Projeto Radix Sort (2019/1)

Italo Marcos Brandão - 18/0147358
Victor A. de Carvalho - 16/0147140

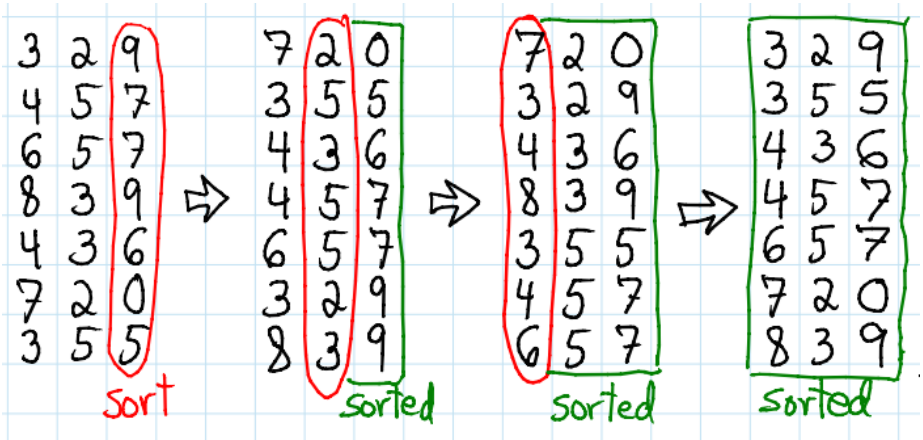
17 de junho de 2019

1 Introdução e contextualização do problema

Este relatório tem como finalidade contribuir e esquematizar todo o estudo que realizamos sobre os algoritmos *radix sort* e *merge sort* bem como estruturar conceitualmente toda a nossa estratégia de prova dos algoritmos propostos nesse projeto utilizando o sistema de verificação PVS(*Specification and Verification System*) para provar os teoremas.

1.1 *Radix Sort*

O algoritmo *Radix Sort* é um algorítmo de ordenação que ordena os números de acordo com as casas decimais. Comecando pelas unidades, dezenas, centenas e assim por diante como mostrado na imagem abaixo.



2 Explicação das soluções

2.1 Questão 1

O objetivo da questão é provar que uma lista de naturais ordenada até o d-ésimo (que está ordenada do d-ésimo ao k-ésimo dígito), estará ordenada para o próximo dígito (dígito k+1), usando indução forte.

2.1.1 Base Indutiva - Caso não-recursivo

O *radixsort* opera de forma recursiva do dígito **d** ao **k**, e na última chamada **(d = k)** usa *merge sort* na lista. O caso-base consiste da aplicação do *merge sort* na lista **d** e devemos provar que a lista ordenada pro dígito **d+1**, se mantém ordenada pro **d**. Pelo lema *"merge sort d sorts"*, está definido que listas ordenadas por *merge sort* pelo seu d-ésimo dígito estarão ordenadas até o *i-ésimo* **(d+1)** dígito, ou seja, que a ordenação é mantida, sendo assim a prova pro caso-base está concluída, pois o lema diz exatamente o que queremos provar.

2.1.2 Passo Indutivo - Caso recursivo

O caso recursivo segue o mesmo raciocínio, porém, indução forte requer aplicação de prova para os casos anteriores, como na proposição: se vale para **k**, vale para **k+1**. Ou, de forma análoga: "se vale para **k-1** vale para *k*". Porém, estamos tratando da indução em **k-d**, logo, a proposição se torna *"se vale para k-d-1, vale para k-d"*. A expressão **"k-d-1"** pode ser reescrita como **k-(d+1)**. A proposição "se vale para k-(d+1), vale para k-d"implica [1] que se a ordenada por *merge sort* até o dígito d se mantém ordenada para o d+1 [2], a lista ordenada por *radix sort* também estará ordenada [3]. Logo, provando que vale para k-(d+1) a prova está concluída. O mesmo lema do caso anterior (merge sort d sorts) nos dá a resposta. Note que o caso "d+1"do k-(d+1) foi provado na base de indução, assim como a proposição, "uma lista ordenada até d por merge sort, está ordenada pra d+1". Logo, temos por hipótese que o antecedente [2] vale, então, pro consequente [3] também irá valer, e a prova do passo indutivo está concluída.

2.2 Questão 2

O objetivo da questão é provar que a função *radix sort* preserva os elementos das listas dadas como argumento, usando indução forte.

2.2.1 Base Indutiva - Caso não-recursivo

O *radixsort* opera de forma recursiva do dígito **d** ao **k**, e na última chamada **(d = k)** usa *merge sort* na lista. O caso-base consiste em provar que a aplicação do *merge sort* em uma lista vai **preservar** (permutar) os elementos já ordenados. Pelo lema *"merge sort permutes"*, está definido que uma lista é a permutação dela mesma após ser ordenada por *merge sort*, ou seja, o *merge sort* não altera os elementos já ordenados, sendo assim a prova pro caso-base está concluída, pois o lema diz exatamente o que queremos provar.

2.2.2 Passo Indutivo - Caso recursivo

O caso recursivo segue o mesmo raciocínio. Assim como na questão anterior temos a implicação: *"se vale para k-(d+1), vale para k-d"*, porém, agora a porposição implica que, uma lista previamente ordenada é permutação da mesma lista após ela ser ordenada por *radixsort*. Logo, provando que vale para k-(d+1) a prova está concluída. Temos os sequentes:

Lista de sequentes:
[1]: permutations(l, merge_sort). Uma lista é permutação de si quando ordenada por *merge sort*. Esse sequente é o caso-base e hipótese de indução.
[2]: permutations(l, radix_sort). Uma lista é permutação de si quando ordenada por *radix sort*. Esse sequente é o caso indutivo e sequente que queremos provar.
[3]: permutations(merge_sort, radix_sort).Uma lista ordenada por *merge sort* é permutação de si quando reordenada por *radix sort*. Esse sequente corresponde à aplicação da hipótese de indução.

Podemos instanciar a lista *l* do sequente **[2]** como *merge sort*, para aplicar a hipótese de indução, criando assim o sequente **[3]**. Usamos a hipótese aplicando novamente o lema *"merge sort permutes"*. Então temos agora que a permutação de *l* e *merge sort* em *l* são equivalentes pelo sequente **[1]**, e após a instanciação temos também que as permutações de uma lista ordenada por *merge sort* e ordenada por *radix sort* são equivalentes, como no sequente **[3]**.

Queremos provar que o *radix sort* mantém a lista ordenada. Por hipótese de indução, uma lista ordenada por *merge sort* preserva os elementos, e após aplicar *radix sort* na mesma lista os elementos também se preservam (sequente [3]), então, *radix sort* preservou a lista, e *merge sort* também preservou, o que implica que a lista inicial *l* se manteve preservada após a aplicação de ambos os algoritmos. O lemma *"permutations is transitive"* faz a conexão (ou transitividade) entre os sequentes. Logo, uma lista *l* ordenada por *merge sort* se mantém preservada, e essa lista (já ordenada por *merge sort*) continuará ordenada após aplicar *radix sort*. Logo, a lista inicial *l* preserva seus elementos após usar o *radix sort* por último. A aplicação do lema *"permutations is transitive"* equivale à regra **Silogismo Hipotético** (Anexo 1), de Dedução Natural. Considere as premissas $P \rightarrow Q$ como o sequente [1], $Q \rightarrow R$ como o sequente [3], e a inferência $P \rightarrow R$ como o sequente [2]. E assim a prova do passo indutivo está concluída.

3 Anexos

Anexo 1: Regra Silogismo Hipotético.

$$P \Rightarrow Q\,, Q \Rightarrow R \equiv P \Rightarrow R$$

4 Conclusões

O algoritmo *radix sort* é consistente na tarefa de ordenar elementos, no sentido de que foi provado por indução (e usando lemas definidos previamente) que os elementos já ordenados são preservados após cada passo de ordenação, usando o *merge sort* como algoritmo auxiliar nos casos mais simples.

5 Lista de referências

M. Ayala-Rincon and F. L. C. de Moura.Applied Logic for Computer Scientists - Computational Deduction and Formal Proofs. Undergraduate Topics in Computer Science. Springer, 2017.

N. Shankar, S. Owre, J. M. Rushby and D. W. J. Stringer-Calvert. PVS Prover Guide Version 2.4. SRI International, 2001.

Robert J. Chassell. An Introduction to Programming in Emacs Lisp. GNU Press, 2009.