

Exercício Prático 1

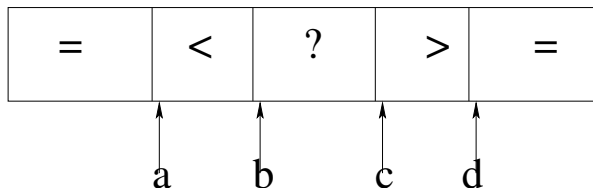
Considere a seguinte variante do quicksort utilizada para ordenar valores que contêm várias chaves:

```

sort( $s, n, d$ )
  if  $n \leq 1$  ou  $d > k$  retorne
  else
    escolha um valor  $v$  de  $s$ 
    particione  $s$  utilizando como pivô a chave de índice  $d$  de  $v$ , de modo a formar
      seqüências  $s_{<}, s_{=}, s_{>}$ , estas seqüências têm tamanho  $n_{<}, n_{=}, n_{>}$ 
    sort( $s_{<}, n_{<}, d$ )
    sort( $s_{=}, n_{=}, d + 1$ )
    sort( $s_{>}, n_{>}, d$ )
  
```

Neste algoritmo cada valor armazenado na seqüência s possui k chaves. A chamada inicial do algoritmo é: $\text{sort}(s, n, 1)$, ou seja o algoritmo inicia comparando a chave primária de cada valor presente na seqüência. Para os valores que possuem uma chave primária idêntica, ou seja pertencem a partição $s_{=}$ quando d é igual a um, o algoritmo ordena estes valores utilizando a chave secundária, a seguir o algoritmo prossegue de uma forma recursiva.

A chave da variante do quicksort mostrada acima é um algoritmo de particionamento ternário. Um exemplo de algoritmo de particionamento ternário eficiente baseia-se no seguinte laço invariante:



O laço principal do algoritmo de particionamento tem dois laços mais internos. O primeiro laço mais interno move o índice b para cima: este passa por cima dos elementos menores que o pivô, troca os elementos iguais para a parte a , e para quando encontra um elemento maior que o pivô. O segundo laço mais interno move o índice c para baixo: este passa por cima de elementos maiores que o pivô, troca os elementos iguais para a parte a para a parte d , e para quando encontra um elemento menor. O laço principal então troca os elementos apontados por b e c , incrementa b e decrementa c , e continua até que b e c se cruzem. Depois disso, os elementos iguais, residentes na extremidade do vetor, são trocados para o meio do vetor, sem qualquer comparação adicional. Este código particiona um vetor de n -elementos utilizando $n - 1$ comparações.

Implemente, em C, a variante do quicksort, com o objetivo de ordenar, lexicograficamente, palavras.

Observações:

1. A sua implementação será avaliada com relação a correção, clareza e eficiência. Note que este último critério será um critério comparativo, no qual a referência será a implementação do professor (o executável da implementação do professor pode ser obtido na página do curso no link new. O programa foi compilado para Red hat 6.1).

- (a) Um arquivo de teste primário pode ser encontrado na página do curso no link `new`.
- (b) Nenhuma diretiva de otimização de código deve ser utilizada, quando da compilação.
- (c) O nome do arquivo de teste deve ser passado como um argumento na chamada do programa.
- (d) Os arquivos de testes devem conter uma palavra por linha.
- (e) A saída deve ser o conjunto de palavras ordenadas, em ordem lexicográfica. E deve ser mostrada na tela, uma palavra por linha.
- (f) O seu programa deve ser capaz de manipular 80000 palavras e até 1000000 caracteres.
- (g) Nenhum tratamento especial necessita ser feito caso o arquivo seja maior que o especificado no item anterior.
- (h) Caso o arquivo passado como argumento não exista o usuário deve ser informado por meio de uma mensagem na tela.
- (i) O tempo deve ser calculado da seguinte forma:


```
tempo = clock();           // instrucao exatamente anterior a chamada da funcao
                             de ordenacao
quicksort(.....) ;       // chamada da funcao de ordenacao
tempo = clock() - tempo;  // instrucao exatamente posterior a chamada da
                             funcao de ordenacao
temposeg = tempo / CLOCKS_PER_SEC;
```
- (j) O seu algoritmo pode escolher o pivô aleatoriamente, contudo, você NÃO deve modificar a semente do gerador de números aleatórios.
- (k) Você pode fazer qualquer aprimoramento no quicksort, de modo a torná-lo mais eficiente; contudo, estas transformações devem ser devidamente comentadas.