

Lógica Computacional 117366  
Descrição do Projeto  
Formalização de Propriedades do Algoritmo *Radix Sort*  
03 de Abril de 2019  
Prof. Mauricio Ayala-Rincón & Flávio L. C. de Moura

**Participação do bolsista de doutorado:**  
Thiago Mendonça Ferreira Ramos  
thiagomendoncaferreiraramos@yahoo.com.br

**Observação:** Os laboratórios do LINF têm instalado o *software* necessário para o desenvolvimento do projeto (PVS 6.0 com as bibliotecas PVS da NASA).

## 1 Introdução

Algoritmos de ordenação são fundamentais em ciência da computação. Suponha que se deseja ordenar cartões alfabeticamente sabendo que existe um nome gravado em cada cartão. Poderíamos separar estes cartões em 26 pilhas, uma para cada letra, ordenar cada pilha separadamente segundo algum método de ordenação, e depois combinar as pilhas ordenadas. Caso o nome gravado nos cartões sejam números de 5 dígitos decimais, poderíamos separá-los em 10 pilhas de acordo com o primeiro dígito. O problema é que ao ordenarmos os números de 5 dígitos da esquerda para a direita, *i.e.* do dígito mais significativo para o menos significativo, exigiria um adequado manejo dos números para evitar que cada passo desorganize o que tinha sido ordenado no passo anterior.

O que ocorre se fizermos este processo do dígito menos significativo para o mais significativo?

Por exemplo, dado o vetor

132		221		413		123
123		341		221		132
323	primeiro ordena-	132	em segunda, as	123	por fim, ordena-	221
413	mos a coluna das	123	dezenas:	323	mos a coluna das	323
221	unidades:	323		132	centenas:	341
341		413		341		413

Este processo sempre funciona? No segundo semestre de 2018 foi estabelecida a correção de um passo intermediário necessário para responder afirmativamente a esta pergunta. Este passo intermediário consiste em utilizar um algoritmo **estável** como algoritmo auxiliar a ser aplicado em cada passo. No exemplo acima, o algoritmo auxiliar é responsável por ordenar uma coluna. Um algoritmo é dito estável se a posição relativa de dois elementos iguais permanece inalterada durante o processo de ordenação. Nesse projeto utilizamos o algoritmo *merge sort* como algoritmo auxiliar estável. Exemplos de algoritmos não estáveis são *quicksort*, *heapsort* e *binary-insertion sort*.

Algoritmos que ordenam segundo o critério apresentado no exemplo acima são conhecidos como *radix sort* e podem ser utilizados quando se conhece algo sobre a estrutura ou o intervalo de variação das chaves a serem ordenadas. Esses algoritmos também podem ser considerados algoritmos de classificação; p.ex., classificação de cartas de um baralho por seus naipes [copas, espadas, paus e ouros] e números [A, 2-10, J, Q, K]; classificação de registros de funcionários por seu sexo, data de contratação e salário, etc. *Radix sort* foi originalmente utilizado pelas máquinas que ordenavam cartões perfurados (que não existem mais atualmente).

O objetivo do projeto é demonstrar formalmente a correção de *radix sort*. Para as provas de correção serão aplicadas técnicas dedutivas da lógica de predicados, implementadas no assistente de demonstração PVS, como descrito em [AdM17].

Descrições detalhadas de algoritmos de ordenação e classificação, entre eles *radix sort*, podem ser encontradas em livros texto [CLRS01, BvG99, Knu73]. Formalizações em PVS de diversos algoritmos de ordenação sobre os naturais acompanham o livro [AdM17] e estão disponíveis na Internet. Essas formalizações também foram estendidas para espaços de medida abstratos e estão disponíveis na biblioteca de PVS de NASA LaRC.

## 2 Descrição do Projeto

Com base na *teoria sorting* especificada na linguagem do assistente de demonstração PVS ([pvs.csl.sri.com](http://pvs.csl.sri.com), executável em plataformas Unix/Linux e OSX), os alunos deverão formalizar propriedades de uma especificação do algoritmo *radix sort*. O arquivo com as questões é denominado `radix_sort`.

O algoritmo está especificado como abaixo, sendo a segunda função `radixsort` a função principal, que toma uma lista `l` de naturais e a ordena. A primeira função `radixsort` ordena recursivamente uma lista de naturais por dígitos `dth` até `kth`, utilizando a função `merge_sort`, que ordena de forma estável os números na lista por seu `dth` dígito.

```
radixsort(l : list[nat], k : nat, d : nat | d <= k) : RECURSIVE list[nat] =
  IF d = k THEN merge_sort(l,d)
  ELSE radixsort(merge_sort(l,d), k, d+1)
  ENDIF
  MEASURE k - d
```

```
radixsort(l : list[nat]) : list[nat] =
  IF length(l) <= 1 THEN l
  ELSE radixsort(l, max_digits(l), 0)
  ENDIF
```

### 2.1 Questões

A primeira questão, consiste em demonstrar que se uma lista está ordenada até o  $d$ -ésimo dígito, e se aplica `radixsort` entre os dígitos  $d$  e  $k$ , então ficará ordenada até o dígito  $k + 1$ :

```
radix_sort_d_sort : CONJECTURE
  FORALL(l : list[nat], k : nat, d:nat | d <= k) :
    is_sorted_ud?(l,d) =>
      is_sorted_ud?(radixsort(l, k, d), k+1)
```

A segunda questão, está relacionada com o fato de que a função `radixsort` preserva os elementos das listas dadas como argumento:

```
radixsort_permutes : CONJECTURE
  FORALL(l : list[nat], k : nat, d : nat | d <=k):
    permutations(l, radixsort(l,k,d))
```

Finalmente, a terceira questão é demonstrar que função `radixsort` gera uma lista ordenada:

```
radixsort_sorts : CONJECTURE
FORALL(l : list[nat]):
  is_sorted?(radixsort(l))
```

### 3 Etapas do desenvolvimento do projeto

Os alunos deverão definir grupos de trabalho limitados a **três** membros até o dia 10 de abril de 2019.

O projeto será dividido em duas etapas como segue:

- Verificação das Formalizações. Os grupos deverão ter prontas as suas formalizações na linguagem do assistente de demonstração PVS e enviar via e-mail para o professor os arquivos de especificação e de provas desenvolvidos (`radixsort.pvs` e `radixsort.prf`) até o dia **10.06.2019**, 8:00am. Na semana de **10-12.06.2019**, durante os dias de aula, realizar-se-á a verificação do trabalho para a qual os grupos deverão, em acordo com o professor, determinar um horário (de 30 minutos) no qual **todos os membros do grupo deverão comparecer**.

**Avaliação (peso 6.0):**

- Um dos membros, selecionado por sorteio, explicará os detalhes da formalização em máximo 10 minutos.
  - Os quatro membros do grupo poderão complementar a explicação inicial em máximo 10 minutos.
  - A formalização será testada nos seguintes 10 minutos.
- Entrega do Relatório Final.  
**Avaliação (peso 4.0):** Cada grupo de trabalho devera entregar um Relatório Final inédito, editado em  $\text{\LaTeX}$ , limitado a oito páginas (12 pts, A4, espaçamento simples), do projeto até o dia **17.06.2019** com o seguinte conteúdo:
    - Introdução e contextualização do problema.
    - Explicação das soluções.
    - Especificação do problema e explicação do método de solução.
    - Descrição da formalização.
    - Conclusões.
    - Lista de referências.

### Referências

- [AdM17] M. Ayala-Rincón and F. L. C. de Moura. *Applied Logic for Computer Scientists - Computational Deduction and Formal Proofs*. Undergraduate Topics in Computer Science. Springer, 2017.
- [BvG99] S. Baase and A. van Gelder. *Computer Algorithms — Introduction to Design and Analysis*. Addison-Wesley, 1999.

- [CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Electrical Engineering and Computer Science Series. MIT press, second edition, 2001.
- [Knu73] D. E. Knuth. *Sorting and Searching*, volume Volume 3 of *The Art of Computer Programming*. Reading, Massachusetts: Addison-Wesley, 1973. Also, 2nd edition, 1998.