Algoritmos de Ordenação: *MergeSort*

ACH2002 - Introdução à Ciência da Computação II

Delano M. Beder

Escola de Artes, Ciências e Humanidades (EACH)
Universidade de São Paulo
dbeder@usp.br

10/2008

Material baseado em slides do professor Marcos Chaim

Projeto por Indução Forte

Hipótese de indução forte

Sabemos ordenar um conjunto de $1 \leq k < n$ inteiros.

- **Caso base:** n = 1. Um conjunto de um unico elemento está ordenado.
- **Passo da Indução (Primeira Alternativa):** Seja S um conjunto de $n \geq 2$ inteiros. Podemos particionar S em dois conjuntos S_1 e S_2, de tamanhos $\lfloor n/2 \rfloor$ e $\lceil n/2 \rceil$. Como $n \geq 2$, ambos S_1 e S_2 possuem menos do que n elementos. Por hipótese de indução, sabemos ordenar os conjuntos S_1 e S_2.

  Podemos então obter S ordenado intercalando os conjuntos ordenados S_1 e S_2.

Projeto por Indução Forte

- Esta indução dá origem ao algoritmo de divisão e conquista *MergeSort*.
- Na implementação do algoritmo, o conjunto S é um vetor de tamanho n.
- A operação de divisão é imediata, o vetor é dividido em dois vetores com metade do tamanho do original, que são ordenados recursivamente.
- O trabalho do algoritmo está concentrado na conquista: a intercalação dos dois subvetores ordenados.
- Para simplificar a implementação da operação de intercalação e garantir sua complexidade linear, usamos um vetor auxiliar.

Ordenação por Intercalação – *MergeSort*

- Ordenação por intercalação/junção/fusão: mais conhecida como *MergeSort*.
- **Dividir**: divide a seqüência de n elementos a serem ordenados em duas subseqüências de tamanho $\lceil n/2 \rceil$ e $\lfloor n/2 \rfloor$.
- **Conquistar**: ordena as duas subseqüências recursivamente por intercalação.
- **Combinar**: faz a intercalação das duas seqüências ordenadas de modo a produzir a resposta ordenada.

Ordenação por Intercalação – *MergeSort*

- **Dividir** é fácil. Basta dividir a seqüência em dois.
- **Conquistar** também não é difícil.
  - Dividindo o problema em *dois* necessariamente vamos chegar a uma seqüencia de tamanho *um* cuja ordenação é trivial.
- **Combinar**, esse é o problema! Como combinar?

Fusão de Vetores

No livro [2], foi apresentado um método para fusão de vetores:

Fusão de Vetores

```
int [] fusao(int [] a, int [] b) {
 int posa = 0,
     posb = 0,
     posc = 0;
 int [] c = new int [a.length + b.length];

 // Enquanto nenhuma das seqüências está vazia...

 while (posa < a.length && posb < b.length) {
  // Pega o menor elemento das duas seqüências

  if(b[posb] <= a[posa]) {
    c[posc] = b[posb];
    posb++;
  } else {
    c[posc] = a[posa];
    posa++;
  }
  posc++;
 }
```


Fusão de Vetores (continuação)

Fusão de Vetores (continuação)

```
 // Completa com a seqüência que ainda não acabou

 while (posa < a.length) {
   c[posc] = a[posa];
   posc++;
   posa++;
 }

 while (posb < b.length) {
   c[posc] = b[posb];
   posc++;
   posb++;
 }

 return c; // retorna o valor resultado da fusão
}
```


Ordenação por Intercalação – *MergeSort*

Esse algoritmo não é exatamente o que desejamos. Ele retorna um novo arranjo que contém a **fusão**.

O que queremos, no entanto, é realizar a fusão de subseqüências de um vetor. Algo assim:

```
void merge(int [] A, int p, int q, int r) {
// A subseqüência A[p...q] está ordenada
// A subseqüência A[q+1...r] está ordenada

// Faz a junção das duas subseqüências
...
// A subseqüência A[p...r] está ordenada
}
```

Ordenação por Intercalação – *MergeSort*

Utilizando a mesma idéia da fusão de dois arranjos, com a assinatura e restrições definidas acima, tem-se:

Fusão

```
void merge(int [] A, int p, int q, int r) {
    // A subseqüência A[p...q] está ordenada
    // A subseqüência A[q+1...r] está ordenada
1:  int i, j, k;
    // Faz cópias - seq1 = A[p...q] e seq2 = A[q+1...r]
2:  int tamseq1 = q - p + 1; // tamanho da subseqüência 1
3:  int tamseq2 = r - q; // tamanho da subseqüência 2
4:  int [] seq1 = new int [tamseq1];
5:  for(i=0; i < seq1.length; i++) {
      seq1[i] = A[p+i];
    }

6:  int [] seq2 = new int [tamseq2];
7:  for(j=0; j < seq2.length; j++) {
      seq2[j] = A[q+j+1];
    }
```


Ordenação por Intercalação – *MergeSort*

Fusão (Continuação)

```
    // Faz a junção das duas subseqüências

8:  k = p; i = 0; j = 0;

9:  while (i < seq1.length && j < seq2.length) {
    // Pega o menor elemento das duas seqüências

10:   if(seq2[j] <= seq1[i]) {
11:     A[k] = seq2[j];
12:     j++;
      }
      else {
13:     A[k] = seq1[i];
14:     i++;
      }
15:   k++;
    }
```


Ordenação por Intercalação – *MergeSort*

Fusão (Continuação 2)

```
    // Completa com a seqüência que ainda não acabou

16:  while (i < seq1.length) {
17:    A[k] = seq1[i];
18:    k++;
19:    i++;
     }

20:  while (j < seq2.length) {
21:    A[k] = seq2[j];
22:    k++;
23:    j++;
     }
     // A subseqüência A[p...r] está ordenada
}
```


Ordenação por Intercalação – *MergeSort*

Agora que já sabemos como combinar (*merge*), podemos terminar o algoritmo de ordenação por intercalação:

```
void mergeSort(int [] numeros, int ini, int fim) {

  if(ini < fim) {
    //Divisao
1:  int meio = (ini + fim)/2;

    // Conquista
2:  mergeSort(numeros, ini, meio);
3:  mergeSort(numeros, meio+1, fim);

    // Combinação
4:  merge(numeros, ini, meio, fim);
  }
  // Solução trivial: ordenacao de um único número.
}
```

MergeSort

1. *MergeSort* (A, 0, 7)

2	8	7	1	3	5	6	4	ini = 0, fim = 7, meio = 3

1.1. *MergeSort* (A, 0, 3)

2	8	7	1	3	5	6	4	ini = 0, fim = 3, meio = 1

1.1.1. *MergeSort* (A, 0, 1)

2/2	8	7	1	3	5	6	4	ini = 0, fim = 1, meio = 0

1.1.1.1. *MergeSort* (A, 0, 0) ×
1.1.1.2. *MergeSort* (A, 1, 1) ×
1.1.1.3. *Merge* (A, 0, 1)

2	8	7	1	3	5	6	4
2	8	7	1	3	5	6	4

MergeSort

1.1.2. *MergeSort* (A, 2, 3)

2	8	7/7	1	3	5	6	4	ini = 2, fim = 3, meio = 2

1.1.2.1. *MergeSort* (A, 2, 2) ×
1.1.2.2. *MergeSort* (A, 3, 3) ×
1.1.2.3. *Merge* (A, 2, 3)

2	8	7	1	3	5	6	4
2	8	1	7	3	5	6	4

1.1.3. *Merge* (A, 0, 3)

2	8	7	1	3	5	6	4
1	2	7	8	3	5	6	4

MergeSort

1.2. *MergeSort* (A, 4, 7)

1	2	7	8	3	5	6	4	ini = 4, fim = 7, meio = 5

1.2.1. *MergeSort* (A, 4, 5)

1	2	7	8	3/3	5	6	4	ini = 4, fim = 5, meio = 4

1.2.1.1. *MergeSort* (A, 4, 4) ×
1.2.1.2. *MergeSort* (A, 5, 5) ×
1.2.1.3. *Merge* (A, 4, 5)

1	2	7	8	3	5	6	4
1	2	7	8	3	5	6	4

MergeSort

1.2.2. *MergeSort* (A, 6, 7)

1	2	7	8	3	5	6/6	4	ini = 6, fim = 7, meio = 6

1.2.2.1. *MergeSort* (A, 6, 6) ×
1.2.2.2. *MergeSort* (A, 7, 7) ×
1.2.2.3. *Merge* (A, 6, 7)

1	2	7	8	3	5	6	4
1	2	7	8	3	5	4	6

1.2.3. *Merge* (A, 4, 7)

1	2	7	8	3	5	4	6
1	2	7	8	3	4	5	6

1.3. *Merge* (A, 0, 7)

1	2	7	8	3	4	5	6
1	2	3	4	5	6	7	8

Análise da Fusão (*Merge*)

Complexidada temporal:

- T(Linhas 1-3): $O(1)$.
- T(Linhas 4-7): O(seq1.length + seq2.length)
- T(Linhas 9-23): O(seq1.length + seq2.length)

Como é uma seqüência,

$\Rightarrow$ T(1-23) = O(seq1.length + seq2.length) + O(seq1.length + seq2.length) + $O(1)$

$\Rightarrow$ T(1-23) = O(max(seq1.length + seq2.length, seq1.length + seq2.length, 1))

$\Rightarrow$ T(1-23) = O(seq1.length + seq2.length)

Se fizermos n_1 = seq1.length e n_2 = seq2.length

$\Rightarrow$ T(1-23) = $O(n_1 + n_2)$.

Análise da ordenação por intercalação – *MergeSort*

- Ordenação por intercalação utiliza a abordagem dividir e conquistar.
- Então podemos antes fazer análise genérica dos algoritmos que utilizam essa abordagem.
- Dividir e conquistar envolve três passos:
  1. Dividir
  2. Conquistar
  3. Combinar
- Portanto, a complexidade de tempo de algoritmos *dividir e conquistar* para um entrada de tamanho n é:
  - T(n) = Dividir(n) + Conquistar(n) + Combinar(n).

Análise da ordenação por intercalação – *MergeSort*

- Para entradas pequenas, isto é, para $n \leq c$, podemos assumir que $T(n) = O(1)$.
- Vamos supor que o problema seja dividido em a subproblemas, cada um com $1/b$ do tamanho original.
- Se levamos $D(n)$ para dividir o problema em subproblemas e $C(n)$ para combinar as soluções dados aos subproblemas, então tem-se a recorrência $T(n)$ tal que:

$$T(n) = \begin{cases} O(1) & \text{se} \quad n \leq 0 \\ aT(n/b) + D(n) + C(n) & \text{caso contrário} \end{cases}$$

Análise da ordenação por intercalação – *MergeSort*

- Sem perda de generalidade, podemos supor que n é uma potência de 2: $n = 2^i$ para $i \geq 0$.
- Para $n = 1$, isto é, a ordenação de um vetor com um único elemento, a complexidade temporal é $T(1) = O(1)$, pois é o caso base e não requer fusão.

Análise da ordenação por intercalação – *MergeSort*

- **Dividir:** a etapa de dividir simplesmente calcula o ponto médio do subvetor, o que demora um tempo constante.
  - $D(n) = O(1)$
- **Conquistar:** resolvemos recursivamente dois subproblemas; cada um tem o tamanho $n/2$
  - Contribui com $2T(n/2)$ para o tempo de execução
- **Combinar:** Já foi calculado que o método **merge** em um subvetor de tamanho n
  - $C(n) = n_1 + n_2$ é $O(n)$.

Análise da ordenação por intercalação – *MergeSort*

Portanto, a complexidade $T(n)$ para o algoritmo de ordenação por intercalação *MergeSort* é:

$$T(n) = \begin{cases} O(1) & \text{se} \quad n = 1 \\ 2T(n/2) + n & \text{caso contrário} \end{cases}$$

Teorema Mestre (CLRS): temos que $a = 2$, $b = 2$ e $f(n) = n$.

Desta forma $n^{\log_b a} = n^{\log_2 2} = n^1 = n$.

Desde que $f(n) \in \Theta(n^{\log_2 2}) = \Theta(n)$, nós podemos aplicar o caso 2 do Teorema Mestre.

$T(n) = \Theta(n^{\log_b a} \log n) = \Theta(n^{\log_2 2} \log n) = \Theta(n \log n)$.

Logo, a solução é $T(n) \in \Theta(n \log n)$.

Análise da ordenação por intercalação – *MergeSort*

- É possível fazer a intercalação dos subvetores ordenados sem o uso de vetor auxiliar ? Sim! Basta deslocarmos os elementos de um dos subvetores, quando necessário, para dar lugar ao mínimo dos dois subvetores.
- No entanto, a etapa de intercalação passa a ter complexidade $\Theta(n^2)$, resultando na seguinte recorrência:

$$T(n) = \begin{cases} O(1) & \text{se} \quad n = 1 \\ 2T(n/2) + n^2 & \text{caso contrário} \end{cases}$$

- Ou seja, a complexidade do *MergeSort* passa a ser $\Theta(n^2)$. Como era de se esperar, a eficiência da etapa de intercalação é crucial para a eficiência do *MergeSort*.

Referências

Referências utilizadas: [1] (páginas 21-48).

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest & C. Stein. *Algoritmos - Tradução da 2a. Edição Americana*. Editora Campus, 2002.

[2] F. Kon, A. Goldman, P.J.S. Silva. *Introdução à Ciência de Computação com Java e Orientado a Objetos*, IME - USP, 2005.