Лента

Условие

За тази задача трябва да напишете две програми, които да работят заедно.

А разполага с лента, съставена от N клетки, като във всяка записва 1 или 0 (един бит).

Б започва в някаква неизвестна за *А* и *Б* клетка и трябва да разбере в коя (клетките са номерирани $1, 2, ..., N$ от ляво надясно). За целта *Б* може да се предвижва до съседната си вдясно клетка си (знае дали има такава) и вижда записаната от *А* стойност във всяка клетка, която посети. Разполага и със стойността на N

Тестване

За всеки тест *А* се изпълнява веднъж, но *Б* се изпълнява до $1\ 000$ пъти - записаните стойности от *А* не се променят, но варира началната позиция на *Б*. Задачата се оценява *адаптивно*, т.е. оценяващата програма ще избира началните позиции, за които да тества *Б*, спрямо кодирането на лентата, получено от *А*.

Оценяване

Тестовете са разделени на групи. Нека означим с Q максималния брой разрешени придвижвания на Вашата програма *Б* за дадена група.

1. (10 точки) $N \leq 1\ 000$, $Q = 5\ 000$
2. (10 точки) $N \leq 1\ 000\ 000$, $Q = 5\ 000$
3. (10 точки) $N \leq 1\ 000\ 000$, $Q = 100$
4. (10 точки) $N \leq 1\ 000\ 000$, $Q = 70$
5. (10 точки) $N \leq 1\ 000\ 000$, $Q = 50$
6. (50 точки) $N \leq 1\ 000\ 000$, $Q = 50$ Нека означим с C максималния брой придвижвания, направени от Вашата програма за *Б*. Точките Ви се определят по формулата $min(50, 50 \times \frac{50-C}{29})$

Решение

В анализа ще означаваме с Q максималния брой придвижвания на *Б* за даден алгоритъм.

Щом *Б* намери индекса на посетена от нея клетка, *Б* може да намери и индекса на началната, вадейки броя на придвижванията до клетката с известен индекс. Затова алгоритмите долу описват само как да се намери индекса на една от посетените клетки.

Теоретичен минимум

Полезно е наблюдението, че с по-малко от $\lceil \log_2 N \rceil$ бита информация (т.е $\lceil \log_2 N \rceil - 1$ придвижвания) *Б* не може да намери всяка стартова позиция: Броят стартови позиции би бил по-голям от броя на конфигурациите от битове, видени от Б. Следователно на поне една двойка начални позиции би съответствала една и съща конфигурация. Следователно тези две начални позиции биха били неразличими. От тук следва, че за всеки алгоритъм $Q \geq \lceil \log_2 N \rceil - 1 = 19$ (преместванията са с 1 по-малко от прочетените битове).

Алгоритми с $Q > O(\log N)$

Подзадача 1

А може да не записва нищо по лентата. Б се движи надясно до края на лентата, намирайки клетка с номер N. $Q = N$

Подзадача 2

А записва първо една 1 и една 0, след това две 1 следвани от две 0 и т.н. до $\lceil \sqrt{N} \rceil$ 1 и 0. Нека всяка такава последователност от еднакви символи наричаме сегмент.

Първо *Б* се движи надясно до края на началния сегмент - всеки край на сегмент се характеризира с два съседни, различни по стойност бита. След това *Б* продължава до края на следващия сегмент и така намира неговата дължина. Алгоритъмът обхожда най-много два цели сегмента, съставени от най-много $2 \cdot \lceil \sqrt{N} \rceil$ клетки и съответно $Q = 2 \cdot \lceil \sqrt{N} \rceil - 1$.

Тази информация е достатъчна на Б, за да намери индекса на началото на втория сегмент. Един начин да се постигне това е като се обхождат сегментите в реда, в който ги е записал *А* и се сумират дължините им. Началният индекс на текущия сегмент е равен на сумата от дължините на предхождащите го сегменти. Този алгоритъм е линеен по броя на сегментите и съответно $O(\sqrt{N})$.

Алгоритми с $Q = O(\log N)$

Това са алгоритмите, преминаващи следващите подзадачи. Те се базират на една основна идея, която се оптимизира по различни начини, за да се постигнат решения с по-ниски стойности Q.

Основен подход

Идеята е в последователности от няколко ($O(\log N)$) клетки да се кодира индекса на първата от тях. Основната трудност произлиза от това, че *Б* трябва освен това да разбере къде свършва едно кодиране и започва следващото. Това се постига, чрез поредица от битове *разделител* - последователност от битове, която се среща единствено между кодиранията и никъде вътре в тях. *А* записава последователно кодиранията на индексите на клетки, като между всеки две има по един *разделител*.

Алгоритъм с $Q = 80$

За кодиране *А* ползва двоичното представяне на индекса на началната клетка - необходими са $\lceil \log_2 N \rceil = 20$ бита. Върху лентата *А* ще записва числата *от дясно наляво* (т.е най-малко значимите битове са вляво, а най-значимите са вдясно) - това ще улесни някои от оптимизациите, описани по-долу.

За разделител ще ползваме 20 бита 1, следвани от един бит 0. Големият брой битове 1 гарантира, че последователността *разделител* няма да се срещне в някое от кодиранията, а 0-та служи за откриване на края на разделителя - ако я няма и *Б* види повече от 20 последователни бита 1, *Б* няма как да знае колко от тях са част от предхождащото разделителя кодиране и колко принадлежат на следващото го.

Б се движи надясно докато не прочете цял разделител, т.е. поне 20 бита 1, следвани от един 0. В този момент *Б* знае, че се намира в началото на кодиране на индекс и чете още 20 бита, които му казват индекса на началната клетка на кодирането.

В най-лошият случай *Б* прочита част от разделител (до 20 бита), цяло кодиране, което бива игнорирано (20 бита), цял разделител (21 бита) и още едно кодиране, носещо информация (20 бита). Общо $Q = 20 + 20 + 21 + 20 - 1 = 80$.

Лесни оптимизации

Има две оптимизации, които не изискват особени наблюдения или разсъждения. Поотделно всяка постига $Q = 60$, а заедно достигат $Q = 45$.

Оптимизиация 1

$Q = 60$ може да се постигне като единствено се подобри стратегията на *Б*. След като прочете цял разделител, *Б* знае и къде се намира началото на предходното кодиране. В случай, че го е прочел цялото, няма смисъл да чете следващото и може да спести 20 операции. Има два случая:

I. *Б* е прочел цяло кодиране преди разделителя

В най-лошия случай *Б* чете част от разделител (до 20 бита), цяло кодиране (20 бита) и цял разделител (21 бита).
Общо $Q = 20 + 20 + 21 - 1 = 60$.

II. *Б* не е прочел цяло кодиране преди разделителя

В най-лошия случай *Б* чете край на кодиране (до 19 бита), цял разделител (21 бита) и цяло кодиране (20 бита).
Общо $Q = 19 + 21 + 20 - 1 = 59$.

В по-лошия случай (I.) алгоритъмът е с $Q = 60$.

Оптимизация 2

Горният алгоритъм използва 20 бита за кодиране, достатъчни за всички N индекса. Броят на началата на кодирания е значително по-малък и би могъл да се кодира са и с по-малко битове.

В случай, че ползваме 15 бита за кодиране и 16 бита за разделител (15 бита 1, следвани от един 0) имаме най-много $\lceil \frac{N}{15+16} \rceil = \lceil \frac{1\ 000\ 000}{31} \rceil = 32259$ кодирани позиции. Броя на възможните кодировки $2^{15} = 32768$ е достатъчен ($32768 > 32259$).

Най-лошият случай се подобрява от $Q = 20 + 20 + 21 + 20 - 1 = 81$ на $Q = 15 + 15 + 16 + 15 - 1 = 60$.

Оптимизации 1 и 2

Двете оптимизации са напълно независими и ако се използват заедно се постига алгоритъм с $Q = 15 + 15 + 16 - 1 = 45$.

Трудни оптимизации

Тези оптимизации изискват нови идеи и са значително по-сложни. Заедно с оптимизации *1* и *2*, поотделно *3* и *4* постигат съответно $Q = 36$ и $Q = 30$, а ако се използват и четирите, се достига до $Q = 21$.

Оптимизация 3

Тук се намалява броят на придвижванията, които прави *Б*, преминавайки през *разделители*. Могат да се използват по-къси разделители, стига кодировките да не съдържат толкова на брой последователни единици, от колкото са съставени разделителите.

Преброяването на конфигурациите от конкретен битове без конкретен брой последователни единици е нетривиално на ръка. Лесно обаче, може да се напише програма, вършеща тази работа.

Броят на конфигурациите от 16 бита без 5 последователни единици е 52656. Ако ползваме 16 битови кодировки и разделители от 5 бита 1 и един 0, броят на индексите

на кодировките е $\lceil \frac{N}{16+6} \rceil = \lceil \frac{1\ 000\ 000}{22} \rceil = 45455$. Тъй като броят на индексите за кодиране е по-малък от броя на стойностите, които могат да се кодират ($45455 < 52656$), *Б* може еднозначно да определи позицията си.

Нека *А* първо генерира всички *валидни* конфигурации от 16 бита (такива, които не съдържат 5 последователни бита 1). Това може да стане като се обходят линейно всички 2^{16} конфигурации и се провери дали всяка от тях е *валидна*. Нека *А* записва върху лентата *валидните* конфигурации в нарастващ лексикографски ред - кодиранията отново се записват *огледално* и биват сравнявани лексикографски, като се четат *от дясно наляво*.

Докато кодира лентата, *А* може да запише в масив от 2^{16} елемента индексът, на който е записано всяко кодиране. След като *Б* е прочел кодирането, *Б* може да прочете от масива за $O(1)$ индекса на началната клетка на кодирането. Алтернативно *Б* може да намери индекса на началната клетка без допълнителен масив, използвайки двоично търсене в списъка от *валидни* конфигурации за $O(\log N)$.

Аналогично на оптимизация *1* можем да изчислим, че $Q = 15 + 6 + 16 - 1 = 36$. Тук случай II. е по-лошият, тъй като дължината на кодирането е значително по-голяма от тази на разделителя.

Оптимизация 4

В най-лошият случай горните алгоритми четат почти две цели кодирания, но ползват само едно. Без *А* да променя стратегията си (използваме само оптимизации *1* и *2*), *Б* може да открие къде се намира, четейки точно $15 + 16 = 31$ бита (дължината на едно кодиране + дължината на един разделител).

Има два случая или *Б* е прочел край на разделител, цяло кодиране и начало на следващ разделител, или е прочел край на кодиране, цял разделител и начало на следващо кодиране. Тъй като *Б* разбира, когато прочете цял разделител (*разделителите* са създадени точно с тази цел), лесно може да разграничи двата случая.

I. *Б* е прочел край на разделител, цяло кодиране и начало на следващ разделител

Частта от първия разделител, която е прочетена се състои от някакъв брой (потенциално 0) последователни битове 1, следвани от един бит 0. Именно първият бит 0, прочетен от *Б*, представлява края на разделителя - в противен случай разделителят трябва да съдържа повече от един бит 0, а в него има точно един такъв, т.е. достига се противоречие.

След като е открил края на първия разделител, *Б* знае кои от прочетените битове съставят кодирането и чрез кодирането намира началната си позиция.

II. *Б* е прочел край на кодиране, цял разделител, начало на следващо кодиране

Това е по-трудният случай, тъй като *Б* не разполага с цяло кодиране. Ще възстановим второто кодиране, използвайки факта, че двете кодирания са последователни двоични числа.

Нека означим първото кодиране със s, а второто с t. Нека прочетените битове от първото кодиране са k, съответно прочетените от второто са $15-k$. Нека означим със s_1 десните k бита на s, а със s_2 левите $15-k$, аналогично дефинираме t_1 и t_2. Ако с $\oplus$ означаваме конкатенация (слепване) имаме, че $s = s_2 \oplus s_1$ и $t = t_2 \oplus t_1$. Тъй като кодиранията са записани *огледално*, s_1 и t_1 представляват по-значимите битове на s и t , а s_2 и t_2 по-малко значимите.

На *Б* са известни s_1 и t_2 и знаем, че $t = s + 1$. Търси се t.

В случай, че $t_2 > 0$, то съществува поне едно, кодиране, което е по-малко от t кодиране и започва с t_1, например $(t_2 - 1) \oplus t_1$. Тъй като s е най-голямото кодиране по-малко от t, следва, че s също започва с t_1, т.е. $s_1 = t_1$ и $t = t_2 \oplus s_1$.

В случай, че $t_2 = 0$, то t е най-малкото кодиране, започващо с t_1. Тъй като $s < t$ ($t = s + 1$), имаме $s_1 < t_1$. Но също, тъй като $t = s + 1$, имаме $t_1 \leq s_1 + 1$. От горните две неравенства следва, че $t_1 = s_1 + 1$ и съответно $t = t_2 \oplus (s_1 + 1)$

$Q = 15 + 16 - 1 = 30$

Оптимизации 3 и 4

Оптимизация *3* не се променя - използваме за разделител 5 бита 1 и един 0 и кодираме индексите със 16 битови *валидни* конфигурации (без 5 последователни бита 1).

Б отново чете битове за едно кодиране и един разделител, вече му трябват само $16 + 6 = 22$. Случай I. от оптимизация *4* не се променя, затова не се описва тук. Случай II. се усложнява, защото вече не е вярно, че $t = s + 1$.

Отново на *Б* са известни s_1 и t_2 и знаем, че t е най-малкото *валидно* кодиране такова, че $t \geq s + 1$. Търси се t.

В случай, че $t_2 > 0$, отново съществува поне едно *валидно* кодиране, което започва с t_1 и е по-малко от t. Тъй като t е валидно кодиране, t_1 също отговаря на условието за *валидност* т.е. не съдържа 5 последователни бита 1. От тук следва и че $\overbrace{(0...0)}^{16-k} \oplus t_1$ е *валидно* кодиране (добавяне на битове 0, няма как да наруши *валидността*). От $t_2 > 0$ следва, че $\overbrace{(0...0)}^{16-k} \oplus t_1 < t_2 \oplus t_1$, т.е. $\overbrace{(0...0)}^{16-k} \oplus t_1 < t$ - намерихме едно *валидно* кодиране, което започва с t_1 и е по-малко от t. А от това, че s е най-голямото *валидно* кодиране, по-малко от t, следва, че s също започва с t_1, т.е. $s_1 = t_1$ и $t = t_2 \oplus s_1$.

В случай, че $t_2 = 0$, t отново е най-малкото *валидно* кодиране, започващо с t_1. Следователно $t_1 > s_1$ или $t_1 \geq s_1 + 1$. От тук следва и че $t \geq t_2 \oplus (s_1 + 1)$. $s_1 + 1$ и

съответно $t_2 \oplus (s_1 + 1)$ обаче може да не отговарят на условието за *валидност*, т.е. може да съдържат 5 или повече последователни бита 1. Понеже s е следвано от t в лексикографското подреждане на *валиднитие* кодирания, t трябва да е най-малкото *валидно* кодиране, отговарящо на $t \geq t_2 \oplus (s_1 + 1)$. *Б* може да немери t чрез двоично търсене в списъка от *валидни* конфигурации, съставен от *А* за $O(\log N)$.

Ако се използват и четирите оптимизации, $Q = 16 + 6 - 1 = 21$, почти достигайки теоретичния минимум $Q = 19$.

Алтернативен подход за оптимизация 4

Оптимизация *4* не променя запълването на лентата от *А*, а само начина, по който *Б* го интерпретира. Горе беше доказано, че това е възможно, и предложен алгоритъм за *Б*. В случай, че състезателите *повярват*, че е възможно *Б* да намери позицията си с някакъв брой ходове, съществува значително по-лесен за измисляне алгоритъм, който да открива началната позиция на *Б*.

Този подход се основава на наблюдението, че необходимо и достатъчно условие *Б* да може да открие всяка начална позиция за P хода (четейки $P + 1$ бита) е да не съществува двойка еднакви последователности от битове в лентата с дължина $P + 1$. Достатъчно е, защото щом *Б* е прочел $P + 1$ бита, може да е сигурен, че е започнал в началото на единственото срещане на прочетената от него последователност от $P + 1$ бита.

Нека е изпълнено горното условие за някакво P. *А* запълва лентата без да променя стратегията си. Докато го прави, *А* записва в речник D (примерно *std::map* в C++) началния индекс на всяка подпоследователност от $P + 1$ бита. Стратегията на *Б* се състои в това да прочете $P + 1$ бита и да прочете от D началната си позиция за $O(\log N)$ операции.

След като даден състезател е измислил опитимизации *1, 2* и (потенциално) *3*, той може да приложи горния алгоритъм върху редицата, генерирана от *А*, и да намери най-малката стойност на P, за която *Б* има стратегия. Така може да постигне резултата от оптимизация *4*, без да минава през всички разсъждения.

Пробабилистичен подход

Този подход се основава на наблюдението за необходимото и достатъчно условие *Б* да има стратегия с P хода.

Ако *А* генерира лентата на случаен принцип, се очаква да няма две еднакви последователности с дължина 36 (открито, чрез тестване). Ако по случайност има такива две последователности, *А* може да генерира нови редици, докато не попадне на такава без.

Стратегията на *Б* е същата като при алтернативния подход за оптимизация *4*.

В чистия си вид решението е с $Q = 36 - 1 = 35$, но може да бъде подобрено чрез алчни алгоритми.