (xPos <= mid)
41 modifyX(index * 2, l, mid, val,
42 xPos, yPos);
43 else
44 modifyX(index * 2 + 1, mid + 1,
45 r, val, xPos, yPos);
46 modifyY(1, 1, N, val, yPos, index,
47 false);
48 }
49 }
50 void queryY(int index, int l, int r, int
51 yql, int yqr, int xIndex, int& vmax,
52 int& vmin) {
53 if (yql <= l && r <= yqr) {
54 vmax = max(vmax,
55 maxST[xIndex][index]);
56 vmin = min(vmin,
57 minST[xIndex][index]);
58 }
59 else {
60 int mid = (l + r) / 2;
61 if (yql <= mid)
62 queryY(index * 2, l, mid, yql,
63 yqr, xIndex, vmax, vmin);
64 if (mid < yqr)
65 queryY(index * 2 + 1, mid + 1, r,
66 yql, yqr, xIndex, vmax,
67 vmin);
68 }
69 }
70 void queryX(int index, int l, int r, int
71 xql, int xqr, int yql, int yqr, int&
72 vmax, int& vmin) {
73 if (xql <= l && r <= xqr) {
74 queryY(1, 1, N, yql, yqr, index,
75 vmax, vmin);
76 }
77 else {
78 int mid = (l + r) / 2;
79 if (xql <= mid)
80 queryX(index * 2, l, mid, xql,
81 xqr, yql, yqr, index,
82 false);
83 }
84 }
85 }

```

```

59 maxST[xIndex][index] =
60 minST[xIndex][index] = val;
61 return;
62 }
63 maxST[xIndex][index] =
64 max(maxST[xIndex * 2][index],
65 maxST[xIndex * 2 + 1][index]);
66 minST[xIndex][index] =
67 min(minST[xIndex * 2][index],
68 minST[xIndex * 2 + 1][index]);
69 }
70 else {
71 int mid = (l + r) / 2;
72 if (yPos <= mid)
73 modifyY(index * 2, l, mid, val,
74 yPos, xIndex, xIsLeaf);
75 else
76 modifyY(index * 2 + 1, mid + 1,
77 r, val, yPos, xIndex,
78 xIsLeaf);
79 }
80 maxST[xIndex][index] =
81 max(maxST[xIndex][index * 2],
82 maxST[xIndex][index * 2 + 1]);
83 minST[xIndex][index] =
84 min(minST[xIndex][index * 2],
85 minST[xIndex][index * 2 + 1]);
86 }
87 }
88 void modifyX(int index, int l, int r, int
89 val, int xPos, int yPos) {
90 if (l == r) {
91 modifyY(1, 1, N, val, yPos, index,
92 true);
93 }
94 else {
95 int mid = (l + r) / 2;
96 if (xPos <= mid)
97 modifyX(index * 2, l, mid, val,
98 xPos, yPos);
99 else
100 modifyX(index * 2 + 1, mid + 1,
101 r, val, xPos, yPos);
102 modifyY(1, 1, N, val, yPos, index,
103 false);
104 }
105 }
106 void queryY(int index, int l, int r, int
107 yql, int yqr, int xIndex, int& vmax,
108 int& vmin) {
109 if (yql <= l && r <= yqr) {
110 vmax = max(vmax,
111 maxST[xIndex][index]);
112 vmin = min(vmin,
113 minST[xIndex][index]);
114 }
115 else {
116 int mid = (l + r) / 2;
117 if (yql <= mid)
118 queryY(index * 2, l, mid, yql,
119 yqr, xIndex, vmax, vmin);
120 if (mid < yqr)
121 queryY(index * 2 + 1, mid + 1, r,
122 yql, yqr, xIndex, vmax,
123 vmin);
124 }
125 }
126 void queryX(int index, int l, int r, int
127 xql, int xqr, int yql, int yqr, int&
128 vmax, int& vmin) {
129 if (xql <= l && r <= xqr) {
130 queryY(1, 1, N, yql, yqr, index,
131 vmax, vmin);
132 }
133 else {
134 int mid = (l + r) / 2;
135 if (xql <= mid)
136 queryX(index * 2, l, mid, xql,
137 xqr, yql, yqr, index,
138 false);
139 }
140 }
141 }

```

```

59 queryX(index * 2, l, mid, xql,
60 xqr, yql, yqr, vmax, vmin);
61 if (mid < xqr)
62 queryX(index * 2 + 1, mid + 1, r,
63 xql, xqr, yql, yqr, vmax,
64 vmin);
65 }
66 }
67 int main() {
68 while (scanf("%d", &N) != EOF) {
69 int val;
70 for (int i = 1; i <= N; ++i) {
71 for (int j = 1; j <= N; ++j) {
72 scanf("%d", &val);
73 modifyX(1, 1, N, val, i, j);
74 }
75 }
76 int q;
77 int vmax, vmin;
78 int xql, xqr, yql, yqr;
79 char op;
80 scanf("%d", &q);
81 while (q--) {
82 getchar(); //for \n
83 scanf("%c", &op);
84 if (op == 'q') {
85 scanf("%d %d %d %d", &xql,
86 &yql, &xqr, &yqr);
87 vmax = -0x3f3f3f3f;
88 vmin = 0x3f3f3f3f;
89 queryX(1, 1, N, xql, xqr,
90 yql, yqr, vmax, vmin);
91 printf("%d %d\n", vmax, vmin);
92 }
93 else {
94 scanf("%d %d %d", &xql, &yql,
95 &val);
96 modifyX(1, 1, N, val, xql,
97 yql);
98 }
99 }
100 return 0;
101 }

```

4.5 權值線段樹

```

1 //權值線段樹 + 離散化 解決區間第k小問題
2 //其他網路上的解法：2個heap、Treap、AVL tree
3 #define maxn 30005
4 int nums[maxn];
5 int getArr[maxn];
6 int id[maxn];
7 int st[maxn << 2];
8 void update(int index, int l, int r, int qx)
9 {
10 if (l == r)
11 {
12 ++st[index];
13 return;
14 }
15 int mid = (l + r) / 2;
16 if (qx <= mid)
17 update(index * 2, l, mid, qx);
18 else
19 update(index * 2 + 1, mid + 1, r, qx);
20 st[index] = st[index * 2] + st[index * 2
21 + 1];
22 }
23 //找區間第k個小的
24 int query(int index, int l, int r, int k) {
25 if (l == r)
26 return id[l];
27 int mid = (l + r) / 2;
28 //k比左子樹小
29 if (k <= st[index * 2])

```

```

29     return query(index * 2, 1, mid, k);
30 else
31     return query(index * 2 + 1, mid + 1,
32                 r, k - st[index * 2]);
33 }
34 int main() {
35     int t;
36     cin >> t;
37     bool first = true;
38     while (t--) {
39         if (first)
40             first = false;
41         else
42             puts("");
43         memset(st, 0, sizeof(st));
44         int m, n;
45         cin >> m >> n;
46         for (int i = 1; i <= m; ++i) {
47             cin >> nums[i];
48             id[i] = nums[i];
49         }
50         for (int i = 0; i < n; ++i)
51             cin >> getArr[i];
52         //離散化
53         //防止m == 0
54         if (m)
55             sort(id + 1, id + m + 1);
56         int stSize = unique(id + 1, id + m + 1) - (id + 1);
57         for (int i = 1; i <= m; ++i) {
58             nums[i] = lower_bound(id + 1, id + stSize + 1, nums[i]) - id;
59         }
60         int addCount = 0;
61         int getCount = 0;
62         int k = 1;
63         while (getCount < n) {
64             if (getArr[getCount] == addCount) {
65                 printf("%d\n", query(1, 1, stSize, k));
66                 ++k;
67                 ++getCount;
68             }
69             else {
70                 update(1, 1, stSize, nums[addCount + 1]);
71                 ++addCount;
72             }
73         }
74         return 0;
75 }

```

4.6 Trie

```

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

```

```

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

```

4.7 AC Trie

```

1 const int maxn = 1e4 + 10; // 單字字數
2 const int maxl = 50 + 10; // 單字字長
3 const int maxc = 128; // 單字字符數
4 const char minc = ' '; // 首個 ASCII
5
6 int trie[maxn*maxl][maxc]; // 原字典樹
7 int val[maxn*maxl]; // 結尾(單字編號)
8 int cnt[maxn*maxl]; // 結尾(重複個數)
9 int fail[maxn*maxl]; // failure link
10 bool vis[maxn*maxl]; // 同單字不重複
11
12 struct ACTrie {
13     int seq, root;
14
15     ACTrie() {
16         seq = 0;
17         root = newNode();
18     }
19
20     int newNode() {
21         for(int i=0; i<maxc; i++) trie[seq][i]=0;
22         val[seq] = cnt[seq] = fail[seq] = 0;
23         return seq++;
24     }
25
26     void insert(char* s, int wordId=0) {
27         int p = root;
28         for(; *s; s++) {
29             int c = *s - minc;
30             if(!trie[p][c]) trie[p][c] = newNode();
31             p = trie[p][c];
32         }
33         val[p] = wordId;
34         cnt[p]++;
35     }
36
37     void build() {
38         queue<int> q({root});
39         while(!q.empty()) {

```

```

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

```

4.8 單調隊列

```

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

```

5 Geometry

5.1 Template

```

37 while(head<=tail&&a[q[tail]]<=a[i])tail--;
38 q[++tail]=i;
39 while(q[head]<=i-k) head++;
40 cout<<a[q[head]]<<" ";
41 }
42 cout<<endl;
43 }
44
45 int main(){
46     cin>>n>>k; //每k個連續的數
47     for(int i=1;i<=n;i++) cin>>a[i];
48     getmin();
49     getmax();
50     return 0;
51 }

```

```

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-8;
6 const TP inf = 1e30;
7 const int maxn = 5e4 + 10;
8
9 struct Vector {
10     TP x, y;
11     Vector(TP x=0, TP y=0): x(x), y(y) {}
12     DBL length();
13 };
14 using Point = Vector;
15 using Polygon = vector<Point>;
16
17 Vector operator+(Vector a, Vector b) {
18     return Vector(a.x+b.x, a.y+b.y); }
19 Vector operator-(Vector a, Vector b) {
20     return Vector(a.x-b.x, a.y-b.y); }
21 Vector operator*(Vector a, DBL b) {
22     return Vector(a.x*b, a.y*b); }
23 Vector operator/(Vector a, DBL b) {
24     return Vector(a.x/b, a.y/b); }
25
26 TP dot(Vector a, Vector b) {
27     return a.x*b.x + a.y*b.y;
28 }
29 TP cross(Vector a, Vector b) {
30     return a.x*b.y - a.y*b.x;
31 }
32 DBL Vector::length() {
33     return sqrt(dot(*this, *this));
34 }
35 DBL dis(Point a, Point b) {
36     return sqrt(dot(a-b, a-b));
37 }
38
39 Vector unit_normal_vector(Vector v) {
40     DBL len = v.length();
41     return Vector(-v.y/len, v.x/len);
42 }
43
44 struct Line {
45     Point p;
46     Vector v;
47     DBL ang;
48     Line(Point _p={}, Vector _v={}) {
49         p = _p;
50         v = _v;
51         ang = atan2(v.y, v.x);
52     }
53     bool operator<(const Line& l) const {
54         return ang < l.ang;
55     };
56
57 struct Segment {
58     Point s, e;
59     Segment(): s({0, 0}), e({0, 0}) {}
60     Segment(Point s, Point e): s(s), e(e) {}
61     DBL length() { return dis(s, e); }
62 };
63
64 struct Circle {
65     Point o;
66     DBL r;
67     Circle(): o({0, 0}), r(0) {}
68     Circle(Point o, DBL r=0): o(o), r(r) {}
69     Circle(Point a, Point b) { // ab 直徑
70         o = (a + b) / 2;
71         r = dis(o, a);
72     }
73     Circle(Point a, Point b, Point c) {
74         Vector u = b-a, v = c-a;

```

```

75     DBL c1=dot(u, a+b)/2, c2=dot(v, a+c)/2;
76     DBL dx=c1*v.y-c2*u.y, dy=u.x*c2-v.x*c1;
77     o = Point(dx, dy) / cross(u, v);
78     r = dis(o, a);
79 }
80 bool cover(Point p) {
81     return dis(o, p) <= r;
82 }
83 };

```

5.2 Polygon

```

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

```

5.3 Intersection

```

1 // 除 intersection(Line a, Line b) 之外，
2 // 皆尚未丟 online judge
3
4 int dcmp(DBL a, DBL b=0.0) {
5     return (a > b) - (a < b);
6 }
7
8 bool hasIntersection(Point p, Segment s) {
9     return dcmp(cross(p-s.s, s.s-s.e))==0&&
10        dcmp(dot(p.x-s.s.x, p.x-s.e.x))<=0&&
11        dcmp(dot(p.y-s.s.y, p.y-s.e.y))<=0;
12 }
13
14 bool hasIntersection(Point p, Line l) {
15     return dcmp(cross(p-l.p, l.v)) == 0;
16 }
17
18 DBL dis(Line l, Point p) {
19     DBL t = cross(p, l.v) + cross(l.v, l.p);
20     return abs(t) / sqrt(dot(l.v, l.v));
21 }
22
23 Point intersection(Line a, Line b) {
24     Vector u = a.p - b.p;
25     DBL t = 1.0*cross(b.v, u)/cross(a.v, b.v);
26     return a.p + a.v*t;
27 }

```

5.4 最小圓覆蓋


```

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

```

5.5 旋轉卡尺

```

1 // 回傳凸包內最遠兩點的距離
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;
10    for(int i=0, j=1, n=p.size(); i<n; i++) {
11        Line l(p[i], p[(i+1)%n] - p[i]);
12        for(; test(l, p[j], p[(j+1)%n]); j=(j+1)%n);
13        mx = max({
14            mx,
15            dot(p[(i+1)%n]-p[j], p[(i+1)%n]-p[j]),
16            dot(p[i]-p[j], p[i]-p[j])
17        });
18    }
19    return mx;
20 }

```

5.6 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:
 - ang, 該點與基準點的 atan2 值
 - d2, 該點與基準點的 (距離)²
- 注意計算 d2 的型態範圍限制

```

1 using TP = long long;
2 using Polygon = vector<Point>;
3
4 const TP inf = 1e9; // 座標點最大值
5
6 Polygon convex_hull(Point* p, int n) {
7     auto dblcmp = [](DBL a, DBL b=0.0) {
8         return (a>b) - (a<b);
9     };
10    auto rmv = [&](Point a, Point b, Point c) {
11        return cross(b-a, c-b) <= 0; // 非浮點數
12        return dblcmp(cross(b-a, c-b)) <= 0;
13    };
14
15    // 選最下裡最左的當基準點, 可在輸入時計算
16    TP lx = inf, ly = inf;
17    for(int i=0; i<n; i++) {
18        if(p[i].y<ly || (p[i].y==ly&&p[i].x<lx)){
19            lx = p[i].x, ly = p[i].y;
20        }
21    }
22
23    for(int i=0; i<n; i++) {
24        p[i].ang=atan2(p[i].y-ly, p[i].x-lx);
25        p[i].d2 = (p[i].x-lx)*(p[i].x-lx) +
26                (p[i].y-ly)*(p[i].y-ly);
27    }
28    sort(p, p+n, [&](Point& a, Point& b) {

```

```

29     if(dblcmp(a.ang, b.ang))
30         return a.ang < b.ang;
31     return a.d2 < b.d2;
32 });
33
34 int m = 1; // stack size
35 Point st[n] = {p[n]=p[0]};
36 for(int i=1; i<=n; i++) {
37     for(; m>1&&rmv(st[m-2], st[m-1], p[i]); m--);
38     st[m++] = p[i];
39 }
40 return Polygon(st, st+m-1);
41 }

```

5.7 半平面相交

```

1 using DBL = double;
2 using TP = DBL; // 存點的型態
3 using Polygon = vector<Point>;
4
5 const int maxn = 5e4 + 10;
6
7 // Return: 能形成半平面交的凸包邊界點
8 Polygon halfplaneIntersect(vector<Line>& nar){
9     sort(nar.begin(), nar.end());
10    // DBL 跟 0 比較, 沒符號數不用
11    auto dblcmp = [](DBL v){return (v>0)-(v<0);};
12    // p 是否在 l 的左半平面
13    auto lft = [&](Point p, Line l) {
14        return dblcmp(cross(l.v, p-l.p)) > 0;
15    };
16
17    int ql = 0, qr = 0;
18    Line L[maxn] = {nar[0]};
19    Point P[maxn];
20
21    for(int i=1; i<nar.size(); i++) {
22        for(; ql<qr&&!lft(P[qr-1], nar[i]); qr--);
23        for(; ql<qr&&!lft(P[ql], nar[i]); ql++);
24        L[++qr] = nar[i];
25        if(dblcmp(cross(L[qr].v, L[qr-1].v))==0) {
26            if(lft(nar[i].p, L[--qr])) L[qr]=nar[i];
27        }
28        if(ql < qr)
29            P[qr-1] = intersection(L[qr-1], L[qr]);
30    }
31    for(; ql<qr && !lft(P[qr-1], L[ql]); qr--);
32    if(qr-ql <= 1) return {};
33    P[qr] = intersection(L[qr], L[ql]);
34    return Polygon(P+ql, P+qr+1);
35 }

```

6 DP

6.1 以價值為主的背包

```

1 /*w 變得太大所以一般的 01 背包解法變得不可能
2 觀察題目 w 變成 10^9
3 而 v_i 變成 10^3
4 N 不變 10^2
5 試著湊湊看 dp 狀態
6 dp[maxn][maxv] 是可接受的複雜度
7 剩下的是轉移式, 轉移式變成
8 dp[i][j] = w ->
9     當目前只考慮到第 i 個商品時, 達到獲利 j 時最少的 weight
10     = w
11 所以答案是 dp[n][1 ~ maxv] 找價值最大且裝的下的 */
12 #define maxn 105
13 #define maxv 100005
14 long long dp[maxn][maxv];
15 long long weight[maxn];
16 long long v[maxn];
17 int main() {
18     int n;
19     long long w;
20     scanf("%d %lld", &n, &w);
21     for (int i = 1; i <= n; ++i) {
22         scanf("%lld %lld", &weight[i], &v[i]);
23     }
24     memset(dp, 0x3f, sizeof(dp));
25     dp[0][0] = 0;
26     for (int i = 1; i <= n; ++i) {
27         for (int j = 0; j <= maxv; ++j) {
28             if (j - v[i] >= 0)
29                 dp[i][j] = dp[i-1][j - v[i]] + weight[i];
30             dp[i][j] = min(dp[i-1][j], dp[i][j]);
31         }
32     }
33     long long res = 0;
34     for (int j = maxv - 1; j >= 0; --j) {
35         if (dp[n][j] <= w) {
36             res = j;
37             break;
38         }
39     }
40     printf("%lld\n", res);
41     return 0;
42 }

```

6.2 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];

```

6.3 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     return 0;
26 }

```

6.4 Deque 最大差距

```

1 /*定義dp[l][r]是l ~ r時與先手最大差異值
2 轉移式: dp[l][r] = max{a[l] - solve(l + 1,
3     r), a[r] - solve(l, r - 1)}
4 裡面用減的主要是因為求的是相減且會一直換手,
5 所以正負正負...*/
6 #define maxn 3005
7 bool vis[maxn][maxn];
8 long long dp[maxn][maxn];
9 long long a[maxn];
10 long long solve(int l, int r) {
11     if (l > r) return 0;
12     if (vis[l][r]) return dp[l][r];
13     vis[l][r] = true;
14     long long res = a[l] - solve(l + 1, r);
15     res = max(res, a[r] - solve(l, r - 1));
16     return dp[l][r] = res;
17 }
18 int main() {
19     ...
20     printf("%lld\n", solve(1, n));
21 }

```

6.5 LCS 和 LIS

```

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

```

6.6 RangeDP

```

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 }

```

6.7 stringDP

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

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

6.8 樹 DP 有幾個 path 長度為 k

```

1 #define maxn 50005
2 #define maxk 505
3 //dp[u][u的child距距離u長度k的數量]
4 long long dp[maxn][maxk];
5 vector<vector<int>>> G;
6 int n, k;
7 long long res = 0;
8 void dfs(int u, int p) {
9     //u自己
10    dp[u][0] = 1;
11    for (int v: G[u]) {
12        if (v == p)
13            continue;
14        dfs(v, u);
15        for (int i = 1; i <= k; ++i) {
16            //子樹v距離i - 1的等於對於u來說距離i的
17            dp[u][i] += dp[v][i - 1];
18        }
19    }
20    //統計在u子樹中距離u為k的數量
21    res += dp[u][k];
22    long long cnt = 0;
23    for (int v: G[u]) {
24        if (v == p)
25            continue; //重點算法
26        for (int x = 0; x <= k - 2; ++x) {
27            cnt +=
28                dp[v][x] * (dp[u][k - x - 1] - dp[v][k - x - 2]);
29        }
30    }
31    res += cnt / 2;
32 }
33 int main() {

```

```

34     ...
35     dfs(1, -1);
36     printf("%lld\n", res);
37     return 0;
38 }

```

6.9 TreeDP reroot

```

1 /*re-root dp on tree 0(n + n + n) -> 0(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->res[0] = this->dfs2(0, 0);
18        memset(this->visited, 0,
19            sizeof(this->visited));
20        this->dfs3(0, n);
21        return this->res;
22    }
23 private:
24     vector<vector<int>>> G;
25     bool visited[30005];
26     int subtreeSize[30005];
27     vector<int> res;
28     //求subtreeSize
29     int dfs(int u) {
30         this->visited[u] = true;
31         for (int v: this->G[u])
32             if (!this->visited[v])
33                 this->subtreeSize[u] +=
34                     this->dfs(v);
35         //自己
36         this->subtreeSize[u] += 1;
37         return this->subtreeSize[u];
38     }
39     //求res[0], 0到所有點的距離
40     int dfs2(int u, int dis) {
41         this->visited[u] = true;
42         int sum = 0;
43         for (int v: this->G[u])
44             if (!visited[v])
45                 sum += this->dfs2(v, dis + 1);
46         //要加上自己的距離
47         return sum + dis;
48     }
49     //算出所有的res
50     void dfs3(int u, int n) {
51         this->visited[u] = true;
52         for (int v: this->G[u]) {
53             if (!visited[v]) {
54                 this->res[v] = this->res[u] +
55                     n - 2 *
56                     this->subtreeSize[v];
57                 this->dfs3(v, n);
58             }
59         }
60     }
61 };

```

6.10 WeightedLIS

```
1 #define maxn 200005
2 long long dp[maxn];
3 long long height[maxn];
4 long long B[maxn];
5 long long st[maxn << 2];
6 void update(int p, int index, int l, int r,
7             long long v) {
8     if (l == r) {
9         st[index] = v;
10        return;
11    }
12    int mid = (l + r) >> 1;
13    if (p <= mid)
14        update(p, (index << 1), l, mid, v);
15    else
16        update(p, (index << 1)+1, mid+1, r, v);
17    st[index] =
18        max(st[index<<1], st[(index<<1)+1]);
19 }
20 long long query(int index, int l, int r, int
21                ql, int qr) {
22    if (ql <= l && r <= qr)
23        return st[index];
24    int mid = (l + r) >> 1;
25    long long res = -1;
26    if (ql <= mid)
27        res =
28            max(res, query(index<<1, l, mid, ql, qr));
29    if (mid < qr)
30        res =
31            max(res, query((index<<1)+1, mid+1, r, ql, qr));
32    return res;
33 }
34 int main() {
35     int n;
36     scanf("%d", &n);
37     for (int i = 1; i <= n; ++i)
38         scanf("%lld", &height[i]);
39     for (int i = 1; i <= n; ++i)
40         scanf("%lld", &B[i]);
41     long long res = B[1];
42     update(height[1], 1, 1, n, B[1]);
43     for (int i = 2; i <= n; ++i) {
44         long long temp;
45         if (height[i] - 1 >= 1)
46             temp =
47                 B[i]+query(1,1,n,1,height[i]-1);
48         else
49             temp = B[i];
50         update(height[i], 1, 1, n, temp);
51         res = max(res, temp);
52     }
53     printf("%lld\n", res);
54     return 0;
55 }
```