UFSC-CTC-INE

INE5384 - Estruturas de Dados

Ordenação de Dados (V)

Prof. Ronaldo S. Mello

2002/2

RadixSort

- Algoritmo de ordenação por distribuição que ordena com base nos dígitos de um número
  - prioriza (inicia por) dígitos menos significativos
- Comparação com o *BucketSort*
  - contabiliza também ocorrências de elementos
  - melhor desempenho médio
    - ***m*** não está associado a um valor máximo previsto para um conjunto de elementos e sim ao conjunto de dígitos que podem existir em um número
      - Sistema decimal: ***m = 10*** ***(m é pequeno!)***
    - o número máximo de iterações do algoritmo depende do número máximo de dígitos (***p***) que um número pode ter ***(p é pequeno, em geral)***

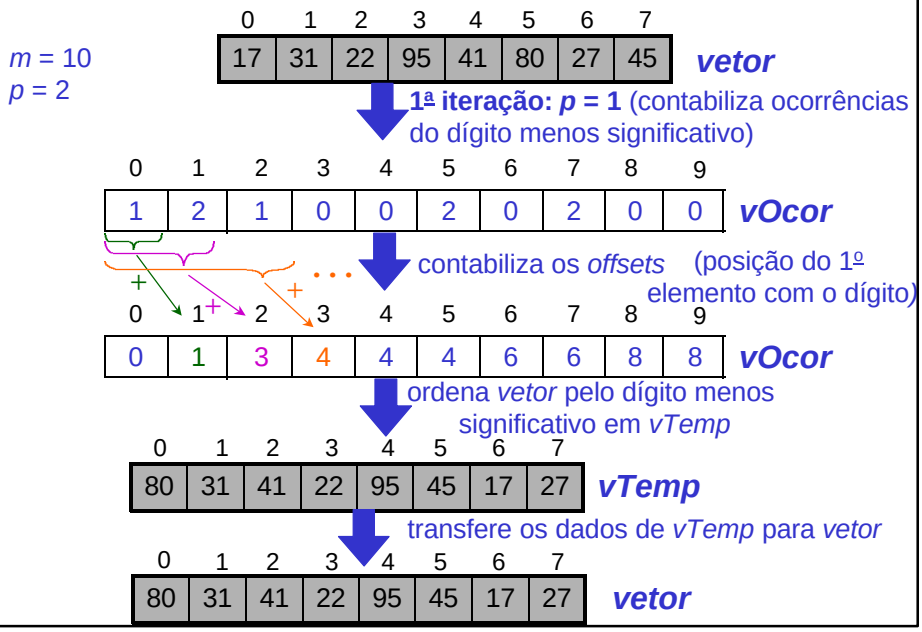
RadixSort

- Restrições
  - ordena elementos numéricos inteiros
    - ***m*** depende do conjunto de dígitos (*m* = 10, em geral)
  - elementos não ultrapassam um número de dígitos ***p***
- Exemplo
  - ordenação dos 80 empregados da empresa pelo seu tempo de serviço (em anos)
    - n = 80
    - m = 10
    - p = 2 (0 a 99)

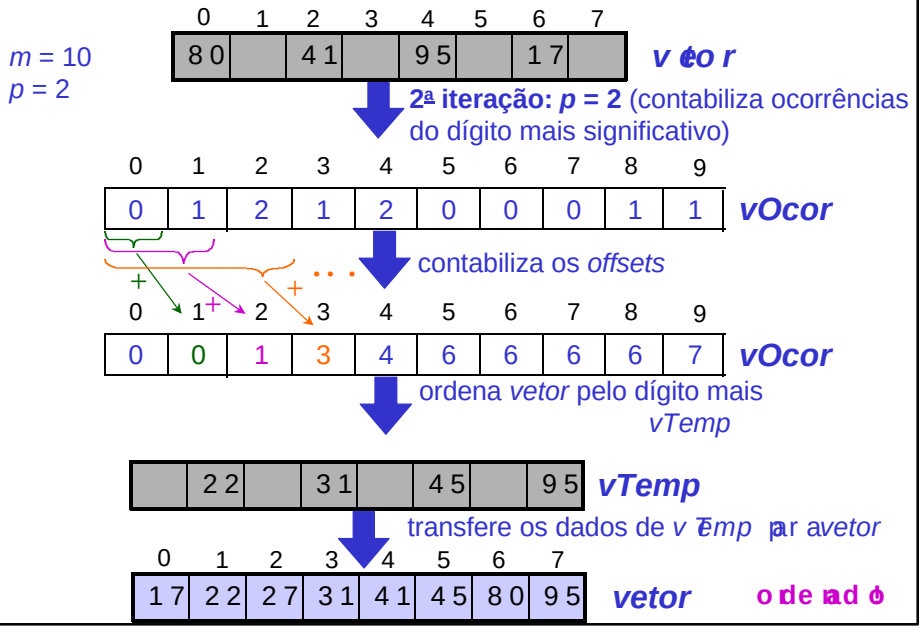
RadixSort - Funcionamento

- Utiliza dois vetores auxiliares:
  - *vOcor* (número de ocorrências de elementos)
  - *vTemp* (temporário - mantém os elementos do vetor ordenados por um certo dígito)

RadixSort - Funcionamento



RadixSort - Funcionamento



RadixSort - *Implementação*

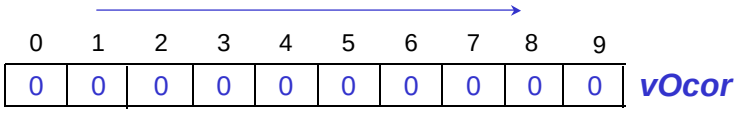
```
Classe OrdenadorRadixSort
SubClasse de Ordenador
início
    m, p inteiro;
    vOcor, vTemp inteiro[ ];

        construtor OrdenadorBucketSort (m inteiro, p inteiro);
        início
                    this.m ← m;
                    this.p ← p;
                    vOcor ← NOVO inteiro[m];
                    vTemp ← NOVO inteiro[n];
        fim;
. . .
fim;
```

RadixSort – Método Ordena

- Realiza *p* iterações. Na i-ésima iteração:
  1. inicializa *vOcor* com zero
  2. *vOcor* recebe o número de ocorrências de cada i-ésimo dígito
  3. uma vez preenchido o *vOcor*, são contabilizados nele os *offsets* para cada dígito (posições onde iniciam os elementos que possuem um certo valor de dígito)
  4. com base nestes *offsets*, *vTemp* recebe os elementos do *vetor* ordenado pelo *i*-ésimo dígito
  5. transfere-se os elementos de *vTemp* para *vetor*

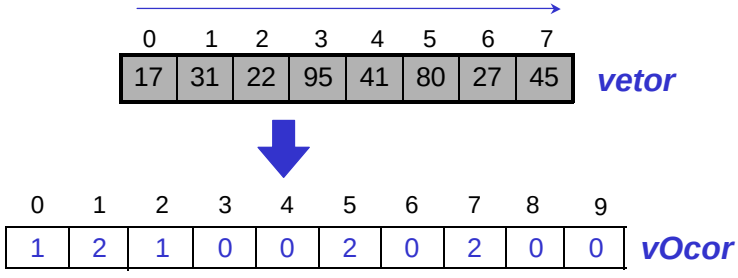
RadixSort – Etapa 1

1. Inicializa *vOcor* com zero
   – complexidade: ***O(m . p)***



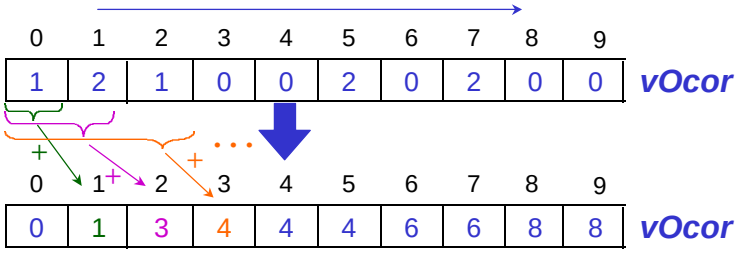
RadixSort – Etapa 2

2. Varre *vetor* e contabiliza o número de ocorrências de cada i-ésimo dígito em *vOcor*
   – complexidade: ***O(n . p)***



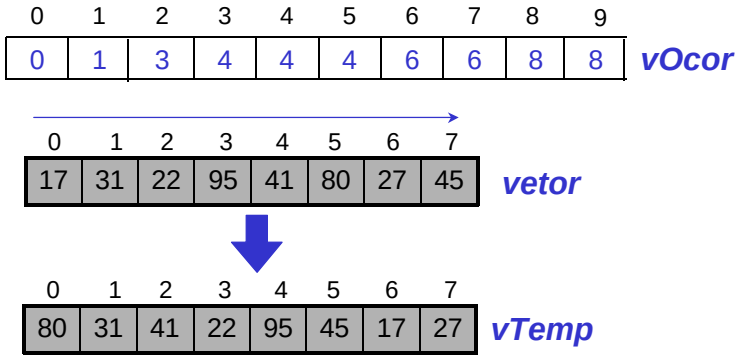
RadixSort – Etapa 3

3. Contabiliza os *offsets* em *vOcor*
   - complexidade: ***O(m . p)***



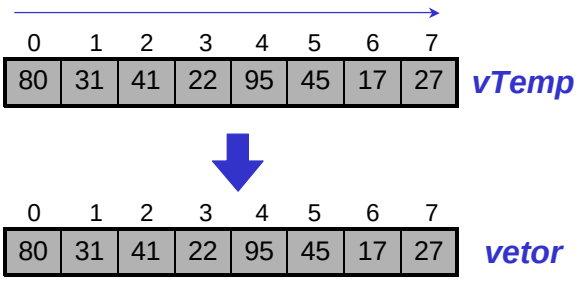
RadixSort – Etapa 4

4. Ordena *vetor* em *vTemp* com base nos *offsets* em *vOcor*
   - complexidade: ***O(n . p)***



RadixSort – Etapa 5

- Transfere os dados de *vTemp* para *vetor*
  - complexidade: ***O(n . p)***



RadixSort - Complexidade

- Complexidades envolvidas:
  - *O(m.p)* e *O(n.p)* ⇒ *O(p (m + n))*
  - considerando que *m* e *p* são pequenos (para um sistema decimal, tem-se em geral: *m = 10 e p <<= 10*), sua complexidade é assumida como linear no número de dados
- Complexidade do *RadixSort*: ***O(n)***

Exercício

- *Implementar* para a classe *OrdenadorRadixSort*:
  - *ordena*()