INSTITUTO FEDERAL
Paraná

Campus
Paranavaí

Matemática Discreta e Lógica

Prof. Me. Azuaite A. Schneider

IFPR

5 de setembro de 2024

Introdução

Seja $P(n)$ uma sentença matemática que depende de uma variável natural n, a qual se torna verdadeira ou falsa quando substituímos n por um número natural dado qualquer. Estas sentenças são chamadas sentenças abertas definidas sobre o conjunto dos números naturais $\mathbb{N}$.

Exemplos:

1. $P(n)$: "n é ímpar."
2. $P(n)$: "$n^2 - n + 41$ é um número primo."
3. $P(n)$: "$2n + 6$ é par."

4. $P(n)$: “$1 + 3 + 5 + \cdots + (2n + 1) = (n + 1)^2$”. Será que conseguiremos encontrar algum m tal que $P(m)$ seja falso?

Indução

O *Método de Indução* é um procedimento matemático para provar propriedades que são verdadeiras para uma sequência de objetos.

Para ilustrar o uso desta técnica, imagine que você está subindo em uma escada sem fim. Como você pode saber se será capaz de alcançar um degrau arbitrariamente alto? Suponha que você faça as seguintes afirmações sobre as suas habilidades de subir escadas:

1. Você pode alcançar o primeiro degrau.
2. Se você alcançar um degrau, você pode sempre passar ao degrau seguinte. (Note que esta afirmação é uma implicação.)

Se tanto a sentença 1 como a implicação na sentença 2 são verdadeiras; então, pela sentença 1 você pode chegar ao primeiro degrau e pela sentença 2 pode chegar ao segundo; novamente pela 2 pode chegar ao terceiro; pela sentença 2 novamente pode chegar ao quarto degrau, e assim sucessivamente. Você pode, então, subir tão alto quanto você queira.

Neste caso, ambas as afirmações são necessárias. Se apenas a sentença 1 é verdadeira, você não tem garantias de que poderá ir além do primeiro degrau, e se apenas a segunda sentença é verdadeira, você poderá não chegar ao primeiro degrau a fim de iniciar o processo de subida da escada.

Vamos assumir que os degraus da escada são numerados com os números naturais positivos (do conjunto $\mathbb{N}^*$) dado por $1, 2, 3$, e assim por diante

Agora, vamos considerar uma propriedade especifica que um número pode ter. Ao invés de "alcançarmos um degrau arbitrário" podemos mencionar que um número natural positivo arbitrário tem essa propriedade. Usaremos a notação simplificada $P(n)$ para denotar que um número natural positivo n tem a propriedade P. Como podemos usar a técnica de subir escadas para provar que para todos naturais positivos n temos $P(n)$?

Observação

O conjunto dos números naturais $0, 1, 2, 3, ...$ é denotado por $\mathbb{N}$. O símbolo $\mathbb{N}^*$, com o asterisco do lado direito, denota o conjunto $1, 2, 3, ...$, dos números naturais sem o zero. Quando queremos dizer que um elemento x pertence ao conjunto dos números naturais usamos a simbologia $x \in \mathbb{N}$.

De forma análoga ao que ilustramos com a escada, as duas afirmações de que precisamos para a demonstração são:

1. $P(1)$ (1 tem a propriedade P.)
2. Para qualquer $k \in \mathbb{N}^*$, $P(k) \rightarrow P(k+1)$ (Se algum número tem a propriedade P, então o número seguinte também a tem.)

Se ambas as proposições 1 e 2, puderem ser provadas, então $P(n)$ é válida para qualquer inteiro positivo n, da mesma forma que poderíamos subir até um degrau arbitrário da escada.

O fundamento para argumentos desse tipo é o *primeiro princípio da indução matemática*. A primeira afirmação, não precisa ser válida necessariamente para o número 1, ela pode ser válida para um número natural específico n_0 e então a segunda deve ser válida para qualquer número natural n que seja maior que n_0. Mais especificamente, a primeiro princípio da indução matemática é enunciado da seguinte forma:

Primeiro Princípio da Indução Matemática (PFI 1)

Seja $P(n)$ um enunciado que descreve uma propriedade sobre um número $n \in \mathbb{N}^*$ maior ou igual a um número natural n_0 fixado.

Se pudermos provar que valem as duas condições:

C1: $P(n_0)$ é verdadeira (ou seja, vale a propriedade P para n_0)

C2: É verdadeira a implicação $P(n) \rightarrow P(n+1)$ para todo $n \geq n_0$

Então podemos afirmar que a propriedade $P(n)$ é verdadeira para todo $n \geq n_0$

No caso de C2, como uma implicação só é falsa se sua premissa (termo antecedente) for verdadeira e a conclusão (termo consequente) falsa, basta excluir essa possibilidade para termos a validade da implicação desejada. Assim o que normalmente se faz é tomar um número natural positivo k genérico qualquer, maior ou igual a n_0 e admitindo que $P(k)$ seja verdadeiro, mostrar que necessariamente $P(k+1)$ também deve ser verdadeiro.

Essa prova, junto com a verificação de C1, garante a validade da propriedade P para qualquer número natural maior que n_0.

Mais especificamente, os passos para a demonstração por indução são:

1. **Base**: Verificar que a propriedade vale para $n_0 = 1$ (ou outro valor de n_0 se for o caso), isto é, $P(1)$ é válida.

2. **Hipótese da indução**: Supor que $P(k)$ é válida, ou seja, que a propriedade P é válida para um k qualquer.

3. **Tese**: Provar que $P(k+1)$ é válida usando a hipótese da indução.

Exemplo

Prove que a equação

$$1 + 3 + 5 + \ldots + (2n - 1) = n^2 \tag{1}$$

é verdadeira para qualquer número natural positivo n.

Neste caso, a propriedade $P(n)$ é que a equação (1) é verdadeira.

O lado esquerdo desta equação é a soma dos n primeiros números ímpares, ou seja, todos os números ímpares de 1 ate $2n - 1$. Essa equação diz que essa soma é igual ao quadrado da quantidade de números que estamos somando.

Por exemplo, se $n = 3$, isto é, se somamos os 3 primeiros números ímpares temos $1+3+5 = 9$, que é exatamente igual ao lado direito $n^2 = 3^2 = 9$.

Ainda que possamos verificar que a equação é verdadeira para um particular valor de n, pela substituição deste valor na equação, não podemos substituir todos os possíveis valores de n para provar que ela sempre é verdadeira, dado que existem infinitos números naturais. Por isso, uma demonstração por exemplos não funciona. É mais apropriada uma demonstração por indução matemática.

Demonstração

1. **Base**: Para $n_0 = 1$ temos que $P(1)$: $\quad 1 = n_0^2 = 1$

   $P(1)$ claramente é verdadeira, ou seja, se somarmos o primeiro número ímpar ele é igual ao quadrado de 1 que foi a quantidade de termos que somamos.

2. **Hipótese de Indução**: Suponha que $P(k)$ é válida para um número k natural qualquer. Ou seja, é válida a equação

$$P(k): \quad 1 + 3 + 5 + \ldots + (2k - 1) = k^2 \tag{2}$$

3. **Tese:** Mostremos que a equação vale para o termo seguinte, isto é, que $P(k+1)$ também é válida.

   Obtemos a expressão para $P(k+1)$ somando mais um termo a expressão em (2). Assim

$$P(k+1): \quad 1+3+5+\ldots+(2k-1)+[2(k+1)-1] = (k+1)^2 \tag{3}$$

   O que queremos mostrar então, é que a equação (3) é válida.

   Vamos usar a hipótese de indução do lado esquerdo da equação (3) para provar o que queremos.

Pela hipótese de indução, a equação (2) é válida, logo podemos trocar $1+3+5+\ldots+(2k-1)$ do lado esquerdo da equação (3) por k^2. Assim o lado esquerdo da equação fica da forma

$$\underbrace{1+3+5+\ldots+(2k-1)}_{\text{igual a } k^2}+[2(k+1)-1] = k^2+[2(k+1)-1]$$

$$= k^2+2k+1$$

$$= (k+1)^2$$

Note que ao trocar $1 + 3 + 5 + \ldots + (2k - 1)$ por k^2, no lado esquerdo da equação, obtemos $k^2 + [2(k + 1) - 1]$, que por sua vez é igual a $k^2 + 2k + 1$ e que por fim, é igual a $(k + 1)^2$.

Logo

$$1 + 3 + 5 + \ldots + (2k - 1) + [2(k + 1) - 1] = (k + 1)^2,$$

o que mostra a que $P(k + 1)$ é verdadeira, provando assim que C2 é válida e que portanto a equação (1) é verdadeira para qualquer número natural positivo n.

Exemplo

Prove que, para $n \in \mathbb{N}^*$, que o número $2^{2n} - 1$ é divisível por 3 .

Seguiremos os passos da demonstração por indução.

1. **Base**: Para $n_0 = 1$ temos que

$$P(1): \quad 2^{2 \cdot 1} - 1 = 2^2 - 1 = 4 - 1 = 3$$

   é verdadeira, pois 3 é divisível por 3.

2. **Hipótese de Indução**: Suponha que $P(k)$ é válida para um número k natural qualquer. Ou seja $2^{2k} - 1$ é divisível por 3.

Isso equivale a escrever $2^{2k} - 1 = 3m$ para algum inteiro m, ou ainda passando o 1 para o lado direito da equação, equivale a dizer que

$$2^{2k} = 3m + 1, \qquad \text{com } m \in \mathbb{N} \tag{4}$$

3. **Tese**: Mostremos que $P(k+1)$ também é válido, ou seja, que $2^{2(k+1)} - 1$ é divisível por 3.

Veja que

$$
\begin{aligned}
2^{2(k+1)} - 1 &= 2^{2k+2} - 1 \\
&= 2^2 \cdot 2^{2k} - 1 && \text{(Usando propriedade da potênciação)} \\
&= 2^2 \cdot (3m+1) - 1 && \text{(Usando a equação (4) e trocando} \\
& && 2^{2k} \text{ por } 3m+1) \\
&= 4 \cdot (3m+1) - 1 \\
&= 12m + 4 - 1 && \text{(Usando propriedade distributiva)} \\
&= 12m + 3 \\
&= 3(4m+1) && \text{(Colocando 3 em evidência)}
\end{aligned}
$$

Assim $2^{2(k+1)} - 1 = 3(4m+1)$, ou seja, é um múltiplo 3, uma vez que ele é igual a multiplicação do 3, pelo número natural $(4m + 1)$ e portanto divisível por 3.

Logo por indução, mostramos que $2^{2n} - 1$ é divisível por 3.

Exemplo

Prove que a propriedade $n < 2^n$ vale para qualquer $n \in \mathbb{N}^*$.

Seguiremos os passos da indução:

1. **Base**: Para $n_0 = 1$ temos que $P(1)$ é válida, pois $1 < 2^1$ é verdade.

2. **Hipótese de Indução**: Suponha que, para algum $k \in \mathbb{N}^*$, tem-se que:

$$P(k): \quad k < 2^k \text{ é verdadeira}$$

3. **Tese**: Mostremos que $P(k+1)$ é válida, isto é, que $k+1 < 2^{k+1}$.
   Faremos isso usando a hipótese de indução

$$
\begin{aligned}
k &< 2^k && \text{(Hipótese de indução)} \\
k+1 &< 2^k + 1 && \text{(Somamos 1 dos dois lados da equação)} \\
k+1 < 2^k + 1 &\leq 2^k + 2^k && \text{(Como } 1 < 2^k,\ \forall k \in \mathbb{N}, \text{ tem-se } 2^k + 1 \leq 2^k + 2^k\text{)} \\
k+1 &< 2^k + 2^k && \text{(Só suprimindo a expressão do meio da desigualdade)} \\
k+1 &< 2 \cdot 2^k && \text{(}2^k + 2^k \text{ é igual a } 2 \cdot 2^k, \text{ pois quando somo duas} \\
& && \text{coisas iguais o resultado é 2 vezes essa coisa.)} \\
k+1 &< 2^{k+1} && \text{(Usando prop. de potências temos que } 2^1 \cdot 2^k = 2^{k+1}\text{)}
\end{aligned}
$$

Com essa sequência de passos, mostramos portanto, que $P(k+1)$ é válida e logo a propriedade $n < 2^n$ vale para qualquer $n \in \mathbb{N}^*$.

Recursividade e Relações de Recorrência

Uma definição na qual o objeto que está sendo definido aparece como parte da definição é chamada definição indutiva ou definição recursiva.

Definições recorrentes

As definições recursivas são compostas de duas partes:

1. uma base, onde alguns casos simples do item que está sendo definido são dados explicitamente;

2. um passo indutivo ou recursivo, onde outros casos do item que está sendo definido são dados em termos dos casos anteriores.

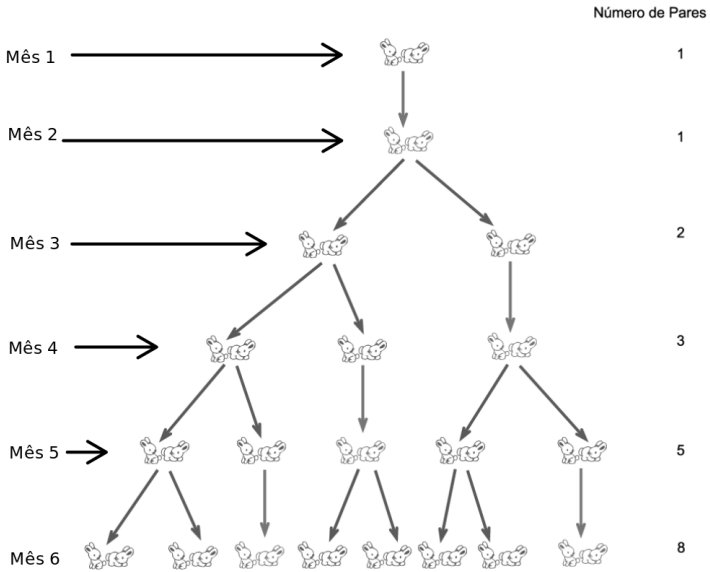
Exemplo – Sequência de Fibonacci

A famosa sequência de Fibonacci é um exemplo de um objeto que é definido recursivamente.

Vamos apresentá-la por meio do seguinte problema:

“Um homem colocou um casal de coelhos em um lugar cercado por todos os lados por um muro. Quantos casais de coelhos podem ser gerados a partir desse casal em um ano se, supostamente, todos os meses cada casal dá à luz um novo casal, que é fértil a partir do segundo mês?”

A figura 1 ilustra essa situação até o sexto mês.


Número de Pares
Mês 1
1
Mês 2
1
Mês 3
2
Mês 4
3
Mês 5
5
Mês 6
8

Observe que

- no mês 1 temos o casal de coelhos inicial;
- no mês 2 temos apenas o mesmo casal de coelhos, já que eles ainda não geraram novos coelhos;
- no mês 3 temos primeiro casal de coelho anterior e um novo casal, filhos deles, totalizando 2 casais.
- no mês quatro, temos o primeiro casal e mais um casal filho deles e o casal que nasceu no segundo mês, totalizando 3 casais.

Observe que a quantidade de casais do terceiro mês, é igual a soma da quantidade de casais do primeiro e do segundo mês. E que a quantidade de casais do quarto mês, é igual a soma da quantidade de casais do segundo e do terceiro mês. E isso ocorre nos meses seguintes, ou seja, a quantidade de casais do mês é sempre igual a soma da quantidade de casais dos dois meses anteriores.

A sequência da quantidade de casais de cada mês, matematicamente, pode ser ser definida por duas partes. Denotando por $F(n)$ a quantidade de casais do mês n temos

1. $F(1) = 1$ e $F(2) = 1$

2. $F(n) = F(n-2) + F(n-1)$ para $n > 2$

Os termos $F(1) = 1$ e $F(2) = 1$ são a **base**, ou seja, os casos simples da sequência que está sendo definida e são dados explicitamente.

A relação "$F(n) = F(n-2) + F(n-1)$ para $n > 2$", nos dá um **passo recursivo**, que diz que o número de casais de um mês, a partir do terceiro mês, é definido em termos dos dois meses anteriores.

Tendo calculado o número de casais do terceiro e do quarto mês, por exemplo, o número de casais do quinto mês, de acordo com o passo 2, será dado por

$$F(5) = F(3) + F(4) = 2 + 3 = 5.$$

Se continuarmos esse processo, teremos a sequência de *Fibonacci* dada por

$$1, \quad 1, \quad 2, \quad 3, \quad 5, \quad 8, \quad 13, \quad 21, \quad 34, \quad 55, \quad 89, \quad \ldots$$

A parte 1 nos fornece então, um ponto de partida na medida em que trata alguns casos simples; enquanto a parte 2 nos permite construir novos casos a partir desses casos simples, para então construir outros casos a partir desses novos, e assim por diante.

A recursão é uma ideia importante que pode ser utilizada para definir sequências de objetos, coleções mais gerais de objetos, operações sobre objetos e até mesmo algoritmos.

Sequências definidas por recorrência

De maneira geral, uma **sequência** S é uma lista de objetos que são enumerados segundo alguma ordem; existe um primeiro objeto, um segundo, e assim por diante. $S(k)$ denota o k-ésimo objeto da sequência.

Uma sequência é definida recursivamente, explicitando-se seu primeiro valor (ou seus primeiros valores) e, a partir daí, definindo-se outros valores na sequência em termos dos valores iniciais, assim como foi feito, por exemplo, na sequência de *Fibonacci*.

Exemplo

Considere a sequência S definida recursivamente por:

1. $S(1) = 4$
2. $S(2) = 3 \cdot S(n-1)$ para $n \geq 2$

A parte 1 diz que o primeiro objeto da sequência é 4 e a parte 2 diz que um objeto, a partir do segundo, é sempre obtido do anterior multiplicado por 3. Assim

- $S(2) = 3 \cdot S(1) = 3 \cdot 4 = 12$
- $S(3) = 3 \cdot S(2) = 3 \cdot 12 = 36$
- $S(4) = 3 \cdot S(3) = 3 \cdot 36 = 108$

e assim por diante, obtemos a sequência

$$4, \quad 12, \quad 36, \quad 108, \quad 324, \quad 972...$$

Exemplo

Uma regra semelhante à dada no passo 2 do exemplo anterior, na qual se define um valor da sequência a partir de um ou mais valores anteriores, é chamada de **relação de recorrência**.

Considere a sequência T definida recursivamente como se segue:

1. $T(1) = 2$
2. $T(n) = T(n-1) + 5$ para $n \geq 2$

A parte 1, diz que o primeiro objeto da sequência é 2, e a parte 2, diz que um objeto, a partir do segundo, é obtido do anterior somando-se 5. Temos então a sequência

$$2, \quad 7, \quad 12, \quad 17, \quad 22, \quad 27, \quad , 32, ...$$

Conjuntos definidos por recorrência

Os objetos de uma sequência são ordenados, isto é existe um primeiro objeto, um segundo, e assim por diante. Um conjunto de objetos é uma coleção de objetos na qual nenhuma regra de ordenação é imposta. Alguns conjuntos podem ser definidos recursivamente.

Exemplo

Uma definição por recorrência para um conjunto de pessoas que são ancestrais de João poderia ter a seguinte base e passo recursivo.

1. Os pais de João são seus ancestrais.
2. Os pais de um ancestral de João são seus ancestrais.

Pelo passo 1, teríamos que os pais de João são ancestrais dele. Pelo passo 2, os pais dos pais de João, são ancestrais dele. Pelo passo 2 novamente, os pais dos pais dos pais de João são seus ancestrais e assim por diante temos o conjunto de todos os ancestrais de João.

Exemplo

Em lógica podemos definir o conjunto das *wff's* por recursividades da seguinte maneira.

1. Qualquer proposição simples é uma *wff*.

2. Se P e Q são *wff's*, então também o serão *wff's* $(P \wedge Q), (P \vee Q), (P \rightarrow Q), (\neg P)$ e $(P \leftrightarrow Q)$.

   Se considerarmos A e B como proposições simples, então pelo passo 1, elas são *wff's*. Pelo passo 2, $(A \wedge B), (A \vee B), (A \rightarrow B), (\neg A)$ e $(A \leftrightarrow B)$ também serão *wff's*.

Pelo passo 2 novamente, uma vez que $(A \wedge B), (A \vee B), (A \rightarrow B), (\ A)$ e $(A \leftrightarrow B)$ são *wff's*, então, por exemplo

$$(A \wedge B) \rightarrow (A \rightarrow B)$$

também é uma *wff*.

E assim, por diante, podemos construir *wff's* mais complexas, a partir das anteriores.

Operações definidas por recorrência

Algumas operações sobre objetos podem ser definidas recursivamente.

Dado um número real "a" diferente de zero e um número inteiro não-negativo n, a operação de potenciação a^n, é definida como a multiplicação do número a por ele mesmo n vezes.

Por exemplo, $3^5 = 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3$.

Uma definição recursiva para a potenciação a^n é:

1. $a^0 = 1$
2. $a^n = \left(a^{n-1}\right) \cdot a \quad$ para $n \geq 1$

Vamos tomar, por exemplo, $a = 3$.

O primeiro passo da recursividade diz que $3^0 = 1$.

O segundo passo diz que $3^1 = 3^0 \cdot 3 = 1 \cdot 3 = 3$

Novamente pelo segundo passo, $3^2 = 3^1 \cdot 3 = 3 \cdot 3 = 9$

De novo pelo segundo passo, $3^3 = 3^2 \cdot 3 = 9 \cdot 3 = 27$.

E assim por diante.

Algoritmos definidos por recorrência

Um **algoritmo** é um conjunto finito de instruções precisas para realizar uma operação computacional ou para resolver um problema.

Um **algoritmo** é **recursivo** se resolver um problema reduzindo-o a um mesmo problema com valores iniciais menores.

Vamos tomar por exemplo, a sequência S definida no exemplo dado anteriormente.

1. $S(1) = 4$
2. $S(n) = 3 \cdot S(n-1)$, para $n \geq 2$

Suponha que desejamos escrever um algoritmo para obter $S(n)$ para algum inteiro positivo n. Podemos utilizar duas abordagens diferentes. Se desejamos encontrar $S(12)$, por exemplo, podemos começar com $S(1) = 2$ e então calcular $S(2)$, $S(3)$, e assim por diante, como feito no exemplo até que finalmente obtemos $S(12)$.

Esta abordagem envolve **iterações** através de uma espécie de **laço**. No slide seguinte apresentamos um pseudocódigo S que usa esse algoritmo interativo.

Algoritmo 1

```
S(inteiro n)

Variáveis locais:
inteiro i                 //índice do laço
Valor_Atual               // valor atual da função S

   se  n=1 então
      retorne 4
   senão
      i=2
      Valor_Atual =4
      enquanto  i< = n faça
         Valor_Atual =3  *  Valor_Atual
         i=i+1
      fim do enquanto //agora Valor_Atual tem o valor S(n)
      retorne Valor_Atual
   fim do se
fim da função S
```

Suponha por exemplo, que o usuário tenha fornecido o valor $n = 4$.

Como $n > 1$, entra-se na cláusula **senão**, que atribui os valores $i = 2$ e $Valor\_Atual = 4$ e em seguida entra no laço **enquanto** que contém instruções que serão executadas enquanto $i \leq 4$.

Dentro do laço é executado

$$Valor\_Atual = 3 \cdot Valor\_Atual = 3 \cdot 4 = 12 \quad e \quad i = i + 1 = 2 + 1 = 3$$

Dessa forma, a variável *Valor_Atual* assume agora o valor 12 e a variável *i* assume o valor 3.

Como $i \leq 4$ executa-se novamente as instruções do laço **enquanto**.

Assim é realizada a operação

$$Valor\_Atual = 3 \cdot Valor\_Atual = 3 \cdot 12 = 36 \quad e \quad i = i + 1 = 3 + 1 = 4$$

Como $i = 4$ executa-se pela última vez as instruções do laço **enquanto**.

Assim é realizada a operação

$$Valor\_Atual = 3 \cdot Valor\_Atual = 3 \cdot 36 = 108 \quad e \quad i = i + 1 = 4 + 1 = 5$$

Veja que agora, como $i = 5$, não são mais executadas as instruções do laço **enquanto**. É retornado então, o último valor que a variável *Valor_Atual* assumiu, ou seja, 108 e é encerrada a cláusula **se** e a função S.

A segunda abordagem para calcular $S(n)$ usa a definição de recursão de S diretamente. A versão a seguir é de um algoritmo recursivo, escrito como uma função em pseudocódigo.

```
Algoritmo 2

S(inteiro n)
    se n=1 então
        retorne 4
    senão
        retorne 2 * S(n-1)
    fim do se

fim da função S
```

O corpo desta função consiste em uma única proposição do tipo **se - então - senão**. Para entendermos o seu funcionamento, vamos acompanhar sua execução para calcular $S(4)$.

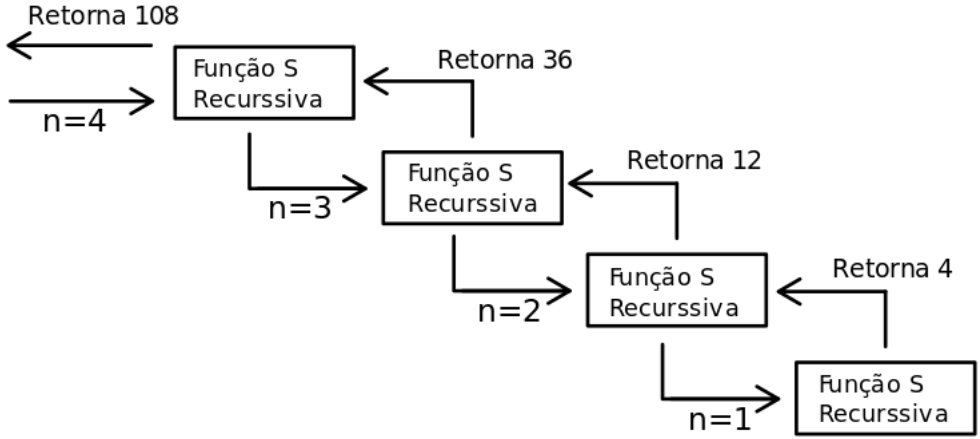
A função é inicialmente chamada para o valor de entrada $n = 4$. Como n é diferente de 1, a execução é direcionada para a cláusula **senão**. Neste ponto o procedimento para calcular $S(4)$ é momentaneamente suspenso, até que o valor de $S(3)$ seja conhecido. Qualquer informação relevante para a obtenção de $S(4)$ é armazenada na memória do computador numa pilha, para ser posteriormente carregada quando o cálculo puder ser completado. (Uma pilha é uma coleção de dados onde qualquer novo item ao entrar é armazenado no topo, e somente o item do topo da pilha pode ser removido ou acessado.)

A função automaticamente é novamente chamada para um valor de entrada $n = 3$. Novamente a cláusula **senão** é executada e o cálculo de $S(3)$ é momentaneamente suspenso com as informações relevantes armazenadas na pilha, enquanto a função é chamada novamente, agora com $n = 2$ como entrada.

Mais uma vez, a cláusula **senão** é executada e o cálculo de $S(2)$ é momentaneamente suspenso com as informações relevantes armazenadas na pilha, enquanto a função é chamada novamente, agora com $n = 1$ como entrada.

Desta vez a cláusula **se** é executada e retorna o valor 4. Esta última chamada da função está completa e o valor 4 é usado na penúltima chamada da função, que pode recuperar qualquer informação relevante para o caso $n = 2$ da pilha, calcular $S(2)$ e disponibilizar o resultado à chamada anterior (n=3). É calculado então $S(3)$, que por sua vez é usado para calcular $S(4)$. Finalmente, esta chamada original de S é capaz de esvaziar a pilha e concluir seus cálculos, retornando o valor de $S(4)$.

O diagrama da figura 1 ilustra como o processo ocorre para o exemplo que estamos utilizando.


Retorna 108
Função S Recurssiva
n=4
Retorna 36
Função S Recurssiva
n=3
Retorna 12
Função S Recurssiva
n=2
Retorna 4
Função S Recurssiva
n=1

Figura

Quais são as vantagens e desvantagens de algoritmos iterativo e recursivo para executar uma determinada tarefa?

Neste exemplo, a versão recursiva é menor, pois dispensa o gerenciamento do laço. Descrever a execução de uma versão recursiva é mais complexo do que descrever a versão iterativa, porém todos os passos são realizados automaticamente. Não temos contato com o que está acontecendo internamente, é necessário apenas termos em conta que uma quantidade grande de chamadas da função pode demandar muita memória para o armazenamento na pilha das informações necessárias às chamadas anteriores. Se for necessária mais memória do que o que se tem disponível, ocorre um "transbordamento" da pilha. Além de usar muita memória, algoritmos recursivos podem demandar muitos outros procedimentos computacionais e, por isso, podem ser mais lentos do que os não-recursivos.

De alguma forma, a recursão fornece uma maneira natural de se abordar uma grande variedade de problemas, alguns dos quais poderiam ter soluções não-recursivas extremamente complexas.

Problemas onde se desejam calcular valores para uma sequência definida recursivamente são naturalmente escritos em algoritmos recursivos. Muitas linguagens de programação permitem o uso de recursão, porém não todas.