CCM-001/MCTA003-17 – Análise de Algoritmos e Estruturas de Dados

Centro de Matemática, Computação e Cognição

Universidade Federal do ABC

Profa. Carla Negri Lintzmayer

Mergesort

Algorithm 1 Algoritmo *Mergesort* para ordenação.

```
1: Função MergeSort(A, inicio, fim)
2:     Se inicio < fim então
3:         meio = ⌊(inicio + fim)/2⌋
4:         MergeSort(A, inicio, meio)
5:         MergeSort(A, meio + 1, fim)
6:         Combina(A, inicio, meio, fim)
```

Algorithm 2 Algoritmo *Combina* auxiliar ao *Mergesort*.

```
1: Função Combina(A, inicio, meio, fim)
2:     n1 = meio − inicio + 1
3:     n2 = fim − meio
4:     Crie vetores auxiliares B[1..n1] e C[1..n2]
5:     Para i = 1 até n1 faça
6:         B[i] = A[inicio + 1 − 1]
7:     Para j = 1 até n2 faça
8:         C[i] = A[meio + j]
9:     i = 1
10:    j = 1
11:    k = inicio
12:    Enquanto i ≤ n1 e j ≤ n2 faça
13:        Se B[i] ≤ Cj então
14:            A[k] = B[i]
15:            i = i + 1
16:        Senão
17:            A[k] = C[j]
18:            j = j + 1
19:        k = k + 1
20:    Enquanto i ≤ n1 faça
21:        A[k] = B[i]
22:        i = i + 1
23:        k = k + 1
24:    Enquanto j ≤ n2 faça
25:        A[k] = C[j]
26:        j = j + 1
27:        k = k + 1
```

1 O Combina funciona?

- Ele recebe um vetor A, posições *inicio*, *meio* e fim, e considera que $A[inicio..meio]$ está ordenado e que $A[meio + 1..fim]$ também está ordenado.
- Ele promete devolver $A[inicio..fim]$ totalmente ordenado (*é isso que precisamos provar*).

Vamos analisar agora o primeiro laço **Enquanto**, da linha **??**.
Considere a frase "No início de qualquer iteração, temos $k = k_x$, $i = i_x$ e $j = j_x$, e

$P(k_x, i_x, j_x)$ = (i) o vetor $A[inicio..k_x - 1]$ está ordenado e contém os elementos de $B[1..i_x - 1]$ e $C[1..j_x - 1]$ (logo $k_x = i_x + j_x - 1$), e (ii) $B[i_x]$ e $C[j_x]$ são os menores elementos dentre os que ainda não foram copiados";

Essa frase é uma invariante desse laço, pois:

Início: antes do laço começar, $i = 1$, $j = 1$ e $k = inicio$. Temos então $P(inicio, 1, 1)$ = (i) o vetor $A[inicio..inicio - 1]$ está ordenado e contém os elementos de $B[1..0]$ e $C[1..0]$, e (ii) $B[1]$ e $C[1]$ são os menores elementos dentre os que ainda não foram copiados. Isso de fato é verdade, pois $A[inicio..inicio - 1]$ é vazio e portanto está ordenado e contém $B[1..0]$ e $C[1..0]$, também vazios. Além disso, $B[1]$ e $C[1]$ são de fato os menores elementos dentre os que ainda não foram copiados, pois nenhum elemento foi copiado e B e C estão em ordem.

Manutenção: considere uma iteração qualquer. Sejam k', i' e j' os valores nas variáveis k, i e j no início dela. Sejam k'', i'' e j'' os valores de k, i e j no início da próxima iteração. Suponha então que $P(k', i', j')$ vale no início dessa iteração. Precisamos mostrar que $P(k'', i'', j'')$ valerá no início da próxima iteração.

Na iteração, duas coisas podem acontecer. Considere primeiro que $B[i'] \leq C[j']$. Nesse caso, copiamos $B[i']$ para $A[k']$ e incrementamos apenas os valores das variáveis i e k. Assim, temos $k'' = k' + 1$, $i'' = i' + 1$ e $j'' = j'$.

Como $P(k', i', j')$ vale no início da iteração, então $B[i']$ é maior do que os elementos que estão em $A[inicio..k' - 1]$. Então, $A[inicio..k'] = A[inicio..k'' - 1]$ está em ordem e contém elementos de $B[1..i'] = B[1..i'' - 1]$ e $C[1..j' - 1] = C[1..j'' - 1]$. Como $B[i'] \leq C[j']$, então $B[i']$ é menor do que todos os elementos ainda não copiados, que estão em $B[i' + 1..n_1]$ e $C[j'..n_2]$. Como B está em ordem, então $B[i' + 1]$ é menor do que qualquer elemento em $B[i' + 2..n_1]$. Logo, $B[i' + 1]$ e $C[j']$ são os menores elementos dentre os que ainda não foram copiados. Então, nesse caso, temos que $P(k'', i'', j'')$ vale ao fim dessa iteração.

Se $B[i] > C[j]$, então com uma análise similar podemos mostrar que $P(k'', i'', j'')$ vale ao fim dessa iteração. Nesse caso, $k'' = k' + 1$, $i'' = i'$ e $j'' = j' + 1$.

Assim, no fim do laço **Enquanto**, da linha **??**, sejam k_f, i_f e j_f os valores de k, i e j, respectivamente. Note que também podemos ter dois casos para o laço ter acabado. Se $i_f = n_1 + 1$, então a invariante nos diz que

> o vetor $A[inicio..k_f - 1]$ está ordenado e contém os elementos de $B[1..n_1]$ e $C[1..j_f - 1]$ (logo $k_f = n_1 + j_f - 1$), e $B[n_1 + 1]$ e $C[j_f]$ são os menores elementos dentre os que ainda não foram copiados.

Aqui, temos a informação de que B foi todo copiado para A e que $C[j_f..n_2]$ ainda não foi. O teste do segundo laço **Enquanto**, da linha **??**, falha. O terceiro laço **Enquanto**, da linha **??**, será executado e copiará $C[j_f..n_2]$ para A a partir da posição k_f. Como serão $n_2 - j_f + 1$ elementos copiados nesse laço e já tínhamos $k_f = n_1 + j_f - 1$ elementos copiados no laço anterior, ao todo temos $n_2 - j_f + 1 + n_1 + j_f - 1 = n_1 + n_2$ elementos copiados para A. Como $C[j_f]$ é o menor elemento dentre os que não foram copiados e C está em ordem, $A[inicio..fim]$ ficará totalmente em ordem (pois $fim - inicio + 1 = n_1 + n_2$).

Uma análise similar pode ser feita para o caso do laço **Enquanto** da linha **??** ter terminado porque $j_f = n_2 + 1$.

Combina, portanto, termina com $A[inicio..fim]$ ordenado e contendo todos os elementos que haviam em B e C.

2 O MergeSort funciona?

- Ele recebe um vetor A e posições $inicio$ e fim.
- Ele promete ordenar $A[inicio..fim]$.

Queremos provar que "MergeSort ordena qualquer vetor A com n elementos", qualquer que seja o conteúdo de A. Vamos fazer isso por indução no valor de n.

No caso base do algoritmo, temos $inicio \geq fim$, o que implica $n \leq 1$. Aqui, o algoritmo MergeSort não faz nada. De fato, se $inicio > fim$, $A[inicio..fim]$ é vazio e, por vacuidade, está ordenado. Se $inicio = fim$, $A[inicio..fim]$ contém um elemento e, portanto, também está ordenado. Então o MergeSort funciona no caso base.

Considere então que $inicio < fim$ e seja $n = fim - inicio + 1$. Note que $n > 1$. Suponha que "MergeSort ordena qualquer vetor A com k elementos", onde $k < n$. Precisamos provar que ele ordena o vetor com n elementos.

A primeira coisa que o MergeSort faz é calcular a posição $meio = \lfloor (inicio+fim)/2 \rfloor$, do elemento central de $A[inicio..fim]$.

Em seguida, faz uma chamada a MergeSort(A, $inicio$, $meio$), isto é, uma chamada passando um vetor com $meio - inicio + 1$ elementos. Veja que

$$\begin{aligned} meio - inicio + 1 &= \left\lfloor \frac{inicio + fim}{2} \right\rfloor - inicio + 1 \\ &\leq \frac{inicio + fim}{2} - inicio + 1 \\ &= \frac{fim - inicio + 2}{2} = \frac{n+1}{2}\,. \end{aligned}$$

E $(n+1)/2 < n$ sempre que $n > 1$. Assim, essa chamada reduz o tamanho do vetor inicial e, por hipótese, corretamente ordena $A[inicio..meio]$.

Em seguida, outra chamada recursiva é feita, a MergeSort(A, $meio + 1$, fim), que é uma chamada passando um vetor com $fim - meio$ elementos. Veja que

$$\begin{aligned} fim - meio &= fim - \left\lfloor \frac{inicio + fim}{2} \right\rfloor \\ &\leq fim - \left(\frac{inicio + fim}{2} \right) \\ &= \frac{fim - inicio}{2} \leq \frac{n}{2}\,. \end{aligned}$$

E $n/2 < n$ sempre que $n > 1$. Essa chamada, também por hipótese, corretamente ordena $A[meio + 1..fim]$.

O próximo passo do algoritmo é chamar Combina(A, $inicio$, $meio$, fim). Como vimos na seção anterior, o Combina funciona sempre que $A[inicio..meio]$ e $A[meio + 1..fim]$ já estão ordenados, o que é o caso, como visto acima. Logo, $A[inicio..fim]$ é totalmente ordenado.