D. Census (census)

الحد الزمني: 1 ثواني
حد الذاكرة: 128MiB

معلومة مش معروفة أوي عن مدينة Cesenatico إنها مقر لجمعية سرية متكونة من N باحثات وعالمات كمبيوتر. الجمعية دي سرية جداً؛ مفيش ولا عضوة تعرف التانية. كل عضوة ليها ID مميز: رقم صحيح مش سالب I.

الطريقة الوحيدة للتواصل بين العضوات هي طريقة غير مباشرة، عن طريق أرقام مكتوبة بالطباشير في أماكن مختلفة في المدينة. كل 100 سنة، الجمعية بتعمل إحصاء عشان تعد العضوات بتوعها. بعد ما الإحصاء يخلص، كل عضوة لازم تكون عارفة العدد الإجمالي لعضوات الجمعية.

الإحصاء بياخد كذا يوم. كل يوم، كل عضوة لسه مشاركة في العملية هتختار وتعمل حركة واحدة بس: إنها تقرأ (read)، أو تكتب (write)، أو توقف مشاركة (stop).

- لو العضوة اختارت تقرأ (read)، بتختار مكان P. بالنهار، بتروح للمكان P وتقرأ الرقم المكتوب هناك.
- لو العضوة اختارت تكتب (write)، بتختار مكان P ورقم V. بليل متأخر، بتروح للمكان P وتغير الرقم اللي كان مكتوب هناك لـ V. وبما إن الدنيا ضلمة، مش هتقدر تقرأ الرقم القديم قبل ما تكتب الجديد.
- لو العضوة اختارت توقف (stop)، مش هتعمل أي حاجة تاني في الأيام اللي جاية.

لو عضوة شافت التانية وهي بتكتب رقم، ممكن تتعرف عليها. عشان كده، ممنوع منعاً باتاً إن عضوتين أو أكتر يختاروا يكتبوا في نفس المكان في نفس اليوم. (مفيش المنع ده على القراية، لأنها ممكن تتعمل في السر.)

لو عضوة أو أكتر قرأوا من مكان فيه عضوة تانية عايزة تكتب فيه في نفس اليوم، كل القرايات بتحصل قبل الكتابة.

إزاي الجمعية المفروض تخطط لعملية الإحصاء عشان تقلل عدد الأيام لحد ما كل واحدة تعرف العدد الصح للعضوات؟

implementation

⇨ دي مسألة تفاعلية (interactive problem)، واللي فيها عدد مش معروف من الـ instances (من $1 \le N \le 100$) من الكود بتاعك هيشتغلوا في نفس الوقت. كل instance بيمثل عضوة في الجمعية.

فيه 10^{18} مكان. رقم المكان P لازم يكون $0 \le P < 10^{18}$. في الأول، القيمة المكتوبة في كل الأماكن هي $V = 0$.

القيمة الجديدة V اللي هتتكتب في أي مكان لازم دايماً تكون رقم صحيح بحيث إن $0 \le V \le 10^9$. في أغلب الـ subtasks، V ممكن يكون بس 0 أو 1. بص على جزء الـ Scoring عشان تفاصيل أكتر.

لما الـ instance بتاع الكود بتاعك يشتغل، لازم أول حاجة يقرأ سطر فيه رقمين صحيحين، I و M ($0 \le I \le M-1$): الـ ID المميز بتاع عضوة الجمعية اللي الـ instance ده بيمثلها والعدد الإجمالي للـ IDs المتاحة. في كل test case، كل الـ instances هياخدوا نفس القيمة M وقيم مختلفة لـ I. خد بالك إن ممكن يكون فيه IDs مش متوزعة على أي عضوة.

بعد كده، لكل يوم في عملية الإحصاء، الكود بتاعك لازم يختار الحركة اللي عايز يعملها ويطبع سطر على الأساس ده:

الحركة	المعنى
`P r`	اقرأ من المكان P. بعد ما تطبع السطر ده، الكود بتاعك لازم يقرأ سطر فيه القيمة الحالية المكتوبة في P.
`P V w`	اكتب في المكان P القيمة الجديدة V. لو كذا instance كتبوا في نفس الـ P في نفس اليوم، هتاخد نتيجة Not correct. ما عدا في الأمثلة والـ subtask 3، لازم تكتب $0 \le V \le 1$؛ شوف جزء الـ Scoring.

$N!$	**جاوب ووقف:** بلّغ إن فيه N عضوات ووقف مشاركة في الإحصاء. بعد ما تجاوب، **الكود بتاعك لازم يقفل بشكل طبيعي.** (خد بالك إن ممكن instances تانية للكود بتاعك تكمل شغل لأيام إضافية قبل ما يجاوبوا ويقفلوا.)

لو أي instance للكود بتاعك جاوب قيمة غلط لـ N، أو خالف البروتوكول، أو استخدم أكتر من 500 يوم، أو عدى حدود الوقت/الميموري (لكل process)، حلك هيتحسب Not correct لل test case دي.

غير كده، الكود بتاعك هيبقى Correct (Partially) (أو جزئياً صح) على الـ test case وهياخد سكور بناءً على قيمة D: أقصى عدد من الأيام خده أي instance عشان يجاوب. عشان تاخد السكور كامل، لازم تحل كل test case بـ $D \leq 61$ و $V \leq 1$. شوف جزء الـ Scoring عشان التفاصيل.

الـ Flushing. لو مش بتستخدم الـ templates اللي متوفرة، اتأكد إنك تعمل flush لل standard output بعد ما تطبع كل سطر، وإلا ممكن الكود بتاعك يتحسب Not correct. في Python، ده بيحصل تلقائي لو استخدمت `input()` عشان تقرأ السطور. في C++، أمر `cout << endl;` بيعمل flush بالإضافة لإنه بيطبع سطر جديد؛ لو بتستخدم printf، استخدم `fflush(stdout)`.

constraints

- $1 \leq N \leq 100$.
- $1 \leq M \leq 100\,000$.
- تقدر تستخدم كحد أقصى 500 يوم.

scoring

الكود بتاعك هيتختبر على كذا test case متقسمين لـ subtasks. عشان تاخد السكور بتاع أي subtask، لازم تحل كل الـ tests اللي جواه صح.

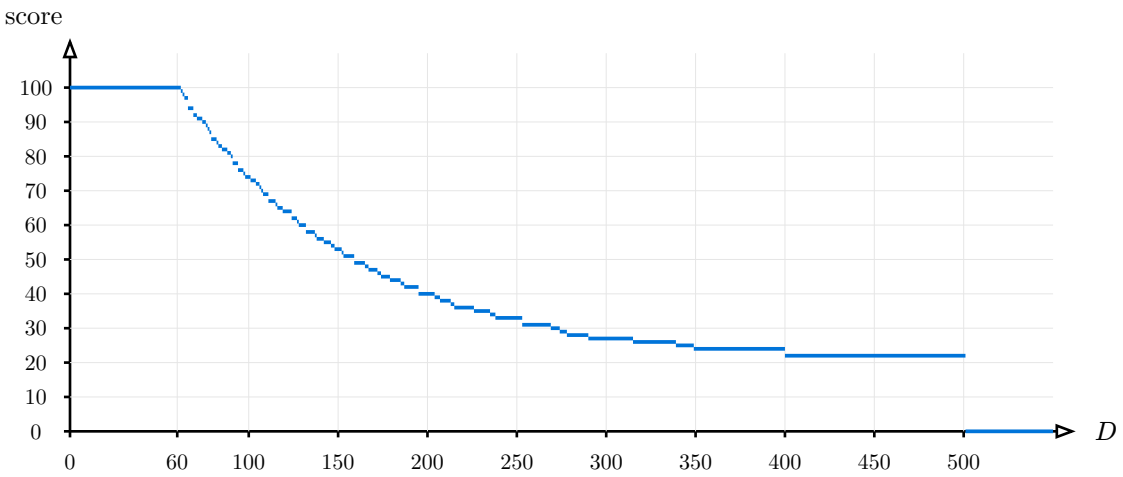
- **المهمة الفرعية 0 [0 نقاط]**: الأمثلة (تقدر تكتب أي رقم صحيح $0 \leq V \leq 1\,000\,000\,000$).
- **المهمة الفرعية 1 [11 نقاط]**: $M \leq 100$، والـ N عضوات ليهم IDs من $0, 1, ..., N-1$.
- **المهمة الفرعية 2 [12 نقاط]**: $1 \leq N \leq 2$.
- **المهمة الفرعية 3 [22 نقاط]**: $M \leq 8000$، وتقدر تكتب أي رقم صحيح $0 \leq V \leq 1\,000\,000\,000$.
- **المهمة الفرعية 4 [55 نقاط]**: مفيش شروط إضافية.

في الـ subtasks 1 و 2 و 4، تقدر بس تكتب $V = 0$ أو $V = 1$ في كل حركة كتابة.

بفرض إن X_s هي أقصى نقط لل subtask s (المعروضة فوق)، و D_s هي أكبر عدد أيام أي من برامجك بتستخدمه في test جوه الـ subtask s. ساعتها:

$$\text{score}_s = \begin{cases} X_s & \text{if} \quad D_s \leq 61 \\ X_s \cdot \left(0.2 + 0.8 \cdot 1.01^{(60-D_s)}\right) & \text{if} \quad 61 < D_s \leq 500 \\ 0 & \text{if} \quad 500 < D_s. \end{cases}$$

قيمة الـ score_s بتتقرب لأقرب رقم صحيح لكل subtask، والسكور الإجمالي بتاعك هو مجموعهم. عشان تاخد السكور الكامل للمسألة، لازم $D \leq 61$ و $V \leq 1$ في كل test case.



شكل 1: إجمالي السكور، على افتراض إن كل subtask بيتحل بنفس الحد الأقصى لـ D.

أمثلة للإدخال/الإخراج

المثال الأول. كل زوج من العواميد بيوضح التواصل بين الجريدر و instance واحد.

Inst. 0 .Gra	Inst. 1 .Gra	Inst. 2 .Gra	Inst. 3 .Gra	Inst. 4 .Gra
100 0	100 1	100 2	100 3	100 4
w 12 1	w 50 1	w 99 0	w 7 1	r 5
r 50	r 7	r 12	w 1 1	0
1	1	1	5 !	5 !
5 !	r 1	w 0 0		
	1	5 !		
	5 !			

المثال التاني.

الجريدر	الـ Instance 0
8000 0	
	w 0 0
	w 1 1
	r 2
1	
	2 !

الجريدر	الـ Instance 1
8000 3	
	w 2 1
	r 1
0	
	r 2
1	
	r 1
1	
	2 !

explanation

المثال الأول. عندنا $N = 5$ عضوات ليهم IDs ورا بعض $0, 1, 2, 3, 4$ و $M = 100$ (ينفع لل subtasks 1، 3، و 4). الـ instance i بيمثل العضوة اللي الـ ID بتاعها i. التفاعل اللي فوق ده مجرد تسلسل قانوني ممكن للعمليات ومش **المقصود** منه إنه يكون استراتيجية فعالة أو منطقية؛ هو معروض بس عشان يوضح البروتوكول بيشتغل إزاي.

المثال التاني. عندنا $N = 2$ عضوات، ليهم IDs 0 و 3، و $M = 8000$ (ينفع لل subtasks 2، 3، و 4). في اليوم الأول، العضوة اللي معاها ID 0 بتكتب 0 في المكان 0 (مفيش تغيير)، والعضوة اللي معاها ID 3 بتكتب 1 في المكان 2.

...	4	3	2	1	0	**location**
...	0	0	1	0	0	**value**

في اليوم التاني، اللي معاها ID 0 بتكتب 1 في المكان 1، واللي معاها ID 3 بتقرأ من نفس المكان ده. خد بالك إن القراية بتحصل بالنهار، قبل الكتابة بليل. عشان كده، اللي معاها ID 3 لسه هتشوف 0.

...	4	3	2	1	0	**location**
...	0	0	1	1	0	**value**

في اليوم التالت، هما الاتنين بيقرأوا المكان 2، اللي مكتوب فيه 1.

في اليوم الرابع، اللي معاها ID 0 بتجاوب إن فيه 2 عضوات (صح)، بينما اللي معاها ID 3 بتقرأ الـ 1 اللي في المكان 1. اللي معاها ID 0 بتقفل على طول بعد كده ومش بتشارك في الأيام اللي جاية.

أخيراً، في اليوم $D = 5$، العضوة الباقية كمان بتجاوب صح بـ $N = 2$.

Testing

عشان نسهل عليك تست التست للحل بتاعك، بنوفر أداة بسيطة تقدر تحملها من الـ CMS. استخدام الأداة دي اختياري. خد بالك إن الجريدر الرسمي على الـ CMS مختلف عن أداة التست دي.

عشان تستخدم الأداة دي هتحتاج ملف input. تقدر تستخدم أمثلة الـ inputs اللي متوفرة `census.input0.txt` و `census.input1.txt`، أو تعمل واحد بتاعك. ملف الـ input لازم يبدأ بعدد العضوات N والـ IDs المتاحة M، وبعدهم سطر فيه N أرقام بتحدد الـ IDs بتاعة عضوات الجمعية.

لبرامج Python، مثلاً `census.py` (اللي بتشتغل عادة بـ `pypy3 census.py`) شغل أداة التست كالتالي:

```
python3 testing_tool.py pypy3 census.py < census.input0.txt
```

لبرامج C++، الأول اعمل compile للحل بتاعك:

```
g++ -DEVAL -std=gnu++20 -O2 -pipe -static -s -o census census.cpp
```

وبعدين شغل أداة التست:

```
python3 testing_tool.py ./census < census.input0.txt
```

خد بالك إن في المسألة دي الـ standard output بيُستخدم للتواصل مع الجريدر، عشان كده المفروض متستخدموش في الـ debugging. بدال كده، تقدر تستخدم الـ standard error output (أو stderr). في C++ تقدر تستخدم `cerr << msg << endl;`. وفي Python تقدر تستخدم `print(msg, file=sys.stderr)`.

أداة التست هتقرأ وتعرض رسايل الـ stderr دي مع الـ queries اللي بتعملها كل الـ instances بتاعت برنامجك. خد بالك إن لأسباب فنية ممكن يظهروا مش متزامنين بالظبط مع بعض.