АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА
ТЪРСЕНЕТО НА НЕМО

- ✓ **Тагове**: сортиране, търсене, рекурсия, оптимизация на паметта, случайни числа, обработка на тестове

Подзадача 1

За решаването на тази подзадача е достатъчно да генерираме числовата редица чрез зададената формула и да сортираме елементите ѝ в ненамаляващ ред. Поради неголемите ограничения, това може да бъде направено и чрез не особено ефективен алгоритъм със сложност $O(N^2)$, като например метода на мехурчето.

Подзадача 2

В тази подзадача вече е необходим по-бърз алгоритъм за сортиране. Не е необходимо състезателите да имплементират сами такъв – може да се ползва стандартната сортировка в библиотеката ***algorithm***, която е със сложност $O(N \log_2 N)$. Реализация в ***searching-stlsort.cpp***.

Подзадача 3

Тази подзадача е първата, която евентуално би могла да направи отсяване между състезателите, понеже се очаква, че първите две ще бъдат решени от всички. Решение със сложност $O(N \log_2 N)$, дори и доста оптимизирано, не би могло да приключи изпълнението си в рамките на определения лимит за време. Затова трябва да използваме допълнителното ограничение за тази подзадача, а именно, че $D \leq 10^7$ т.е. всички числа в редицата са по-малки от 10 милиона. Въвеждаме, допълнителен масив $cnt[]$, в който ще записваме за всяка стойност от 0 до $D-1$ колко пъти се среща в редицата. С помощта на този масив лесно можем да намерим K-тото по големина число чрез обхождане от ляво надясно, като поддържаме броя на числата по-малки или равни по стойност на текущия индекс в масива. Сложността е $O(N+D)$ по време и $O(D)$ по памет. Решението е реализирано в ***searching-cntsort.cpp***.

Подзадача 4

Тук решението на предишната подзадача е неприложимо, защото е невъзможно да декларираме масив с големина 1 милиард елемента. Това е подзадачата, в която се съдържат повечето ключови наблюдения за решението на задачата. Някои от състезателите вероятно ще се насочат към решение с двоично търсене по отговора по следната схема:

1. Нека стойностите L и R в началото да имат стойности съответно 0 и $D-1$;
2. Преброяваме числата, чиито стойности са в интервала $[L, M]$, а останалите очевидно са в интервала $[M+1, R]$, където $M = \frac{(L+R)}{2}$;
3. Нека този брой е C. Тогава, ако $C \leq K$ отговорът на задачата е K-тото число измежду числата със стойности в интервала $[L, M]$, а в противен случай отговорът е $(K-C)$-тото поредно число със стойност в интервала $[M+1, R]$ (понеже вече сме намерили C числа с по-малка стойност извън този интервал);
4. Продължаваме да решаваме задачата за новия интервал и новата стойност на K, докато $L \neq R$.

АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА ТЪРСЕНЕТО НА НЕМО

Остана да уточним как точно преброяваме числата в т. 2. Първата възможност е просто да обходим редицата. Така ще получим решение със сложност $O(N \log_2 D)$, което асимптотично е малко по-лошо от решението за подзадача 2, защото в общия случай $D > N$. Обаче може да забележим, че веднъж излязла от разглеждания интервал, стойността на дадено число, никога няма да бъде преброена отново. Следователно, ако поддържаме само числата в редицата, чиито стойности са били в последния интервал $[L, R]$, когато ни се наложи да преброим колко числа са в новия интервал $[L', R']$, може да разглеждаме само тези от числата, които пазим, защото $L \leq L', R' \leq R$. Така получаваме решение, което в най-добрия случай има сложност $\Omega(N)$. Този случай се достига, когато на всяка стъпка числата се разделят на две равни части и броят операции се определя от сумата $N + \frac{N}{2} + \frac{N}{4} + \cdots + 1$, която е ограничена от $2N$. За съжаление, това решение отново е $O(N \log_2 D)$, защото нямаме никаква гаранция, че ще получим такова разделяне. Например, ако всички числа са равни, двоичното търсене отново ще трябва да направи $\log_2 D$ стъпки, на всяка от които ще се разглеждат всички N числа.

Една възможна оптимизация би била да инициализираме L и R като най-малката и най-голямата стойност, която се среща в редицата. Това обаче отново има ограничен ефект, тъй като можем да имаме стойности близки до 0 и до D, но все пак голяма част от числовия интервал да бъде празна. Така пак ще бъдат направени доста стъпки, в които се разглеждат голяма част от числата. Това решение е реализирано в ***searching-binsearch.cpp***.

Може би тук е моментът да уточня, че нарочно не съм се престаравал с тестовете за тази подзадача, за да дам възможност повече участници да се доближат до 80 точки чрез идеите описани до момента. От друга страна, начинът, по който е зададен входът, изключително затруднява създаването на контратестове. С голяма вероятност може да очакваме, че стойностите на числата са равномерно разпределени в числовия интервал $[0, D)$, когато са изпълнени следните условия:

- $B \neq 1$ и $1 \ll (C \bmod D) \ll D$ – иначе ще се получи нарастваща или намаляваща аритметична прогресия, съсредоточена в сравнително малка част от интервала;
- $A * B + C \not\equiv A \pmod D$ – иначе всички числа са равни;
- N е сравнително голямо.

Ако пък някое от тези условия не е изпълнено, подзадачата може доста лесно да се реши с разглеждане на точно тези частни случаи.

Предвиденото решение за тази подзадача, използва подобна идея, която вероятно е добре позната на състезателите, които, като част от подготовката си, са разглеждали в детайли по-известните алгоритми за сортиране (в частност quick sort). Сега ще изложа накратко в какво се състои той.

Целта е да сортираме масива $arr[]$ от начална позиция L до крайна позиция R. Избираме едно число от този интервал на масива, например това на позиция $M = \frac{L+R}{2}$. Разделяме числата на две части – такива по-малки и такива по-големи от стойността на избраното число. След това решаваме задачата рекурсивно за получените две части от редицата, докато $L \neq R$.

АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА
ТЪРСЕНЕТО НА НЕМО

Сложността на алгоритъма силно зависи от избора на числото, по което се извършва разделянето на числата в интервала. В най-лошия случай (ако всеки път избираме да разделяме по най-малкото или най-голямото число) сложността е $O(N^2)$, но се оказва, че средно алгоритъмът работи за $O(N \log_2 N)$. Всъщност бързодействието му се дължи на изключително ефективния начин за извършване на разделянето, който е измислен от сър Тони Хор през 1959 г. Оставям разглеждането на този начин за упражнение на читателя :)

Единствената промяна, която трябва да направим по изложения алгоритъм, е да не сортираме двете части на интервала след разделянето, а само тази, която съдържа търсеното число. Сложността на решението в този случай средно е $O(N)$ и то е реализирано в ***searching-nomemopt.cpp***.

Подзадача 5

Като цяло решението на тази подзадача не се различава идейно от предложеното за подзадача 4, но все пак има някои технически детайли, които би трябвало да затруднят и най-добрите състезатели в тази възрастова група.

Първият проблем, който се появява, е, че не можем да съхраним цялата редица в позволената памет. Имаме 10 милиона числа от 64-битовия тип “long long”, т.е. ≈ 80 MiB, докато лимитът е 64 MiB. Ще приложим стандартна техника, с която ще жертваме бързодействие с цел да спечелим памет. Вместо с реалните стойности на числата бързото сортиране ще работи с техните индекси (които се побират в “int”) и всеки път, когато ни трябва стойността на някое число, тя ще бъде изчислявана наново. За да може изчисляването да се случва за време, което да не повлияе значително върху бързодействието, ще запазим стойността на всяко четвърто число от редицата. Останалите ще бъдат пресмятани по формулата чрез най-много 3 операции, в средния случай – 2. Така паметта става $N \times 4B + \frac{N}{4} \times 8B = N \times 6B = 6 \times 10^7 B \approx 60\ MiB$, което се побира точно в лимита за памет. Реализация на този подход може да намерите в ***searching-norand.cpp***.

Друг начин да се постигне същият ефект, е да се извършат няколко начални стъпки на алгоритъма, така че да се изключат достатъчен брой числа, за да можем да можем да запазим останалите в един масив и да приложим решението за подзадача 4. Този подход сякаш създава повече технически проблеми (главно свързани с избора на число, по което да се извърши разделянето, и с факта, че самото разделяне не може да стане по метода на Хор) и затова не се препоръчва.

Дотук решението би трябвало да донесе около 95 точки. За последните 5 точки ще се върнем отново на несъвършенството при избора на число за разделяне. Алгоритъмът би извършил голям брой операции при входните данни от тест 37 (защо?). Затова в авторовото решение ***searching.cpp*** използва генераторът на псевдослучайни числа Mersenne-Twister, който е включен в библиотеката ***random***. В случай че решението отново не преминава всички тестове, може да се пробва генераторът да бъде инициализиран с различен seed. Това е и причината задачата да има пълен фийдбек.

Автор: Добрин Башев