

## Contents

1	字串	1
1.1	最長迴文子字串	1
1.2	KMP	1
2	math	1
2.1	SG	1
2.2	質數與因數	1
2.3	歐拉函數	2
2.4	大步小步	2
3	algorithm	3
3.1	三分搜	3
3.2	差分	3
3.3	greedy	3
3.4	dinic	5
3.5	SCC Tarjan	5
3.6	ArticulationPoints Tarjan	6
3.7	最小樹狀圖	6
3.8	JosephusProblem	7
3.9	KM	8
3.10	LCA 倍增法	8
3.11	MCMF	9
3.12	Dancing Links	10
4	DataStructure	10
4.1	線段樹 1D	10
4.2	線段樹 2D	11
4.3	權值線段樹	11
4.4	Trie	12
4.5	單調隊列	12
5	geometry	13
5.1	intersection	13
5.2	半平面相交	13
5.3	凸包	14
6	DP	15
6.1	抽屜	15
6.2	Deque 最大差距	15
6.3	LCS 和 LIS	15
6.4	RangeDP	15
6.5	stringDP	15
6.6	TreeDP 有幾個 path 長度為 k	15
6.7	TreeDP reroot	16
6.8	WeightedLIS	16

## 1 字串

### 1.1 最長迴文子字串

```

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

```

```

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 math

### 2.1 SG

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

### 2.2 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
12             j=0;j<primeSize&&i*prime[j]<=MAXN;++j){
13             isPrime[i*prime[j]]=false;
14             if(i%prime[j]==0) break;
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 歌德巴赫猜想
59 solution : 把偶數 N (6≤N≤10^6) 寫成兩個質數的和。
60 #define N 2000000
61 int ox[N],p[N],pr;
62 void PrimeTable(){
63     ox[0]=ox[1]=1;
64     pr=0;
65     for(int i=2;i<N;i++){
66         if(!ox[i]) p[pr++]=i;
67         for(int j=0;i*p[j]<N&&j<pr;j++)
68             ox[i*p[j]]=1;
69     }
70 }
71
72 int main(){
73     PrimeTable();
74     int n;
75     while(cin>>n,n){
76         int x;
77         for(x=1;x+=2)
78             if(!ox[x]&&!ox[n-x]) break;
79         printf("%d = %d + %d\n",n,x,n-x);
80     }
81 }
82
83 problem : 給定整數 N ,
84 求 N 最少可以拆成多少個質數的和。
85 如果 N 是質數,則答案為 1。
86 如果 N 是偶數(不包含2),則答案為 2 (強歌德巴赫猜想)。
87 如果 N 是奇數且 N-2 是質數,則答案為 2 (2+質數)。
88 其他狀況答案為 3 (弱歌德巴赫猜想)。
89
90 bool isPrime(int n){
91     for(int i=2;i<n;i++){
92         if(i*i>n) return true;
93         if(n%i==0) return false;
94     }
95     return true;
96 }
97
98 int main(){
99     int n;
100     cin>>n;
101     if(isPrime(n)) cout<<"1\n";
102     else if(n%2==0||isPrime(n-2)) cout<<"2\n";
103     else cout<<"3\n";
104 }

```

## 2.3 歐拉函數

1 //計算閉區間  $[1,n]$  中有幾個正整數與  $n$  互質

```

2
3 int phi(){
4     int ans=n;
5     for(int i=2;i*i<=n;i++){
6         if(n%i==0){
7             ans=ans-ans/i;
8             while(n%i==0) n/=i;
9         }
10    if(n>1) ans=ans-ans/n;
11    return ans;
12 }

```

## 2.4 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
    $B^0, B^1, B^2, \dots, B^{P-1}$ 
5  也就是說如果有解則  $L < P$ , 枚舉  $0, 1, 2, \dots, P-1$ 
   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  $B^y \equiv N(B^{mx})^{-1} \pmod{P}$ 
11 先求出  $B^0, B^1, B^2, \dots, B^{m-1}$ ,
12 再枚舉  $N(B^{mx})^{-1}, N(B^{mx})^{-2}, \dots$  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 unordered_map 存放  $B^0, B^1, B^2, \dots, B^{m-1}$ ,
18 枚舉  $x$  查詢 unordered_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  $O(\sqrt{P} \log P) / O(\sqrt{P})$ 。
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;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?-1: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 }

```

### 3 algorithm

«««< HEAD ===== »»»> cc6ba60f318928240554b362b91a42dc337ff3d4

#### 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, 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 };
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)<<sqrt(ans)<<"\n";
59     }
60 }

```

#### 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 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。
7
8 刪數字問題
9 //problem
10 給定一個數字 N(≤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 }
36 最小區間覆蓋長度
37 //problem
38 給定 n 條線段區間為 [Li,Ri]，
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  $[Li, Ri]$ ，要從左界  $>Ri$  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。

44 **//problem**

45 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];
55     cin>>n>>r;
56     for(int i=1; i<=n; ++i) cin>>a[i];
57     int i=1, ans=0;
58     while(i<=n){
59         int R=min(i+r-1, n), L=max(i-r+1, 0);
60         int nextR=-1;
61         for(int j=R; j>=L; --j){
62             if(a[j]){
63                 nextR=j;
64                 break;
65             }
66         }
67         if(nextR==-1){
68             ans=-1;
69             break;
70         }
71         ++ans;
72         i=nextR+r;
73     }
74     cout<<ans<<'\n';
75 }
```

76 最多不重疊區間

77 **//problem**

78 給你  $n$  條線段區間為  $[Li, Ri]$ ，  
79 請問最多可以選擇幾條不重疊的線段(頭尾可相連)?

80 **//solution**

81 依照右界由小到大排序，  
82 每次取到一個不重疊的線段，答案  $+1$ 。

83 **//code**

```
84 struct Line{
85     int L, R;
86     bool operator<(const Line &rhs) const{
87         return R<rhs.R;
88     }
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  $Li=\max(0, Fi-Di)$ ，  
114 原本  $Fi$  為第  $i$  項工作的完成時間，  
115 求一種工作排序使  $\max Li$  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
126 int main(){
127     int n;
128     Work a[10000];
129     cin>>n;
130     for(int i=0; i<n; ++i)
131         cin>>a[i].t>>a[i].d;
132     sort(a, a+n);
133     int maxL=0, sumT=0;
134     for(int i=0; i<n; ++i){
135         sumT+=a[i].t;
136         maxL=max(maxL, sumT-a[i].d);
137     }
138     cout<<maxL<<'\n';
139 }
```

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 **//code**

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 任務調度問題

184 **//problem**

185 給定  $N$  項工作，每項工作的需要處理時長為  $T_i$ ，  
186 期限是  $D_i$ ，如果第  $i$  項工作延遲需要受到  $p_i$  單位懲罰，  
187 請問最少會受到多少單位懲罰。

188 **//solution**

189 依照懲罰由大到小排序，

190 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 如果有空間就放進去，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  $a_i$ 
    單位獎勵，
197 請問最多會獲得多少單位獎勵。
198 //solution
199 和上題相似，這題變成依照獎勵由大到小排序。
200 //code
201 struct Work{
202     int d,p;
203     bool operator<(const Work &rhs)const{
204         return p>rhs.p;
205     }
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     }

```

### 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 && 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 }

```

### 3.5 SCC Tarjan

```

1 //單純考SCC，每個SCC中找成本最小的蓋，如果有多個一樣小的要數出來
2 //注意以下程式有縮點，但沒存起來，存法就是開一個array
   -> ID[u] = SCCID
3 #define maxn 100005
4 #define MOD 1000000007
5 long long cost[maxn];
6 vector<vector<int>> G;
7 int SCC = 0;
8 stack<int> sk;
9 int dfn[maxn];
10 int low[maxn];
11 bool inStack[maxn];
12 int dfsTime = 1;
13 long long totalCost = 0;
14 long long ways = 1;
15 void dfs(int u) {
16     dfn[u] = low[u] = dfsTime;
17     ++dfsTime;
18     sk.push(u);
19     inStack[u] = true;
20     for (int v: G[u]) {
21         if (dfn[v] == 0) {
22             dfs(v);
23             low[u] = min(low[u], low[v]);
24         }
25         else if (inStack[v]) {
26             //屬於同個SCC且是我的back edge
27             low[u] = min(low[u], dfn[v]);
28         }
29     }
30     //如果是SCC
31     if (dfn[u] == low[u]) {
32         long long minCost = 0x3f3f3f3f;
33         int currWays = 0;
34         ++SCC;
35         while (1) {
36             int v = sk.top();
37             inStack[v] = 0;
38             sk.pop();
39             if (minCost > cost[v]) {
40                 minCost = cost[v];
41                 currWays = 1;
42             }
43             else if (minCost == cost[v]) {

```

```

44         ++currWays;
45     }
46     if (v == u)
47         break;
48 }
49 totalCost += minCost;
50 ways = (ways * currWays) % MOD;
51 }
52 }
53 int main() {
54     int n;
55     scanf("%d", &n);
56     for (int i = 1; i <= n; ++i)
57         scanf("%lld", &cost[i]);
58     G.assign(n + 5, vector<int>());
59     int m;
60     scanf("%d", &m);
61     int u, v;
62     for (int i = 0; i < m; ++i) {
63         scanf("%d %d", &u, &v);
64         G[u].emplace_back(v);
65     }
66     for (int i = 1; i <= n; ++i) {
67         if (dfn[i] == 0)
68             dfs(i);
69     }
70     printf("%lld %lld\n", totalCost, ways % MOD);
71     return 0;
72 }

```

### 3.6 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 }

```

### 3.7 最小樹狀圖

```

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

```



```

55         e.t = idx[e.t];
56     }
57     N = seq;
58     root = idx[root];
59 }
60 }
61 =====
62 O(m+nlog n)時間內解決最小樹形圖問題的演算法。
63 typedef long long ll;
64 #define maxn 102
65 #define INF 0x3f3f3f3f
66 struct UnionFind {
67     int fa[maxn << 1];
68     UnionFind() { memset(fa, 0, sizeof(fa)); }
69     void clear(int n) {
70         memset(fa + 1, 0, sizeof(int) * n);
71     }
72     int find(int x) {
73         return fa[x] ? fa[x] = find(fa[x]) : x;
74     }
75     int operator[](int x) { return find(x); }
76 };
77 struct Edge {
78     int u, v, w, w0;
79 };
80 struct Heap {
81     Edge *e;
82     int rk, constant;
83     Heap *lch, *rch;
84     Heap(Edge *_e):
85         e(_e), rk(1), constant(0), lch(NULL), rch(NULL) {}
86     void push() {
87         if (lch) lch->constant += constant;
88         if (rch) rch->constant += constant;
89         e->w += constant;
90         constant = 0;
91     }
92 };
93 Heap *merge(Heap *x, Heap *y) {
94     if (!x) return y;
95     if (!y) return x;
96     if (x->e->w + x->constant > y->e->w + y->constant)
97         swap(x, y);
98     x->push();
99     x->rch = merge(x->rch, y);
100     if (!x->lch || x->lch->rk < x->rch->rk)
101         swap(x->lch, x->rch);
102     if (x->rch)
103         x->rk = x->rch->rk + 1;
104     else
105         x->rk = 1;
106     return x;
107 }
108 Edge *extract(Heap *&x) {
109     Edge *r = x->e;
110     x->push();
111     x = merge(x->lch, x->rch);
112     return r;
113 }
114 vector<Edge> in[maxn];
115 int n, m, fa[maxn << 1], nxt[maxn << 1];
116 Edge *ed[maxn << 1];
117 Heap *Q[maxn << 1];
118 UnionFind id;
119 void contract() {
120     bool mark[maxn << 1];
121     //將圖上的每一個節點與其相連的那些節點進行記錄
122     for (int i = 1; i <= n; i++) {
123         queue<Heap *> q;
124         for (int j = 0; j < in[i].size(); j++)
125             q.push(new Heap(&in[i][j]));
126         while (q.size() > 1) {
127             Heap *u = q.front();
128             q.pop();
129             Heap *v = q.front();
130             q.pop();
131             q.push(merge(u, v));

```

```

132     }
133     Q[i] = q.front();
134 }
135 mark[1] = true;
136 for(int a=1,b=1,p;Q[a];b=a,mark[b]=true){
137     //尋找最小入邊以及其端點，保證無環
138     do {
139         ed[a] = extract(Q[a]);
140         a = id[ed[a]->u];
141     } while (a == b && Q[a]);
142     if (a == b) break;
143     if (!mark[a]) continue;
144     //對發現的環進行收縮，以及環內的節點重新編號，
145     //總權值更新
146     for (a = b, n++; a != n; a = p) {
147         id.fa[a] = fa[a] = n;
148         if (Q[a]) Q[a]->constant -= ed[a]->w;
149         Q[n] = merge(Q[n], Q[a]);
150         p = id[ed[a]->u];
151         nxt[p == n ? b : p] = a;
152     }
153 }
154 }
155 ll expand(int x, int r);
156 ll expand_iter(int x) {
157     ll r = 0;
158     for(int u=nxt[x];u!=x;u=nxt[u]){
159         if (ed[u]->w0 >= INF)
160             return INF;
161         else
162             r+=expand(ed[u]->v,u)+ed[u]->w0;
163     }
164     return r;
165 }
166 ll expand(int x, int t) {
167     ll r = 0;
168     for (; x != t; x = fa[x]) {
169         r += expand_iter(x);
170         if (r >= INF) return INF;
171     }
172     return r;
173 }
174 void link(int u, int v, int w) {
175     in[v].push_back({u, v, w, w});
176 }
177 int main() {
178     int rt;
179     scanf("%d %d %d", &n, &m, &rt);
180     for (int i = 0; i < m; i++) {
181         int u, v, w;
182         scanf("%d %d %d", &u, &v, &w);
183         link(u, v, w);
184     }
185     //保證強連通
186     for (int i = 1; i <= n; i++)
187         link(i > 1 ? i - 1 : n, i, INF);
188     contract();
189     ll ans = expand(rt, n);
190     if (ans >= INF)
191         puts("-1");
192     else
193         printf("%lld\n", ans);
194     return 0;
195 }

```

### 3.8 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) == 1){
16                 printf("%d\n", k);
17                 break;
18             }
19         }
20     }
21     return 0;
22 }

```

### 3.9 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
17 #define maxn 505
18 int W[maxn][maxn];
19 int Lx[maxn], Ly[maxn];
20 bool S[maxn], T[maxn];
21 //L[i] = j -> S_i配給T_j, -1 for 還沒匹配
22 int L[maxn];
23 int n;
24 bool match(int i) {
25     S[i] = true;
26     for (int j = 0; j < n; ++j) {
27         // KM重點
28         // Lx + Ly >= selected_edge(x, y)
29         // 要想辦法降低Lx + Ly
30         // 所以選Lx + Ly == selected_edge(x, y)
31         if (Lx[i] + Ly[j] == W[i][j] && !T[j]) {
32             T[j] = true;
33             if ((L[j] == -1) || match(L[j])) {
34                 L[j] = i;
35                 return true;
36             }
37         }
38     }
39     return false;
40 }
41 // 修改二分圖上的交錯路徑上點的權重
42 // 此舉是在通過調整vertex
43 // labeling看看能不能產生出新的增廣路(KM的增廣路要求Lx[i]
44 // + Ly[j] == W[i][j])
45 // 在這裡優先從最小的diff調調看，才能保證最大權重匹配
46 void update()
47 {
48     int diff = 0x3f3f3f3f;
49     for (int i = 0; i < n; ++i) {
50         if (S[i]) {
51             for (int j = 0; j < n; ++j) {
52                 if (!T[j])
53                     diff = min(diff, Lx[i] + Ly[j] - W[i][j]);
54             }
55         }
56     }
57 }

```

```

52     for (int i = 0; i < n; ++i) {
53         if (S[i]) Lx[i] -= diff;
54         if (T[i]) Ly[i] += diff;
55     }
56 }
57 void KM()
58 {
59     for (int i = 0; i < n; ++i) {
60         L[i] = -1;
61         Lx[i] = Ly[i] = 0;
62         for (int j = 0; j < n; ++j)
63             Lx[i] = max(Lx[i], W[i][j]);
64     }
65     for (int i = 0; i < n; ++i) {
66         while(1) {
67             memset(S, false, sizeof(S));
68             memset(T, false, sizeof(T));
69             if (match(i))
70                 break;
71             else
72                 update(); //去調整vertex
73                             //labeling以增加增廣路徑
74         }
75     }
76 }
77 int main() {
78     while (scanf("%d", &n) != EOF) {
79         for (int i = 0; i < n; ++i)
80             for (int j = 0; j < n; ++j)
81                 scanf("%d", &W[i][j]);
82         KM();
83         int res = 0;
84         for (int i = 0; i < n; ++i) {
85             if (i != 0)
86                 printf("%d", Lx[i]);
87             printf("%d", Lx[i]);
88             res += Lx[i];
89         }
90         puts("");
91         for (int i = 0; i < n; ++i) {
92             if (i != 0)
93                 printf("%d", Ly[i]);
94             printf("%d", Ly[i]);
95             res += Ly[i];
96         }
97         puts("");
98         printf("%d\n", res);
99     }
100     return 0;
101 }
102 }

```

### 3.10 LCA 倍增法

```

1  //倍增法預處理O(nlogn)，查詢O(logn)，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] +
18             dis[u][i-1];
19     }
20 }

```



```

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

```

### 3.11 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 //
  所以parent存的也是對應edges[]中的edgeIndex(主要是方便)
17 int parent[maxn];
18 //maxFlow時需要紀錄到node u時的bottleneck
19 //同時也代表著u該次流出去的量
20 long long outFlow[maxn];
21 void addEdge(int u, int v, int cap, int cost) {
22     edges.emplace_back(Edge{u, v, cap, 0, cost});
23     edges.emplace_back(Edge{v, u, 0, 0, -cost});
24     m = edges.size();
25     G[u].emplace_back(m - 2);
26     G[v].emplace_back(m - 1);
27 }
28 //一邊求最短路的同時一邊MaxFlow
29 bool SPFA(long long& maxFlow, long long& minCost) {
30     //memset(outFlow, 0x3f, sizeof(outFlow));
31     memset(dis, 0x3f, sizeof(dis));
32     memset(inqueue, false, sizeof(inqueue));
33     queue<int> q;
34     q.push(s);
35     dis[s] = 0;
36     inqueue[s] = true;
37     outFlow[s] = INF;
38     while (!q.empty()) {
39         int u = q.front();
40         q.pop();
41         inqueue[u] = false;
42         for (const int edgeIndex: G[u]) {
43             const Edge& edge = edges[edgeIndex];
44             if ((edge.cap > edge.flow) &&
45                 (dis[edge.v] > dis[u] + edge.cost)) {
46                 dis[edge.v] = dis[u] + edge.cost;
47                 parent[edge.v] = edgeIndex;
48                 outFlow[edge.v] = min(outFlow[u],
49                     (long long)(edge.cap -
50                         edge.flow));
51                 if (!inqueue[edge.v]) {
52                     q.push(edge.v);
53                     inqueue[edge.v] = true;
54                 }
55             }
56         }
57     }
58     //如果dis[t] > 0代表根本不賺還倒賠
59     if (dis[t] > 0)
60         return false;
61     maxFlow += outFlow[t];
62     minCost += dis[t] * outFlow[t];
63     //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)
64     int curr = t;
65     while (curr != s) {
66         edges[parent[curr]].flow += outFlow[t];
67         edges[parent[curr] ^ 1].flow -= outFlow[t];
68         curr = edges[parent[curr]].u;
69     }
70     return true;
71 }
72 long long MCMF() {
73     long long maxFlow = 0;
74     long long minCost = 0;
75     while (SPFA(maxFlow, minCost))
76         ;
77     return minCost;
78 }
79 int main() {
80     int T;
81     scanf("%d", &T);
82     for (int Case = 1; Case <= T; ++Case) {
83         //總共幾個月，囤貨成本
84         int M, I;
85         scanf("%d %d", &M, &I);
86         //node size
87         n = M + M + 2;
88         G.assign(n + 5, vector<int>());

```

```

86     edges.clear();
87     s = 0;
88     t = M + M + 1;
89     for (int i = 1; i <= M; ++i) {
90         int produceCost, produceMax, sellPrice,
91             sellMax, inventoryMonth;
92         scanf("%d %d %d %d %d", &produceCost,
93             &produceMax, &sellPrice, &sellMax,
94             &inventoryMonth);
95         addEdge(s, i, produceMax, produceCost);
96         addEdge(M + i, t, sellMax, -sellPrice);
97         for (int j = 0; j <= inventoryMonth; ++j)
98             if (i + j <= M)
99                 addEdge(i, M + i + j, INF, I * j);
100     }
101     printf("Case %d: %lld\n", Case, -MCMF());
102 }

```

### 3.12 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 };

```

## 4 DataStructure

### 4.1 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
57         tag[i] = 0;
58     }
59     if (ql <= mid) update(ql, qr, l, mid, i * 2, c);
60     if (qr > mid) update(ql, qr, mid + 1, r, i * 2 + 1, c);
61     st[i] = pull(i * 2, i * 2 + 1);
62 }
63 //如果是直接改值而不是加值, query與update中的tag與st的
64 //改值從+=改成=

```

## 4.2 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
6     yPos, int xIndex, bool xIsLeaf) {
7     if (l == r) {
8         if (xIsLeaf) {
9             maxST[xIndex][index] =
10                 minST[xIndex][index] = val;
11             return;
12         }
13         maxST[xIndex][index] = max(maxST[xIndex *
14             2][index], maxST[xIndex * 2 + 1][index]);
15         minST[xIndex][index] = min(minST[xIndex *
16             2][index], minST[xIndex * 2 + 1][index]);
17     }
18     else {
19         int mid = (l + r) / 2;
20         if (yPos <= mid)
21             modifyY(index * 2, l, mid, val, yPos,
22                 xIndex, xIsLeaf);
23         else
24             modifyY(index * 2 + 1, mid + 1, r, val,
25                 yPos, xIndex, xIsLeaf);
26
27         maxST[xIndex][index] =
28             max(maxST[xIndex][index * 2],
29                 maxST[xIndex][index * 2 + 1]);
30         minST[xIndex][index] =
31             min(minST[xIndex][index * 2],
32                 minST[xIndex][index * 2 + 1]);
33     }
34 }
35 void modifyX(int index, int l, int r, int val, int
36     xPos, int yPos) {
37     if (l == r) {
38         modifyY(1, 1, N, val, yPos, index, true);
39     }
40     else {
41         int mid = (l + r) / 2;
42         if (xPos <= mid)
43             modifyX(index * 2, l, mid, val, xPos,
44                 yPos);

```

```

33     else
34         modifyX(index * 2 + 1, mid + 1, r, val,
35             xPos, yPos);
36     modifyY(1, 1, N, val, yPos, index, false);
37 }
38 void queryY(int index, int l, int r, int yql, int
39     yqr, int xIndex, int& vmax, int& vmin) {
40     if (yql <= l && r <= yqr) {
41         vmax = max(vmax, maxST[xIndex][index]);
42         vmin = min(vmin, minST[xIndex][index]);
43     }
44     else
45     {
46         int mid = (l + r) / 2;
47         if (yql <= mid)
48             queryY(index * 2, l, mid, yql, yqr,
49                 xIndex, vmax, vmin);
50         if (mid < yqr)
51             queryY(index * 2 + 1, mid + 1, r, yql,
52                 yqr, xIndex, vmax, vmin);
53     }
54 }
55 void queryX(int index, int l, int r, int xql, int
56     xqr, int yql, int yqr, int& vmax, int& vmin) {
57     if (xql <= l && r <= xqr) {
58         queryY(1, 1, N, yql, yqr, index, vmax, vmin);
59     }
60     else {
61         int mid = (l + r) / 2;
62         if (xql <= mid)
63             queryX(index * 2, l, mid, xql, xqr, yql,
64                 yqr, vmax, vmin);
65         if (mid < xqr)
66             queryX(index * 2 + 1, mid + 1, r, xql,
67                 xqr, yql, yqr, vmax, vmin);
68     }
69 }
70 int main() {
71     while (scanf("%d", &N) != EOF) {
72         int val;
73         for (int i = 1; i <= N; ++i) {
74             for (int j = 1; j <= N; ++j) {
75                 scanf("%d", &val);
76                 modifyX(1, 1, N, val, i, j);
77             }
78         }
79         int q;
80         int vmax, vmin;
81         int xql, xqr, yql, yqr;
82         char op;
83         scanf("%d", &q);
84         while (q--) {
85             getchar(); //for \n
86             scanf("%c", &op);
87             if (op == 'q') {
88                 scanf("%d %d %d %d", &xql, &yql,
89                     &xqr, &yqr);
90                 vmax = -0x3f3f3f3f;
91                 vmin = 0x3f3f3f3f;
92                 queryX(1, 1, N, xql, xqr, yql, yqr,
93                     vmax, vmin);
94                 printf("%d %d\n", vmax, vmin);
95             }
96             else {
97                 scanf("%d %d %d", &xql, &yql, &val);
98                 modifyX(1, 1, N, val, xql, yql);
99             }
100         }
101     }
102     return 0;
103 }

```

## 4.3 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 -
32             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
57             + 1);
58         for (int i = 1; i <= m; ++i) {
59             nums[i] = lower_bound(id + 1, id + stSize
60                 + 1, nums[i]) - id;
61         }
62         int addCount = 0;
63         int getCount = 0;
64         int k = 1;
65         while (getCount < n) {
66             if (getArr[getCount] == addCount) {
67                 printf("%d\n", query(1, 1, stSize,
68                     k));
69                 ++k;
70                 ++getCount;
71             }
72             else {
73                 update(1, 1, stSize, nums[addCount +
74                     1]);
75                 ++addCount;
76             }
77         }
78     }
79 }

```

```

74     return 0;
75 }

```

## 4.4 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", 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 * ****Output****
58 * Case 1: 2
59 * ****
60

```

## 4.5 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 }

```

## 5 geometry

### 5.1 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 }

```

### 5.2 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         L[++r] = narrow[i];

```

```

68     if(l < r) P[r-1] = L[r-1].intersection(L[r]);
69 }
70 while(l<r && !onLeft(P[r-1], L[l])) r--;
71 if(r-1 <= 1) return 0;
72 P[r] = L[r].intersection(L[l]);
73
74 int m=0;
75 for(int i=1; i<=r; i++) {
76     poly[m++] = P[i];
77 }
78
79 return m;
80 }
81 }
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
102 int main() {
103     while(~scanf("%d", &n) && n) {
104         for(int i=0; i<n; i++) {
105             double x, y;
106             scanf("%lf%lf", &x, &y);
107             pt[i] = {x, y};
108         }
109         for(int i=0; i<n; i++) {
110             vec[i] = pt[(i+1)%n] - pt[i];
111             normal[i] = vec[i].unit_normal_vector();
112         }
113
114         printf("%.6lf\n", bsearch());
115     }
116     return 0;
117 }

```

### 5.3 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 }

```

## 6 DP

### 6.1 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 = (底下i -
6     // 1個抽屜且1個安全且最上面L) + (底下n -
7     // 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] = dp[i - 1][j + 1][1] + dp[i -
11         1][j][0];
12         dp[i][j][1] = dp[i - 1][j - 1][1] + dp[i -
13         1][j - 1][0];
14     }
15 } //答案在 dp[n][s][0] + dp[n][s][1];

```

### 6.2 Deque 最大差距

```

1 /*定義dp[l][r]是l ~ r時與先手最大差異值
2 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 }

```

### 6.3 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

```

3. 對 B 做 LIS
4. 重複的數字在編號轉換時要變成不同的數字，越早出現的數字要越小
5. 如果有數字在 B 裡面而不在 A 裡面，直接忽略這個數字不做轉換即可

### 6.4 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 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 }

```

### 6.5 stringDP

- Edit distance

$S_1$  最少需要經過幾次增、刪或換字變成  $S_2$

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

- Longest Palindromic Subsequence

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

### 6.6 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 }

```

## 6.7 TreeDP reroot

```

1  /*f(0)與f(2)的關係
2  f(2) = f(0) + a - b
3  a = n - b, (subtree(2)以外的節點)
4  b = subtreeSize(2), (subtree(2))
5  所以f(n)是n為root到所有點的距離
6  f(2) = f(0) + n - 2 * subtreeSize(2)
7  這就是快速得到答案的轉移式
8  f(child) = f(parent) + n - 2 * subtreeSize(child)
9  流程
10  1. root = 0去求各項subtreeSize
11  2. 求f(root)
12  3. 以f(0)去求出re-root後的所有f(v), v != 0
13  整體來說
14  暴力解 O(n ^ 2)
15  re-root dp on tree O(n + n + n) -> O(n)*/
16 class Solution {
17 public:
18     vector<int> sumOfDistancesInTree(int n,
        vector<vector<int>>& edges) {
19         this->res.assign(n, 0);
20         G.assign(n + 5, vector<int>());
21         for (vector<int>& edge: edges) {

```

```

22         G[edge[0]].emplace_back(edge[1]);
23         G[edge[1]].emplace_back(edge[0]);
24     }
25     memset(this->visited, 0,
        sizeof(this->visited));
26     this->dfs(0);
27     memset(this->visited, 0,
        sizeof(this->visited));
28     this->res[0] = this->dfs2(0, 0);
29     memset(this->visited, 0,
        sizeof(this->visited));
30     this->dfs3(0, n);
31     return this->res;
32 }
33 private:
34     vector<vector<int>> G;
35     bool visited[30005];
36     int subtreeSize[30005];
37     vector<int> res;
38     //求subtreeSize
39     int dfs(int u) {
40         this->visited[u] = true;
41         for (int v: this->G[u]) {
42             if (!this->visited[v]) {
43                 this->subtreeSize[u] += this->dfs(v);
44             }
45         }
46         //自己
47         this->subtreeSize[u] += 1;
48         return this->subtreeSize[u];
49     }
50     //求res[0], 0到所有點的距離
51     int dfs2(int u, int dis) {
52         this->visited[u] = true;
53         int sum = 0;
54         for (int v: this->G[u]) {
55             if (!visited[v]) {
56                 sum += this->dfs2(v, dis + 1);
57             }
58         }
59         //要加上自己的距離
60         return sum + dis;
61     }
62     //算出所有的res
63     void dfs3(int u, int n) {
64         this->visited[u] = true;
65         for (int v: this->G[u]) {
66             if (!visited[v]) {
67                 this->res[v] = this->res[u] + n - 2 *
                    this->subtreeSize[v];
68                 this->dfs3(v, n);
69             }
70         }
71     }
72 };

```

## 6.8 Weighted LIS

```

1 #define maxn 200005
2 long long dp[maxn];
3 long long height[maxn];
4 long long B[maxn];
5 long long st[maxn << 2];
6 void update(int p, int index, int l, int r, long long
    v) {
7     if (l == r) {
8         st[index] = v;
9         return;
10    }
11    int mid = (l + r) >> 1;
12    if (p <= mid)
13        update(p, (index << 1), l, mid, v);
14    else
15        update(p, (index << 1) + 1, mid + 1, r, v);

```

```
16     st[index] = max(st[index << 1], st[(index << 1) +
17         1]);
18 }
19 long long query(int index, int l, int r, int ql, int
20     qr) {
21     if (ql <= l && r <= qr)
22         return st[index];
23     int mid = (l + r) >> 1;
24     long long res = -1;
25     if (ql <= mid)
26         res = max(res, query(index << 1, l, mid, ql,
27             qr));
28     if (mid < qr)
29         res = max(res, query((index << 1) + 1, mid +
30             1, r, ql, qr));
31     return res;
32 }
33 int main() {
34     int n;
35     scanf("%d", &n);
36     for (int i = 1; i <= n; ++i)
37         scanf("%lld", &height[i]);
38     for (int i = 1; i <= n; ++i)
39         scanf("%lld", &B[i]);
40     long long res = B[1];
41     update(height[1], 1, 1, n, B[1]);
42     for (int i = 2; i <= n; ++i) {
43         long long temp;
44         if (height[i] - 1 >= 1)
45             temp = B[i] + query(1, 1, n, 1, height[i]
46                 - 1);
47         else
48             temp = B[i];
49         update(height[i], 1, 1, n, temp);
50         res = max(res, temp);
51     }
52     printf("%lld\n", res);
53     return 0;
54 }
```