Lógica Computacional 117366
Descrição do Projeto
Formalização de Propriedades de Correção do Algoritmo para Ordenação de Ford-Johnson

11 de maio de 2018

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

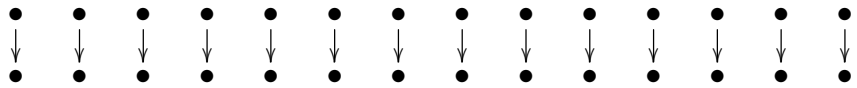
Observação: Os laboratórios do LINF têm instalado o *software* necessário para o desenvolvimento do projeto (PVS 6.0 com as bibliotecas PVS da NASA).

1 Introdução

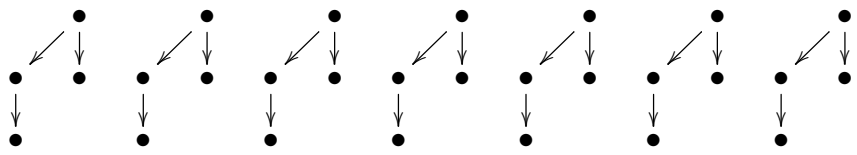
Algoritmos de ordenação são fundamentais em ciência da computação. Neste projeto considerar-se-ão algoritmos relacionados com a primeira fase do método de ordenção de Ford-Johnson [FJJ59] (algoritmo FJ, por brevidade), também conhecido como *merge insertion sort*. Para prova de correção serão aplicadas técnicas dedutivas da lógica de predicados, implementadas no assistente de demonstração PVS, como descrito em [AdM17].

Uma excelente descrição do algoritmo FJ encontra-se no terceiro volume de *The Art of Computer Programming* (pag. 180 da segunda edição) [Knu73], onde o número de comparações que o algoritmo precisa é comparado com os limites teóricos mínimos. Não obstante ser o melhor algorithmo conhecido na literatura (e.g., [CLRS01, BvG99]), não é ótimo; com efeito, em [Man79] é demonstrado que o algoritmo pode ser batido para um número infinito de tamanhos de entrada iniciando por 189 e em [SM81], iniciando por 47. Algoritmos ótimos ainda são questões em aberto para entradas pequenhas como aquelas com 14, 15 e 16 elementos [KK72]. Para entradas com 13 elementos, o limite teórico ($\lceil \log_2 n! \rceil$) é de 33 comparações, mas está demonstrado que ordenar 13 chaves requer 34 comparações, que é o mesmo número requerido pelo algoritmo de Ford-Johnson e suas variantes [ARdAdS07]. Para resultados mais recentes em ordenação ótima, veja por exemplo [IT17] ou a entrada da Wikipedia para "comparison sort".

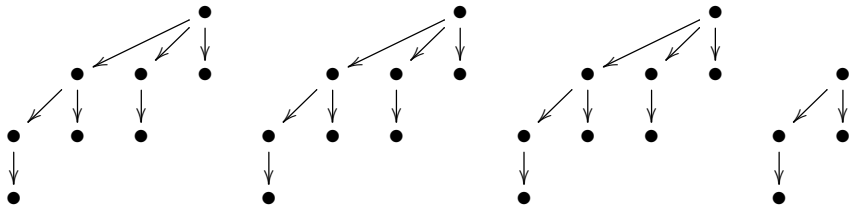
O objetivo do projeto é apenas relativo à verificação da correção da primeira fase da execução do algorithmo FJ, que essencialmente consiste na construção recursiva de uma estrutura de dados na qual são determinadas as relações de ordem entre pares de elementos da entrada. Assim, se temos uma entrada com 28 elementos, numa primeira iteração são comparados 14 pares de elementos, gerando uma *sequência* de 14 pares ordenados como abaixo.



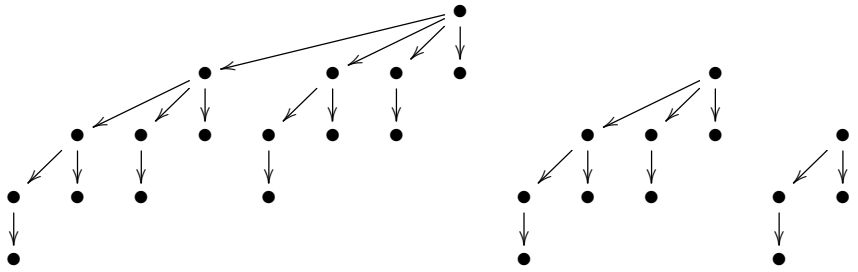
Numa segunda iteração, são comparados os pares de elementos maiores nos 14 pares ordenados, gerando assim uma *sequência* de 7 quátruplas com a estrutura de ordem ilustrada abaixo.



Numa terceira iteração são comparados 3 pares de máximos de seis quátruplas, obtendo uma *sequência* de 3 óctuplas como abaixo, e sobrando uma *lista* unitária com uma estrutura de quátrupla, como abaixo.



Logo, na quarta iteração, obtém-se uma *sequência* unitária com uma única estrutura ordenada com 16 elementos, sobrando uma *lista* de estruturas com uma óctupla e uma quátrupla.



Em geral, para uma entrada com n elementos, são realizadas $\lfloor \log n \rfloor$ iterações, obtendo uma estrutura com $2^{\lfloor \log n \rfloor}$ elementos e uma lista de estruturas sobrantes com possíveis tamanhos $2^0, 2^1, \ldots 2^{\lfloor \log n \rfloor - 1}$.

O objetivo do projeto é demonstrar que uma dada especificação concreta efetivamente produz a estrutura desejada para qualquer entrada com n elementos.

2 Descrição do Projeto

Com base nas *teorias* `sorting_seq` e `seq_extras` especificadas na linguagem do assistente de demonstração PVS (`pvs.csl.sri.com`, executável em plataformas Unix/Linux) e disponíveis na página da disciplina e na biblioteca `nasalib`, respectivamente, os alunos deverão formalizar propriedades de uma especificaçõao para a primeira fase da execução recursiva da ordenação FJ. O arquivo com as questões é denominado `ford_johnson`, e foi desenvolvido em cooperação com o bolsista IC CNPq/UnB Nikson Bernardes.

2.1 Busca em sequências de naturais

O objetivo é demonstrar, formalmente, que a especificação dessa fase do algoritmo FJ gera corretamente a estrutura desejada:

```
seqfjBottleneck(s : finseqfj): RECURSIVE finseqfj =
 IF s`seqfj`length > 1 THEN seqfjBottleneck(compare2to2(s))
 ELSE s
 ENDIF
MEASURE s`seqfj`length
```

Nessa especificação, a função `compare2to2`, disponível na especificação, desenvolve uma iteração da expansão recursiva nessa fase do algoritmo FJ, o que também deve ser demonstrado correto. A estrutura de dados utilizada, `finseqfj`, consiste de uma sequência de estruturas e de uma lista de estruturas como as descritas previamente.

```
finseqfj: TYPE = [# seqfj: finseq[fjnode[nat]], oddList: list[fjnode[nat]] #]
```

Essa estrutura de dados, utiliza como base o tipo de dados `fjnode`, que é o que contém as estruturas descritas, consistente de um valor máximo `val` e uma lista ordenada de estruturas *roteadas* com valores menores `smallerones`. Usa-se também um *flag* que não é relevante na primeira fase do algoritmo.

```
fjnode [T: TYPE]: DATATYPE
 BEGIN
  consFjnode(val: T, smallerones: list[fjnode], a?: boolean):consFjnode?
END fjnode
```

Dois predicados expressam a correção da construção dessas estruturas de dados. A primeira, expressa a correção da construção de `fjnode`s:

```
nstruct?(node:fjnode)(n): RECURSIVE boolean =
IF n>0 THEN
 length(smallerones(node)) = n AND
 FORALL(i:below[length(smallerones(node))]) :
          nstruct?(nth(smallerones(node),i))(n-1-i)
ELSE
 length(smallerones(node)) = 0
ENDIF
MEASURE n
```

A segunda expressa a correção das estruturas `finseqfj` com suas sequências e listas, relativas a uma sequência plana s inicial.

```
nstructER?(s:finseqfj | plain_finseqfj(s))
          (s1, (n:nat | 2^n <= length(s`seqfj))) : bool =
LET l = s`seqfj`length IN
 permutation(s, s1) AND
 s1`seqfj`length = floor(l/2^n) AND
 (FORALL(k:below[s1`seqfj`length]):
           nstruct?(s1`seqfj`seq(k))(n)) AND
  LET lOdd = length_odd(l,n)  IN
          length(s1`oddList) = lOdd AND
  (FORALL(m:below[n]) : odd?(floor(l/2^m)) IFF
      EXISTS(i:below[lOdd]) : nstruct?(nth(s1`oddList,i))(m)) AND
  (FORALL(i:below[length(s1`oddList)]):
    i < length(s1`oddList) - 1 IMPLIES
         length(smallerones(nth(s1`oddList,i))) >
         length(smallerones(nth(s1`oddList,i+1)))) AND
   (length(s1`oddList)>0 IMPLIES length(smallerones(car(s1`oddList))) < n)
```

Objetivo central é mostrar que após as iterações necessárias a estrutura gerada satisfaz o predicado de correção estrutural `nstructER?`.

3 Questões

Concretamente devem ser formalizados três propriedades.

A primeira questão, consiste em demonstrar que a função `compare2to2` gera uma estrutura `finseqfj` que preserva exatamente os elementos da estrutura de entrada, o que está expresso pelo predicado `permutation`. Esse predicado, basicamente verifica que cada elemento ocorre exatamente o mesmo número de vezes nas duas estruturas.

```
comparePeservesElements: CONJECTURE
FORALL(s:finseqfj):  permutation(s,compare2to2(s))
```

A segunda questão, está relacionada com a aplicação desse resultado para formalizar o fato de que ao realizar todas as iterações da primeira fase, obtém-se uma estrutura que é permutação da estrutura original.

```
binsertionsort_works : CONJECTURE
FORALL (v):
       sorted(binsertionsort(v)) AND permutations(v,binsertionsort(v))
```

Finalmente, a terceira questão está relacionada com a correção estrutural:

```
strucutural_correctionFP : LEMMA
FORALL (s0:finseqfj | plain_finseqfj(s0) AND s0`seqfj`length>0)
       (s:finseqfj | s`seqfj`length > 0):
  LET n = logb(s0`seqfj`length,2) IN
  nstructER?(s0)(seqfjBottleneck(s0), n)
```

4 Etapas do desenvolvimento do projeto

Os alunos deverão definir grupos de trabalho limitados a **quatro** membros até o dia 18 de maio.

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 **11.06.2018**. Na semana de **11-15.06.2018**, 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 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 **18.06.2018** com o seguinte conteúdo:
  - Introdução e contextualização do problema.
  - Explicação da soluções.
  - Especificação do problema e explicação do método de solução.
  - Descrição da formalização.
  - Conclusões.
  - 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.

[ARdAdS07] M. Ayala-Rincón, B. T. de Abreu, and J. de Siqueira. A Variant of the Ford-Johnson Algorithm that is more Space Efficient. *Information Processing Letters*, 102(5):201–207, 2007.

[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.

[FJJ59] L. R. Ford Jr. and S. B. Johnson. A tournament problem. *American Mathematical Monthly*, 66(5):387–389, 1959.

[IT17] K. Iwama and J. Teruyama. Improved average complexity for comparison-based sorting. In *Algorithms and Data Structures - 15th International Symposium, WADS, Proceedings*, volume 10389 of *Lecture Notes in Computer Science*, pages 485–496. Springer, 2017.

[KK72] D. E. Knuth and E. B. Kaehler. An Experiment in Optimal Sorting. *Information Processing Letters*, 1:173–176, 1972. Reprinted as Chapter 30 in *Selected Papers on Algorithms*, D. E. Knuth, SLSI, 2000.

[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.

[Man79] G. K. Manacher. The Ford-Johnson sorting algorithm in not optimal. *J. of the ACM*, 26(3):441–456, 1979.

[SM81] J. Schulte Mönting. Merging of 4 or 5 elements with n elements. *Theor. Comput. Sci.*, 14:19–37, 1981.