



**Ciências  
ULisboa**

Faculdade  
de Ciências  
da Universidade  
de Lisboa

# **ALGORITMOS E ESTRUTURAS DE DADOS**



**2017/2018**

## 1º ANO - -

### Sem. : 1º

|       |   |  |    |    |      |   |
|-------|---|--|----|----|------|---|
| 26748 |  | <u>Arquiteturas de Sistemas Computacionais</u> | 1º | S1 | Obr. | 6 |
| 13538 |  | <u>Cálculo</u>                                 | 1º | S1 | Obr. | 6 |
| 26722 |  | <u>Introdução à Programação</u>                | 1º | S1 | Obr. | 6 |
| 13539 |  | <u>Lógica de Primeira Ordem</u>                | 1º | S1 | Obr. | 6 |
| 26759 |  | <u>Produção de Documentos Técnicos</u>         | 1º | S1 | Obr. | 3 |
| 9437  |   | <u>OPÇÃO - FCSE</u>                            | 1º | S1 | Opc. | 3 |

### Sem. : 2º

|       |   |   |    |    |      |   |
|-------|---|---|----|----|------|---|
| 2672  |    | <u>Algoritmos e Estruturas de Dados</u>           | 1º | S2 | Obr. | 6 |
| 13540 |   | <u>Elementos de Álgebra Linear</u>                | 1º | S2 | Obr. | 6 |
| 34706 |   | <u>Física A</u>                                   | 1º | S2 | Obr. | 6 |
| 22701 |  | <u>Introdução às Probabilidades e Estatística</u> | 1º | S2 | Obr. | 6 |
| 26724 |   | <u>Laboratórios de Programação</u>                | 1º | S2 | Obr. | 6 |

# ALGORITMOS



# Como determinar o número de pessoas numa sala de aula?

---



# Como determinar o número de pessoas num anfiteatro?

---

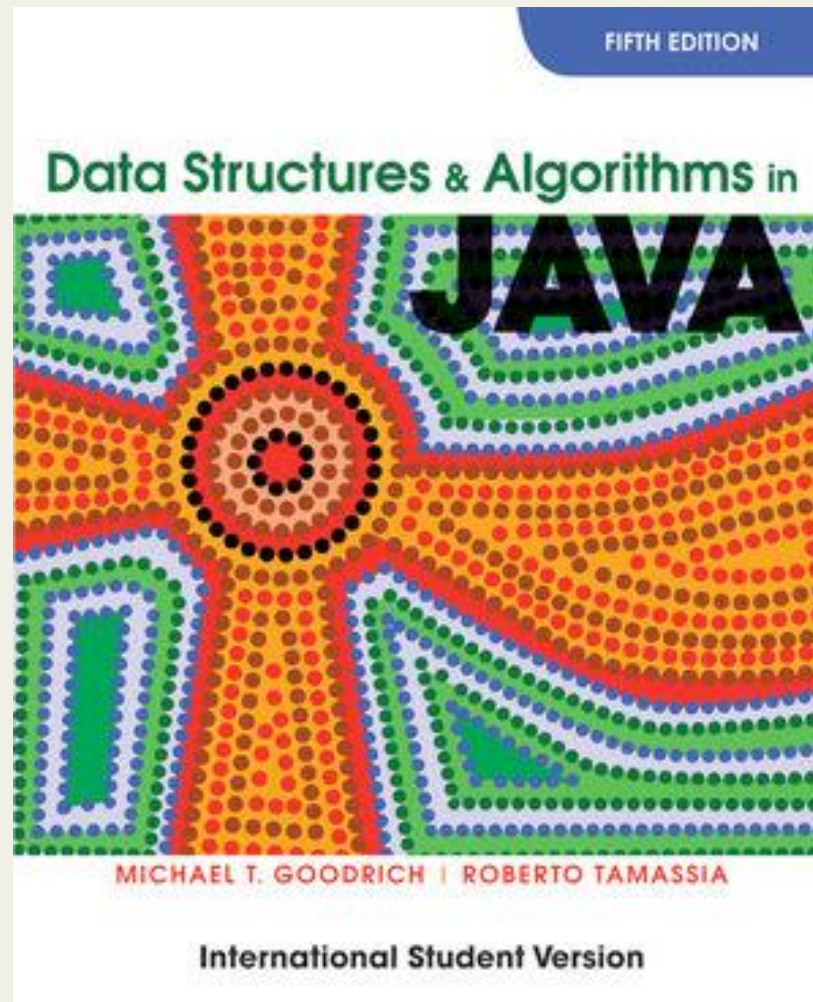
1. Levanta-te!
2. Pensa para ti: “Sou o número 1”
3. Forma um par com alguém ainda em pé. Somem os vossos números. O resultado da soma passa a ser o vosso número.
4. Um dos elementos do par senta-se.
5. Se ainda estás em pé, volta ao passo 3.

# ESTRUTURAS DE DADOS



# Como facilitar a procura de um *conceito* num livro de estudo?

---





# Como facilitar a procura de um conceito num livro de estudo?

## Index

abstract data type, viii, 60  
 array list, 234–235  
 deque, 223  
 dictionary, 420–421  
 graph, 594–599  
 list, 244–247  
 map, 382–385  
 ordered map, 403  
 partition, 539–542  
 priority queue, 334–338  
 queue, 214–216  
 sequence, 264  
 set, 534–542  
 stack, 199–200  
 string, 8–9, 554–556  
 tree, 284–285  
 abstraction, 60  
 $(a, b)$  tree, 679–681  
     depth property, 679  
     size property, 679  
 acyclic, 622  
 adaptability, 58, 59  
 adaptable priority queue, 368  
 adapter, 235  
 Adel'son-Vel'skii, 500  
 adjacency list, 600, 603  
 adjacency matrix, 600, 605  
 adjacent, 595  
 ADT, *see* abstract data type  
 Aggarwal, 686  
 Aho, 232, 278, 500, 591  
 Ahuja, 662  
 algorithm, 166  
 algorithm analysis, 166–184  
     average case, 169–170  
     worst case, 170  
 alphabet, 8, 555  
 amortization, 241–242, 539–542  
 ancestor, 282, 621  
 antisymmetric, 335  
 API, 80, 200  
 arc, 594  
 Archimedes, 166, 196  
 Ariadne, 607  
 arithmetic, 21  
 Arnold, 56  
 array, 34–38, 96–116  
     capacity, 35  
     length, 34  
 array list, 234–242, 404  
     abstract data type, 234–235  
     implementation, 236–242  
 associative stores, 382  
 asymmetric, 595  
 asymptotic analysis, 174–184  
 asymptotic notation, 170–174  
     big-Oh, 171–173, 176–184  
     big-Omega, 174  
     big-Theta, 174  
 attribute, 611  
 autoboxing, 26  
 AVL tree, 443–454  
     balance factor, 451  
     height-balance property, 443  
 back edge, 609, 625, 626, 656  
 Baeza-Yates, 500, 552, 592, 686  
 bag, 429  
 balance factor, 451  
 balanced search tree, 468  
 Barůvka, 660, 662  
 base class, 63  
 base type, 5, 11, 19  
 Bayer, 500, 686  
 Bentley, 379, 430  
 best-fit algorithm, 669  
 BFS, *see* breadth-first search  
 biconnected graph, 659

big-Oh notation, 171–173, 176–184  
 big-Omega notation, 174  
 big-Theta notation, 174  
 binary recursion, 146  
 binary search, 315, 404–407  
 binary search tree, 432–439  
     insertion, 435–436  
     removal, 436  
     rotation, 447  
     trinode restructuring, 447  
 binary tree, 296–309, 503  
     array-list representation, 310–312  
     complete, 347, 349–354  
     full, 296  
     improper, 296  
     left child, 296  
     level, 299  
     linked structure, 301–309  
     proper, 296  
     right child, 296  
 binomial expansion, 688  
 bipartite graph, 660  
 bit vector, 548  
 blocking, 674  
 Booch, 94, 278  
 bootstrapping, 467  
 Boyer, 591  
 Brassard, 196  
 breadth-first search, 619–621, 626  
 brute force, 564  
 brute-force pattern matching, 564  
 B-tree, 681  
 bubble-sort, 277  
 bucket array, 386  
 bucket-sort, 529–530  
 Budd, 94, 278  
 buffer, 39  
 buffer overflow attack, 35  
 Burger, 686

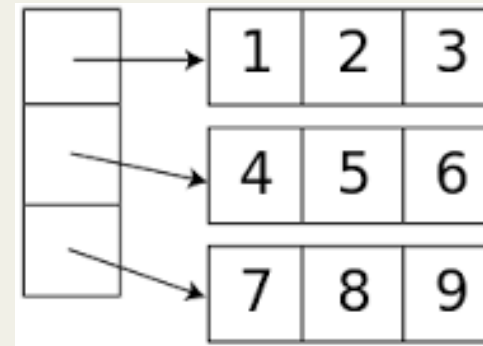
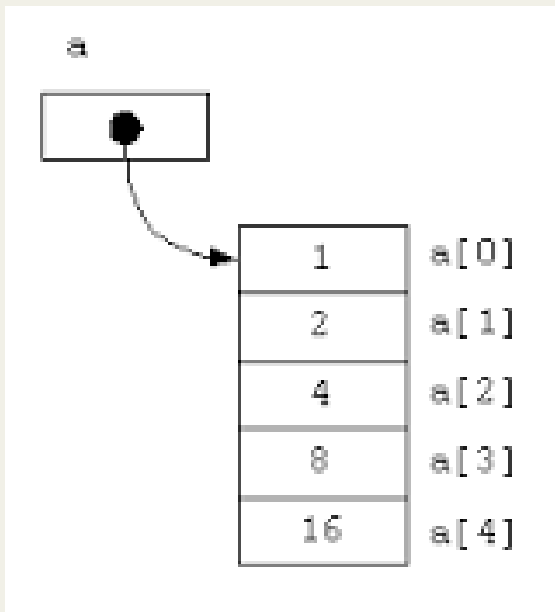
casting, 85–88  
     implicit, 26  
 catch, 78  
 ceiling function, 165  
 cell, 34  
 character-jump heuristic, 566  
 Chernoff bound, 549, 692  
 child, 281  
 children, 281  
 ciphertext, 109  
 circularly linked list, 130, 276  
 Clarkson, 552  
 class, 2–13, 58, 60  
 class inheritance diagram, 63  
 clustering, 396  
 coding, 47  
 Cole, 591  
 collection, 278  
 collision resolution, 387, 393–397  
 Comer, 686  
 comparator, 336, 337  
 complete binary tree, 347, 349–354  
 Complete Binary Tree Property, 347  
 complete graph, 656  
 composition pattern, 336  
 compound object, 16, 116  
 compression function, 387, 392  
 concatenation, 9, 23  
 conditional probability, 691  
 connected components, 598, 610, 621  
 constant function, 158  
 constructor, 17, 68  
 constructor chaining, 67  
 contradiction, 185  
 contrapositive, 185  
 control flow, 27–33  
 core memory, 672  
 Cormen, 500, 662  
 Cornell, 56

## Index





Como armazenar e organizar dados em memória, para os usar de forma eficiente?



# Importância

---

*“The difference between a bad programmer and a good one is whether [the programmer] considers code or data structures more important. Bad programmers worry about the code. Good programmers worry about data structures and their relationships.”*

*— Linus Torvalds (creator of Linux)*

*“Algorithms + Data Structures = Programs.” — Niklaus Wirth*

build, systems,  
and generating  
treatment of basic  
and dynamic data  
structures, sorting,  
recursion algorithms,  
language structures,  
and compiling

NIKLAUS WIRTH

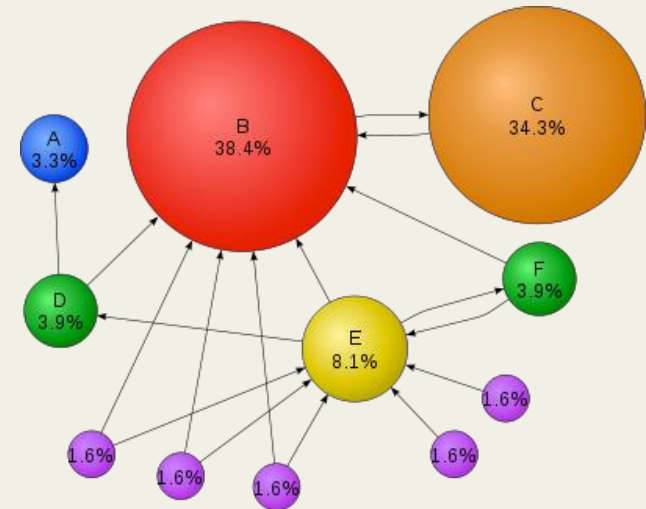
Algorithms +  
Data  
Structures =  
Programs

PRINTED ON  
RECYCLED PAPER  
WITH AN  
AUTOMATIC  
DIGITALIZATION

# Importância

Algoritmos e estruturas de dados têm um impacto cada vez maior em diferentes domínios.

- **Web**
  - encaminhamento de pacotes na rede
  - partilha distribuída de ficheiros
  - pesquisas, recomendações, publicidade
- **Multimedia**
  - mp3, jpg, reconhecimento de músicas
- **Segurança**
  - encriptação de dados, sistemas de votação
- **Finança**
  - Venda e compra de acções nas bolsas
- **Biologia**
  - Projecto do genoma humano
  - *Protein folding*



## Algorithms Replacing Wall Street Analysts, Investors



Published: Monday, 29 Apr 2013 | 4:29 PM ET

Text S

## The Secret of Airbnb's Pricing Algorithm

The sharing economy needs machine intelligence to set prices

You Will Be Hired By An Algorithm For Your Next Job

BY AASHISH SHARMA ON JUNE 29, 2015 – IN APPS & SOFTWARE / SCIENCE / TECH

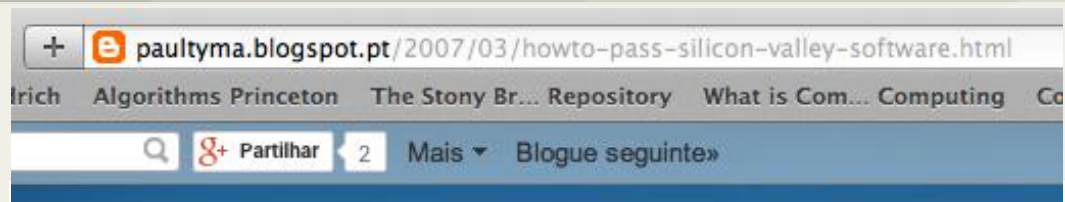
## Transplant algorithms to save lives

Published tisdag 10 mars 2015 kl 09.16



**Ciências**  
ULisboa | **Informática**

# Importância



## Howto Pass a Silicon Valley Software Engineering Interview

I do a fair bit of interviewing. This probably averages about 2 to 3 interviews per week mostly for Java developers. I'm also giving a birds-of-a-feather at the Software Developer's conference tonight in the Santa Clara Hvatt bar on just this topic (7:30-9:00pm). If you've written plenty of code, you should be familiar with when to use what data structures and to know their runtime characteristics. You should know that a hashtable's worst case search time is linear - and you should have an idea how to avoid it. And why you might use a binary tree instead of a hashtable even though it has an  $O(\log n)$  lookup. And that  $O(1)$  is effectively the same as  $O(100)$ . Surely the subtleties are situation dependent - but that's why you understand it - to apply it in the right situation.

**This is all datastruct and algorithms 101. I perpetually hear developers tell me they learned that stuff in school but now forgot it. Personally, I wonder what the hell**

have been coding? If you've just been gluing APIs together then that's nice, but it's not very interesting. Even if you don't interact with them directly, knowing data structures and algorithms is key to understanding performance. This is not premature optimization - this is choosing the right tool for the job. And that choice is often wonderfully subtle.

# Importância

## Tech Prep Tips

The best tip is: go get a computer science degree. The more computer science you have, the better. You don't have to have a CS degree, but it helps. It doesn't have to be an advanced degree, but that helps too.

However, you're probably thinking of applying to Google a little sooner than 2 to 8 years from now, so here are some shorter-term tips for you.

**Algorithm Complexity:** you need to know Big-O. It's a must. If you struggle with basic big-O complexity analysis, then you are almost guaranteed not to get hired. It's, like, one chapter in the beginning of one theory of computation book, so just go read it. You can do it.

**Sorting:** know how to sort. Don't do bubble-sort. You should know the details of at least one  $n \cdot \log(n)$  sorting algorithm, preferably two (say, quicksort and merge sort). Merge sort can be highly useful in situations where quicksort is impractical, so take a look at it.

For God's sake, don't try sorting a linked list during the interview.

**Hashtables:** hashtables are arguably the single most important data structure known to mankind. You *absolutely have to know how they work*. Again, it's like one chapter in one data structures book, so just go read about them. You should be able to implement one using only arrays in your favorite language, in about the space of one interview.

**Trees:** you should know about trees. I'm tellin' ya: this is basic stuff, and it's embarrassing to bring it up, but some of you out there don't know basic tree construction, traversal and manipulation algorithms. You should be familiar with binary trees, n-ary trees, and trie-trees at the very *very* least. Trees are probably the best source of practice problems for your long-term warmup exercises.



# Objectivos: SABER

---

Pretende-se que o aluno fique a conhecer

- **conceitos** fundamentais e **técnicas** básicas para desenhar e estudar **algoritmos** e **estruturas de dados**
  - **algoritmos**: métodos apropriados para resolver problemas de forma computacional
  - **estruturas de dados**: esquemas de organização dos dados, usualmente em memória (de forma a poder ser usado de forma eficiente)
- importantes algoritmos, tipos e estruturas de dados, úteis na implementação de sistemas de software



# Objectivos: FAZER

---

Pretende-se que o aluno seja capaz de

- analisar algoritmos (iterativos ou recursivos)
- fazer uma adequada estruturação de dados no contexto do paradigma orientado a objectos
- comparar a adequação de diferentes estruturas de dados existentes para um dado problema
- implementar diferentes estruturas de dados

# Tópicos

---

| <b>Análise Algoritmos</b>     | Complexidade assintótica, no melhor caso, pior caso e caso esperado  |
|-------------------------------|--|
| <b>Recursão</b>               | Dividir para conquistar, Recursão linear, binária, múltipla, <i>Tail recursion</i> , Memorização, Sistemas de recorrência            |
| <b>Tipos Dados Abstractos</b> | Filas, Pilhas, Listas, Conjuntos, Árvores, Filas com Prioridade, Mapas, Dicionários  |
| <b>Estruturas de Dados</b>    | Listas e outras estruturas ligadas, Vectores circulares, Tabelas de dispersão, Árvores Binárias de Pesquisa, Amontoados, Árvores AVL |
| <b>Ordenação</b>              | Insert sort, Selection sort, Merge sort, Heap sort, Quick sort, Radix sorts  |

# Avaliação

---

- **Avaliação contínua** (cotado para 20 valores, **0% ou 10%**)
  - Exercícios a entregar nas aulas TPs
- **Projeto** (cotado para 20 valores, **15%**)
  - Grupos de dois ou três alunos
  - Entrega a 20 de Maio
- **Exame** (cotado para 20 valores, **85% ou 75%**)
  - Prova individual escrita de 3h (sem consulta)

# Avaliação

---

Os alunos que desejem **não fazer a avaliação contínua** têm de enviar um email ao responsável da disciplina **até ao dia 26 de Fevereiro**.

## Condição de Aprovação

### ☒ Avaliação contínua

$$a.aprovado() \Leftrightarrow a.nota\_exame() \geq 9.5 \ \&\&$$

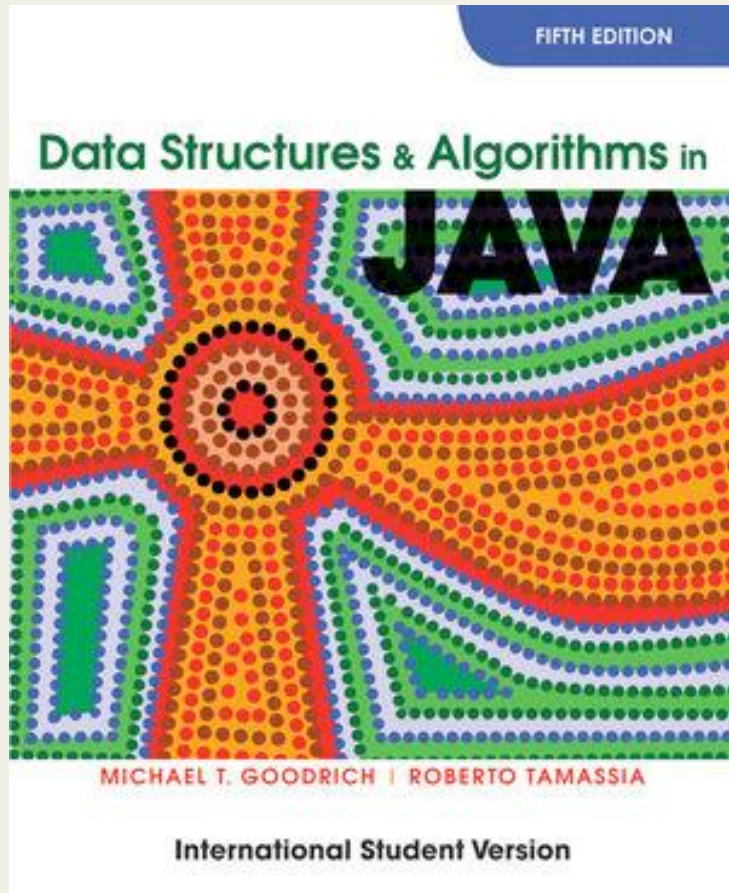
$$0.10*a.notaAvC() + 0.15*a.notaProj() + 0.75*a.notaExame() \geq 9.5$$

### ☐ Avaliação contínua

$$a.aprovado() \Leftrightarrow a.nota\_exame() \geq 9.5 \ \&\&$$

$$0.15*a.notaProj() + 0.85*a.notaExame() \geq 9.5$$

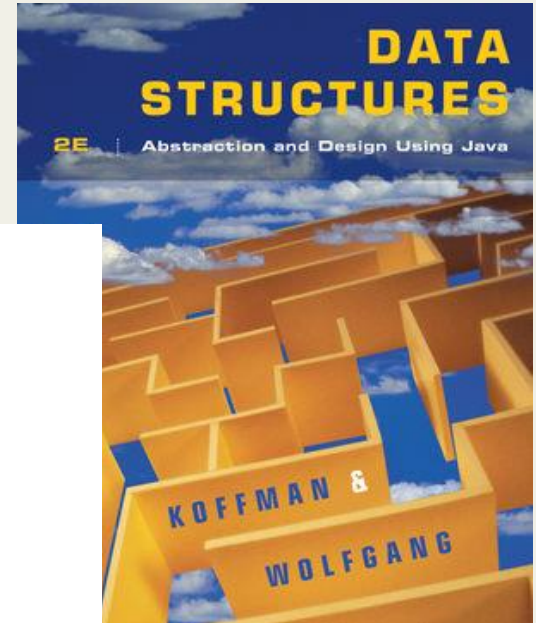
# Bibliografia



## Specifying and Monitoring Java Classes with CONGU 2.0

V. T. Vasconcelos, A. Lopes and I. Nunes

Lecture Notes  
**Algoritmos e Estruturas de Dados**  
Departamento de Informática  
Faculdade de Ciências da Universidade de Lisboa  
March 2014



**Ciências**  
ULisboa | Informática

# Outros Recursos



← → ↺ 🏠 ⓘ 🔒 <https://www.di.fc.ul.pt/~mal/aed/AED ADTs.html>

## Os tipos de dados de AED

[Pilhas](#) | [Filas](#) | [Dequeues](#) | [Árvores Binárias](#) | [Listas](#) | [Multi-conjuntos](#) | [Conjuntos ordenados](#)

### Pilhas

- [Especificação](#)
- [Implementação](#) (com vector que cresce)
- [Refinamento](#)
- [StackRandomTest](#)

### Filas

- [Especificação](#)
- [Implementação](#) (com lista simplesmente ligada, a completar)
- [Refinamento](#)
- [QueueRandomTest](#)



# Bibliografia



Ciências  
ULisboa  
Faculdade  
de Ciências  
da Universidade  
de Lisboa

moodle

Dashboard ► Licenciatura ► Engenharia Informática ► Algoritmos e Estruturas de Dados (26723)

## Referências principais

- M. T. Goodrich e R. Tamassia, "[Data Structures and Algorithms in Java](#)", John Wiley & Sons, ISBN978-0-470-93439-5, 2011. (Há uma 6ª edição de 2014).
- V.Vasconcelos, A.Lopes and I.Nunes, [Specifying and Monitoring Java Classes with ConGu 2.0](#), Lecture Notes, 2014.

## Referências adicionais

- E. Koffman e P. Wolfgang, "[Data Structures: Abstraction and Design Using Java, 2nd edition](#)", Wiley, ISBN 978-0-470-12870-1, 2011 Exemplos, soluções de exercícios e outros recursos podem ser encontrados no "[Student Companion site](#)" deste livro.
- A. Vermeulen et al., "[The Elements of Java Style](#)", Cambridge University Press, ISBN 9780521777681, 2000.
- J. Bloch, "[Effective Java, 2nd edition](#)", Addison-Wesley, ISBN 0321356683, 2008

## • Sobre o ConGu

- Ferramenta integrada no plugin QUEST para o Eclipse, disponível [aqui](#)
- Pequeno tutorial sobre a utilização da ferramenta [aqui](#)
- A bancada com especificações dos tipos de dados estudados em AED e algumas das suas implementações, disponível [aqui](#)



Ciências  
ULisboa | Informática

## *Modus Operandis*

---

- **Aulas Teóricas:**
  - Servem para **enquadrar e motivar** os assuntos, problemas e suas soluções, esclarecendo os aspectos mais complexos e dando ênfase a questões mais subtis.
  - Preparam para uma **compreensão mais facilitada** da matéria, mas não substituem a aquisição da matéria através do estudo individual, a qual tem ser consolidada pelo estudo cuidado do livro.

## *Modus Operandis*

---

- **Aulas Teórico-práticas:**
  - Servem para **apoiar** a aquisição e consolidação de conhecimentos através da prática de resolução de exercícios e problemas, mas não são suficientes para se dominar a matéria, a qual deve ser consolidada através do estudo do livro e da resolução de exercícios adicionais.
  - São centradas na **resolução de exercícios** das séries de exercícios da disciplina
  - Será durante as aulas TPs que serão enunciados os **exercícios de avaliação contínua** que os alunos de cada turma têm que realizar na aula/fazer em casa e entregar na semana seguinte

## Alguns dados: as presenças nas TPs

| Nota AvC | #aprovados | #reprovados |
|----------|------------|-------------|
| 5        | 17         | 4           |
| 4        | 20         | 10          |
| 3        | 12         | 11          |
| 2        | 7          | 24          |
| 1        | 1          | 4           |
| 0        | 8          | 12          |
|          |            |             |
|          |            |             |
| Nota AvC | %aprovados | %reprovados |
| $\geq 4$ | 57%        | 22%         |
| $\leq 2$ | 25%        | 62%         |

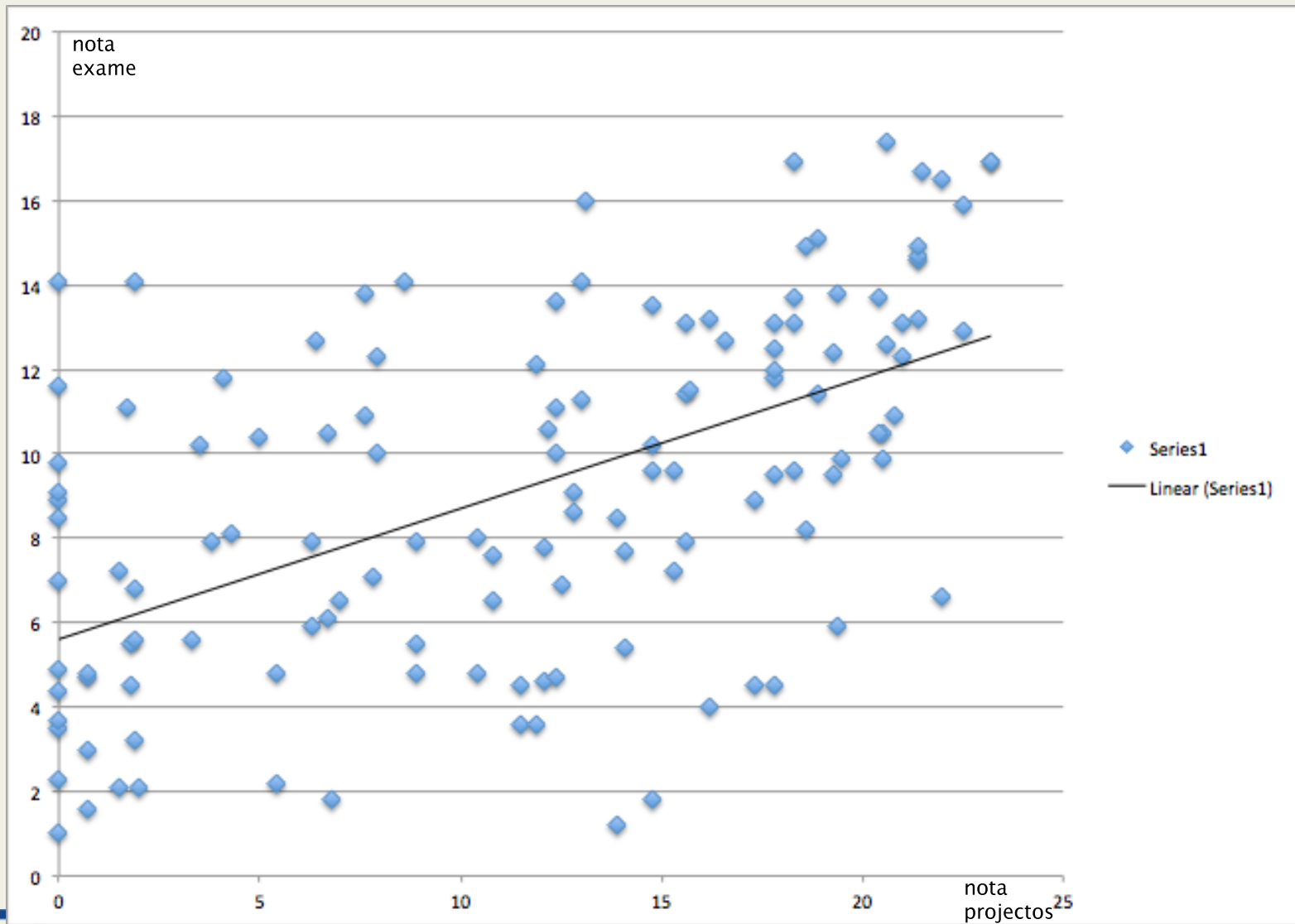
#aprovados



#reprovados



# Alguns dados: as notas dos projetos



# Projetos vs Exames (~ pergunta de 3 valores)

```
/** Reference to the left child. */
private Node<E> left;

/** Reference to the right child. */
private Node<E> right;

/** Construct a node with given data and no children.
 * @param data The data to store in this node
 * @requires data!=null
 */
private Node(E data) {
    this(data, null, null);
}

/** Construct a node with given data and two children.
 * @param data The data to store in this node
 * @param left The left child
 * @param right The right child
 * @requires data!=null
 */
private Node(E data, Node<E> left, Node<E> right) {
    this.data = data;
    this.left = left;
    this.right = right;
}
}
```

```
/** The nodes of the list of digits */
private static class Node {
    /** invariant digit >= 0 && digit <= 9;
     * private byte digit;
     * private Node next;

    /** requires d >= 0 && d <= 9;
     * private Node(int d) {
     *     this.digit = (byte) d;
     *     this.next = null;
     * }
    }

    /**
     * Constructs a BigInteger from a non-negative integer
     * @param num the integer
     * @requires num >= 0
     */
    public BigInteger(int num){
        least = null; //faked
    }

    /**requires node != null
     * private BigInteger(Node node) {
     *     least = node;
     * }

    /**
     * @return the successor of this BigInteger
     */
    public BigInteger suc(){ //TOCOMPLETE
    }
```

2. Considere o excerto dado de uma classe *BigInteger* cujos objetos permitem representar números naturais maiores do que os suportados pelos tipos primitivos *built-in* do Java. A classe está implementada recorrendo a uma lista ligada de dígitos, com o dígito menos significativo no início.

Implemente o método *suc* e caracterize o seu tempo de execução em termos do número natural representado por *this*. Note que a implementação deste método **NÃO EXIGE** a implementação do método que faz a soma de dois *BigInteger*.



# O vosso esforço

---

- **AED = 6 créditos ECTS**
  - European Credit Transfer and Accumulation System
- O volume de trabalho para realizar este curso com **sucesso** corresponde a aproximadamente **170 horas de trabalho** por parte do estudante.
  - $168 \text{ horas} / 17 \text{ semanas} = 10 \text{ horas/semana}$
  - 1/3 correspondem ao tempo da participação nas aulas
  - ou seja, **fora das aulas têm de dedicar 7 horas/semana**

# 7 h/semana?

---




## Fora das aulas

---

- **Estudo** do livro e outros elementos da bibliografia
- **Programar** todas as classes abordadas nas aulas
  - escrever o código
  - compilar
  - testar
- Fazer os **exercícios** e **projetos** propostos
- Não usar código escrito por outros a menos que o mesmo seja autorizado e referir sempre a fonte
  - trabalho com colegas é encorajado
  - verificar as soluções também
  - copiar código escrito por outros não...

# Comunicação

- **Docentes -> Alunos**
  - informação no moodle de ciências
  - fórum de notícias
- **Alunos -> Docentes**
  - aulas teóricas e práticas
  - horários de dúvidas
  - fórum de dúvidas no moodle



Minha página principal ▶ As minhas disciplinas ▶ Ano lectivo 2014 - 2015 ▶ Departamento de Informática ▶ 1º Ciclo ▶ 1º Ano  
▶ Algoritmos e Estruturas de Dados ▶ Geral ▶ Fórum Notícias

NAVEGAÇÃO

Minha página principal


- Página inicial do site
- ▶ Páginas do site
- ▶ Meu perfil
- ▼ Disciplina atual
  - ▼ Algoritmos e Estruturas de Dados
    - ▶ Participantes
    - ▶ Medalhas

Fórum Notícias

PROCURAR NOS FÓRUMS

NOTÍCIAS GERAIS E AVISOS

CRIAR UM NOVO TÓPICO

| Tópico                  | Iniciado por  | Respostas | Última mensagem                          |
|-------------------------|---|-----------|--|
| Início das Aulas de AED |  Antónia Lopes | 0         | Antónia Lopes<br>Qui, 12 Feb 2015, 18:22 |