14 Aula 14: 26/SET/2019

14.1 Aulas passadas

Problema: Labirinto

Esse problema serve para ilustrar que recursão na resolução de problemas mais elaborados. No caso, essa Pode ser uma alternativa para `Percolation.open()` do EP07.

Problema: `maximoR()` foi um exercício de recursão que também serviu para mostrar que podemos fazer uma função `maximo()` que é *casca* para uma que recebe mais parâmetros. Essa casca será possivelmente útilo no EP09.

Problema: máximo divisor comum:

- `mdc()`: algoritmo ingênuo e lento
- `euclidesR()`: algoritmo de Euclides, muito eficiente e recursivo
- `euclidesI()`: algoritmo de Euclides, versão iterativa.

O algoritmo de Euclides é em essência recursivo, mas admite um implementação iterativa muito simples.

Vimos as **três leis de recursão**, um algoritmo recursivo deve:

- **Resolver**: resolver se for um **caso base**
- **Reduzir**: simplifica o problema fazendo alguma chamada recursiva de um subproblema mais **próximo de um caso base**
- **Combinar**: obter um solução do problema combinando as soluções de subproblemas e retorná-la.

Em aulas anteriores também vimos:

- Hanoi: problema de motivação
- Fatorial: usado para ilustrar o mecanismo
- Hanoi de volta: para ilustrar análise de consumo de tempo
- Fibonacci: para ilustrar custo de resolver o mesmo subproblema várias vezes e comparar custo com solução iterativa.

14.2 Aula de hoje

Vamos trabalhar a técnica de memorização (*memoization*)e o problema de busca em listas não ordenadas e ordenadas. Busca binária é usada em busca em listas ordenadas é um algoritmo muito eficiente. O algoritmo é essencialmente recursivo, mas podemos desenvolver programas recursivos e iterativos que implementam a ideia.

14.3 Problema 1: A culpa não é da recursão!

Slides: `.../slides/slides_fibonacci.pdf`

Programas: `fibonacciI.py`, `fibonacciR.py`, `fibonacciR-s.py` e `fibonacciRM.py`

Fazer análise experimental de `fibonacciI()` e `fibonacciR()`:

```
../py> time python fibonacciI.py 4
fibonacci(4) = 3
```

```
real        0m0.040s
user        0m0.028s
sys         0m0.008s

.../py> python fibonacciR.py 4
fibonacci(4) = 3

.../py python fibonacciR-s.py 4
fibonacciR(4)
  fibonacciR(3)
    fibonacciR(2)
      fibonacciR(1)
      fibonacciR(0)
    fibonacciR(1)
  fibonacciR(2)
    fibonacciR(1)
    fibonacciR(0)
fibonacci(4) = 3
```

Observe que o alto consumo de tempo de `fibonacciR()` não tem nada a ver com a recursão, mas sim com o fato do mesmo subproblema estar sendo resolvido várias vezes.

Veja **árvore da recursão** nos slides.

Contar vagamente o número de operações de cada função.

Arquivos relacionados:

- `fibonacciI.py`: iterativo
- `fibonacciR.py`: recursivo
- `fibonacciRM.py`: recursivo com memorização
- `fibonacciR-s.py`: ilustra ordem das chamadas recursivas e a chamada em cada nível da recursão de `fibinacciR()`
- `fibonacciRM-s.py`: ilustra ordem das chamadas recursivas e a chamada em cada nível da recursão `fibonacciRM.py`

```python
#-----------------------------------------------
def fibonacciI(n):
    '''(int) -> int

    Recebe um inteiro não negativos n e retorna o
    n-ésimo número de fibonacci.
    '''
    if n == 0: return 0
    if n == 1: return 1
    anterior = 0
    atual    = 1
    for i in range(1,n):
        proximo = atual + anterior
        anterior = atual
        atual = proximo
    return atual
```

```
#-----------------------------------------------
def fibonacciR(n):
    '''(int) -> int

    Recebe um inteiro não negativos n e retorna o
    n-ésimo número de fibonacci.
    '''
    if n == 0: return 0
    if n == 1: return 1
    return fibonacciR(n-1) + fibonacciR(n-2)

#-----------------------------------------------
def finonacci(n):
    '''(int) -> int

    Recebe um inteiro não negativos n e retorna o
    n-ésimo número de fibonacci.
    '''
    fib = (n+1)*[0]
    return fibonacciRM(n, fib)

#-----------------------------------------------
def fibonacciRM(n, fib):
    '''(int) -> int

    Recebe um inteiro não negativos n e retorna o
    n-ésimo número de fibonacci.
    '''
    if n == 0: return 0
    if n == 1: return 1
    if fib[n] > 0: return fib[n]
    fib[n-1] = fibonacciM(n-1, fib)
    # fib[n-2] = fibonacciM(n-2) # já foi calculado
    fib[n] = fib[n-1] + fib[n-2]
    return fib[n]
```

14.4 Busca em listas

Problema 2

Escreva uma função `busca_sequencial` que recebe uma lista de *valores comparáveis* e um valor `x`, e retorna o índice da posição onde `x` ocorre na lista, ou `None` em caso contrário.

Esse problema ilustra análise de consumo de tempo de algoritmo ao contarmos o número de comparações realizadas. Suloções podem envolver:

- `return` no meio do loop,
- apenas um `return` no final com indicador de passagem e
- uso de sentinela.

```
def busca_sequencial_1(elemento, lista):
    ''' (obj, list) -> int or None

    Procura por elemento na lista.
    Caso existir, retorna o índice do elemento na lista.
    Em caso contrário, retorna None.
    '''
    posicao = None
    i = 0
    n = len(lista)
    while i < n:
        if elemento == lista[i]:
            posicao = i
        i += 1

    return posicao

## usa indicador de passagem
def busca_sequencial_2(elemento, lista):
    ''' (obj, list) -> int or None

    Procura por elemento na lista.
    Caso existir, retorna o índice do elemento na lista.
    Em caso contrário, retorna None.
    '''
    posicao = None
    i = 0
    n = len(lista)
    while i < n and posicao == None:
        if elemento == lista[i]:
            posicao = i
        i += 1

    return posicao
```

```python
## return quando acha
def busca_sequencial_3(elemento, lista):
    ''' (obj, list) -> int or None
    Procura por elemento na lista.
    Caso existir, retorna o índice do elemento na lista.
    Em caso contrário, retorna None.
    '''
    n = len(lista)
    for i in range(n):
        if elemento == lista[i]:
            return i
    return None # supérfluo,  explicita as intenções

## com sentinela
def busca_sequencial_4(elemento, lista):
    ''' (obj, list) -> int or None

    Procura por elemento na lista.
    Caso existir, retorna o índice do elemento na lista.
    Em caso contrário, retorna None.
    '''
    lista.append(elemento)

    i = 0
    while elemento != lista[i]: i += 1

    lista.pop() # corrige a lista

    if i < len(lista): return i

    return None # supérfluo,  explicita as intenções
```

14.5 Busca Binária

Problema 3

Escreva uma função `busca_binária()` que recebe uma lista de objetos **ordenada** e um objeto `x`, e retorna o índice da posição onde `x` ocorre na lista, ou `None` em caso contrário.

```
#=============================================================================
def busca_binI(elemento, lista):
    ''' (obj, list) -> int or None

    Procura por elemento na lista.
    Caso existir, retorna o índice do elemento na lista.
    Em caso contrário, retorna None.
    '''
    e = 0
    d = len(lista)
    while e < d:
        meio = (e + d)//2
        elem = lista[meio]
        if elem == elemento: return meio
        if elem < elemento: e = meio+1
        else: d = meio
    return None # supérfluo

#=============================================================================
def busca(elemento, lista):
    ''' (obj, list) -> int or None

    Procura por elemento na lista.
    Caso existir, retorna o índice do elemento na lista.
    Em caso contrário, retorna None.
    '''
    return busca_binR(elemento, lista, 0, len(lista))

#=============================================================================
def busca_binR(elemento, lista, e, d):
    ''' (obj, list, int, int) -> int or None '''
    if d <= e: return None
    meio = (e + d)//2
    elem = lista[meio]
    if elem == elemento: return meio
    if elem < elemento: return busca_binR(elemento, lista, meio+1, d)
    return busca_binRED(elemento, lista, e, meio)
```