Leetcode 1223 e Variações

Daniel Saad

11 de maio de 2026

1 Enunciado

O problema 1223 da plataforma Leetcode pode ser resumido da seguinte forma: tome n o número de lançamentos de um dado com 6 faces, numeradas de 1 a 6, determinar o número de sequências distintas de lançamentos possíveis sem que não haja mais do que d_i lançamentos **consecutivos** da mesma face i. Como o número de sequências pode ser muito grande, o problema solicita que o resultado seja dado módulo $10^9 + 7$.

O problema também garante que $d_i \geq 1$ para toda face i, o que significa que não há face proibida.

Exemplo 1. Tome $n = 2$ e $d = \langle 1, 1, 2, 2, 3, 4 \rangle$. O número de sequências possíveis é 34, pois das 36 sequências geradas por dois lançamentos de um dado de seis faces, as sequências $\langle 1, 1 \rangle$ e $\langle 2, 2 \rangle$ não são permitidas, haja vista que não deve haver mais do que 1 lançamento consecutivo da face 1 ou da face 2.

Esse é um problema de combinatória, mas podemos resolvê-la utilizando programação dinâmica.

2 Solução

Precisamos modelar a solução recursivamente. Considere que $M(i, j, k)$ seja o número de sequências distintas possíveis após i lançamentos, onde o último lançamento se deu na face j e houve k lançamentos consecutivos da face j nos últimos k lançamentos.

Queremos computar $M(i, j, k)$ para todo $1 \leq i \leq n$, $1 \leq j \leq 6$ e $1 \leq k \leq d_j$. Seja x a face do lançamento $i - 1$. Temos dois casos:

Caso 1: $x = j$**.** Nesse cenário, $M(i, j, k) = M(i - 1, j, k - 1)$. Isto é, o número de sequências distintas possíveis após i lançamentos, onde o último lançamento se deu na face j e houve k lançamentos consecutivos da face j nos últimos k lançamentos, é igual ao número de sequências distintas possíveis após $i - 1$ lançamentos, onde o último lançamento se deu na face j e houve $k - 1$ lançamentos consecutivos da face j nos últimos $k - 1$ lançamentos.

Esse caso só se aplica quando $k \leq d_j$, pois não é permitido ter mais do que d_j lançamentos consecutivos da face j. Caso contrário, $M(i, j, k) = 0$.

Caso 2: $x \neq j$. A face j é diferente da face x lançamento $i-1$. Nesse caso, $M(i,j,k)$ é a soma de todas as outras possibilidades de lançamentos para o lançamento $i-1$, considerando todas as faces $x \neq j$ e todos os número de ocorrências consecutivas da face x isto é:

$$M(i,j,k) = \sum_{\substack{x \in \{1 \ldots 6\} \\ x \neq j}} \sum_{l=1}^{d_x} M(i-1,x,l)$$

Casos bases. $M(1,i,1) = 1$, para toda face $1 \leq i \leq 6$, já que o problema garante que $d_i \geq 1$ para toda face i. E $M(1,i,k) = 0$, para toda face $1 \leq i \leq 6$ e $k > 1$, pois não é possível ter mais de um lançamento consecutivo da mesma face em apenas um lançamento.

2.1 Equação de Recorrência

Sintetizando em uma equação de recorrência, temos:

$$M(i,j,k) = \begin{cases} 1, & i = 1 \text{ e } k = 1 \\ 0, & i = 1 \text{ e } k > 1 \\ M(i-1,j,k-1), & i > 1 \text{ e } k > 1 \\ \displaystyle\sum_{\substack{x \in \{1 \ldots 6\} \\ x \neq j}} \sum_{l=1}^{d_x} M(i-1,x,l), & i > 1 \text{ e } k = 1 \end{cases}$$

2.2 Algoritmo Top-down

Considere que $M[i][j][k] = \perp$, isto é, a tabela de programação dinâmica é inicializada com um valor especial $\perp$ que indica que o valor ainda não foi computado. O Algoritmo 1 descreve uma abordagem top-down para computar $M(i,j,k)$.

A resposta final para o problema é dada por

$$\sum_{j=1}^{6} \sum_{k=1}^{d_j} M(n,j,k),$$

Ou seja, a soma de todas as sequências distintas possíveis após n lançamentos, considerando todas as faces j e todos os números de ocorrências consecutivas da face j.

2.3 Complexidade

Seja $k = \max\{d_i \mid 1 \leq i \leq 6\}$. Temos $\Theta(n \cdot 6 \cdot k) = \Theta(n \cdot k)$ estados na tabela de programação dinâmica, e cada estado é computado em $\Theta(6 \cdot k) = \Theta(k)$ tempo, resultando em uma complexidade total de $\Theta(nk^2)$ operações.

Algoritmo 1: COMPUTE(i, j, k)

```
1  if( i = 1 )
2      if( k = 1 )
3          M[i][j][k] ← 1
4      else
5          M[i][j][k] ← 0
6  if( M[i][j][k] ≠ ⊥ )
7      return M[i][j][k]
8  M[i][j][k] ← 0
9  if( k > 1 )
10     M[i][j][k] ← COMPUTE(i − 1, j, k − 1)
11 else
12     for( x ← 1 to 6 )
13         if( x ≠ j )
14             for( l ← 1 to d_x )
15                 M[i][j][k] ← (M[i][j][k]+ COMPUTE(i − 1, x, l))  mod 10^9 + 7
16 return M[i][j][k]
```

3 Variações

E se quiséssemos excluir uma das faces do dado, isto é, $d_i = 0$ para algum i. Bastaria colocar como condição: $M(i, j, k) = 0$ para toda face j com $d_j = 0$, conforme Algoritmo 2.

Algoritmo 2: COMPUTE(i, j, k)

```
1  if( i = 1 )
2      if( k = 1 )
3          M[i][j][k] ← 1
4      else
5          M[i][j][k] ← 0
6  if( M[i][j][k] ≠ ⊥ )
7      return M[i][j][k]
8  if( d_j = 0 )
9      M[i][j][k] ← 0
10     return M[i][j][k]
11 M[i][j][k] ← 0
12 if( k > 1 )
13     M[i][j][k] ← COMPUTE(i − 1, j, k − 1)
14 else
15     for( x ← 1 to 6 )
16         if( x ≠ j )
17             for( l ← 1 to d_x )
18                 M[i][j][k] ← (M[i][j][k]+ COMPUTE(i − 1, x, l))   mod 10^9 + 7
19 return M[i][j][k]
```