MC102 — Recursão

Rafael C. S. Schouery
rafael@ic.unicamp.br

Universidade Estadual de Campinas

Atualizado em: 2023-03-02 14:31

Recursão

Recursão é o conceito de uma função chamar a si mesma para resolver algum problema computacional

- Sabemos resolver instâncias pequenas do problema
  - Chamamos de caso base
- Sabemos resolver uma instância maior a partir das menores
  - Chamamos de caso geral

Por exemplo, para calcular $n! = \prod_{i=1}^{n} i$:

- Base: Sabemos resolver $0!$, pois $0! = 1$
- Geral: Sabemos resolver $n!$ a partir de $(n-1)!$ pois

$$n! = \prod_{i=1}^{n} i = n \prod_{i=1}^{n-1} i = n(n-1)!$$

Exemplo: Fatorial

Calculando $n!$ recursivamente:

```python
def fatorial(n):
    if n == 0:
        return 1
    else:
        return n * fatorial(n - 1)
```

Mas como isso pode funcionar se a função chama a si mesma?

O Python sabe a linha de código que fez a chamada de função

- Isso gera a pilha de chamadas
- Quando uma função é chamada, ela vai "em cima" da atual

O Python sabe também qual era o valor das variáveis locais

- Ele consegue restaurar esses valores quando voltar

Exemplo: Fatorial

Calculando $n!$ recursivamente:

```
def fatorial(n):
    if n == 0:
        return 1
    else:
        return n * fatorial(n - 1)
```

```
fatorial        n: 4
  return n * 6
```

Exemplo: Último termo da PA

Uma Progressão Aritmética (PA)

- é uma sequência de números $(a_1, a_2, \ldots, a_n)$
- onde existe um número r tal que
- $a_i = a_{i-1} + r$, para todo $1 < i \leq n$

Queremos um algoritmo recursivo para calcular a_n

- **Base**: Se $n = 1$, $a_n = a_1$
- **Geral**: Se $n > 1$, $a_n = a_{n-1} + r$
  - Estamos indo de uma instância maior para uma menor

Solução:

```python
def termo(a_1, r, n):
    '''Calcula o n-ésimo termo de uma PA
    iniciando em a_1 com razão r'''
    if n == 1:
        return a_1
    else:
        return r + termo(a_1, r, n - 1)
```

Exemplo: Progressão aritmética com impressão

E se quisermos imprimir a progressão aritmética?

```python
def pa(a_1, r, n):
    if n == 1:
        print(a_1)
        return a_1
    else:
        atual = r + pa(a_1, r, n - 1)
        print(atual)
        return atual
```

Vamos depurar!

Exemplo: Fibonnaci

A sequência de Fibonnaci é: 1, 1, 2, 3, 5, 8, ...

- Começa com 1, 1
- Cada elemento a seguir é a soma dos dois anteriores

Ou seja,

$$f(n) = \begin{cases} f(n-1) + f(n-2), & \text{se } n > 2 \\ 1, & \text{se } n = 1 \text{ ou } n = 2 \end{cases}$$

É o que chamamos na matemática de recorrência

- Uma função definida recursivamente
- $n!$ é outro exemplo de recorrência

Note que temos dois casos bases e um caso geral

- E que resolvemos uma instância a partir de duas menores

Exemplo: Fibonnaci

```python
def fib(n):
    if n == 1 or n == 2:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)
```

```
fib              n:  5
   return 3 + fib(n - 2)
```

Exemplo: Algoritmo de Euclides

x é um divisor de y se existe um inteiro k tal que $y = k \cdot x$

- O máximo divisor comum de a e b, denotado por $mdc(a, b)$ é o maior inteiro que divide a e b simultaneamente

Qual o $mdc(a, 0)$?

- Qualquer número x é divisor de 0, pois $0 = 0 \cdot x$

Além disso, se x divide a e b, então x divide $a \bmod b$

- $a = k \cdot b + r$ e, portanto, $a \bmod b = r = a - kb$
- $a = q_a \cdot x$ e $b = q_b \cdot x$
- $a \bmod b = a - k \cdot b = q_a \cdot x - k \cdot q_b \cdot x = (q_a - k \cdot q_b) \cdot x$

Assim:

$$mdc(a, b) = \begin{cases} mdc(b, a \bmod b), & \text{se } b \neq 0 \\ a, & \text{se } b = 0 \end{cases}$$

Exemplo: Algoritmo de Euclides

$$mdc(a, b) = \begin{cases} mdc(b, a \bmod b), & \text{se } b \neq 0 \\ a, & \text{se } b = 0 \end{cases}$$

Temos um caso base e um caso geral:

- Para usar recursão, temos que ir do maior para o menor
  - E chegarmos na base em algum momento
- Note que $a \bmod b < b$
  - Ou seja, estamos sempre diminuindo o segundo termo

```python
def mdc(a, b):
    if b == 0:
        return a
    else:
        return mdc(b, a % b)
```

Exemplo: Algoritmo de Euclides

```python
def mdc(a, b):
    if b == 0:
        return a
    else:
        return mdc(b, a % b)
```

Um mal exemplo de recursão: $3n + 1$

Considere a seguinte sequência de números:

- Comece com um número a_0 a sua escolha
- Se $a_i = 1$, pare
- Se a_i é par, então $a_{i+1} = a_i/2$
- Se a_i é impar, então $a_{i+1} = 3 \cdot a_i + 1$

A Conjectura de Collatz é que, para qualquer a_0 escolhido, essa sequência sempre termina

- Isto é, chegamos em $a_i = 1$

Podemos fazer um código em Python que gera essa sequência

- Mas não temos certeza se a execução terminaria...
- Afinal, não sabemos se a conjectura é verdadeira...

O problema é que não estamos indo de uma instância maior para uma menor quando a_i é ímpar!

Dicas

- Você sempre precisa ter pelo menos um caso base!
  - Senão, você alcança um caso pequeno que sabe resolver
- Você precisa sempre chegar em algum caso base!
  - Você precisa ir da instância maior para a menor
    - Se crescer pode nunca chegar na base!
    - Se continuar igual, irá ciclar!

Veja esse exemplo:

```python
def fib_defeituoso(n):
    if n == 1:
        return 1
    else:
        return fib_defeituoso(n - 1) + fib_defeituoso(n - 2)
```

O que acontece para `n = 2`?

- Nunca atingimos uma base para `fib_defeituoso(0)`...

Exercícios

1. Faça uma função recursiva que calcula a soma dos números naturais menores ou iguais a n.
2. Faça uma função recursiva que calcula a soma dos números naturais ímpares menores ou iguais a n.
3. Faça uma função recursiva que calcula a soma de uma PA com valor inicial a_1, razão r e n termos.
4. Faça uma função recursiva para contar quantos dígitos um número inteiro positivo tem.
5. Faça uma função recursiva que, dada uma string representando um número inteiro positivo em binário, acha o seu valor em decimal.
6. Faça uma função recursiva que, dada um número inteiro positivo, acha o seu valor em binário (em uma string).

Recursão com listas

Até o momento, vimos o uso de recursão para operações matemáticas

- Fibonnaci, mdc, PA, etc

Mas podemos usar recursão para diversas tarefas computacionais

- Como, por exemplo, algoritmos que lidam com listas

Digamos que queremos imprimir uma lista:

- Base: Se a lista for vazia, não temos nada para imprimir
- Geral: Imprimimos o primeiro elemento e imprimimos o resto recursivamente

```
def imprime(l):
    if len(l) == 0:
        print()
    else:
        print(l[0], end=' ')
        imprime(l[1:])
```

Exemplo: Imprimindo uma lista

```python
def imprime(l):
    if len(l) == 0:
        print()
    else:
        print(l[0], end=' ')
        imprime(l[1:])
```

Esse código é ruim porque estamos usando slices

- Cada slice é uma nova cópia da lista
- O que gasta espaço na memória e tempo para a cópia

Uma versão melhor:

```python
def imprime_rec(l, n):
    if n > 0:  # a base n == 0 está implicita
        imprime_rec(l, n - 1)
        print(l[n - 1], end=' ')

def imprime(l):
    imprime_rec(l, len(l))
    print()
```

Exemplo: Imprimindo uma lista

```python
def imprime_rec(l, n):
    if n > 0:  # a base n == 0 está implicita
        imprime_rec(l, n - 1)
        print(l[n - 1], end=' ')

def imprime(l):
    imprime_rec(l, len(l))
    print()
```

A recursão é no tamanho `n` da sublista a ser impressa:

- Base: Temos `n == 0`
  - ou seja, nada a fazer
- Geral: Temos `n > 0`
  - imprime recursivamente o começo da lista
  - e imprime o último elemento

Vamos depurar!

Exercícios

1. Faça uma função recursiva que calcula a soma dos elementos de uma lista.
2. Faça uma função recursiva que encontra o máximo de uma lista.
3. Faça uma função recursiva que busca um elemento em uma lista.
4. Faça uma função recursiva que recebe uma lista e devolve uma copia da lista invertida.
5. Faça uma função recursiva que checa se duas listas dadas são iguais.

Busca Binária Iterativa

Relembrando:

```python
1 def busca_binaria(l, x):
2     e = 0
3     d = len(l) - 1
4     while e <= d:
5         m = (e + d) // 2
6         if l[m] == x:
7             return m
8         elif l[m] < x:
9             e = m + 1
10        else:
11            d = m - 1
12    return -1
```

Se `x` estiver em `l`, está entre `e` e `d`

- Essa propriedade é mantida no `while` da linha 4

Busca Binária Recursiva

```python
def busca_binaria_rec(l, x, e, d):
    if e > d:
        return -1
    m = (e + d) // 2
    if l[m] == x:
        return m
    if l[m] < x:
        return busca_binaria_rec(l, x, m + 1, d)
    else:
        return busca_binaria_rec(l, x, e, m - 1)

def busca_binaria(l, x):
    return busca_binaria_rec(l, x, 0, len(l) - 1)
```

A recursão é em `e` e `d` e `d - e` sempre diminuí:

- Base: Se `d - e < 0`, a faixa que estamos buscando é vazia
- Base: Se `l[m] == x`, encontramos o elemento `x`
- Geral: Se `x` está em `l` entre `e` e `d` e `l[m] < x`, então `x` está em `l` entre `m + 1` e `d`
- Geral: Se `x` está em `l` entre `e` e `d` e `l[m] > x`, então `x` está em `l` entre `e` e `m - 1`

Enumerando

Recursão pode ser útil também para enumerar:

- Gerar todas as possíveis senhas de acordo com uma regra
- Gerar todas as permutações de elementos
- Gerar todos os subconjuntos de um conjunto
- Etc...

Para tanto, precisamos construir a informação passo a passo

- Armazenando ela em alguma estrutura de dados

Exemplo: Senhas numéricas

Como gerar todas as senhas numéricas de n dígitos?

- Base: Se $n = 0$, não temos nada a fazer
- Geral: Para cada $i \in \{0, \ldots, 9\}$, gere todas as senhas de $n - 1$ dígitos colocando i na frente

```python
def senhas_rec(mem, n):
    if len(mem) == n:  # temos n digitos
        print(''.join(mem), end=' ')
    else:
        for i in range(10):
            mem.append(str(i))  # coloca o dígito na memória
            senhas_rec(l, n)
            mem.pop()  # retira o dígito da memória

def senhas(n):
    mem = []
    senhas_rec(mem, n)
```

Vamos depurar!

Exercícios

1. Faça uma função recursiva que, dado um inteiro positivo n, imprime todas as permutações de elementos de 1 a n.
2. Faça uma função recursiva que dada uma string s, diz se s é palíndromo ou não. Uma string s é um palíndromo se a reversão de s é a própria s (ex: `'ana'` e `'arara'`).
3. Faça uma função recursiva que verifica se duas listas são iguais. Porém, os elementos podem também ser listas e você não deve usar a comparação de listas pronta do Python.
4. Faça uma função recursiva que, dada uma lista l e um inteiro positivo r, imprime todas as combinações de r elementos de l.