D. Garden Decorations

Problem Name	Garden Decorations
Time Limit	7 seconds
Memory Limit	1 gigabyte

Elke dag als ze naar school loopt en op de terugweg naar huis, wandelt Detje door een straat met N huizen, genummerd van 0 tot $N-1$. Op dit moment wordt huis i bewoond door persoon i. De bewoners hebben besloten voor de afwisseling met elkaar van huis te ruilen. De persoon die in huis i gaat wonen is persoon a_i (die nu woont in huis a_i).

Elk huis heeft een tuin met daarin een standbeeld dat een vogel voorstelt. De standbeelden hebben twee mogelijke toestanden: ze hebben hun vleugels ofwel *open* (alsof de vogel vliegt) ofwel *gesloten* (alsof de vogel op de grond staat). De bewoners hebben een hele duidelijke voorkeur voor hoe hun vogelstandbeeld eruit moet zien. Op dit moment is de vogel voor huis i in de toestand die de voorkeur heeft van bewoner i. Bewoners weigeren naar hun nieuwe huis te verhuizen tot het standbeeld bij hun nieuwe huis in hun voorkeurstoestand is. Detje wil ze helpen de vogels zó neer te zetten dat ze kunnen verhuizen.

Ze doet hiervoor het volgende: elke keer dat ze door de straat loopt (op weg naar school óf op weg naar huis) bekijkt ze één voor één de vogels, en eventueel past ze de toestand van sommige beelden aan (door hun vleugels open of dicht te doen). Doordat ze het op school en thuis heel druk heeft, **herinnert ze zich niet wat de toestanden van de vogels waren die ze gezien heeft tijdens haar eerdere wandelingen** Gelukkig heeft ze de lijst $a_0, a_1, ..., a_{N-1}$ opgeschreven, dus ze weet welke bewoner waarheen verhuist.

Help Detje een strategie te ontwikkelen waarmee ze kan bepalen welke vogels ze moet manipuleren om de beelden aan te passen aan de voorkeuren van de bewoners. Ze kan maximaal 60 keer door de straat lopen, maar om een hogere score te behalen, moet ze minder vaak door de straat wandelen.

Implementation

Dit is een multirun probleem. Dat betekent dat je programma meerdere keren zal worden uitgevoerd.

Bij iedere run moet je eerst een regel inlezen met twee gehele getallen, w en N, het nummer van de wandeling en het aantal huizen. De eerste keer dat je programma wordt uitgevoerd, geldt dat $w = 0$, de tweede keer $w = 1$, en zo verder (meer details worden hieronder toegelicht).

Op de tweede regel van de invoer staan N gehele getallen, $a_0, a_1, ..., a_{N-1}$. De betekenis hiervan is dat de persoon die in huis i gaat wonen, nu in huis a_i woont. De getallen a_i vormen een *permutatie*: dat wil zeggen, elk van de getallen 0 tot en met $N - 1$ komt precies één keer voor in de lijst a_is. Merk op dat een bewoner ervoor kan kiezen niet te verhuizen; dat wil zeggen dat $a_i = i$ toegestaan is.

De bewoners ruilen maar één keer van huis. Dat betekent dat voor een vaste test case, de waarde van N en de lijst van a_is gelijk zal zijn voor alle runs van je programma.

Eerste Run.

Bij de eerste keer dat je programma wordt uitgevoerd geldt $w = 0$. In deze run moet je simpelweg één enkel geheel getal W ($0 \leq W \leq 60$) uitvoeren, het aantal keren dat je wilt dat Detje langs de huizen loopt. Je programma moet dan stoppen. Hierna wordt je programma nog W keer uitgevoerd.

Volgende Runs.

Bij de volgende run van je programma geldt $w = 1$; in de run daarna $w = 2$; en zo door tot de laatste run waar $w = W$.

Nadat je w, N and the $a_0, a_1, ..., a_{N-1}$ hebt gelezen, begint Detje door de straat te lopen.

- Als w oneven is, loopt Detje van haar huis naar school, en loopt ze langs de huizen in de volgorde $0, 1, ..., N - 1$.

  Je programma moet nu een regel lezen met b_0, ofwel gelijk aan 0 (gesloten) ofwel gelijk aan 1 (open), de huidige toestand van het beeld voor huis 0. Nadat je b_0, gelezen hebt, moet je een regel uitvoeren met ofwel een 0 ofwel een 1, de nieuwe waarde die je b_0 wilt geven.

  Daarna moet je programma een regel lezen met b_1, de toestand van het beeld voor huis 1; en de nieuwe waarde van b_1 uitvoeren. Dit gaat zo door voor elk van de N huizen. Als Detje het laatste huis gehad heeft (dat is, je hebt b_{N-1} in- en uitgevoerd) **moet je programma stoppen**.

  Let op: je programma kan de volgende waarde b_{i+1} pas lezen nadat je de waarde van b_i hebt uitgevoerd.

- Als w even is, wandelt Detje van school naar huis, en loopt de huizen voorbij in omgekeerde volgorde, $N - 1, N - 2, ...0$. Je begint dus de in- en uitvoer begint met b_{N-1}, dan b_{N-2}, en zo verder tot b_0.

Als $w = 1$, geven de invoerwaarden $b_0, b_1, ..., b_{N-1}$ de oorspronkelijk toestand van de vogelstandbeelden (dit is ook de voorkeur van de bewoners). Als $w > 1$, zijn de invoerwaarden $b_0, b_1, ..., b_{N-1}$ voor je programma gelijk aan de waarden die de vorige run van je programma ze gegeven heeft.

Aan het eind, na de laatste run van je programma, moet de waarde van b_i gelijk zijn aan de oorspronkelijke waarde van b_{a_i} voor elke i, anders krijg je als resultaat Wrong Answer.

Details.

Als de *som* van de running times van de $W + 1$ afzonderlijke runs van je programma groter is dan de tijdlimiet, wordt je programma beoordeeld als Time Limit Exceeded.

Let op dat je standard output flusht na elke regel die je uitvoert, omdat je programma anders beoordeeld kan worden alsTime Limit Exceeded. In Python, gebeurt dit automatisch als je `input()` gebruikt om regels te lezen. In C++, zorgt `cout << endl;` ervoor dat je flusht naast dat je een newline print; als je printf gebruikt, gebruik dan `fflush(stdout)`.

Constraints and Scoring

- $2 \leq N \leq 500$.
- Je mag maximaal $W \leq 60$ beurten gebruiken.

Je oplossing wordt getest met een aantal testgroepen, die elk een aantal punten waard zijn. Iedere testgroep bevat een aantal test cases. Om de punten te krijgen voor een testgroep moet je alle test cases in de groep oplossen. Your solution will be tested on a set of test groups, each worth a number of points.

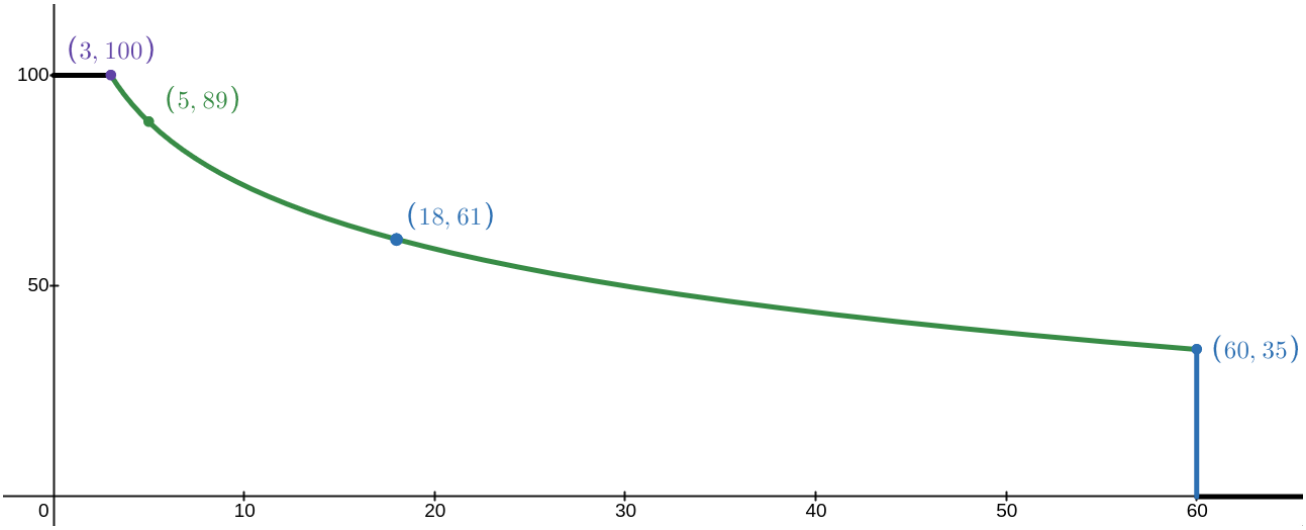
Group	Max score	Limits
1	10	$N = 2$
2	24	$N \leq 15$
3	9	$a_i = N - 1 - i$
4	13	$a_i = (i + 1) \bmod N$
5	13	$a_i = (i - 1) \bmod N$
6	31	Geen aanvullende voorwaarden

Voor iedere testgroep die je programma correct oplost, krijg je een score volgens deze formule:

$$\text{score} = S_g \cdot \left(1 - \tfrac{1}{2} \log_{10}(\max(W_g, 3)/3)\right).$$

Hierin is S_g de maximale score voor de testgroep, en W_g is de maximale waarde van W gebruikt voor een test case in de testgroep. Je score voor elke testgroep wordt afgerond naar het dichtstbijzijnde gehele getal.

De grafiek hieronder toont het aantal punten, als een functie van W, dat je programma krijgt als het alle testgroepen oplost met dezelfde waarde van W. In het bijzonder: om een score van 100 points te behalen op dit probleem moet je elke test case oplossen met $W \leq 3$.



Testing Tool

Om het testen van je programma makkelijker te maken, is er een simpele tool beschikbaar die je kunt downloaden. Zie “attachments” onderaan de Kattis problem page. Gebruik van de tool is niet verplicht. Let op: het officiële beoordelingsprogramma van Kattis is anders dan de testtool.

Om de tool te gebruiken, moet je een input file maken, bijvoorbeeld “sample1.in”, die moet beginnen met een getal N gevolgd door N getallen die de permutatie vastleggeb, en nog een regel met numbers specifying the permutation, and another line with N bits (0 of 1) die de originele toestanden van de vogels beschrijven. Bijvoorbeeld:

```
6
1 2 0 4 3 5
1 1 0 0 1 0
```

Voor Python programmas, zeg `solution.py` (normaal gerund als as `pypy3 solution.py`):

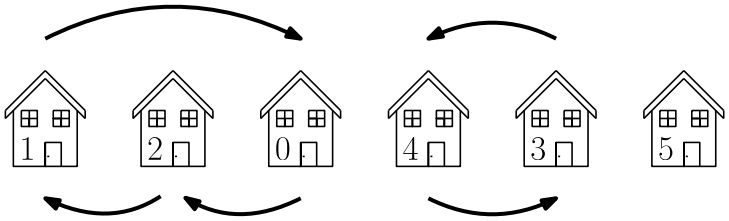
```
python3 testing_tool.py pypy3 solution.py < sample1.in
```

Voor C++ programs, compileer eerst (bijvoorbeeld met `g++ -g -O2 -std=gnu++20 -static solution.cpp -o solution.out`) en run dan:

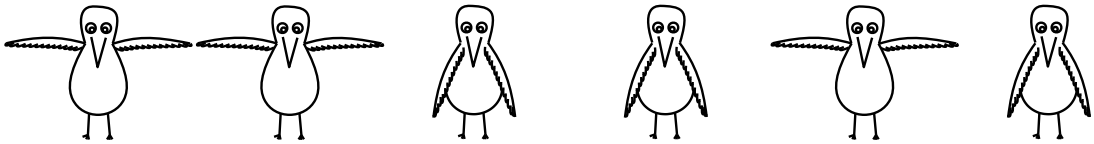
```
python3 testing_tool.py ./solution.out < sample1.in
```

Example

In het voorbeeld hebben we de volgende permutatie van mensen in de huizen:

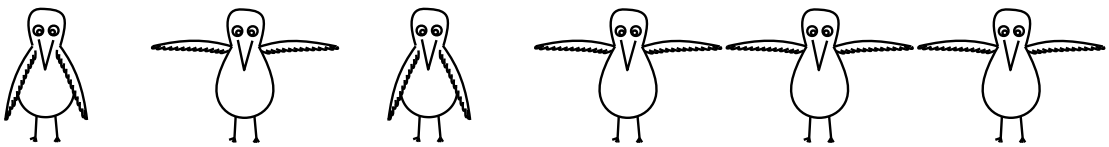


De eerste keer dat het voorbeeldprogramma runt (met $w = 0$), voert het $W = 2$ uit. Dit betekent dat Detje twee keer door de straat loopt (en het programma wordt nog twee keer uitgevoerd). Voor de eerste wandeling zien de vogels in de tuinen er zo uit:



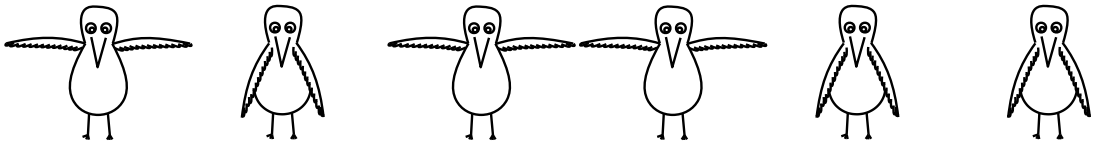
Het programma wordt dan gerund met $w = 1$: dit geeft Detje's eerste wandeling aan. Ze loopt de vogels één voor één langs. Ze begint links, en verandert mogelijk hun toestanden. Het voorbeeldprogramma moet de toestand van de i-de vogel uitvoeren voor we de $(i + 1)$-de vogel zien.

Als Detje op school is aangekomen, is de toestand van de vogels



In de laatste run van het programma (met $w = 2$) wandelt Detje terug van school naar huis. Denk eraan dat ze nu van rechts naar links langs de vogels loopt, en ze in deze volgorde verwerkt! Dit betekent dat ze de toestand van vogel i moet vastleggen voor ze vogel $(i - 1)$ ziet.

Als ze haar wandeling afrond, zien de vogels er zo uit:



Dit is inderdaad de correcte situatie. Bijvoorbeeld: vogelstandbeeld 3 (dat is het vierde beeld van links) is open (nu geldt $b_3 = 1$), en dat is correct want 4 gaat daarheen verhuizen ($a_3 = 4$) en zij had aan het begin een open vogelstandbeeld (oorspronkelijk gold $b_4 = 1$).

grader output	your output
0 6	
1 2 0 4 3 5	
	2

grader output	your output
1 6	
1 2 0 4 3 5	
1	
	0
1	
	1
0	
	0
0	
	1
1	
	1
0	
	1

grader output	your output
2 6	
1 2 0 4 3 5	
1	
	0
1	
	0
1	
	1
0	
	1
1	
	0
0	
	1