**АНАЛИЗ**

**НА РЕШЕНИЕТО НА ЗАДАЧА**

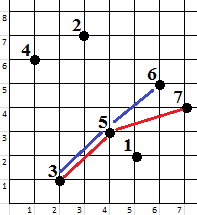
**ОТБОР**

Тази задача всъщност много наподобява задачата за най-дълга нарастваща редица, но тук гледаме обща нарастваща подредица на две редици, които са пермутации.

Решението за 19 точки е стандартно динамично – пазим за всеки елемент, например във втората редица, дължината на най-дългата обща нарастваща подредица и обхождаме последователно, като за определянето на отговора за индекс, преглеждаме предните по-малки елементи и проверяваме дали и в двете редици се намира този елемент преди текущия. Сложността е .

Има две еквивалентни преобразувания, по които можем да добием представа какво точно трябва да правим. Нека представим данните като точки с абсциса – стойността на числото, а ордината индекса в първата редица, където се среща това число. Така ако обхождаме втората редица последователно отговора за текущо число геометрично ще представлява максималната редица от „нарастващи“ точки в правоъгълника от началото на координатната система до текущата точка. Другото преобразование ще е координатите да са индексите на даден номер в двете редици и за да направим същото решение трябва да обхождаме точките в нарастващ ред на номерата.

Решение за 48 точки - за примера от задачата:

Нанасяме точките с абсциса индекса+1 в първата редица, а ордината – индекса+1 във втората редица. Получава се следната картинка:

Вижда се, че решенията са 2 – (3,5,6) и (3,5,7). Намирането на тези точки в правоъгълника (0;0) - (x;y) става като последователно (в реда на номерата) в двумерно дърво на Фенуик се поставя точките с най-дългата обща нарастваща редица, завършваща в тях. Такова решение, за да е успешно трябва да не ползваме двумерен масив за дървото на Фенуик, иначе ще трябва страшно много памет. Вместо това по едното измерение ще използваме unordered\_map. Така ще използваме само толкова памет, колкото ни е необходима (). Това решение ще хване повечето точки, но константата му е голяма заради unordered\_map. Сложността е .

Решение за 81 точки.

Тук абсцисата на точката ще е номер, а ординатата – индекса на номера в първата пермутация. Ще използваме индексно дърво по координатата х и за всеки връх в дървото ще си пазим подредени точките по координатата y. Имаме следните заявки към дървото – добавяне на нова точка и вземане на максимум с дадено ограничение за у. Това ни принуждава да пазим за даден връх в индексното дърво точките подредени по у в балансирано дърво. Най-лесно за писане и популярно е treap (деремида). Тя позволява бързо добавяне на нов връх, както и бърза заявка за максимум при дадено ограничение за число. Сложността е отново , но с по-малка константа.

Решение за 100 точки.

Можем да модифицираме и другия стандартен алгоритъм за най-дълга нарастваща подредица да работи и в този случай, когато имаме точки. Отново за всяка дължина ще държим най-подходящ кандидат и ще правим двоично търсене за определяна на дължината на най-дълга нарастваща подредица, завършваща в дадена точка. В случая обаче не е достатъчно да пазим само една точка за дадена дължина, а би трябвало за всяка възможна x-координата да пазим точката с най-малкия възможен y. Ако се замислим бихме могли да пазим по-малко кандидати за дадена дължина. Нека имаме две точки (x, y) и (z, t), за които x ≤ z и y < t. Лесно се вижда, че не ни трябва да пазим и двете точки – първата винаги е по-добра от втората. Така като следствие – за кандидатите на дадена дължина можем да получим, че ако ги подредим по нарастване на първата координата, то те ще намаляват по втората си координата. По този начин се улеснява процеса по отговаряне на въпроса на двоичното търсене. Като сме фиксирали дадена дължина и точка (x, y), то трябва да намерим точката с най-голяма първа координата, която е по-малка от x и да видим дали втората ѝ координата е по-малка от y. Няма нужда да гледаме точки кандидати с по-малка първа координата, защото техните втори координати ще са по-големи от y (поради начина, по който ги пазим). Затова всъщност за всяка дължина можем да пазим точките в set, за да отговаряме бързо на тези въпроси, а и защото като добавим точка кандидат е възможно да се наложи да махнем част от вече сложените точки. Крайната сложност е , но с много малка константа и много лесен код (защото при индекс за първа координата, нямаме повторения).

*Анализ и реализация: Илиян Йорданов*

*Условие и тестове: Павел Петров*