

Quatro propriedades básicas

Obviamente, nem todas as receitas podem ser aceitas como descrições de algoritmos. Em particular, as instruções devem ser claras e executáveis de forma simples. Então, quando uma dada receita pode ser encarada como um algoritmo legítimo? Para tal, precisamos nos certificar de que a receita apresenta quatro propriedades básicas:

- A primeira é óbvia: o texto da receita deve ser *finito*. Soa estranho, pois como poderia o texto ser infinito? É uma precaução básica para evitar algoritmos “triviais”. Por exemplo, no caso do MDC, poderíamos ter um “algoritmo” com a seguinte descrição:

1. Se $M=1$ e $N=1$, o MDC é 1; páre.
2. Se $M=2$ e $N=1$, o MDC é 1; páre.
3. Se $M=1$ e $N=2$, o MDC é 1; páre.
4. Se $M=3$ e $N=1$, o MDC é 1; páre.
5. Se $M=2$ e $N=2$, o MDC é 2; páre.
-

que continuaria indefinidamente, listando todos os pares de possíveis valores para M e N , junto com o MDC correspondente. É certo que sempre encontraremos o MDC correto se “executarmos” esse algoritmo, com quaisquer valores dados para M e N , pois sempre vamos nos deparar com uma condição verdadeira quando atingirmos a linha onde estão listados os valores dados para M e N .

Obviamente, não queremos classificar essa receita infinita como um algoritmo legítimo. Nem conseguiríamos armazená-la num computador, quanto menos executá-la após lá armazenada ...

- O texto do algoritmo deve ser composto apenas por *instruções elementares*. Mas o que seria uma instrução elementar? Aqui, elementar depende muito do contexto. Por exemplo,
 - “juntar mais dois ovos” pode perfeitamente ser elementar para um cozinheiro. Mas pode não o ser para uma criança de cinco anos.
 - “substitua o valor de M pelo novo valor $(M-N)$ ” é elementar para quem domina aritmética básica e está à vontade com a noção de que “ M ” nada mais é que um local onde se pode armazenar números inteiros, de alguma forma, isto é, M é uma “variável”.
 - “se o valor do dólar for subir 10% no próximo mês *com certeza*, compre agora dez mil dólares”. Essa instrução não é elementar, nem mesmo para corretores experimentados. Não há como, *com certeza*, prever o valor do dólar no futuro, nem como adivinhar os números da próxima extração da loteria.

No nosso caso, então, o que seria permitido como uma instrução elementar? A resposta é simples: exatamente aquelas instruções que podem ser corretamente

interpretadas e executadas por um computador. Isto porque sempre desejaremos executar o algoritmo num computador digital.

Nesse ponto a situação se complica, pois cada arquitetura de computador define seu particular conjunto de instruções elementares válidas. Teríamos que conhecer em detalhes as instruções de cada uma dessas arquiteturas. Pior, se quisermos executar o algoritmo em máquinas diferentes, a receita teria que ser reescrita usando-se as instruções da nova máquina. Mais ainda, transcrever as ações do algoritmo nos termos das instruções válidas para uma máquina seria um processo extremamente trabalhoso e muito permeável a erros. Felizmente, essas agruras podem, hoje, ser evitadas através do uso de linguagens de programação de alto nível, como veremos.

Aliás, a maior parte desse curso discutirá, em detalhes, exatamente uma dessas linguagens de alto nível em duas versões, chamadas de C e C#. A última é uma versão mais moderna da primeira, com várias funcionalidades interessantes. Porém, ambas estão muito em voga ainda hoje, especialmente nas áreas tecnológicas e exatas, como engenharia.

- O texto do algoritmo deve ser uma *seqüência metódica e sistemática* de passos. Em particular, devemos sempre especificar a instrução inicial. Se nada for dito em contrário, esta será sempre entendida como a primeira instrução da seqüência dada.

Após cada instrução do algoritmo, deve estar claro qual deve ser a próxima instrução a ser executada na seqüência. Se nada for dito em contrário, a próxima instrução na ordem da seqüência dada será sempre entendida como a próxima instrução a ser executada. Esta já foi a sistemática adotada nos exemplos anteriores.

- O algoritmo deve *sempre parar*, após começar a executar com um *conjunto válido* de dados de entrada. Isto é, uma vez iniciada a execução, sempre vamos alcançar uma das instruções de “páre” espalhadas ao longo do texto da receita.

Esta é uma propriedade crucial. Não desejamos que o algoritmo fique executando indefinidamente, após iniciado com dados de entrada *válidos*. É também, muitas vezes, uma propriedade difícil de garantir que esteja sendo satisfeita. No entanto, se não for satisfeita, a receita não poderá se classificada como um algoritmo legítimo.

Note que se o algoritmo for iniciado com *dados de entrada inválidos*, então não é preciso garantir que venha sempre a terminar sua execução.

Melhor ilustrar com um exemplo.

- No caso do algoritmo para obter o MDC de dois inteiros positivos, podemos construir um argumento simples que sempre garante a parada do processo.

Observe que iniciamos na linha 1 com M e N contendo um valor positivo, ou seja, $M \geq 1$, $N \geq 1$ e $M+N \geq 2$.

Se executarmos a linha 2, temos $M > N \geq 1$, ou seja $M \geq 2$ e $(M-N) \geq 1$. Somando os novos valores de M e N após executarmos as ações previstas na linha 2 teremos $(M-N) + N = M$. Ou seja, de novo os valores armazenados em M e N permanecem pelo um e a soma desses novos valores permanece pelo menos 2. O caso da linha 3 é similar. Concluimos que após executar o passo 2 ou o passo 3, e retornar ao passo 1, sempre teremos $M \geq 1$, $N \geq 1$ e $M+N \geq 2$. Assim, sempre que estivermos prestes a executar o passo 1 do algoritmo sempre será verdade que $M \geq 1$, $N \geq 1$ e $M+N \geq 2$.

Algoritmos: o que os caracteriza?

Mas observe que, como $M > N$ e $N \geq 1$, no caso de executarmos a linha 2, então a soma dos novos valores $(M-N)+N$ será *estritamente menor* que a soma antes de executarmos a ação desta linha.

Concluimos que o algoritmo não pode executar indefinidamente. Por um lado, o valor de $M+N$ decresce a cada iteração e retorno à linha 1. Por outro lado devemos manter a relação $M+N \geq 2$ no retorno à linha 1. Claramente, não podemos executar a linha 2, ou a linha 3, mais de $M+N-2$ vezes, onde M e N aqui representam os valores iniciais de M e de N , respectivamente.

Portanto, o algoritmo é forçado a parar, executando a ação da linha 1, única alternativa que resta.

- Examinemos agora o comportamento do algoritmo quando o iniciamos com dados de entrada que não são válidos. Por exemplo, vamos assumir que M e N contêm os valores 3 e -1. Em particular o valor de N é inválido. A execução do algoritmo produziria a seguinte tabela de passos:

Passo	Linha	M	N	Comentários
1	1	3	-1	-1 <> 3
2	2	4	-1	3 > -1; 3 - (-1) = 4
3	1	4	-1	-1 <> 4
4	2	5	-1	4 <> -1; 4 - (-1) = 5
5	1	5	-1	-1 <> 5
6	2	6	-1	5 <> -1; 5 - (-1) = 6

Inicialmente, no passo 1 teremos $M=3$ e $N=-1$, ou seja M e N não são iguais. Isso forçaria a execução da linha 2 no passo 2, conforme mostra a tabela. Como $M > N$ nesse ponto, o novo valor de M seria $M-N=3-(-1)=4$, retornando ao passo 1. De novo $M > N$ nesse ponto, e executaríamos a linha 2 de novo, armazenando em M o novo valor $M-N=4-(-1)=5$. E assim por diante. Ou seja, o valor de M cresce indefinidamente e nunca teremos $M=N$ no início da linha 1. Portanto, o algoritmo não pára de executar ao ser iniciado com os valores de M e N em 3 e -1, respectivamente.

Significa que o algoritmo não é correto? Não, o algoritmo é correto, pois sempre produz a resposta desejada *quando os dados de entrada são válidos*. O algoritmo simplesmente não foi projetado para lidar com os casos em que M , ou N , não são positivos. Um algoritmo modificado poderia fazê-lo, dada uma definição para o MDC de números não positivos, incluindo o zero.