Algoritmos de Ordenação - Counting e Radix

Profª. Rose Yuri Shimizu

Roteiro

Roteiro

Counting Sort

Em breve ;).

Roteiro

1. Algoritmos de Ordenação
   - Counting Sort
   - Radix Sort

Radix Sort

Ordenação: compara-se as chaves/dados

- Detalhes do processamento de comparação são "abstraídos"
- Como o processador faz as comparações?
  - Como os dados são representados por bits, as comparações ocorrem bit a bit
  - Todos os bits dos dados são processados a cada iteração

Radix Sort

Ordenação: compara-se as chaves/dados

- Detalhes do processamento de comparação são "abstraídos"
- Como o processador faz as comparações?
  - Como os dados são representados por bits, as comparações ocorrem bit a bit
  - Todos os bits dos dados são processados a cada iteração

Radix Sort

Como os dados são compostos por dígitos

- Ideia: diminuir as comparações de cada etapa
  - ao invés de comparar a chave inteira em toda iteração ou recursão
  - compara-se somente parte dela
  - ordenando parcialmente
- Exemplo:
  - ao procurar/posicionar uma palavra em um dicionário
  - cada letra que forma a palavra
  - contribui para localizar a página exata da palavra
  - cada letra restringe as possibilidades de posições

Radix Sort

170 045 075 090 802 024 002 066

Radix Sort

170 045 075 090 802 024 002 066

170 090 802 002 024 045 075 066

Radix Sort

170 090 802 002 024 045 075 066

Radix Sort

170 090 802 002 024 045 075 066

802 002 024 045 066 170 075 090

Radix Sort

802 002 024 045 066 170 075 090

Radix Sort

802 002 024 045 066 170 075 090

002 024 045 066 075 090 170 802

Radix Sort

170 045 075 090 802 024 002 066

002 024 045 066 075 090 170 802

- Ordena pela unidade, dezena, centena, etc.
- Problema?

Radix Sort

- Envolve operações custosas
  - 802: 802%10, (802/10)%10, (802/100)%10
- Como o computador trabalha com a base binária, melhor dividir as partes da chaves (dígitos) por bits ou bytes
- Exemplo com inteiros: partes de 1 byte

Radix Sort

- Envolve operações custosas
  - 802: 802%10, (802/10)%10, (802/100)%10
- Como o computador trabalha com a base binária, melhor dividir as partes da chaves (dígitos) por bits ou bytes
- Exemplo com inteiros: partes de 1 byte

Radix Sort

- Envolve operações custosas
  - 802: 802%10, (802/10)%10, (802/100)%10
- Como o computador trabalha com a base binária, melhor dividir as partes da chaves (dígitos) por bits ou bytes
- Exemplo com inteiros: partes de 1 byte

Radix Sort

- Envolve operações custosas
  - 802: 802%10, (802/10)%10, (802/100)%10
- Como o computador trabalha com a base binária, melhor dividir as partes da chaves (dígitos) por bits ou bytes
- Exemplo com inteiros: partes de 1 byte
  - até 255 $\rightarrow$ 0, 1, 2, 3, ..., 255
  - 257 = 00000001 00000001 $\rightarrow$ 256 + 1
  - 258 = 00000001 00000010 $\rightarrow$ 256 + 2
  - 65792 = 00000001 00000001 00000000 $\rightarrow$ 65536 + 256 + 0

Radix Sort

- Envolve operações custosas
  - 802: 802%10, (802/10)%10, (802/100)%10
- Como o computador trabalha com a base binária, melhor dividir as partes da chaves (dígitos) por bits ou bytes
- Exemplo com inteiros: partes de 1 byte
  - até 255 $\rightarrow$ 0, 1, 2, 3, ..., 255
  - 257 = 00000001 00000001 $\rightarrow$ 256 + 1
  - 258 = 00000001 00000010 $\rightarrow$ 256 + 2
  - 65792 = 00000001 00000001 00000000 $\rightarrow$ 65536 + 256 + 0

Radix Sort

- Envolve operações custosas
  - 802: 802%10, (802/10)%10, (802/100)%10
- Como o computador trabalha com a base binária, melhor dividir as partes da chaves (dígitos) por bits ou bytes
- Exemplo com inteiros: partes de 1 byte
  - até 255 $\rightarrow$ 0, 1, 2, 3, ..., 255
  - 257 = 00000001 00000001 $\rightarrow$ 256 + 1
  - 258 = 00000001 00000010 $\rightarrow$ 256 + 2
  - 65792 = 00000001 00000001 00000000 $\rightarrow$ 65536 + 256 + 0

Radix Sort

- Envolve operações custosas
  - 802: 802%10, (802/10)%10, (802/100)%10
- Como o computador trabalha com a base binária, melhor dividir as partes da chaves (dígitos) por bits ou bytes
- Exemplo com inteiros: partes de 1 byte
  - até 255 $\rightarrow$ 0, 1, 2, 3, ..., 255
  - 257 = 00000001 00000001 $\rightarrow$ 256 + 1
  - 258 = 00000001 00000010 $\rightarrow$ 256 + 2
  - 65792 = 00000001 00000001 00000000 $\rightarrow$ 65536 + 256 + 0

Radix Sort

- Envolve operações custosas
  - 802: 802%10, (802/10)%10, (802/100)%10
- Como o computador trabalha com a base binária, melhor dividir as partes da chaves (dígitos) por bits ou bytes
- Exemplo com inteiros: partes de 1 byte
  - até 255 $\rightarrow$ 0, 1, 2, 3, ..., 255
  - 257 = 00000001 00000001 $\rightarrow$ 256 + 1
  - 258 = 00000001 00000010 $\rightarrow$ 256 + 2
  - 65792 = 00000001 00000001 00000000 $\rightarrow$ 65536 + 256 + 0

Radix Sort

- Radix:
  - ordenar pela a raiz(radix) da representação dos dados
  - extraindo o i-ésimo digito da chave
- Obs.: caso a chave seja pequena não compensa a extração dos bits: use o ***counting sort***

Radix Sort

Para extrair o D-ésimo dígito da chave

```
1  #define bitsbyte 8
2  #define bytesword 4
3
4  //00000001 << 8 = 00000001 00000000 = 2^8 = 256
5  #define R (1 << bitsbyte)
6
7  //extraindo o D-ésimo dígito de N
8  #define digit(N,D) (((N) >> ((D)*bitsbyte)) & (R-1))
9
```

- $((N) >> ((D) * bitsbyte))$: remove os primeiros D bytes
- $\&(R-1)$: pegando os últimos bitbytes (máscara)

Radix Sort

Para extrair o D-ésimo dígito da chave

- $((N) >> ((D) * bitsbyte))$: remove os primeiros D bytes
- $\&(R - 1)$: pegando os últimos bitbytes (máscara)

```
1     65792 = 00000001 00000001 00000000
2
3     //pegando o digito 1
4     00000001 00000001 00000000 >> 1*8 =
5     00000000 00000001 00000001
6
7     //aplicando a máscara
8       00000000 00000001 00000001
9     & 00000000 00000000 11111111
10    ---------------------------------
11      00000000 00000000 00000001
12
```

- E para strings? como acessar o byte-ésimo digito?

Radix Sort

Métodos de classificação - LSD

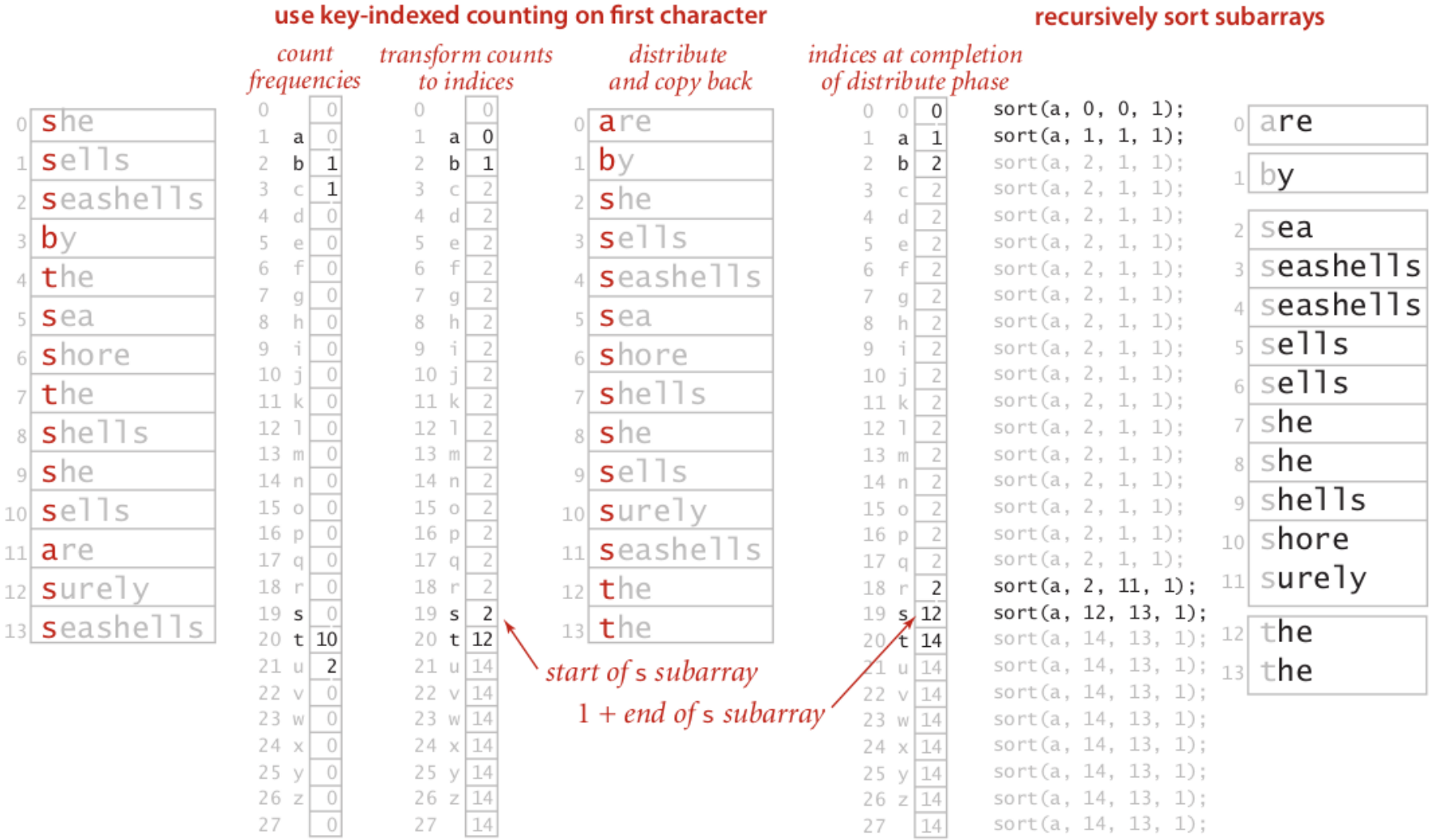
- A partir dígito menos significativo (least significant digit – LSD): direita para esquerda
  - Ordena estavelmente chaves de comprimento fixo
    - Tamanho da palavra (word) que representa o dado
    - int: 4 * 8 bits = 32 bits = 4 bytes
    - strings de tamanho W
- Complexidade: aprox. 7WN + 3WR acessos:
  - N chaves
  - chaves de tamanho W
  - cujo alfabeto são de tamanho R
- R, em geral, é muito menor que N, portanto a complexidade é proporcional a WN
- Vamos ver os códigos para inteiros e strings.

Radix Sort

Métodos de classificação - MSD

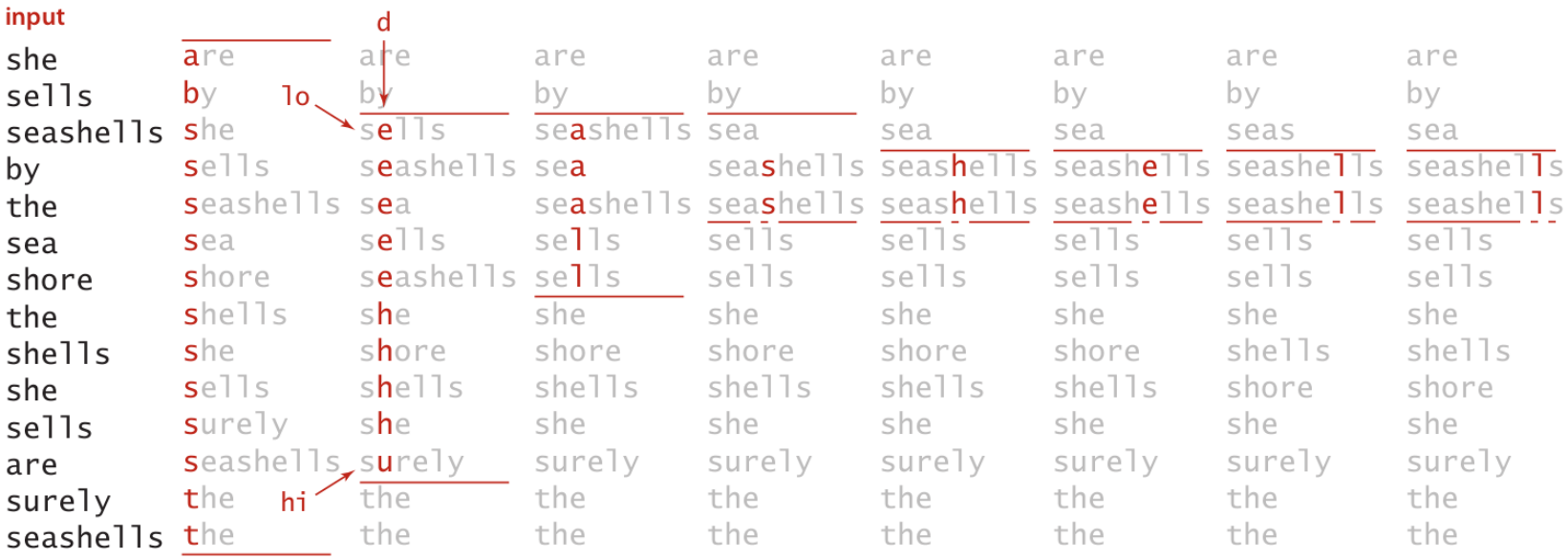
- A partir dígito mais significativo (most significant digit – MSD): esquerda para direita
  - Ordenação de propósito geral: chaves com tamanhos variáveis
    - Conjunto de várias palavras
    - string: N * 8 bits = N * 1 byte, N variável
  - Usa-se primeiro o counting sort
  - Depois, recursivamente, ordena-se os sub-vetores de cada caractere
- Complexidade: aprox. 7WN + 3WR acessos
- Vamos ver os códigos para inteiros e strings.

Radix Sort - MSD - strings

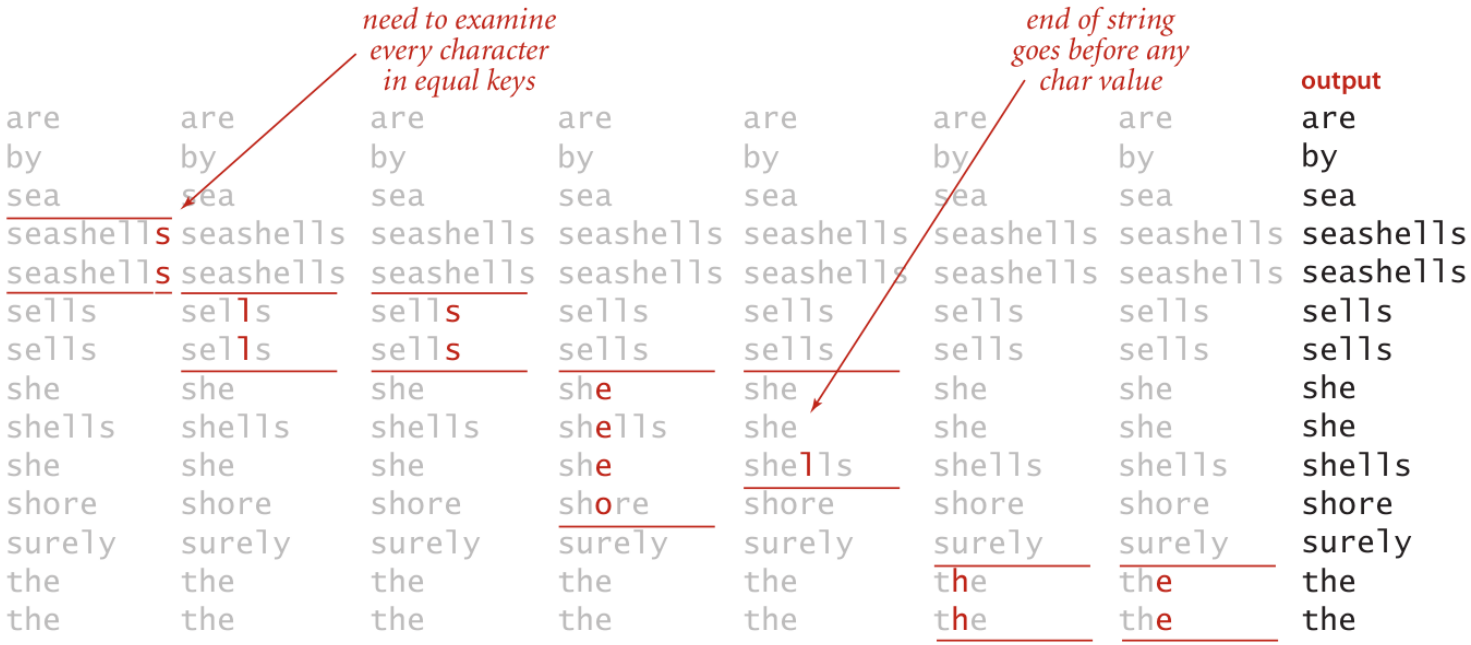


Trace of MSD string sort: top level of `sort(a, 0, 14, 0)`

Radix Sort - MSD - strings



Radix Sort - MSD - strings



Radix Sort - MSD - inteiros

170 045 075 090 802 024 002 066

045 075 090 024 002 066 170 802

Radix Sort - MSD - inteiros

045 075 090 024 002 066 170 802

002 802 024 045 066 075 170 090

Radix Sort - MSD - inteiros

002 802 024 045 066 075 170 090

170 090 002 802 024 045 075 066

Radix Sort - MSD - inteiros

170 045 075 090 802 024 002 066

170 090 002 802 024 045 075 066

Radix Sort

Métodos de ordenação

algorithm	stable?	inplace?	order of growth of typical number calls to charAt() to sort N strings from an R-character alphabet (average length w, max length W)		sweet spot
			running time	extra space	
insertion sort for strings	yes	yes	between N and N^2	1	small arrays, arrays in order
quicksort	no	yes	$N \log^2 N$	$\log N$	general-purpose when space is tight
mergesort	yes	no	$N \log^2 N$	N	general-purpose stable sort
3-way quicksort	no	yes	between N and $N \log N$	$\log N$	large numbers of equal keys
LSD string sort	yes	no	NW	N	short fixed-length strings
MSD string sort	yes	no	between N and Nw	$N + WR$	random strings