O Princípio da Indução e suas Aplicações

João Lucas Azevedo Yamin Rodrigues da Cunha
Mariana Alencar do Vale
Pedro Vitor Valença Mizuno
Vitor Ribeiro Custódio

4 de dezembro de 2018

Resumo

Neste trabalho mostraremos diversas utilizações do princípio de indução em Ciência da Computação. São apresentadas duas provas envolvendo indução matemática e estrutural, bem como o desenvolvimento de seu raciocínio.

1 Introdução

Neste documento será descrito o desenvolvimento de dois problemas matemáticos utilizando a indução matemática. Indução é um método de prova matemática usado para demonstrar a verdade de um número infinito de proposições dentro do conjunto dos naturais ou, no caso do segundo problema, dentro da estrutura de uma fórmula. A ideia por trás da indução é, através de casos base ou minimais que sabemos/provamos que são verdadeiros, provarmos, de forma recursiva, que a propriedade é válida para todo o conjunto infinito.

Utilizaremos este método para provar dois problemas. O primeiro será provar a equivalência entre os princípios da indução forte e da indução matemática. No segundo problema, iremos provar a corretude do algoritmo *Insertion Sort*, muito utilizado para ordenação de listas. Além disso, iremos avaliar as principais diferenças de uso e de estrutura na utilização da indução forte, como será demonstrado no problema 1.

2 Desenvolvimento das provas

2.1 Equivalência entre princípios de indução

No primeiro problema proposto pela atividade, devemos apresentar a equivalência entre os princípios da indução matemática e da indução forte. Para provarmos a equivalência, precisamos:

i) Provar que conseguimos obter os principios da indução forte pela indução fraca;

ii) Provar que conseguimos obter os principios da indução fraca pela indução forte;

Primeiro, precisamos definir indução matemática e indução forte

- Indução fraca
  O princípio da indução nos permite provar propriedades sobre os números naturais $\mathbb{N}$=1,2,..., e a partir de alguns exemplos. Para provar que uma propriedade *M* sobre os naturais é válida precisamos mostrar duas coisas:

  a) **Base Indutiva**: mostrar n_0 que satisfaz a propriedade, isto é, provar $M(n_0)$;
  b) **Passo Indutivo**: assumindo que um natural *n* satisfaça a propriedade *M*, precisamos provar que n+1 também satisfaz a propriedade, isto é, precisamos construir uma prova de $M(n) \rightarrow M(n+1)$;

- Indução Forte
  A indução forte geralmente é utilizada quando não se pode demonstrar com facilidade a veracidade de uma propriedade utilizando a indução fraca.

  a) **Base Indutiva**: O princípio da **Base Indutiva** da indução forte é o mesmo da base fraca
  b) **Passo Indutivo**: Ao invés de considerarmos verdadeiro *M(n)* para provar *M(n+1)*, consideramos que $\forall$ k $\leq$ n satisfazem a *M(k)* e então provamos *M(n+1)*, ou seja, $M(1) \wedge M(2) \wedge ... \wedge M(n) \rightarrow M(n+1)$;

Definidas as induções, podemos iniciar a construção da prova;

A prova da indução consiste em duas etapas, a base indutiva e o passo indutivo. Como a base indutiva é a mesma, sua prova é trivial. No caso do passo indutivo, teremos que provar que de indução fraca conseguimos chegar a indução forte e que da indução forte conseguimos chegar a indução fraca.

1. Neste item, provaremos I.Forte $\Rightarrow$ I.Fraca:
   No **passo indutivo** da Indução Forte, a partir de uma propriedade *M*, assumimos que $M(1) \wedge M(2) \wedge ... \wedge M(n)$ são verdadeiros para então obter *M(n+1)*. Disso, temos *M(n)* verdadeiro, que faz parte da hipótese da fraca e com isso obtemos que de $M(n) \rightarrow M(n+1)$, logo encontramos o passo indutivo da indução fraca.

2. Neste item, provaremos que I.Fraca $\Rightarrow$ I.Forte
   No **passo indutivo** da Indução Fraca, a partir de uma propriedade *M*, assumimos que *M(n)* é verdadeiro para então chegar em *M(n+1)*
   Sendo *M(n)* verdadeiro, temos, por hipótese, $M(n\text{-}1) \rightarrow M(n)$. Continuando o processo até que obtemos $M(n_0) \rightarrow M(n_0 + 1)$. Assim temos que $M(n_0) \wedge M(n_0+1) \wedge ... \wedge M(n)$ geram o *M(n+1)* verdadeiro. Com isso, obtemos a indução forte a partir da fraca.

2.2 Correção do algoritmo de ordenação por inserção

No segundo problema proposto pela atividade, devemos provar a correção do *insertion sort*, um algoritmo de ordenação de listas. A definição do algoritmo é explicitada a seguir:

$$InsertionSort(l) = \begin{cases} l, & se \;\; l = [] \\ Insert(h, InsertionSort(l')), & se \;\; l = h :: l' \end{cases} \tag{1}$$

onde Insert é definido como

$$Insert(x,l) = \begin{cases} x :: [], \;\; se \;\; l = [] \\ x :: l, \;\; se \;\; l = h :: l' \;\; e \;\; x \leq h \end{cases} \tag{2}$$

Vamos compreender alguns termos desse algoritmo. Sendo *l* uma lista, esta pode ser definida como:

$$l = \begin{cases} [\,]\text{, se a lista for vazia} \\ h\ ::\ l'\text{, onde } h \text{ é o primeiro elemento de } l \text{ e } l' \text{ é o restante da lista} \end{cases}$$

Precisamos definir também o que significa uma lista estar ordenada, que é, supostamente, o produto final do *insertion sort*. Seja *l* uma lista e *E(l)* o conjunto de todos os elementos da lista *l*. Assim,

$$E\,(\,l\,)\ =\ \{\ a_0,\ a_1,\ a_2,\ \ldots\ ,\ a_n\ \}\ ,\ n\ =\ |\,l\,|\ -\ 1$$

Logo, seja *l* a lista original e *l'* a lista resultante do processo de ordenação de *l*. Dizemos que *l t* foi corretamente ordenada se

$$i\,)\ E(l') = E(l)$$
$$ii\,)\ \ a_i \leq a_{i+1}, \forall a_i\ \in E(l')\,, i \in [0, |l|)$$

Ou seja, quando:

i) todos os elementos da lista original estiverem presentes na lista ordenada;

ii) quando todo elemento da lista ordenada for menor ou igual a seu sucessor.

Tendo como base estas definições, começamos a desenvolver. Inicialmente, sabemos que utilizaremos indução, pois, além de ser a proposta do problema, o que queremos provar precisa ser verdadeiro para todos os casos dentro de um conjunto infinito de possibilidades.

Para começarmos a indução, iremos separar em dois casos: o caso base e o caso indutivo. O caso base é necessário, pois a partir dele provamos que o algoritmo é válido para pelo menos um caso, depois disso podemos tentar provar para os casos seguintes.

O caso base para o algoritmo *insertion sort*, será para quando a lista estiver vazia, ou seja, quando não tiver nenhum elemento. Nesse caso, a lista tem tamanho zero e, segundo a definição do algoritmo, a saída será a própria lista vazia. Assim, satisfaz o item i da definição acima descrita, uma vez que o conjunto dos elementos das listas de entrada e saída são iguais, por ser um conjunto vazio. Esse caso é trivial, uma vez que, por ter nenhum elemento, a lista já está ordenada.

Já no passo indutivo, temos necessariamente que considerar o funcionamento do *Insertion Sort*. Sabemos que o caso em que não temos nenhum elemento já está ordenado, e conseguimos ver também que o caso em que temos apenas um elemento também é simples. Quando uma lista possui um elemento, ela também já está ordenada.

A dificuldade surge quando começamos a trabalhar com dois ou mais elementos. Assim, destinamos nossa atenção às partes do *Insertion Sort*, dentre elas, o próprio *Insert*. No caso base, tendo uma lista vazia, o *Insert* apenas adiciona o elemento à cabeça da lista. Esse caso já era esperado e não adiciona nenhum resultado à linha de raciocínio. Porém, quando a lista possui apenas um elemento, o *Insert* só adiciona um novo elemento x à cabeça da lista se x for menor ou igual à cabeça já existente. Ou seja, uma vez que uma lista de tamanho 1 já está ordenada, o *Insert* não altera a propriedade "estar-ordenada"da lista ao adicionar um elemento.

Dessa forma, conseguindo provar que, sempre que uma lista já está ordenada, ao adicionar um novo elemento através do *Insert*, a propriedade de ordenação da lista não é alterada, o *Insertion Sort* estará correto.

Assim, podemos separar em mais alguns casos de aplicação do *Insertion Sort*. Quando a lista está vazia, já sabemos que está ordenada. Ao adicionar um elemento à uma lista vazia, também estará ordenada. Agora, se formos adicionar um elemento x ao caso em que já temos uma cabeça h, podemos separar em dois casos:

i) Quando $h \geq x$. Nesse cenário, considerando que a lista anterior já estava na ordem crescente (que é o caso quando temos apenas dois elementos), a lista irá se manter ordenada;

ii) Quando $h < x$. Já nesta situação, o processo será reaplicado em x, porém tentando adicioná-lo numa lista menor que a original, em que não esteja contido h. No caso de dois elementos, essa lista menor será uma lista vazia e permitirá que seja feita a adição de x, só que dessa vez no final da lista resultante.

Isso nos leva à dificuldade do *Insert*: quando a adição é de um novo elemento não menor que a cabeça da lista original.

Pensemos agora do ponto de vista indutivo. Vamos considerar que $P(n)$ é "o *Insert* adiciona corretamente um elemento numa lista de n elementos". Então, $P(n)$ é nossa hipótese de indução e assumimos que está correta. Também sabemos que $P(0)$ e $P(1)$ são verdadeiros. Então vamos pensar a respeito de $P(n+1)$, que é uma lista de tamanho n com a adição de um elemento.

Como $P(n)$ é verdadeiro por hipótese de indução, então uma lista de tamanho n pode ser ordenada corretamente pelo algoritmo *Insertion Sort*. Então iremos adicionar um elemento x a essa lista. Conforme os casos acima comentados, temos duas situações.

Na primeira, onde $h \geq x$, como a lista de tamanho n anterior já estava ordenada, então, segundo o *Insert*, o x será adicionado no início da lista e esta estará ordenada.

Já na segunda situação, o Insert não irá adicionar o x no início da lista, e, recursivamente, vai tentar adicionar à cauda restante, ou seja à lista menos a cabeça h. Porém, pela hipótese de indução $P(n)$, em que uma lista de tamanho n consegue ser ordenada, a lista também conseguirá ser ordenada, independente de qual seja a posição em que seja inserido o x e quantas recursões sejam necessárias para que isso aconteça.

Logo, conseguimos cobrir todos os casos de inserção do *Insert*. Portanto, uma vez que o *Insert* está correto, o algoritmo *InsertionSort* também está.

3 Prova

3.1 Problema 1

Primeiramente, consideramos uma propriedade P a qual podemos aplicar em n números naturais, números naturais estes delimitados por um n_0, que é o menor número em que é possível aplicar a propriedade.

A prova da equivalência da base indutiva das induções Forte e Fraca é trivial, visto que seguem o mesmo princípio: demonstrar que $P(n_0)$ é válido, de tal forma que podemos chegar a mesma conclusão usando qualquer um dos métodos de indução.

No caso do passo indutivo, para atingirmos a equivalência, devemos mostrar que a partir da Indução Forte é possível chegar à Fraca e vice-versa.

Indução Forte $\implies$ Indução Fraca: Por meio do passo indutivo da Indução Forte, assumimos que para todo k, $n_0 \leq k \leq n$, $P(k)$ é verdadeiro, sendo assim, $P(n+1)$ é verdadeiro. Dessa hipótese supomos $P(n+1)$ verdadeiro e que $P(k)$ será verdadeiro no domínio delimitado, de tal forma que é possível assumir $P(n)$ verdadeiro, assim, é viável afirmar que $P(n)$ implica em $P(n+1)$, $P(n) \implies P(n+1)$, o qual corresponde ao passo indutivo da Indução Fraca.

Indução Fraca $\implies$ Indução Forte: Por meio do passo indutivo da Indução Fraca, assumimos que n aplicado em $P(n)$ é verdadeiro, para então provar $P(n+1)$ verdadeiro. Seguindo tal hipótese, podemos entender que para $P(n)$ ser considerado verdadeiro, há um $n-1$ tal que $P(n-1)$ é também válido. Tal lógica é possível ser aplicada em $n-1$ e assim por diante, de tal forma que todo k, $n_0 \leq k \leq n$, é satisfatório para a propriedade P, tal que $P(k)$ é verdadeiro. Assim, chegamos à conclusão que

$$P(n_0) \wedge P(n_1) \wedge ... \wedge P(n-2) \wedge P(n-1) \wedge P(n) \implies P(n+1),$$

o qual corresponde ao passo indutivo da Indução Forte.

Por fim, concluímos que Indução Forte e Fraca são equivalentes.

3.2 Problema 2

Utilizando as definições da seção 2.2, podemos sintetizar a prova da seguinte forma. Aplicando a indução estrutural no algoritmo *Insertion Sort*, podemos separar dois casos distintos de listas: o caso base, no qual a lista é vazia, e o caso indutivo, em que a lista é composta por pelo menos um elemento.

No primeiro caso, a solução é trivial. Por não ter nenhum elemento, a lista já está ordenada. O algoritmo, nessa situação, não altera a lista de nenhuma forma. Por isso, o algoritmo está correto para este caso.

No segundo caso, vamos considerar uma proposição $P(n)$: "o algoritmo *Insertion Sort* está correto para uma lista de n elementos". Ou seja, que para uma lista com n elementos, o algoritmo é capaz de ordená-la corretamente. Consideremos $P(n)$ nossa hipótese de indução e, consequentemente, verdadeira.

Como o algoritmo *Insertion Sort* é correto para uma lista de n elementos, queremos provar que também é correto para uma lista de $n+1$ elementos, ou seja, que $P(n+1)$ é verdadeira.

Adicionando, então, um elemento a uma lista de n elementos, temos duas situações possíveis, ambas abordadas na definição de *Insertion Sort* e *Insert*:

1. $h \geq x$: Nesse caso, a cabeça da lista original é maior que o elemento x que queremos adicionar. Como a lista original já pode ser ordenada pelo algoritmo, como diz a hipótese de indução, então x não comprometerá a ordenação da lista. Logo, o *Insertion Sort* é correto neste caso;

2. $h < x$: Nesta situação, a cabeça da lista original é menor que o elemento x. Então, pela definição de *Insert*, x não pode ser adicionado ao início da lista. Uma chamada recursiva será feita, e o *Insert* tentará adicionar x na cauda da lista original, ou seja, esta sem sua cabeça. Porém, essa nova lista composta de x e

a lista original sem a cabeça possui um total de n elementos. E, por hipótese de indução, $P(n)$, o algoritmo é correto. Então a lista é ordenada.

4 Conclusão

Diante dos dois problemas apresentados, foram realizadas aplicações diferentes do mesmo conceito de Indução como solicitado pela especificação da atividade. O desenvolvimento desta permitiu aos integrantes do grupo se familiarizarem mais com a metodologia, o que concebeu, do nosso ponto de vista, uma resolução correta das questões.