

Projeto (Opcional - 20 pontos)

Projeto e Análise de Algoritmos (2025/2)

Flávio L. C. de Moura

14 de outubro de 2025

1 Propriedades de Algoritmos

Ao longo do semestre, exploramos dois aspectos essenciais dos algoritmos: complexidade (tempo e espaço) e correção. O foco deste projeto é no segundo aspecto. Em geral, a correção de um algoritmo é uma propriedade que envolve diversas etapas, e portanto as provas em papel e lápis estão sujeitas a erros. A utilização de ferramentas computacionais, como assistentes de provas, nos permite minimizar a ocorrência de erros quando conseguimos expressar as propriedades a serem provadas.

1.1 A Formalização de Propriedades de Algoritmos

O objetivo desse projeto é a construção de uma prova formal, ou seja, utilizando um *software* de prova (e não apenas uma prova em papel e lápis), que estabeleça uma (várias) propriedade(s) do algoritmo escolhido.

1.1.1 Propostas

Apresentaremos uma proposta bem estruturada que poderá ser utilizada por quaisquer dos grupos, e em seguida, apresentaremos as linhas gerais que devem ser seguidas pelos grupos que optarem pela formalização de outros algoritmos.

A utilização de inteligências artificiais são permitidas e recomendadas, e todo o processo de desenvolvimento deve ser incluído no relatório final.

1. A correção da ordenação por inserção binária

O algoritmo de ordenação por inserção binária funciona como o algoritmo de ordenação por inserção padrão com a diferença que utilizamos o algoritmo de busca binária para fazer a inserção de novos elementos em um subvetor ordenado.

A busca binária `bsearch x l` recebe o elemento `x` e a lista ordenada `l` como argumentos, e retorna a posição `i` de `l` onde `x` deve ser inserido. Considere o código a seguir:

```

Function bsearch x l {measure length l} :=
  match l with
  | [] => 0
  | [y] => if (x <=? y)
            then 0
            else 1
  | h1::h2::tl =>
    let len := length l in
    let mid := len / 2 in
    let l1 := firstn mid l in
    let l2 := skipn mid l in
    match l2 with
    | [] => 0
    | h2'::l2' => if (x <? h2')
                  then bsearch x l1
                  else
                    if x =? h2'
                    then mid
                    else mid + (bsearch x l2)
    end
  end.

```

A inserção, por sua vez, é feita pela função a seguir:

```

Fixpoint insert_at i (x:nat) l :=
  match i with
  | 0 => x::l
  | S k => match l with
            | nil => [x]
            | h::tl => h::(insert_at k x tl)
          end
  end.

```

O algoritmo de ordenação por inserção binária pode ser construído a partir da combinação do trabalho das funções `bsearch` e `insert_at` como a seguir:

```

Definition binsert x l :=
  let pos := bsearch x l in
  insert_at pos x l.

```

Ou ainda, condensando o código das duas funções:

```

Function binsert x l {measure length l} :=
  match l with

```

```

| [] => [x]
| [y] => if (x <=? y)
        then [x; y]
        else [y; x]
| h1::h2::tl =>
    let len := length l in
    let mid := len / 2 in
    let l1 := firstn mid l in
    let l2 := skipn mid l in
    match l2 with
    | [] => l
    | h2'::l2' => if x =? h2'
                  then l1 ++ (x :: l2)
                  else
                      if x <? h2'
                      then bininsert x l1
                      else bininsert x l2
    end
end.

```

O algoritmo principal pode, então, ser construído como a seguir:

```

Fixpoint bininsertion_sort (l: list nat) :=
  match l with
  | [] => []
  | h::tl => bininsert h (bininsertion_sort tl)
  end.

```

A correção do algoritmo $[\text{bininsertion_sort}]$ consiste em provar o seguinte teorema:

```

Theorem bininsertion_sort_correct: forall l,
  Sorted le (bininsertion_sort l) /\
  Permutation l (bininsertion_sort l).

```

O objetivo desta proposta é provar o teorema `bininsertion_sort_correct`. Observe que pode ser conveniente dividir esta prova em outras provas menores. Isto significa que a formalização pode ficar mais simples e mais organizada com a inclusão de novos lemas.

O código descrito nessa seção pode ser clonado a partir do repositório https://github.com/flaviodemoura/bininsertion_sort. O algoritmo, ou seja, as funções podem ser modificadas como desejar, mas justifique as mudanças e/ou ajustes feitos no relatório.

2. Propostas alternativas

Alternativamente, os grupos podem trabalhar com outros algoritmos desde que não sejam excessivamente simples. O algoritmo escolhido pode utilizar o paradigma funcional ou imperativo, mas consulte o professor para que a escolha seja aprovada, e uma estratégia de prova seja construída.

2 Etapas do Projeto

Os alunos interessados nesse projeto devem se organizar em grupos de até 3 alunos até o dia *[2025-10-19 dom]*.

Em seguida, cada grupo deve criar um repositório com permissão de leitura para o professor até o dia *[2025-10-20 seg]*.

O prazo para finalização do projeto é o dia *[2025-12-07 dom]*.

O repositório deve conter

1. O arquivo **relatorio.pdf** contendo o nome e matrícula dos participantes, assim como todas as informações pertinentes ao projeto, como a estruturação dos arquivos de prova, a estruturação das provas (em linguagem natural, evite colar código no relatório) e as explicações detalhadas de como foi o desenvolvimento do projeto.
2. Os arquivos com as provas como descrito no relatório.

Os grupos que tiverem interesse podem agendar uma apresentação para o dia *[2025-12-08 seg]*. O agendamento deve ser feito até o dia *[2025-11-24 seg]*.