UNIVERSIDADE DO ESTADO DO AMAZONAS

ANÁLISE DO ALGORITMO DE ORDENAÇÃO SHELLSORT

Carlos Alves Lavor Neto
Guilherme Santos da Silva
Paulo Guilherme Ferreira de Siqueira

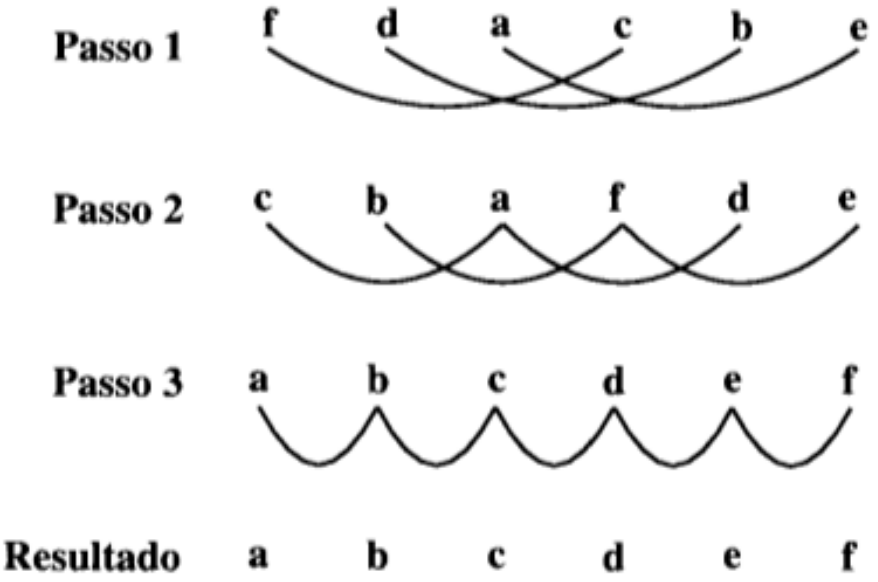
MANAUS - AM
05/2023

1. INTRODUÇÃO

Ordenação é o processo de arranjar um conjunto de valores semelhantes em ordem crescente ou decrescente. Muito embora os compiladores forneçam um método sort() para esse fim, é necessário analisar os diversos algoritmos de ordenação e onde especificamente eles têm uma eficácia maior que um método mais geral.
Existem muitos métodos de ordenação diferentes, e cada um deles podem ser comparados quanto a certos fatores que determinam a eficácia de um algoritmo de ordenação. Nesse documento analisaremos o Shellsort, um dos algoritmos derivados da ordenação por inserção, e o compararemos com outros algoritmos semelhantes.

2. SHELL SORT

A ordenação Shell é assim chamada devido ao seu inventor, D. L. Shell. Porém, o nome provavelmente pegou porque seu método de operação é frequentemente descrito como conchas do mar empilhadas umas sobre as outras. O método geral é derivado da ordenação por inserção e é baseado na diminuição dos incrementos.. Primeiro, todos os elementos que estão três posições afastados um do outro são ordenados. Em seguida, todos os elementos que estão duas posições afastados são ordenados. Finalmente, todos os elementos adjacentes são ordenados.



3. IMPLEMENTAÇÃO

Nesta seção segue um exemplo de implementação do Shellsort utilizando a linguagem C.

```
/* A ordenação Shell. */
void shell(char *item, int count)
{

  register int i, j, gap, k;
  char x, a[5];

  a[0]=9; a[1]=5; a[2]=3; a[3]=2; a[4]=1;

  for(k=0; k<5; k++) {
    gap = a[k];
    for(i=gap; i<count; ++i) {
      x = item[i];
      for(j=i-gap; x<item[j] && j>=0; j=j-gap)
        item[j+gap] = item[j];
      item[j+gap] = x;
    }
  }
}
```

4. ANÁLISE

Desde sua introdução, o Shellsort foi objeto de investigação por diversos autores. É importante notar que, para algumas sequências, a complexidade de tempo resultante foi mostrada ser $o(n^2)$, como por exemplo, aquelas estudadas em [Frank and Lazarus 1960] de tempo O(n3/2) e em [Incerpi and Sedgewick 1983] de tempo $O(n1+\sqrt{8}\cdot \ln (5/2)/ \ln (n))$. Também é importante notar que algumas destas sequências possuem

sua complexidade de pior caso exata desconhecida.

Tabela 1. Sequências de passos estudadas na literatura para o Shellsort.

Termo geral [Autor Ano]	Sequência	Pior caso
$\left\lfloor \frac{n}{2^k} \right\rfloor, k \geq 1$ [Shell 1959]	$\left\lfloor \frac{n}{2} \right\rfloor, \left\lfloor \frac{n}{4} \right\rfloor, ..., 1$	$\Theta(n^2)$ [Frank and Lazarus 1960]
$2^k - 1, k \geq 1$ [Hibbard 1963]	$1, 3, 7, 15, 31, 63, ...$	$\Theta(n^{3/2})$ [Pratt 1972]
$2^p 3^q, p \in \mathbb{N}$ e $q \in \mathbb{N}$ [Pratt 1972]	$1, 2, 3, 4, 6, 8, ...$	$\Theta(n \log^2 n)$ [Pratt 1972]
$a_k = 3 \cdot a_{k-1} + 1, k \geq 2$ e $a_1 = 1$ [Knuth 1973]	$1, 4, 13, 40, 121, ...$	$\Theta(n^{3/2})$ [Pratt 1972]
$\prod_{0<k<r \quad k \neq ((r^2+r)/2)-q} a_k$, onde $r = \left\lfloor \sqrt{2q + \sqrt{2q}} \right\rfloor$ e $a_k = \min\left(m \in \mathbb{N} : m \geq \left(\frac{5}{2}\right)^{k+1}, \forall p : 0 \leq p < k \Rightarrow \text{mdc}(a_p, m) = 1\right)$ [Incerpi and Sedgewick 1983]	$1, 3, 7, 21, 48, ...$	$O(n^{1+\sqrt{8 \ln(5/2)/\ln n}})$ [Incerpi and Sedgewick 1983]
$4^k + 3 \cdot 2^{k-1} + 1, k \geq 2$ [Sedgewick 1986]	$1, 8, 23, 77, 281, ...$	$O(n^{4/3})$ [Sedgewick 1986]
$a_k = \max\left(\left\lfloor \frac{5a_{k-1}}{11} \right\rfloor, 1\right), k \geq 2$ e $a_1 = n$ [Gonnet and Baeza-Yates 1991]	$\left\lfloor \frac{5n}{11} \right\rfloor, \left\lfloor \frac{5}{11} \left\lfloor \frac{5n}{11} \right\rfloor \right\rfloor, ..., 1$	em aberto
$\left\lceil \frac{9^k - 4^k}{5 \cdot 4^{k-1}} \right\rceil, k \geq 1$ [Tokuda 1992]	$1, 4, 9, 20, 46, ...$	em aberto
Sequência obtida empiricamente [Ciura 2001]	$1, 4, 10, 23, 57, 132, 301, 701, 1750$	em aberto

É importante notar que a análise do caso médio é desconhecida em quase toda a sua totalidade.

5. COMPARAÇÕES
   5.1 NÚMERO DE COMPARAÇÕES
   5.2 QUANTIDADE DE TROCAS OU MOVIMENTAÇÕES
   5.3 COMPLEXIDADE DE TEMPO

1. VETOR DE TAMANHO 100:

	BubbleSort			SelectionSort		
Tamanho do Vetor: 100	Desordenado	Ascendente	Desecendente	Desordenado	Ascendente	Desecendente
Quantidade de comparações	4797	99	4950	4950	4950	4950
Quantidade de trocas	2583	0	4950	99	99	99
Tempo de Execução	0,000020s	0,000001s	0,000021s	0,000013s	0,000012s	0,000012s

	InsertionSort			ShellSort		
Tamanho do Vetor: 100	Desordenado	Ascendente	Descendente	Desordenado	Ascendente	Descendente
Quantidade de comparações	2497	99	4950	334	503	260
Quantidade de trocas	2596	0	5049	837	0	763
Tempo de Execução	0,000011s	0,00000s	0,000021s	0,000007s	0,000003s	0,000003s

	MergeSort			QuickSort		
Tamanho do Vetor: 100	Desordenado	Ascendente	Descendente	Desordenado	Ascendente	Descendente
Quantidade de comparações	544	356	316	660	4950	4950
Quantidade de trocas	672	672	672	410	5049	2549
Tempo de Execução	0,000009s	0,000004s	0,000004s	0,000005s	0,000021s	0,000016s

2. VETOR DE TAMANHO 1000:

	BubbleSort			SelectionSort		
Tamanho do Vetor: 1000	Desordenado	Ascendente	Desecendente	Desordenado	Ascendente	Desecendente
Quantidade de comparações	502965	999	49990629	504450	499500	499500
Quantidade de trocas	265624	0	499500	1098	999	999
Tempo de Execução	0,001667s	0,00003s	0,001874s	0,001269s	0,001101s	0,001079s

	InserctionSort			ShellSort		
Tamanho do Vetor: 1000	Desordenado	Ascendente	Descendente	Desordenado	Ascendente	Descendente
Quantidade de comparações	254895	999	499500	6833	8006	4700
Quantidade de trocas	255993	0	500499	15342	0	12706
Tempo de Execução	0,00949s	0,000002s	0,001678s	0,000109s	0,000031s	0,000037s

	MergeSort			QuickSort		
Tamanho do Vetor: 1000	Desordenado	Ascendente	Descendente	Desordenado	Ascendente	Descendente
Quantidade de comparações	8987	5044	4932	16421	499500	499500
Quantidade de trocas	10648	9976	9976	8489	500499	250499
Tempo de Execução	0,000090s	0,000053s	0,000046s	0,000076s	0,002334s	0,001484s

3. VETOR DE TAMANHO 10000:

	BubbleSort			SelectionSort		
Tamanho do Vetor: 10000	Desordenado	Ascendente	Desecendente	Desordenado	Ascendente	Desecendente
Quantidade de comparações	49990629	9999	49995000	49995000	49995000	49995000
Quantidade de trocas	24945632	0	49995000	9999	9999	9999
Tempo de Execução	0,211105s	0,000025s	0,183301s	0,105982s	0,104174s	0,103819s

	InserctionSort			ShellSort		
Tamanho do Vetor: 10000	Desordenado	Ascendente	Desecendente	Desordenado	Ascendente	Desecendente
Quantidade de comparações	24902822	9999	49995000	134940	120005	62560
Quantidade de trocas	24912821	0	5000499	254945	0	182565
Tempo de Execução	0.055177s	0,000027s	0.105336s	0,001404s	0,000303s	0,000519s

	MergeSort			QuickSort		
Tamanho do Vetor: 10000	Desordenado	Ascendente	Descendente	Desordenado	Ascendente	Descendente
Quantidade de comparações	120528	69008	64608	170055	49995000	49995000
Quantidade de trocas	133616	133616	133616	114840	50004999	25004999
Tempo de Execução	0,001147s	0,000754s	0,000587s	0,000973s	0.180859s	0.140030s

5.4 ESTABILIDADE
5.5 IN-PLACE

	Bubble Sort	Selection Sort	Insertion Sort	Shell Sort	QuickSort	Merge Sort
Estabilidade	sim	não	sim	não	não	sim
In-place	sim	sim	sim	não	sim	não

6. CONCLUSÃO

É levemente complicado perceber que o ShellSort conduz a bons resultados ou até mesmo que consiga ordenar o vetor por conta de sua natureza de complexidade desconhecida. Porém foi mostrado que ele executa bem todas as funções e que tem uma eficiência maior que os métodos de ordenação mais simples. Cada passo da ordenação envolve relativamente poucos elementos ou elementos que já estão razoavelmente em ordem, logo, a ordenação Shell é eficiente e cada passo aumenta a ordenação dos dados.

A sequência exata para os incrementos pode mudar. A única regra é que o último incremento deve ser 1. É importante evitar sequências que são potências de 2 — por razões matemáticas complexas, elas reduzem a eficiência do algoritmo de ordenação (mas a ordenação ainda funciona).