Estruturas de Dados I

Árvores B e B+

Igor Machado Coelho

04/11/2026

1. Árvores de Busca Multivias
2. Árvores B
3. Árvores B+
4. Considerações Finais
5. Agradecimentos

Section 1

Árvores de Busca Multivias

Pré-Requisitos

São requisitos para essa aula:

- Introdução/Fundamentos de Programação (em alguma linguagem de programação)
- Interesse em aprender C/C++
- Aula de Árvores de Busca (ABB)
- Aula de Dicionários Ordenados e Árvores Balanceadas (AVL e Rubro-Negra)
- Tópico avançado de Sacos Ordenados (busca binária com `buscaMaiorIgual`)

Agradecimentos especiais à prof. Karina Mochetti (IC-UFF), cujo material sobre Árvores B inspirou esses slides.

Motivação: Dados no Disco

Bancos de dados e sistemas de arquivos guardam bilhões de chaves em **memória secundária** (disco, SSD), onde cada acesso é **milhares de vezes** mais lento que na memória principal.

Em uma AVL com $n = 10^9$ chaves, a altura pode chegar a $1{,}44 \lg n \approx 43$: até **43 acessos ao disco** por busca!

Ideia: o disco lê um **bloco** inteiro de uma vez (ex: 4 KiB). E se cada nó guardasse **muitas** chaves, ocupando um bloco inteiro?

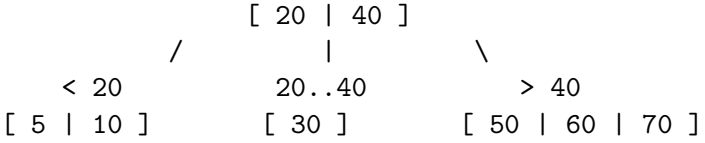
- cada acesso traz centenas de chaves, e não apenas uma
- a árvore fica **muito mais baixa**: com 500 a 1000 filhos por nó, bastam **4 níveis** para 10^9 chaves!

Para isso, precisamos de árvores cujos nós tenham **mais de dois filhos**.

Árvore de Busca Multivias

Uma *Árvore de Busca Multivias* (ABM) é uma árvore em que cada nó guarda $k \geq 1$ chaves **ordenadas** $c_0 < c_1 < \ldots < c_{k-1}$ e possui $k+1$ filhos $F_0, F_1, \ldots, F_k$ (possivelmente vazios), tais que:

- todas as chaves em F_0 são menores que c_0
- todas as chaves em F_i estão **entre** c_{i-1} e c_i
- todas as chaves em F_k são maiores que c_{k-1}



Ou seja, cada chave de um nó **divide a busca** em partes: k chaves dividem o espaço em $k+1$ intervalos, um para cada filho.

Observação: a ABB é exatamente uma ABM com $k = 1$ (uma chave, dois filhos)!

Busca em uma Árvore de Busca Multivias

Para buscar a chave c em um nó, precisamos achar **em qual intervalo** ela está. Isso é exatamente o `buscaMaiorIgual` do Saco Ordenado: a posição p da **primeira chave** $\geq c$!

- se $c_p = c$: **encontrou**
- senão, c está entre c_{p-1} e c_p: desce para o filho F_p

```
busca(30):     [ 20 | 40 ]        buscaMaiorIgual(30) = 1 (chave 40)
                     |             40 != 30, desce para F_1
                [ 30 ]             buscaMaiorIgual(30) = 0 (chave 30): enconti

busca(45):     [ 20 | 40 ]        buscaMaiorIgual(45) = 2 (passou do fim)
                          \         desce para F_2
               [ 50 | 60 | 70 ]    buscaMaiorIgual(45) = 0 (chave 50)
                                    50 != 45, desce para F_0: vazio, não encor
```

Como as chaves do nó estão ordenadas, `buscaMaiorIgual` usa **busca binária**: $O(\log k)$ comparações por nó.

Section 2

Árvores B

Árvore B

Uma *Árvore B* de **ordem** m é uma Árvore de Busca Multivias em que:

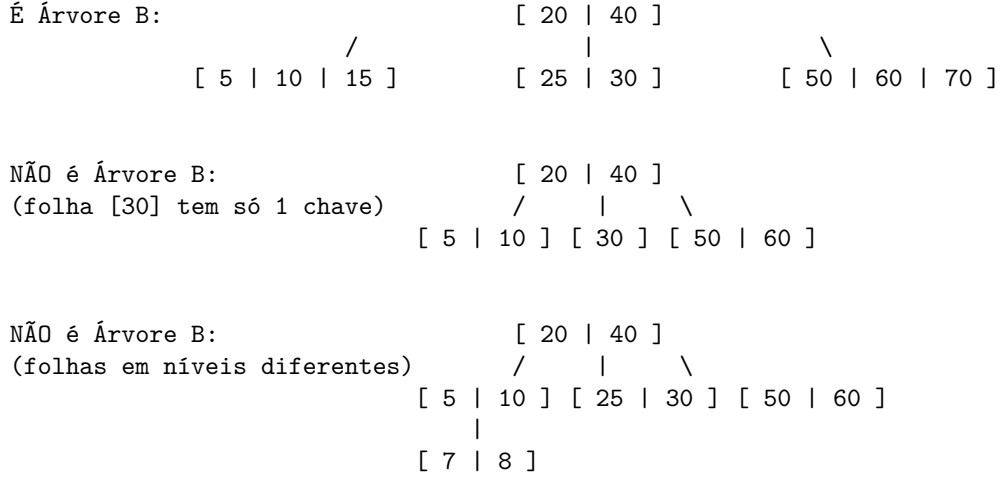
1. todo nó tem **no máximo** m filhos (ou seja, no máximo $m-1$ chaves)
2. todo nó, exceto a raiz, tem **no mínimo** $\lceil m/2 \rceil$ filhos (ou seja, no mínimo $\lceil m/2 \rceil - 1$ chaves)
3. a raiz, se não for folha, tem no mínimo **2** filhos
4. **todas as folhas estão no mesmo nível**

Ou seja, a aridade de cada nó **varia** entre $\lceil m/2 \rceil$ e m (cada nó fica pelo menos "meio cheio"), e a árvore é sempre **perfeitamente balanceada**: não existem rotações nem fator de balanceamento!

Ordem m	Filhos por nó	Chaves por nó	Nome
3	2 a 3	1 a 2	árvore 2-3
4	2 a 4	1 a 3	árvore 2-3-4
5	3 a 5	2 a 4	
1000	500 a 1000	499 a 999	(típica em disco)

Exemplos: Árvore B de Ordem 5

Ordem 5: cada nó tem de 2 a 4 chaves (a raiz pode ter 1), e todas as folhas no mesmo nível.



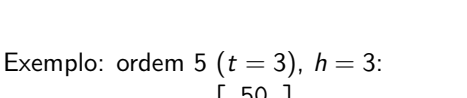
Exercício: a terceira árvore é uma Árvore de Busca Multivias? Quais regras da Árvore B ela

Altura de uma Árvore B

Seja $t = \lceil m/2 \rceil$. Qual o **menor** número de chaves em uma Árvore B de ordem m com altura h (raiz no nível 1)?

Nível	Nós (mínimo)	Chaves por nó
1 (raiz)	1	1
2	2	$t-1$
3	$2t$	$t-1$
$i \geq 2$	$2t^{i-2}$	$t-1$

Exemplo: ordem 5 ($t = 3$), $h = 3$:



$1 + 2 \cdot 2 + 6 \cdot 2 = 17$ chaves

$$n \geq 1 + \sum_{i=2}^{h} 2t^{i-2}(t-1) = 1 + 2(t-1) \underbrace{\sum_{i=2}^{h} t^{i-2}}_{\frac{t^{h-1}-1}{t-1}} = 2t^{h-1} - 1 \quad \Rightarrow \quad h \leq 1 + \log_t \frac{n+1}{2}$$

Ou seja, $h = O(\log_m n)$. Com $m = 1000$ e $n = 10^9$: altura no máximo **4** (contra ≈ 43 na AVL)!

Estrutura de uma Árvore B

Cada nó guarda um vetor de chaves e um vetor de filhos. Usamos **uma posição a mais** em cada vetor, para que um nó possa **estourar** temporariamente durante a inserção:

```
constexpr int M = 5;     // ordem
struct NoB {
   int k;                // número de chaves
   int chaves[M];        // M-1 + 1 extra
   NoB* filhos[M + 1];   // M   + 1 extra
};
bool folha(NoB* no) {
   return no->filhos[0] == 0;
}
struct ArvoreB {
   NoB* raiz;
   void cria() { raiz = 0; }
   // ... tem, adiciona, ...
};
```

```
k = 3:
chaves: | 10 | 20 | 30 |    |
filhos: F0   F1   F2   F3   -
```

As chaves de `filhos[i]` ficam entre `chaves[i-1]` e `chaves[i]`.

Em uma folha, todos os filhos são vazios.

Para um **Dicionário**, basta um vetor `valores[M]` paralelo às chaves.

Busca em Árvore B

A busca desce **um único caminho** da raiz até, no máximo, uma folha:

```
// primeira posição com chave >= c
int buscaMaiorIgual(NoB* no, int c) {
   int ini = 0, fim = no->k;
   while (ini < fim) {
      int meio = (ini + fim) / 2;
      if (no->chaves[meio] < c)
         ini = meio + 1;
      else
         fim = meio;
   }
   return ini;
}
```

```
bool tem(NoB* no, int c) {
   while (no) {
      int p = buscaMaiorIgual(no, c);
      if (p < no->k &&
          no->chaves[p] == c)
         return true;       // achou!
      no = no->filhos[p];   // desce
   }
   return false;
}
```

Custo: $O(\log m)$ comparações por nó, em $O(\log_m n)$ níveis: $O(\log m \cdot \log_m n) = O(\log n)$ comparações, mas apenas $O(\log_m n)$ **páginas** lidas do disco!

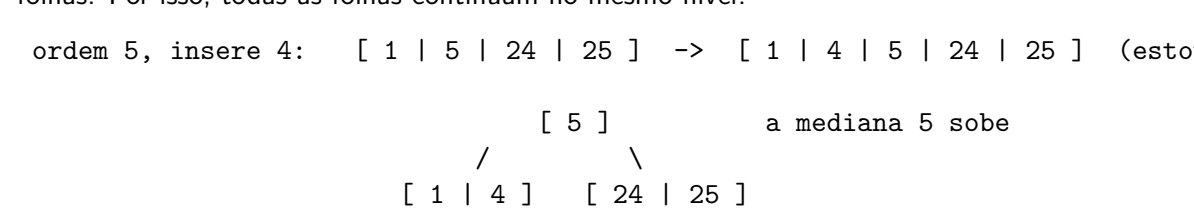
Desafio: escreva `minimo()` e `maximo()` da Árvore B. (Dica: desça sempre pelo primeiro, ou

Inserção em Árvore B

A nova chave é sempre inserida em uma **folha**: busque a folha onde ela deveria estar, e insira de forma **ordenada**. Dois casos:

- **Caso 1: a chave cabe na página** (menos de $m - 1$ chaves): pronto!
- **Caso 2: a página estoura** (fica com m chaves): **divida** a página (*split*):
  - a chave do meio (a **mediana**) **sobe** para o pai
  - as chaves menores formam uma página, e as maiores, outra
  - se o pai também estourar, ele se divide da mesma forma, e assim por diante

Se a **raiz** estourar, a mediana forma uma **nova raiz**: a árvore cresce **pela raiz**, e não pelas folhas. Por isso, todas as folhas continuam no mesmo nível!

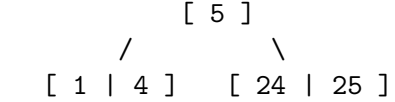


Exemplo: Inserção em Árvore B de Ordem 5 (1/2)

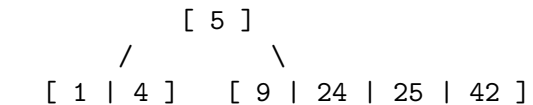
Insira, nesta ordem: 25 5 1 24 4 9 42 14 16 12

25, 5, 1, 24: [1 | 5 | 24 | 25] (cabem: até 4 chaves)

4: estoura! [1 | 4 | 5 | 24 | 25] -> mediana 5 sobe, nova raiz:



9, 42:



Exemplo: Inserção em Árvore B de Ordem 5 (2/2)

```
14: estoura!      [ 9 | 14 | 24 | 25 | 42 ]  ->  mediana 24 sobe para o pai:

                                [ 5 | 24 ]
                              /      |      \
                     [ 1 | 4 ]  [ 9 | 14 ]  [ 25 | 42 ]

16, 12:                         [ 5 | 24 ]
                              /      |           \
                     [ 1 | 4 ]  [ 9 | 12 | 14 | 16 ]  [ 25 | 42 ]
```

Exercício: continue inserindo 13. Quem sobe? A raiz muda?

Implementação: Inserção (I)

Duas operações auxiliares: inserir uma chave (com seu filho direito) **dentro** de uma página, e **dividir** uma página que estourou:

```
// insere c, e seu filho direito dir,
//   na posição p da página
void insereNaPagina(NoB* no, int p,
                    int c, NoB* dir) {
   for (int i = no->k; i > p; i--) {
      no->chaves[i] = no->chaves[i - 1];
      no->filhos[i + 1] = no->filhos[i];
   }
   no->chaves[p] = c;
   no->filhos[p + 1] = dir;
   no->k++;
}
```

```
struct Promocao {  // o que sobe
   int mediana;
   NoB* direito;   // 0: nada subiu
};
Promocao divide(NoB* no) {
   int meio = M / 2;  // pre: k == M
   NoB* d = new NoB{};
   d->k = no->k - meio - 1;
   for (int i = 0; i < d->k; i++)
      d->chaves[i] =
         no->chaves[meio + 1 + i];
   for (int i = 0; i <= d->k; i++)
      d->filhos[i] =
         no->filhos[meio + 1 + i];
   no->k = meio;  // esquerda fica
```

Implementação: Inserção (II)

A inserção desce recursivamente até a folha, e **sobe** devolvendo o que foi promovido em cada divisão:

```
Promocao insereRec(NoB* no, int c) {
   int p = buscaMaiorIgual(no, c);
   if (p < no->k && no->chaves[p] == c)
      return {0, 0};              // já existe
   if (folha(no))
      insereNaPagina(no, p, c, 0);
   else {
      Promocao pr =
         insereRec(no->filhos[p], c);
      if (!pr.direito)
         return {0, 0};           // nada subiu
      insereNaPagina(no, p, pr.mediana,
                     pr.direito);
   }
   if (no->k < M) return {0, 0}; // coube!
```

```
void adiciona(int c) {
   if (!raiz) {
      raiz = new NoB{.k = 1,
                     .chaves = {c}};
      return;
   }
   Promocao pr = insereRec(raiz, c);
   if (pr.direito)  // raiz dividiu:
      raiz = new NoB{
         .k = 1,
         .chaves = {pr.mediana},
         .filhos = {raiz, pr.direito
}
```

Custo: $O(m \log_m n)$, pois cada nível pode

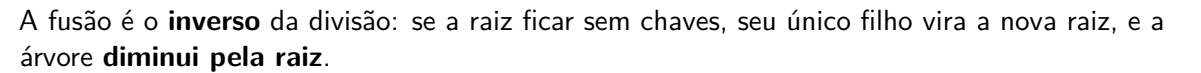
Remoção em Árvore B (ideia)

Assim como na ABB, a remoção de uma chave em nó **interno** é reduzida à remoção em uma **folha**: troque a chave pelo seu **antecessor** (o máximo de `filhos[p]`) ou **sucessor** (o mínimo de `filhos[p+1]`), que sempre estão em folhas.

Removendo da folha, ela pode ficar com **menos** de $\lceil m/2 \rceil - 1$ chaves (*underflow*). Dois casos:

- **Empréstimo:** um irmão vizinho tem chaves sobrando: a chave do pai desce para a folha, e uma chave do irmão sobe para o pai (uma "rotação" através do pai!)
- **Fusão:** os irmãos estão no mínimo: a folha, o irmão e a chave do pai entre eles **se fundem** em uma única página. O pai perde uma chave, e pode sofrer *underflow* também

A fusão é o **inverso** da divisão: se a raiz ficar sem chaves, seu único filho vira a nova raiz, e a árvore **diminui pela raiz**.



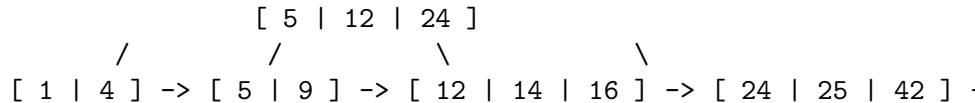
Section 3

Árvores B+

Árvore B+

Uma *Árvore B+* é uma variação da Árvore B, usada na maioria dos bancos de dados e em sistemas de arquivos (como NTFS e XFS), em que:

- **todas** as chaves (e seus valores) ficam nas **folhas**
- os nós internos guardam apenas **cópias** de chaves, que servem só para **guiar** a busca
- as folhas são **encadeadas** em uma lista, em ordem crescente



Com isso:

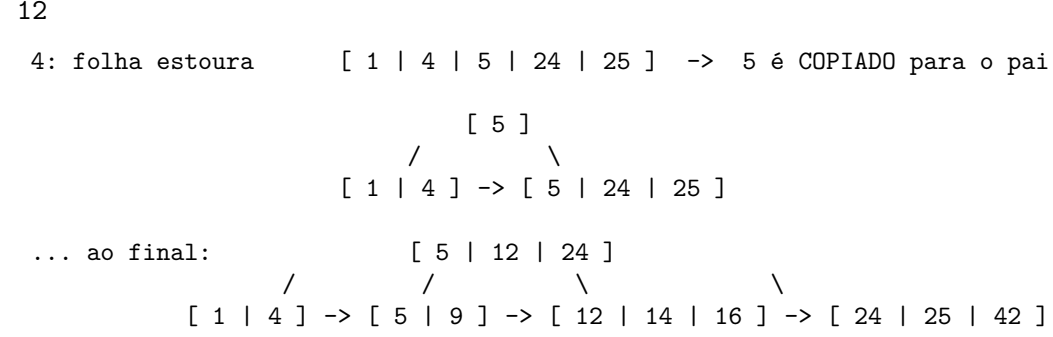
- toda busca termina em uma **folha** (mesmo que a chave apareça em um nó interno)
- os nós internos ficam **menores** (sem valores): cabem mais chaves por página, e a árvore fica ainda mais **baixa**
- o **percurso em ordem** é só percorrer a lista de folhas!

Inserção em Árvore B+

A inserção é igual à da Árvore B, com uma diferença na **divisão de folhas**:

- **folha estourou:** a mediana é **copiada** para o pai, e **continua** na folha da direita (os dados ficam nas folhas!)
- **nó interno estourou:** a mediana **sobe**, como na Árvore B

Mesma sequência do exemplo anterior, em uma B+ de ordem 5: 25 5 1 24 4 9 42 14 16 12



Busca por Intervalo na Árvore B+

Lembra do **Dicionário Ordenado**? Na B+, o intervalo $[a, b]$ fica muito simples:

1. desça da raiz até a folha de `buscaMaiorIgual(a)`: $O(\log_m n)$ páginas
2. percorra a **lista de folhas** até passar de b: $O(k/m)$ páginas para k chaves

```
chaves em [9, 25]:                [ 5 | 12 | 24 ]
                                   /
          [ 1 | 4 ] -> [ 5 | 9 ] -> [ 12 | 14 | 16 ] -> [ 24 | 25 | 42 ]
                          ^                                    ^
                       início                          para em 42 > 25

resultado: 9 12 14 16 24 25
```

Na Árvore B comum, o mesmo percurso precisaria **subir e descer** pela árvore, como o `sucessor` da ABB. Na B+, o `proximo()` do iterador é só avançar na folha, ou pular para a folha seguinte!

É exatamente a consulta `SELECT ... WHERE x BETWEEN 9 AND 25` de um banco de dados

Árvore B × Árvore B+

	Árvore B	Árvore B+
Onde ficam os valores	em todos os nós	só nas folhas
Nós internos	chaves e valores	só cópias de chaves (guias)
Chaves repetidas	não	sim (cópias nos nós internos)
Busca de uma chave	pode parar em nó interno	sempre vai até a folha
Percurso em ordem / intervalo	sobe e desce na árvore	percorre a lista de folhas
Divisão de folha	mediana sobe	mediana é **copiada**
Uso típico	sistemas de arquivos, memória	índices de bancos de dados

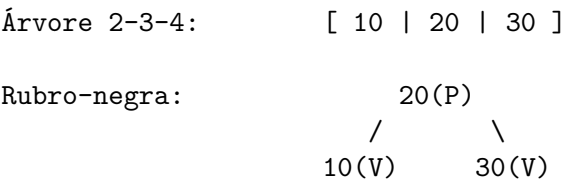
Ambas têm altura $O(\log_m n)$ e são perfeitamente balanceadas.

Section 4

Considerações Finais

Árvores B e Árvores Rubro-Negras

Na aula anterior, vimos que cada nó preto de uma rubro-negra, junto com seus filhos vermelhos, equivale a um nó com até 3 chaves. Agora sabemos o nome disso: uma **Árvore B de ordem 4** (árvore 2-3-4)!



- uma **divisão** na Árvore B corresponde a uma **recoloração** (caso 3) na rubro-negra
- a **altura negra** da rubro-negra é a **altura** da Árvore B correspondente

Por isso, a rubro-negra é uma "Árvore B de ordem 4 disfarçada" de árvore binária: ganha o balanceamento perfeito da Árvore B, mas mantém nós binários, mais simples em memória principal.

Exercícios

1. Insira, em uma Árvore B de ordem 5 inicialmente vazia: 20 40 10 30 15 35 7 26 18 22 5 42 13 46. Desenhe a árvore após inserir 5, 9 e 14 chaves.
2. Repita o exercício 1 em uma Árvore B+ de ordem 5. Compare as alturas.
3. Quantas comparações são necessárias para buscar o 12 na árvore final do exemplo de inserção? E na B+ correspondente?
4. Qual o número **máximo** de chaves em uma Árvore B de ordem m com h níveis? E o mínimo?
5. Implemente `minimo()`, `maximo()` e `conta()` (número de chaves) para a Árvore B.
6. Por que a inserção sempre ocorre em uma **folha**? O que garante que todas as folhas continuam no mesmo nível?

Simulador online: https://www.cs.usfca.edu/~galles/visualization/BTree.html (há também a versão B+, em `BPlusTree.html`)

Árvores B e B+: Revisão Geral

- O que é uma Árvore de Busca Multivias? Como cada chave de um nó divide a busca?
- Quais as regras de uma Árvore B de ordem m? Entre quantos filhos varia cada nó?
- Por que a Árvore B é sempre perfeitamente balanceada, sem rotações?
- Por que a altura $O(\log_m n)$ é tão importante quando os dados estão em disco?
- O que acontece quando uma página estoura? E quando a raiz estoura?
- Qual a diferença entre a divisão de uma folha na Árvore B e na Árvore B+?
- Por que a Árvore B+ é preferida para consultas por intervalo?

Bibliografia Recomendada

Além da bibliografia do curso, recomendamos para esse tópico:

- Szwarcfiter, J.L; Markenzon, L. Estruturas de Dados e seus Algoritmos. Rio de Janeiro, LTC, 1994.

Bibliografia Adicional:

- Mochetti, K. Estruturas de Dados: Árvore B (slides), IC-UFF.
- Bayer, R.; McCreight, E. Organization and Maintenance of Large Ordered Indexes. Acta Informatica 1, 1972.
- Cormen, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C. Introduction to Algorithms, 3rd ed.. The MIT Press, 2009 (Capítulo 18).
- Knuth, D.E. The Art of Computer Programming - Vol III. 2nd Edition. Addison Wesley, 1998.
- Comer, D. The Ubiquitous B-Tree. ACM Computing Surveys 11(2), 1979.
- Livro "The C++ Programming Language" de Bjarne Stroustrup
- Dicas e normas C++: https://github.com/isocpp/CppCoreGuidelines

Section 5

Agradecimentos

Pessoas

Em especial, agradeço aos colegas que elaboraram bons materiais, como a prof. Karina Mochetti (IC-UFF), cujo material de Árvores B inspirou esta aula, o prof. Fabiano Oliveira (IME-UERJ), e o prof. Jayme Szwarcfiter cujos conceitos formam o cerne desses slides.

Estendo os agradecimentos aos demais colegas que colaboraram com a elaboração do material do curso de Pesquisa Operacional, que abriu caminho para verificação prática dessa tecnologia de slides.

Software

Esse material de curso só é possível graças aos inúmeros projetos de código-aberto que são necessários a ele, incluindo:

- pandoc
- LaTeX
- GNU/Linux
- git
- markdown-preview-enhanced (github)
- visual studio code
- atom
- revealjs
- groomit-mpx (screen drawing tool)
- xournal (screen drawing tool)
- ...

Empresas

Agradecimento especial a empresas que suportam projetos livres envolvidos nesse curso:

- github
- gitlab
- microsoft
- google
- ...

Reprodução do material

Esses slides foram escritos utilizando pandoc, segundo o tutorial ilectures:

- https://igormcoelho.github.io/ilectures-pandoc/

Exceto expressamente mencionado (com as devidas ressalvas ao material cedido por colegas), a licença será Creative Commons.

Licença: CC-BY 4.0 2020

Igor Machado Coelho

This Slide Is Intentionally Blank (for goomit-mpx)