CEOI'2012 Day1, Task: race

In the non-crossing case ($k = 0$), the solution is fairly straightforward using dynamic programming. The only thing to notice is that if we go along the race track, the harbors already visited divide the shore into intervals of harbors, such that once we enter an interval (ie. we visit one of its harbors), we can never leave it. For example if the first four stages of the race track are as in Fig. 1, then the ending of the race track cannot leave the $B - D$ interval.

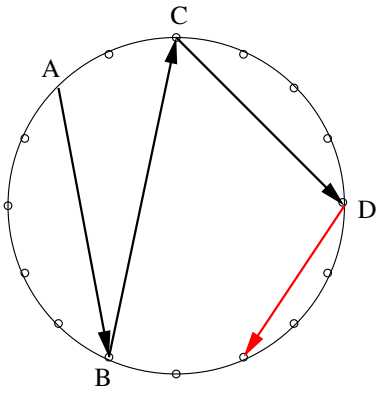

Figure 1:

For harbors A and B, let $ed(A, B)$ be the length of the longest possible track starting from A that stays within the interval $[A, B)$, by which we mean the collection of harbors running in the counterclockwise direction from A to, and not including B. Similarly we define $edc(A, B)$ to be the maximum on the interval $[A, B)_c$ running in the clockwise direction. Keeping the aboves in mind, it is easy to see that

$$ed(A, B) = \max_{C \in [A,B) \exists A-C \text{ route}} \{1 + \max\{ed(C, B), edc(C, A)\}\},$$

defining the maximum to be 0 if there is no such C. A similar formula computes $edc(A, B)$, and one can use these to produce an $O(N(N + E))$ solution, where N denotes the number of harbors and E denotes the number of direct routes. A correct solution to this case was worth 40 points in the competition.

Let us investigate now the case when we allow a crossing on the first stage. It is interesting to note that the optimum is at most 1 greater than the non-crossing optimum: indeed, if we omit the first stage we get a non-crossing race track. However, this observation does not give us a longest track, and it is actually not needed for our solution. Instead, we focus on the intersection. Suppose the track starts at harbor A, the next harbor being B, and later it crosses the $A - B$ stage going from harbor C to harbor D. For fixed A, B, C, D, we really have two cases: the harbors A, C, B, D are in either clockwise or counterclockwise order. For now, suppose they are in clockwise order (the other case works analogously). The race track must start with stage $A - B$, then it is "monotone counterclockwise" in the interval $[B, C]$, reaching C. The next stage is $C - D$, and the ending is an arbitrary non-crossing track either in the interval $[D, A)_c$ or in $[D, B)$ (see Fig. 2). Let $md(B, C)$ be

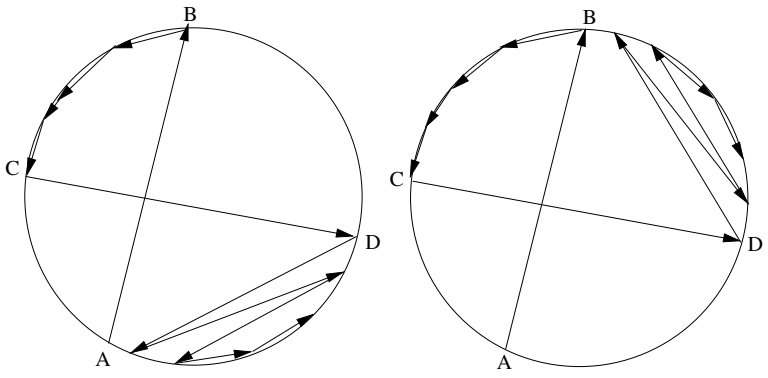

Figure 2:

the length of the longest monotone counterclockwise track from B to C. This function can also be computed for all intervals in $O(N(N+M))$ time, using dynamic programming. Now we see that with A, B, C, D fixed, the longest such race track has length $1 + md(B, C) + 1 + \max\{edc(D, A), ed(D, C)\}$, which can be calculated in constant time. Checking every possible pair of routes $A - B$ and $C - D$, this gives a solution of complexity $O(N^2 + M^2)$, earning 70-75 points on our test data.

For the optimal solution, we need yet another observation. Indeed, suppose we only fix B, C and D, and wish to choose A in the interval (C, D) optimally. If the ending of the race track is in the interval $[D, B)$, then the choice of A does not matter. On the other hand, if the ending is in $[D, A)_c$, then we want to maximize $edc(D, A)$. But this is obviously monotone: the farther away A is from D, the longest ending is possible. So we do not actually have to check the above formula for each A in the interval, it is enough to check it for the harbor A closest to C such that there is an $A - B$ route. Now for finding this A, we do not even need D to be fixed, so we can start by fixing B and C, then find the only interesting A, and then go through the possible D's. These ideas give an $O(N(N+M))$ algorithm, an efficient implementation of which passes all test cases.

Implementation

```
1  program boatrace;
2  type int=longint;
3  const
4      MAXN=500;
5  var
6      a,aa,md,mdc,ed,edc:array[0..MAXN,0..MAXN] of int;
7      adj:array[0..MAXN,0..MAXN] of boolean;
8      last,b,bb:array[0..MAXN] of int;
9      n,kk,max,maxi:int;
10
11 procedure inp;
12 var
13     i,j,x:int;
14 begin
15     readln(n,kk);
16     for i:=0 to n-1 do
17     for j:=0 to n-1 do adj[i,j]:=false;
18     for i:=0 to n-1 do begin
19         read(x);
20         while x>0 do begin
21             adj[i,x-1]:=true;
22             read(x);
23         end;
24         readln;
25     end;
26
27     for i:=0 to n-1 do begin
28         b[i]:=0; bb[i]:=0;
29     end;
30
31     for i:=0 to n-1 do begin
32         j:=(i+1) mod n;
33         while i<>j do begin
34             if adj[i,j] then begin
35                 inc(b[i]);
36                 a[i,b[i]]:=j; md[i,j]:=1; mdc[i,j]:=1;
37             end;
38             if adj[j,i] then begin
39                 inc(bb[i]); aa[i,bb[i]]:=j;
40             end;
41             inc(j);
42             if j=n then j:=0;
43         end;
```

```
44      end;
45  end;//inp
46
47  procedure calc;
48  var
49      i,j,t,h:int;
50      x,y:int;
51  begin
52      for i:=0 to n-1 do begin
53          md[i,i]:=0; mdc[i,i]:=0; //ed, edc:open interval
54          ed[i,i]:=0; edc[i,i]:=0; //md, mdc:closed interval
55      end;
56
57      for t:=1 to n do begin
58          for i:=0 to n-1 do begin
59              for j:=1 to b[i] do begin
60                  y:=a[i,j];
61                  x:=(i+t) mod n; h:=(y+n-i) mod n;
62                  if h<t then begin
63                      if (md[y,x]>0) and (md[y,x]>=md[i,x]) then
64                          md[i,x]:=md[y,x]+1;
65                      if ed[y,x]>=ed[i,x] then ed[i,x]:=ed[y,x]+1;
66                      if edc[y,i] >=ed[i,x] then ed[i,x]:=edc[y,i]+1;
67                  end;
68                  x:=(i+n-t) mod n; h:=(i+n-y) mod n;
69                  if h<t then begin
70                      if (mdc[y,x]>0) and (mdc[y,x]>=mdc[i,x]) then
71                          mdc[i,x]:=mdc[y,x]+1;
72                      if edc[y,x]>=edc[i,x] then edc[i,x]:=edc[y,x]+1;
73                      if ed[y,i] >=edc[i,x] then edc[i,x]:=ed[y,i]+1;
74                  end;
75              end;
76          end;
77      end;
78      max:=0;
79      for i:=0 to n-1 do if edc[i,i]>max then begin    //edc[i,i+1]
80          max:=edc[i,i]; maxi:=i;
81      end;
82      for i:=0 to n-1 do if ed[i,i]>max then begin    //ed[i,i-1]
83          max:=ed[i,i]; maxi:=i;
84      end;
85  end;//calc
86
87  procedure opt;
88  var
89      i,j,k:int;
90      x,y,z,tmp:int;
91  begin
92      for i:=0 to n-1 do begin
93  //case 1: iyzx positive (with the above notation C=i, A=y=last[x], D=z, B=x)
94          for j:=2 to n-1 do begin          //calculate A=last[x]
95              x:=(i+j) mod n; k:=1;
96              while (k<=bb[x]) and ((aa[x,k]+n-i-1) mod n>j-1) do inc(k);
97              if k<=bb[x] then last[x]:=aa[x,k] else last[x]:=-1;
98          end;
99
100         for j:=2 to n-2 do if adj[i,(i+j) mod n] then begin
101             z:=(i+j) mod n;
102             for k:=j+1 to n-1 do begin
```

```
103                 x:=(i+k) mod n;
104                 if (last[x]>-1) and ((n+last[x]-i) mod n<j) and (md[x,i]>0)
105                 then begin
106                     if edc[z,last[x]]>ed[z,x]
107                         then tmp:=1+md[x,i]+1+edc[z,last[x]]
108                         else tmp:=1+md[x,i]+1+ed[z,x];
109                     if tmp>max then begin
110                         max:=tmp; maxi:=last[x];
111                     end;
112                 end;
113             end;
114         end;
115 //case 2: iyzx negative
116         for j:=1 to n-2 do begin
117             x:=(i+j) mod n; k:=1;
118             while (k<=bb[x]) and ((aa[x,k]+n-i) mod n>j) do inc(k);
119                 if k>1 then last[x]:=aa[x,k-1] else last[x]:=-1;
120         end;
121         for j:=2 to n-2 do if adj[i,(i+j) mod n] then begin
122             z:=(i+j) mod n;
123             for k:=1 to j-1 do begin
124                 x:=(i+k) mod n;
125                 if (last[x]>-1) and ((n+last[x]-i) mod n>j) and (mdc[x,i]>0)
126                 then begin
127                     if ed[z,last[x]]>edc[z,x]
128                         then tmp:=1+mdc[x,i]+1+ed[z,last[x]]
129                         else tmp:=1+mdc[x,i]+1+edc[z,x];
130                     if tmp>max then begin
131                         max:=tmp; maxi:=last[x];
132                     end;
133                 end;
134             end;
135         end;
136     end;
137 end;//opt
138
139 begin//prog
140     inp;
141     calc;
142     if kk=1 then opt;
143     writeln(max);
144     writeln(maxi+1);
145 end.
```