Lógica Computacional 117366
Descrição do Projeto
Formalização de Propriedades do Algoritmo *Radix Sort*

01 de Outubro de 2019

Profs. Mauricio Ayala-Rincón & Flávio L. C. de Moura

Estagiária de Docência:
Ariane Alves Almeida
arianealvesalmeida@gmail.com

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 sepadamente 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		22**1**		4**1**3		**1**23
123		34**1**		2**2**1		**1**32
323	primeiro ordenamos a coluna das unidades:	13**2**	em seguinda, as dezenas:	1**2**3	por fim, ordenamos a coluna das centenas:	**2**21
413		12**3**		3**2**3		**3**23
221		32**3**		1**3**2		**3**41
341		41**3**		3**4**1		**4**13

Este processo sempre funciona? No primeiro semestre de 2019 foi estabelecida a correção de esse mecanismo de ordenação para naturais. O 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 assumido estável de forma geral, i.e., não apenas sobre naturais, mas relativo a *pré-ordens totais* sobre um tipo não interpretado `T`. Exemplos de algoritmos não estáveis são *quicksort*, *heapsort* e *binary-insertion sort*.

Uma pré-ordem total sobre um tipo `T` é uma relação de ordem *reflexiva*, *transitiva* e que satisfaz a propriedade de *dicotomia*.

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 uma versão de *radix sort* agindo sobre um tipo não interpretado `T`, e utilizando apenas duas pré-ordens `<<` e `<=`. 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, Lev12]. 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 sobre um tipo não interpretados, como considerado neste projeto, 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`, executável em plataformas Unix/Linux e OSX), os alunos deverão formalizar propriedades de uma especificação do algoritmo *radix sort*. Os arquivos com as questões são denominados `mergesort` e `radix_sort`.

O algoritmo está especificado como abaixo. A função `radixsort` toma uma lista `l` de objetos de tipo `T` e a ordena primeiro conforme a ordem `<=` e logo conforme a ordem `<<` utilizando a função `merge_sort`, que assumimos ordena de forma estável.

```
radixsort(l : list[T]) : list[T] = merge_sort[T,<<](merge_sort[T,<=](l))
```

A estabilidade ou conservatividade de `merge_sort` é especificada no arquivo `mergesort` como abaixo.

```
merge_sort_is_conservative: AXIOM
  FORALL(l: list[T], m,n: below[length(merge_sort(l))]):
    (m < n AND nth(merge_sort(l),m) <= nth(merge_sort(l),n) AND
      nth(merge_sort(l),n) <= nth(merge_sort(l),m)) IMPLIES
    EXISTS (i, j: below[length(l)]):
      i < j AND
      nth(merge_sort(l),m) = nth(l,i) AND
      nth(merge_sort(l),n) = nth(l,j)
```

Estabilidade essencialmente significa que se objetos do tipo `T` na lista sendo ordenada são indistinguíveis segundo a ordem considerada, então eles são mantidos pelo algoritmo de ordenação em posições da lista com a mesma ordem relativa.

2.1 Questões

A primeira questão, consiste em demonstrar que `merge_sort` preserva o conteúdo da lista sendo ordenada. Isto é expresso utilizando o predicado `permutations`. A questão se encontra no arquivo `mergesort`.

```
merge_sort_is_permutation: CONJECTURE
FORALL (l : list[T]) :
  permutations(merge_sort(l), l)
```

A segunda e terceira questões encontram-se no arquivo `radix_sort`. A segunda questão consiste em provar que a função `radixsort` produz uma permutação da lista de entrada.

```
 radixsort_permutes : CONJECTURE
  FORALL(l : list[T]): permutations[T](radixsort(l), l)
```

Finalmente, a terceira questão consiste em demonstrar que função `radixsort` gera uma lista ordenada com respeito à ordem lexicográfica construída das pré-ordens `<<` e `<=`:

```
  radixsort_sorts : CONJECTURE
  FORALL(l : list[T]):
   is_sorted?[T,lex](radixsort(l))
```

Nesta conjectura, `lex` é especificada como abaixo.

```
 lex(x, y:T) : bool =
  (x << y AND NOT (y << x))  OR
(( x << y AND y << x) AND x <= y )
```

3 Etapas do desenvolvimento do projeto

Os alunos deverão definir grupos de trabalho limitados a **três** membros até o dia 9 de Outubro 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 (`mergesort.pvs`, `radixsort.pvs` e, `mergesort.pvs e radixsort.prf`) até o dia **11.11.2019**, 8:00am. Na semana de **11-14.11.2019**, 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 20 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. minutos.
- Entrega do Relatório Final.
  Avaliação (peso 4.0): Cada grupo de trabalho devera entregar um Relatório Final inédito, editado em LaTeX, limitado a oito páginas (12 pts, A4, espaçamento simples), do projeto até o dia **18.11.2019**, 8:00am, 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.

Apenas pontuação de 50% será considerada para relatórios entregues fora desse prazo, até o dia 22 de Novembro. Depois do 22 de Novembro os relatórios não serão considerados.

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.

[Lev12] A. Levitin. *Introduction to the Design & Analysis of Algorithms*. Pearson, third edition, 2012.