Construção e Análise de Algoritmos

aula 10: O método das equações de recorrência

1 Introdução

Considere um problema P qualquer.

Na aula passada, vimos que ao descobrir como

- Quebrar o problema P em problemas menores do mesmo tipo
- Combinar as soluções dos subproblemas para obter a solução do problema original
- Resolver as instâncias triviais do problema obtidas por divisão sucessiva

o problema P está essencialmente resolvido.

De fato, apenas com essas informações é possível construir um algoritmo recursivo genérico que resolve o problema P

```
 Procedimento DC-Gen ( P )
  {
      Se ( P é trivial )
       {
           S  <-- Resolve (P)
           Resorna (S)
       }

      (P1,P2)  <--  Quebra (P)

      S1  <--  DC-Gen (P1)
      S2  <--  DC-Gen (P2)

       S  <--  Combina (S1,S2)

      Retorna (S)
  }
```

Na aula de hoje, nós vamos ver que também existe uma maneira sistemática de analisar o tempo de execução de algoritmos desse tipo.

2 Analisando a árvore de recursão

Para tornar as coisas mais concretas, suponha que o procedimento `Quebra()` sempre produz dois subproblemas P1 e P2 com a metade do tamanho de P.

Então, se a primeira chamada a `DC-Gen()` passa um problema P de tamanho n, as duas chamadas recursivas realizadas por esse procedimento passam subproblemas de tamanho $n/2$.

`< Figura: árvore de recursão 2 níveis >`

E, de maneira análoga, cada chamada recursiva irá quebrar o seu problema de tamanho $n/2$ em dois subproblemas de tamanho $n/4$, e passá-los em suas respectivas chamadas recursivas.

`< Figura: árvore de recursão 3 níveis >`

Nós já sabemos que esse processo de quebras sucessivas continua até que o problema se torne trivial.

E, assumindo que isso acontece quando o problema fica de tamanho 1, nós já sabemos que a árvore terá $\log_2 n$ níveis

`< Figura: árvore de recursão log n níveis >`

Nesse ponto, nós podemos perguntar:

- Qual é o tempo de execução do algoritmo `DC-Gen`?

E a resposta é simples:

- A soma dos tempos de execução de todas as chamadas recursivas.

Sim, mas o quanto é isso?

A ideia é examinar a anatomia da árvore.

A primeira observação é que existem dois tipos de chamadas recursivas.

Vejamos.

Na base da árvore de recursão nós temos chamadas a `DC-Gen` que recebem instâncias triviais do problema.

Essas instâncias são resolvidas pelo procedimento `Resolve ()` que sempre leva tempo $O(1)$.

(porque?)

Portanto, a contribuição das folhas da árvore de recursão para o tempo de execução do algoritmo, nesse caso, é

$$n \times O(1) = O(n)$$

Todas as demais chamadas recursivas recebem problemas não triviais, e executam o seguinte trecho de código:

```
 Procedimento DC-Gen ( P )
  {
     ( . . . )

4.     (P1,P2)  <--  Quebra (P)

5.     S1  <--  DC-Gen (P1)
6.     S2  <--  DC-Gen (P2)

7.      S  <--  Combina (S1,S2)

8.     Retorna (S)
  }
```

cujo tempo de execução é determinado pelos tempos dos procedimentos `Quebra ()` e `Combina()` — isto é, desconsiderando os tempos das chamadas recursivas.

Nesse ponto, é preciso tornar as coisas concretas novamente, e especificar os tempos desses procedimentos.

Suponha primeiramente que ambos os procedimentos executam em tempo $O(1)$.

`< Figura: nós internos da árvore com tempo O(1) >`

Então, a contribuição dessa porção da árvore para o tempo total de execução do algoritmo é dada pela quantidade de nós:

$$1 + 2 + 4 + 8 + \ldots + n/2$$

onde cada nível i contribui com 2^i nós.

Na aula ?? nós vimos como calcular esse tipo de soma

`< Figura: soma de potências de 2 >`

logo, a resposta é $n - 1 = O(n)$.

Agora, suponha que os procedimentos `Quebra (P)` e `Combina (S1,S2)` executam em tempo total $O(n)$ — onde n é o tamanho do problema P recebido pela chamada recursiva.

`< Figura: nós internos da árvore com tempo O(n) >`

Nesse caso, nós podemos agrupar as chamadas que trabalham com instâncias do problema do mesmo tamanho (i.e., as chamadas do mesmo nível da árvore), e escrever

$$n \;+\; \Big(\frac{n}{2} + \frac{n}{2}\Big) \;+\; \Big(\frac{n}{4} + \frac{n}{4} + \frac{n}{4} + \frac{n}{4}\Big) \;+\; \ldots \;+\; \big(\underbrace{2 \,+\, 2 \,+\, \ldots \,+\, 2}_{n/2}\big)$$

A observação fácil aqui é que todos os grupos tem soma n, e existem $\log_2 n$ grupos (pois esse é o número de níveis da árvore)

Logo, a contribuição dessa porção da árvore para o tempo de execução do algoritmo, nesse caso, é

$$\log_2 n \;\times\; O(n) \quad = \quad O(n \log n)$$

O que dá o tempo total de execução

$$O(n) \;+\; O(n \log n) \quad = \quad O(n \log n)$$

Finalmente, suponha que os procedimentos `Quebra (P)` e `Combina (S1,S2)` executam em tempo $O(n^2)$.

Nesse caso, a contribuição das chamadas recursivas no mesmo nível da árvore também será igual, e nós podemos escrever

$$n^2 \;+\; 2 \cdot \Big(\frac{n}{2}\Big)^2 \;+\; 4 \cdot \Big(\frac{n}{4}\Big)^2 \;+\; \ldots \;+\; \frac{n}{2} \cdot \Big(\frac{n}{n/2}\Big)^2$$

E, simplificando as coisas e fazendo as contas, nós obtemos

$$\begin{aligned} & n^2 \;+\; \tfrac{n^2}{2} \;+\; \tfrac{n^2}{4} \;+\; \ldots \;+\; \tfrac{n^2}{n/2} \\ = \;\; & n^2 \cdot \Big[1 \;+\; \tfrac{1}{2} \;+\; \tfrac{1}{4} \;+\; \ldots \;+\; \tfrac{1}{n/2}\Big] \\ = \;\; & O(n^2) \end{aligned}$$

De modo que o tempo total de execução do algoritmo é

$$O(n) \;+\; O(n^2) \quad = \quad O(n^2)$$

Em princípio, esse tipo de análise sempre pode ser feito.

Mas, a seguir, nós vamos ver uma maneira sistemática de obter esse resultado.

3 O método das equações de recorrência

Examinando novamente a árvore de recursão do algoritmo `DC-Gen`, quando ele executa sobre um problema de tamanho n, nós observamos que ela contém duas cópias da árvore associada ao problema de tamanho $n/2$

`< Figura: duas cópias da subárvore >`

Ora, mas isso significa que

$$\textit{Tempo-total}\big(\texttt{DC}-\texttt{Gen(n)}\big) \;=\; 2\cdot\textit{Tempo-total}\big(\texttt{DC}-\texttt{Gen(n/2)}\big)$$
$$+\;\textit{Tempo-chamada}\big(\texttt{DC}-\texttt{Gen(n)}\big)$$

E, utilizando a seguinte notação mais compacta:

- $T_{DC}(n) \;=\; \textit{Tempo-total}\big(\texttt{DC}-\texttt{Gen(n)}\big)$
- $T_Q(n) \;=\; \textit{Tempo}\big(\texttt{Quebra(n)}\big)$
- $T_C(n) \;=\; \textit{Tempo}\big(\texttt{Combina(n)}\big)$

nós obtemos

$$T_{DC}(n) \;=\; 2\cdot T_{DC}(n/2) \;+\; T_Q(n) \;+\; T_C(n)$$

De fato, essa equação corresponde exatamente à segunda parte do procedimento `DC-Gen ()`:

```
Procedimento DC-Gen ( P )
 {
    ( . . . )

   Tq(n)   (P1,P2)  <--  Quebra (P)

Tdc(n/2)   S1  <--  DC-Gen (P1)
Tdc(n/2)   S2  <--  DC-Gen (P2)

   Tc(n)    S  <--  Combina (S1,S2)

    O(1)   Retorna (S)
 }
```

E a primeira parte do procedimento

```
Procedimento DC-Gen ( P )
 {
        |  Se ( P é trivial )
        |   {
   O(1) |       S  <-- Resolve (P)
        |       Resorna (S)
        |   }

     ( . . . )
 {
```

corresponde à seguinte equação

$$T_{DC}(1) \;=\; O(1)$$

Em resumo, o tempo de execução de uma versão do algoritmo `DC-Gen ()`, que quebra um problema de tamanho n em dois subproblemas de tamanho $n/2$, obedece as seguintes equações:

$$
\begin{aligned}
T_{DC}(n) &= 2 \cdot T_{DC}(n/2) \; + \; T_Q(n) \; + \; T_C(n) \\
T_{DC}(1) &= O(1)
\end{aligned}
$$

A grande vantagem dessa notação matemática compacta (em oposição à árvore de recursão) é que agora nós podemos analisar o tempo de execução de outras variantes do algoritmo `DC-Gen`.

Por exemplo, suponha agora que o algoritmo `DC-Gen` quebra um problema de tamanho n em dois problemas de tamanho $n/4$, e que os procedimentos `Quebra ()` e `Combina ()` executam em tempo $O(n)$.

Nesse caso, as equações que descrevem o tempo de execução do algoritmo tem a forma

$$
\begin{aligned}
T_{DC}(n) &= 2 \cdot T_{DC}(n/4) \; + \; O(n) \\
T_{DC}(1) &= O(1)
\end{aligned}
$$

Uma maneira de se obter a solução desse sistema de equações consiste em desenrolar a recursão:

$$
\begin{aligned}
T_{DC}(n) &= 2 \cdot T_{DC}(n/4) \; + \; n \\
&= 2 \cdot \left[2 \cdot T_{DC}(n/4^2) \; + \; \tfrac{n}{4} \right] \; + \; n \\
&= 2^2 \cdot T_{DC}(n/4^2) \; + \; 2 \cdot \tfrac{n}{4} \; + \; n \\
&= 2 \cdot \left[2 \cdot T_{DC}(n/4^3) \; + \; \tfrac{n}{4^2} \right] \; + \; 2 \cdot \tfrac{n}{4} \; + \; n \\
&= 2^3 \cdot T_{DC}(n/4^3) \; + \; 2^2 \cdot \tfrac{n}{4^2} \; + \; 2 \cdot \tfrac{n}{4} \; + \; n
\end{aligned}
$$

e assim por diante ...

A observação interessante, nesse ponto, é que os termos que estão aparecendo no lado direito

$$T_{DC}(n) \quad = \quad 2^3 \cdot T_{DC}(n/4^3) \ + \ \underbrace{2^2 \cdot \frac{n}{4^2} \ + \ 2 \cdot \frac{n}{4} \ + \ n}_{\star}$$

correspondem aos tempos de execução das chamadas recursivas nos primeiros 3 níveis da árvore (já agrupados).

E que o termo do lado esquerdo

$$T_{DC}(n) \quad = \quad \underbrace{2^3 \cdot T_{DC}(n/4^3)}_{\star} \ + \ 2^2 \cdot \frac{n}{4^2} \ + \ 2 \cdot \frac{n}{4} \ + \ n$$

correspondem às chamadas do próximo nível que ainda não foram analisadas.

Ou seja, a equação de recorrência para $T_{DC}(n)$ é um instrumento matemático que permite analisar o tempo de execução de uma árvore de recursão sem desenhar a árvore — o que é ótimo.

Continuando a análise, nós observamos que reduzindo o problema a 1/4 do tamanho original a cada passo, são necessários $\log_4 n$ quebras para obter um problema de tamanho 1 (trivial):

$$T_{DC}(n) \quad = \quad \underbrace{2^{\log_4 n} \cdot T_{DC}(1)}_{(1)} \ + \ \underbrace{2^{\log_4 n - 1} \cdot \frac{n}{4^{\log_4 n - 1}} \ + \ \ldots \ + \ 2 \cdot \frac{n}{4} \ + \ n}_{(2)}$$

E agora nós observamos que a parte (1) corresponde ao tempo associado às folhas da árvore de recursão, e que a parte (2) corresponde ao tempo associado aos nós internos da árvore.

O primeiro termo é bem fácil de calcular, uma vez que observamos que

$$\log_4 n \quad = \quad \frac{\log_2 n}{2}$$

(porque?)

Vejamos.

$$\begin{aligned} 2^{\log_4 n} \cdot T_{DC}(1) \quad &= \quad 2^{(\log_2 n)/2} \cdot O(1) \\ &= \quad \left(2^{\log_2 n}\right)^{1/2} \cdot O(1) \\ &= \quad \left(n\right)^{1/2} \cdot O(1) \\ &= \quad O(\sqrt{n}) \end{aligned}$$

O segundo termo parece complicado, mas a seguinte aproximação facilita as coisas:

$$\begin{aligned} (2) \quad &\leq \quad n \ + \ \tfrac{2n}{4} \ + \ \tfrac{2^2 n}{4^2} \ + \ \tfrac{2^3 n}{4^3} \ + \ \ldots \\ &= \quad n \cdot \Big[\underbrace{1 \ + \ \frac{1}{2} \ + \ \frac{1}{4} \ + \ \frac{1}{8} \ + \ \ldots}_{1}\Big] \\ &= \quad O(n) \end{aligned}$$

Portanto,

$$T_{DC}(n) \quad = \quad O(\sqrt{n}) \ + \ O(n) \quad = \quad O(n)$$

Vejamos um outro exemplo.

Suponha que o algoritmo `DC-Gen` quebra um problema de tamanho n em três problemas de tamanho $n/2$, e que os procedimentos `Quebra ()` e `Combina ()` executam em tempo $O(n)$.

As equações que descrevem o tempo de execução do algoritmo, nesse caso, são

$$\begin{aligned} T_{DC}(n) &= 3 \cdot T_{DC}(n/2) \,+\, O(n) \\ T_{DC}(1) &= O(1) \end{aligned}$$

Desenrolando a recursão, nós obtemos

$$\begin{aligned} T_{DC}(n) &= 3 \cdot T_{DC}(n/2) \,+\, n \\ &= 3 \cdot \Big[2 \cdot T_{DC}(n/2^2) \,+\, \tfrac{n}{2} \Big] \,+\, n \\ &= 3^2 \cdot T_{DC}(n/2^2) \,+\, 3 \cdot \tfrac{n}{2} \,+\, n \\ &= 3^3 \cdot T_{DC}(n/2^3) \,+\, 3^2 \cdot \tfrac{n}{2^2} \,+\, 3 \cdot \tfrac{n}{2} \,+\, n \end{aligned}$$

e por aí vai ...

Como o problema é reduzido à metade a cada passo, são necessários $\log_2 n$ passos para reduzi-lo ao tamanho 1.

$$T_{DC}(n) \;=\; \underbrace{3^{\log_2 n} \cdot T_{DC}(1)}_{(1)} \;+\; \underbrace{3^{\log_2 n - 1} \cdot \frac{n}{2^{\log_2 n - 1}} \;+\; \ldots \;+\; 3 \cdot \frac{n}{2} \;+\; n}_{(2)}$$

Nós calculamos o primeiro termo fazendo a mudança de base do logaritmo:

$$\log_3 n \;=\; \frac{\log_2 n}{\log_2 3} \qquad \Rightarrow \qquad \log_2 n \;=\; \log_3 n \,\cdot\, \log_2 3$$

Daí,

$$\begin{aligned} (1) &= 3^{\log_2 n} \cdot T_{DC}(1) \\ &= \big(3^{\log_3 n}\big)^{\log_2 3} \cdot O(1) \\ &= \big(n\big)^{\log_2 3} \cdot O(1) \\ &= O\big(n^{\log_2 3}\big) \end{aligned}$$

Para a parte (2), a aproximação que nós fizemos no exemplo anterior não funciona, pois dessa vez os termos estão crescendo.

Mas, não é difícil observar que esses termos formam uma PG

$$n \;+\; 3 \cdot \frac{n}{2} \;+\; 3^2 \cdot \frac{n}{2^2} \;+\; \ldots \;+\; 3^{\log_2 n - 1} \cdot \frac{n}{2^{\log_2 n - 1}}$$

Nesse ponto, nós podemos utilizar a fórmula para o cálculo da soma dos termos da PG.

Ou nós podemos utilizar o seguinte fato:

- a soma dos termos de uma PG (de razão diferente de 1) é da ordem de magnitude do seu maior termo

(Esse fato pode ser verificado examinando a fórmula.)

Portanto,

$$(2) \quad = \quad O\big((3/2)^{\log_2 n - 1} \cdot n\big)$$

Mas,

$$\begin{aligned}
\left(\tfrac{3}{2}\right)^{\log_2 n - 1} &= \tfrac{2}{3} \cdot \left(\tfrac{3}{2}\right)^{\log_2 n} = \tfrac{2}{3} \cdot \tfrac{3^{\log_2 n}}{2^{\log_2 n}} \\
&= \tfrac{2}{3} \cdot \tfrac{n^{\log_2 3}}{n} = \tfrac{2}{3} \cdot n^{(\log_2 3) - 1} \\
&= O(n^{(\log_2 3) - 1})
\end{aligned}$$

Logo, somando as duas partes, nós obtemos

$$T_{DC}(n) \quad = \quad O(n^{\log_2 3}) \; + \; O(n^{(\log_2 3) - 1}) \quad = \quad O(n^{\log_2 3})$$

A observação interessante aqui é que esse é um exemplo onde a soma dos tempos de resolução das instâncias triviais do problema (i.e., a parte (1) da expressão) é maior que a soma dos tempos das quebras e combinações (i.e., a parte (2) da expressão).

Nesse ponto, alguém pode observar

- *Peraí! esse negócio está ficando repetitivo ...*