Lista de exercícios 2

MAC0122 - Princípios de
Desenvolvimento de Algoritmos

Novembro 2017

1 Recursão

Uma lista de exercícios foi sugerida no PACA pelo professor Coelho. Você pode encontra-la aqui: https://paca.ime.usp.br/mod/forum/discuss.php?d=36536

Sugerimos que você faça pelo menos esses exercícios (faça todos se possível):

- Exercício 1
- Exercício 2 e 3 (é interessante comparar as implementações, também). Adicionalmente, quantas chamadas recursivas cada uma das implementações faz?
- Exercício 4
- Exercício 7
- Exercício 10
- Exercício 11
- Exercício 12
- Exercício 15
- Exercício 17. Você consegue pensar em uma implementação eficiente (algo como $O(logn)$ ao invés de $O(n)$)? *Dica (caso precise): pesquise sobre exponenciação rápida na internet.*

2 Intercalação e mergesort

Uma lista de exercícios foi sugerida no PACA pelo professor Coelho. Você pode encontra-la aqui: https://paca.ime.usp.br/mod/forum/discuss.php?d=36537

Sugerimos que você faça pelo menos esses exercícios (faça todos se possível):

- Exercício 1
- Exercício 2
- Exercício 5
- Exercício 7
- Exercício 9
- Exercício 11
- Exercício 12
- Exercício 13
- Exercício 18
- Exercício 22

3 Separação e quicksort

Uma lista de exercícios foi sugerida no PACA pelo professor Coelho. Você pode encontra-la aqui: https://paca.ime.usp.br/mod/forum/discuss.php?d=36545. Abaixo você encontra alguns exercícios interessantes, além dos já sugeridos pelo professor Coelho.

Veja mais detalhes sobre quicksort na página do professor Paulo Feofiloff (clique aqui).

3.1 Separação

Uma possível implementação para a função de separação pode ser encontrada abaixo[1]:

```
def separa (v, p, r):
  '''(list, int, int) -> int
  A  funo  rearranja o vetor v[p..r] e devolve um elemento
  j do conjunto p..r tal que v[p..j-1] <= v[j] < v[j+1..r]
  '''
  c = v[p]
  i = p + 1
  j = r

  while True:
    while i <= r && v[i] <= c:
      i = i + 1
    while c < v[j]:
      j = j - 1
    if i >= j:
      break
    t = v[i]
    v[i] = v[j]
    v[j] = t
    i = i + 1
    j = j - 1

  v[p] = v[j]
  v[j] = c
  return j
```

3.1.1 Desempenho

Na função *separa*() acima, as variaveis i e j funcionam como ponteiros para as entradas do vetor v. Note que enquanto a primeira variável começa em $p+1$, a segunda começa em r e o algoritmo para quando os valores de i e j

[1]Essa versão foi adaptada do site do professor Paulo Feofiloff para Python. Veja referências

se cruzam.

Por essa razão, a função executa uma quantidade proporcional a $r - (p + 1)$ operações. Note que $r - (p + 1)$ é limitado ao tamanho do vetor. Portanto, o consumo de tempo da função é proporcional a n, ou seja, $O(n)$.

3.1.2 Exercícios

1. Mostre o resultado da operação separa (v, 0, 15), sendo v o vetor
   33 22 55 33 44 22 99 66 55 11 88 77 33 88 66 66

2. Qual o resultado da função separa quando os elementos de v[p..r] são todos iguais? E quando v[p..r] é crescente? E quando v[p..r] é decrescente? E quando cada elemento de v[p..r] tem um de dois valores possíveis?

3. Escreva uma versão recursiva da função separa.

4. A função *separa* produz um rearranjo estável do vetor, ou seja, preserva a ordem relativa de elementos de mesmo valor?

5. *Desafio.* Escreva uma solução do problema da separação que devolva um índice j tal que (r-p)/10 j-p 9(r-p)/10.

3.2 Quicksort

Uma possível implementação para a função de *quicksort* pode ser encontrada abaixo[2]:

```
def quicksort(v, p, r):
  '''(list, int, int) -> int
   O algoritmo resolve recursivamente usando a estratgia da
      diviso e conquista
   Se p >= r (essa  a base recurso) no   preciso fazer nada.
      Se p < r, o cdigo reduz a instncia v[p..r] do problema
      ao par de instncias v[p..j-1] e v[j+1..r]. Como p  j
      r, essas duas instncias so estritamente menores que a
      instncia original. Assim, por hiptese de induo,
      v[p..j-1] estar em ordem crescente no fim da linha 4 e
```

[2]Essa versão também foi adaptada do site do professor Paulo Feofiloff

```
    v[j+1..r] estar em ordem crescente no fim da linha 5.
    Como v[p..j-1]  v[j] < v[j+1..r] graas  funo separa,
    o vetor todo estar em ordem crescente no fim da linha
    5, como promete a documentao da funo.
'''
if p < r:
    j = separa(v, p, r)
    quicksort (v, p, j - 1)
    quicksort (v, j + 1, r)
```

O algoritmo usa a estratégia da divisão e conquista e é bem semelhante ao *mergesort*.

3.2.1 Desempenho

O desempenho do *quicksort* no caso médio é $O(nlogn)$. No pior caso, entretanto, ele não é melhor que os algoritmos elementares de ordenação e sua complexidade é $O(n^2)$. Veja a página do professor Paulo Feofiloff para mais informações.

3.2.2 Exercícios

1. Simule a execução do *quicksort* para o vetor v definido como
   33 22 55 33 44 22 99 66
2. Escreva uma versão iterativa do *quicksort*.
3. Escreva um programa que teste, experimentalmente, a correção de sua implementação do algoritmo Quicksort. Use permutações aleatórias de 1..n para os testes. Compare o resultado da ordenação com 1..n.
4. Em que cenários o *quicksort* performa mal, isto é, qual é o pior caso?

4 Método de Monte Carlo

1. Vimos em aula o paradoxo do aniversário. Faça os experimentos e mostre quantas pessoas são necessários para que a probabilidade de existir um par de pessoas com o mesmo aniversário seja de 50%. Compute ainda para as probabilidades 75% e 90%.

2. Compute experimentalmente o valor aproximado de π. *Dica: pense em gerar pontos aleatórios e verificar se eles estão contidos em um círculo pré-definido ou não.*

Na página https://introcs.cs.princeton.edu/java/21function/, resolva os seguintes *Web Exercises*:

1. (4) SAT scores.
2. (5) Voting machines.

Referências

[1] Paulo Feofiloff. *Projeto de Algoritmos (em C)*. https://www.ime.usp.br/ pf/algoritmos/, 2017.