Análise de algoritmos

Introdução

Métodos formais tem como propósito formalizar a noção de especificação de algoritmos de maneira analisar e provar que o nosso algoritmo realmente faz o que diz que faz, por esse motivo tem 4 conceitos iniciais muito importantes de se entender para avaliar os algoritmos.

1. Pré-condição
2. Pós-condição
3. Invariantes
4. Variantes

Pré-condição

Uma pré-condição é uma condição sobre a entrada dos dados para o algoritmo funcionar, ele é um predicado da lógica de predicados que retorna um valor booleano

que deve ser sempre verdadeira antes da execução do algoritmo.

Pós-condição

Uma pós-condição é a condição sobre a saída do algoritmo, onde também é um predicado que deve sempre ser verdadeiro após a execução do algoritmo.

Invariantes

Invariantes é uma verificação em uma certa fase de execução do algoritmo que também é um predicado que deve sempre ser verdadeiro, iremos ver que as invariantes são muito importantes para a correção de algoritmos e podermos ver qual são os valores que nunca mudam do algoritmo.

Variantes

Variantes são os valores que vivem alterando dentro do algoritmo, normalmente são as **variáveis** criadas para pegar valores ou alterar valores dentro do algoritmo.

Princípio da Invariante

Formulado por Robert W. Floyd em 1967. Resumidamente, ele é uma versão do **Princípio da Indução** especificamente criado para a construção de provas de propriedades sobre **máquina de estados**.

Uma máquina de estados é um modelo abstrato adequado para um algoritmo interativo cuja computação se dá passo a passo.

Um algoritmo como sequencia de estados podemos ver cada passo que os dados vão sendo alterados para termos uma visualização maior do processo.

O princípio da invariante mostra como o estado inicial do algoritmo corresponde ao caso base da indução, não sofrendo alterações durante o percurso, onde temos uma hipótese de indução que deve ser provada pelo Princípio da indução.

Invariante de laço

Em algoritmos interativos, uma das variantes de maior importância é a chamada **invariante de laço,** nos focamos em uma parte do algoritmo que corresponde a um laço de repetição (em programação conhecidos como For, While), onde estamos interessados em definir uma propriedade que demonstre uma relação entre os elementos do estado de uma computação que deve ser verdadeira antes da interação do laço e deve ser verdadeira depois do final da iteração do laço.

Exemplo completo inicial

- Detalhes da questão

Dizemos que temos um algoritmo que soma dois valores inteiros e a subtração de números inteiros são as únicas operações aritméticas que um determinado agente computacional consegue fazer.

Vamos analisar um algoritmo de somas sucessivas para multiplicar dois números inteiros positivos.

Dado que somente temos o algoritmo de soma de números inteiros, os valores de m que é o nosso **multiplicando** (o valor que é o que vai ser multiplicado) e o n que vai ser nosso **multiplicador** (o valor que vai dizer quantas vezes vai ser repetido):

$$m \times n$$

- Brainstorm da Pré-condição

o valor de m e de n devem ser maiores que zero, porque temos uma multiplicação de números inteiros positivos como passado na explicação, por isso devemos dizer para o nosso algoritmo quais são os valores possíveis para m e n, onde podemos mostrar matematicamente assim:

$$m \geq 0 \ , \ n \geq 0$$

- Prova de mesa e apresentação de como o algoritmo funciona:

Visual	Funcionamento do Algoritmo
5 * 3	5 * 3 + 0
	4 * 3 + 3
	3 * 3 + 6
	2 * 3 + 9

Visual	Funcionamento do Algoritmo
	1 * 3 + 12
	0 * 3 + 15
	15

Podemos ver nessa prova de mesa que é pego o valor do multiplicador (nesse caso o valor 3) e vai somando ele no final até acabar o número do multiplicando, onde vai sendo subtraído 1 dele até chegar em zero.

```
5 * 3 + 0
(5 - 1) * 3 + (0 + 3) = 4 * 3 + 3
(4 - 1) * 3 + (3 + 3) = 3 * 3 + 6
...
(1 - 1) * 3 + (12 + 3) = 0 * 3 + 15
Resultado final = 15
```

agora que entendemos como o algoritmo está tratando os valores de m e n em cada soma sucessiva podemos construir uma fórmula usando lógica proposicional e formalização usando equações recursivas:

Cabeçalho do Algoritmo

estruturação com o nome e os tipos de dados esperados dos atributos dos algoritmo.

1. Temos dois atributos de entrada para o algoritmo, o m que é apresentado como um número inteiro e o n que também é um número inteiro, o resultado desse algoritmo é um número inteiro também, podemos apresentar um número inteiro matematicamente usando o símbolo $\mathbb{Z}$

$$\mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$$

2. Agora damos o nome para o nosso algoritmo, nesse caso chamamos de $Mult$**,** com isso podemos construir o cabeçalho do algoritmo assim:

$$Mult : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$$

Pré-condição

Após construído o cabeçalho dos tipos esperados e o nome, temos que dizer as pré-condições, onde devemos citar cada informação importante para o bom funcionamento do nosso algoritmo.

Neste algoritmo temos a questão que os valores do multiplicando e do multiplicador devem ser inteiros positivos, por isso devemos dizer na pré-condição que o valor de m **e** n **devem ser maior ou igual a zero** para que o nosso algoritmo não possa pegar valores negativos.

Quando queremos colocar em um algoritmo escrito em linguagem natural, utilizamos a palavra **Requer** para podermos mostrar no algoritmo a pré-condição.

Como fica:

$$Requer : m \geq 0 \ , \ n \geq 0$$

Podemos também escrever a palavra **Pré-condição** para não causar desconforto ou falta de entendimento de como estão sendo tratados as informações:

$$Pré - condição : m \geq 0 \ , \ n \geq 0$$

Algoritmo

Agora devemos mostrar o que é feito em nosso algoritmo, onde podemos mostrar usando formulas indutivas ou recursivas

De uma função recursiva podemos criar as fórmulas que fazem a geração dos valores, onde temos no exemplo dado o seguinte:

$$0 \times 3 + 15 = 15$$

Isso nos mostra que quando chega em zero o multiplicando, ele vai nos entregar somente o valor somado, com isso podemos ter a seguinte construção da fórmula e seus atributos

$$Mult(m, n, p)$$

Onde m é o multiplicando, o n é o multiplicador e o p é o resultado das somas consecutivas, que no fim vai ser nosso resultado.

Quando construímos funções recursivas precisamos apresentar primeiro o caso de parada, mostrando quando vai ser parado o processo recursivo.

No nosso caso, quando o m se tornar zero ele deve retornar o valor de p.

$$Mult(0, n, p) = p$$

Agora vamos pensar no caso da recursão em si, um caso recursivo é uma re-chamada da função recursiva com as devidas modificações para a próxima chamada da função recursiva.

No nosso caso, a recursão é a subtração do valor do multiplicando m por 1 e somando o valor do multiplicador n no atributo p.

$$Mult(m, n, p) = Mult(m - 1, n, p + n)$$

Podemos testar essas fórmulas com os valores vistos no exercício.

$$Mult(5, 3, 0) = Mult(5 - 1, 3, 0 + 3 = Mult(4, 3, 3)$$

Fórmula	Funcionamento
$Mult(m, n, p) = Mult(m - 1, n, p + n)$	$Mult(5, 3, 0) = Mult(5 - 1, 3, 0 + 3 = Mult(4, 3, 3)$
$Mult(m, n, p) = Mult(m - 1, n, p + n)$	$Mult(4, 3, 0) = Mult(4 - 1, 3, 3 + 3 = Mult(3, 3, 6)$
$Mult(m, n, p) = Mult(m - 1, n, p + n)$	$Mult(3, 3, 6) = Mult(3 - 1, 3, 6 + 3 = Mult(2, 3, 9)$
$Mult(m, n, p) = Mult(m - 1, n, p + n)$	$Mult(2, 3, 9) = Mult(2 - 1, 3, 9 + 3 = Mult(1, 3, 12)$
$Mult(m, n, p) = Mult(m - 1, n, p + n)$	$Mult(1, 3, 12) = Mult(1 - 1, 3, 12 + 3 = Mult(0, 3, 15)$
$Mult(0, n, p) = p$	$Mult(0, 3, 15) = 15$

Como pode ser visto acima, foi sendo reutilizado a função recursiva até chegar no caso base onde o primeiro valor da função era zero, dai ele fez a função base concluindo o algoritmo.

Com isso provamos que nossas funções recursivas estão entregando o resultado esperado.

Então resumidamente elas são apresentadas assim:

$$\begin{cases} Mult(0, n, p) = p \\ Mult(m, n, p) = Mult(m - 1, n, p + n) \end{cases}$$

Agora que criamos a função recursiva para resolver nosso algoritmo, vamos apresentar as pós-condições após ter sido resolvido.

Pós-condição

A pós-condição é apresentada utilizando a lógica proposicional, onde apresentamos o que deve ser entregue pelo algoritmo.

No final do algoritmo queremos dizer que o resultado do algoritmo é o retorno esperado da multiplicação entre o valor de m e n.

Como podemos ver,

$$\forall m, n \in \mathbb{Z}.(m \geq 0 \land n \geq 0 \Rightarrow Mult(m, n, p) = m \times n)$$

Nessa fórmula está escrito em lógica proposicional:

> Para todos os valores de m e n pertencente ao conjunto dos números inteiros, onde m é maior ou igual a zero e n é maior ou igual a zero, tal que a função $Mult(m, n, p)$ entrega a multiplicação de m por n

Apresentamos essa verificação na nossa análise dessa forma:

$$Pós - condição : \forall m, n \in \mathbb{Z}.(m \geq 0 \land n \geq 0 \Rightarrow Mult(m, n, p) = m \times n)$$

Variantes e Invariantes

Antes de vermos a sequência de estados, vamos ver os valores das **variáveis** do algoritmo, como mostra a tabela abaixo dos valores sendo modificados durante o processo:

Sequência	m	n	p
0	5	3	0
1	4	3	3
2	3	3	6
3	2	3	9
4	1	3	12
5	0	3	15

Tendo isso em vista, podemos construir a sequência de dados nomeando cada passo com S_k onde k é a sequência atual do algoritmo, com seus valores mostrados acima,

onde podemos mostrar em uma tabela usando a seguinte fórmula:

$$S_k = (m_k, n_k, p_k$$

m_k : é o valor da variável m na sequência k, tendo seu valor alterado em cada sequência.

n_k : é o valor da variável n na sequência k, tendo seu valor alterado em cada sequência.

p_k : é o valor variável p na sequência k, tendo seu valor alterado em cada sequência.

Vamos ver a máquina de estados do nosso algoritmo abaixo:

Sequência de Estados	**Variáveis**
$S_0 = (m, n, 0) = (5, 3, 0)$	(m_0, n_0, p_0)
$S_1 = (m-1, n, n) = (4, 3, 3)$	(m_1, n_1, p_1)
$S_2 = (m-2, n, n+n) = (3, 3, 6)$	(m_2, n_2, p_2)
$S_3 = (m-3, n, n+n+n) = (2, 3, 9)$	(m_3, n_3, p_3)
$S_4 = (m-4, n, n+n+n+n+n) = (1, 3, 12)$	(m_4, n_4, p_4)
$S_5 = (m-5, n, n+n+n+n+n) = (0, 3, 15)$	(m_5, n_5, p_5)

Dessa forma, podemos visualizar o funcionamento do algoritmo e como os dados interagem dentro da sequência de estados, onde também já identificamos as **variantes** do nosso algoritmo.

Agora podemos ver que os valores de m,n e p mudam a cada repetição do laço após k interações, com isso podemos ver que a asserção de valores desde o inicio até o fim não modificou, onde isso significa que ele não **variou** durante o algoritmo e as interações do laço e continuou verdadeiro até o fim, abaixo podemos ver as invariantes de cada estado:

Estado	**Invariante**
$S_0 = (5, 3, 0) = (m_0, n_0, p_0)$	$5 \times 3 = 5 \times 3 + 0$
$S_1 = (4, 3, 3) = (m_1, n_1, p_1)$	$5 \times 3 = 4 \times 3 + 3$
$S_2 = (3, 3, 6) = (m_2, n_2, p_2)$	$5 \times 3 = 3 \times 3 + 6$
$S_3 = (2, 3, 9) = (m_3, n_3, p_3)$	$5 \times 3 = 2 \times 3 + 9$
$S_4 = (1, 3, 12) = (m_4, n_4, p_4)$	$5 \times 3 = 1 \times 3 + 12$

Estado	Invariante
$S_5 = (0, 3, 15) = (m_5, n_5, p_5)$	$5 \times 3 = 0 \times 3 + 15$

Podemos ver que o que sempre é verdadeiro é a fórmula da multiplicação com seus valores acima de zero, por isso a **Invariante** é a seguinte fórmula:

$$m_0 \times n_0 = m_k \times n_k + p_k, k \in \mathbb{N} \ \wedge \ k \geq 0$$

m_0 é o valor inicial da variável m do inicio do algoritmo.

n_0 é o valor inicial da variável n do inicio do algoritmo.

m_k é o valor da variável m em cada passo do laço de repetição.

n_k é o valor da variável n em cada passo do laço de repetição.

p_k é o valor da variável p em cada passo do laço de repetição.

$k \in \mathbb{N} \ \wedge \ k \geq 0$ significa que a variável k pertence ao conjunto dos números naturais e seu valor é maior ou igual a zero.

Além disso, a **outra invariante** muito importante também é a pré-condição, onde o valor de m deve ser maior ou igual a zero e n deve ser maior ou igual a zero.

$$m \geq 0 \ \wedge \ n \geq 0$$

Estrutura do Algoritmo

Agora que temos todas as partes essenciais da análise do nosso algoritmo, podemos estruturar ele em forma de algoritmo como pseudo-código:

```
begin nomeAlgoritmo
  <Pré-condição do algoritmo>
  intruções pré-laço
  loop
    <invariante de laço>
    condição de parada
    instruções do loop
  end loop
  instruções do laço
  <pós-condição do algoritmo>
end nomeAlgoritmo
```

Seguindo essa estrutura, nosso pseudo-código fica assim:

```
begin Mult(m,n,p)
  <m >= 0, n >= 0>
  k = 0;
  loop
    <m >= 0, n >= 0>
    <m0 x n0 = mk x nk + pk>
    exit when m = 0;
    p = p + n;
    m = m - 1;
  end loop
  return p;
  <Mult(m,n,p) = m x n>
end
```

Conclusão

A visualização da análise do algoritmo fica assim:

$$Mult : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$$

$$Pré - condição : m \geq 0 \ , \ n \geq 0$$

$$Pós - condição : \forall m, n \in \mathbb{Z}.(m \geq 0 \land n \geq 0 \Rightarrow Mult(m, n, p) = m \times n)$$

$$\begin{cases} Mult(0, n, p) = p \\ Mult(m, n, p) = Mult(m - 1, n, p + n) \end{cases}$$

$$Invariante : m \geq 0 \ \land \ n \geq 0 \ \land \ m_0 \times n_0 = m_k \times n_k + p_k, k \in \mathbb{N} \ \land \ k \geq 0$$