Universidade de Brasília
Instituto de Ciências Exatas

Departamento de Ciência da Computação

117536 - Projeto e Análise de Algoritmos
Turma: B

Relatório sobre **Análise da Complexidade do Bubblesort**

Eduardo de Azevedo dos Santos - 14/0136967
Rafael Silva de Alencar - 13/0130834

5 de dezembro de 2019

1 Introdução

Algoritmos estão presentes nas mais diversas implementações computacionais e sua eficiência é determinante para que sejam utilizáveis por desenvolvedores. Algoritmos ineficientes apresentam alto grau de complexidade, e na maior parte das vezes, seus usos são inviáveis para para aplicações reais. Assim sendo, a análise de algoritmos é de fundamental importância para a computação, pois, por meio de formalismos teóricos, programas são mensurados a fim de se obter um resultado que revela o grau de sua eficiência. Implementações que envolvem o uso de algoritmos necessitam que estes sejam corretos e sua complexidade fundamentalmente deve ser conhecida.

Dentre os problemas clássicos na computação que envolvem análise algorítimica, há a ordenação de números naturais, onde diversos algoritmos com complexidades diferentes resolvem a problema da ordenação. São exemplos importantes de avaliação de complexidade, onde estudades de ciência da computação são introduzidos a análise de algoritmos.

O objetivo deste projeto é apresentar uma análise da complexidade do algoritmo de ordenação *bubblesort* atráves de um sistema de verificação de provas formais. No caso, o sistema a ser utilizado é o PVS (*Prototype Verification System*).

2 Apresentação do Problema

O *bubblesort* é um dos algoritmos de ordenação mais simples e tradicionais da Ciência da Computação. Sua proposta é, como o próprio nome sugere, executar o "borbulhamento"de pequenas chaves da direita para a esquerda de um vetor. O algoritmo começa sua execução pegando uma chave do fim do vetor e movendo ela para a esquerda enquanto é menor que qualquer chave que passe por. Uma vez que passe por uma chave menor, o algoritmo troca para esta chave, a movendo para o lado esquerdo do vetor. Dessa maneira, a cada iteração, as menores chaves se movem para a esquerda até sua posição final, e outras chaves são lentamente movidas para a esquerda. Sendo assim, as partes não ordenadas do vetor se tornam mais ordenadas na medida que o *bubblesort* continua.[1]

2.1 Análise assintótica

A complexidade de tempo do *bubblesort* pode ser analisada com um olhar mais profundo em sua implementação. Seja o seguinte pseudo-código uma implementação do algoritmo:

```
procedure Bubblesort(L):
    tamanho = L.tamanho;

    for i=0 to tamanho - 1 do:
        trocou = false;
        for j=0 to tamanho - 1 - i do:
            if L[j] > L[j+1] then
                troca(L[j], L[j+1]);
                trocou = true;
            end
```

```
        end

        if not trocou then
            break;
        end
    end
end procedure return L
```

Ao observar o código acima, é fácil perceber que o pior caso do tempo de execução é igual a $O(tamanho^2)$. No entanto, uma análise mais aprofundada pode ser necessária para o entendimento do fato.

Uma execução do algoritmo pode ser observada a seguir. Seja a lista L=[4,5,9,1,3]. Ao rodarmos o algoritmo sobre a lista L, temos as seguintes iterações:

```
//Primeira iteracao do for exterior
[4,5,9,1,3]
[4,5,9,1,3]
[4,5,1,9,3]
[4,5,1,3,9]
//Segunda iteracao do for exterior
[4,5,1,3,9]
[4,1,5,3,9]
[4,1,3,5,9]
//Terceira iteracao do for exterior
[1,4,3,5,9]
[1,3,4,5,9]
```

Olhando o código e o comportamento do exemplo acima, pode-se perceber que, no pior caso, o *for* exterior roda uma quantidade de vezes igual ao tamanho da lista de entrada. O laço interno, no entanto, roda $n-1-0$ vezes na primeira iteração, onde n é o tamanho da lista de entrada. Na segunda, $n-1-1$ vezes. Na $n-1$-ésima iteração, roda $n-1-(n-1)$ vezes. Ou seja, no total, o bloco roda $\sum_{i=0}^{n-1}(n-i-1)$ vezes. Provamos a complexidade do *bubblesort* da seguinte maneira:

$$\sum_{i=0}^{n-1}(n-i-1) = n^2 + \sum_{i=0}^{n-1}(i) + n \tag{1}$$

Sabendo do fato que $\sum_{j=1}^{m} j = m(m+1)/2$, substituimos $m = n - 1$, o que nos permite dizer que:

$$n^2 - n(n-1)/2 - n = n^2 - n^2/2 + n/2 - n = (n^2 - n)/2 \tag{2}$$

A partir disso, podemos então concluir que a complexidade do pior caso do tempo de execução do *bubblesort* é $O(n^2)$.

Nem sempre a prova da complexidade de um algorítmo feita em papel e lápis é precisa devido a erros que podem ocorrer no processo. Para que esse erros sejam evitados, a utilização de um assistente de prova como o PVS se faz necessário. Como o próprio nome sugeri, o objetivo dessa ferramenta é auxiliar na formalização de provas, garatindo um grau de confiabilidade ainda maior.

Neste projeto a complexidade do algorítmo de ordenação *bubblesort* foi formalizada através do PVS. Para tal realização se fez necessário a criação de lemas auxilares para a resolução. O lema principal segue abaixo:

```
bublesort_is_quadratic: LEMMA
 FORALL (l:list[nat]): member(LAMBDA(n:nat):
  cbubblesort(l)` 2, Omicron(LAMBDA(n:nat):length(l)^ 2
```

Este lema basicamente estabelece que a complexidade do *bubblesort* é $O(n^2)$. A ideia principal para prová-lo é utilizar indução na estrutura. Posteriormente, um lema auxiliar se fez necessário para continuar a formalização da prova.

```
count_bubblesort_ws_general: LEMMA
 FORALL (l:list[nat]): cbubblesort(l)` 2 <= (length(l)
   * (length(l) - 1))/2
```

Este lema esbalece que o *bubblesort* faz exatamente

$$(n^2 - n)/2 \tag{3}$$

comparações, onde n é o tamanho do comprimento do tamanho da lista de entrada, e assim tem complexidade quadrada.

2.2 Correção da solução proposta

A ideia por trás da correção de um algoritmo é provar que de fato ele faz aquilo que se propõe a fazer. No caso do *bubblesort*, parte da prova da correção consiste em provar que o **bubblesort** gera uma lista ordenada para qualquer lista de tamanho arbitrário dado como entrada:

```
bubblesort_sorts: LEMMA
 FORALL (l:list[nat]): sorted?(bubblesort(l))
```

A prova da corretude neste projeto não foi totalmente concluída. Mas foi realizado uma tentativa de prova parcial.
O comprimento da cauda de uma lista é o resultado do comprimento da cauda menos um:

```
length_cdr: LEMMA
 FORALL(l) length(l) > 0 IMPLIES length(l) - 1 =
  length(cdr(l))
```

A prova é muito simples, não é necessario aplicação de indução, apenas alguns comandos de simplificação de fórmulas no PVS como *skeep* e expansão da função *length*.

É possível notar que a função de borbulhamento *bubbling* preserva o tamanho da lista dado como entrada:

```
bubbling_preserves_length: LEMMA
 FORALL (n:nat) FORALL (l): n < length(l) IMPLIES
   n < length(bubbling(l,n))
```

A parte principal da prova consiste em aplicar indução na estrutura da lista (induct "n"). Posteriormente o lema **length_cdr** é utilizado para dar prosseguimento a prova.

3 Conclusão

O algoritmo de ordenação *bubblesort* é correto, possuindo um tempo de execução no pior caso igual a $O(n^2)$. A análise do algoritmo, assim como a prova de sua corretude pelo programa *PVS* se mostraram um desafio para estudantes de graduação, mas também uma oportunidade de aprendizado muito valiosa. A partir deste projeto, o entendimento sobre análise assintótica de algoritmos possuido pelos estudantes aumentou, assim como sobre o funcionamento e uso da ferramenta *PVS*.

Referências

[1] P. Biggar and D. Gregg, "Sorting in the presence of branch prediction and caches," *Technical Report TCD-CS-2005-57*, 2005.