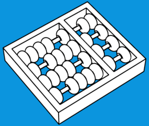
Recursão

MC102 - Algoritmos e Programação de Computadores

21

05/24



Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

"Para entender recursão, você primeiro tem que entender recursão."

Anônimo.

Dúvidas da aula anterior

Dúvidas selecionadas

- Não entendi direito por que a busca binária é mais rápida do que a sequencial.
- Não entendi o porque de 'devolva -1' na busca binária.
- Se no algoritmo de busca binária o elemento procurado estiver na última posição teremos "e=d"?
- Para os casos em que o vetor não está ordenado, vale mais a pena fazer uma busca linear diretamente ou ordenar os valores e fazer a busca binária?
- É possível fazer uma busca binária em uma lista de strings? e se a lista possuir mais de um tipo de objeto?
- Então a otimização faz parte da solução em si, não da maneira como você coda?
- Existe alguma função que trabalhe com busca em listas desordenadas?
- Como aspectos de cada computador como quantidade de threads e cores, velocidade da RAM e placa de vídeo afetam a velocidade dos algoritmos que nós vimos?
- Como eu sei exatamente qual método de busca eu devo utilizar?
- Tem um algoritmo de busca ainda mais eficiente do que o binário? Imagino que a recomendação de filmes nos streamings, por exemplo, utilize algum algoritmo de busca, porém levando em consideração coisas como a similaridade de perfis, o gênero, atores etc. Como funcionaria isso?
- Como aprender a otimizar os códigos de forma mais eficiente e entender qual seria melhor? visto que no caso dos algorítimos vistos na aula anterior são bem diferentes e que com otimização não necessariamente significa que ele seja melhor.
- Tem outros métodos de busca além do binário, seguindo uma lógica similar? Tipo um com base 3?
- Existe algum algoritmo de ordenação/busca baseado em computação quântica?

Introdução

Motivação

- É necessário solucionar um problema para uma entrada grande.
- O problema pode ser reduzido para um caso menor.
- Solucionar o caso menor e a partir dele obter a solução do maior.

O que é recursão?

TÉCNICA em que pelo menos um dos passos de um certo procedimento envolve a **REPETIÇÃO** de todo o procedimento.

TÉCNICA em que um subprograma (**FUNÇÃO**) chama a si mesmo.

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**.

Algoritmos recursivos

Se: a instância é pequena, **então:**

- Resolva o problema diretamente.

Senão:

- Reduza-a a uma (ou várias) instância(s) menor(es) do mesmo problema.
- Aplique o algoritmo à(s) instância(s) menor(es).
- Volte à instância original e resolva-a.

Fim da recursão

IMPORTANTE: toda função recursiva deve **ENCERRAR** a recursão!

Para encerrar a recursão se realiza um teste (**CONDIÇÃO DE PARADA**).

Sem a condição de parada, a recursão se repete eternamente (até dar um erro de memória (**stack**) cheia).

Forma geral

Algoritmo: FUNÇÃO(parâmetros)

```
se condição de parada
    resolver caso base e/ou finalizar
senão
    FUNÇÃO(parâmetros menores)        ▷ chamado recursivo
    solucionar o problema para parâmetros
```

Fatorial

Calculando Fatorial

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:

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

```
fatorial        n:   4
   return n * 6
      return n * 2
         return n * 1
            return n * 1
               return 1
```

Progressão aritmética

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:

```
def termo(a_1, r, n):
    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?

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

Vamos depurar!

Fibonacci

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,

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

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 + 2
      return 1 + 1
         return 1
            return 1
```

Euclides

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 $a \,\mathbf{mod}\, b$:

- $a = k \cdot b + r$ e, portanto, $a \,\mathbf{mod}\, b = r = a - kb$.
- $a = q_a \cdot x$ e $b = q_b \cdot x$.
- $a \,\mathbf{mod}\, b = a - k \cdot b = q_a \cdot x - k \cdot q_b \cdot x = (q_a - k \cdot q_b) \cdot x$.

Assim:

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

Exemplo: Algoritmo de Euclides

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

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 $a \,\textbf{mod}\, b < b$.
  - 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

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

```
mdc        a:  40    b:  70
  return 10
    return 10
      return 10
        return 10
          return 10
```

Considerações

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 não 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)**...

Recursão com listas

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.
  - Imprime o último elemento.

Exercícios

Sobre os algoritmos

Vamos fazer alguns exercícios?

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 na representação decimal.
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).

Exercícios

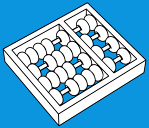
1. Faça uma função recursiva que calcula a soma dos elementos de uma lista.
2. Faça uma função recursiva que encontra o máximo de uma lista.
3. Faça uma função recursiva que busca um elemento em uma lista não ordenada.
4. Faça uma função recursiva que recebe uma lista e devolve uma copia da lista invertida.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

Recursão

MC102 - Algoritmos e Programação de Computadores

21

05/24



Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

UNICAMP