Анализ на задача Кули

Накратко трябва да намерим броя на редиците $a_1, a_2 \dots a_k$, от естествени числа, за които сумата на елементите е равна на S и е изпълнена една от следните зависимости:

- $a_1 > a_2 < a_3 > \cdots$
- $a_1 < a_2 > a_3 < \cdots$

Първа подзадача. Чрез брут форс можем да генерираме всички възможни редици със сума на елементите S и да проверим кои от тях изпълняват една от двете зависимости. Сложност – **O(S^S)**.

Втора подзадача. Можем да подобрим пълното изчерпване от предишната подзадача, като генерираме само валидните редици. Нека $brute(sum, last, flag)$ е броят на редиците със сума sum, за които първия елемент е строго по-голям от $last$ за $flag = 0$ или строго по-малък от $last$ за $flag = 1$. Знаейки, какви условия трябва да са спазени за първия елемент, лесно получаваме следната зависимост:

$$brute(0, last, flag) = 1$$
$$brute(sum, last, 0) = brute(sum - (last + 1), last + 1, 1) +$$
$$+ brute(sum - (last + 2), last + 2, 1) + \cdots + brute(sum - sum, sum, 1)$$

$$brute(sum, last, 1) = brute(sum - (last - 1), last - 1, 0) +$$
$$+ brute(sum - (last - 2), last - 2, 0) + \cdots + brute(sum - 1, 1, 0)$$

Отговора на задачата ни ще е $brute(S, 0, 0) + brute(S, S, 1)$ – броят на редиците от вида > < > ... плюс броят на редиците от вида < > < ... , за които първия елемент е по-малък от S, защото иначе ще броим редицата {S} два пъти. Сложност – O(отговора), което за $S \leq 40$ е не повече от някъде 50млн стъпки.

Трета подзадача. За тази подзадача няма предвидени конкретни решения, тя е за тези, които не са написали решението си, така че да работи с оптимална скорост или за тези, подобрили решението на предишната подзадача с някакви оптимизации.

Четвърта подзадача. Веднъж щом видим формулата от втора подзадача, няма как да не се сетим и за динамичното програмиране. Параметрите са достатъчно малки, за да попълваме намерените отговори в един масив $dp[sum][last][flag]$.
Сложност – **O(S^3)**.

Пета подзадача. Следващата оптимизация след мемоизацията ще е съкращаване на рекурентната формула. Да се загледаме във формулите на два доста близки стейта:

$$dp[sum][last][0] = dp[sum-(last+1)][last+1][1] + \\ + dp[sum-(last+2)][last+2][1] + \cdots + dp[sum-sum][sum][1]$$

$$dp[sum][last+1][0] = dp[sum-(last+2)][last+2][1] + \\ + dp[sum-(last+3)][last+3][1] + \cdots + dp[sum-sum][sum][1]$$

Всички събираеми без едно съвпадат. Можем да се възползваме от този факт и да запишем формулата по следния начин:

$$dp[sum][last][0] = dp[sum-(last+1)][last+1][1] + dp[sum][last+1][0]$$

Разсъждения за $flag = 1$ са същите:

$$dp[sum][last][1] = dp[sum-(last-1)][last-1][0] + dp[sum][last-1][1]$$

Така си спестяваме линейните цикли за всеки стейт. Сложност – **O(S^2).**

Шеста подзадача. Какво още можем да подобрим? Рекурсията сама по себе в някои случаи може да бави решението. За тази подзадача е предвидена итеративната имплементация на динамичното. Така казано може и да звучи просто, но има някои подробности, които си струва да се споменат. Основният проблем при итеративните решения е, че трябва да преценим в какъв ред да запълваме стойностите в масива. Тук за дадено $dp[sum][last][flag]$ трябва да сме изчислили всички с по-малък sum. А $last$? При него трябва да имаме в предвид стойността на $flag$. Ако тя е 0, то трябва да ги изчисляваме в намаляващ ред на $last$ и в противен случай – в нарастващ ред на $last$.

```
for(int last=1;last<=s;last++){
    dp[0][last][0]=dp[0][last][1]=1;
}

for(int sum=1;sum<=s;sum++){

    for(int last=sum-1;last>=0;last--){
        dp[sum][last][0]=dp[sum-(last+1)][last+1][1] + dp[sum][last+1][0];
        dp[sum][last][0]%=mod;
    }

    for(int last=2;last<=s;last++){
        dp[sum][last][1]=dp[sum][last-1][1];
        if(sum-(last-1)>=0)dp[sum][last][1]+=dp[sum-(last-1)][last-1][0];
        dp[sum][last][1]%=mod;
    }

}
```

Автор: Александър Гатев