Universidade do Estado do Rio de Janeiro
Instituto de matemática e estatística
Curso: Ciência da computação
Disciplina: Algoritmos e Estruturas de dados II

Recursão

Autores: Wallace Vinicius, Leonardo Barcellar, Lucas Clemente

Professor: Igor Machado Coelho

Rio de janeiro
2019

Linguagem de programação utilizada: Goland

Para compilar, utilize o site: https://golang.org/

1. $\mathbf{T(n) = 3T(\frac{n}{2}) + n^2}$

   Sendo a = 3, b = 2 e c = 2, o teorema mestre é aplicável no 3º caso, onde $f(n) \in \Omega(n^c)$ e $c > \log_b(a)$:

   - $c > \log_2(3)$
   - $n^2 \in \Omega(n^2)$

   De acordo com o 3º caso $T(n) \in \theta(f(n))$ e como $f(n) = n^2$, então

   - $T(n) = \theta(n^2)$

   Segue o código com a complexidade:

```
package main

func recursao(v []int, n int){
        if(n < 2){
                return
        }
        v1 := make([]int, n/2)
        v2 := make([]int, n/2)
        recursao(v1, n/2)
        recursao(v2, n/2)
        var k int = 2

        for i := 0; i < k*n; i++{
                if(n/2 > i){
                        v1[i] = 0
                        v2[i] = 0
                }
        }
        recursao(v1, n/2)
        recursao(v2, n/2)
}
func main(){
```

```go
		v := make([]int, 12)
		v = []int{5, 8, 11, 2, 25, 17, 10, 7, 9, 28, 32, 12}
		recursao(v, 3)
}
```

2. $\mathbf{T(n) = 4T(\frac{n}{2}) + n^2}$

   Sendo a = 4 e b = 2, o teorema mestre é aplicável no 2º caso, onde $f(n) \in \theta(n^c log^k n)$ e k >= 0. Considerando k = 0 e c = 2:

   - c = $\log_2(4)$

   De acordo com o 2º caso $T(n) \in \theta(n^c log^{k+1} n)$ e como $f(n) = n^2$, então:

   - $T(n) = \theta(n^2 \log_2(n))$

   Segue o código com a complexidade:

```go
package main

import "fmt"

func recursao(v []int, n int){
	if(n < 2){
		return
	}
	v1 := make([]int, n/2)
	v2 := make([]int, n/2)
	recursao(v1, n/2)
	recursao(v2, n/2)
	var count int = 0
	for i := 0; i < n; i++{
		for j := 0; j < n; j++{
			count++
		}
	}
	fmt.Printf("%d\n", count)
	recursao(v1, n/2)
	recursao(v2, n/2)
```

```
}
func main(){
        v := make([]int, 10)
        v = []int{5, 8, 6, 7, 3, 1, 0, 2, 9, 4}
        recursao(v, 10)
}
```

3. $\mathbf{T(n) = T(\frac{n}{2}) + 2^n}$

   Sendo a = 1, b = 2 e c = 2, o teorema mestre é aplicável no 3º caso onde:

   - $f(n) \in \Omega(n^c)$
   - $c > \log_2(1)$

   De acordo com o 3º caso $T(n) \in \theta(f(n))$ e como $f(n) = 2^n$, então:

   - $T(n) = \theta(2^n)$

   Segue o código com a complexidade:

```
package main

import "fmt"
import "math"

func recursao(v []int, n int){
        if(n < 2){
                return
        }
        var nf float64 = float64(n)
        var t int = int(math.Pow(2, nf))
        V := make([]int, t)
        v1 := make([]int, n/2)
        for i := 0; i < t; i++{
                V[i] = i;
                fmt.Printf("%d ->", V[i])
        }
        recursao(v1, n)
}
```

```go
func main(){
        v := make([]int, 4)
        var n int = 4
        recursao(v, n)
}
```

4. $\mathbf{T(n) = 2^n T(\frac{n}{2}) + n^n}$

   Como a não é constante, o teorema mestre não pode ser aplicado, uma vez que o número de chamadas recursivas deve ser constantes para o teorema ser aplicado.

   Segue o código com a complexidade:

```go
package main

import "fmt"
import "math"

func recursao(v []int, n int){
        if(n < 2){
                return
        }
        var t int = int(math.Pow(float64(n), float64(n)))

        v1 := make([]int, n/2)

        var w int = int(math.Pow(float64(2), float64(n)))

        for i := 0; i < t; i++{
                v[i] = i
                fmt.Printf("%d -> ", v[i])
        }
        for i := 0; i < w; i++{
                recursao(v1, n/2)
        }
}
func main(){
        v := make([]int, 5)
```

```
        recursao(v, 2)
}
```

5. $\mathbf{T(n) = 16T(\frac{n}{4}) + n}$

   Sendo a = 16, b = 4, c = 1. O teorema mestre é aplicável no 1º caso, onde

   - $f(n) \in O(n^c)$
   - c $< \log_4(16)$

   De acordo com o 1º caso $T(n) \in \theta(n^{\log_b(a)})$, então

   - $T(n) = \theta(n^2)$

   Segue o código com a complexidade:

```
package main

func recursao(v []int, n int){
        if(n < 4){
                return
        }
        v1 := make([]int, n/4)
        v2 := make([]int, n/4)
        var count int = 0
        recursao(v1, n/4)
        recursao(v2, n/4)
        recursao(v1, n/4)
        recursao(v2, n/4)
        recursao(v1, n/4)
        recursao(v2, n/4)
        recursao(v1, n/4)
        recursao(v2, n/4)
        for i := 0; i < n; i++{
                count++
        }
        recursao(v1, n/4)
        recursao(v2, n/4)
        recursao(v1, n/4)
```

```
        recursao(v2, n/4)
        recursao(v1, n/4)
        recursao(v2, n/4)
        recursao(v1, n/4)
        recursao(v2, n/4)
}
func main(){
        v := make([]int, 10)
        v = []int{5, 8, 6, 7, 3, 1, 0, 2, 9, 4}
        recursao(v, 10)
}
```

6. $\mathbf{T(n) = 2T(\frac{n}{2}) + nlogn}$

   Sendo a = 2, b = 2, c = 1 e k = 1. O teorema mestre é aplicável no 2º caso, onde:

   - $f(n) \in \theta(n^c log^k n)$
   - $c = \log_2(2)$

   De acordo com o 2º caso, $T(n) \in \theta(n^c log^{k+1} n)$, então

   - $T(n) = \theta(nlog^2 n)$

   Segue o código com a complexidade:

```
package main

import "fmt"

func teste(v []int, n int){
        if(n < 2){
                return
        }
        v1 := make([]int, n/2)
        v2 := make([]int, n/2)

        teste(v1, n/2)
        teste(v2, n/2)
}
```

```go
func maior(v []int, n int){
	var inicio int = 0
	var fim int = n-1
	var meio int
	var maior int = v[n-1]

	for ok := true; ok; ok = (inicio <= fim){
		meio = (inicio + fim)/2
		if(maior == v[meio]){
			maior = v[meio]
			break
		}
		if(maior < v[meio]){
			fim = meio - 1
		}else{
			inicio = meio + 1
		}
	}
	fmt.Printf("Maior: %d\n", maior)
	for i := 0; i < n; i++{
		fmt.Printf("%d -> ", v[i])
	}
	fmt.Printf("\n")
}
func main(){
	v := make([]int, 10)
	v = []int{1, 3, 5, 7, 11, 13, 17, 19, 23, 29}
	maior(v, 10)
	teste(v, 10)
}
```

7. $\mathbf{T(n) = 2T(\frac{n}{2}) + \frac{n}{logn}}$

   Como $\frac{n}{\log_2{(n)}} = n(\log_2{(n)})^{-1}$. Sendo k = -1, não é possível aplicar o teorema mestre, pois $k \geq 0$:

   Segue o código com a complexidade:

```go
package main

import "fmt"
import "math"

func recursao(v []int, n int){
	if(n < 2){
		return
	}
	v1 := make([]int, n/2)
	v2 := make([]int, n/2)
	recursao(v1, n/2)
	recursao(v2, n/2)
	for i := 0; float64(i) <= float64(n)/math.Log2(float64(n)); i++{
		fmt.Printf("%d\n", i)
	}

}
func main(){
	v := make([]int, 15)
	v = []int{5, 6, 2, 9, 11, 0, 4, 18, 14, 21, 7, 3, 25, 26, 19}
	recursao(v, 15)
}
```

8. $\mathbf{T(n) = 2T(\frac{n}{4}) + n^{0,51}}$

   Sendo a = 2, b = 4 e c = 0,51, o teorema mestre pode ser aplicado no 3º caso, pois:

   - $f(n) \in \Omega(n^c)$
   - $c > \log_4(2)$

   De acordo com o 3º caso, $T(n) \in \Omega(n^c)$, então

   - $T(n) = \theta(n^{0,51})$

   Segue o código com a complexidade:

```go
package main

import "fmt"
```

```go
import "math"

func recursao(v []int, n int){
	if(n < 4){
		return
	}
	var t float64 = math.Pow(float64(n), float64(0.51))

	v1 := make([]int, n/4)
	v2 := make([]int, n/4)

	for i := 0; float64(i) < t; i++{
		v[i] = i
		fmt.Printf("%d -> ", v[i])
	}
	recursao(v1, n/4)
	recursao(v2, n/4)
}
func main(){
	v := make([]int, 10)
	recursao(v, 2)
}
```

9. $\mathbf{T(n) = 0,5T(\frac{n}{2}) + \frac{1}{n}}$

   Como a $<$ 1, o teorema mestre não pode ser aplicado, pois a seguinte condição deve ser verdadeira:

   - $T(n) = aT(\frac{n}{b} + f(n)), a >= 1, b >= 1$

   Por isso não é possível fazer o código.

10. $\mathbf{T(n) = 16T(\frac{n}{4}) + n!}$

    Sendo a = 16 e b = 4, o teorema mestre é aplicável no 3º caso, onde

    - $f(n) \in \Omega(n^c)$

    Dessa forma, a complexidade é $T(n) = \theta(n!)$

    Segue o código com a complexidade

```go
package main

import "fmt"

func recursao(v []int, n int){
	if(n < 4){
		return
	}
	v1 := make([]int, n/4)
	v2 := make([]int, n/4)

	recursao(v1, n/4)
	recursao(v1, n/4)
	recursao(v1, n/4)
	recursao(v1, n/4)
	recursao(v1, n/4)
	recursao(v1, n/4)
	recursao(v1, n/4)
	recursao(v1, n/4)

	var fat int = n
	var i int = 0

	for ok := true; ok; ok = (i < fat){
		if(n > 1){
			fat = fat*(n-1)
			fmt.Printf("fat: %d\n", fat)
			n--
		}
		i++
	}
	recursao(v2, n/4)
	recursao(v2, n/4)
	recursao(v2, n/4)
	recursao(v2, n/4)
	recursao(v2, n/4)
```

```
        recursao(v2, n/4)
        recursao(v2, n/4)
        recursao(v2, n/4)
}
func main(){
        v := make([]int, 12)
        recursao(v, 12)
}
```

11. $\mathbf{T(n) = \sqrt{2}T(\frac{n}{2}) + logn}$

    Esta função se aplica na definição do teorema mestre, porém não se aplica a nenhum dos casos.

    Segue o código com a complexidade:

```
package main

import "fmt"
import "math"

func recursao(v []int, n int){
        if(n < 2){
                return
        }
        v1 := make([]int, n/2)
        for i := 0; i < int(math.Log(float64(n))); i++{
                v[i]++
        }
        for i := 0; float64(i) < math.Sqrt(float64(2)); i++{
                recursao(v1, n/2)
        }
}
func main(){
        v := make([]int, 8)
        v = []int{1, 5, 2, 6, 8, 3, 2, 1}
        recursao(v, 8)
        fmt.Printf("%d", v)
}
```

12. $\mathbf{T(n) = 3T(\frac{n}{2}) + n}$

Sendo a = 3, b = 2 e c = 1, o teorema mestre é aplicável no 1º caso($f(n) \in O(n^c)$, e $c < \log_b(a)$), onde :

- $c < \log_2(3)$
- $n \in O(n)$

De acordo com o 1º caso, $T(n) \in \theta(n^{\log_b(a)})$ e $f(n) = n$. Portanto,

- $T(n) = \theta(n^{\log_2(3)})$

Segue o código com a complexidade

```go
package main

import "fmt"

func recursao(v []int, n int){
        if(n < 2){
                return
        }
        v1 := make([]int, n/2)
        v2 := make([]int, n/2)
        var cont int = 0
        for i := 0; i < n; i = i + 1{
                cont++;
                v[i] = cont
        }
        recursao(v1, n/2)
        recursao(v2, n/2)
        recursao(v1, n/2)
}
func main(){
        v := make([]int, 10)
        recursao(v, 10)
        fmt.Printf("%d", v)
}
```

13. $\mathbf{T(n) = 3T(\frac{n}{3}) + \sqrt{n}}$

    Sendo a = 3, b = 3 e c = 0,5, o teorema mestre é aplicável no 1º caso($f(n) \in O(n^c)$, e $c < \log_b(a)$), onde:

    - $c < \log_3(3)$
    - $\sqrt{n} \in O(\sqrt{n})$

    De acordo com o 1º caso, $T(n) \in \theta(n^{\log_b(a)})$ e $f(n) = \sqrt{n}$. Portanto,

    - $T(n) = \theta(n)$

    Segue o código com a complexidade:

```
package main

import "math"

func recursao(v []int, n int){
        if(n < 3){
                return
        }
        v1 := make([]int, n/3)
        v2 := make([]int, n/3)
        v3 := make([]int, n/3)
        var i int = 0

        for ok := true; ok; ok = (i < int(math.Sqrt(float64(n)))){
                v1[i] = 0
                v2[i] = 0
                v3[i] = 0
                i++
        }

        recursao(v1, n/3)
        recursao(v2, n/3)
        recursao(v3, n/3)
}
func main(){
```

```
        v := make([]int, 15)
        v = []int{5, 6, 2, 9, 11, 0, 4, 18, 14, 21, 7, 3, 25, 26, 19}
        recursao(v, 15)
}
```

14. $\mathbf{T(n) = 4T(\frac{n}{2}) + cn}$

Sendo a = 4, b = 2 e c = 1, o teorema mestre é aplicável no 1º caso $f(n) \in O(n^c)$, onde $c < \log_b(a)$:

- $c < \log_2(4)$
- $n \in O(n)$

De acordo com o 1º caso, $T(n) \in \theta(n \log_b(a))$ e como $f(n) = cn$. Portanto,

- $T(n) = \theta(n^2)$

Segue o código com a complexidade:

```
package main

func recursao(v []int, n int){
        if(n < 2){
                return
        }
        v1 := make([]int, n/2)
        v2 := make([]int, n/2)
        recursao(v1, n/2)
        recursao(v2, n/2)
        var k int = 2

        for i := 0; i < k*n; i++{
                if(n/2 > i){
                        v1[i] = 0
                        v2[i] = 0
                }
        }
        recursao(v1, n/2)
        recursao(v2, n/2)
```

```
}
func main(){
        v := make([]int, 12)
        v = []int{5, 8, 11, 2, 25, 17, 10, 7, 9, 28, 32, 12}
        recursao(v, 3)
}
```

15. $\mathbf{T(n) = 3T(\frac{n}{4}) + nlogn}$

    Sendo a = 3, b = 4, c = 1, k = 1 e $\log_4(3) \approx 0,79$

    - $c > \log_4(3)$

    Como $f(n) \in \Omega(n^c)$. Dessa forma, o teorema mestre é aplicável no 3º caso. Logo,

    - $T(n) = \theta(nlogn)$

    Segue o código com a complexidade:

```
package main

import "fmt"

func recursao(v []int, n int){
        if(n < 4){
                return
        }
        v1 := make([]int, n/4)
        v2 := make([]int, n/4)
        v3 := make([]int, n/4)

        recursao(v1, n/4)
        recursao(v2, n/4)
        recursao(v3, n/4)
}
func maior(v []int, n int){
        var inicio int = 0
        var fim int = n - 1
        var meio int
```

```go
        var maior = v[n-1]

        for ok := true; ok; ok = (inicio <= fim){
                meio = (inicio + fim)/2
                if(maior == v[meio]){
                        maior = v[meio]
                        break
                }
                if(maior < v[meio]){
                        fim = meio - 1
                }else{
                        inicio = meio + 1
                }
        }
        fmt.Printf("Maior: %d\n", maior)
        for i := 0; i < n; i++{
                fmt.Printf("%d -> ", v[i])
        }
        fmt.Printf("\n")
}
func main(){
        v := make([]int, 16)
        v = []int{1, 3, 5, 7, 11, 13, 17, 19, 23, 29, 30, 34, 35, 40, 41, 45}
        maior(v, 16)
        recursao(v, 16)
}
```

16. $\mathbf{T(n) = 3T(\frac{n}{3}) + \frac{n}{2}}$

    Sendo a = 3, b = 3, c = 1, k = 0 e $\log_3(3) = 1$

    - $c = \log_3(3)$

    Aplicando o 2º caso:

    - $f(n) \in \theta(n^c log^k n)$

    Dessa forma, não se aplica o teorema mestre, já que

    - $f(n) = \frac{n}{2} \notin \theta(n)$

Segue o código com a complexidade:

```go
package main

import "fmt"

func recursao(v []int, n int){
	if(n < 3){
		return
	}

	v1 := make([]int, n/3)
	v2 := make([]int, n/3)
	v3 := make([]int, n/3)
	var cont int = 0
	for i := 0; i < n; i = i + 2{
		cont++
		v[i] = i
	}
	fmt.Printf("%d\n", cont)
	recursao(v1, n/3)
	recursao(v2, n/3)
	recursao(v3, n/3)
}
func main(){
	v := make([]int, 15)
	recursao(v, 15)
}
```

17. $\mathbf{T(n) = 6T(\frac{n}{3}) + n^2 logn}$

Sendo a = 6, b = 3, c = 2, o teorema mestre é aplicável no 3º caso($f(n) \in \Omega(n^c)$), onde $c > \log_b(a)$

- $c > \log_3(6)$
- $n^2logn \in \Omega(n^c)$

Como $T(n) = \theta(f(n))$. Então,

- $T(n) = \theta(n^2 logn)$

Segue o código com a complexidade:

```go
package main

import "fmt"
import "math"

func recursao(v []int, n int){
	if(n < 3){
		return
	}
	v1 := make([]int, n/3)
	v2 := make([]int, n/3)
	v3 := make([]int, n/3)
	var cont int = 0
	for i := 0; i < n; i++{
		for j := 0; i < n; j++{
			v[i] = cont
			v[j] = cont
			cont++
			for k := 0; float64(k) < (math.Log(float64(n))); k++{
				v[j] = v[j] + 5
			}
		}
	}
	recursao(v1, n/3)
	recursao(v2, n/3)
	recursao(v3, n/3)
}
func main(){
	v := make([]int, 15)
	recursao(v, 15)
	for i := 0; i < 15; i++{
		fmt.Printf("%d -> ", v[i])
	}
```

```
}
```

18. $\mathbf{T(n) = 4T(\frac{n}{2}) + \frac{n}{logn}}$

    Sendo a = 4, b = 2, c = 1.

    Como $f(n) = \frac{n}{logn} = nlog^{-1}n$, ou seja, k = -1. K deve ser ≥ 1, o teorema mestre não se aplica.

    Segue o código com a complexidade:

```go
package main

import "fmt"
import "math"

func recursao(v []int, n int){
        if(n < 2){
                return
        }
        v1 := make([]int, n/2)
        v2 := make([]int, n/2)
        recursao(v1, n/2)
        recursao(v2, n/2)
        for i := 0; float64(i) <= float64(n)/math.Log2(float64(n)); i++{
                fmt.Printf("%d\n", i)
        }
        recursao(v1, n/2)
        recursao(v2, n/2)
}
func main(){
        v := make([]int, 15)
        v = []int{5, 6, 2, 9, 11, 0, 4, 18, 14, 21, 7, 3, 25, 26, 19}
        recursao(v, 15)
}
```

19. $\mathbf{T(n) = 64T(\frac{n}{8}) - n^2logn}$

    Não se aplica ao teorema mestre, pois $f(n)$ é uma função negativa. Por esse mesmo motivo não é possível fazer o código.

20. $\mathbf{T(n) = 7T(\frac{n}{3}) + n^2}$

Como a = 7, b = 3 e c = 2, o teorema mestre é aplicável no 3º caso, onde

- $f(n) \in \Omega(n^c)$
- $c > \log_3(7)$

Então, de acordo com o 3º caso, $T(n) \in \Omega(n^c)$, logo:

- $T(n) = \theta(n^2)$

Segue o código com a complexidade:

```go
package main

import "fmt"

func recursao(v []int, n int){
	if(n < 3){
		return
	}
	v1 := make([]int, n/3)
	v2 := make([]int, n/3)
	v3 := make([]int, n/3)
	var cont int = 0
	for i := 0; i < n; i++{
		for j := 0; j < n; j++{
			cont++
			v[i] = cont
		}
	}
	recursao(v1, n/3)
	recursao(v2, n/3)
	recursao(v3, n/3)
	recursao(v1, n/3)
	recursao(v2, n/3)
	recursao(v3, n/3)
	recursao(v1, n/3)
```

```
}
func main(){
        v := make([]int, 27)
        recursao(v, 15)
        fmt.Printf("%d", v)
}
```

21. $\mathbf{T(n) = 4T(\frac{n}{2}) + logn}$

    Esta recorrência não se aplica em nenhum dos casos, mesmo se aplicando na definição do teorema mestre.

    Segue o código com a complexidade:

```
package main

import "fmt"
import "math"

func recursao(v []int, n int){
        if(n < 2){
                return
        }
        v1 := make([]int, n/2)
        v2 := make([]int, n/2)
        recursao(v1, n/2)
        recursao(v2, n/2)
        for i := 0; i < int(math.Log(float64(n))); i++{
                v[i]++
        }
        recursao(v1, n/2)
        recursao(v2, n/2)
}
func main(){
        v := make([]int, 8)
        v = []int{1, 5, 2, 6, 8, 3, 2, 1}
        recursao(v, 8)
        fmt.Printf("%d", v)
}
```

22. $\mathbf{T(n) = 4T(\frac{n}{2}) + n(2 - \cos(n))}$

Não se aplica no teorema mestre, pois $f(n)$ é uma função trigonométrica que não apresenta regularidade. Pelo mesmo motivo não conseguimos desenvolver o código.

Questão Especial

Retirada do site: https://www.urionlinejudge.com.br/judge/pt/problems/view/1029

Recorrência: $T(n) = T(n-1) + T(n-2)$

Complexidade: $T(n) = O(2^n)$

Código:

```go
package main

import "fmt"

var qtd int = 0

func fibonacci(n int) int{
    if n <= 1{
            return n
        }else{
            qtd += 2
            return fibonacci(n-1) + fibonacci(n-2)
        }
}
func main(){
        var n int
        fmt.Printf("Digite um numero qualquer: ")
        fmt.Scanf("%d", &n)
        fmt.Printf("%d° numero da sequencia: %d\n", n, fibonacci(n))
        fmt.Printf("Quantidade de chamadas recursivas: %d", qtd)
}
```