Divisão e Conquista

Prof. Juliano Foleis

Divisão e Conquista

- Técnica de projeto de algoritmos
- Baseia-se em dividir um problema em subproblemas menores recursivamente. Desta forma, quando o problema se torna pequeno suficiente uma solução simples pode ser aplicada. As soluções dos problemas menores são combinadas para obter a solução final do problema original.

Divisão e Conquista - Passos

Divisão e Conquista - Passos

1. Dividir o problema em um número de subproblemas que são instâncias menores do mesmo problema;

Divisão e Conquista - Passos

1. Dividir o problema em um número de subproblemas que são instâncias menores do mesmo problema;
2. Conquistar os subproblemas, resolvendo-os recursivamente. Se os subproblemas são pequenos o suficiente, podem ser resolvidos de forma trivial;

Divisão e Conquista - Passos

1. Dividir o problema em um número de subproblemas que são instâncias menores do mesmo problema;
2. Conquistar os subproblemas, resolvendo-os recursivamente. Se os subproblemas são pequenos o suficiente, podem ser resolvidos de forma trivial;
3. Combinar as soluções dos subproblemas para obter o resultado final.

Exemplo - *MergeSort*

Exemplo - *MergeSort*

1. Dividir o vetor com n elementos em 2 subvetores adjacentes com $\frac{n}{2}$ elementos cada;

Exemplo - *MergeSort*

1. Dividir o vetor com n elementos em 2 subvetores adjacentes com $\frac{n}{2}$ elementos cada;
2. Conquista: ordenar os dois subvetores recursivamente utilizando `MergeSort`;

Exemplo - *MergeSort*

1. Dividir o vetor com n elementos em 2 subvetores adjacentes com $\frac{n}{2}$ elementos cada;
2. Conquista: ordenar os dois subvetores recursivamente utilizando `MergeSort`;
3. Combinar os dois vetores ordenados recursivamente usando a intercalação (`Merge`), obtendo o vetor com n elementos ordenado.

MergeSort - Merge

A principal operação do MergeSort é o procedimento `Merge`, que é responsável por intercalar 2 subvetores adjacentes ordenados, produzindo um vetor ordenado na saída contendo todos os elementos do vetor.

Merge

```
procedure MERGE(V:[int], p:int, q:int, r:int)
  n1 = q-p+1
  n2 = r-q
  L[1..n1+1]
  R[1..n2+1]
  for i = 1; i<=n1; i++ do
    L[i] = V[p+i-1]
  end for
  for j = 1; j<=n2; j++ do
    R[i] = V[q+j]
  end for
  L[n1+1] = infinito, R[n2+1] = infinito
  i = 1, j = 1
  for k = p; k<=r; k++ do
    if L[i]  <= R[i] then
      V[k] = L[i]
      i++
    else
      V[k] = R[j]
      j++
    end if
  end for
end procedure
```

Merge

```
 1 procedure MERGE(V:[int], p:int, q:int, r:int)
 2   n1 = q-p+1
 3   n2 = r-q
 4   L[1..n1+1]
 5   R[1..n2+1]
 6   for i = 1; i<=n1; i++ do
 7     L[i] = V[p+i-1]
 8   end for
 9   for j = 1; j<=n2; j++ do
10     R[i] = V[q+j]
11   end for
12   L[n1+1] = infinito, R[n2+1] = infinito
13   i = 1, j = 1
14   for k = p; k<=r; k++ do
15     if L[i]  <= R[i] then
16       V[k] = L[i]
17       i++
18     else
19       V[k] = R[j]
20       j++
21     end if
22   end for
23 end procedure
```

Análise: Seja $n = r - p + 1$. As linhas 2–5 e 12–15 executam em tempo constante. O laço for `for` das linhas 6–8 executa $n_1 = q - p + 1$ vezes e o laço `for` das linhas 9–11 executa $n_2 = r - q$ vezes. Portanto o custo do trecho das linhas 6–11 tem custo:

$$
\begin{aligned}
& n_1 + n_2 \\
&= (q - p + 1) + (r - q) \\
&= r - p + 1 \\
&= n = \Theta(n).
\end{aligned}
$$

Merge

```
1  procedure MERGE(V:[int], p:int, q:int, r:int)
2    n1 = q-p+1
3    n2 = r-q
4    L[1..n1+1]
5    R[1..n2+1]
6    for i = 1; i<=n1; i++ do
7      L[i] = V[p+i-1]
8    end for
9    for j = 1; j<=n2; j++ do
10     R[i] = V[q+j]
11   end for
12   L[n1+1] = infinito, R[n2+1] = infinito
13   i = 1, j = 1
14   for k = p; k<=r; k++ do
15     if L[i]  <= R[i] then
16       V[k] = L[i]
17       i++
18     else
19       V[k] = R[j]
20       j++
21     end if
22   end for
23 end procedure
```

O laço das linhas $14 - 22$ repete n vezes com operações de tempo constante, portanto, $\Theta(n)$. Assim, o tempo total do procedimento `Merge` é

$$\underbrace{\Theta(n)}_{l.6-11} + \underbrace{\Theta(n)}_{l.14-22} = \Theta(n).$$

MergeSort

```
procedure MergeSort(V:[int], p:int, r:int)
  if p<r then
    q = chao((p + r)/2)
    MergeSort(V, p, q)
    MergeSort(V, q+1, r)
    Merge(V, p, q, r)
  end if
end procedure
```

MergeSort - Caso base

```
1 procedure MergeSort(V:[int], p:int, r:int)
2   if p<r then
3     q = chao((p + r)/2)
4     MergeSort(V, p, q)
5     MergeSort(V, q+1, r)
6     Merge(V, p, q, r)
7   end if
8 end procedure
```

Análise: Como p e r delimitam o vetor sendo ordenado, seja $n = r - p + 1$. No caso base $p \geq r$, o que implica que $n \leq 1$. Neste caso não é necessário processar o vetor, uma vez que ele já se encontra ordenado ou vazio. Assim, o único custo no caso base é o da linha 2, que é $\Theta(1)$.

MergeSort - Caso recursivo

```
1 procedure MergeSort(V:[int], p:int, r:int)
2   if p<r then
3     q = chao((p + r)/2)
4     MergeSort(V, p, q)
5     MergeSort(V, q+1, r)
6     Merge(V, p, q, r)
7   end if
8 end procedure
```

Análise: Conforme analisado anteriormente, o custo de `Merge` na linha 6 é $\Theta(n)$. Seja $T(n)$ a função que descreve o custo do `MergeSort`. O custo da linha 4 é calculado da mesma forma que o custo do procedimento todo. Entretanto, somente os elementos entre p e q são processados no subproblema da linha 4. Como q corresponde ao elemento do meio entre p e r, então neste subproblema apenas a primeira metade dos elementos são processados, resultando em custo $T\left(\frac{n}{2}\right)$.

MergeSort - Caso recursivo

```
1 procedure MergeSort(V:[int], p:int, r:int)
2   if p<r then
3     q = chao((p + r)/2)
4     MergeSort(V, p, q)
5     MergeSort(V, q+1, r)
6     Merge(V, p, q, r)
7   end if
8 end procedure
```

O custo da linha 5 também é calculado da mesma forma que o custo do procedimento todo, mas processando apenas a segunda metade do vetor original, uma vez que processa os elementos entre as posições $q + 1$ e r. Desta forma, o custo da linha 5 é $T\left(\frac{n}{2}\right)$.

MergeSort - Caso recursivo

```
procedure MergeSort(V:[int], p:int, r:int)
  if p<r then
    q = chao((p + r)/2)
    MergeSort(V, p, q)
    MergeSort(V, q+1, r)
    Merge(V, p, q, r)
  end if
end procedure
```

MergeSort - Caso recursivo

```
1 procedure MergeSort(V:[int], p:int, r:int)
2   if p<r then
3     q = chao((p + r)/2)
4     MergeSort(V, p, q)
5     MergeSort(V, q+1, r)
6     Merge(V, p, q, r)
7   end if
8 end procedure
```

Portanto, o custo do `MergeSort` no caso recursivo é:

$$\underbrace{T\left(\frac{n}{2}\right)}_{l.4} + \underbrace{T\left(\frac{n}{2}\right)}_{l.5} + \underbrace{\Theta(n)}_{l.6} = 2T\left(\frac{n}{2}\right) + \Theta(n)$$

MergeSort - Custo Total

```
procedure MergeSort(V:[int], p:int, r:int)
  if p<r then
    q = chao((p + r)/2)
    MergeSort(V, p, q)
    MergeSort(V, q+1, r)
    Merge(V, p, q, r)
  end if
end procedure
```

MergeSort - Custo Total

```
procedure MergeSort(V:[int], p:int, r:int)
  if p<r then
    q = chao((p + r)/2)
    MergeSort(V, p, q)
    MergeSort(V, q+1, r)
    Merge(V, p, q, r)
  end if
end procedure
```

O custo total do `MergeSort` pode ser escrito desta forma:

$$T(n) = \begin{cases} \Theta(1) & \text{se } n \leq 1 \\ 2T(\frac{n}{2}) + \Theta(n) & \text{se } n > 1 \end{cases}$$

MergeSort - Custo Total

$$T(n) = \begin{cases} \Theta(1) & \text{se } n \leq 1 \\ 2T(\frac{n}{2}) + \Theta(n) & \text{se } n > 1 \end{cases}$$

Note que a função de custo é definida em função dela mesma com subproblemas menores. Este tipo de equação é denominada **equação de recorrência**, ou simplesmente, **recorrência**.

Resolver recorrências é uma arte. Embora algumas formas de recorrências possuam soluções conhecidas, como o método mestre, a solução de recorrências é realizada de forma “artesanal”. No entanto, serão apresentadas técnicas que auxiliam a resolução de recorrências:

- Método da Substituição
- Árvore de Recursão
- Método Mestre

Bibliografia

[CRLS] CORMEN, T. H. et al. Algoritmos: Teoria e Prática. Elsevier, 2012. 3a Ed. Capítulo 2 (Dando a Partida), Seção 2.3 (Projeto de Algoritmos)