Determinando o Número de Soluções Não Negativas em Equações Diofantinas com Coeficientes Unitários

Pedro Olímpio N. de O. Pinheiro[1], **Wladimir Araujo Tavares**[1]

[1]Universidade Federal do Ceará - Campus Quixadá
Av. José de Freitas Queiroz, 5003 - Cedro, Quixadá - CE, 63900-000

pedro.olimpio@hotmail.com, wladimirufc@gmail.com

Abstract. *A diophantine equation is a polynomial equation that allows two or more variables to assume only integer values[Wikipedia 2017]. Equations with this characteristic can have infinite possible solutions, but if limited only to the non-negative integers, it makes the quantity of solutions finite and possible to be determined. It is possible to reduce the problem to one that can be solved by an application of counting techniques to determine the amount of anagrams of a word.*

Resumo. *Uma equação diofantina é uma equação polinomial que permite a duas ou mais variáveis assumirem apenas valores inteiros[Wikipedia 2017]. Equações com essa característica pode ter infinitas soluções possíveis, porém se limitado apenas aos inteiros não-negativos, torna a quantidade de soluções finitas e possíveis de serem determinadas. É possível reduzir o problema a um solucionável por uma aplicação de técnicas de contagem para determinar a quantidade de anagramas de uma palavra.*

1. Introdução

Nesse artigo, vamos apresentar uma solução para o seguinte problema: Dados inteiros N, C e M, determine quantas soluções inteiras não-negativas módulo M existem para a equação $x_1 + x_2 + \ldots + x_N = C$, onde $0 \leq x_i \leq C$ para todo i = $1, 2, \ldots, N$. Em [SPOJ 2017], o problema considera uma valor de M fixo igual a 1300031. Uma equação com coeficientes inteiros para as quais se busca apenas soluções inteiras são conhecidas como equações diofantinas. Equações desse tipo podem ter várias soluções possíveis, por exemplo, para $x_1 + x_2 = 3$ temos 4 valores válidos para (x_1, x_2): $(0, 3)$, $(1, 2)$, $(2, 1)$ e $(3, 0)$.

2. Solução

Os resultados obtidos estão sempre em módulo M. Porém são realizadas operações com esses resultados para obter outros resultados. O teorema da Equação 1 garante que as somas realizadas pelos algoritmos força bruta e programação dinâmica fornecem resultados corretos. O teorema da Equação 2 garante que as multiplicações realizadas para calcular os fatoriais dos números necessários e as realizadas para achar o valor da Equação 5 estão todas corretas.

$$\text{Se } a \equiv b \bmod M \text{ e } c \equiv d \bmod M \text{ então } a + c \equiv b + d \bmod M \tag{1}$$

$$\text{Se } a \equiv b \bmod M \text{ e } c \equiv d \bmod M \text{ então } a \cdot c \equiv b \cdot d \bmod M \tag{2}$$

Serão descritas 3 maneiras de descobrir quantas soluções diferentes existem para uma equação diofantina.

2.1. Força Bruta

O algoritmo força bruta consiste em testar para cada variável, todos os valores possíveis que ela pode assumir. A função $Diofatinas(i, C, M)$ devolve o número de soluções inteiras não-negativas mod M de $x_1 + \ldots + x_i = C$. A complexidade de tempo do algoritmo é $O(2^{N+C})$. O algoritmo utiliza de chamadas recursivas para encontrar o a quantidade de soluções de $x_1 + \ldots + x_{i-1} = C$, pois esses são valores válidos utilizando $x_i = 0$, e a quantidade de soluções para $x_1 + \ldots + x_{i-1} + x'_i = C - 1$, pois também são valores válidos para o problema original utilizando $x_i = x'_i + 1$. O pseudo-código para resolução força bruta é apresentado a seguir:

Algoritmo 1: Algoritmo força bruta

```
1 Function Diofantinas(i, C, M)
2     if i = 1 or C == 0 then
3         return 1;
4     end
5     return Diofantinas(i − 1, C, M) + Diofantinas(i, C − 1, M);
```

A complexidade pode ser reduzida aplicando uma programação dinâmica nesse algoritmo, passando a ser $O(C \cdot N)$.

2.2. Programação Dinâmica

O algoritmo de Programação Dinâmica explora duas características do problema: subestrutura ótima e a superposição de subproblemas. As soluções de subproblemas menores são utilizadas para a construção da solução de subproblemas maiores.

$$solution[i][j] = \text{ número de soluções para a equação } x_1 + x_2 + \ldots + x_i = j. \tag{3}$$

Para caso base, temos que para cada $1 \leq i \leq N : solution[i][0] = 1$, pois com qualquer número de variáveis, todas precisam ser 0 para somar 0, logo só existe uma maneira possível. Outro caso base é para cada $0 \leq j \leq C : solution[1][j] = 1$, pois qualquer valor escrito por uma variável pode somente ser representado com aquela variável assumindo o mesmo valor de j. Como são necessários valores da esquerda e de cima da tabela, seu preenchimento será feito da esquerda para a direita e de cima para baixo.

O problema tem a seguinte estrutura recursiva:

$$solution[i][j] = solution[i-1][j] + solution[i][j-1] \tag{4}$$

Algoritmo 2: Algoritmo usando Programação Dinâmica

```
1  begin
2      for i ← 1 to N do
3          solution[i][0] ← 1;
4      end
5      for j ← 0 to C do
6          solution[0][j] ← 1;
7      end
8      for i ← 2 to N do
9          for j ← 1 to C do
10             solution[i][j] ← (solution[i−1][j] + solution[i][j−1]) mod M;
11         end
12     end
13 end
```

Por exemplo, para a equação $x_1 + x_2 + x_3 = 5$ a tabela $solution$ assume os valores da Tabela 1.

Tabela 1. Tabela de Programação Dinâmica

	0	**1**	**2**	**3**	**4**	**5**
1	1	1	1	1	1	1
2	1	2	3	4	5	6
3	1	3	6	10	15	21

2.3. Combinatória

Outra maneira de resolver o problema é converte-lo em um problema de combinatória. São colocados c pontos e $n-1$ separadores, de modo que os pontos representam as unidades e a quantidade de pontos entre dois separadores o valor da variável correspondente, antes do primeiro separador equivale a x_1, entre o primeiro e o segundo a x_2 e assim consecutivamente. Por exemplo, a equação diofantina $x_1 + x_2 + x_3 = 7$ pode ser representada por:

$$\bullet\bullet \mid \bullet\bullet\bullet \mid \bullet\bullet$$

Nessa representação os valores de x_1, x_2 e x_3 são 2, 3, e 2 respectivamente. Assim o problema se reduz a calcular quantos anagramas existem para uma palavra com c pontos e $n-1$ separadores. Para calcular quantos anagramas uma palavra tem, se a palavra não tiver letras repetidas, apenas gere todas as permutações das letras, ou seja, existem $(tamanho\_da\_palavra)!$ anagramas. Porém, quando exite repetição de letras, todos os anagramas alterando apenas a posição daquelas letras repetidas são o mesmo anagrama, logo é necessário, para cada letra que se repete, dividir a quantidade de anagramas total pela quantidade de anagramas de palavras de tamanho igual a quantidade de vezes que a letra se repete. Na aplicação, o tamanho da palavra vale $C + N - 1$ e existem duas letras que se repetem: o $\bullet$ se repete C vezes e o $\mid$ se repete $N-1$ vezes, gerando a fórmula:

$$k = \frac{(C+N-1)!}{C! \cdot (N-1)!} \tag{5}$$

Para realizar a divisão em mod M é necessário calcular o inverso multiplicativo do denominador mod M, ou seja $(C! \cdot (N-1)!)^{-1} \bmod M$. O algoritmo de Euclides estendido fornece esse valor em $O(\log(M))$. É preciso também dos valores dos fatoriais de $(C+N-1)$, de C e de $(N-1)$. É possível obtê-los de forma eficiente aplicando uma programação dinâmica, conforme descrito no Algoritmo 3.

Algoritmo 3: Algoritmo para obter os fatoriais

```
1 begin
2 |   fatorial[0] ← 1;
3 |   for i ← 1 to (C + N − 1) do
4 |   |   fatorial[i] ← i · fatorial[i − 1] mod M;
5 |   end
6 end
```

Após a execução desse algoritmo, que tem complexidade $O(C+N)$, temos para qualquer i, tal que $0 \leq i \leq (C+N-1)$, o valor de $i!$ em $fatorial[i]$. Agora podemos obter o valor para $k \bmod M$ realizando $(C+N-1)! \cdot (C! \cdot (N-1)!)^{-1} \bmod M$. Portanto a complexidade para calcular o número possíveis de solução calculando a quantidade de anagramas tem complexidade $O(C+N+\log(M))$.

3. Conclusão

Concluímos que o problema de determinar quantas soluções possíveis para uma equação diofantina, com valores não negativos para as variáveis e coeficientes unitários, possui uma solução com complexidade de tempo $O(C+N)$ e complexidade de espaço $O(1)$, pois não é necessário guardar em memória todos os fatoriais até $C+N-1$, apenas os três que serão utilizados na fórmula: $C+N-1$, C e $N-1$.

Referências

Rosen, K. H. (2010). *Matemática Discreta e Suas Aplicações*. AMGH Editora LTDA.

SPOJ (2017). Spoj.com - problem diofanto. `http://br.spoj.com/problems/DIOFANTO/`.

Tavares, W. A. (2017). Marathoncode: Inverso multiplicativo. `http://marathoncode.blogspot.com.br/2012/04/inverso-multiplicativo.html?m=1`.

Wikipedia (2017). Equação diofantina - wikipédia, a enciclopédia livre. `https://pt.wikipedia.org/wiki/Equa%C3%A7%C3%A3o_diofantina`.