Contando Iterações via Variantes de Laço

Em trechos iterativos simples de algoritmos, como num laço bem-comportado com a forma

```
"PARA i DE 1 A n   (o que é o mesmo que "PARA (i := 1; i <= n; ++i)
 | ..."                                  | ...")
```

é trivial calcular quantas iterações do laço são realizadas (por exemplo, no caso do laço acima, serão "n" iterações), e em princípio não é necessário apresentar justificativa mais detalhada.

Em casos mais elaborados, porém, um variante de laço

http://lia.ufc.br/~pmsf/2015-1/cana/arquivos/aula_correcao_de_alg_3.pdf

pode ser uma ferramenta útil para uma demonstração precisa do número de iterações realizadas num laço. Essa técnica é ilustrada a seguir.

Uma Análise Simples

Considere o caso do algoritmo de busca binária iterativo:

```
================================================================================
Algoritmo:    busca_binária
Entrada:      vetor v[1..n] e "x" (do tipo dos elementos de "v")
Pré-condição: (R1) n >= 0
              (R2) "v" ordenado segundo <=
Saída:        número natural k
Pós-condição: (S1) "v" não é alterado
              (S2) 1 <= k <= n+1
              (S3) k <= n   ==> v[k] = x
              (S4) k  = n+1 ==> v[j] != x para todo j ∈ [1..n]
--------------------------------------------------------------------------------
01. i := 1, f := n
02. ENQUANTO i <= f
03. | m := (i+f) ÷ 2   // algoritmicamente o mesmo que:  m := i + ⌊(f-i)/2⌋
04. | SE x = v[m]
05. | | RETORNE m.
06. | SE x < v[m]
07. | | f := m-1
08. | SENÃO
09. | | i := m+1
10. RETORNE n+1.
================================================================================
```

Intuitivamente, em cada iteração do laço do algoritmo, o tamanho "t" do intervalo [i..f] diminui aproximadamente pela metade (ou então o algoritmo termina), de forma que:

- Antes da iteração 1, t = n   = n ÷ 2^0
- Antes da iteração 2, t <= n ÷ 2 = n ÷ 2^1
- Antes da iteração 3, t <= n ÷ 4 = n ÷ 2^2
  ...
- Antes da iteração j, t <= n ÷ 2^(j-1).

Logo, se a iteração j chega a acontecer, então (sendo "lg x = log_2(x)"):

```
  1 <= t <= n ÷ 2^(j-1)  ==>
  2^(j-1) <= n           ==>
  j-1 <= lg n            ==>
  j <= 1 + lg n          ==>
  j <= 1 + ⌊lg n⌋  (pois j ∈ N).
```

Isso implica que o algoritmo de busca binária realiza no máximo 1 + ⌊lg n⌋ iterações.

OBSERVAÇÃO: caso desejemos obter não apenas o número de iterações realizadas pelo algoritmo, mas também um limite assintótico para o seu tempo de execução, basta observar que, como cada linha do algoritmo executa em tempo constante, então o tempo de execução de cada iteração é limitado superiormente por "a", para algum real a > 0, e portanto o tempo de execução total do algoritmo é limitado superiormente por

```
  T(n) = a*(1 + ⌊lg n⌋) + b, para alguma constante real b > 0
       = O(lg n).
```

Assim, nós concluímos que o algoritmo (sempre) executa em tempo O(lg n), e, evidentemente, também em tempo Ω(1).

 Usando Variantes da Laço

A análise acima é simples, intuitiva e bastante comum na prática da análise de algoritmos. Além disso, ela é matematicamente precisa, exceto que o fato

```
  "Antes da iteração j, t <= n ÷ 2^(j-1)"
```

foi justificado apenas de forma intuitiva (o que é evidenciado pelo "..." que precede a afirmação do fato na seção anterior). Para provar esse fato de forma precisa, essencialmente o que deve ser mostrado é que, em cada iteração, o tamanho do intervalo [i..f] diminui para a metade ou menos; daí, o fato acima segue imediatamente por indução em "j".

A seguir, o número máximo de iterações do algoritmo de busca binária é limitado superiormente por meio de variantes de laço. Na teoria, essa técnica é simplesmente uma outra roupagem para a prova por indução mencionada acima. Na prática, porém, essa técnica pode estar embutida numa ferramenta computacional de análise de algoritmos ou programas, sendo então particularmente útil.

A observação fundamental da análise é a seguinte:

OBSERVAÇÃO 1: se, num algoritmo qualquer, um laço possui uma expressão E do tipo
------------ dos números naturais como variante, então, numa execução qualquer desse laço, o valor inicial V desse variante é um limite superior para o número de iterações realizadas pelo laço nessa execução.

==========
JUSTIFICATIVA:

Observe que, pela definição de variante, o seguinte vale sobre as iterações do laço (caso elas cheguem a ser executadas):

```
  - Antes da iteração 1,    E  = V
  - Antes da iteração 2,    E <= V - 1
  - Antes da iteração 3,    E <= V - 2
  - ...
  - Antes da iteração j,    E <= V - (j-1)      (imediato por indução)
  - Ao final da iteração j, E <= V - (j-1) - 1
```

Além disso, como um variante nos números naturais nunca assume valor negativo, então, se a iteração "j" chega a ser executada, ao final dela temos:

```
  0 <= E <= V - (j-1) - 1  ==>
  j <= V,
```

o que mostra que o laço executa no máximo V iterações, como desejado.
==========

A observação acima implica que podemos obter um limite superior para o número de iterações realizadas pelo algoritmo iterativo de busca binária diretamente a partir de um variante para o laço do algoritmo. O resultado a seguir apresenta então um variante:

TEOREMA 1: f-i+1 é um variante do laço do algoritmo de busca binária.

==========
PROVA:

Temos que mostrar que E = f-i+1 "sempre" assume valores não negativos (mais precisamente, no início e no fim de cada iteração) e que diminui de valor em toda iteração. Para a primeira parte, temos:

1. Observe que, no início do laço (logo após a execução da linha 1), temos

   ```
   E  = f -i +1
      = n -1 +1
      = n
     >= 0  (pela pré-condição R1),
   ```

   como desejado.

2. Considere agora uma iteração qualquer do laço que seja completada, isto é, que não seja interrompida por execução da linha 5, e portanto ao fim da qual a condição da linha 2 seja novamente avaliada. Assim sendo, pelo texto do algoritmo, os casos possíveis são então:

   a) Nessa iteração, "i" permanece inalterada e "f" assume o valor F = m-1. Nesse caso, como, no início da iteração, vale "i <= f", então a inicialização de "m" garante que

      ```
      i   <= m        ==>
      i-1 <= m-1 = F  ==>
      0   <= F-i+1,
      ```

      o que mostra que E >= 0 ao final da iteração, como desejado.

   b) Nessa iteração, "f" permanece inalterada e "i" assume o valor I = m+1. Analogamente ao caso acima, então, a inicialização de "m" garante que

      ```
      m   <= f      ==>
      m+1 <= f+1    ==>
      0   <= f+1-I
      ```

      o que mostra que E >= 0 ao final da iteração, como desejado.

Por fim, resta mostrar que a expressão E diminui de valor em cada iteração do laço, o que é evidente, pois, em cada iteração do laço, ou "i" permanece inalterada e "f" diminui de valor ou "f" permanece inalterada e "i" aumenta de valor (tal argumento poderia ser detalhado de forma semelhante ao argumento do item 2 acima, mas nós deixamos isso como um exercício abaixo), e isso conclui a demonstração.
==========

EXERCÍCIO 1: Escreva em detalhes o último passo da demonstração acima.

Pela observação 1 e o teorema 1 acima, nós concluímos que o número de iterações realizadas pelo algoritmo de busca binária é limitado superiormente pelo valor de f-i+1 no início do laço, isto é, por n-1+1 = n, o que é, porém, muito mais do que o limite de 1 + ⌊lg n⌋ iterações obtido anteriormente. Nós podemos obter este segundo limite usando outro variante:

TEOREMA 2: g(i,f) é um variante do laço do algoritmo de busca binária, onde
----

$$g(i,f) = \begin{cases} 1 + \lfloor \lg t(i,f) \rfloor, & \text{se } t(i,f) \geq 1, \\ 0, & \text{em caso contrário, e} \end{cases}$$

$$t(i,f) = f - i + 1$$

(observe que t(i,f) é o tamanho do intervalo [i..f] do algoritmo, e que havia sido informalmente denotado por "t" na seção anterior).

(OBSERVAÇÃO: a demonstração abaixo é longa, mas a essência é a mesma já discutida anteriormente: a de que, em cada iteração do laço, o tamanho do intervalo [i..f] cai para a metade ou menos. O tamanho da demonstração se deve basicamente aos vários casos que a análise cuidadosa dos números nos leva a considerar.)

==========
PROVA:

Temos que mostrar que g(i,f) "sempre" assume valores não negativos (mais precisamente, no início e no fim de cada iteração) e que diminui de valor em toda iteração do laço.

Com relação à primeira parte, observe que, no início e no fim de cada iteração do laço, temos:

1. Se "t(i,f) >= 1", então "lg t(i,f) >= 0" e "g(i,f) >= 1 > 0, como desejado.

2. Se "t(i,f) < 1", então "g(i,f) = 0 >= 0", como desejado.

Resta, então, mostrar que g(i,f) diminui de valor em cada iteração do laço. Considere, então, uma iteração qualquer do laço que seja completada, isto é, que não seja interrompida por execução da linha 5, e portanto ao fim da qual a condição da linha 2 seja novamente avaliada. Assim sendo, observe primeiramente que, logo após a inicialização de "m" na linha 3, temos

```
  m  = (i + f) ÷ 2
     = ⌊ (i + f) / 2 ⌋            // Pela definição de "÷"
     = ⌊ (i + (i + f -i)) / 2 ⌋   // Pois "f = i + (f - i)"
     = ⌊ (2i + f - i) / 2 ⌋
     = ⌊ i + (f - i) / 2 ⌋
     = i + ⌊ (f - i) / 2 ⌋        // O que prova o comentário feito no algoritmo
    >= i.                         // Pois "i <= f"
```

De forma semelhante,

```
  m  = ⌊ (i + f) / 2 ⌋            // Ver acima
     = ⌊ (f - (f - i) + f) / 2 ⌋  // Pois "i = f - (f-i)"
     = ⌊ (2f - (f - i)) / 2 ⌋
     = ⌊ f - (f - i) / 2 ⌋        // Pois "i <= f"
    <= f.
```

Nós concluímos então que, logo após a linha 3, vale

```
  i <= m = i + ⌊(f-i)/2⌋ <= f,
```

ou seja,

```
  I <= m = I + ⌊(F0-I)/2⌋ <= F0.
```

Consideremos, agora, os casos possíveis pelo texto do algoritmo:

a) Nessa iteração, "i" permanece inalterada e "f" assume o valor m-1. Assim sendo, para evitar ambiguidades, denotemos por "F0" o valor de "f" no início da iteração e por "F" o valor de "f" ao fim da iteração; além disso, denotemos por "I" o valor de "i" durante essa iteração. Para a continuidade do argumento, é útil dividirmos a análise de acordo com os seguintes subcasos (que apenas distinguem se há ou não arredondamento no cálculo de "⌊(F0-I)/2⌋"):

   a.1) F0 - I = 2h, para algum natural h >= 0. Nesse caso, observe que, se "h = 0", então "m = I = F0" e, no início da iteração, temos

```
  t(i,f) = f -i +1
         = F0 -I +1
         = 1         ==>

  g(i,f) = 1 + ⌊lg t(i,f)⌋
         = 1 + ⌊0⌋
         = 1.
```

   Além disso, ao fim da iteração, temos

```
  t(i,f) = f -i +1
         = F -I +1
         = (I-1) -I +1  // Pois F = m-1 = I-1
         = 0            ==>

  g(i,f) = 0.
```

   Logo, se "h = 0", então g(i,f) diminui de valor durante a iteração em questão.

   Suponha agora que h != 0, isto é, que h >= 1. Temos então:

```
  F0 = I + 2h, e

  m = I + ⌊(F0-I)/2⌋
    = I + ⌊(2h)/2⌋
    = I + ⌊h⌋
    = I + h.
```

   Logo, no início da iteração, temos:

```
  t(i,f)  = f -i +1
          = F0 -I +1
          = I + 2h -I +1
          = 2h + 1
         >= 1            ==>
```

```
  g(i,f) = 1 + ⌊lg t(i,f)⌋
         = 1 + ⌊lg (2h + 1)⌋
         = 1 + ⌊lg (2 (h + 1/2) )⌋
         = 1 + ⌊( lg 2 ) + ( lg (h + 1/2) )⌋
         = 1 + ⌊1 + lg (h + 1/2)⌋
         = 1 + 1 + ⌊lg (h + 1/2)⌋.
```

Além disso, no fim da iteração, temos:

```
  t(i,f)  = f -i +1
          = F -I +1
          = I +h -1 -I +1  // Pois "F = m-1 = I +h -1"
          = h
         >= 1             ==>

  g(i,f) = 1 + ⌊lg t(i,f)⌋
         = 1 + ⌊lg h⌋
         < 1 + 1 + ⌊lg (h + 1/2)⌋,
```

o que mostra que g(i,f) diminui de valor durante a iteração em questão, como desejado.

(OBSERVAÇÃO: No caso "h != 0", observe, em particular, que o tamanho do intervalo [i..f] vale "2h+1" no início da iteração e "h" no fim da iteração, ou seja, ele diminui para a metade ou menos, o que é a essência do argumento.)

a.2) F0 - I = 2h + 1, para algum natural h >= 0. A demonstração é semelhante à do caso "a.1" e é deixada como exercício.

b) Nessa iteração, "f" permanece inalterada e "i" assume o valor m+1. A demonstração é análoga à do caso "a" e é deixada como exercício.

Nós concluímos então que g(i,f) diminui de valor em toda iteração do laço do algoritmo, e portanto que g(i,f) é um variante do laço, CQD.

==========

EXERCÍCIO 2 (IMPORTANTE): Escreva demonstrações detalhadas para os casos "a.2" e "b" da prova acima.

Pela observação 1 e o teorema 2 acima, nós concluímos então que o número de iterações realizadas pelo laço do algoritmo de busca binária é limitado superiormente pelo valor de g(i,f) no início do laço, que é igual a 0 se "n = 0" e é igual a "1 + ⌊lg n⌋" se "n >= 1", e esse era o resultado desejado.