

Ordenação: Radixsort

Algoritmos e Estruturas de Dados II

Introdução

- ▶ Até agora vimos métodos de ordenação que comparam chaves
 - ▶ Esta é uma abordagem geral que funciona para qualquer tipo de chave
- ▶ Uma abordagem alternativa para ordenação é processar as chaves por partes
 - ▶ Por exemplo, começamos pelas primeiras letras do nome quando procuramos um nome num catálogo
 - ▶ Não precisamos comparar chaves

Ideia

- ▶ Quebrar uma chave em vários pedaços
 - ▶ Dígitos de um número em uma dada base (*radix*)
 - ▶ 312 tem os dígitos 3, 1 e 2 na base 10
 - ▶ 312 tem os dígitos 100111000 na base 2
 - ▶ “exemplo” tem 6 caracteres (base 256)
- ▶ Ordenar de acordo com o primeiro pedaço
 - ▶ Números cujo dígito mais à esquerda começa com 0 vêm antes de números cujo dígito mais à esquerda é 1
- ▶ Podemos ordenar repetindo esse processo para todos os pedaços

Radixsort – Ordenando um dígito

- ▶ Para isso iremos contar quantos elementos existem de cada valor

12 3	14 2	08 7	26 3	23 3	01 4	13 2

Digito	Contador
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0

Radixsort – Ordenando um dígito

- ▶ Para isso iremos contar quantos elementos existem de cada valor

12 3	14 2	08 7	26 3	23 3	01 4	13 2

Digito	Contador
0	0
1	0
2	2
3	3
4	1
5	0
6	0
7	1
8	0
9	0

Radixsort – Ordenando um dígito

- Depois calcular a posição deles no vetor ordenado

12 3	14 2	08 7	26 3	23 3	01 4	13 2

Dig	C	Posicao
0	0	0
1	0	0
2	2	0
3	3	2
4	1	5
5	0	0
6	0	0
7	1	6
8	0	0
9	0	0

Radixsort – Ordenando um dígito

- ▶ E finalmente colocar os elementos em suas posições

12 3	14 2	08 7	26 3	23 3	01 4	13 2
		12 3				

Dig	C	Posicao
0	0	0
1	0	0
2	2	0
3	3	3
4	1	5
5	0	0
6	0	0
7	1	6
8	0	0
9	0	0

Radixsort – Ordenando um dígito

- ▶ Para isso iremos contar quantos elementos existem de cada valor

12 3	14 2	08 7	26 3	23 3	01 4	13 2
14 2		12 3				

Dig	C	Posicao
0	0	0
1	0	0
2	2	1
3	3	3
4	1	5
5	0	0
6	0	0
7	1	6
8	0	0
9	0	0

Radixsort – Ordenando um dígito

- ▶ Para isso iremos contar quantos elementos existem de cada valor

12 3	14 2	08 7	26 3	23 3	01 4	13 2
142		123				087

Dig	C	Posicao
0	0	0
1	0	0
2	2	1
3	3	3
4	1	5
5	0	0
6	0	0
7	1	7
8	0	0
9	0	0

Radixsort – Ordenando um dígito

- ▶ Para isso iremos contar quantos elementos existem de cada valor

12 3	14 2	08 7	263	23 3	01 4	13 2
142		123	263			087

Dig	C	Posicao
0	0	0
1	0	0
2	2	1
3	3	4
4	1	5
5	0	0
6	0	0
7	1	7
8	0	0
9	0	0

Radixsort – Ordenando um dígito

- ▶ Para isso iremos contar quantos elementos existem de cada valor

12 3	14 2	08 7	26 3	233	01 4	13 2
142		123	263	233		087

Dig	C	Posicao
0	0	0
1	0	0
2	2	1
3	3	5
4	1	5
5	0	0
6	0	0
7	1	7
8	0	0
9	0	0

Radixsort – Ordenando um dígito

- ▶ Para isso iremos contar quantos elementos existem de cada valor

12 3	14 2	08 7	26 3	23 3	01 4	13 2
142		123	263	233	014	087

Dig	C	Posicao
0	0	0
1	0	0
2	2	1
3	3	5
4	1	6
5	0	0
6	0	0
7	1	7
8	0	0
9	0	0

Radixsort – Ordenando um dígito

- ▶ Para isso iremos contar quantos elementos existem de cada valor

12 3	14 2	08 7	26 3	23 3	01 4	13 2
142	132	123	263	233	014	087

Dig	C	Posicao
0	0	0
1	0	0
2	2	1
3	3	5
4	1	6
5	0	0
6	0	0
7	1	7
8	0	0
9	0	0

Radixsort – Ordenando o vetor

- ▶ Repetimos o mesmo processo para o próximo dígito
 - ▶ Funciona por que o método do contador que usamos anteriormente é estável!

1 2 3	1 4 2	0 8 7	2 6 3	2 3 3	0 1 4	1 3 2
1 4 2	1 3 2	1 2 3	2 6 3	2 3 3	0 1 4	0 8 7
0 1 4	1 2 3	1 3 2	2 3 3	1 4 2	2 6 3	0 8 7

Radixsort – Ordenando o vetor

- ▶ Repetimos o mesmo processo para o próximo dígito
 - ▶ Funciona por que o método do contador que usamos anteriormente é estável!

1 2 3	1 4 2	0 8 7	2 6 3	2 3 3	0 1 4	1 3 2
1 4 2	1 3 2	1 2 3	2 6 3	2 3 3	0 1 4	0 8 7
0 1 4	1 2 3	1 3 2	2 3 3	1 4 2	2 6 3	0 8 7
0 1 4	0 8 7	1 2 3	1 3 2	1 4 2	2 3 3	2 6 3

Radixsort

```
void radix(int *v, int n, int base, int num_digitos) {
    int i, j, w, count[base+1], d, idx;
    int *aux = (int *) malloc(n * sizeof(int));

    for(w = 0; w < num_digitos; w++) {
        for(j = 0; j < base; j++) count[j] = 0; // zera contador
        for(i = 0; i < n; i++) { // conta dígitos
            d = digito(v[i], w, base);
            count[d+1]++;
        } // seta índices para os dígitos
        for(j = 1; j < base; j++) count[j] += count[j-1];
        for(i = 0; i < n; i++) { // adiciona nas posições
            d = digito(v[i], w, base);
            idx = count[d];
            count[d] += 1;
            aux[idx] = v[i]; }
        for(i = 0; i < n; i++) v[i] = aux[i]; // retorna p/ v
    }
}
```

Radixsort – Análise

- ▶ Nenhuma comparação
- ▶ Inspeções de dígitos:
 - ▶ $2 * n * \text{num_digitos}$
 - ▶ Se `num_digitos` for pequeno ou constante, então radixsort tem custo linear $O(n)$
- ▶ Trocas:
 - ▶ $n * \text{num_digitos}$
 - ▶ Número de trocas também é $O(n)$

Vantagens e desvantagens

- ▶ **Vantagens:**

- ▶ Estável
- ▶ Não compara as chaves

- ▶ **Desvantagens:**

- ▶ Nem sempre é fácil otimizar a inspeção de dígitos
 - ▶ Depende do *hardware*
- ▶ Só é bom se o número de dígitos for pequeno