D. Gartendekorationen

Aufgabenname	Garden Decorations
Time Limit	7 Sekunden
Memory Limit	1 Gigabyte

Jeden Tag geht Detje auf dem Weg zur Schule und zurück nach Hause eine Strasse mit N Häusern entlang, die von 0 bis $N-1$ nummeriert sind. Derzeit wird das Haus i von der Person i bewohnt. Um einen Tapetenwechsel zu bekommen, haben die Bewohnerinnen beschlossen, die Häuser miteinander zu tauschen. Die Person, die in das Haus i einziehen wird, ist Person a_i (die derzeit in Haus a_i wohnt).

Jedes Haus hat eine Vogelstatue im Garten. Die Statuen haben zwei mögliche Zustände, entweder mit *offenen* Flügeln (als ob der Vogel fliegt) oder mit *geschlossenen* (als ob er auf dem Boden steht). Die Bewohnerinnen haben sehr starke Vorlieben, wie ihre Vogelstatuen aussehen sollen. Aktuell befindet sich der Vogel vor dem Haus i im bevorzugten Zustand von Bewohnerin i. Die Bewohnerinnen weigern sich, in ihr neues Haus umzuziehen, solange die Statue in ihrem neuen Garten nicht so aussieht wie in ihrem alten. Detje will ihnen helfen, die Vogelstatuen so zu arrangieren, dass sie umziehen können.

Dazu macht sie Folgendes: Immer wenn sie die Strasse entlanggeht (entweder auf dem Weg zur Schule oder nach Hause), beobachtet sie die Vögel, an denen sie vorbeikommt, einen nach dem anderen und verändert möglicherweise einige der Statuen (indem sie ihre Flügel öffnet oder schliesst). Da sie in der Schule und zu Hause sehr beschäftigt ist, **erinnert sie sich nicht an den Zustand der Vögel, die sie auf ihren vorherigen Spaziergängen gesehen hat**. Glücklicherweise hat sie die Liste $a_0, a_1, \ldots, a_{N-1}$ aufgeschrieben, so dass sie weiss, welche Bewohnerin wohin zieht.

Hilf Detje dabei, eine Strategie zu entwerfen, die ihr sagt, welche Vögel sie manipulieren muss, um die Statuen an die Vorlieben der Bewohnerinnen anzupassen. Sie kann höchstens 60 Mal die Strasse entlanggehen, aber um eine höhere Punktzahl zu erreichen, sollte sie die Strasse weniger oft entlanggehen.

Implementierung

Dies ist ein Multirun-Problem, d.h. das Programm wird mehrere Male ausgeführt.

Bei jedem Durchlauf sollte zunächst eine Zeile mit zwei ganzen Zahlen w und N, dem Index des Spaziergangs und der Anzahl der Häuser, eingelesen werden. Im ersten Durchlauf deines Programms ist $w = 0$, im zweiten $w = 1$, und so weiter (weitere Details werden weiter unten erklärt).

In der zweiten Eingabezeile stehen N ganze Zahlen $a_0, a_1, ..., a_{N-1}$, was bedeutet, dass die Person, die in das Haus i einziehen wird, derzeit im Haus a_i wohnt. Die a_is bilden eine *Permutation*, d.h. jede Zahl von 0 bis $N-1$ erscheint genau einmal in der Liste der a_is. Beachte, dass eine Bewohnerin sich entscheiden kann, nicht umzuziehen; das heisst, $a_i = i$ ist erlaubt.

Die Bewohnerinnen wechseln nur einmal das Haus. Das bedeutet, dass für einen festen Testfall der Wert von N und die Liste der a_is für alle Durchläufe des Programms gleich sein werden.

Erster Durchlauf.

Bei der ersten Ausführung deines Programms ist $w = 0$. Bei diesem Durchlauf solltest du einfach eine einzelne ganze Zahl W ($0 \leq W \leq 60$) ausgeben, die Anzahl, wie oft Detje an den Häusern vorbeilaufen soll. Danach sollte sich dein Programm beenden. Anschliessend wird dein Programm noch W weitere Male ausgeführt.

Nachfolgende Durchläufe.

Im nächsten Durchlauf deines Programms ist $w = 1$; im übernächsten $w = 2$; und so weiter bis zum letzten Durchlauf, bei dem $w = W$.

Nachdem du w, N und die $a_0, a_1, ..., a_{N-1}$ abgelesen hast, beginnt Detje, die Strasse entlangzulaufen.

- Wenn w ungerade ist, geht Detje von ihrem Haus zur Schule, und sie wird an den Häusern in der Reihenfolge $0, 1, ..., N-1$ vorbeigehen.

  Dein Programm sollte nun eine Zeile mit b_0 einlesen, entweder 0 (geschlossen) oder 1 (offen), den aktuellen Zustand der Statue vor dem Haus 0. Nachdem du b_0 eingelesen hast, solltest du eine Zeile mit entweder 0 oder 1 ausgeben, dem neuen Wert, auf den du b_0 setzen willst.

  Dann sollte dein Programm eine Zeile mit b_1 einlesen, den Zustand der Statue vor dem Haus 1; und den neuen Wert von b_1 ausgeben. Dies wird für jedes der N Häuser fortgesetzt. Nachdem Dejte das letzte Haus passiert hast (d.h. sie hat b_{N-1} gelesen und geschrieben), **sollte sich dein Programm beenden**.

  Beachte, dass dein Programm den nächsten Wert b_{i+1} erst lesen kann, nachdem du den Wert von b_i geschrieben hast.

- Wenn w gerade ist, geht Detje von der Schule nach Hause, und sie wird an den Häusern in der umgekehrten Reihenfolge $N-1, N-2, ..., 0$.

  Der Vorgang ist derselbe wie bei w ungerade, nur dass man mit dem Einlesen und Ausgeben von b_{N-1} beginnt, dann b_{N-2}, und so weiter bis b_0. Das bedeutet, dass du mit dem Lesen und Schreiben von b_{N-1} beginnst, dann b_{N-2}, u.s.w bis b_0 liest und schreibst.

Wenn $w = 1$, sind die Eingabewerte $b_0, b_1, ..., b_{N-1}$ der ursprüngliche Zustand der Vogelstatuen. Wenn $w > 1$ ist, sind die Eingabewerte $b_0, b_1, ..., b_{N-1}$ für dein Programm so, wie sie beim letzten Durchlauf deines Programms gesetzt wurden.

Am Ende, nach dem letzten Durchlauf deines Programms, muss der Wert von b_i für alle i gleich dem ursprünglichen Wert von b_{a_i} sein, andernfalls erhältst du das Ergebnis „Wrong Answer“.

Details.

Wenn die *Summe* der Ausführungszeiten der $W + 1$ einzelnen Durchläufe deines Programms das Zeitlimit überschreitet, wird deine Einsendung als „Time Limit Exceeded“ gewertet.

Vergewisser dich, dass du die Standardausgabe nach dem Drucken jeder Zeile flushst, sonst könnte dein Programm als „Time Limit Exceeded“ gewertet werden. In Python geschieht dies automatisch, solange man `input()` verwendet, um Zeilen zu lesen. In C++ wird mit `cout << endl;` zusätzlich zur Ausgabe eines Zeilenumbruchs ein Flush durchgeführt; wenn man printf verwendet, sollte man `fflush(stdout)` verwenden.

Einschränkungen und Bewertung

- $2 \leq N \leq 500$.
- Du darfst höchstens $W \leq 60$ Durchläufe brauchen.

Deine Lösung wird an einer Reihe von Testgruppen getestet, von denen jede eine bestimmte Anzahl von Punkten wert ist. Jede Testgruppe enthält eine Reihe von Testfällen. Um die Punkte für eine Testgruppe zu erhalten, musst du alle Testfälle der Testgruppe lösen.

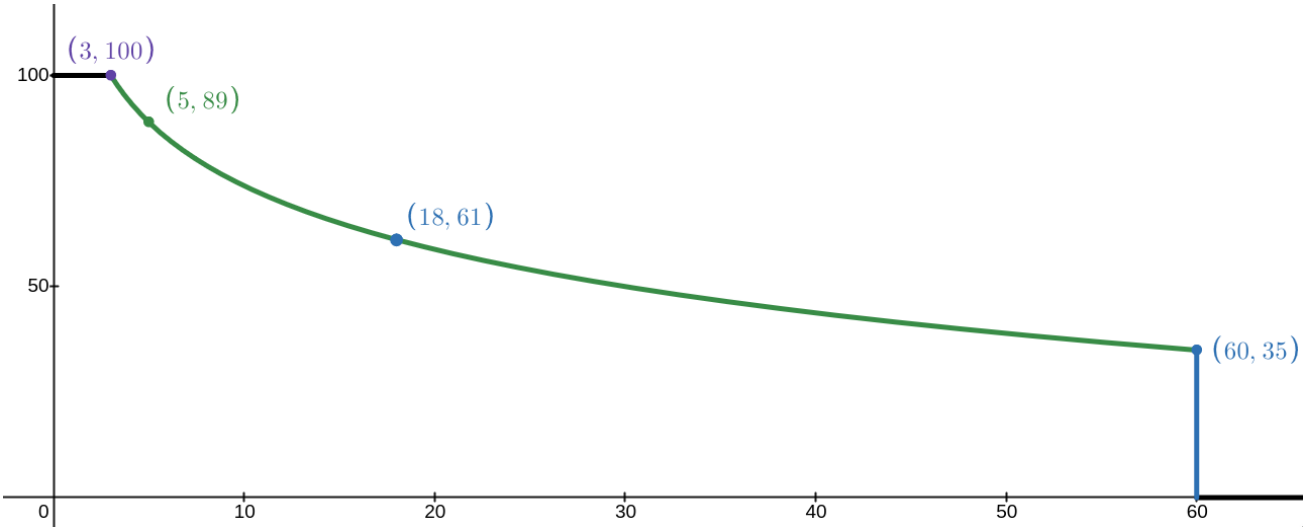
Gruppe	Maximale Punktzahl	Limits
1	10	$N = 2$
2	24	$N \leq 15$
3	9	$a_i = N - 1 - i$
4	13	$a_i = (i + 1) \bmod N$
5	13	$a_i = (i - 1) \bmod N$
6	31	Keine weiteren Einschränkungen

Für jede Testgruppe, die dein Programm richtig löst, erhältst du eine Punktzahl, die auf der folgenden Formel basiert:

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

wobei S_g die maximale Punktzahl für die Testgruppe und W_g der maximale Wert von W ist der für jeden Testfall in der Testgruppe verwendet wird. Die Punktzahl für jede Testgruppe wird auf die nächste ganze Zahl gerundet.

Die folgende Abbildung zeigt die Anzahl der Punkte als Funktion von W, die dein Programm erhält, wenn es alle Testgruppen mit demselben Wert von W löst. Um bei diesem Problem 100 Punkte zu erreichen, musst du also jeden Testfall mit Wle 3$ lösen.



Test-Tool

Um das Testen deiner Lösung zu erleichtern, stellen wir dir ein einfaches Tool zur Verfügung, das du herunterladen kannst. Du findest es unter „attachments“ am Ende der Kattis-Problemseite. Die Verwendung des Tools ist optional. Beachte, dass der offizielle Grader auf Kattis sich von dem Testtool unterscheidet.

Um das Tool zu verwenden, erstelle eine Eingabedatei, z.B. „sample1.in“, die mit einer Zahl N beginnen sollte, gefolgt von einer Zeile mit N Zahlen, die die Permutation angeben, und einer weiteren Zeile mit N Bits (0 oder 1), die die Anfangszustände der Vögel angeben. Zum Beispiel:

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

Bei Python-Programmen sagen wir `solution.py` (normalerweise als `pypy3 solution.py` ausgeführt):

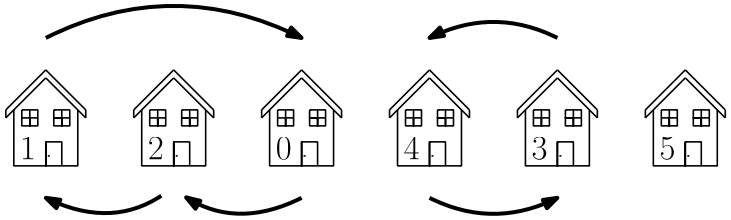
```
python3 testing_tool.py pypy3 solution.py < sample1.in
```

Bei C++-Programmen kompiliere es zuerst (z.B. mit `g++ -g -O2 -std=gnu++20 -static solution.cpp -o solution.out`) und dann führe aus:

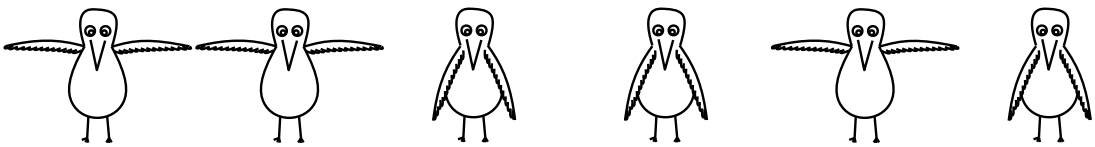
```
python3 testing_tool.py ./solution.out < sample1.in
```

Beispiel

In dem Beispiel haben wir die folgende Permutation der Personen in den Häusern:

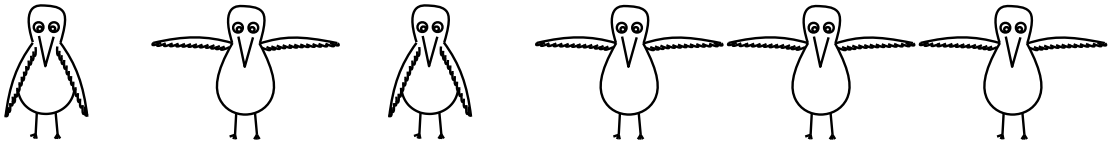


Wenn das Beispielprogramm zum ersten Mal ausgeführt wird (mit $w = 0$), gibt es $W = 2$ aus, was bedeutet, dass Detje zweimal die Strasse entlanggeht (und das Programm noch zwei weitere Male ausgeführt wird). Vor dem ersten Spaziergang sehen die Vögel in den Gärten wie folgt aus:



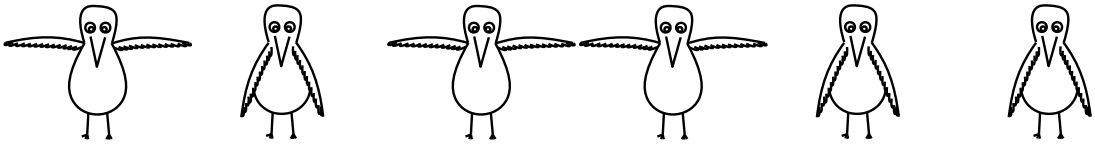
Dann wird das Programm mit $w = 1$ durchlaufen, was den ersten Gang von Detje anzeigt. Sie geht die Vögel einen nach dem anderen durch, beginnend auf der linken Seite, und ändert eventuell deren Zustand. Detje muss den Zustand des i-ten Vogels bestimmen, bevor sie den $(i + 1)$-ten Vogel sieht.

Nachdem Detje in der Schule angekommen ist, ist der Zustand der Vögel:



Im letzten Durchlauf des Programms (mit $w = 2$), geht Detje von der Schule nach Hause. Denk daran, dass sie in diesem Fall die Vögel von rechts nach links durchläuft und sie in dieser Reihenfolge verarbeitet! Das bedeutet, dass sie den Zustand des i-ten Vogels bestimmen muss, bevor sie den $(i - 1)$-ten Vogel sieht.

Nachdem sie ihren Spaziergang beendet hat, sehen die Vögel nun wie folgt aus:



Dies ist in der Tat die richtige Konfiguration. Zum Beispiel ist die Vogelstatue 3 (d.h. die vierte von links) offen (jetzt $b_3 = 1$), was korrekt ist, da Person 4 dorthin ziehen wird ($a_3 = 4$) und sie ursprünglich eine offene Vogelstatue hatte (ursprünglich $b_4 = 1$).

Graderausgabe	Deine Ausgabe
0 6	
1 2 0 4 3 5	
	2

Graderausgabe	Deine Ausgabe
1 6	
1 2 0 4 3 5	
1	
	0
1	
	1
0	
	0
0	
	1
1	
	1
0	
	1

Graderausgabe	Deine Ausgabe
2 6	
1 2 0 4 3 5	
1	
	0
1	
	0
1	
	1
0	
	1
1	
	0
0	
	1