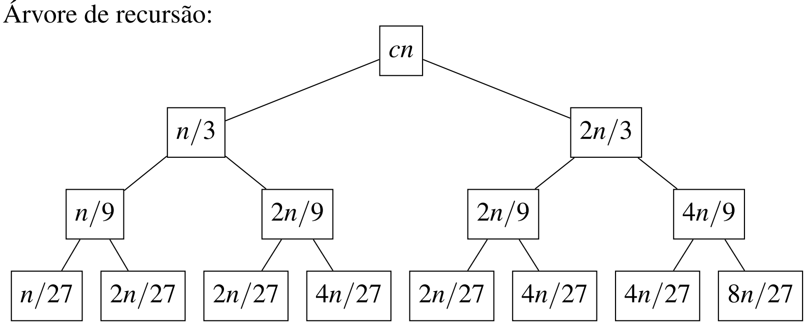
MO417 – Complexidade de Algoritmos

Nathana Facion - RA 191079 - Lista 3

29/03/2017

Exercício 1

R: Árvore de recursão:



Por nível temos:

Nível 1: cn.
Nível 2: $cn/3 + c2n/3 = cn$.
Nível 3: $cn/9 + c2n/9 + c2n/9 + c4n/9 = cn$.
Nível 4: $cn/27 + c2n/27 + c2n/27 + c4n/27 + c2n/27 + c4n/27 + c4n/27 + c8n/27 = cn$.

Conclusão:

Cada nível tem cn em sua soma e temos altura lgn. Com isso cn x lgn = cnlgn. Como dividimos por 3 em cada passo, podemos ver que esse caminho tem tamanho $\log_3 n$, então o custo do algoritmo é:

T(n) $\geq d(n \log_3 n)$

$$\geq \frac{dn}{3}\log_3 \frac{n}{3} + \frac{d2n}{3}\log_3 \frac{2n}{3} + cn$$
$$\geq \frac{dn}{3}\log_3 n - \frac{dn}{3}\log_3 3 + \frac{d2n}{3}\log_3 2n - \frac{d2n}{3}\log_3 3 + cn$$
$$\geq \frac{dn}{3}\log_3 n - dn\log_3 3 + \frac{d2n}{3}\log_3 n + \frac{d2n}{3}\log_3 2 + cn$$
$$\geq dn\log_3 n - dn + \frac{d2n}{3}\log_3 2 + cn$$

Então para funcionar basta que :

$$-dn + \frac{d2n}{3}\log_3 2 + cn \geq 0$$

Se tivermos c = d e $d \geq 1$ então:

$$\geq dn\log_3 n + \frac{d2n}{3}\log_3 2$$

Então podemos afirmar que a recorrência é $\Omega(nlgn)$.

Exercício 2

R:

Algorithm 1 Algoritmo de produto máximo

```
1: procedure SUBSEQUENCIAMAXIMA(A)
2:     if A.Comprimento == 0 then
3:         return 1, 0, 0
4:     maximo ← A[0]
5:     minimo ← A[0]
6:     resultado ← A[0]
7:     indiceFinal ← 0
8:     maxInicial ← 0
9:     minInicial ← 0
10:    for (n=1; i< A.Comprimento; i++) do
11:        if A[n] >= 1 then:
12:            if A[n] > maximo * A[n] then
13:                maxInicial ← n
14:            maximo ← max(A[n], maximo * A[n])
15:            minimo ← min(A[n], minimo * A[n])
16:        else
17:            if (A[n] < 0) then
18:                if A[n] < minimo * A[n]) then
19:                    maxInicial ← n
20:                auxiliar ← maximo
21:                maximo ← max(A[n], minimo * A[n])
22:                minimo ← min(A[n], auxiliar * A[n])
23:                aux ← maxInicial
24:                maxInicial ← minInicial
25:                minInicial ← aux
26:            else
27:                if A[n] == 0 then
28:                    maximo ← 1
29:                    minimo ← 1
30:        if maximo > resultado then
31:            indiceFinal ← n
32:        resultado ← max(resultado, maximo, 1)
33:    return max(resultado, 1), indiceFinal, maxInicial
```

Complexidade:

As linhas de 2 a 9 rodam em tempos constantes c1.
Das linhas 10 a 32 temos um bloco de for que rodará c2 x $O(n)$, sendo que n, nesse caso, simboliza o números de elementos do vetor.(no código n é o nome do índice do vetor, só para não ficar confuso)
Com isso podemos dizer que o algoritmo é $O(n)$ no pior caso e no melhor caso, já que é necessário percorrer todo o vetor para saber o melhor produto.

Correção:

Invariante: A cada iteração do for sabemos que até aquele indice j temos o melhor produto de 1 ... j.

A inicialização é realizada antes do for. Inicializamos as variáveis maximo, minimo e resultado com A[0]. Com isso a iteração no for começa com índice 1. Então, se considerarmos que o vetor tem um único elemento, não entramos no laço for e retornamos o próprio A[0]. Para caso A[0] < 1 então retornamos 1.

O maximo guarda o maior valor de sequência positiva encontrada até o momento (ele só será negativo no caso de ser o primeiro número negativo do vetor). Quando o valor de A[n] for positivo, então o seu valor aumentará multiplicando-o pelo valor na posição A[n]. Quando for negativo, essa multiplicação irá tornar o valor de máximo em um número negativo, portanto, trocamos ele de valor com a variável minimo, pois Mínimo é sempre o menor negativo, e máximo é sempre o maior positivo, logo a múltiplicação trocará seus sinais. Quando A[n] for zero, setamos o valor de Máximo para 1 pois nenhum produto maior será encontrado que use o zero, então o maior produto até aquele ponto já foi encontrada, e está guardada em resultado, e qualquer produto que comece a partir de A[n+1] não usará o 0 como membro.

O mínimo é similar ao máximo, minimo guarda o valor do menor produto negativo encontrado até o momento. Quando A[n] for positivo, o valor de minimo é simplesmente multiplicado por A[n], diminuindo ainda mais seu valor. Quando A[n] for negativo, minimo se tornará um número positivo na multiplicação, então o trocamos os valores entre máximo e mínimo, similar ao que acontece em máximo.

O resultado guarda o maior produto encontrado em toda a sequência de números. Para cada valor de A[n], os valores de máximo e mínimo são multiplicados, e sempre que máximo excede o valor de resultado, resultado recebe o valor de máximo, caso contrário, resultado não muda de valor. Assim, garantimos ter sempre o maior valor já encontrado.

O indiceFinal guarda a posição n do final da sequência de maior produto. Seu valor é sempre atualizado quando resultado recebe o valor de máximo, pois sempre que isso acontece, o valor na posição A[n] foi multiplicado com o valor anterior de resultado, o que significa que aquele é o índice final da maior sequência encontrada até o momento.

O maxInicial guarda a posição de início da maior sequência positiva encontrada. Seu valor é alterado para n em dois momentos, quando A[n] sozinho é maior que A[n] * maximo, que só acontece quando máximo é negativo (caso especial quando A[n-1] foi o primeiro negativo da sequência) o que significa que, até aquele ponto o produto será negativo, e que A[n] é o início de uma possível maior sequência a partir daquele ponto. No segundo caso, é quando A[n] é negativo e A[n] é menor que A[n] * minimo (ocorre a partir da segunda aparição de um número negativo na sequência pois mínimo é negativo e o produto dos dois será positivo). Sempre que o A[n] é negativo, maxInicial e minInicial trocam de valor pois as sequências que eles identificam também trocam, maximo e minimo.

O minInicial é semelhante ao maxInicial, guarda o início da menor sequência negativa encontrada. Seu valor, serve como um auxiliar para o maxInicial, pois ele guarda o valor de maxInicial, quando A[n] é negativo e ocorrerá a troca de valores entre máximo e mínimo

No final, retornamos resultado, maxInicial e indiceFinal, pois garantimos que estes valores estão corretos em representar o maior produto, e o início e fim da sequência que gerou esse valor.

Exercício 3

R:

Versão Recursiva O(n):

Algorithm 2 Algoritmo Máximo: O(n)

```
procedure MAXIMOVETOR(A, e, d)                      ▷ A é um vetor
    if A.Comprimento == 0 then
        return nil
    if n == 0 then
        return A[n]
    else
        return returnmax(A[n], maximoVetor(A, n − 1))
```

Versão Recursiva O($\sqrt(n)$) :

Algorithm 3 Algoritmo Máximo: O($\sqrt(n)$)

```
procedure MAXIMOVETOR(A, e, d)                      ▷ A é um vetor
    if A.Comprimento == 0 then
        return nil
    if A.Comprimento == 1 then
        return A[0]
    m ← ⌊(e + d)/2⌋
    me ← ⌊(m + e)/2⌋
    md ← ⌊(m + d)/2⌋
    if (d − e) ≤ 2 then
        maxed ← max(A[e], A[d])
        maxdm ← max(A[d], A[m])
        return max(maxed, maxdm)
    else
        if A[me] == A[md] then
            return max(maximoVetor(A, md, d), maximoVetor(A, e, me))
        if A[me] > A[md] and A[m] > A[me] then
            return maximoVetor(A, me, md)
        if A[me] > A[md] and A[m] < A[me] then
            return maximoVetor(A, e, m)
        if A[me] > A[md] and A[m] == A[me] then
            return max(maximoVetor(A, e, me), maximoVetor(A, m, md))
        if A[m] == A[md] then
            return max(maximoVetor(A, me, m), maximoVetor(A, md, d))
        if A[m] > A[md] then
            return maximoVetor(A, me, md)
        if A[m] < A[md] then
            return maximoVetor(A, m, d)
```

Complexidade:

Versão Recursiva O(n):A cada iteração nosso problema diminui em 1, ou seja, n - 1. Desta forma ao final da execução do algoritmos temos uma recorrência de T(n-1) + c1 , com isso a complexidade do algoritmo acima é O(n).

$$T(n) = \begin{cases} \theta(1), & n = 1. \\ T(n-1) + \theta(1), & n > 1. \end{cases} \tag{1}$$

$$\sum_{n=1}^{n} \theta(1) = \theta(n)$$

Versão Recursiva O($\sqrt(n)$): A cada passo da recursão estamos dividindo nosso problema, uma parte do vetor A acaba por ser descartado.No pior caso, quando chamamos max, passamos 2 vezes um quarto do vetor, Desta forma temos 2T(n/4) + um número constantes de operações . Ao resolvermos essa recorrência de 2T(n/4) + c1. Temos:

$$T(n) = \begin{cases} \theta(1), & n = 1. \\ 2T(n/4) + \theta(1), & n > 1. \end{cases} \tag{2}$$

Pelo teorema mestre a = 2 , b = 4 e f(n) = 1.
Então:
f(n) $\in \Theta$(n$^{\log_4 2-\varepsilon}$)
1 $\in \Theta$(n$^{\log_4 2-\varepsilon}$)
1 $\in \Theta$(n$^{0,5-\varepsilon}$)

Consideremos $\varepsilon = 0,5$. Com isso T(n) = $\Theta(n^{\log_4 2})$ = $\Theta(\sqrt(n)$.

Para o melhor caso temos:

$$T(n) = \begin{cases} \theta(1), & n = 1. \\ T(n/2) + \theta(1), & n > 1. \end{cases} \tag{3}$$

Pelo teorema mestre a = 1 , b = 2 e f(n) = 1.

Então:
f(n) $\in \Theta$(n$^{\log_2 1}$)
1 $\in \Theta$(n^{0})

Com isso temos $\Theta(n^{\log_2 1} lgn)$ = $\Theta(lgn)$

Correção:

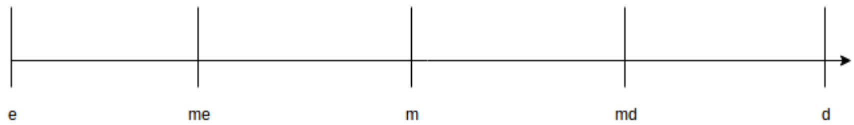
Versão Recursiva O($\sqrt(n)$):

Faremos a prova por indução.

Caso base: considere n = 1, então temos um único elemento e ele é a próprio máximo.

Hipótese: Para valores onde n - 1, conseguimos encontrar o máximo valor do vetor.

Passo: Provaremos para um vetor A de tamanho n. O algoritmo contem algumas variáveis muito importantes para encontrarmos o valor máximo, são elas: *m*(meio), *me*(meio esquerdo) e *md*(meio direito). Considere me, md e m três indices do vetor A. Por meio deles é possível identificar onde o valor máximo se encontra.



Baseado na imagem acima podemos perceber que me é a metade de m até e , similar ocorre para o lado direito. Com isso podemos dizer que $me = m/2$, $md = (m+d)/2$ e $m = n/2$. Vamos mostrar todas possíveis comparações possíveis por meio desses indices. Isso foi mostrado no código por meio de 7 ifs e abaixo queremos demonstrar que as comparações cobrem todos intervalos possíveis.

Para os casos de *md* **e** *m*, temos as seguintes comparações atingindo os seguintes intervalos:
$\Longrightarrow A[m] < A[md]$: $imaximo > m$.

$\Longrightarrow A[m] > A[md]$: $imaximo < md$
$\Longrightarrow A[m] = A[md]$: $imaximo < m$, $imaximo > md$, $A[imaximo] = A[m]$.

Para os casos de *md* **e** *me*, temos as seguintes comparações atingindo os seguintes intervalos:
$\Longrightarrow A[me] < A[md]$: $imaximo > me$.
$\Longrightarrow A[me] > A[md]$: $imaximo < md$
$\Longrightarrow A[me] = A[md]$: $imaximo < me$, $imaximo > md$, $A[imaximo] = A[me]$.

Para os casos de *m* **e** *me*, temos as seguintes comparações atingindo os seguintes intervalos:
$\Longrightarrow A[m] < A[me]$: $imaximo > m$.
$\Longrightarrow A[m] > A[me]$: $imaximo < me$
$\Longrightarrow A[m] = A[me]$: $imaximo < m$, $imaximo > me$, $A[imaximo] = A[m]$.

Como vimos todos os intervalos são cobertos com as comparações realizadas. Assim qualquer que seja o imaximo, o indice do maior elemento, poderemos encontrá-lo. A cada recursão eliminamos metade do vetor ou um quarto, e nos aproximamos mais de encontrar o imaximo. Como de acordo com o a hipótese de indução podemos encontrar o máximo do vetor para n - 1 elementos , aplicamos a hipótese. Com isso o problema é resolvido.