**Análise de Algoritmos**

****

|  |  |  |  |
| --- | --- | --- | --- |
| **Digitado por:** | **Professor:** | **Versão** | **Ano** |
| Ricardo Kim | André Perin | 1.0 | 2017 |

**ÍNDICE**

[1. Apresentação e Ementa 3](#_Toc484985237)

[2. Introdução a análise de algoritmos 4](#_Toc484985238)

[3. Complexidade de algoritmos 5](#_Toc484985239)

[3.1. Exemplos 8](#_Toc484985240)

[3.1.1. Exemplo 1 8](#_Toc484985241)

[3.1.2. Exemplo 2 8](#_Toc484985242)

[3.1.3. Exemplo 3 8](#_Toc484985243)

[3.2. Exercícios 9](#_Toc484985244)

[3.3. Respostas 9](#_Toc484985245)

[3.4. Importância da taxa de crescimento 10](#_Toc484985246)

[4. Notação Big-O 11](#_Toc484985247)

[4.1. Recursão 12](#_Toc484985248)

[4.1.1. Exemplo clássico 12](#_Toc484985249)

[4.1.2. Definição recursiva 12](#_Toc484985250)

[4.1.3. Recursão linear 12](#_Toc484985251)

[4.1.4. Exemplo de recursão linear 12](#_Toc484985252)

[4.2. exercícios 13](#_Toc484985253)

[5. Recursão binária 15](#_Toc484985254)

[5.1. Exemplo: 15](#_Toc484985255)

# Apresentação e Ementa

* Contato: [prof.andreperin@usjt.br](mailto:prof.andreperin@usjt.br)
* Livros:
  + Estrutura de dados e algoritmos em Java, goodrich m.j. 5ªed.
* Materia
  + Apresentação/Introdução
  + Medida de complexidade de algoritmos
  + Analise assíntona
  + Notação big O
  + Calculo de complexidade de estruturas de dados elementares/complexas
  + Calculo de complexidade de estruturas Complexas/Algoritmo interativos
  + Busca de dados em dicionários
  + Hashing
  + Skid List
  + Árvores
    - Binárias
    - AVL
  + Grafos
    - Representação
    - Busca

# Introdução a análise de algoritmos

* Projetos orientados a objeto
  + Etapas
    - Projeto
    - Codificação
    - Teste e depuração
  + Classes
  + Atributos
  + Métodos
  + Objetivos
    - Robustez
    - Adaptabilidade
    - Re usabilidade
  + Princípios
    - Abstração
    - Encapsulamento
    - Modularidade
  + Heranças
    - Especialização
    - Extensão
  + Polimorfismo
    - Extensão de uma classe para outra
      * Ex:
        + Classe triangulo extende para classe polígono
* Pseudo códigos
  + Expressões:
    - Inteiro A
    - A 🡨 3
    - A = 3
  + Laços:
    - Enquanto (while):
      * Enquanto (condição)  
         instrução  
        Fim Enquanto
    - Repete (do while)
      * Repete (instrução){  
        enquanto(condição)
    - Para (for)
      * Para a 🡨1 até a <10 faça a 🡨 a+1  
        instrução  
        Fim Para
  + Indexação de vetores
    - Inteiro a[i]
    - Inteiro a[i,j]
    - Inteiro a[i,j,k]
  + Chamada de métodos
    - A 🡨 método (x)
  + Retorno de métodos
    - Retorna y
  + Comentários
    - {comentario}

# Complexidade de algoritmos

* Introdução
  + Estrutura de dados
  + Algoritmos
* Analise de algoritmo
  + Caracterização
  + Relação
* Tempo de execução
  + O tempo de execução tipicamente cresce em função do tamanho da entrada.
  + Pode variar
    - Melhor dos casos
    - Média dos casos
    - Pior dos casos
* Estudos experimentais
  + Implementar o algoritmo
  + Executar o programa com entradas diferentes (tamanho de valores)
  + Obter medida do tempo
  + Colocar os resultados em gráficos
* Limitações
  + Implementação
  + Resultados podem conter ruídos
  + Manter ambientes de Hardware e Software idênticos
* Análise Teórica
  + Utiliza descrição de alto nível
  + Caracteriza em função ao tamanho de entrada
  + Considera todas as entradas possíveis
  + Permite independência de Hardware e Software
* Pseudo código na analise de algoritmos
  + Descrição de alto níve
  + Estruturado
  + Menos detalhado que um programa
  + Oculta problemas de projeto e implementação
* Modelo Random Access Machine (RAM)
  + Células de memória potencialmente ilimitadas
  + Células numeradas com acesso a qualquer célula em tempo indeterminado
* Funções utilizadas na análise de algoritmos
  + Constante
  + Logarítma
  + Linear
  + N Log N
  + Quadratica
  + Cúbica
  + Exponencial
* Operações Básicas Desempenhadas em algoritmo
  + Identificados no pseudo-código
  + Altamente independente da linguagem de programação
  + A definição exata não é tão importante
  + Tem como premissa considerar os tempos constantes no modelo RAM
* Exemplo
  + Avaliação de uma expressão numérica
  + Atribuição de valor de uma variável
  + Indexação de uma array
  + Chamada de método
  + Retorno de método
* Contagem de operações primitivas

Algoritmo arrayMax (A, n)

CurrentMax 🡨 A[0]

Para i 🡨 1 até n-1 com i 🡨 i+1 faça

Se A[i] > currentMax Então

currentMax 🡨 A[i]

Fim Se

Fim Para

Retorna currentMax

Algoritmo 🡪

## Exemplos

### Exemplo 1

ALGORITMO ProcuraMatriz ( E )

i <- 0

ENQUANTO i < n FAÇA

J <- 0

ENQUANTO j < m FAÇA

SE A [ i, j ] = E ENTÃO

p [ 0 ] <- i

p [ 1 ] <- j

i <- n

j <- m

FIM SE

j <- j + 1

FIM ENQUANTO

i <- i + 1

FIM ENQUANTO

RETORNA P

### Exemplo 2

ALGORITMO Prog1 ( n )

cont <- 0

i <- 1

ENQUANTO ( i <= n ) FAÇA

i <- i + 2

cont <- cont + 1

FIM ENQUANTO

RETORNA cont

### Exemplo 3

ALGORITMO Prog2(n)

cont <- 0

i <- 1

ENQUANTO (i<=N) FAÇA

j <- 1

ENQUANTO (i<=n) FAÇA

j <- j\*2

cont <- cont+1

FIM ENQUANTO

i <- i+1

FIM ENQUANTO

RETORNA cont

## Exercícios

1. Descreva um pseudocódigo, o algoritmo convTemp(int c) que converte uma temperatura dada em °C para °F. Determine o esforço computacional desse algoritmo.
2. Descreva um algoritmo para inserir um elemento em uma determinada posição de uma lista indexada. Determine o esforço computacional desse algoritmo.
3. Dada uma Matriz A[X,Y] Descreva um algoritmo que verifique o conteúdo dessa matriz e retorno a posição do elemento verificado.

## Respostas

1. float convTemp(int c)  
    retorna 1.8 \* c + 32   
   1. insereNaLista( vetor, posição, valor)

i <- 0

ENQUANTO vetor[i] != 0 FAÇA

i <- i +1

FIM ENQUANTO

j <- 0

ENQUANTO i > j FAÇA

A[j] <- A[j-1]

j <- j -1

FIM ENQUANTO

* 1. removeElemento(A,i)

e <- A[i] →2

ENQUANTO A[i] != 0 FAÇA

A[i] <- A[i+1]

i <- i+ 1

FIM ENQUANTO

RETORNA e

1. procuraMatriz(e)

i <- 0

j <- 0

ENQUANTO A[i][j] != e FAÇA

ENQUANTO A[i][j] != e FAÇA

j <- j +1

FIM ENQUANTO

i <- i + 1

FIM ENQUANTO

R[0] <- I

R[1] <- j

RETORNA R

## Importância da taxa de crescimento

|  |  |  |  |
| --- | --- | --- | --- |
| Tempo de Execução | Tempo para | Tempo para | Tempo para |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

A taxa de crescimento não é afetada por termos de ordem inferior ou por fatores constantes.  
Ex:

[Falta plotar gráfico]

# Notação Big-O

Dada as funções e , dezemos que é se existem constante positivas e no tal que:

Ex:

[Falta plotar gráfico]

Regras:

Se for um polinômio de grau , então é , isto é:

* Desprezar os termos de menor ordem
* Desprezar os fatores constantes
* Usar a menor classe de função possível
* Usar a expressão mais simples da classe
* Análise assíntona de algoritmos
  + Determina o tempo de execução em notação Big-O
  + Deve-se encontrar o pior caso em função da entradas
  + Expressar essa função em notação Big-O

## Recursão

Quando um método chama a sí próprio

### Exemplo clássico

A função fatorial

### Definição recursiva

### Recursão linear

Teste para casos base

1. Começar testando um conjunto de casos base (deve haver pelo menos um)
2. Cada cadeira de chamadas recursivas deve chegar a um caso base,e o tratamento de cada caso não deve usar recursão

### Exemplo de recursão linear

Um algoritmo que calcula a soma dos n primeiros inteiros de uma matriz A

ALGORITMO SomaLinear (A,n)

SE n = 1 ENTÃO => 1

RETORNA A[0] => 2

SENÃO

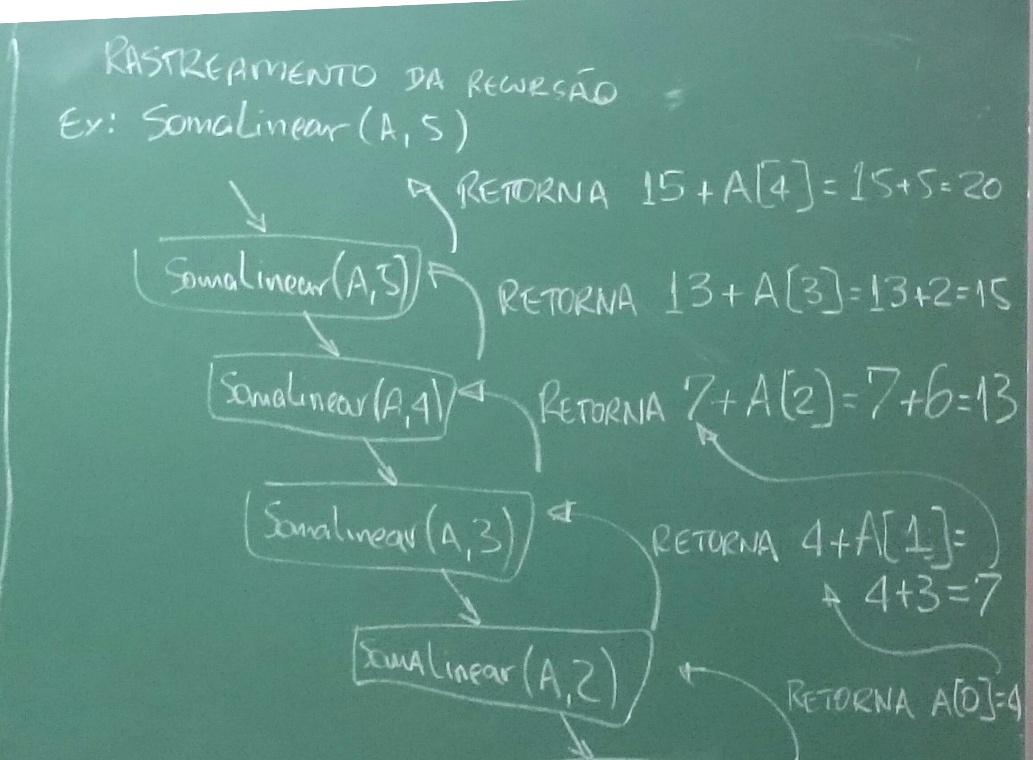
RETORNA SomaLinear (A, n-1) + A[n-1] => (n-1) +4

FIM SE

Operações:

Rastreamento da recursão

Ex: soma linear (A,5)



## exercícios

1. Descrever um algoritmo recursivo em pseudocódigo para inverter uma matriz A[i,j].calcular a função de esforço computacional e a equivalente na notação Big-O

ALGORITMO iMatriz(A,I,j)

SE i < j ENTÃO

tmp <= A[i,j]

A[i,j] <= A[j,i]

A[j,i] <=tmp

RETORNA iMatriz(A,i+1,j-1)

FIM SE

1. Cálulo de potencias, descrever um algoritmo recursivo para calcular a potência de um número

Dado:

ALGORITMO Power (x,n)

SE n = 0 ENTÃO **=> 2**

RETORNA 1 **=> 2**

SENÃO

RETORNA x \* power (x,n-1) **=> 4**

FIM SE

Execução de: O(n)

1. Quadrados recursivos, um algoritmo mais eficiente

Dado:

ALGORITMO Power (x,n)

SE n = 0 ENTÃO **=> 2**

RETORNA 1 **=> 1**

FIM SE

SE n é impar ENTÃO **=>2**

y <- Power (x, (n-1)/2) **=>4n/2**

RETORNA x \*y \* y **=> 3**

SE NÃO

y <- Power (n, n/2) **=>3n/2**

RETORNA y \* y **=> 2**

FIM SE

A= 2 +1

B = 2N +3

C = 2N +2

|  |  |
| --- | --- |
| n | F(n) |
| 0 | A = 3 |
| 1 | A+B = 8 |
| 2 | A+B+C = 12 |
| 3 | A+2B+C = 17 |
| 4 | A+2B+2C = 21 |
| 5 | A+3B+2C = 26 |
| 6 | A+3B+3C = 30 |

Comparar os dois algoritmos e calcular o tempo de execução para cada caso

# Recursão binária

## Exemplo:

Série de fibonnacci

Cálculo da série de fibonacci

ALGORITMO

SE k = 1 ENTÃO

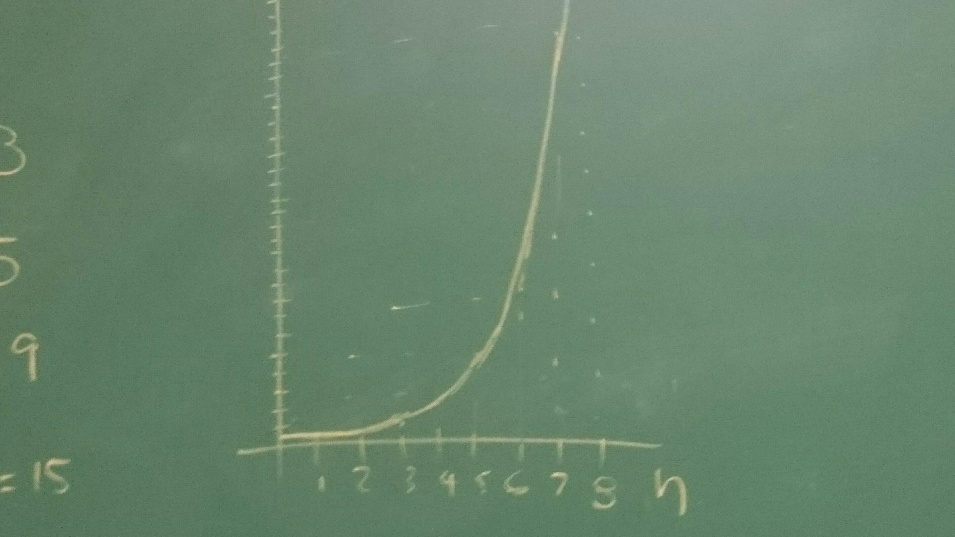
RETORNA k

SE NÃO

RETORNA BinaryFib(k-1) + BinaryFib(k-2)

FIM SE

Seja o número de chamadas recursivas por BinaryFib(k)



**Pilha**

Implementação baseada em vetor métodos:

* Size
* Push
* Pop

Uma variável rastreia o índica do topo da pilha

Se o armazenamento ultrapassar o liite do vetor deve ser lançada uma exceção (FullStackException)

Algorimo:

ALGORITMO Size()

Retorna t+1;

ALGORITMO Pop()

SE isEmpty() ENTAO

Lança EmptyStackException

SENÃO

T <- t-1

Retorna S[t+1]

FIM SE

ALGORITMO Push(o)

SE t = S.length -1 ENTAO

Lança FullStackException

SENÃO

T <- t+1

S[t] <-0

FIM SE