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

08 de outubro de 2018

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

Estagiário de docência:
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 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 a ordenação do dígito mais significativo para o menos significativo exigiria um manejo adequado dos números de forma a evitar que cada novo passo desorganizasse 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	primeiro ordenamos a coluna das unidades:	22**1**	em seguinda, as dezenas:	4**1**3	por fim, ordenamos a coluna das centenas:	**1**23
123		34**1**		2**2**1		**1**32
323		13**2**		1**2**3		**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? O objetivo deste projeto é estabelecer 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. Neste 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 *merge sort* como um algoritmo estável adequado para *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`, 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`.

Funções para se obter o determinado dígito de um dado número natural, ou que calcule o número de dígitos de um natural estão definidas no arquivo `radix_sort.pvs`. Por exemplo, a função recursiva `n_digits` que computa o número de dígitos do natural `n` dado como argumento é definida por:

```
n_digits(n:nat) : RECURSIVE posnat =
  IF n < 10 THEN 1
  ELSE 1 + n_digits(ndiv(n, 10))
  ENDIF
  MEASURE n
```

2.1 Questões

A primeira questão, consiste em demonstrar que, para qualquer natural `n`, $10^{\texttt{n\_digits(n)}} > \texttt{n}$:

```
d_digits_gt : CONJECTURE
   FORALL(n : nat):
  10^(n_digits(n)) > n
```

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

```
  merge_permutes : CONJECTURE
  FORALL(l1, l2 : list[nat], d : nat):
  permutations(append(l1,l2),merge(l1,l2,d))
```

onde

```
merge(l1, l2 : list[nat], d : nat) : RECURSIVE list[nat] =
IF null?(l1) OR null?(l2) THEN append(l1, l2)
ELSIF d_nth(car(l1),d) <= d_nth(car(l2),d) THEN cons(car(l1), merge(cdr(l1),l2, d))
ELSE cons(car(l2), merge(l1, cdr(l2), d))
ENDIF
MEASURE length(l1) + length(l2)
```

A função `merge` acima recebe duas listas de naturais como argumento juntamente com um natural `d`, e retorna a fusão (*merge*) destas listas baseada na comparação das entradas consideradas até o seu `d`-ésimo dígito.

Finalmente, a terceira questão está relacionada com a *estabilidade* da função `merge_sort`:

```
merge_sort_d_sorts : CONJECTURE
 FORALL(l : list[nat], d : nat) :
  is_sorted_ud?(l,d)  =>
  is_sorted_ud?(merge_sort(l,d),d+1)
```

3 Etapas do desenvolvimento do projeto

Os alunos deverão definir grupos de trabalho limitados a **três** membros até o dia 16 de outubro de 2018.

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 (`ford_johnson.pvs` e `ford_johnson.prf`) até o dia **12.11.2018**. Na semana de **12-14.11.2018**, 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 LaTeX, limitado a oito páginas (12 pts, A4, espaçamento simples), do projeto até o dia **21.11.2018** 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.