

## Contents

### 1 字串

- 1.1 最長迴文子字串 . . . . .
- 1.2 KMP . . . . .

### 2 STL

- 2.1 multiset . . . . .
- 2.2 unordered\_set . . . . .

### 3 math

- 3.1 質數與因數 . . . . .
- 3.2 歐拉函數 . . . . .
- 3.3 atan . . . . .
- 3.4 大步小步 . . . . .

### 4 algorithm

- 4.1 basic . . . . .
- 4.2 二分搜 . . . . .
- 4.3 三分搜 . . . . .
- 4.4 差分 . . . . .
- 4.5 greedy . . . . .
- 4.6 dinic . . . . .
- 4.7 Nim Game . . . . .
- 4.8 SCC Tarjan . . . . .
- 4.9 ArticulationPoints Tarjan . . . . .
- 4.10 最小樹狀圖 . . . . .
- 4.11 Astar . . . . .
- 4.12 JosephusProblem . . . . .
- 4.13 KM . . . . .
- 4.14 LCA 倍增法 . . . . .
- 4.15 LCA 樹壓平 RMQ . . . . .
- 4.16 MCMF . . . . .
- 4.17 莫隊 . . . . .
- 4.18 Dancing Links . . . . .

### 5 DataStructure

- 5.1 BIT . . . . .
- 5.2 ChthollyTree . . . . .
- 5.3 線段樹 1D . . . . .
- 5.4 線段樹 2D . . . . .
- 5.5 權值線段樹 . . . . .
- 5.6 Trie . . . . .
- 5.7 單調隊列 . . . . .

### 6 geometry

- 6.1 intersection . . . . .
- 6.2 半平面相交 . . . . .
- 6.3 凸包 . . . . .

### 7 DP

- 7.1 以價值為主的背包 . . . . .
- 7.2 抽屜 . . . . .
- 7.3 Barcode . . . . .
- 7.4 Deque 最大差距 . . . . .
- 7.5 LCS 和 LIS . . . . .
- 7.6 RangeDP . . . . .
- 7.7 stringDP . . . . .
- 7.8 TreeDP 有幾個 path 長度為 k . . . . .
- 7.9 TreeDP reroot . . . . .
- 7.10 WeightedLIS . . . . .

## 1 字串

### 1.1 最長迴文子字串

```

1 #include<bits/stdc++.h>
2 #define T(x) ((x)%2 ? s[(x)/2] : '. ')
3 using namespace std;
4
5 string s;
6 int n;
7
8 int ex(int l,int r){
9     int i=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 }
```

### 1.2 KMP

```

1 #define maxn 1000005
2 int nextArr[maxn];
3 void getNextArr(const string& str) {
4     nextArr[0] = 0;
5     int prefixLen = 0;
6     for (int i = 1; i < str.size(); ++i) {
7         prefixLen = nextArr[i - 1];
8         //如果不一樣就在之前算過的prefix中搜有沒有更短的前後綴
9         while (prefixLen > 0 && str[prefixLen] !=
10             str[i])
11             prefixLen = nextArr[prefixLen - 1];
12         //一樣就繼承之前的前後綴長度+1
13         if (str[prefixLen] == str[i])
14             ++prefixLen;
15         nextArr[i] = prefixLen;
16     }
17     for (int i = 0; i < str.size() - 1; ++i) {
18         vis[nextArr[i]] = true;
19     }
20 }
```

## 2 STL

### 2.1 multiset

```

1 與 set 用法雷同，但會保留重複的元素。
2 資料由小到大排序。
3 宣告：
4     multiset<int> st;
5 刪除資料：
6     st.erase(val);
7     //會刪除所有值為 val 的元素。
8     st.erase(st.find(val));
9     //只刪除第一個值為 val 的元素。
```

### 2.2 unordered\_set

```

1 unordered_set 的實作方式通常是用雜湊表(hash table)，
2 資料插入和查詢的時間複雜度很低，為常數級別O(1)，
3 相對的代價是消耗較多的記憶體，空間複雜度較高，
4 無自動排序功能。
5
6 unordered_set 判斷元素是否存在
```

```

7 unordered_set<int> myunordered_set;
8 myunordered_set.insert(2);
9 myunordered_set.insert(4);
10 myunordered_set.insert(6);
11 cout << myunordered_set.count(4) << "\n"; // 1
12 cout << myunordered_set.count(8) << "\n"; // 0

```

### 3 math

#### 3.1 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 prime[MAXN];
5 int primeSize=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]) prime[primeSize++]=i;
11         for(int j=0; j<primeSize&&i*prime[j]<=MAXN; ++j){
12             isPrime[i*prime[j]]=false;
13             if(i%prime[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<(int)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
37 //ax+by=GCD(a,b)
38
39 int ext_euc(int a, int b, int &x, int &y){
40     if(b==0){
41         x=1, y=0;
42         return a;
43     }
44     int d=ext_euc(b, a%b, y, x);
45     y-=a/b*x;
46     return d;
47 }
48
49 int main(){
50     int a, b, x, y;
51     cin>>a>>b;
52     ext_euc(a, b, x, y);
53     cout<<x<<' ' <<y<<endl;
54     return 0;
55 }
56
57
58
59 歌德巴赫猜想
60 solution : 把偶數 N (6≤N≤10^6) 寫成兩個質數的和。
61 #define N 20000000
62 int ox[N], p[N], pr;

```

```

63 void PrimeTable(){
64     ox[0]=ox[1]=1;
65     pr=0;
66     for(int i=2; i<N; i++){
67         if(!ox[i]) p[pr++]=i;
68         for(int j=0; i*p[j]<N&&j<pr; j++){
69             ox[i*p[j]]=1;
70         }
71     }
72
73 int main(){
74     PrimeTable();
75     int n;
76     while(cin>>n, n){
77         int x;
78         for(x=1; x+=2)
79             if(!ox[x]&&!ox[n-x]) break;
80         printf("%d = %d + %d\n", n, x, n-x);
81     }
82 }
83
84 problem : 給定整數 N ,
85 求 N 最少可以拆成多少個質數的和。
86 如果 N 是質數, 則答案為 1。
87 如果 N 是偶數(不包含2), 則答案為 2 (強歌德巴赫猜想)。
88 如果 N 是奇數且 N-2 是質數, 則答案為 2 (2+質數)。
89 其他狀況答案為 3 (弱歌德巴赫猜想)。
90
91 bool isPrime(int n){
92     for(int i=2; i<n; ++i){
93         if(i*i>n) return true;
94         if(n%i==0) return false;
95     }
96     return true;
97 }
98
99 int main(){
100     int n;
101     cin>>n;
102     if(isPrime(n)) cout<<"1\n";
103     else if(n%2==0||isPrime(n-2)) cout<<"2\n";
104     else cout<<"3\n";
105 }

```

#### 3.2 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    }
11    if(n>1) ans=ans-ans/n;
12    return ans;
13 }

```

#### 3.3 atan

```

1 說明
2     atan() 和 atan2() 函數分別計算 x 和 y/x 的反正切。
3
4 回傳值
5     atan() 函數會傳回介於範圍 -pi/2 到 pi/2
6     弧度之間的值。
7     atan2() 函數會傳回介於 -pi 至 pi 弧度之間的值。
8     如果 atan2() 函數的兩個引數都是零,
9     則函數會將 errno 設為 EDOM, 並傳回值 0。
10
11 範例
12 int main(void){

```

```

12 double a,b,c,d;
13 c=0.45;
14 d=0.23;
15 a=atan(c);
16 b=atan2(c,d);
17 printf("atan(%lf)=%lf/n",c,a);
18 printf("atan2(%lf,%lf)=%lf/n",c,d,b);
19 }
20 // atan(0.450000)=0.422854
21 // atan2(0.450000,0.230000)=1.098299

```

### 3.4 大步小步

```

1 題意
2 給定 B,N,P，求出 L 滿足  $B^L \equiv N \pmod{P}$ 。
3 題解
4 餘數的循環節長度必定為 P 的因數，因此
    $B^0, B^1, B^2, \dots, B^{P-1}, \dots$ ，
5 也就是說如果有解則  $L < N$ ，枚舉 0,1,2,L-1
   能得到結果，但會超時。
6 將 L 拆成  $mx+y$ ，只要分別枚舉 x,y 就能得到答案，
7 設  $m=\sqrt{P}$  能保證最多枚舉  $2\sqrt{P}$  次。
8  $B^{mx+y} \equiv N \pmod{P}$ 
9  $B^{mx} B^y \equiv N \pmod{P}$ 
10  $B^y \equiv N(B^{-m})^x \pmod{P}$ 
11 先求出  $B^0, B^1, B^2, \dots, B^{m-1}$ ，
12 再枚舉  $N(B^{-m}), N(B^{-m})^2, \dots$  查看是否有對應的  $B^y$ 。
13 這種算法稱為大步小步演算法，
14 大步指的是枚舉 x（一次跨 m 步），
15 小步指的是枚舉 y（一次跨 1 步）。
16 複雜度分析
17 利用 map/unorder_map 存放  $B^0, B^1, B^2, \dots, B^{m-1}$ ，
18 枚舉 x 查詢 map/unorder_map 是否有對應的  $B^y$ ，
19 存放和查詢最多  $2\sqrt{P}$  次，時間複雜度為  $O(\sqrt{P} \log \sqrt{P}) / O(\sqrt{P})$ 。
20
21 using LL = long long;
22 LL B, N, P;
23 LL fpow(LL a, LL b, LL c){
24     LL res=1;
25     for(;b>=1;){
26         if(b&1)
27             res=(res*a)%c;
28         a=(a*a)%c;
29     }
30     return res;
31 }
32 LL BSGS(LL a, LL b, LL p){
33     a%=p, b%=p;
34     if(a==0)
35         return b==0?1:-1;
36     if(b==1)
37         return 0;
38     map<LL, LL> tb;
39     LL sq=ceil(sqrt(p-1));
40     LL inv=fpow(a, p-sq-1, p);
41     tb[1]=sq;
42     for(LL i=1, tmp=1; i<sq; ++i){
43         tmp=(tmp*a)%p;
44         if(!tb.count(tmp))
45             tb[tmp]=i;
46     }
47     for(LL i=0; i<sq; ++i){
48         if(tb.count(b)){
49             LL res=tb[b];
50             return i*sq+(res==sq?0:res);
51         }
52         b=(b*inv)%p;
53     }
54     return -1;
55 }
56 int main(){
57     IOS; //輸入優化
58     while(cin>>P>>B>>N){

```

```

59         LL ans=BSGS(B,N,P);
60         if(ans!=-1)
61             cout<<"no solution\n";
62         else
63             cout<<ans<<"\n";
64     }
65 }

```

## 4 algorithm

### 4.1 basic

```

1 min_element：找尋最小元素
2 min_element(first, last)
3 max_element：找尋最大元素
4 max_element(first, last)
5 sort：排序，預設由小排到大。
6 sort(first, last)
7 sort(first, last, cmp)：可自行定義比較運算子 cmp。
8 find：尋找元素。
9 find(first, last, val)
10 lower_bound：尋找第一個小於 x 的元素位置，
   如果不存在，則回傳 last。
11 lower_bound(first, last, val)
12 upper_bound：尋找第一個大於 x 的元素位置，
   如果不存在，則回傳 last。
13 upper_bound(first, last, val)
14 next_permutation：將序列順序轉換成下一個字典序，
   如果存在回傳 true，反之回傳 false。
15 next_permutation(first, last)
16 prev_permutation：將序列順序轉換成上一個字典序，
   如果存在回傳 true，反之回傳 false。
17 prev_permutation(first, last)

```

### 4.2 二分搜

```

1 int binary_search(int target) {
2     // For range [ok, ng) or (ng, ok], "ok" is for the
3     // index that target value exists, with "ng" doesn't.
4     int ok = maxn, ng = -1;
5     // For first lower_bound, ok=maxn and ng=-1,
6     // for last lower_bound, ok = -1 and ng = maxn
7     // (the "check" funtion
8     // should be changed depending on it.)
9     while(abs(ok - ng) > 1) {
10         int mid = (ok + ng) >> 1;
11         if(check(mid)) ok = mid;
12         else ng = mid;
13     }
14     // Be careful, "arr[mid]>=target" for first
15     // lower_bound and "arr[mid]<=target" for
16     // last lower_bound. For range (ng, ok],
17     // convert it into (ng, mid] and (mid, ok] than
18     // choose the first one, or convert [ok, ng) into
19     // [ok, mid) and [mid, ng) and then choose
20     // the second one.
21     return ok;
22 }
23 lower_bound(arr, arr + n, k); //最左邊 ≥ k 的位置
24 upper_bound(arr, arr + n, k); //最左邊 > k 的位置
25 upper_bound(arr, arr + n, k) - 1; //最右邊 ≤ k 的位置
26 lower_bound(arr, arr + n, k) - 1; //最右邊 < k 的位置
27 (lower_bound, upper_bound) //等於 k 的範圍
28 equal_range(arr, arr+n, k);

```

### 4.3 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, Point& p)
12     {
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 int main(){
36     IOS; //輸入優化
37     int T;
38     cin>>T;
39     for(int ti=1; ti<=T; ++ti){
40         double time;
41         Point x1, y1, d1, x2, y2, d2;
42         cin>>time>>x1>>y1>>x2>>y2;
43         d1=(y1-x1)/time;
44         d2=(y2-x2)/time;
45         double L=0, R=1e8, m1, m2, f1, f2;
46         double ans = x1.dist(x2);
47         while(abs(L-R)>1e-10){
48             m1=(L+R)/2;
49             m2=(m1+R)/2;
50             f1=((d1*m1)+x1).dist((d2*m1)+x2);
51             f2=((d1*m2)+x1).dist((d2*m2)+x2);
52             ans = min(ans, min(f1, f2));
53             if(f1<f2) R=m2;
54             else L=m1;
55         }
56         cout<<"Case " <<ti<<": ";
57         cout<<fixed<<setprecision(4)<<sqrt(ans)<<'\n';
58     }
59 }

```

## 4.4 差分

```

1  用途：在區間  $[l, r]$  加上一個數字  $v$ 。
2   $b[l] += v$ ; ( $b[0 \sim l]$  加上  $v$ )
3   $b[r+1] -= v$ ; ( $b[r+1 \sim n]$  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  $a[i] = b[0] + b[1] + b[2] + \dots + b[i]$ ，
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 }

```

## 4.5 greedy

1 貪心演算法的核心為，  
2 採取在目前狀態下最好或最佳（即最有利）的選擇。  
3 貪心演算法雖然能獲得當前最佳解，  
4 但不保證能獲得最後（全域）最佳解，  
5 提出想法後可以先試圖尋找有沒有能推翻原本的想法的反例，  
6 確認無誤再實作。

### 刪數字問題

9 //problem  
10 給定一個數字  $N(\leq 10^{100})$ ，需要刪除  $K$  個數字，  
11 請問刪除  $K$  個數字後最小的數字為何？  
12 //solution  
13 刪除滿足第  $i$  位數大於第  $i+1$  位數的最左邊第  $i$  位數，  
14 扣除高位數的影響較扣除低位數的大。

```

15 //code
16 int main(){
17     string s;
18     int k;
19     cin>>s>>k;
20     for(int i=0; i<k; ++i){
21         if((int)s.size()==0) break;
22         int pos = (int)s.size()-1;
23         for(int j=0; j<(int)s.size()-1; ++j){
24             if(s[j]>s[j+1]){
25                 pos=j;
26                 break;
27             }
28         }
29         s.erase(pos, 1);
30     }
31     while((int)s.size()>0&&s[0]=='0')
32         s.erase(0, 1);
33     if((int)s.size()) cout<<s<<'\n';
34     else cout<<0<<'\n';
35 }

```

### 最小區間覆蓋長度

37 //problem  
38 給定  $n$  條線段區間為  $[L_i, R_i]$ ，  
39 請問最少要選幾個區間才能完全覆蓋  $[0, S]$ ？

40 //solution  
41 先將所有區間依照左界由小到大排序，  
42 對於當前區間  $[L_i, R_i]$ ，要從左界  $> R_i$  的所有區間中，  
43 找到有著最大的右界的區間，連接當前區間。

### 長度 $n$ 的直線中有數個加熱器，

46 在  $x$  的加熱器可以讓  $[x-r, x+r]$  內的物品加熱，  
47 問最少要幾個加熱器可以把  $[0, n]$  的範圍加熱。  
48 //solution

49 對於最左邊沒加熱的點  $a$ ，選擇最遠可以加熱  $a$  的加熱器，  
50 更新已加熱範圍，重複上述動作繼續尋找加熱器。

```

51 //code
52 int main(){
53     int n, r;
54     int a[1005];

```

```

56     cin>>n>>r;
57     for(int i=1;i<=n;++i) cin>>a[i];
58     int i=1,ans=0;
59     while(i<=n){
60         int R=min(i+r-1,n),L=max(i-r+1,0)
61         int nextR=-1;
62         for(int j=R;j>=L;--j){
63             if(a[j]){
64                 nextR=j;
65                 break;
66             }
67         }
68         if(nextR!=-1){
69             ans=-1;
70             break;
71         }
72         ++ans;
73         i=nextR+r;
74     }
75     cout<<ans<<'\n';
76 }
77 最多不重疊區間
78 //problem
79 給你 n 條線段區間為 [Li,Ri]，
80 請問最多可以選擇幾條不重疊的線段(頭尾可相連)?
81 //solution
82 依照右界由小到大排序，
83 每次取到一個不重疊的線段，答案 +1。
84 //code
85 struct Line{
86     int L,R;
87     bool operator<(const Line &rhs)const{
88         return R<rhs.R;
89     }
90 };
91 int main(){
92     int t;
93     cin>>t;
94     Line a[30];
95     while(t--){
96         int n=0;
97         while(cin>>a[n].L>>a[n].R,a[n].L||a[n].R)
98             ++n;
99         sort(a,a+n);
100        int ans=1,R=a[0].R;
101        for(int i=1;i<n;i++){
102            if(a[i].L>=R){
103                ++ans;
104                R=a[i].R;
105            }
106        }
107        cout<<ans<<'\n';
108    }
109 }
110 最小化最大延遲問題
111 //problem
112 給定 N 項工作，每項工作的需要處理時長為  $T_i$ ，
113 期限是  $D_i$ ，第  $i$  項工作延遲的時間為  $L_i=\max(0, F_i-D_i)$ ，
114 原本  $F_i$  為第  $i$  項工作的完成時間，
115 求一種工作排序使  $\max L_i$  最小。
116 //solution
117 按照到期時間從早到晚處理。
118 //code
119 struct Work{
120     int t, d;
121     bool operator<(const Work &rhs)const{
122         return d<rhs.d;
123     }
124 };
125 int main(){
126     int n;
127     Work a[10000];
128     cin>>n;
129     for(int i=0;i<n;++i)
130         cin>>a[i].t>>a[i].d;
131     sort(a,a+n);

```

```

132     int maxL=0,sumT=0;
133     for(int i=0;i<n;++i){
134         sumT+=a[i].t;
135         maxL=max(maxL,sumT-a[i].d);
136     }
137     cout<<maxL<<'\n';
138 }
139 最少延遲數量問題
140 //problem
141 給定 N 個工作，每個工作的需要處理時長為  $T_i$ ，
142 期限是  $D_i$ ，求一種工作排序使得逾期工作數量最小。
143 //solution
144 期限越早到期的工作越先做。將工作依照到期時間從早到晚排序，
145 依序放入工作列表中，如果發現有工作預期，
146 就從目前選擇的工作中，移除耗時最長的工作。
147 上述方法為 Moore-Hodgson's Algorithm。
148 //problem
149 給定烏龜的重量和可承受重量，問最多可以疊幾隻烏龜？
150 //solution
151 和最少延遲數量問題是相同的問題，只要將題敘做轉換。
152 工作處理時長  $\rightarrow$  烏龜重量
153 工作期限  $\rightarrow$  烏龜可承受重量
154 多少工作不延期  $\rightarrow$  可以疊幾隻烏龜
155 //code
156 struct Work{
157     int t, d;
158     bool operator<(const Work &rhs)const{
159         return d<rhs.d;
160     }
161 };
162 };
163 int main(){
164     int n=0;
165     Work a[10000];
166     priority_queue<int> pq;
167     while(cin>>a[n].t>>a[n].d)
168         ++n;
169     sort(a,a+n);
170     int sumT=0,ans=n;
171     for(int i=0;i<n;++i){
172         pq.push(a[i].t);
173         sumT+=a[i].t;
174         if(a[i].d<sumT){
175             int x=pq.top();
176             pq.pop();
177             sumT-=x;
178             --ans;
179         }
180     }
181     cout<<ans<<'\n';
182 }
183
184 任務調度問題
185 //problem
186 給定 N 項工作，每項工作的需要處理時長為  $T_i$ ，
187 期限是  $D_i$ ，如果第  $i$  項工作延遲需要受到  $p_i$  單位懲罰，
188 請問最少會受到多少單位懲罰。
189 //solution
190 依照懲罰由大到小排序，
191 每項工作依序嘗試可不可以放在  $D_i-T_i+1, D_i-T_i, \dots, 1, 0$ ，
192 如果有空閒就放進去，否則延後執行。
193
194 //problem
195 給定 N 項工作，每項工作的需要處理時長為  $T_i$ ，
196 期限是  $D_i$ ，如果第  $i$  項工作在期限內完成會獲得  $a_i$ 
197 單位獎勵，
198 請問最多會獲得多少單位獎勵。
199 //solution
200 和上題相似，這題變成依照獎勵由大到小排序。
201 //code
202 struct Work{
203     int d,p;
204     bool operator<(const Work &rhs)const{
205         return p>rhs.p;
206     }

```

```

206 };
207 int main(){
208     int n;
209     Work a[100005];
210     bitset<100005> ok;
211     while(cin>>n){
212         ok.reset();
213         for(int i=0;i<n;++i)
214             cin>>a[i].d>>a[i].p;
215         sort(a,a+n);
216         int ans=0;
217         for(int i=0;i<n;++i){
218             int j=a[i].d;
219             while(j--){
220                 if(!ok[j]){
221                     ans+=a[i].p;
222                     ok[j]=true;
223                     break;
224                 }
225             }
226             cout<<ans<<'\\n';
227         }
228     }

```

#### 4.6 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 && e.cap>e.flow) {
32                 level[e.t] = level[e.s] + 1;
33                 q.push(e.t);
34             }
35         }
36     }
37     return level[T];
38 }
39 int dfs(int cur, int lim) {
40     if(cur==T || lim==0) return lim;
41     int result = 0;
42     for(int& i=dfs_idx[cur]; i<G[cur].size() && lim; 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, e.cap-e.flow));
46         if(flow <= 0) continue;
47         e.flow += flow;
48         result += flow;
49         E[G[cur][i]^1].flow -= flow;

```

```

50         lim -= flow;
51     }
52     return result;
53 }
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 }

```

#### 4.7 Nim Game

```

1  //兩人輪流取銅板，每人每次需在某堆取一枚以上的銅板，
2  //但不能同時在兩堆取銅板，直到最後，
3  //將銅板拿光的人贏得此遊戲。
4  #define maxn 23+5
5  int SG[maxn];
6  int visited[1000+5];
7  int pile[maxn], ans;
8  void calculateSG(){
9      SG[0]=0;
10     for(int i=1;i<=maxn;i++){
11         int cur=0;
12         for(int j=0;j<i;j++){
13             for(int k=0;k<=j;k++){
14                 visited[SG[j]^SG[k]]=i;
15                 while(visited[cur]==i) cur++;
16                 SG[i]=cur;
17             }
18         }
19     }
20     int main(){
21         calculateSG();
22         int Case=0,n;
23         while(cin>>n,n){
24             ans=0;
25             for(int i=1;i<=n;i++) cin>>pile[i];
26             for(int i=1;i<=n;i++){
27                 if(pile[i]&1) ans^=SG[n-i];
28             }
29             cout<<"Game " <<Case<<" ";
30             if(!ans) cout<<"-1 -1 -1\\n";
31             else{
32                 bool flag=0;
33                 for(int i=1;i<=n;i++){
34                     if(pile[i]){
35                         for(int j=i+1;j<=n;j++){
36                             for(int k=j;k<=n;k++){
37                                 if((SG[n-i]^SG[n-j]^SG[n-k])==ans){
38                                     cout<<i-1<<" "<<j-1<<" "<<k-1<<endl;
39                                     flag=1;
40                                     break;
41                                 }
42                             }
43                         }
44                     }
45                     if(flag) break;
46                 }
47                 if(flag) break;
48             }
49             return 0;
50         }
51     }
52     /*
53     input
54     4 1 0 1 100
55     3 1 0 5
56     2 2 1
57     0
58     output
59     Game 1: 0 2 3
60     Game 2: 0 1 1
61     Game 3: -1 -1 -1
62     */

```



## 4.8 SCC Tarjan

```

1 //單純考 scc，每個 scc 中找成本最小的蓋，如果有多個一樣小的要
2 //注意以下程式有縮點，但沒存起來，存法就是開一個 array
3   -> ID[u] = SCCID
4 #define maxn 100005
5 #define MOD 1000000007
6 long long cost[maxn];
7 vector<vector<int>> G;
8 int SCC = 0;
9 stack<int> sk;
10 int dfn[maxn];
11 int low[maxn];
12 bool inStack[maxn];
13 int dfsTime = 1;
14 long long totalCost = 0;
15 long long ways = 1;
16 void dfs(int u) {
17     dfn[u] = low[u] = dfsTime;
18     ++dfsTime;
19     sk.push(u);
20     inStack[u] = true;
21     for (int v: G[u]) {
22         if (dfn[v] == 0) {
23             dfs(v);
24             low[u] = min(low[u], low[v]);
25         }
26         else if (inStack[v]) {
27             //屬於同個 scc 且是我的 back edge
28             low[u] = min(low[u], dfn[v]);
29         }
30     }
31     //如果是 scc
32     if (dfn[u] == low[u]) {
33         long long minCost = 0x3f3f3f3f;
34         int currWays = 0;
35         ++SCC;
36         while (1) {
37             int v = sk.top();
38             inStack[v] = 0;
39             sk.pop();
40             if (minCost > cost[v]) {
41                 minCost = cost[v];
42                 currWays = 1;
43             }
44             else if (minCost == cost[v]) {
45                 ++currWays;
46             }
47             if (v == u)
48                 break;
49         }
50         totalCost += minCost;
51         ways = (ways * currWays) % MOD;
52     }
53 }
54 int main() {
55     int n;
56     scanf("%d", &n);
57     for (int i = 1; i <= n; ++i)
58         scanf("%lld", &cost[i]);
59     G.assign(n + 5, vector<int>());
60     int m;
61     scanf("%d", &m);
62     int u, v;
63     for (int i = 0; i < m; ++i) {
64         scanf("%d %d", &u, &v);
65         G[u].emplace_back(v);
66     }
67     for (int i = 1; i <= n; ++i) {
68         if (dfn[i] == 0)
69             dfs(i);
70     }
71     printf("%lld %lld\n", totalCost, ways % MOD);
72     return 0;
73 }

```

## 4.9 ArticulationPoints Tarjan

```

1 vector<vector<int>> G;
2 int N, timer;
3 bool visited[105];
4 int dfn[105]; // 第一次 visit 的時間
5 int low[105];
6 // 最小能回到的父節點 (不能是自己的 parent) 的 visTime
7 int res;
8 //求割點數量
9 void tarjan(int u, int parent) {
10     int child = 0;
11     bool isCut = false;
12     visited[u] = true;
13     dfn[u] = low[u] = ++timer;
14     for (int v: G[u]) {
15         if (!visited[v]) {
16             ++child;
17             tarjan(v, u);
18             low[u] = min(low[u], low[v]);
19             if (parent != -1 && low[v] >= dfn[u])
20                 isCut = true;
21         }
22         else if (v != parent)
23             low[u] = min(low[u], dfn[v]);
24     }
25     //If u is root of DFS tree -> 有兩個以上的 children
26     if (parent == -1 && child >= 2)
27         isCut = true;
28     if (isCut) ++res;
29 }
30 int main() {
31     char input[105];
32     char* token;
33     while (scanf("%d", &N) != EOF && N) {
34         G.assign(105, vector<int>());
35         memset(visited, false, sizeof(visited));
36         memset(low, 0, sizeof(low));
37         memset(dfn, 0, sizeof(dfn));
38         timer = 0;
39         res = 0;
40         getchar(); // for \n
41         while (fgets(input, 105, stdin)) {
42             if (input[0] == '\0')
43                 break;
44             int size = strlen(input);
45             input[size - 1] = '\0';
46             --size;
47             token = strtok(input, " ");
48             int u = atoi(token);
49             int v;
50             while (token = strtok(NULL, " ")) {
51                 v = atoi(token);
52                 G[u].emplace_back(v);
53                 G[v].emplace_back(u);
54             }
55         }
56         tarjan(1, -1);
57         printf("%d\n", res);
58     }
59     return 0;
60 }

```

## 4.10 最小樹狀圖

```

1 定義
2 有向圖上的最小生成樹 (Directed Minimum Spanning Tree)
3 稱為最小樹形圖。
4 const int maxn = 60 + 10;
5 const int inf = 0x3f3f3f3f;
6 struct Edge {
7     int s, t, cap, cost;
8 }; // cap 為頻寬 (optional)
9 int n, m, c;

```

```

10 int inEdge[maxn], idx[maxn], pre[maxn], vis[maxn];
11 // 對於每個點，選擇對它入度最小的那條邊
12 // 找環，如果沒有則 return;
13 // 進行縮環並更新其他點到環的距離。
14 int dirMST(vector<Edge> edges, int low) {
15     int result = 0, root = 0, N = n;
16     while(true) {
17         memset(inEdge, 0x3f, sizeof(inEdge));
18         // 找所有點的 in edge 放進 inEdge
19         // optional: low 為最小 cap 限制
20         for(const Edge& e : edges) {
21             if(e.cap < low) continue;
22             if(e.s!=e.t && e.cost<inEdge[e.t]) {
23                 inEdge[e.t] = e.cost;
24                 pre[e.t] = e.s;
25             }
26         }
27         for(int i=0; i<N; i++) {
28             if(i!=root && inEdge[i]==inf)
29                 return -1; //除了root 還有點沒有in 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 && idx[cur]==-1) {
39                 if(cur == root) break;
40                 vis[cur] = i;
41                 cur = pre[cur];
42             }
43             if(cur!=root && idx[cur]==-1) {
44                 for(int j=pre[cur]; j!=cur; j=pre[j])
45                     idx[j] = seq;
46                 idx[cur] = seq++;
47             }
48         }
49         if(seq == 0) return result; // 沒有 cycle
50         for(int i=0; i<N; i++)
51             // 沒有被縮點的點
52             if(idx[i] == -1) idx[i] = seq++;
53         // 縮點並重新編號
54         for(Edge& e : edges) {
55             if(idx[e.s] != idx[e.t])
56                 e.cost -= inEdge[e.t];
57             e.s = idx[e.s];
58             e.t = idx[e.t];
59         }
60         N = seq;
61         root = idx[root];
62     }
63 }

```

=====

65 Tarjan 的DMST 演算法

66 Tarjan 提出了一種能夠在

67  $O(m+n\log n)$ 時間內解決最小樹形圖問題的演算法。

68 流程

69 Tarjan 的演算法分為收縮與伸展兩個過程。

70 接下來先介紹收縮的過程。

71 我們要假設輸入的圖是滿足強連通的，

72 如果不滿足那就加入  $O(n)$  條邊使其滿足，

73 並且這些邊的邊權是無窮大的。

74 我們需要一個堆存儲結點的入邊編號，入邊權值，

75 結點總代價等相關信息，由於後續過程中會有堆的合併操作，

76 這裡採用左偏樹 與並查集實現。

77 演算法的每一步都選擇一個任意結點  $v$ ，

78 需要保證  $v$  不是根節點，並且在堆中沒有它的入邊。

79 再將  $v$  的最小入邊加入到堆中，

80 如果新加入的這條邊使堆中的邊形成了環，

81 那麼將構成環的那些結點收縮，

82 我們不妨將這些已經收縮的結點命名為超級結點，

83 再繼續這個過程，如果所有的頂點都縮成了超級結點，

```

84 那麼收縮過程就結束了。
85 整個收縮過程結束後會得到一棵收縮樹，
86 之後就會對它進行伸展操作。
87 堆中的邊總是會形成一條路徑  $v_0 \leftarrow v_1 \leftarrow \dots \leftarrow v_k$ ，
88 由於圖是強連通的，這個路徑必然存在，
89 並且其中的  $v_i$  可能是最初的單一結點，
90 也可能是壓縮後的超級結點。
91 最初有  $v_0=a$ ，其中  $a$  是圖中任意的一個結點，
92 每次都選擇一條最小入邊  $v_k \leftarrow u$ ，
93 如果  $u$  不是  $v_0, v_1, \dots, v_k$  中的一個結點，
94 那麼就將結點擴展到  $v_{k+1}=u$ 。
95 如果  $u$  是他們其中的一個結點  $v_i$ ，
96 那麼就找到了一個關於  $v_i \leftarrow \dots \leftarrow v_k \leftarrow v_i$  的環，
97 再將他們收縮為一個超級結點  $c$ 。
98 向隊列  $P$  中放入所有的結點或超級結點，
99 並初始選擇任一節點  $a$ ，只要佇列不為空，就進行以下步驟：
100 選擇  $a$  的最小入邊，保證不存在自環，
101 並找到另一頭的結點  $b$ 。
102 如果結點  $b$  沒有被記錄過說明未形成環，
103 令  $a \leftarrow b$ ，繼續目前操作尋找環。
104 如果  $b$  被記錄過了，就表示出現了環。
105 總結點數加一，並將環上的所有結點重新編號，對堆進行合併，
106 以及結點/超級結點的總權值的更新。
107 更新權值操作就是將環上所有結點的入邊都收集起來，
108 並減去環上入邊的邊權。
109 typedef long long ll;
110 #define maxn 102
111 #define INF 0x3f3f3f3f
112 struct UnionFind {
113     int fa[maxn << 1];
114     UnionFind() { memset(fa, 0, sizeof(fa)); }
115     void clear(int n) {
116         memset(fa + 1, 0, sizeof(int) * n);
117     }
118     int find(int x) {
119         return fa[x] ? fa[x] = find(fa[x]) : x;
120     }
121     int operator[](int x) { return find(x); }
122 };
123 struct Edge {
124     int u, v, w, w0;
125 };
126 struct Heap {
127     Edge *e;
128     int rk, constant;
129     Heap *lch, *rch;
130     Heap(Edge *_e):
131         e(_e), rk(1), constant(0), lch(NULL), rch(NULL) {}
132     void push() {
133         if (lch) lch->constant += constant;
134         if (rch) rch->constant += constant;
135         e->w += constant;
136         constant = 0;
137     }
138 };
139 Heap *merge(Heap *x, Heap *y) {
140     if (!x) return y;
141     if (!y) return x;
142     if (x->e->w + x->constant > y->e->w + y->constant)
143         swap(x, y);
144     x->push();
145     x->rch = merge(x->rch, y);
146     if (!x->lch || x->lch->rk < x->rch->rk)
147         swap(x->lch, x->rch);
148     if (x->rch)
149         x->rk = x->rch->rk + 1;
150     else
151         x->rk = 1;
152     return x;
153 }
154 Edge *extract(Heap *x) {
155     Edge *r = x->e;
156     x->push();
157     x = merge(x->lch, x->rch);
158     return r;

```



```

159 }
160 vector<Edge> in[maxn];
161 int n, m, fa[maxn << 1], nxt[maxn << 1];
162 Edge *ed[maxn << 1];
163 Heap *Q[maxn << 1];
164 UnionFind id;
165 void contract() {
166     bool mark[maxn << 1];
167     //將圖上的每一個節點與其相連的那些節點進行記錄
168     for (int i = 1; i <= n; i++) {
169         queue<Heap *> q;
170         for (int j = 0; j < in[i].size(); j++)
171             q.push(new Heap(&in[i][j]));
172         while (q.size() > 1) {
173             Heap *u = q.front();
174             q.pop();
175             Heap *v = q.front();
176             q.pop();
177             q.push(merge(u, v));
178         }
179         Q[i] = q.front();
180     }
181     mark[1] = true;
182     for (int a=1, b=1, p; Q[a]; b=a, mark[b]=true) {
183         //尋找最小入邊以及其端點，保證無環
184         do {
185             ed[a] = extract(Q[a]);
186             a = id[ed[a]->u];
187         } while (a == b && Q[a]);
188         if (a == b) break;
189         if (!mark[a]) continue;
190         //對發現的環進行收縮，以及環內的節點重新編號，
191         //總權值更新
192         for (a = b, n++; a != n; a = p) {
193             id.fa[a] = fa[a] = n;
194             if (Q[a]) Q[a]->constant -= ed[a]->w;
195             Q[n] = merge(Q[n], Q[a]);
196             p = id[ed[a]->u];
197             nxt[p == n ? b : p] = a;
198         }
199     }
200 }
201 ll expand(int x, int r);
202 ll expand_iter(int x) {
203     ll r = 0;
204     for (int u=nxt[x]; u!=x; u=nxt[u]) {
205         if (ed[u]->w0 >= INF)
206             return INF;
207         else
208             r+=expand(ed[u]->v, u)+ed[u]->w0;
209     }
210     return r;
211 }
212 ll expand(int x, int t) {
213     ll r = 0;
214     for (; x != t; x = fa[x]) {
215         r += expand_iter(x);
216         if (r >= INF) return INF;
217     }
218     return r;
219 }
220 void link(int u, int v, int w) {
221     in[v].push_back({u, v, w, w});
222 }
223 int main() {
224     int rt;
225     scanf("%d %d %d", &n, &m, &rt);
226     for (int i = 0; i < m; i++) {
227         int u, v, w;
228         scanf("%d %d %d", &u, &v, &w);
229         link(u, v, w);
230     }
231     //保證強連通
232     for (int i = 1; i <= n; i++)
233         link(i > 1 ? i - 1 : n, i, INF);
234     contract();
235     ll ans = expand(rt, n);

```

```

236     if (ans >= INF)
237         puts("-1");
238     else
239         printf("%lld\n", ans);
240     return 0;
241 }

```

## 4.11 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) const {
14         return this->w > other.w;
15     }
16 };
17 struct Item_astar {
18     int u, g, f;
19     bool operator <(const Item_astar& other) 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];
29 //用反向圖去求出每一點到終點的最短距離，並以此當作 $h(u)$ 
30 void dijkstra(int s, int t) {
31     memset(visited, 0, sizeof(visited));
32     priority_queue<Item_pqH> pq;
33     pq.push({s, 0});
34     h[s] = 0;
35     while (!pq.empty()) {
36         Item_pqH curr = pq.top();
37         pq.pop();
38         visited[curr.u] = true;
39         for (Edge& edge: invertG[curr.u]) {
40             if (!visited[edge.v]) {
41                 if (h[edge.v] > h[curr.u] + edge.w) {
42                     h[edge.v] = h[curr.u] + edge.w;
43                     pq.push({edge.v, h[edge.v]});
44                 }
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 + edge.w,
63                     curr.g + edge.w + h[edge.v]});
64             }
65         }
66     }

```

```

66     return -1;
67 }
68 int main() {
69     int n, m;
70     while (scanf("%d %d", &n, &m) && (n != 0 && m != 0)) {
71         G.assign(n + 5, vector<Edge>());
72         invertG.assign(n + 5, vector<Edge>());
73         int s, t, k;
74         scanf("%d %d %d", &s, &t, &k);
75         int u, v, w;
76         for (int i = 0; i < m; ++i) {
77             scanf("%d %d %d", &u, &v, &w);
78             G[u].emplace_back(Edge{u, v, w});
79             invertG[v].emplace_back(Edge{v, u, w});
80         }
81         memset(h, 0x3f, sizeof(h));
82         dijkstra(t, s);
83         printf("%d\n", Astar(s, t, k));
84     }
85     return 0;
86 }

```

## 4.12 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 int getWinner(int n, int k) {
5     int winner = 0;
6     for (int i = 1; i <= n; ++i)
7         winner = (winner + k) % i;
8     return winner;
9 }
10 int main() {
11     int n;
12     while (scanf("%d", &n) != EOF && n){
13         --n;
14         for (int k = 1; k <= n; ++k){
15             if (getWinner(n, k) == 11){
16                 printf("%d\n", k);
17                 break;
18             }
19         }
20     }
21     return 0;
22 }

```

## 4.13 KM

```

1 /*題意: 給定一個W矩陣, 現在分成row、column兩個1維陣列
2 W[i][j]=k即代表column[i] + row[j]要>=k
3 求row[] 與 column[]的所有值在滿足矩陣W的要求之下
4 row[] + column[]所有元素相加起來要最小
5 利用KM求二分圖最大權匹配
6 Lx -> vertex labeling of X
7 Ly -> vertex labeling of y
8 一開始Lx[i] = max(W[i][j]), Ly = 0
9 Lx[i] + Ly[j] >= W[i][j]
10 要最小化全部的(Lx[i] + Ly[j])加總
11 不斷的調整vertex
12     labeling去找到一條交錯邊皆滿足Lx[i] + Ly[j]
13     == W[i][j]的增廣路
14 最後會得到正確的二分圖完美匹配中的最大權分配(先滿足最多
15 意義是將最大化所有匹配邊權重和的問題改成最小化所有點的
16 #define maxn 505
17 int W[maxn][maxn];
18 int Lx[maxn], Ly[maxn];
19 bool S[maxn], T[maxn];
20 //L[i] = j -> S_i配給T_j, -1 for 還沒匹配
21 int L[maxn];
22 int n;

```

```

21 bool match(int i) {
22     S[i] = true;
23     for (int j = 0; j < n; ++j) {
24         // KM重點
25         // Lx + Ly >= selected_edge(x, y)
26         // 要想辦法降低Lx + Ly
27         // 所以選Lx + Ly == selected_edge(x, y)
28         if (Lx[i] + Ly[j] == W[i][j] && !T[j]) {
29             T[j] = true;
30             if ((L[j] == -1) || match(L[j])) {
31                 L[j] = i;
32                 return true;
33             }
34         }
35     }
36     return false;
37 }
38 // 修改二分圖上的交錯路徑上點的權重
39 // 此舉是在通過調整vertex
40 // labeling看看能不能產生出新的增廣路(KM的增廣路要求Lx[i]
41 // + Ly[j] == W[i][j])
42 // 在這裡優先從最小的diff調調看, 才能保證最大權重匹配
43 void update()
44 {
45     int diff = 0x3f3f3f3f;
46     for (int i = 0; i < n; ++i) {
47         if (S[i]) {
48             for (int j = 0; j < n; ++j) {
49                 if (!T[j])
50                     diff = min(diff, Lx[i] + Ly[j] - W[i][j]);
51             }
52         }
53         for (int i = 0; i < n; ++i) {
54             if (S[i]) Lx[i] -= diff;
55             if (T[i]) Ly[i] += diff;
56         }
57     }
58 void KM()
59 {
60     for (int i = 0; i < n; ++i) {
61         L[i] = -1;
62         Lx[i] = Ly[i] = 0;
63         for (int j = 0; j < n; ++j)
64             Lx[i] = max(Lx[i], W[i][j]);
65     }
66     for (int i = 0; i < n; ++i) {
67         while(1) {
68             memset(S, false, sizeof(S));
69             memset(T, false, sizeof(T));
70             if (match(i))
71                 break;
72             else
73                 update(); //去調整vertex
74                             //labeling以增加增廣路徑
75         }
76     }
77 }
78 int main() {
79     while (scanf("%d", &n) != EOF) {
80         for (int i = 0; i < n; ++i)
81             for (int j = 0; j < n; ++j)
82                 scanf("%d", &W[i][j]);
83         KM();
84         int res = 0;
85         for (int i = 0; i < n; ++i) {
86             if (i != 0)
87                 printf(" ");
88             printf("%d", Lx[i]);
89             res += Lx[i];
90         }
91         puts("");
92         for (int i = 0; i < n; ++i) {
93             if (i != 0)
94                 printf(" ");
95             printf("%d", Lx[i]);
96             res += Lx[i];
97         }
98         puts("");
99     }
100 }

```

```

93         printf(" %d", Ly[i]);
94     else
95         printf("%d", Ly[i]);
96     res += Ly[i];
97 }
98 puts("");
99 printf("%d\n", res);
100 }
101 return 0;
102 }

```

#### 4.14 LCA 倍增法

```

1 //倍增法預處理  $O(n \log n)$ ，查詢  $O(\log n)$ ，利用 lca 找樹上任兩點距離
2 #define maxn 100005
3 struct Edge {
4     int u, v, w;
5 };
6 vector<vector<Edge>> G; // tree
7 int fa[maxn][31]; // fa[u][i] -> u 的第  $2^i$  個祖先
8 long long dis[maxn][31];
9 int dep[maxn]; // 深度
10 void dfs(int u, int p) { // 預處理 fa
11     fa[u][0] = p; // 因為 u 的第  $2^0 = 1$  的祖先就是 p
12     dep[u] = dep[p] + 1;
13     // 第  $2^i$  的祖先是 (第  $2^{i-1}$  個祖先) 的第  $2^{i-1}$  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] + dis[u][i-1];
18     }
19     // 遍歷子節點
20     for (Edge& edge: G[u]) {
21         if (edge.v == p) continue;
22         dis[edge.v][0] = edge.w;
23         dfs(edge.v, u);
24     }
25 }
26 }
27 long long lca(int x, int y)
28 { // 此函數是找 lca 同時計算 x、y 的距離 -> dis(x, 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, deltaDep >>= 1)
35         if (deltaDep & 1) res += dis[y][i], y = fa[y][i];
36     if (y == x) // x = y -> x、y 彼此是彼此的祖先
37         return res;
38     // 往上找，一起跳，但 x、y 不能重疊
39     for (int i = 30; i >= 0 && y != x; --i) {
40         if (fa[x][i] != fa[y][i]) {
41             res += dis[x][i] + dis[y][i];
42             x = fa[x][i];
43             y = fa[y][i];
44         }
45     }
46     // 最後發現不能跳了，此時 x 的第  $2^0 = 1$  個祖先 (或說 y 的第  $2^0 = 1$  的祖先) 即為 x、y 的 lca
47     res += dis[x][0] + dis[y][0];
48     return res;
49 }
50 }
51 int main() {
52     int n, q;
53     while (~scanf("%d", &n) && n) {
54         int v, w;
55         G.assign(n + 5, vector<Edge>());

```

```

56         for (int i = 1; i <= n - 1; ++i) {
57             scanf("%d %d", &v, &w);
58             G[i + 1].push_back({i + 1, v + 1, w});
59             G[v + 1].push_back({v + 1, i + 1, w});
60         }
61         dfs(1, 0);
62         scanf("%d", &q);
63         int u;
64         while (q--) {
65             scanf("%d %d", &u, &v);
66             printf("%lld%c", lca(u + 1, v + 1), (q) ? ' ' : '\n');
67         }
68     }
69     return 0;
70 }

```

#### 4.15 LCA 樹壓平 RMQ

```

1 //樹壓平求 LCA RMQ (sparse table)
2 //  $O(n \log n)$  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 overflow
5 #define maxn 100005
6 struct Edge {
7     int u, v, w;
8 };
9 int dep[maxn], pos[maxn];
10 long long dis[maxn];
11 int st[maxn * 2][32]; // sparse table
12 int realLCA[maxn * 2][32];
13 // 最小深度對應的節點，及真正的 LCA
14 int Log[maxn]; // 取代 std::log2
15 int tp; // timestamp
16 vector<vector<Edge>> G; // tree
17 void calLog() {
18     Log[1] = 0;
19     Log[2] = 1;
20     for (int i = 3; i < maxn; ++i)
21         Log[i] = Log[i / 2] + 1;
22 }
23 void buildST() {
24     for (int j = 0; Log[tp]; ++j) {
25         for (int i = 0; i + (1 << j) - 1 < tp; ++i) {
26             if (st[i - 1][j] < st[i - 1][j + (1 << 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 - 1)];
32                 realLCA[i][j] = realLCA[i - 1][j + (1 << i - 1)];
33             }
34         }
35     }
36 } //  $O(n \log n)$ 
37 int query(int l, int r) { // [l, r] min
38     // depth 即為 lca 的深度
39     int k = Log[r - l + 1];
40     if (st[l][k] < st[r - (1 << k) + 1][k])
41         return realLCA[l][k];
42     else
43         return realLCA[r - (1 << k) + 1][k];
44 }
45 void dfs(int u, int p) { // euler tour
46     pos[u] = tp;
47     st[tp][0] = dep[u];
48     realLCA[tp][0] = dep[u];
49     ++tp;
50     for (int i = 0; i < G[u].size(); ++i) {
51         Edge& edge = G[u][i];
52         if (edge.v == p) continue;
53         dep[edge.v] = dep[u] + 1;

```

```

51     dis[edge.v] = dis[edge.u] + edge.w;
52     dfs(edge.v, u);
53     st[tp++][0] = dep[u];
54 }
55 }
56 long long getDis(int u, int v) {
57     if (pos[u] > pos[v])
58         swap(u, v);
59     int lca = query(pos[u], pos[v]);
60     return dis[u] + dis[v] - 2 * dis[query(pos[u],
61         pos[v])];
62 }
63 int main() {
64     int n, q;
65     callog();
66     while (~scanf("%d", &n) && n) {
67         int v, w;
68         G.assign(n + 5, vector<Edge>());
69         tp = 0;
70         for (int i = 1; i <= n - 1; ++i) {
71             scanf("%d %d", &v, &w);
72             G[i].push_back({i, v, w});
73             G[v].push_back({v, i, w});
74         }
75         dfs(0, -1);
76         buildST();
77         scanf("%d", &q);
78         int u;
79         while (q--) {
80             scanf("%d %d", &u, &v);
81             printf("%lld%c", getDis(u, v), (q) ? ' '
82                 : '\n');
83         }
84     }
85     return 0;
86 }

```

#### 4.16 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 //SPFA用
11 bool inqueue[maxn];
12 //SPFA用的dis[]
13 long long dis[maxn];
14 //maxFlow一路扣回去時要知道parent
15 //<注> 在這題因為G[]中存的是edgeIndex in edges[]
16 //
17 //    所以parent存的也是對應edges[]中的edgeIndex(主要是方便)
18 int parent[maxn];
19 //maxFlow時需要紀錄到node u時的bottleneck
20 //同時也代表著u該次流出去的量
21 long long outFlow[maxn];
22 void addEdge(int u, int v, int cap, int cost) {
23     edges.emplace_back(Edge{u, v, cap, 0, cost});
24     edges.emplace_back(Edge{v, u, 0, 0, -cost});
25     m = edges.size();
26     G[u].emplace_back(m - 2);
27     G[v].emplace_back(m - 1);
28 }
29 //一邊求最短路的同時一邊MaxFlow
30 bool SPFA(long long& maxFlow, long long& minCost) {
31     //memset(outFlow, 0x3f, sizeof(outFlow));
32     memset(dis, 0x3f, sizeof(dis));
33     memset(inqueue, false, sizeof(inqueue));
34     queue<int> q;
35     q.push(s);
36     dis[s] = 0;
37     inqueue[s] = true;
38     outFlow[s] = INF;
39     while (!q.empty()) {
40         int u = q.front();
41         q.pop();
42         inqueue[u] = false;
43         for (const int edgeIndex: G[u]) {
44             const Edge& edge = edges[edgeIndex];
45             if ((edge.cap > edge.flow) &&
46                 (dis[edge.v] > dis[u] + edge.cost)) {
47                 dis[edge.v] = dis[u] + edge.cost;
48                 parent[edge.v] = edgeIndex;
49                 outFlow[edge.v] = min(outFlow[u],
50                     (long long)(edge.cap -
51                         edge.flow));
52                 if (!inqueue[edge.v]) {
53                     q.push(edge.v);
54                     inqueue[edge.v] = true;
55                 }
56             }
57         }
58     }
59     //如果dis[t] > 0代表根本不賺還倒賠
60     if (dis[t] > 0)
61         return false;
62     maxFlow += outFlow[t];
63     minCost += dis[t] * outFlow[t];
64     //一路更新回去這次最短路流完後要維護的MaxFlow演算法相關(如反)
65     int curr = t;
66     while (curr != s) {
67         edges[parent[curr]].flow += outFlow[t];
68         edges[parent[curr] ^ 1].flow -= outFlow[t];
69         curr = edges[parent[curr]].u;
70     }
71     return true;
72 }
73 long long MCMF() {
74     long long maxFlow = 0;
75     long long minCost = 0;
76     while (SPFA(maxFlow, minCost))
77         ;
78     return minCost;
79 }
80 int main() {
81     int T;
82     scanf("%d", &T);
83     for (int Case = 1; Case <= T; ++Case) {
84         //總共幾個月，囤貨成本
85         int M, I;
86         scanf("%d %d", &M, &I);
87         //node size
88         n = M + M + 2;
89         G.assign(n + 5, vector<int>());
90         edges.clear();
91         s = 0;
92         t = M + M + 1;
93         for (int i = 1; i <= M; ++i) {
94             int produceCost, produceMax, sellPrice,
95                 sellMax, inventoryMonth;
96             scanf("%d %d %d %d %d", &produceCost,
97                 &produceMax, &sellPrice, &sellMax,
98                 &inventoryMonth);
99             addEdge(s, i, produceMax, produceCost);
100             addEdge(M + i, t, sellMax, -sellPrice);
101             for (int j = 0; j <= inventoryMonth; ++j) {
102                 if (i + j <= M)
103                     addEdge(i, M + i + j, INF, I * j);
104             }
105         }
106         printf("Case %d: %lld\n", Case, -MCMF());
107     }
108     return 0;
109 }

```

## 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 Dancing Links

```

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

```

```

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

```

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

```

### 5.2 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 val)
6         : l(l), r(r), val(val){}
7     bool operator < (const Node& other) const{
8         return this->l < other.l;
9     }

```

```

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, 0, 0));
17     //運氣很好直接找到左端點是pos的區間
18     if (it != chthollyTree.end() && it->l == pos)
19         return it;
20     //到這邊代表找到的是第一個左端點大於pos的區間
21     //it -
22     //即可找到左端點等於pos的區間(不會是別的，因為沒有重疊的)
23     --it;
24     long long l = it->l, r = it->r;
25     long long val = it->val;
26     chthollyTree.erase(it);
27     chthollyTree.insert(Node(l, pos - 1, val));
28     //回傳左端點是pos的區間iterator
29     return chthollyTree.insert(Node(pos, r, val)).first;
30 }
31 //區間賦值
32 void assign(long long l, long long r, long 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begin
35     //因為end可以在原本begin的區間中
36     set<Node>::iterator end = split(r + 1), begin =
37         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 val) {
45     set<Node>::iterator end = split(r + 1);
46     set<Node>::iterator begin = split(l);
47     for (set<Node>::iterator it = begin; it != end;
48         ++it)
49         it->val += val;
50 }
51 //查詢區間第k小 -> 直接把每個區間丟去vector排序
52 long long getKthSmallest(long long l, long long r,
53     long long k) {
54     set<Node>::iterator end = split(r + 1);
55     set<Node>::iterator begin = split(l);
56     //pair -> first: val, second: 區間長度
57     vector<pair<long long, long long>> vec;
58     for (set<Node>::iterator it = begin; it != end;
59         ++it) {
60         vec.push_back({it->val, it->r - it->l + 1});
61     }
62     sort(vec.begin(), vec.end());
63     for (const pair<long long, long long>& p: vec) {
64         k -= p.second;
65         if (k <= 0)
66             return p.first;
67     }
68     //不應該跑到這
69     return -1;
70 }
71 //快速幂
72 long long qpow(long long x, long long n, long long
73     mod) {
74     long long res = 1;
75     x %= mod;
76     while (n)
77     {
78         if (n & 1)
79             res = res * x % mod;
80         n >>= 1;
81         x = x * x % mod;
82     }
83     return res;

```



```

76 }
77 //區間n次方和
78 long long sumOfPow(long long l, long long r, long
    long n, long long mod) {
79     long long total = 0;
80     set<Node>::iterator end = split(r + 1);
81     set<Node>::iterator begin = split(l);
82     for (set<Node>::iterator it = begin; it != end;
        ++it)
83     {
84         total = (total + qpow(it->val, n, mod) *
            (it->r - it->l + 1)) % mod;
85     }
86     return total;
87 }

```

### 5.3 線段樹 1D

```

1 #define MAXN 1000
2 int data[MAXN]; //原數據
3 int st[4 * MAXN]; //線段樹
4 int tag[4 * MAXN]; //懶標
5 inline int pull(int l, int r) {
6     // 隨題目改變sum、max、min
7     // l、r是左右樹的index
8     return st[l] + st[r];
9 }
10 void build(int l, int r, int i) {
11     // 在[l, r]區間建樹，目前根的index為i
12     if (l == r) {
13         st[i] = data[l];
14         return;
15     }
16     int mid = l + ((r - l) >> 1);
17     build(l, mid, i * 2);
18     build(mid + 1, r, i * 2 + 1);
19     st[i] = pull(i * 2, i * 2 + 1);
20 }
21 int query(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]; //下傳懶標至左節點
31         tag[i * 2 + 1] += tag[i]; //下傳懶標至右節點
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(int ql, int qr, int l, int r, int i, int c) {
42     // [ql, qr]是查詢區間，[l, r]是當前節點包含的區間
43     // c是變化量
44     if (ql <= l && r <= qr) {
45         st[i] += (r - l + 1) * c;
46         //求和，此需乘上區間長度
47         tag[i] += c;
48         return;
49     }
50     int mid = l + ((r - l) >> 1);
51     if (tag[i] && l != r) {
52         //如果當前懶標有值則更新左右節點
53         st[i * 2] += tag[i] * (mid - l + 1);
54         st[i * 2 + 1] += tag[i] * (r - mid);
55         tag[i * 2] += tag[i]; //下傳懶標至左節點
56         tag[i * 2 + 1] += tag[i]; //下傳懶標至右節點

```

```

56     tag[i] = 0;
57 }
58 if (ql <= mid) update(ql, qr, l, mid, i * 2, c);
59 if (qr > mid) update(ql, qr, mid + 1, r, i * 2 + 1, c);
60 st[i] = pull(i * 2, i * 2 + 1);
61 }
62 //如果是直接改值而不是加值，query與update中的tag與st的
63 //改值從+=改成=

```

### 5.4 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 val, int
    yPos, int xIndex, bool xIsLeaf) {
6     if (l == r) {
7         if (xIsLeaf) {
8             maxST[xIndex][index] =
                minST[xIndex][index] = val;
9             return;
10        }
11        maxST[xIndex][index] = max(maxST[xIndex *
            2][index], maxST[xIndex * 2 + 1][index]);
12        minST[xIndex][index] = min(minST[xIndex *
            2][index], minST[xIndex * 2 + 1][index]);
13    }
14    else {
15        int mid = (l + r) / 2;
16        if (yPos <= mid)
17            modifyY(index * 2, l, mid, val, yPos,
                xIndex, xIsLeaf);
18        else
19            modifyY(index * 2 + 1, mid + 1, r, val,
                yPos, xIndex, xIsLeaf);
20
21        maxST[xIndex][index] =
            max(maxST[xIndex][index * 2],
                maxST[xIndex][index * 2 + 1]);
22        minST[xIndex][index] =
            min(minST[xIndex][index * 2],
                minST[xIndex][index * 2 + 1]);
23    }
24 }
25 void modifyX(int index, int l, int r, int val, int
    xPos, int yPos) {
26     if (l == r) {
27         modifyY(1, 1, N, val, yPos, index, true);
28     }
29     else {
30         int mid = (l + r) / 2;
31         if (xPos <= mid)
32             modifyX(index * 2, l, mid, val, xPos,
                yPos);
33         else
34             modifyX(index * 2 + 1, mid + 1, r, val,
                xPos, yPos);
35         modifyY(1, 1, N, val, yPos, index, false);
36     }
37 }
38 void queryY(int index, int l, int r, int yql, int
    yqr, int xIndex, int& vmax, int& vmin) {
39     if (yql <= l && r <= yqr) {
40         vmax = max(vmax, maxST[xIndex][index]);
41         vmin = min(vmin, minST[xIndex][index]);
42     }
43     else
44     {
45         int mid = (l + r) / 2;
46         if (yql <= mid)
47             queryY(index * 2, l, mid, yql, yqr,
                xIndex, vmax, vmin);
48         if (mid < yqr)

```

```

49     queryY(index * 2 + 1, mid + 1, r, yql,
        yqr, xIndex, vmax, vmin);
50 }
51 }
52 void queryX(int index, int l, int r, int xql, int
    xqr, int yql, int yqr, int& vmax, int& vmin) {
53     if (xql <= l && r <= xqr) {
54         queryY(1, 1, N, yql, yqr, index, vmax, vmin);
55     }
56     else {
57         int mid = (l + r) / 2;
58         if (xql <= mid)
59             queryX(index * 2, l, mid, xql, xqr, yql,
                yqr, vmax, vmin);
60         if (mid < xqr)
61             queryX(index * 2 + 1, mid + 1, r, xql,
                xqr, yql, yqr, vmax, vmin);
62     }
63 }
64 int main() {
65     while (scanf("%d", &N) != EOF) {
66         int val;
67         for (int i = 1; i <= N; ++i) {
68             for (int j = 1; j <= N; ++j) {
69                 scanf("%d", &val);
70                 modifyX(1, 1, N, val, i, j);
71             }
72         }
73         int q;
74         int vmax, vmin;
75         int xql, xqr, yql, yqr;
76         char op;
77         scanf("%d", &q);
78         while (q--) {
79             getchar(); //for \n
80             scanf("%c", &op);
81             if (op == 'q') {
82                 scanf("%d %d %d %d", &xql, &yql,
                    &xqr, &yqr);
83                 vmax = -0x3f3f3f3f;
84                 vmin = 0x3f3f3f3f;
85                 queryX(1, 1, N, xql, xqr, yql, yqr,
                    vmax, vmin);
86                 printf("%d %d\n", vmax, vmin);
87             }
88             else {
89                 scanf("%d %d %d", &xql, &yql, &val);
90                 modifyX(1, 1, N, val, xql, yql);
91             }
92         }
93     }
94     return 0;
95 }

```

## 5.5 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     if (l == r)
10     {
11         ++st[index];
12         return;
13     }
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 + 1];
21 }
22 //找區間第k個小的
23 int query(int index, int l, int r, int k) {
24     if (l == r)
25         return id[l];
26     int mid = (l + r) / 2;
27     //k比左子樹小
28     if (k <= st[index * 2])
29         return query(index * 2, l, mid, k);
30     else
31         return query(index * 2 + 1, mid + 1, r, k -
            st[index * 2]);
32 }
33 int main() {
34     int t;
35     cin >> t;
36     bool first = true;
37     while (t--) {
38         if (first)
39             first = false;
40         else
41             puts("");
42         memset(st, 0, sizeof(st));
43         int m, n;
44         cin >> m >> n;
45         for (int i = 1; i <= m; ++i) {
46             cin >> nums[i];
47             id[i] = nums[i];
48         }
49         for (int i = 0; i < n; ++i)
50             cin >> getArr[i];
51         //離散化
52         //防止m == 0
53         if (m)
54             sort(id + 1, id + m + 1);
55         int stSize = unique(id + 1, id + m + 1) - (id
            + 1);
56         for (int i = 1; i <= m; ++i) {
57             nums[i] = lower_bound(id + 1, id + stSize
                + 1, nums[i]) - id;
58         }
59         int addCount = 0;
60         int getCount = 0;
61         int k = 1;
62         while (getCount < n) {
63             if (getArr[getCount] == addCount) {
64                 printf("%d\n", query(1, 1, stSize,
                    k));
65                 ++k;
66                 ++getCount;
67             }
68             else {
69                 update(1, 1, stSize, nums[addCount +
                    1]);
70                 ++addCount;
71             }
72         }
73     }
74     return 0;
75 }

```

## 5.6 Trie

```

1 const int maxn = 300000 + 10;
2 const int mod = 20071027;
3 int dp[maxn];
4 int mp[4000*100 + 10][26];
5 char str[maxn];
6 struct Trie {
7     int seq;
8     int val[maxn];
9     Trie() {
10         seq = 0;
11         memset(val, 0, sizeof(val));

```

```

12     memset(mp, 0, sizeof(mp));
13 }
14 void insert(char* s, int len) {
15     int r = 0;
16     for(int i=0; i<len; i++) {
17         int c = s[i] - 'a';
18         if(!mp[r][c]) mp[r][c] = ++seq;
19         r = mp[r][c];
20     }
21     val[r] = len;
22     return;
23 }
24 int find(int idx, int len) {
25     int result = 0;
26     for(int r=0; idx<len; idx++) {
27         int c = str[idx] - 'a';
28         if(!(r = mp[r][c])) return result;
29         if(val[r])
30             result = (result + dp[idx + 1]) % mod;
31     }
32     return result;
33 }
34 };
35 int main() {
36     int n, tc = 1;
37     while(~scanf("%s%d", str, &n)) {
38         Trie tr;
39         int len = strlen(str);
40         char word[100+10];
41         memset(dp, 0, sizeof(dp));
42         dp[len] = 1;
43         while(n--) {
44             scanf("%s", word);
45             tr.insert(word, strlen(word));
46         }
47         for(int i=len-1; i>=0; i--)
48             dp[i] = tr.find(i, len);
49         printf("Case %d: %d\n", tc++, dp[0]);
50     }
51     return 0;
52 }
53 /****Input****
54 * abcd
55 * 4
56 * a b cd ab
57 * ****
58 * ****Output***
59 * Case 1: 2
60 * ****

```

## 5.7 單調隊列

```

1 //單調隊列
2 "如果一個選手比你小還比你強，你就可以退役了。"--單調隊列
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
15 void getmin() {
16     // 得到這個隊列裡的最小值，直接找到最後的就行了
17     int head=0, tail=0;
18     for(int i=1; i<k; i++) {
19         while(head<=tail&&a[q[tail]]>=a[i]) tail--;
20         q[++tail]=i;
21     }
22     for(int i=k; i<=n; i++) {
23         while(head<=tail&&a[q[tail]]>=a[i]) tail--;

```

```

24         q[++tail]=i;
25         while(q[head]<=i-k) head++;
26         cout<<a[q[head]]<<" ";
27     }
28     cout<<endl;
29 }
30
31 void getmax() { // 和上面同理
32     int head=0, tail=0;
33     for(int i=1; i<k; i++) {
34         while(head<=tail&&a[q[tail]]<=a[i]) tail--;
35         q[++tail]=i;
36     }
37     for(int i=k; i<=n; i++) {
38         while(head<=tail&&a[q[tail]]<=a[i]) tail--;
39         q[++tail]=i;
40         while(q[head]<=i-k) head++;
41         cout<<a[q[head]]<<" ";
42     }
43     cout<<endl;
44 }
45
46 int main(){
47     cin>>n>>k; //每k個連續的數
48     for(int i=1; i<=n; i++) cin>>a[i];
49     getmin();
50     getmax();
51     return 0;
52 }

```

## 6 geometry

### 6.1 intersection

```

1 using LL = long long;
2
3 struct Point2D {
4     LL x, y;
5 };
6
7 struct Line2D {
8     Point2D s, e;
9     LL a, b, c; // L: ax + by = c
10     Line2D(Point2D s, Point2D e): s(s), e(e) {
11         a = e.y - s.y;
12         b = s.x - e.x;
13         c = a * s.x + b * s.y;
14     }
15 };
16
17 // 用克拉馬公式求二元一次解
18 Point2D intersection2D(Line2D l1, Line2D l2) {
19     LL D = l1.a * l2.b - l2.a * l1.b;
20     LL Dx = l1.c * l2.b - l2.c * l1.b;
21     LL Dy = l1.a * l2.c - l2.a * l1.c;
22
23     if(D) { // intersection
24         double x = 1.0 * Dx / D;
25         double y = 1.0 * Dy / D;
26     } else {
27         if(Dx || Dy) // Parallel lines
28             else // Same line
29     }
30 }

```

### 6.2 半平面相交

```

1 // Q: 給定一張凸包(已排序的點)，
2 // 找出圖中離凸包外最遠的距離
3
4 const int maxn = 100 + 10;
5 const double eps = 1e-7;

```

```

6
7 struct Vector {
8     double x, y;
9     Vector(double x=0.0, double y=0.0): x(x), y(y) {}
10
11     Vector operator+(Vector v) {
12         return Vector(x+v.x, y+v.y);
13     }
14     Vector operator-(Vector v) {
15         return Vector(x-v.x, y-v.y);
16     }
17     Vector operator*(double val) {
18         return Vector(x*val, y*val);
19     }
20     double dot(Vector v) { return x*v.x + y*v.y; }
21     double cross(Vector v) { return x*v.y - y*v.x; }
22     double length() { return sqrt(dot(*this)); }
23     Vector unit_normal_vector() {
24         double len = length();
25         return Vector(-y/len, x/len);
26     }
27 };
28
29 using Point = Vector;
30
31 struct Line {
32     Point p;
33     Vector v;
34     double ang;
35     Line(Point p={}, Vector v={}): p(p), v(v) {
36         ang = atan2(v.y, v.x);
37     }
38     bool operator<(const Line& l) const {
39         return ang < l.ang;
40     }
41     Point intersection(Line l) {
42         Vector u = p - l.p;
43         double t = l.v.cross(u) / v.cross(l.v);
44         return p + v*t;
45     }
46 };
47
48 int n, m;
49 Line narrow[maxn]; // 要判斷的直線
50 Point poly[maxn]; // 能形成半平面交的凸包邊界點
51
52 // return true if point p is on the left of line l
53 bool onLeft(Point p, Line l) {
54     return l.v.cross(p-l.p) > 0;
55 }
56
57 int halfplaneIntersection() {
58     int l, r;
59     Line L[maxn]; // 排序後的向量隊列
60     Point P[maxn]; // s[i] 跟 s[i-1] 的交點
61
62     L[l=r=0] = narrow[0]; // notice: narrow is sorted
63     for(int i=1; i<n; i++) {
64         while(l<r && !onLeft(P[r-1], narrow[i])) r--;
65         while(l<r && !onLeft(P[l], narrow[i])) l++;
66
67         L[++r] = narrow[i];
68         if(l < r) P[r-1] = L[r-1].intersection(L[r]);
69     }
70
71     while(l<r && !onLeft(P[r-1], L[l])) r--;
72     if(r-l <= 1) return 0;
73
74     P[r] = L[r].intersection(L[l]);
75
76     int m=0;
77     for(int i=1; i<=r; i++) {
78         poly[m++] = P[i];
79     }
80
81     return m;
82 }

```

```

83 Point pt[maxn];
84 Vector vec[maxn];
85 Vector normal[maxn]; // normal[i] = vec[i] 的單位法向量
86
87 double bsearch(double l=0.0, double r=1e4) {
88     if(abs(r-l) < eps) return l;
89
90     double mid = (l + r) / 2;
91
92     for(int i=0; i<n; i++) {
93         narrow[i] = Line(pt[i]+normal[i]*mid, vec[i]);
94     }
95
96     if(halfplaneIntersection())
97         return bsearch(mid, r);
98     else return bsearch(l, mid);
99 }
100
101 int main() {
102     while(~scanf("%d", &n) && n) {
103         for(int i=0; i<n; i++) {
104             double x, y;
105             scanf("%lf%lf", &x, &y);
106             pt[i] = {x, y};
107         }
108         for(int i=0; i<n; i++) {
109             vec[i] = pt[(i+1)%n] - pt[i];
110             normal[i] = vec[i].unit_normal_vector();
111         }
112
113         printf("%.6lf\n", bsearch());
114     }
115     return 0;
116 }
117

```

## 6.3 凸包

```

1 // Q: 平面上給定多個區域，由多個座標點所形成，再給定
2 // 多點(x,y)，判斷有落點的區域(destroyed)的面積總和。
3 #include <bits/stdc++.h>
4 using namespace std;
5
6 const int maxn = 500 + 10;
7 const int maxCoordinate = 500 + 10;
8
9 struct Point {
10     int x, y;
11 };
12
13 int n;
14 bool destroyed[maxn];
15 Point arr[maxn];
16 vector<Point> polygons[maxn];
17
18 void scanAndSortPoints() {
19     int minX = maxCoordinate, minY = maxCoordinate;
20     for(int i=0; i<n; i++) {
21         int x, y;
22         scanf("%d%d", &x, &y);
23         arr[i] = (Point){x, y};
24         if(y < minY || (y == minY && x < minX)) {
25             // If there are floating points, use:
26             // if(y<minY || (abs(y-minY)<eps && x<minX)) {
27                 minX = x, minY = y;
28             }
29     }
30     sort(arr, arr+n, [minX, minY](Point& a, Point& b){
31         double theta1 = atan2(a.y - minY, a.x - minX);
32         double theta2 = atan2(b.y - minY, b.x - minX);
33         return theta1 < theta2;
34     });
35     return;
36 }
37

```

```

38 // returns cross product of u(AB) x v(AC)
39 int cross(Point& A, Point& B, Point& C) {
40     int u[2] = {B.x - A.x, B.y - A.y};
41     int v[2] = {C.x - A.x, C.y - A.y};
42     return (u[0] * v[1]) - (u[1] * v[0]);
43 }
44
45 // size of arr = n >= 3
46 // st = the stack using vector, m = index of the top
47 vector<Point> convex_hull() {
48     vector<Point> st(arr, arr+3);
49     for(int i=3, m=2; i<n; i++, m++) {
50         while(m >= 2) {
51             if(cross(st[m], st[m-1], arr[i]) < 0)
52                 break;
53             st.pop_back();
54             m--;
55         }
56         st.push_back(arr[i]);
57     }
58     return st;
59 }
60
61 bool inPolygon(vector<Point>& vec, Point p) {
62     vec.push_back(vec[0]);
63     for(int i=1; i<vec.size(); i++) {
64         if(cross(vec[i-1], vec[i], p) < 0) {
65             vec.pop_back();
66             return false;
67         }
68     }
69     vec.pop_back();
70     return true;
71 }
72
73     1 | x1   x2   x3   x4   x5           xn |
74 A = - |   x   x   x   x   x ... x   |
75     2 | y1   y2   y3   y4   y5           yn |
76 double calculateArea(vector<Point>& v) {
77     v.push_back(v[0]); // make v[n] = v[0]
78     double result = 0.0;
79     for(int i=1; i<v.size(); i++)
80         result += v[i-1].x*v[i].y - v[i-1].y*v[i].x;
81     v.pop_back();
82     return result / 2.0;
83 }
84
85 int main() {
86     int p = 0;
87     while(~scanf("%d", &n) && (n != -1)) {
88         scanAndSortPoints();
89         polygons[p++] = convex_hull();
90     }
91
92     int x, y;
93     double result = 0.0;
94     while(~scanf("%d%d", &x, &y)) {
95         for(int i=0; i<p; i++) {
96             if(inPolygon(polygons[i], (Point){x, y}))
97                 destroyed[i] = true;
98         }
99     }
100     for(int i=0; i<p; i++) {
101         if(destroyed[i])
102             result += calculateArea(polygons[i]);
103     }
104     printf("%.2lf\n", result);
105     return 0;
106 }

```

## 7 DP

### 7.1 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 ->
   當目前只考慮到第i個商品時，達到獲利j時最少的weight總和
   = w
9 所以答案是dp[n][1 ~ maxv]找價值最大且裝的下的*/
10 #define maxn 105
11 #define maxv 100005
12 long long dp[maxn][maxv];
13 long long weight[maxn];
14 long long v[maxn];
15 int main() {
16     int n;
17     long long w;
18     scanf("%d %lld", &n, &w);
19     for (int i = 1; i <= n; ++i) {
20         scanf("%lld %lld", &weight[i], &v[i]);
21     }
22     memset(dp, 0x3f, sizeof(dp));
23     dp[0][0] = 0;
24     for (int i = 1; i <= n; ++i) {
25         for (int j = 0; j <= maxv; ++j) {
26             if (j - v[i] >= 0)
27                 dp[i][j] = dp[i - 1][j - v[i]] +
28                     weight[i];
29             dp[i][j] = min(dp[i - 1][j], dp[i][j]);
30         }
31     }
32     long long res = 0;
33     for (int j = maxv - 1; j >= 0; --j) {
34         if (dp[n][j] <= w) {
35             res = j;
36             break;
37         }
38     }
39     printf("%lld\n", res);
40     return 0;
41 }

```

### 7.2 抽屜

```

1 // dp[n][s][t] n: 幾個抽屜 s: 幾個是安全的 t: (0 or
2 1) 最上面的抽屜是U or L
3 // 分兩種case
4 // case 1: dp[n][s][0] = dp[n - 1][s + 1][1] + dp[n -
5 1][s][0]
6 // 此時最上面放U，則
7 // dp[n - 1][s + 1][1]: 現在要放的U會導致底下n -
8 1個抽屜最上面L變不安全，為了得到n個抽屜s個安全，所以要s
9 + 1
10 // dp[n - 1][s][0]: n -
11 1個抽屜有s個安全，現在在其上面再放一個U不影響s的數量
12 // case 2: dp[n][s][1] = dp[n - 1][s - 1][1] + dp[n -
13 1][s - 1][0]
14 // 在最上面放L，底下n - 1個抽屜有s -
15 1個安全，無論上方是U、L皆不影響
16 long long dp[70][70][2];
17 // 初始條件
18 dp[1][0][0] = dp[1][1][1] = 1;
19 for (int i = 2; i <= 66; ++i) {
20     // i個抽屜0個安全且上方0 = (底下i -
21     1個抽屜且1個安全且最上面L) + (底下n -
22     1個抽屜0個安全且最上方為0)
23     dp[i][0][0] = dp[i - 1][1][1] + dp[i - 1][0][0];
24     for (int j = 1; j <= i; ++j) {
25         dp[i][j][0] = dp[i - 1][j + 1][1] + dp[i -
26         1][j][0];
27         dp[i][j][1] = dp[i - 1][j - 1][1] + dp[i -
28         1][j - 1][0];
29     }
30 }

```

```

18     }
19 }
20 //答案在 dp[n][s][0] + dp[n][s][1]);

```

### 7.3 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) != EOF) {
21         memset(dp, -1, sizeof(dp));
22         printf("%lld\n", dfs(N, K));
23     }
24     return 0;
25 }

```

### 7.4 Deque 最大差距

```

1 /*定義dp[l][r]是l ~ r時與先手最大差異值
2 Deque可以拿頭尾
3 所以轉移式中dp[l][r]與dp[l + 1][r]、dp[l][r - 1]有關
4 轉移式:
5 dp[l][r] = max{a[l] - solve(l + 1, r), a[r] -
6 solve(l, r - 1)}
7 裡面用減的主要是因為求的是相減且會一直換手，所以正負正負
8 #define maxn 3005
9 bool vis[maxn][maxn];
10 long long dp[maxn][maxn];
11 long long a[maxn];
12 long long solve(int l, int r) {
13     if (l > r)
14         return 0;
15     if (vis[l][r])
16         return dp[l][r];
17     vis[l][r] = true;
18     long long res = a[l] - solve(l + 1, r);
19     res = max(res, a[r] - solve(l, r - 1));
20     return dp[l][r] = res;
21 }
22 int main() {
23     ...
24     printf("%lld\n", solve(1, n));
25 }

```

### 7.5 LCS 和 LIS

```

1 //最長共同子序列(LCS)
2 給定兩序列 A,B，求最長的序列 C，
3 C 同時為 A,B 的子序列。
4 //最長遞增子序列 (LIS)
5 給你一個序列 A，求最長的序列 B，
6 B 是一個 (非) 嚴格遞增序列，且為 A 的子序列。
7 //LCS 和 LIS 題目轉換

```

```

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

```

### 7.6 RangeDP

```

1 //區間dp
2 int dp[55][55]; // dp[i][j] -> [i,
3 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) + solve(m, j) +
15            cuts[j] - cuts[i]);
16    }
17    return dp[i][j] = cost;
18 }
19 int main() {
20     int l;
21     int n;
22     while (scanf("%d", &l) != EOF && l){
23         scanf("%d", &n);
24         for (int i = 1; i <= n; ++i)
25             scanf("%d", &cuts[i]);
26         cuts[0] = 0;
27         cuts[n + 1] = 1;
28         memset(dp, -1, sizeof(dp));
29         printf("The minimum cutting is %d.\n",
30             solve(0, n + 1));
31     }
32     return 0;
33 }

```

### 7.7 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

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

### 7.8 TreeDP 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)
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    //統計橫跨u但還是在u的子樹中合計長度為k的:
23    //考慮u有一子節點v, 在v子樹中距離v長度為x的
24    //以及不在v子樹但在u子樹中(這樣才會是橫跨u)且距離u長度
    // - x - 1的
25    //共有 0.5 * (dp[v][x] * (dp[u][k - x - 1] -
    // dp[v][k - x - 2]))
26    //以上算式是重點, 可使複雜度下降, 否則枚舉一定超時
27    //其中 dp[u][k - x - 1]是所有u子樹中距離u為k - x
    // - 1的節點
28    // - dp[v][k - x -
    // 2]是因為我們不要v子樹的節點且距離u為k - x -
    // 1的(要v子樹以外的),
29    //那些點有dp[v][k - x - 2], 最後0.5是由於計算中i
    // -> j以及j -> i(i、j是不同節點)
30    //都會被算一遍, 所以要 * 0.5
31    long long cnt = 0;
32    for (int v: G[u]) {
33        if (v == p)
34            continue;
35        for (int x = 0; x <= k - 2; ++x) {
36            cnt += dp[v][x] * (dp[u][k - x - 1] -
    dp[v][k - x - 2]);
37        }
38    }
39    res += cnt / 2;
40 }
41 int main() {
42     scanf("%d %d", &n, &k);
43     G.assign(n + 5, vector<int>());
44     int u, v;
45     for (int i = 1; i < n; ++i) {
46         scanf("%d %d", &u, &v);
47         G[u].emplace_back(v);
48         G[v].emplace_back(u);
49     }
50     dfs(1, -1);
51     printf("%lld\n", res);
52     return 0;
53 }

```

## 7.9 TreeDP reroot

```

1 /*Re-root經典題
2 1. 選0作為root
3 2. 以0為root去求出所有節點的 subtreeSize
4 3. 觀察到re-root後的關係式
5 配合思考圖片
6 f(0)與f(2)的關係
7 f(2) = f(0) + a - b
8 a = n - b, (subtree(2)以外的節點)
9 b = subtreeSize(2), (subtree(2))
10 所以f(n)是n為root到所有點的距離
11 f(2) = f(0) + n - 2 * subtreeSize(2)
12 這就是快速得到答案的轉移式
13 f(child) = f(parent) + n - 2 * subtreeSize(child)
14 流程
15 1. root = 0去求各項 subtreeSize

```

```

16 2. 求f(root)
17 3. 以f(0)去求出re-root後的所有f(v), v != 0
18 整體來說
19 暴力解 O(n ^ 2)
20 re-root dp on tree O(n + n + n) -> O(n)*
21 class Solution {
22 public:
23     vector<int> sumOfDistancesInTree(int n,
    vector<vector<int>>& edges) {
24         this->res.assign(n, 0);
25         G.assign(n + 5, vector<int>());
26         for (vector<int>& edge: edges) {
27             G[edge[0]].emplace_back(edge[1]);
28             G[edge[1]].emplace_back(edge[0]);
29         }
30         memset(this->visited, 0,
    sizeof(this->visited));
31         this->dfs(0);
32         memset(this->visited, 0,
    sizeof(this->visited));
33         this->res[0] = this->dfs2(0, 0);
34         memset(this->visited, 0,
    sizeof(this->visited));
35         this->dfs3(0, n);
36         return this->res;
37     }
38 private:
39     vector<vector<int>> G;
40     bool visited[30005];
41     int subtreeSize[30005];
42     vector<int> res;
43     //求subtreeSize
44     int dfs(int u) {
45         this->visited[u] = true;
46         for (int v: this->G[u]) {
47             if (!this->visited[v]) {
48                 this->subtreeSize[u] += this->dfs(v);
49             }
50         }
51         //自己
52         this->subtreeSize[u] += 1;
53         return this->subtreeSize[u];
54     }
55     //求res[0], 0到所有點的距離
56     int dfs2(int u, int dis) {
57         this->visited[u] = true;
58         int sum = 0;
59         for (int v: this->G[u]) {
60             if (!visited[v]) {
61                 sum += this->dfs2(v, dis + 1);
62             }
63         }
64         //要加上自己的距離
65         return sum + dis;
66     }
67     //算出所有的res
68     void dfs3(int u, int n) {
69         this->visited[u] = true;
70         for (int v: this->G[u]) {
71             if (!visited[v]) {
72                 this->res[v] = this->res[u] + n - 2 *
    this->subtreeSize[v];
73                 this->dfs3(v, n);
74             }
75         }
76     }
77 };

```

## 7.10 Weighted LIS

```

1 /*概念基本上與LIS相同, 但不能用greedy的LIS, 所以只能用dp版LIS
2 但有個問題是dp版要O(n^2)
3 n最大200000一定超時, 所以這題要改一下dp的LIS

```

```

4 | 在DP版中有一層迴圈是要往前搜  $height[j] < height[i](j$ 
   |    $in\ 1 \sim i - 1)$  的然後挑  $B[j]$  最大的
5 | 這for loop造成  $O(n^2)$ 
6 | 注意到子問題是在  $1 \sim i - 1$  中挑出  $B[j]$  最大的
7 | 這一步可以用線段樹優化
8 | 所以最後可以在  $O(n \log n)$  完成*/
9 | #define maxn 200005
10 | long long dp[maxn];
11 | long long height[maxn];
12 | long long B[maxn];
13 | long long st[maxn << 2];
14 | void update(int p, int index, int l, int r, long long
   |   v) {
15 |     if (l == r) {
16 |         st[index] = v;
17 |         return;
18 |     }
19 |     int mid = (l + r) >> 1;
20 |     if (p <= mid)
21 |         update(p, (index << 1), l, mid, v);
22 |     else
23 |         update(p, (index << 1) + 1, mid + 1, r, v);
24 |     st[index] = max(st[index << 1], st[(index << 1) +
   |       1]);
25 | }
26 | long long query(int index, int l, int r, int ql, int
   |   qr) {
27 |     if (ql <= l && r <= qr)
28 |         return st[index];
29 |     int mid = (l + r) >> 1;
30 |     long long res = -1;
31 |     if (ql <= mid)
32 |         res = max(res, query(index << 1, l, mid, ql,
   |           qr));
33 |     if (mid < qr)
34 |         res = max(res, query((index << 1) + 1, mid +
   |           1, r, ql, qr));
35 |     return res;
36 | }
37 | int main() {
38 |     int n;
39 |     scanf("%d", &n);
40 |     for (int i = 1; i <= n; ++i)
41 |         scanf("%lld", &height[i]);
42 |     for (int i = 1; i <= n; ++i)
43 |         scanf("%lld", &B[i]);
44 |     long long res = B[1];
45 |     update(height[1], 1, 1, n, B[1]);
46 |     for (int i = 2; i <= n; ++i) {
47 |         long long temp;
48 |         if (height[i] - 1 >= 1)
49 |             temp = B[i] + query(1, 1, n, 1, height[i]
   |               - 1);
50 |         else
51 |             temp = B[i];
52 |         update(height[i], 1, 1, n, temp);
53 |         res = max(res, temp);
54 |     }
55 |     printf("%lld\n", res);
56 |     return 0;
57 | }

```