18 Aula 18: 10/OUT/2019

18.1 Aula passada

- `intercale()`
- mergesort: recursão, iteração, consumo de tempo `n lg n`
- algoritmos por **divisão-e-conquista**

18.2 Hoje

Motivação Um problema do mergesort é o consumo de espaço extra de $\mathtt{O}(\mathtt{n})$. Será que podemos ter um algoritmo de ordenação com consumo de tempo $\mathtt{O}(\mathtt{n} \lg \mathtt{n})$ e memória $\mathtt{O}(\mathtt{1})$?

- `separe()`
- `quicksort()`
- algoritmo por **divisão-e-conquista**

18.3 Separação

18.3.1 Problema

Rearranjar uma dada lista `v[e:d]` e retorna um índice `m`, $\texttt{e} \leq \texttt{m} < \texttt{d}$, tais que $\texttt{v}[\texttt{e} : \texttt{m}] \leq \texttt{v}[\texttt{m}] < \texttt{v}[\texttt{m}{+}\texttt{1} : \texttt{d}]$.

```
Entra:

     e                                                d
   +----+----+----+----+----+----+----+----+----+----+
v  | 99 | 33 | 55 | 77 | 11 | 22 | 88 | 66 | 33 | 44 |
   +----+----+----+----+----+----+----+----+----+----+


Sai:

     e                   m                            d
   +----+----+----+----+----+----+----+----+----+----+
v  | 33 | 11 | 22 | 33 | 44 | 55 | 99 | 66 | 77 | 88 |
   +----+----+----+----+----+----+----+----+----+----+
```

18.3.2 Solução CLRS

```python
#-------------------------------------------------
def separe(e, d, v):
    ''' (int,int,list) -> None

    Recebe uma lista v[e:d] e rearranja seu elementos e
    retorna um índice q tais que

        v[e:m] <= v[m] < v[m+1:d]

    Consumo de tempo é O(n) onde n = p-r
    '''
    x = v[d-1] # pivo
    i = p-1
    for j in range(e,d): #A#
        if v[j] <= x:
            i += 1
            v[i],v[j] = v[j],v[i]
    return i
```

18.3.3 Correção

Em `#A#` vale que $\texttt{v}[\texttt{e} : \texttt{i}] \leq \texttt{x} < \texttt{v}[\texttt{i} + \texttt{1} : \texttt{j}]$

18.3.4 Solução PSADS

```
def separeR(e,d,v):
   x = v[e] # pivo
   e = p+1
   d = r-1
   while e <= d:
       while e <= d and v[e] <= x:
           e += 1
       while d >= e and v[d] >  x:
           d -= 1
       # v[e] >  x and v[e  :e-1] <= x
       # v[d] <= x and v[d+1:  d] >  x
       if e < d:
           v[e],v[d] = v[d],v[e]
           e += 1
           d -= 1

   v[e],v[d] = v[d],v[e]

   return d
```

18.3.5 Consumo de tempo

O consumo de tempo da função `separe()` é proporcional a `n := r-p`.

O consumo de tempo da função `separe()` é `O(n)`.

18.4 Quicksort

```
def quick_sort(e,d,v):
    '''(int,int,list) -> None

    Recebe uma lista v[e:d] e rearranja seu elementos
    de maneira que fique crescente.
    '''
    if p < r-1:
        q = separe(e,d,v)
        quick_sort(e,m,v)
        quick_sort(m,d,v)
```

18.4.1 Correção

A função está correta?

A correção da função, que se apoia na correção da função `separe()` pode ser demonstrada por indução em `n := r-p`.

18.4.2 Consumo de tempo

Desenhar "árvore da recursão" e verificar que no **pior caso** há cerca de `n` níveis e o consumo de tempo desses níveis é `n`, `n-1`,...,`1`.

Desenhar "árvore da recursão" e verificar que no **melhor caso** há cerca de $\log_2 \texttt{n}$ níveis e o consumo de tempo de cada um desses níveis é aproximadamente `n`.

18.4.3 Conclusões

O consumo da função `quick_sort()` no **pior caso** é proporcional a $\texttt{n}^2$.

O consumo da função `quick_sort()` no **pior caso** é $\texttt{O}(\texttt{n}^2)$.

O consumo da função `quick_sort()` no **melhor caso** é proporcional a $\texttt{n} \log \texttt{n}$.

O consumo da função `quick_sort()` no **melhor caso** é $\texttt{O}(\texttt{n} \log \texttt{n})$.

18.5 Divisão e conquista

Algoritmos por divisão-e-conquista, tipicamente, têm três passos em cada nível da recursão:

- **dividir**: o problema é dividido em subproblemas de tamanho menor
- **conquistar**: os subproblema são resolvidos recursivamente e subproblemas "pequenos" são resolvido diretamente.
- **combinar**: as soluções dos subproblemas são combinadas para obter uma solução do problema original

18.6 Versão iterativa

```
def quick_sort(v):
    n = len(v)
    p = 0
    r = n
    pilha = Stack()
    pilha.push((e,d))
    while pilha.isEmpty():
        e, r = pilha.pop()
        if p < r-1: # r-p > 1 ou n > 1
            q = separe(e, d, v)
            # segmento inicial
            pilha.push((e, q))
            # segmente finaç
            pilha.push((q+1, d))
```

18.7 k-ésimo menor elemento

Problema. encontrar o `k`-ésimo menor elemento de uma lista `v[0:n]` supondo `k` <= `n`.

Solução inspirada em `separe()`. Ao final o `k`-ésimo menor elemento está na posição `v[]`.

```
def k_esimo(k, e, d, v):
    q = separe(e, d, v)
    if q == k-1: return
    if q >= k:
        return k_esimo(k, e, m, v)
    return k_esimo(k, q+1, d, v)
```