Universidade de Brasília
Instituto de Ciências Exatas

Departamento de Ciência da Computação

117536 - Projeto e Análise de Algoritmos
Turma: B

Relatório sobre **Formalização de Propriedades do Algoritmo Radix Sort utilizando Prototype Verification System (PVS)**

Davi Assis Martinez - 17/0140113
João Antonio D. de Moraes - 16/0126975
Matheus Lemos - 16/0137969

18 de novembro de 2019

1 Introdução

O presente relatório é relativo ao projeto do segundo semestre da disciplina de Lógica Computacional 1, onde se procurou demonstrar a corretude do algoritmo de ordenacão *radixsort*.

2 Apresentação do Problema

2.1 Algoritmos de ordenação e Radix sort

O estudo dos algoritmos de ordenação é essencial à Ciência da Computação. Seja em um contexto acadêmico ou apenas em atividades cotidianas, os seres humanos lidam constantemente com a tarefa de organizar itens a partir de um ou mais critérios - alfabético, númerico crescente ou descrescente, cronológico, por cores, entre outros. Essa é uma das primeiras habilidades intelectuais que costumamos apreender ainda na infância. É compreensível já termos estabelecido um arsenal de técnicas de ordenação para computadores partindo de uma modelagem lógica do problema.

Considere a necessidade de ordenar cartas de um baralho por naipe e número. Uma pessoa familiarizada com a estrutura de um baralho utilizaria suas mãos e olhos para reorganizá-las, tendo uma compreensão ampla do processo. Neste aspecto, máquinas computacionais são comparativamente limitadas em razão de sua natureza binária: a princípio, cada checagem de ordem entre os itens é feita par a par. Isso implica a necessidade de modelagem mais detalhada para definir um processo que complete a tarefa corretamente em tempo hábil.

Já foram criados diversos métodos para obter a ordenação de uma lista de itens de forma lógica. Entre eles, há um algoritmo chamado Radix sort que considera uma dada ordem lexicográfica para ordenar os elementos. Seu nome é derivado da palavra "radix"(latim para "raiz") pois o processo ordena os itens pelas raízes, ou menores unidades possíveis dos itens, como os dígitos em números ou letras em palavras. Este tipo de ordem é definido como a seguir:

> Dados dois conjuntos parcialmente ordenados A e B, a ordem lexicográfica sobre o produto cartesiano A x B é definida como
>
> $(a, b) \leq (a*, b*) \iff (a < a*) \vee (a = a* \wedge b \leq b*)$.

Estamos acostumados a esta ordem pois a ordem alfabética utilizada em dicionários tem a mesma natureza.

Segue uma breve explicação descrição do algoritmo em uma analogia com strings, cuja raiz é um caractere. Dada uma lista de strings como entrada, o algoritmo inicia avaliando o primeiro caractere de cada string. A depender da aplicação, pode se tomar o dígito mais significativo ou o menos significativo

como o primeiro. Daí, ordenar os itens utilizando um algoritmo estável, como o Merge sort, a partir desse caractere. Depois, ordena considerando o próximo caractere de cada string, seguindo assim a cada iteração até o último caractere. No caso de strings de tamanhos diferentes, é atribuído o valor mínimo aos caracteres inexistentes da string menor.

O objetivo principal do projeto é a formalização de algumas propriedades deste algoritmo utilizando o PVS. A primeira questão consiste na demonstração de que o Merge sort retorna uma permutação da lista de entrada. Na segunda, provamos que o Radix sort também retorna uma permutação da lista de entrada. Ao fim, na terceira questão, provamos que o Radix sort retorna uma lista ordenada segundo a ordem lexicográfica dada.

2.2 Análise assintótica

Compreendemos agora que o Radix sort ordena uma lista de "palavras"que tem unidades internas ainda menores: caracteres, dígitos, bytes, entre outros. Para isto, a lista é ordenada n vezes, sendo n o tamanho da maior palavra na coleção. O critério da ordenação é a primeira unidade de cada palavra, depois a segunda, até o fim da maior palavra. Por isso, atinge complexidade *O(nk)* no pior caso, onde n é o tamanho da maior palavra e k a quantidade de palavras na lista.

2.3 Formalização utilizando o Prototype Verification System

O ambiente de desenvolvimento Prototype Verification System - PVS foi utilizado para a formalização das propriedades exigidas no projeto. Ele oferece uma linguagem de especificação lógica própria com sistemas de *typechecking* e obtenção de provas. Essa combinação confere ao PVS poder peculiar: em uma primeira etapa, ele garante não haver erros na especificação utilizando a checagem de tipos; apenas depois disso ele permite a tentativa de provar o enunciado utilizando uma série de comandos reservados para realizar operações lógicas.

Utilizamos os arquivos obtidos no repositório GitHub do projeto, entre eles a especificação do Radix sort, de outro algoritmo de ordenação Merge sort e também bibliotecas como a sorting. Neles, estão contidos definições, lemas, teoremas e conjecturas lógicas que serão utilizados ao longo do trabalho.

2.4 Correção da solução proposta

A primeira questão exige a prova do seguinte enunciado: dada uma lista l como entrada, o algoritmo Mergesort produz como saída uma permutação desta lista.

O primeiro comando utilizado é o (measure-induct "length(l)" "l") para que o PVS gere novos objetivos de prova considerando indução no tamanho da lista l. O segundo comando, (skeep), aplica skolemização ao enunciado, eliminando o quantificador mais externo e dividindo o enunciado em sequente e consequente. A seguir, expandimos a definição de permutations na prova, revelando a utilização da função occurrences em seu conteúdo. Expandimos dessa vez a definição de merge sort apenas no consequente, passo que revela as chamadas recursivas utilizadas no Mergesort. Após isto, a prova é desenvolvida em duas possibilidades (If e Else) que perduram ate o uso do *lift-if* que garante a prova no caso If, então quando dado um comando de *Prop* só é necessário solucionar um caso, evitando, assim, mais uma ramificação.

Na questão 2 foi realizada uma prova expandindo-se a definicão de radixsort, o que implica em utilizar as definicões do *merge_sort*, utilizando a ordem "estritamente menor". Mais especificamente, foi desenvolvida uma prova sobre a o lema *merge_sort_is_permutation*, que diz que o *merge_sort* de uma lista é uma permutacão dos mesmos elementos da lista. Ou seja, o conteúdo da lista permanece o mesmo. Após expandidas essas definicões foi aplicada a definicão de *permutations*, estabelecida no arquivo mergesort.pvs. Essa definicão introduz que para todo x:T pertencente a lista l, qualquer ocorrência no merge_sort(l) também está em l. Após feitas as devidas instanciacões, foi introduzido novamente o lema *merge_sort_is_permutation*, dessa vez com a ordem "menor ou igual". Essa instanciacão foi feita de modo a substituir o sequente "merge_sort(l)"por "merge_sort(merge_sort[T,¡=](l))", que é o sequente que se deseja encontrar, o que finaliza a prova.

Na terceira questão era necessário a finalização da prova do funcionamento do Radixsort. Expandindo sua fórmula fica mais claro o seu funcionamento e consequentemente sua prova. Há o uso do lemma *merge_sort_is_sorted[T,¡¡]* para que haja o desenvolvimento da prova da organização no caso de de um termo da lista a ser ordenada for menor que outro termo da mesma lista. O mesmo acontece com o uso do lemma *is_sorted_implies_ monotone[T,¡¡]* que garante que o "i-ésimo"termo de uma lista é menor que o "j-ésimo".

Referências

[1] S. Owre, N. Shankar, J. M. Rushby, and D. W. Stringer-Calvert, "Pvs system guide," 2001.

[2] N. Shankar, S. Owre, J. M. Rushby, and D. W. Stringer-Calvert, "Pvs prover guide," 2001.

[3] V. Joshi, "Getting to the root of sorting with radix sort." Acessado em: https://medium.com/basecs/getting-to-the-root-of-sorting-with-radix-sort-f8e9240d4224, Apr 2019.

[4] "Ordem lexicográfica." Acessado em: https://pt.wikipedia.org/wiki/Ordem_lexicografica, Sep 2017.

[5] "Radix sort." Acessado em: https://en.wikipedia.org/wiki/Radix_sort, Nov 2019.

[6] R. Sedgewick and M. Schidlowsky, ch. 10. Addison-Wesley, 3 ed., 2003.