

MC102 - Algoritmos e programação de computadores

Aula 22: Busca e ordenação recursiva

Pesquisa binária

- Caso uma lista A esteja ordenada, sabemos que, para qualquer i e j , $i < j$, se, e somente se, $A[i] \leq A[j]$.
- Portanto, comparando um determinado elemento com o elemento procurado, saberemos
 - se o elemento procurado é o elemento comparado,
 - se ele está antes do elemento comparado ou
 - se está depois.

Pesquisa binária recursiva

- Se fizermos isso sempre com o elemento do meio da lista, a cada comparação dividiremos a lista em duas, reduzindo nosso espaço de pesquisa.
- Se em um determinado momento o vetor, após sucessivas divisões, tiver tamanho zero, então o elemento não está no vetor.

Ver mais detalhes em `busca-bin.c`

Intercalando duas seqüências ordenadas

Problema: Dadas duas seqüências ordenadas, como criar uma terceira seqüência, com todos os elementos das duas primeiras, tal que a terceira seqüência esteja também ordenada?

Intercalando duas seqüências ordenadas

1ª solução: Copie as duas seqüências para um único vetor e ordene-as

Intercalando duas seqüências ordenadas

- Embora essa solução seja válida, ela não é a mais eficiente, pois não aproveita o fato de que as duas seqüências estão ordenadas.
- Uma outra abordagem seria tentar encontrar um elemento que seja o menor possível entre todos os elementos existentes na seqüência. Então, movemos esse elemento de sua posição original para a nova seqüência.

Intercalando duas seqüências ordenadas

- Se fizermos isso para determinarmos cada uma das posições da nova seqüência, teríamos uma seqüência ordenada, uma vez que colocamos sempre o menor elemento possível para aquela posição.
- Como as seqüências estão ordenadas, é muito simples encontrar o menor elemento dentre todos os elementos. Ele deve ser, obrigatoriamente, um dentre os menores elementos de cada seqüência (pois se isso não fosse verdade, uma das seqüências não estaria ordenada).

Intercalando duas seqüências ordenadas

- Basta então comparar esses dois elementos e escolhermos o menor. Ele é, garantidamente, o menor elemento dentre todos os elementos existentes nas duas seqüências.

Intercalando duas seqüências ordenadas

Versão final da intercalação: Como fazer isso se as seqüências não ordenadas estão no mesmo vetor, em posições consecutivas (ou seja, uma delas começa na posição *inicio* e vai até a posição *meio* - 1 e a outra, começa na posição *meio* e vai até a posição *fim*?)

Veja detalhes em `intercala.c`

MergeSort

- Uma técnica para resolver um problema é dividi-lo em parte menores, para as quais saibamos resolver o problema, e a partir desses resultados, resolver o problema maior. Essa técnica é chamada de divisão e conquista.
- A idéia do MergeSort é dividir a ordenação em duas ordenações menores (divisão) e, a partir dessas listas ordenadas, utilizar a intercalação para reunir as duas listas.

MergeSort

- Ora, ordenação é exatamente o que queremos fazer!!! Podemos utilizar uma chamada recursiva (divide-se a lista em duas até que consigamos um caso bem simples onde saibamos resolver e vamos conquistando para montar listas cada vez maiores até que chegaremos na lista total ordenada).
- Agora, precisamos de um caso simples. Trivialmente, ordenar uma lista de um elemento significa não fazer nada (uma lista de um elemento está sempre ordenada).

Veja detalhes em `mergesort.c`

MergeSort

- O algoritmo *mergesort* executa $n \log n$ comparações, que é o menor número possível de ordenações para algoritmos sem qualquer tipo de restrição e baseados em comparações. (Nenhum dos dois fatos da sentença acima pode ser demonstrado de forma trivial).
- Entretanto, este algoritmo necessita de um espaço de memória extra proporcional ao tamanho do vetor para fazer a intercalação.

Quicksort

- Novamente dividiremos o problema de ordenação em problemas menores, até encontrarmos um caso que possa ser resolvido trivialmente.
- Escolhemos um elemento do vetor arbitrariamente (chamado de pivô) e dividimos a lista em duas listas menores: uma contendo os elementos menores que o pivô e outra com os elementos maiores que o pivô. Uma posição intermediária ficará disponível para o próprio pivô.

Quicksort

- Após a divisão da lista, ordenamos as duas sublistas geradas recursivamente.
- O caso base novamente é a ordenação de listas com um elemento ou vazias, que já estão ordenadas por definição.

Veja detalhes em `quicksort.c`

Quicksort

- O algoritmo *quicksort* executa $n \log n$ comparações no caso médio, mas pode executar n^2 operações no pior caso.
- Esse pior caso pode ser evitado facilmente, por exemplo, escolhendo os pivôs aleatoriamente.
- Existem outras técnicas que garantidamente evitam o caso n^2 (por exemplo, escolher sempre a mediana do vetor como pivô).