UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
ESCOLA DE INFORMÁTICA APLICADA
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

Primeiro Trabalho - Opção 1
Radix Sort

Bruno Lírio Alves
Pedro Paulo Gouveia
Thales Veras

Vânia Felix

Rio de Janeiro, 2 de outubro de 2013.

2. Motivação

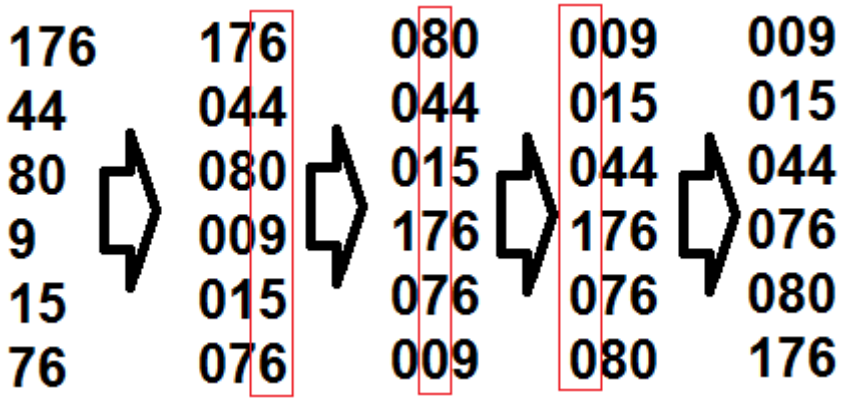
2.1 Definição do Problema

Necessidade de ordenar uma série de objetos ordenáveis (números, caracteres ou objetos chaveados por números ou caracteres) que podem estar distribuídos aleatoriamente.

Como entrada teríamos um agrupamento de números, texto, caracteres de ordenação desconhecida que serão submetidas a um algoritmo que gerará como saída um agrupamento ordenado de objetos.

No caso específico do radix-sort, sua concepção inicial se deu pelo fato da necessidade de ordenar cartões perfurados, esses cartões geralmente possuíam 80 colunas, em cada uma dessas 80 colunas a máquina poderia fazer perfurações em 12 posições.

2.2 Exemplos de Instâncias do Problema



A operação de radix sort sobre uma lista de 6 números de 3 *d* dígitos acima na imagem. A primeira coluna é a entrada com dígitos de 0 (algarismos que não tem valor, no caso do sistema decimal o dígito 0), ou seja, isso é elemento A. As colunas restantes mostram a lista após ordenações sucessivas sobre posições de dígitos cada vez mais significativas. As setas verticais indicam “nova organização” a cada dígito sobre o qual é feita a ordenação para produzir cada lista a partir da anterior.

3. Algoritmo

3.1 Descrição da ideia do algoritmo (como ele resolve o problema)

O algoritmo RADIXSORT realiza ordenação ao analisar a sequência total por rodadas, a cada vez ordenando os elementos pelo algarismo em posições iguais, à partir do algarismo menos significativo até o mais significativo. Os elementos são distribuidos em uma sequência de filas a cada rodada de acordo com o valor do algarismo analisado. Ao fim das rodadas

necessárias pelo tamanho dos elementos à serem ordenados eles estarão ordenados sem que seja feita qualquer comparação entre elementos da sequência.

3.2 A partir da ideia, mostrar exemplos de solução para as instâncias apresentadas

Originalmente, a lista não ordenada é:

176, 44, 80, 9, 15, 76

Classificando por dígito menos significativo (primeiro algarismo) dá:

80, 44, 15, 176, 76, 9

Observe que mantemos 176 antes de 76, pois 176 ocorreu antes de 76 na lista original.

Classificando por dígito seguinte (segundo algarismo) dá:

9, 15, 44, 176, 76, 80

Nota-se que 176 vem novamente antes de 76 como 176 vem antes de 76 na lista anterior e o 9 vai para o começo da lista pois considera-se 09.

Classificando por dígitos mais significativos (terceiro algarismo) dá:

9, 15, 44, 76, 80, 176

É importante perceber que cada um dos passos acima exige apenas uma única passagem ao longo dos dados.

3.3 Descrição Formal do Algoritmo

Pseudo-código:

```
para i = 1..d faça
        para j = 1..n faça
                k:= i-ésimo dígito menos significativo da representação de
                        L[j].chave na base b
                Fk <= L[j]
        j := 1
        para k = 0..b - 1 faça
                enquanto Fk (diferente de) Ø faça
                        L[j] <= Fk
                        j := j + 1
```

Onde Fk é a k-ésima fila, L representa a lista de elementos a serem ordenado, Fk reprensenta a k-ésima fila, L[j] <= Fk é a remoção de uma chave da fila Fk e Fk <= L[j] é a inserção de uma chave na k-ésima fila.

3.4 Detalhar as etapas da aplicação da técnica: tipos de estruturas de dados utilizadas na solução algorítmica, etc. (quando couber)

O radix-sort, como tratado no presente trabalho faz uso de fila que auxilia nas “reordenações” a cada passo da iteração que solucionará o problema, bem como de suas operações de inserção e remoção.

3.5 Opcional: Implementação/codificada em linguagem de programação

```
void RADIXSORT(int elemento[], int digito) {
   int i;
   int b[digito];
   int maior = elemento[0];
   int exp = 1;

   for (i = 0; i < digito; i++) {
      if (elemento[i] > maior)
         maior = elemento[i];
   }

   while (maior/exp > 0) {
      int bucket[10] = { 0 };
      for (i = 0; i < digito; i++)
         bucket[(elemento[i] / exp) % 10]++;
      for (i = 1; i < 10; i++)
         bucket[i] += bucket[i - 1];
      for (i = digito - 1; i >= 0; i--)
         b[--bucket[(elemento[i] / exp) % 10]] = elemento[i];
      for (i = 0; i < digito; i++)
         elemento[i] = b[i];
      exp *= 10;
   }
}
```

4. Análise do Algoritmo

4.1 Prova de corretude

Consideremos um conjunto de n chaves e que cada uma dessas chaves possui d dígitos, sendo o dígito mais à direita o menos significativo e o d-ésimo dígito o menos significativo, supondo ainda que as chaves estão em um sistema de base i, o algoritmo utiliza i filas para ordenar este conjunto e o faz numa iteração de d passos. No primeiro passo o conjunto de chaves é percorrido em toda sua extensão e a cada chave é analisado o dígito menos significativo, então esta chave é atribuída à fila correspondente ao valor do dígito em análise (lembrando que para uma base i, são utilizadas i filas), ou seja é realizada uma operação de inserção no fim da fila, assim ao final deste passo, com todas as chaves distribuídas pelas filas, essas são percorridas em sequência e uma operação de remoção no início de cada fila que possui uma ou mais chaves é realizada e tais chaves ordenadas num novo conjunto. Neste momento, as chaves encontram-se ordenadas em relação ao dígito menos significativo, por indução, concluímos que ao final dos i-1 passos restantes da iteração as chaves se encontrarão ordenadas.

4.2 Demonstração da complexidade/tempo do algoritmo

Supondo um sistema no qual cada dígito está no intervalo de 0 a k, sendo 0 o dígito de ordem mais baixa e k-1 o de ordem mais alta. Sejam n números de d dígitos a serem ordenados, se para uma coluna temos Θ(k), para os n números teríamos $\Theta(n + k)$, por indução sobre as colunas, $\Theta(d(n + k))$. Como temos que d, o número de dígitos de cada número, é uma constante e $k = \Theta(n)$, ou de outra forma $d * \Theta(n + k) = \Theta(n + k) = \max(\Theta(n),\Theta(k))$ portanto, podemos executar o radix-sort em tempo linear.

4. Conclusão e Discussões

4.1 Discutir as vantagens e desvantagens do procedimento adotado.

O radix sort é um algoritmo simples, de fácil entendimento e apesar de não parecer a primeira vista, intuitivo. Além disso, é um dos mais rápido e em certo sentido possui uma razoável flexibilidade que permite tanto a ordenação de números como de letras ou strings. No entanto, pode acontecer dos números a serem verificados não possuírem o mesmo número d de algarismos o que requer verificações e acertos adicionais o que pode ser bastante custoso em termos de tempo e por outro lado é um pouco inflexível em relação ao tipo de dado a ser ordenado, podendo demandar uma implementação diferente para cada tipo de dado.

4.2 É possível ter um algoritmo melhor?

Apesar de não fazer ordenação sem usar comparações, não é possível fazer mais rápido que MERGESORT O (n lg n).

5. Referências Bibliográficas

Radix Sort. (2003). WIKIPEDIA. ***Site.*** Disponível em: <http://pt.wikipedia.org/wiki/Radix_sort>. Acesso em: 01 out. 2013.

CORMEN, Thomas H.. **Algoritmos: teória e prática.** Rio de Janeiro: Editora Campus, 2002.

SZWARCFITER, Jayme Luiz **Estruturas de Dados e Seus Algoritmos.** Rio de Janeiro Editora LTC, 2010.