

Radix Sorting

- Os registros a serem ordenados podem ter chaves bastante complexas, como por exemplo sequências de caracteres (lista telefônica)
 - Ordenação via comparação de chaves
- Várias aplicações têm chaves que são *inteiros*, definidos dentro de um intervalo
- Radix sorting: métodos que tiram proveito das propriedades digitais destes números
 - Chaves tratadas como números na base M (raiz ou *radix*)
 - Processamento e comparação de *pedaços* (bits) das chaves
 - Ex: ordenar faturas de cartão de crédito considerando pilhas com o mesmo dígito em cada posição. Neste caso $M = 10$

- **Operação fundamental:** dada um chave representada por um número binário a , extrair um conjunto contíguo de bits
 - Extrair os dois bits mais significativos de um número de 10 bits
 - Desloque (shift) os bits para a direita 8 posições, e faça um “and” bit-a-bit com a máscara 0000000011
 - O *shift* elimina os bits à direita dos desejados e o *and* zera os bits a esquerda deles. Logo o resultado contém apenas os bits desejados.
- Pascal: operações shift e and são simuladas usando **div** e **mod**.
 - deslocar k bits de x para a direita: $x \text{ div } 2^k$
 - zerar todos os bits de x exceto os j bits mais à direita: $x \text{ mod } 2^j$

- *Radix sorting* depende da função $bits(x, j, k)$, que retorna os j bits que aparecem k bits à direita em x :
 - function $bits(x, k, j : integer)$: integer
 - implementa $(x \div 2^k) \bmod 2^j$
 - implementação eficiente pre-computa (ou mantém como constantes) as potências de 2.

- *Radix sorting*: idéia geral
 - Examine os bits das chaves um a um (ou em grupos), movendo as chaves com bits 0 para antes das chaves com bit 1
 - Dois algoritmos: *radix exchange* e *straight radix*
 - *Radix exchange sort*: examina bits da esquerda para a direita
 - Ordem relativa de duas chaves depende apenas do valor dos bits na posição mais significativa na qual eles diferem (primeira posição diferente da esquerda para a direita)
 - *Straight radix sort*: examina bits da direita para a esquerda

- *Radix Exchange Sorting:*
 - Rearranje os registros do arquivo tal que todas as chaves com o bit mais significativo igual a 0 aparecem antes das chaves com o bit 1.
 - Repita este processo para o 2o bit mais significativo, o 3o bit mais significativo
- Em cada passo, particiona o vetor em função do valor do bit sendo considerado: estratégia parecida com Quicksort

```
procedure radixexchange (esq, dir, b: integer; var A: vetor);
var t, i, j: integer;
```

```
begin
```

```
  if (dir > esq ) and (b >= 0) then begin
```

```
    i:= esq;
```

```
    j:= dir;
```

```
    repeat
```

```
      while (bits(A[i], b, 1) == 0) and (i < j) do i:= i + 1;
```

```
      while (bits(A[j], b, 1) == 1) and (i < j) do j:= j - 1;
```

```
      t:= A[i]; A[i] := A[j]; A[j] := t;
```

```
    until j=i;
```

```
    if (bits(A[r], b, 1) = 0) then j:= j+1;
```

```
    radixexchange(esq, j-1, b-1);
```

```
    radixexchange(j, dir, b-1);
```

```
  end;
```

```
end;
```

- Se $A[1..N]$ contém inteiros positivos menores que 2^{32} , então eles podem ser representados como números binários de 31 bits.
 - Bit mais à esquerda é o bit 30, bit mais à direita é o bit 0.
- Para ordená-los, basta chamar **radixexchange(1,N,30)**
- Note que somente os bits que diferem uma chave das demais são investigados
 - Uma vez que se atinge um prefixo que diferencia a chave das demais, os demais bits (menos significativos) daquela chave não são mais investigados.
 - Em contrapartida, todos os bits de chaves iguais são investigados
 - Radix exchange não funciona bem se arquivo contém muitas chaves iguais
- Problema em potencial: partições degeneradas (todas chaves com o mesmo bit)
 - Radix exchange ligeiramente melhor que Quicksort se chaves verdadeiramente aleatórias, mas Quicksort se adapta melhor para situações menos aleatórias.

- *Straight Radix Sorting:*
 - Examina os bits da direita para a esquerda, m bits por vez.
 - $W =$ Número de bits de uma chave
 - $W =$ múltiplo de m
 - A cada passo reordene as chaves utilizando *Distribution Counting* considerando os m bits.
 - Neste caso, o vetor de contadores tem tamanho $M = 2^m$
 - Cada passo executa em tempo linear.
 - Número de passos: W / m
 - Tem-se um compromisso entre espaço e tempo
 - Quanto maior m , mais rápido o algoritmo roda, mas mais espaço é necessário para o vetor de contadores.
 - Além do vetor de contadores, precisa também de um vetor auxiliar de tamanho N

```

procedure straightradix(var A:vetor; n: integer)
var i, j, k, pass:integer;
    count: array[0..M-1] of integer;
    b : vetor;
begin
    for pass:=0 to (W div m) -1 do begin
        for j: = 0 to M-1 do count[j] := 0;
        for i:= 1 to N do begin
            k:= bits(Ai], pass*m, m);
            count[k] = count[k] + 1;
        end;
        for j:=1 to M-1 do count[j]:= count[j-1] + count[j];
        for i:= N downto 1 do begin
            k:= bits(Ai], pass*m, m);
            b[count[k]] := Ai];
            count[k] := count[k] -1;
        end;
        for i:= 1 to N do A[i] := b[i];
    end;
end;

```

- Análise de complexidade de Radix Exchange:
 - Em cada passo, em média metade dos bits são 0 e metade são 1:
 - $C(N) = C(N/2) + N$
 - Algoritmo faz em média $N \log N$ comparações de bits
- Radix Exchange e Straight Radix fazem menos que Nb comparações de bits para ordenar N chaves de b bits
 - Linear no número de bits
- Straight radix: ordena N chaves de b bits em b/m passos, cada um com complexidade $O(N)$, usando um espaço extra para 2^m contadores (e um vetor extra).
 - Fazendo $m = b/4$: algoritmo tem número constante (4) de passos: $O(N)$
- Combinando estratégias:
 - Utilize straight radix com $m = b/4$;
 - Realize apenas 2 passos do *distribution counting*;
 - Utilize *inserção* para terminar a ordenação
 - Método muito rápido para arquivos grandes com chaves aleatórias.
- Embora tenha custo linear, Straight Radix realiza número grande de operações no loop interno (constante grande)
 - Superior a Quicksort somente para arquivos muito grandes
- Principal desvantagem do Straight Radix: espaço extra