

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

117536 - Projeto e Análise de Algoritmos
Turma: B

Relatório sobre Complexidade do Bubblesort

Camila F. T. Pontes - 15/0156120
Diogo C. Ferreira - 11/0027931

2 de dezembro de 2019

1 Introdução

O problema de ordenação de sequências numéricas surge frequentemente em aplicações computacionais. Este problema pode ser definido formalmente da seguinte maneira [1]:

- **Entrada:** uma sequência de n números, a_1, a_2, \dots, a_n
- **Saída:** uma permutação (reordenação) da sequência de entrada, a'_1, a'_2, \dots, a'_n tal que $a'_1 \leq a'_2 \leq \dots \leq a'_n$

Existem diversos algoritmos de ordenação que resolvem o problema apresentado acima, *e.g.* *insertion sort*, *selection sort*, *merge sort*, *quicksort*, dentre outros. Neste trabalho, vamos analisar um dos algoritmos de ordenação

mais simples, o *bubblesort*. A ideia geral do *bubblesort* é percorrer diversas vezes o vetor de entrada e, a cada passagem, mover para o final da porção ainda não ordenada o maior elemento. Uma implementação recursiva desse algoritmo é apresentada a seguir (Algoritmo 1).

Algorithm 1 Implementação recursiva do *bubblesort*

```

1: function BUBBLESORT(int array  $A$ , int  $n$ )
2:   if  $n = 1$  then
3:     return
4:   end if
5:   for  $i \leftarrow 0$  to  $n - 1$  do
6:     if  $A[i] > A[i + 1]$  then
7:       swap( $A[i]$ ,  $A[i + 1]$ )
8:     end if
9:   end for
10:  bubblesort( $A, n - 1$ )
11: end function

```

Uma das formas de comparar o desempenho do *bubblesort* com o desempenho de outros algoritmos de ordenação é através de uma análise de complexidade. A complexidade do *bubblesort* é de ordem quadrática ($O(n^2)$), *i.e.*, no pior caso, são feitas n^2 trocas durante a ordenação, onde n é o número de elementos do vetor de entrada. Neste trabalho, um assistente de prova será utilizado para provar a complexidade do *bubblesort*. A prova será realizada utilizando o *Prototype Verification System* (PVS) [2].

Os principais objetivos deste trabalho são:

- Implementar uma versão recursiva do *bubblesort* com um contador para o número de trocas realizadas durante a ordenação;
- Provar a complexidade assintótica do *bubblesort* utilizando o PVS.

2 Apresentação do Problema

2.1 Análise assintótica

2.1.1 Funções auxiliares

Para a realização da análise assintótica do Bubblesort, inicialmente foram definidas três funções auxiliares com o objetivo de rastrear as contagens do número total de comparações realizadas pelo algoritmo após retornar a lista ordenada.

- `bubbling_count`: recebe uma lista l , um natural n (equivalentes aos parâmetros da função `bubbling` original) e um contador $count$, que é incrementando quando alguma comparação for realizada. Seu valor de retorno é o par $(l, count)$ com a lista e o contador devidamente atualizados.
- `bubblesort_aux_count`: recebe uma lista l , um natural n (equivalentes aos parâmetros da função `bubblesort_aux` original) e um contador $count$, que é passado para a função `bubbling_count` chamada internamente. Seu valor de retorno é o par $(l, count)$ com a lista e o contador devidamente atualizados.
- `bubblesort_count`: recebe uma lista l , um natural n , equivalentes aos parâmetros da função `bubblesort` original. Ela chama `bubblesort_aux_count` da mesma forma que `bubblesort` chama `bubblesort_aux`, mas com o parâmetro do contador iniciando em 0. Seu valor de retorno é o par $(l, count)$ com a lista ordenada e o contador atualizados com o número total de comparações realizadas pelo algoritmo.

2.1.2 Lemas

Lemas utilizados na prova da complexidade de `bubbling_count`:

- `bubbling_equiv`: afirma que `bubbling` e `bubbling_count` são equivalentes
- `bubbling_length`: afirma que a função `bubbling_count` não altera o tamanho da lista de entrada

- `bubbling_counts_n`: afirma que `bubbling_count` realiza exatamente n comparações, onde n é o tamanho da lista de entrada, e que, portanto, sua complexidade é linear

Lemas utilizados na prova da complexidade de `bubblesort_aux_count`:

- `bubblesort_aux_equiv`: afirma que `bubblesort_aux` e `bubblesort_aux_count` são equivalentes
- `bubblesort_counts_n2`: afirma que `bubblesort_aux_count` realiza exatamente $n(n+1)/2$ comparações, onde n é o tamanho da lista de entrada, e que, portanto, sua complexidade é quadrática

Lemas utilizados na prova da complexidade de `bubblesort_count`:

- `bubblesort_equiv`: afirma que `bubblesort` e `bubblesort_count` são equivalentes
- `bubblesort_counts_n2`: afirma que `bubblesort_count` realiza exatamente $n(n-1)/2$ comparações, onde n é o tamanho da lista de entrada, e que, portanto, sua complexidade é linear

3 Conclusão

Neste trabalho, foi realizada uma prova assistida da complexidade quadrática do algoritmo de ordenação *bubblesort* utilizando o PVS. A prova foi realizada de forma indireta, mostrando que a contagem do número de trocas realizadas por uma implementação recursiva do algoritmo *bubblesort* é igual a uma função quadrática do tamanho da lista de entrada. Além disso, foi mostrado que a função *bubblesort* é equivalente à sua versão com adição do contador do número de trocas. Portanto, concluímos que o algoritmo *bubblesort* possui complexidade assintótica $O(n^2)$.

Referências

- [1] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

- [2] Sam Owre, John M Rushby, and Natarajan Shankar. Pvs: A prototype verification system. In *International Conference on Automated Deduction*, pages 748–752. Springer, 1992.