Trabalho apresentado no XXXVIII CNMAC, Campinas - SP, 2018.

Proceeding Series of the Brazilian Society of Computational and Applied Mathematics

O Algoritmo ShellSort e o Número de Frobenius

Raquel Marcolino de Souza[1]
Fabiano de Souza Oliveira[2]
Paulo Eustáquio Duarte Pinto[3]
Instituto de Matemática e Estatística, UERJ, Rio de Janeiro, RJ

ShellSort é um algoritmo de ordenação por comparação de um vetor V com n elementos que executa o algoritmo InsertionSort em diferentes subsequências de V [4]. Tais subsequências são definidas por uma *sequência de passos* $p_1, \ldots, p_k$ com $p_1 = 1$, $p_i < p_{i+1}$ para todo $1 \leq i < k$ e $p_k < n$. Diversos autores propuseram sequências especiais descritas em [1,2]. O ShellSort é descrito pelo Algoritmo 1, no qual InsertionSort(V, S) representa a ordenação por inserção da subsequência de V definida pela sequência de índices S.

Entrada: $V[1..n]$ de inteiros, sequência de passos $p_1, p_2, \ldots, p_k$
Resultado: V ordenado
para $j \leftarrow k$ **até** 1 **passo** -1 **faça**
  para $i \leftarrow 1$ **até** p_j **faça**
    InsertionSort(V, $S_{i,j}$), com $S_{i,j} = i, i + p_j, i + 2p_j, \ldots, i + \left\lfloor \frac{n-i}{p_j} \right\rfloor p_j$

Algoritmo 1: Algoritmo do ShellSort.

Considere o conjunto $A = \{a_1, \ldots, a_n\} \subset \mathbb{N}$, $|A| > 1$, tal que $a_1, \ldots, a_n > 1$ e $\text{MDC}(a_1, \ldots, a_n) = 1$. Seja $F(A) \subset \mathbb{N}$ tal que, para cada elemento $f \in F(A)$, não existem $k_1, \ldots, k_n \in \mathbb{N}$ de modo que $k_1 a_1 + \ldots + k_n a_n = f$, isto é, f não pode ser representado como uma combinação linear dos elementos do conjunto A. O *número de Frobenius*, denotado por $g(A)$, é o maior número de F. O problema de determinar $g(A)$ possui solução algébrica para $|A| = 2$, a saber, $g(\{a_1, a_2\}) = a_1 a_2 - a_1 - a_2$, mas sua solução algébrica é um problema em aberto quando $n \geq 3$. Contudo, elaboramos um algoritmo pseudo-polinomial como uma variação do algoritmo da mochila que é eficiente na prática. Tal algoritmo determina o número de Frobenius dado um limite superior $t \in \mathbb{N}$ para $g(A)$. Mais especificamente, dados $A = \{a_1, a_2, \ldots, a_n\}$ e $t \in \mathbb{N}$, o algoritmo determina $g(A, t) = \max\{f \in F(A) : f \leq t\}$, que é igual a $g(A)$ quando $t \geq g(A)$. No contexto do ShellSort, este algoritmo pode ser estendido para determinar também $g_m(A, t) = \max\{f \in F(A) : f \leq t, f \bmod m = 0\}$. Durante o processamento do passo p_s, pode ser mostrado que o número máximo de comparações para cada elemento do vetor neste passo é de $g_{p_s}(\{p_k, p_{k-1}, \ldots, p_{s+1}\}, n)$. Tal demonstração se baseia em mostrar

[1]raquelmarcolino25@gmail.com
[2]fabiano.oliveira@ime.uerj.br
[3]pauloedp@ime.uerj.br

que não ocorrem comparações de elementos cuja diferença de posições seja acima deste valor, pois os elementos já estariam ordenados entre si no processamento de algum dos passos anteriores. Portanto, se o limite superior do intervalo percorrido por um elemento durante a iteração relativa ao passo p_s é $g_{p_s}(\{p_k, p_{k-1}, \ldots, p_{s+1}\}, n)$, então o número de comparações máximo para cada elemento é $\lfloor g_{p_s}(\{p_k, p_{k-1}, \ldots, p_{s+1}\}, n)/p_s \rfloor + 1$. Com base nesta afirmação, é possível enunciar o Algoritmo 2, que determina o número máximo de comparações de qualquer sequência do ShellSort.

Entrada: Sequência de passos $p_1, p_2, \ldots, p_k$ e inteiro n
Resultado: Número máximo de comparações realizadas para este caso
ncomp $\leftarrow 0$
para $s \leftarrow k$ **até** 1 **passo** -1 **faça**
  limite $\leftarrow \lfloor g_{p_s}(\{p_k, p_{k-1}, \ldots, p_{s+1}\})/p_s \rfloor + 1$
  para $i \leftarrow 1$ **até** n **faça**
    limite$_i \leftarrow \lfloor (n-i)/p_s \rfloor$; ncomp $\leftarrow$ ncomp + min{limite, limite$_i$};
retorna *ncomp*

Algoritmo 2: Determinação do número máximo de comparações do ShellSort.

Como resultado, o algoritmo computa um limite superior para a complexidade de pior caso, em número de comparações, dados uma sequência de passos e o número n de elementos do vetor a ser ordenado. Este algoritmo é utilizado com o método empírico de se escolher diversos valores de n e de se determinar o limite superior para tais valores pelo Algoritmo 2. O resultado é analisado então com a ferramenta EMA [3] para estimar a complexidade assintótica correspondente a tal conjunto de pontos e mostrado na Tabela 1. Apesar de não haver uma garantia de que o pior caso assim obtido seja justo, ele assim se mostrou ao se comparar com os resultados conhecidos na literatura.

Tabela 1: Tabela de complexidades retornadas pelo EMA para cada sequência.

Autor, Ano	**Literatura**	**Empírico**
Shell, 1959	$\Theta(n^2)$	$O(n^2)$
Frank & Lazarus, 1960	$\Theta(n^{1,5})$	$O(n^{1,5})$
Hibbard, 1963	$\Theta(n^{1,5})$	$O(n^{1,5})$
Pratt, 1972	$\Theta(n \log^2 n)$	$O(n \log^2 n)$
Knuth, 1973	$\Theta(n^{1,5})$	$O(n^{1,5})$
Sedgewick, 1986	$O(n^{1,333\ldots})$	$O(n^{1,329})$
Gonnet & Baeza-Yates, 1991	em aberto	$O(n^{1,145})$, $O(n \log^3 n)$
Tokuda, 1992	em aberto	$O(n^{1,255})$, $O(n \log^{5,5} n)$

Referências

[1] M. Ciura, Best increments for the average case of shellsort. *International Symposium on Fundamentals of Computation Theory*, 2001.

[2] G. H. Gonnet, R. A. Baeza-Yates, Handbook of Algorithms and Data Structures: in Pascal and C. *Addison-Wesley*, 1991.

[3] F. S. Oliveira, `http://fabianooliveira.ime.uerj.br/ema`. Última vez acessado: 13 de Março de 2018.

[4] D. L. Shell, A high-speed sorting procedure. *Comm. of the ACM*, volume 2, 1959.