Ordenação: Radixsort

Algoritmos e Estruturas de Dados II

Introdução

- Até agora vimos métodos de ordenação que comparam chaves
  - Esta é uma abordagem genérica que funciona para qualquer tipo de chaves
- 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 comprar 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

123	142	087	263	233	014	132

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

123	142	087	263	233	014	132
		123				

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

123	142	087	263	233	014	132
142		123				

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

123	142	087	263	233	014	132
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

123	142	087	263	233	014	132
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

123	142	087	263	233	014	132
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

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

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!

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

12**3**	14**2**	08**7**	26**3**	23**3**	01**4**	13**2**
1**4**2	1**3**2	1**2**3	2**6**3	2**3**3	0**1**4	0**8**7
014	**1**23	**1**32	**2**33	**1**42	**2**63	**0**87
014	087	123	132	142	233	263

Radixsort

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

   for(w = 0; w < num_digitos; w++) {
      for(j = 0; j < base; j++) count[j] = 0;
      for(i = 1; i <= n; i++) {
         d = digito(v[i], w, base);
         count[d+1]++;
      }
      for(j = 1; j < base; j++) count[j] += count[j-1];
      for(i = 0; i < n; i++) {
         d = digito(v[i], w, base);
         posicao = count[d];
         count[d] += 1;
         aux[posicao] = v[i]; }
      for(i = 0; i < n; i++) v[i] = aux[i];
   }
}
```

Radixsort – Análise

- Nenhuma comparação
- Inspeções de dígitos:
  - **`2*n*num_digitos`**
  - Se **`num_digitos`** for pequeno ou constante, então radixsort tem custo linear $O(n)$
- Trocas:
  - **`n*num_digitos`**
  - Número de trocas também é $O(n)$
- Quicksort é comparável ao radixsort porque o número de dígitos é da ordem de $\lg(n)$ na base 2 e $\log_{10}(n)$ na base 10

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
    - Em geral o número de digítos tem crescimento $O(\lg(n))$