Анализ на задача permutation

Подзадача 1 и 2

Малките ограничения на n предполагам brute-force тип решения. Първата е очевидна за знаешите функцията `std::next_permutation`. Втората подзадача очаква да се досетим, че генерирайки пермутацията се интересуваме единствено от кои елементи все още не са използвани, а не някакви конкретни части от структурата на пермутацията. Сложностите са $O(n! \times n)$ и $O(2^n \times n)$ съответно.

Подзадача 3

Задачи от типа на "преброй редиците, отговарящи на условия A, B, C и т.н" предполагат решения, основани на идеята за динамично програмиране. Обикновено е невъзможно да се освободим от нуждата да пазим в стейта на динамичното колко на брой елемента сме генерирали, така че това ще го запазим. Ограничението $n \leq 3000$ ни подсказва, че ще пазим едно число за допълнително информация в каква точна редица сме се озовали. Въпросът, който активно трябва да си задаваме е "Каква е важната информация за редица от две по две различни числа, отговарящи на дадени неравенства, която трябва да пазим?" В допълнение, нека запазим основния принцип на динамичното, който е: задачата решена за вход n да се възползва от задачата решена за вход $n-1$. Така въпросът ни става по-скоро "Каква е важната информация за пермутация, отговаряща на дадени неравенства, която трябва да пазим?"

Добавяйки ново число, единственото ограничения върху него е зададено от поредния символ във входния низ. Ако досега сме генерирали пермутация $3, 1, 2$ и трябва да удовлетворим ограниченията $><>$, трябва да добавим четвърто число, което да е по-малко от 2. Ключовият момент в задачата е следният: ще "вмъкнем" новата стойност, като преместим по-големите или по-малките числа с едно. Така $3, 1, 2$ може да преходи в $4, 2, 3, 1$ или $4, 1, 3, 2$ при низа $><>$. Ако ограниченията бяха $><<$, то пермутациите щяха да са $4, 1, 2, 3$ и $3, 1, 2, 4$. Преходите са $dp[pos][val] = \sum_{prev<val} dp[pos-1][prev]$ за $<$ и $dp[pos][val] = \sum_{prev \geq val} dp[pos-1][prev]$ за $>$.

Подзадача 4

Предходната идея може да се оптимизира с префиксни суми.

Анализ: Иван Лупов