Alguns algoritmos elementares de ordenação

Armando Matos

Departamento de Ciência de Computadores
Universidade de Porto

2008

Introdução

2 métodos elementares de ordenação...

2 métodos elementares de ordenação de listas:

- Ordenação por selecção do mínimo
- "Bubble sort" (método da bolha)
- "Quick sort", já descrito nas aulas teóricas

Notação e hipóteses

- `a`: a lista a ordenar
- `n` = `len(a)`, número de elementos da lista `a`
- índices de `a`: de 0 a `n-1`
- $\log(x)$: logaritmo de x na base 2

Sobre a eficiência dos métodos de ordenação...

"Ordenação por selecção do mínimo" e "Bubble sort": não são algoritmos muito eficientes.
Consideramos o tempo de execução no pior caso
$t(n) = \max\{\text{tempo de execução para listas com } n \text{ elementos}\}$

- Selecção do mínimo e "bubbelsort": $\exists k > 0 : \ t(n) \geq kn^2$.
- Os mais rápidos algoritmos de ordenação baseados em comparações (como o "heapsort" e o "mergesort"): $\exists k > 0 : \ t(n) \leq kn \log n$.
- E o "quick sort"? Rápido em média, lento no pior caso.

Comparando n^2 com $n \log n$

Exemplo

$n = 10\,000$, cada comparação corresponde a 1μseg.

$n^2 = 10^8$ corresponde a $10^8 \times 10^{-6} = 100$ s $\approx$ 2 minutos!

$n \log n$ corresponde a $10^4 \log(10^4) \times 10^{-6} \approx$ 0.13 segundos!

Ordenação por selecção do mínimo

Ordenação por selecção do mínimo

Descrição sucinta:

para $i = 0, 1, \ldots, n-2$:
troca $a[i]$ com o menor elemento em $a[i \ldots n-1]$

Ordenação por selecção do mínimo: exemplo

a=[8,2,1,5,3,6]

Sublinhado: elemento de índice i,
Azul: elemento já na posição definitiva
Vermelho: mínimo entre as posições i e $n-1$

i=0, [8,2,1,5,3,6] → [1,2,8,5,3,6]
i=1, [1,2,8,5,3,6] → [1,2,8,5,3,6]
i=2, [1,2,8,5,3,6] → [1,2,3,5,8,6]
i=3, [1,2,3,5,8,6] → [1,2,3,5,8,6]
i=4, [1,2,3,5,8,6] → [1,2,3,5,6,8]
A lista ordenada é necessariamente [1,2,3,5,6,8].

Ordenação por selecção do mínimo: programa

```
def ordena(a):
    i=0
    n=len(a)
    while i <= n-2:
        # Ja' ordenados para i em {0,1,...,i-1}
        k=ind_min(a,i,n-1)
        a[i],a[k] = a[k],a[i]
        i=i+1
    return a
```

Ordenação por selecção do mínimo: função auxiliar

`ind_min(a,x,y)` retorna o índice do (do primeiro) valor mínimo em

$$a[x], a[x+1], \ldots a[y]$$

```
def ind_min(a,x,y):
    imin=x
    i=x+1
    while i<=y:
        if a[i] < a[imin]:
            imin=i
        i=i+1
    return imin
```

Eficiência dos algoritmos de ordenação I

Algoritmos baseados em comparações entre elementos da lista,
Vamos usar (tal como nos algoritmos de pesquisa) como medida de eficiência no tempo de execução

$c(n)$: número de comparações entre elementos da lista

ou seja, comparações da forma `a[i]` $\underline{\text{op}}$ `a[j]` onde o operador $\underline{\text{op}}$ pode ser "$<$", "$\leq$", "$=$"...
Para certos algoritmos esta medida pode não corresponder ao tempo de execução...

Eficiência do método da selecção do mínimo

- `i=0`: `ind_min(a,0,n-1)` faz $n-1$ comparações
- `i=1`: `ind_min(a,1,n-1)` faz $n-2$ comparações
- ...
- `i=n-2`: `ind_min(a,n-2,n-1)` faz 1 comparação

Total:

$$c(n) = 1 + 2 + \ldots + (n-1) = n(n-1)/2$$

“Bubblesort”

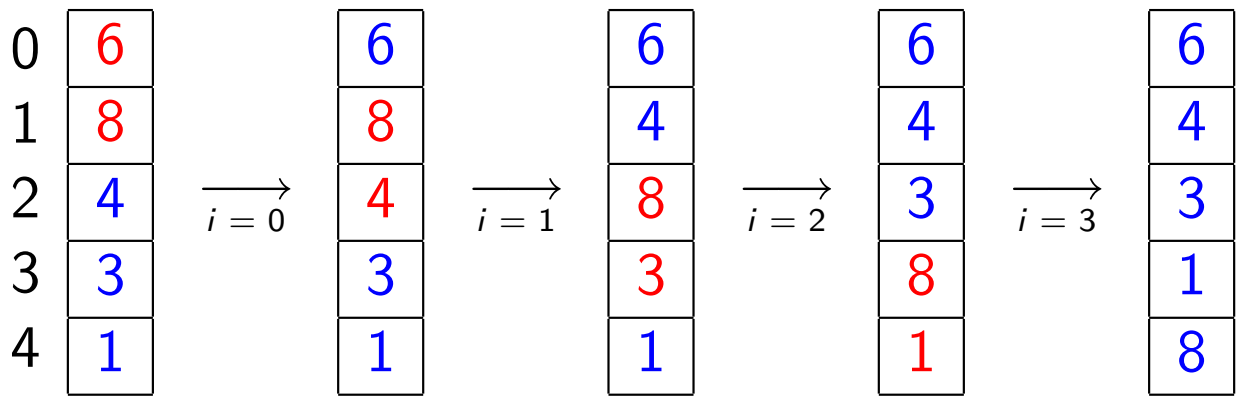
“Bubblesort”, ideia

- Efectuar um número suficiente de vezes:
  - (“varrimento”): comparar cada elemento da lista com o seguinte e trocar, se necessário.

“Bubblesort”, um varrimento

Para `i`=0, 1,...,n-2:

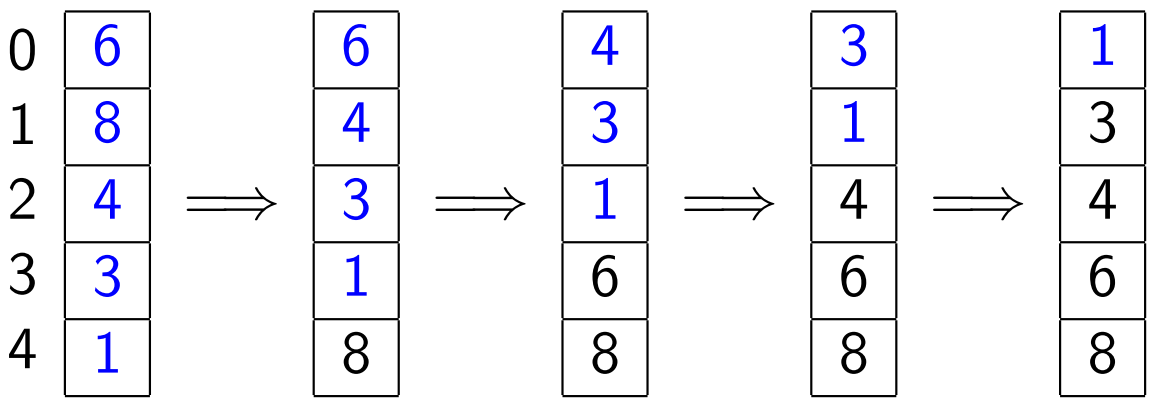
se `a[i]>a[i+1]`: `a[i]` $\leftrightarrow$ `a[i]`.



A lista ainda não fica ordenada... mas, ao fim de um varrimento, o maior elemento (8, neste exemplo) vai para a última posição; logo basta fazer $n-1$ varrimentos.

“Bubblesort”, os varrimentos necessários

Os 4 varrimentos para `n=5`



A preto: elementos na posição final.

“Bublesort”: programa

```
# metodo de ordenacao "bubblesort"
def ordena(a):
    n=len(a)
    for v in range(1,n):            # n-1 vezes
        for i in range(0,n-1):      # 0 a n-2
            if a[i] > a[i+1]:
                (a[i],a[i+1]) = (a[i+1],a[i])
    return a
```

Eficiência do método "bubblesort"

Em cada um dos $n-1$ varrimentos, fazem-se $n-1$ comparações (`i`: 0, 1, ..., `n-2`). Logo

$$c(n) = (n-1)^2$$

Melhorando (um pouco) a eficiência do "bubblesort"

Exercícios: a partir destas ideias implemente programas

- `i` não necessita de ir sempre até `n-2`: após o primeiro varrimento, o maior elemento fica na última posição e no segundo varrrimento as comparações podem ir apenas até `i=n-2`, etc.
- O número de varrimentos pode ser menor se, em cada varrimento, se detectar se houve alguma troca ou não; se não houve, pode-se terminar (exercício de uma folha TP).
- Os varrimentos podem ser efectuados alternadamente da esquerda para a direita e da direita para a esquerda.

Nota. “Quick sort”: programa

```
# metodo de ordenacao "quick sort"

def qs(a):
    n=len(a)
    if n==0:
        return a
    esq = [x   for x in a[1:n]   if x<=a[0]]
    dir = [x   for x in a[1:n]   if x> a[0]]
    return qs(esq) + [a[0]] + qs(dir)
```

Notas sobre a eficiência

Eficiência - um esquema geral algorítmico

Enquanto existir algum `i<j` com `a[i]` $>$ `a[j]`:

selecciona-se um desses pares `(i,j)`

$a[i], a[j] = a[j], a[i]$

Não é um algoritmo (no sentido clássico) porque a escolha de (i,j) não é determinística.

Correcção parcial

- O esquema anterior, se terminar, dá resposta correcta? Sim!
  Prova: (trivial) se o "algoritmo" terminar (condição do "enquanto"), não existe qualquer par `(i,j)` com `i<j` e `a[i]` $>$ `a[j]`. Logo `a` está ordenado.

Terminação I

- O esquema anterior termina sempre?
  Seja $d(a)$ o número de pares "desordenados" na lista a:

$$d(a) = \#\{(i,j) \ : \ (i < j) \wedge (a[i] > a[j])\}$$

(a prova continua. . .)

Terminação II

- Temos sempre (Exercício: prove!) $0 \leq d(a) \leq n(n-1)/2$
- Com cada troca no esquema algorítmico $d(a)$ diminui estritamente (Exercício: prove!).
- Logo o esquema algorítmico termina.

(fim da prova)

Eficiência e troca de elementos consecutivos

- Com cada troca de elementos consecutivos, $d(a)$ diminui de 1. (Exercício: prove!).
- Corolário: qualquer algoritmo que modifique a lista fazendo apenas trocas de elementos consecutivos (como o "bubblesort") tem um tempo de execução $\geq kn^2$ para uma constante $k > 0$. (Exercício:prove!).

Resumo

Resumo I

- Apresentamos 3 métodos de ordenação: selecção do mínimo, "bubblesort" e "quick sort".
- Existem muitos outros métodos de ordenação
- Existe uma vasta teoria matemática (que tem a ver, entre outros assuntos, com a teoria das permutações) sobre os algoritmos de ordenação. Ver Knuth, *The Art of Computer Programming*, vol. III.

Resumo II

- O método selecção do mínimo e "bubblesort" têm um tempo de execução (em média e no pior caso) essencialmente quadrático no tamanho das listas.
- Os métodos de ordenação assintoticamente mais rápidos têm um tempo de execução essencialmente proporcional a $n \log n$, mesmo no pior caso.
- O método "quick sort" tem um tempo de execução
  - Em média: proporcional a $n \log n$
  - No pior caso: proporcional a n^2

fim