**Анализ на решеието на задача**

**U-конгруентност**

Ако означим броя на различните числа във входа с *cnt*, то очевидно става дума за лексикографската подредба на извадките без повторение на *U* елемента от *cnt*-ти клас. Наричаме множеството от различните входни числа „азбучно“.

Съотношенията между елементите могат да се запазят, ако се използват последователни цели положителни числа. Първото в лексикографската наредба ще се получи, когато заменим най-малкия елемент в азбучното множество с 1, следващия по големина с 2 и т. н.

Това разсъждение води до следния алгоритъм за намиране съответствие на извадка с азбучното множество:

1. Правим списък *L* на елементите в азбучното множество.
2. Подреждаме *L* растящо по големина.
3. На всеки елемент от подредения списък *L* съпоставяме следващия елемент от извадката (разбира се, предполагаме, че, както е стандартно, в извадката елементите-индекси са нарастващо подредени).

Сега вече работим с целите положителни числа от 1 до *U* като с индекси. В това индексно множество търсим „средната“ извадка измежду всички извадки на *U* елемента от *cnt*-ти клас.

1. **Изчерпващ алгоритъм**

Ясно е, че ще е излишно разхищение на памет при по-големи *U* да се запомнят всички извадки, само за да се намери „средната“. Ако не знаем колко е общият им брой, можем да подходим по следния начин: образуваме първата извадка *a*={1, 2, …, *cnt*} и последната *b*={*U*‑*cnt*+1,*U*‑*cnt*+2,…,*U*‑2,*U*-1,*U*}. Чрез известния алгоритъм за получаване на следващата лексикографски растяща извадка и не толкова известния, но лесно съобразим алгоритъм за получаване на следващата лексикографски намаляваща извадка, преобразуваме двете извадки до съвпадане на *a* и *b*. (Поради дефиницията на „средна редица“ прилагаме първо втория алгоритъм върху *b*, а след това първия върху *a*.) Ако знаем, че общият брой е , можем просто да приложим върху *a* известния алгоритъм ‑1 пъти, което намалява броя на итерациите наполовина. И при двата подхода ще е необходимо съвсем малко количество памет, с което разполагаме.

1. **Генериращ алгоритъм**

Изчерпващият алгоритъм прави твърде много итерации (с порядък и работи добре само за по-малки стойности на *U* или когато *cnt* е по-далече от *U*/2. На това ниво състезателите бива да познават и да могат да реализират алгоритъм за бързо генериране на *m*-тата лексикографска извадка на *n* елемента от *k*-ти клас. Ограниченията на данните са такива, че той може да бъде директно приложен в най-стандартната си форма: се събира в стандартния за C++ тип **long long**. Анализирайки по-дълбоко изчислителния процес в този алгоритъм, можем да забележим, че необходимите в него стойности на функцията (брой комбинации на *n* елемента от *k*-ти клас), започвайки от , или не се променят на следващата стъпка, или *n* намалява с 1, или и *n*, и *k* намаляват с по 1. Това ни позволява да използваме предишно изчислената стойност за и очевидните формули и . Така константно получаваме само нужните ни стойности на функцията.

След намирането на „средната“ извадка, остава само да се заменят стойностите на получените индекси със съответстващите им числа от азбучното множество. Без излишно разхищение на памет всичко може да стане само с един голям масив, в който сме запомнили входните данни. Ако не икономисваме памет, можем да използваме и други алгоритми.

Щом броят на числата във входа е *N*, сложността на тази част от алгоритъма в най-лошия случай е O((*U*-1)×*N*): до *U*-1 обхождания за промяна на елементите (при *cnt*=*U*, естествено, редицата в семейството е една и „средната“ е самата входна). Можем да заменим тези обхождания с двоично търсене на съответния елемент в подредения списък, с което коефициентът пред *N* да падне на (в най-лошия случай). Ако ползваме проста динамична идея за съответстващия елемент от азбучното множество, общата сложност на тази част пада до O(*N*). Извеждането може да стане едновременно с обхождането за преработка.

*Автор: Павлин Пеев*