	На пълното решение	На подзадачите
Тагове	Метод на показалките Динамично оптимиране	Пълно изчерпване Двоично търсене Префиксни суми

Анализ

Нека за всеки елемент $1 \leq i \leq N$ от редицата да дефинираме $jump_i$ ($i < jump_i$) като най-левият елемент, за който подмасивът $[i, jump_i]$ на редицата да включва всички числа от 1 до L. Нека да представяме подмасивите $[i, jump_i]$ като отсечки на числова ос, като цената на една отсечка е минималният брой елементи, които трябва да се махнат от нея, така че да се получи пермутация на азбуката (дължината ѝ $= jump_i - i + 1 - L$). Задачата е да намерим минималната сумарна цена на множество от K непресичащи се отсечки.

Подзадача №1

В тази подзадача са тестовите примери. Тя е за обратна връзка от системата.

Подзадача №2

За подзадачата се очаква решение с пълно изчерпване. Фиксират се кои елементи от редицата да се изтрият и се прилага следната *greedy* идея върху останалите: намира се най-лявата позиция от редицата, за която тя е край на пермутация на азбуката, като тази пермутация не трябва да се пресича с някоя предишна преброена. Добавя се едно към броя на пермутации в тази подредица. Горните две стъпки се повтарят, докато не се намери такава позиция.

Постигната сложност: $O(2^N \times N + L)$

Имплементация: **second_15.cpp**

Подзадача №3

За всеки елемент i се намира $jump_i$ с вложен цикъл, като постепенно се увеличава брояч j. Поддържа се броят на различни числа в $[i, j]$. $jump_i = j$ за най-лявото j, за което броят на различните числа в $[i, j]$ станат равни на L. После се разглежда всяка двойка отсечки, които не се пресичат и се намира тези със сумарно минимална цена.

Постигната сложност: $O(N^2 + L)$.

Имплементация: **third_14.cpp**

Подзадача №4

Всъщност, може да се забележи, че $jump_i \leq jump_{i+1}$. Причината за това е, че ако в интервала $[i+1, jump_i - x]$ има L различни числа, а в $[i, jump_i]$ няма, то се достига до противоречие. Поради това, $jump_i$ може да се намери с метода на показалките. След като са намерени отсечките, нека min_i да е равна на минималната цена на отсечка, с начало $\geq i$. Тогава, за всяка отсечка се разглежда каква ще е минималната цена, ако се фиксира за по-лявата: $jump_i - i + 1 - L + min_{jump_i}$.

Постигната сложност: $O(N + L)$.

Имплементация: **forth_29.cpp**

Подзадача №5

Може да се намери $jump_i$ по същия начин, по който се намира в трета подзадача, поради ниските ограничения. Тъй като K е твърде голямо, не е възможно да се прави нещо подобно на трета и четвърта подзадача, заради това трябва да се смени идеята. Опитите да открия вярно *greedy* решение претърпяха неуспех, макар и обстоятелствата да индикират, че е възможно съществуването на такова. Това определя единствената възможност – да се използва динамично оптимиране. Стейтът на ДП-то би могъл да е следния: $f(index, left)$, където $index$ е минималното възможно начало за следваща отсечка от множеството, а $left$ е колко отсечки ни остава да изберем. С вложен цикъл може да се фиксира следващата избрана отсечка.

Постигната сложност: $O(N^2K + L)$.

Имплементация: **fifth_45.cpp**

Подзадача №6

Тази подзадача е за състезателите, които не са могли да се досетят за метода на показалките, но са успяли да се сетят за по-добро ДП и по-различен начин за намиране на $jump_i$. За да се реализира този по-различен начин, ще трябва да се поддържа бройката на числа със стойност x в интервал. Това лесно може да стане с префиксни суми, което, разбира се, би изисквало много време и памет. Поради малкото L, това не би ни било проблем. Безстрашните състезатели биха могли да реализират това с доста по-напреднали подходи, например *персистентно сегментно дърво*, *wavelet дърво* или *merge sort дърво*, но тези структури от данни са далеч извън конспекта за С група, може би и за повечето от В група. След като се изчислят префиксните суми, $jump_i$ би могло да се намери чрез двоично търсене. Трябва да се намери минималното j, за което интервалът $[i, j]$ има L на брой различни числа. С префиксните суми се проверява дали всяко число от 1 до L се среща в интервала $[i, j]$.

След като се намери $jump_i$, ще трябва да се оптимизира ДП-то. Всъщност, не е нужен вложен цикъл, а просто може да се разгледа дали се взима отсечката на позиция $index$, или не.

Постигната сложност: $O(NK + NL\ log_2 N)$.

Имплементация: **sixth_76.cpp**

Подзадача №7

$jump_i$ се намира чрез метода на показалките, като се прилага динамичното от шеста подзадача.

Постигната сложност: $O(NK + L)$.

Имплементация: **seventh_89.cpp**

Подзадача №8

Проблемът, подобно на *price* от ЕТИ 2021 година, е паметта. Може да се реализира подобна оптимизация на паметта, като се махне второто измерение на динамичното. За повече подробности по оптимизацията, погледнете имплементацията или анализа на *price*.

Постигната сложност: $O(NK + L)$.

Имплементация: **author_100.cpp**

Автор: Борис Михов