Engenharia de Computação

Lógica e Programação de Computadores

Victor Machado da Silva, MSc
victor.silva@professores.ibmec.edu.br

Ibmec
PROTAGONISTAS PARA O MUNDO

Índice

PROTAGONISTAS PARA O MUNDO

Índice

PROTAGONISTAS PARA O MUNDO

Apresentação do curso

Apresentação do curso

- Contato: victor.silva@professores.ibmec.edu.br
- Aulas às segundas e quartas-feiras, de 9:49 às 11:50
- Grupo no Whatsapp: https://chat.whatsapp.com/Bb5EKrfEIG7AqENWjqh5Uc
- Material no GitHub: https://github.com/victor0machado/2020.2-logprog

Ibmec
PROTAGONISTAS PARA O MUNDO

Apresentação do curso

Sem.	Data	Tópico
01	03/08	Introdução à disciplina
01	05/08	Revisão de Python: tipos de dados, listas, dicionários, manipulação de arquivos, decorador @timeit
02	10/08	Algoritmos: sintaxe em pseudocódigo, recursividade
02	12/08	Algoritmos: complexidade de algoritmos, notação O, algoritmos ótimos
03	17/08	Algoritmos: exercícios
03	19/08	Aplicação prática: desenvolvendo jogos com pygame
04	24/08	Aplicação prática: desenvolvendo jogos com pygame
04	26/08	Listas lineares: alocação, busca, busca binária, remoção
05	31/08	Listas lineares: pilhas e filas – inserção e remoção, alocação encadeada, listas circulares
05	02/09	Listas lineares: exercícios
06	07/09	SEM AULA (7 DE SETEMBRO)
06	09/09	Árvores: definições e representações básicas, árvores binárias, percurso em árvores binárias
07	14/09	Árvores: conversão de uma floresta, árvores com costura
07	16/09	Árvores: exercícios
08	21/09	Árvores binárias de busca: conceitos, busca e inserção; árvore de partilha
08	23/09	Árvores binárias de busca: exercícios
09	28/09	Atividades em grupo / dúvidas e discussões
09	30/09	Atividades em grupo / dúvidas e discussões
10	05/10	Atividades em grupo / dúvidas e discussões
10	07/10	**P1**
11	12/10	SEM AULA (N. SRA. APARECIDA)

Ibmec
PROTAGONISTAS PARA O MUNDO

Apresentação do curso

Sem.	Data	Tópico
11	14/10	Árvores balanceadas: árvores AVL, árvores graduadas e rubro-negras, árvores B
12	19/10	Árvores balanceadas: exercícios
12	21/10	Listas de prioridades: implementação, alteração de prioridades, máximos e mínimos
13	26/10	Listas de prioridades: exercícios
13	28/10	Algoritmos de ordenação: ordenação bolha, por inserção e por intercalação
14	02/11	SEM AULA (FINADOS)
14	04/11	Algoritmos de ordenação: ordenação rápida, ordenação em heap, árvore de decisão
15	09/11	Algoritmos de ordenação: exercícios
15	11/11	Grafos: definições, representação gráfica, terminologia, caminhos e ciclos
16	16/11	Grafos: representação como estruturas de dados, métodos para percurso em grafos
16	18/11	Grafos: exercícios
17	23/11	Atividades em grupo / dúvidas e discussões
17	25/11	Atividades em grupo / dúvidas e discussões
18	30/11	Atividades em grupo / dúvidas e discussões
18	02/12	**P2**
19	07/12	SEM AULA
19	09/12	Atividade em grupo / dúvidas
20	14/12	**PS**
20	16/12	SEM AULA

Ibmec
PROTAGONISTAS PARA O MUNDO

Apresentação do curso

Avaliação

- Proporção:
  - Exercícios periódicos (AC): 20%
  - Projeto (AP1): 40%
  - Projeto (AP2): 40%
- Detalhes das entregas:
  - Exercícios da AC são individuais
  - Projetos de AP1 e AP2 em grupos de no mínimo 2 e no máximo 3 pessoas
  - Entrega via Integrees
- AS será uma prova com consulta, que substituirá a menor nota entre AP1 e AP2.

Sugestões de materiais para estudo

- Python é uma linguagem intuitiva para o aprendizado, porém é importante termos à mão livros, apostilas e outros materiais para auxiliar os estudos. Abaixo encontram-se algumas sugestões:
  - Documentação oficial em Python: https://docs.python.org/pt-br/3/index.html
  - Jayme Luiz Szwarcfiter, Lilian Markenzon – Estruturas de Dados e seus Algoritmos (LTC)
  - Dilermando Piva Junior et al – Estruturas de Dados e Técnicas de Programação (Campus)
  - Stack Overflow: https://stackoverflow.com/questions/tagged/python
  - Artigos no Medium.com: https://medium.com/search?q=python

Ibmec
PROTAGONISTAS PARA O MUNDO

Sugestões de materiais para estudo

- Canais interessantes no Youtube sobre Python e programação:
  - Programação Dinâmica: https://www.youtube.com/c/ProgramacaoDinamica/
  - Curso em Vídeo: https://www.youtube.com/c/CursoemVideo/
  - Sentdex (em inglês): https://www.youtube.com/c/sentdex
  - Filipe Deschamps: https://www.youtube.com/c/FilipeDeschamps
  - DevMedia: https://www.youtube.com/c/DevmediaBrasil

Configurando o ambiente Python

PROTAGONISTAS PARA O MUNDO

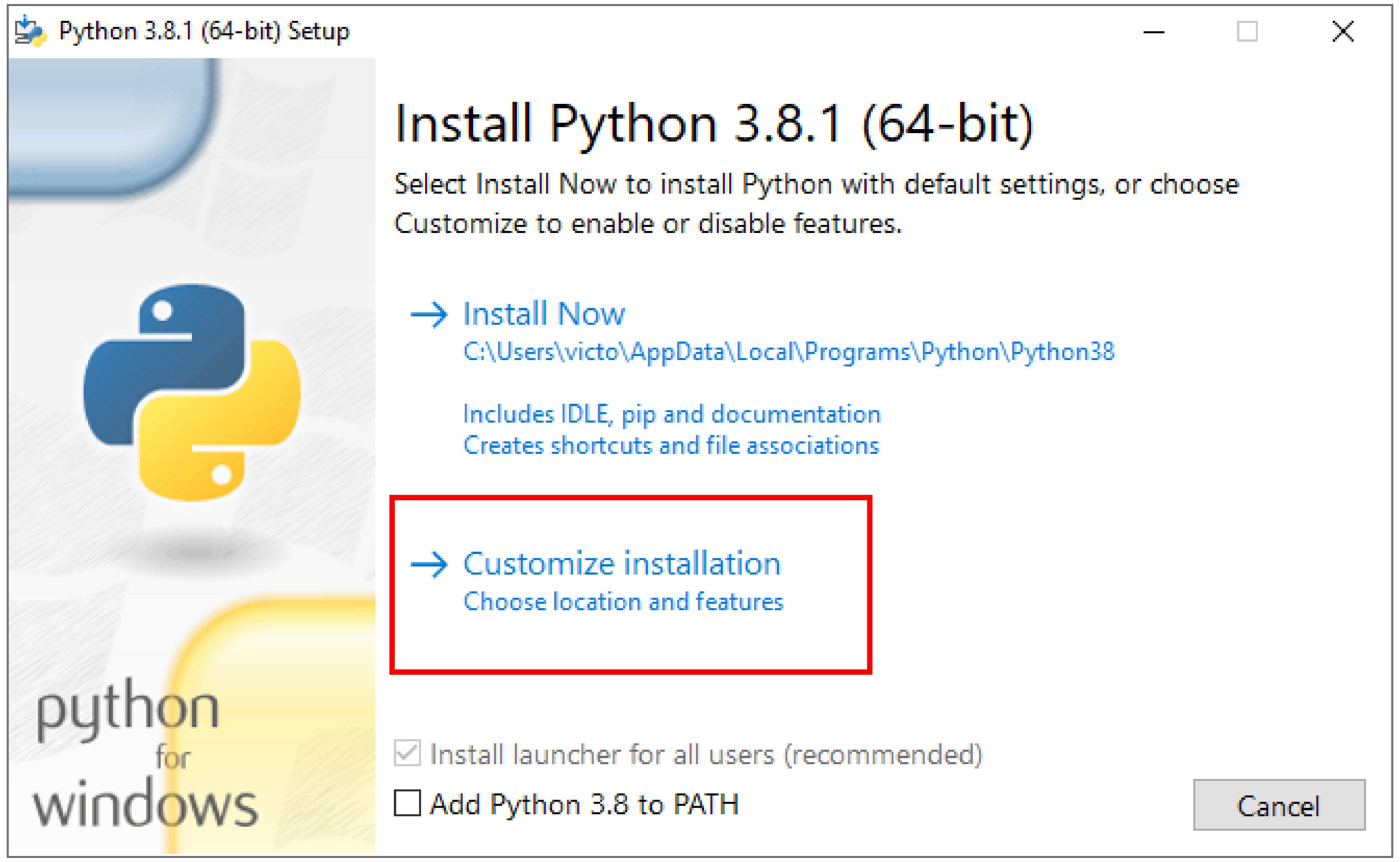
O que é Python?

- Python é uma linguagem de programação de alto nível, lançada por Guido van Rossum em 1991. Atualmente é uma das linguagens de uso mais abrangentes no mundo todo, principalmente nas áreas de Data Science e em aplicações de *back-end*, ou seja, de processamento de dados que não interagem diretamente com o usuário final.
- Diversas organizações utilizam Python atualmente:
  - Google
  - Yahoo!
  - NASA
  - AirCanada

Ibmec
PROTAGONISTAS PARA O MUNDO

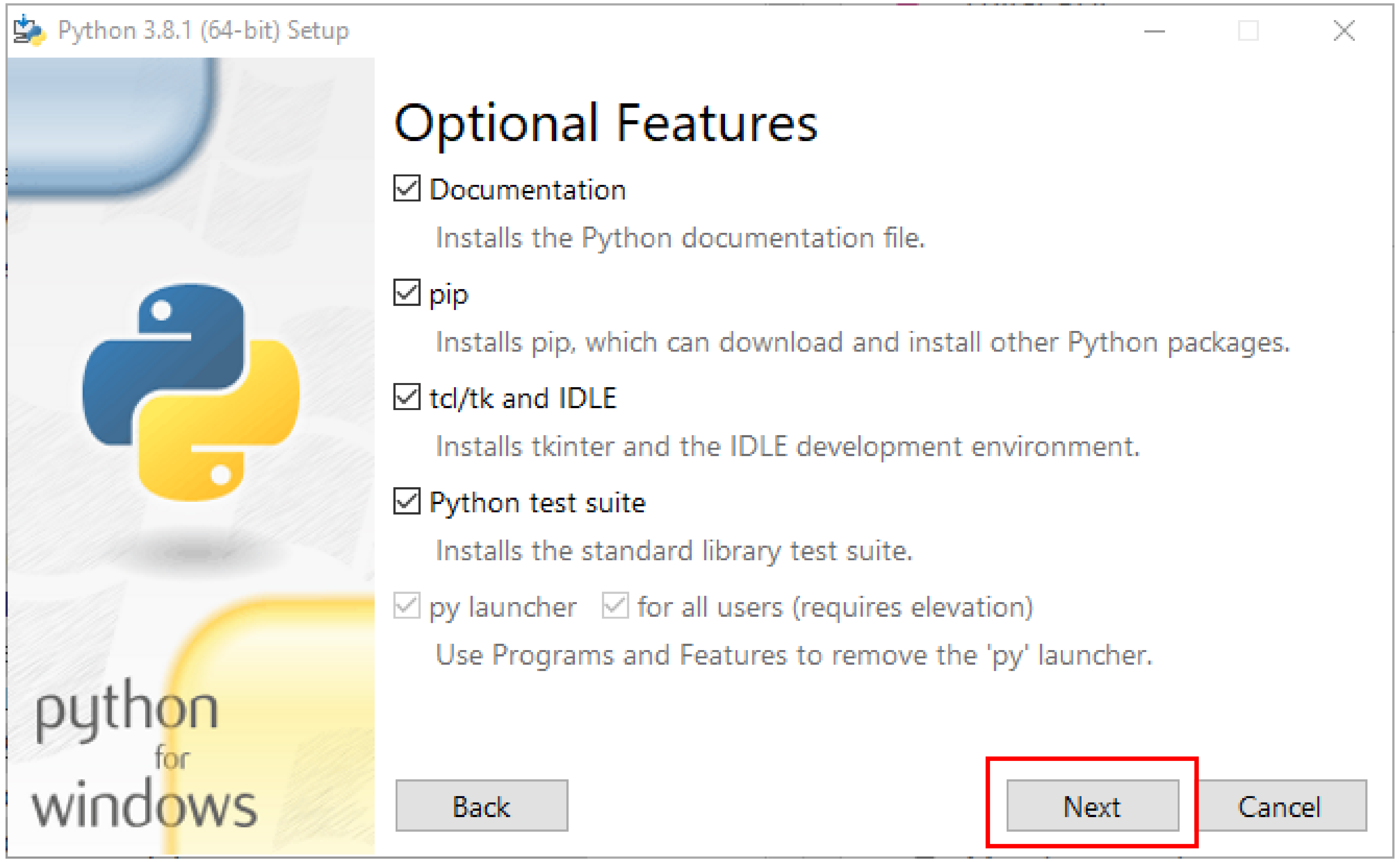
Como instalar Python?

- Neste curso podemos trabalhar com qualquer versão recente do Python, 3.7 ou superior. Para baixar o instalador para Windows, clique neste link. O download do instalador para macOS se encontra neste link.
- Este curso focará no uso do Python para Windows. Para o uso no macOS, veja no site oficial da linguagem informações particulares.
- Ao clicar no instalador, selecione a opção “Customize installation”.



Como instalar Python?

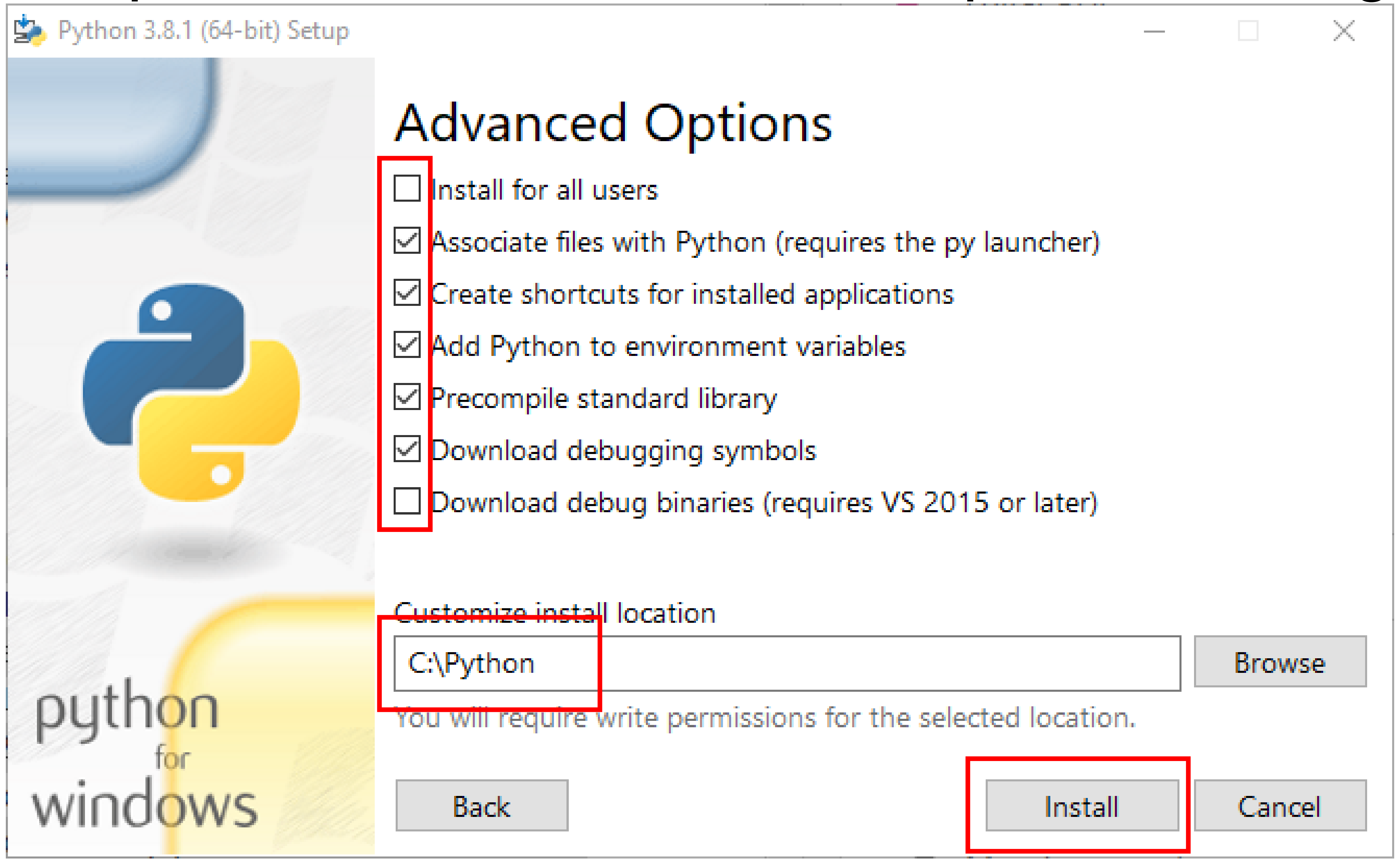
- Na tela “Optional Features”, clique em “Next”.



Ibmec
PROTAGONISTAS PARA O MUNDO

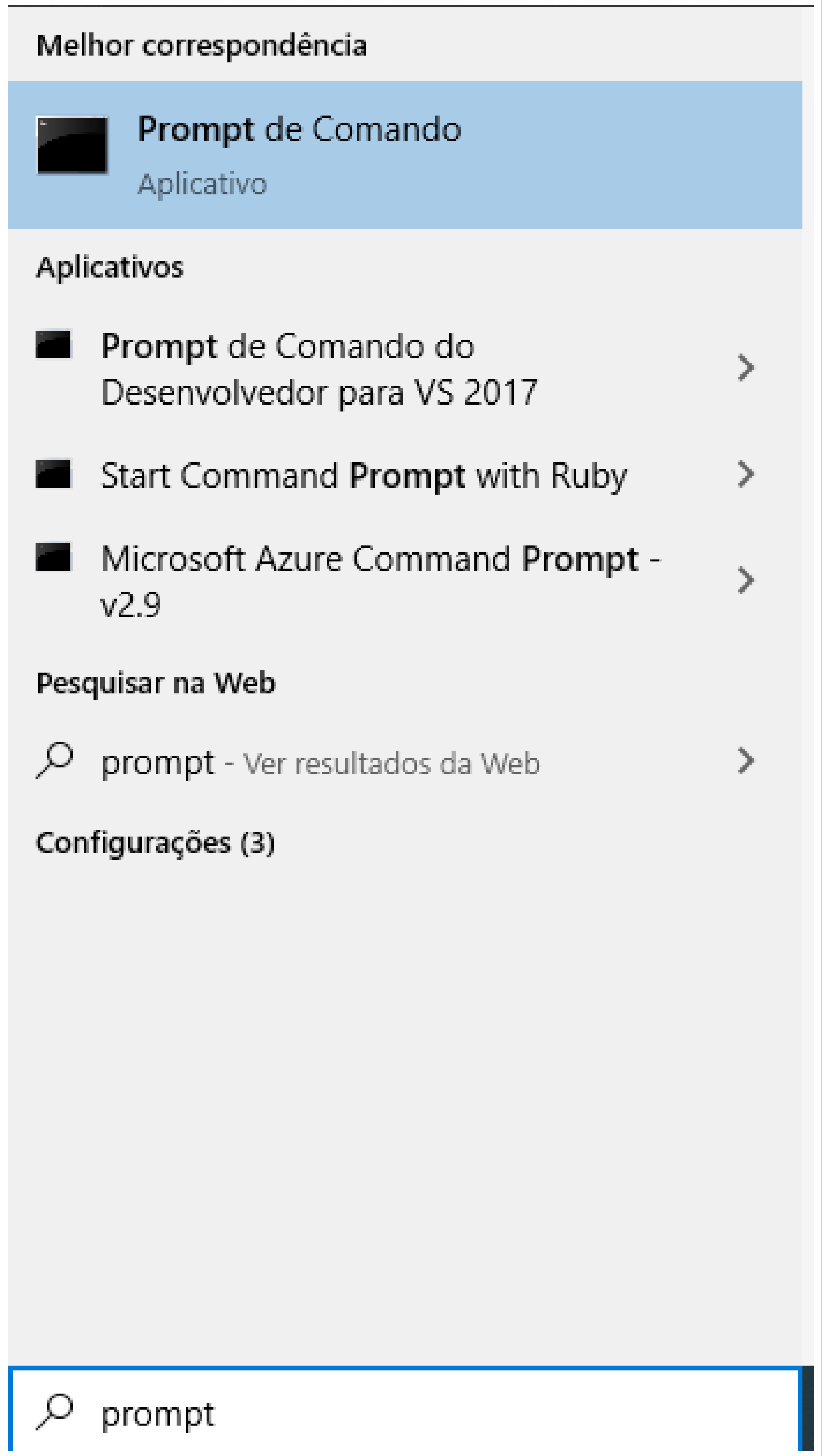
Como instalar Python?

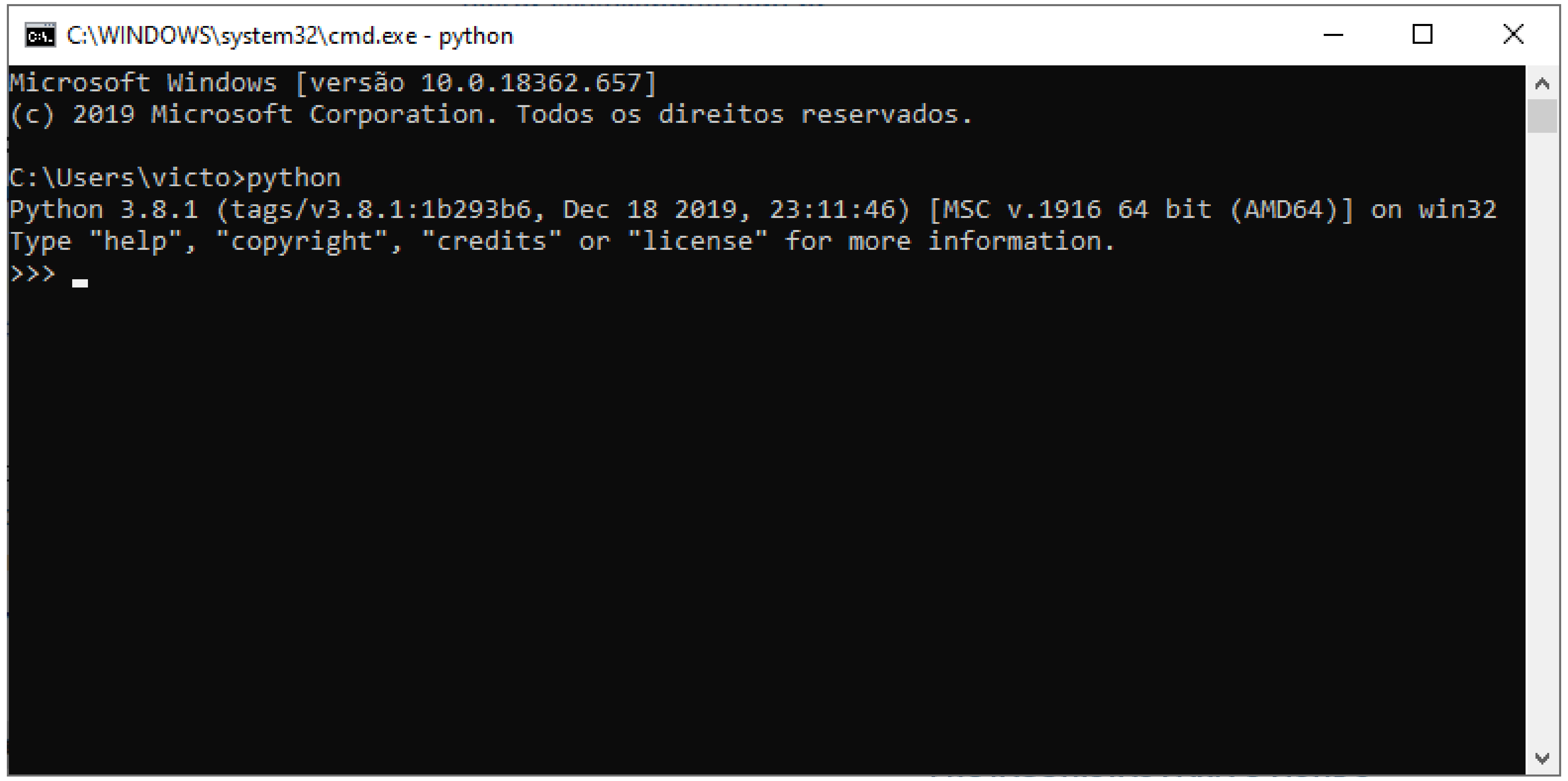
- Na tela “Advanced Options”, deixe as caixas de opções marcadas conforme a imagem abaixo.
- No campo “Customize install location”, escolha um caminho de fácil acesso. O caminho sugerido é “C:\Python”.
- Com tudo pronto, clique em “Install”, e conclua após a mensagem de sucesso.



Como instalar Python?

- Para conferir se a instalação foi bem sucedida, faça os seguintes passos:
  - Clique no botão do Windows ;
  - Digite a caixa de pesquisa **prompt de comando**, e abra o programa;
  - Na janela que abrir, digite o comando **python** e pressione Enter;
  - O Windows deve inicializar um editor de Python na mesma janela, como mostrado abaixo.





Algumas dicas iniciais após instalar o Python

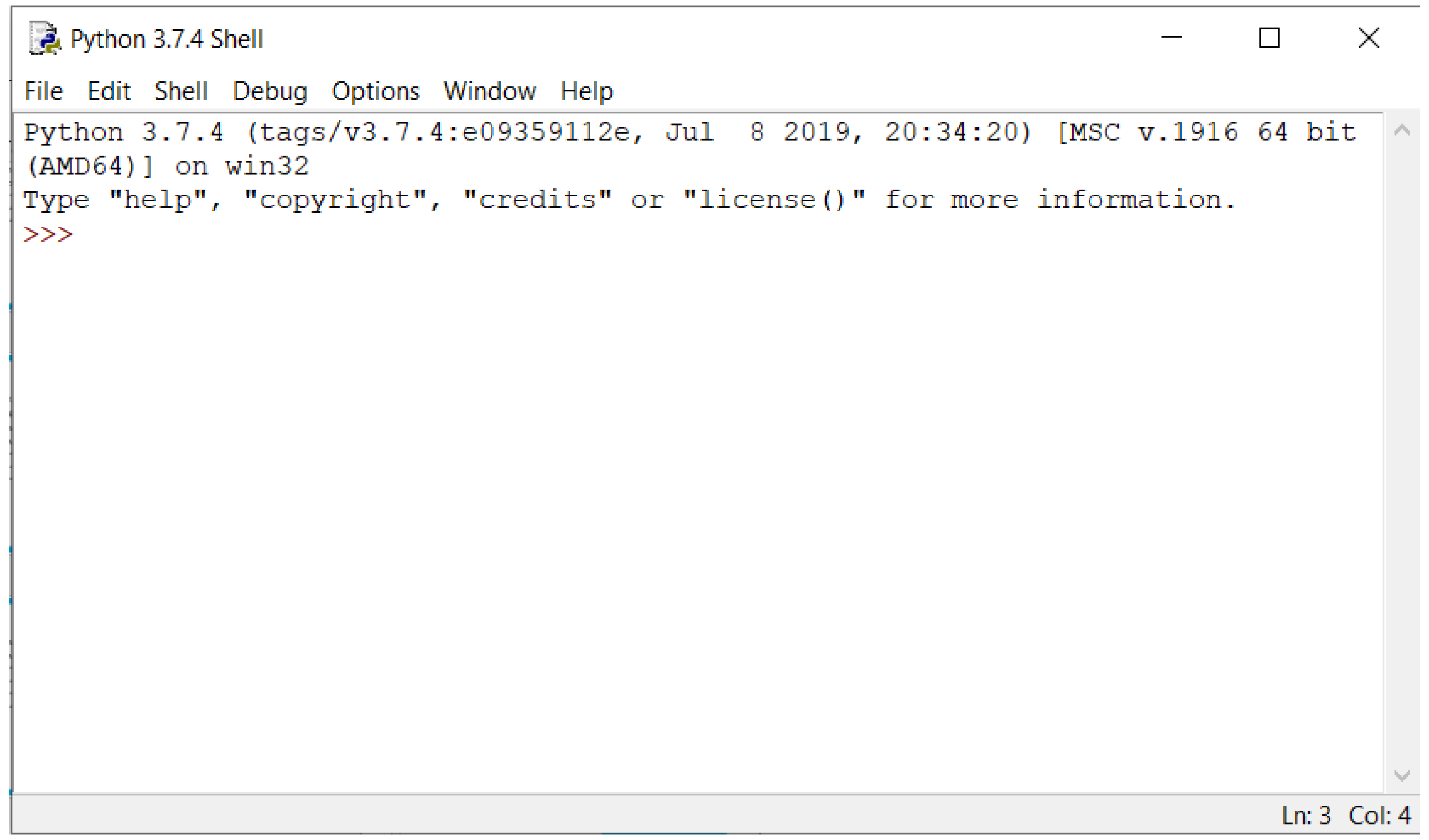
1. Antes de abrir o programa, abra o Windows Explorer, clique em **Este Computador** e, em seguida, no drive do seu computador (C: ou D:, dependendo da sua máquina);
2. Neste diretório, crie uma pasta chamada **Projetos**. Esta pasta será usada para armazenar todos os seus projetos de software;
3. Dentro da pasta de projetos, crie a pasta da disciplina (p.ex., **algoritmos**);
4. Evite utilizar caminhos muito longos (p.ex., C:\Users\12304010\Projetos\Nome-da-pessoa\Documentos\etc...) ou incluir espaços no caminhos (p.ex., C:\Victor Machado). O primeiro é muito trabalhoso para utiliza-lo recorrentemente, e o segundo pode causar alguns problemas na execução do código;
5. Sempre que criar arquivos Python, comece o nome do arquivo com uma letra (p.ex., **main.py**, **app.py**, **aula.py**). Evite usar números, espaços ou acentos nos nomes dos arquivos.

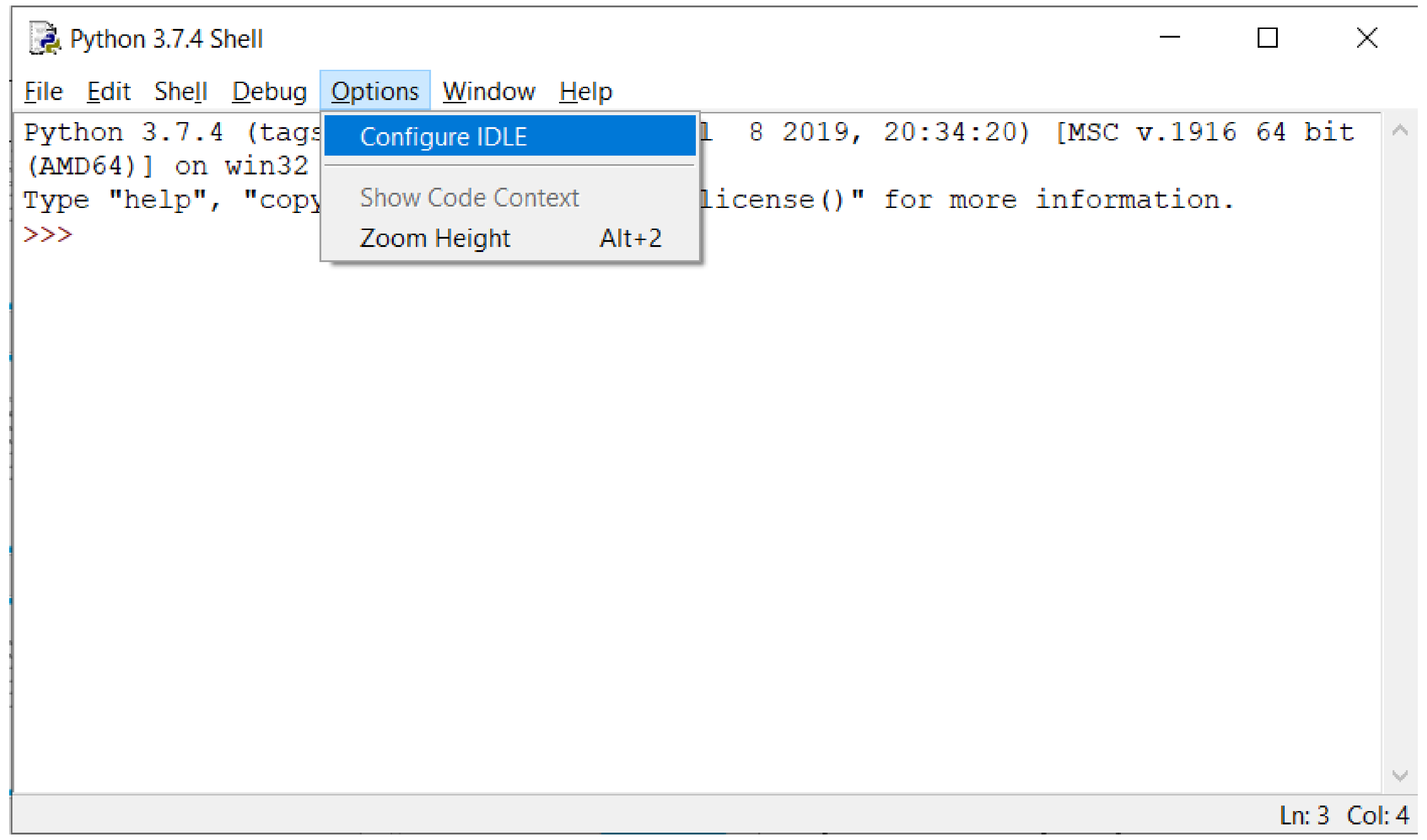
Instalando e configurando IDEs

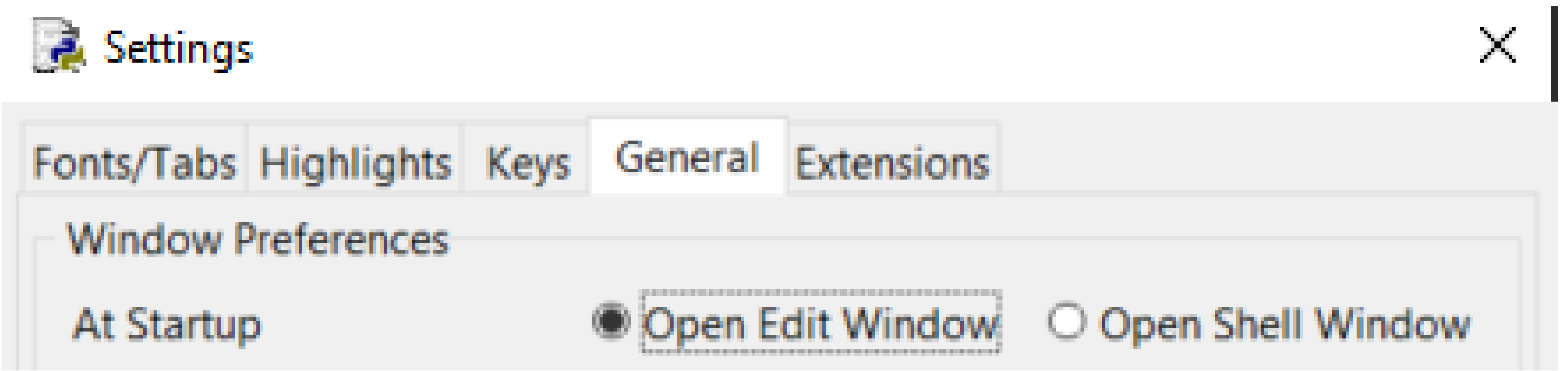
Ibmec
PROTAGONISTAS PARA O MUNDO

Configurando o IDLE

- Abra IDLE utilizando o ícone do Windows e procurando pelo programa
- O programa abre no modo **Shell**, que é a janela de execução do código. Usamos essa tela apenas para checar os resultados ou quando queremos executar códigos de uma linha (testar uma função, por exemplo)
- Para abrir o editor automaticamente, vá no menu **Options > Configure IDLE**
- Na aba **General**, em **Windows Preferences**, marque a opção Open Edit Window. Clique em Ok e reinicie o IDLE

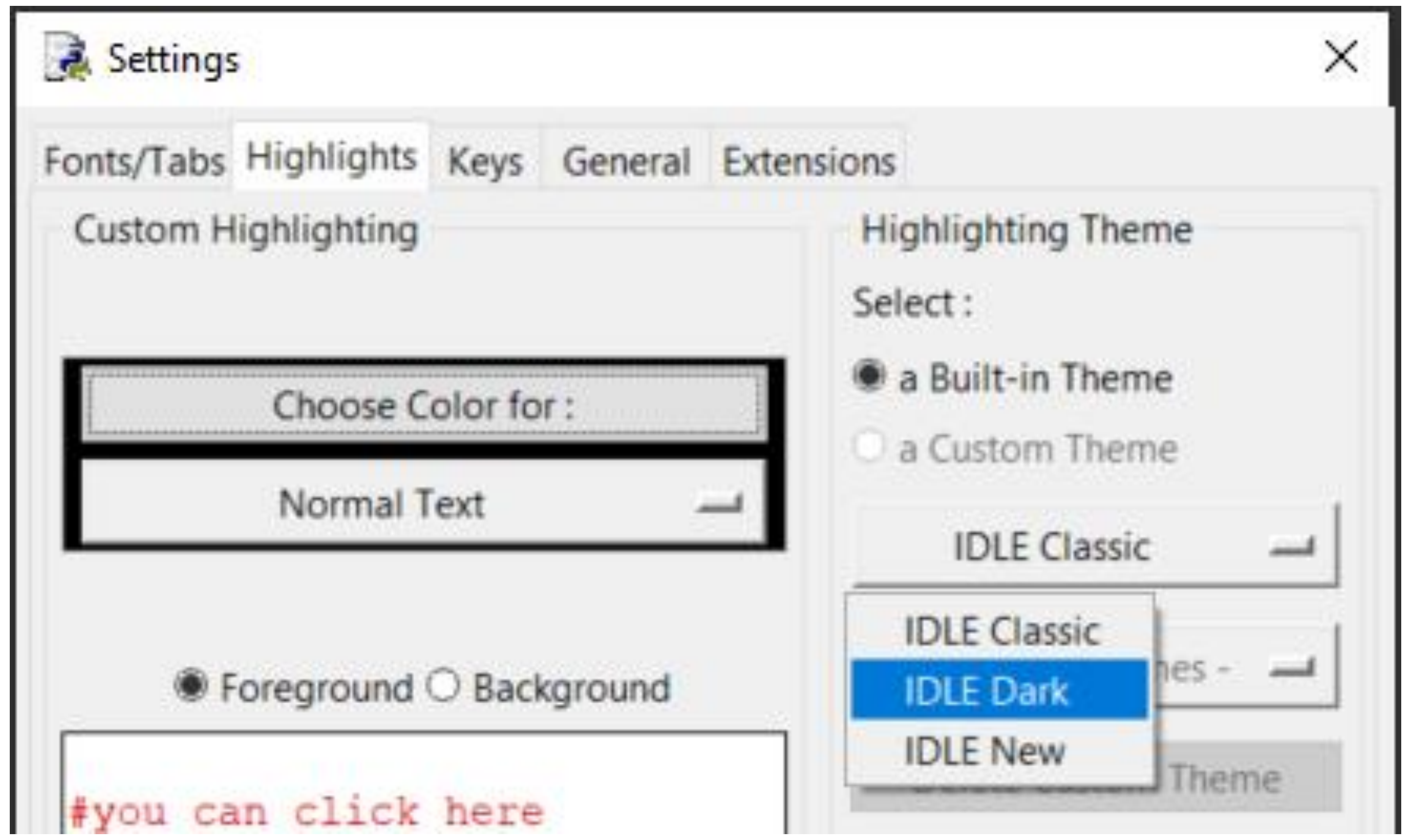






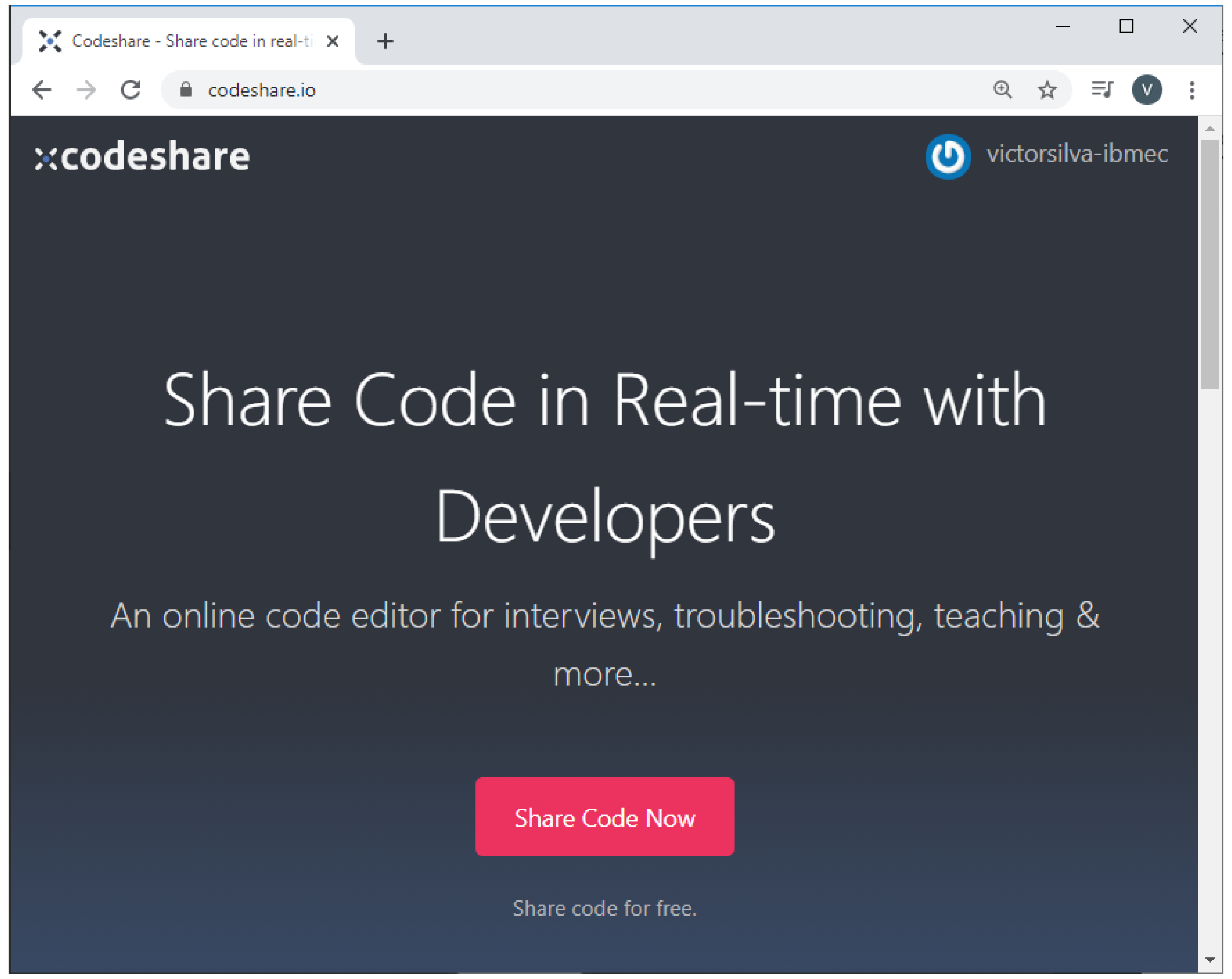
Configurando o IDLE

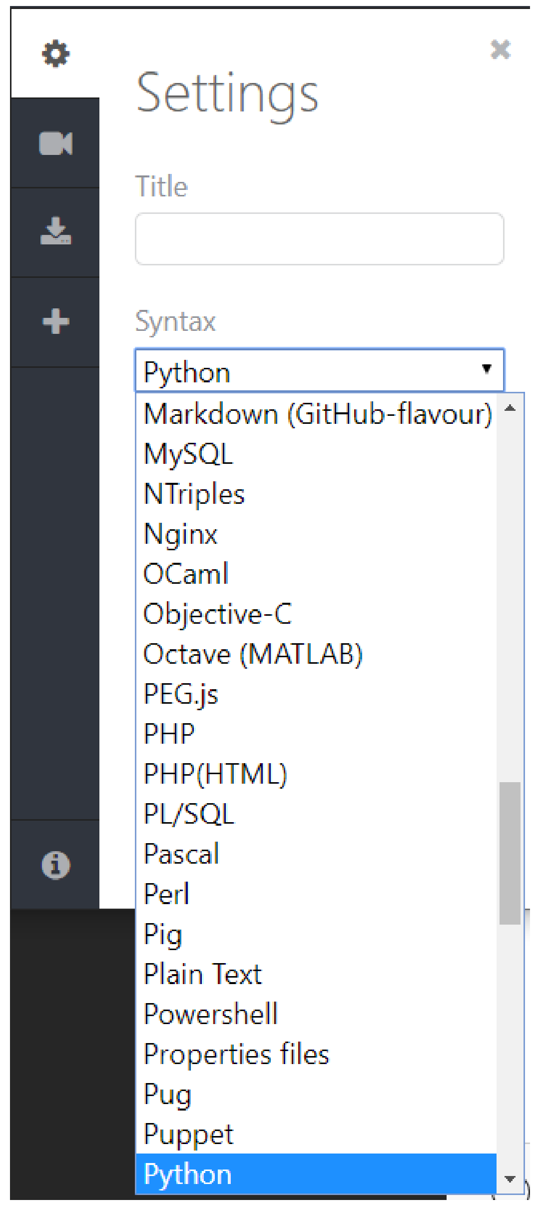
- Normalmente é usual programadores trabalharem com editores de texto que tenham um fundo escuro, o que prejudica menos a visão
- Na mesma tela de **Configurações**, vá para a aba **Highlights**, e na coluna da direita, em **Highlight Theme**, clique em **IDLE Classic** e selecione a opção **IDLE Dark**
- Clique em OK para mudar o tema para escuro

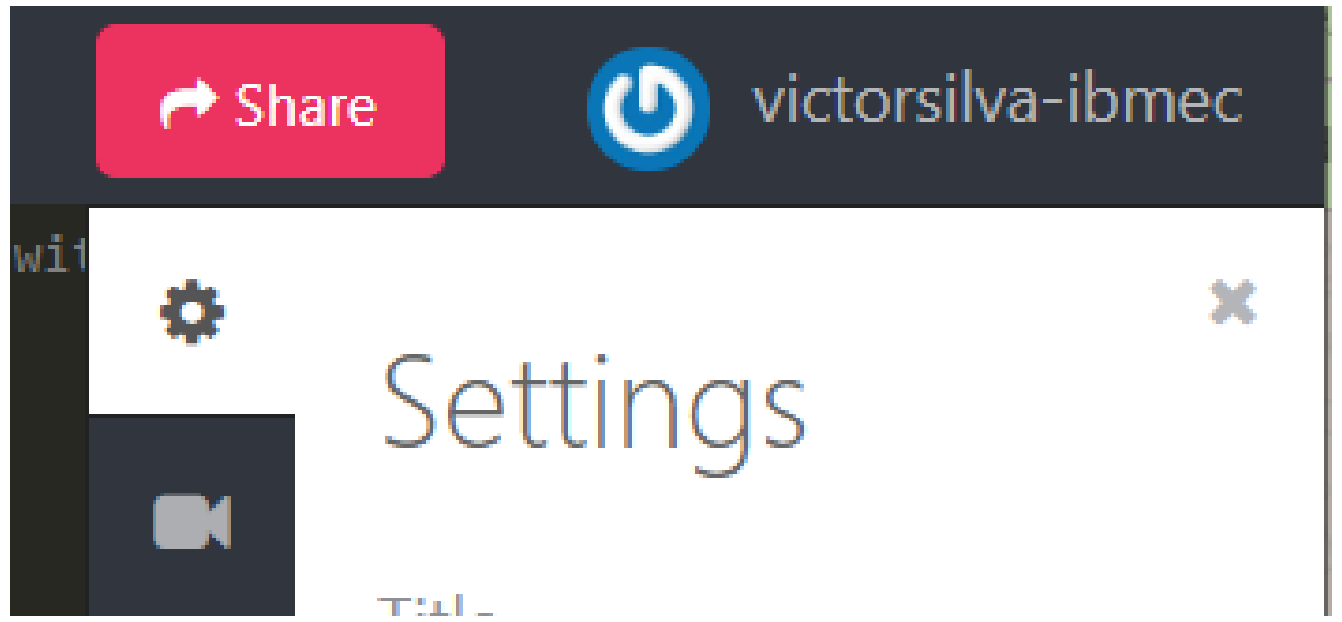


Usando o CodeShare

- Acesse http://codeshare.io e faça o cadastro
- Tendo cadastrado, na tela inicial clique em “Share Code Now”
- Um editor de texto vai aparecer. Na coluna da direita, clique na engrenagem, e em **Syntax** marque a opção **Python**
- Para compartilhar, clique em **Share**, no canto superior da janela
- O CodeShare vai liberar um link para ser usado por outras pessoas

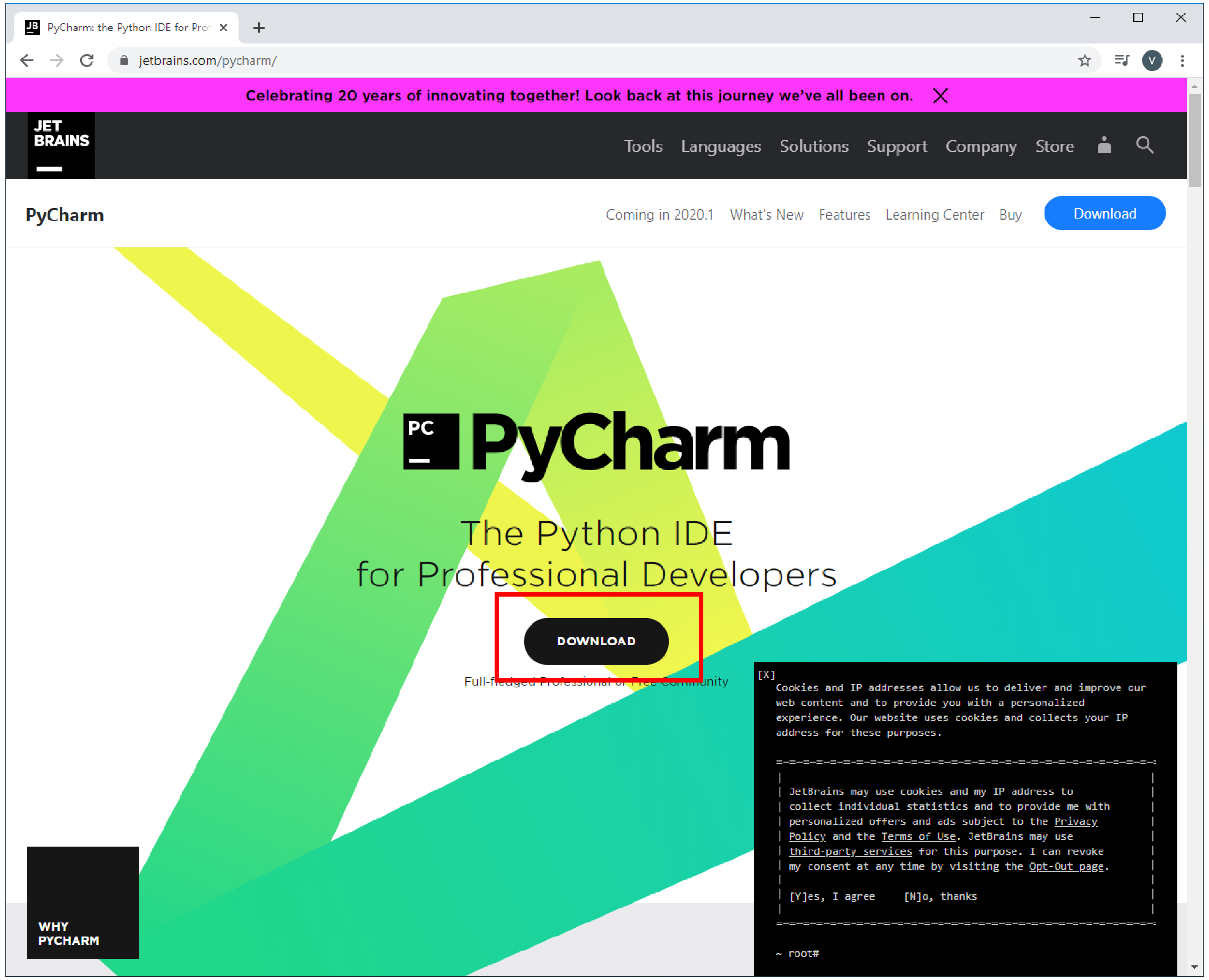






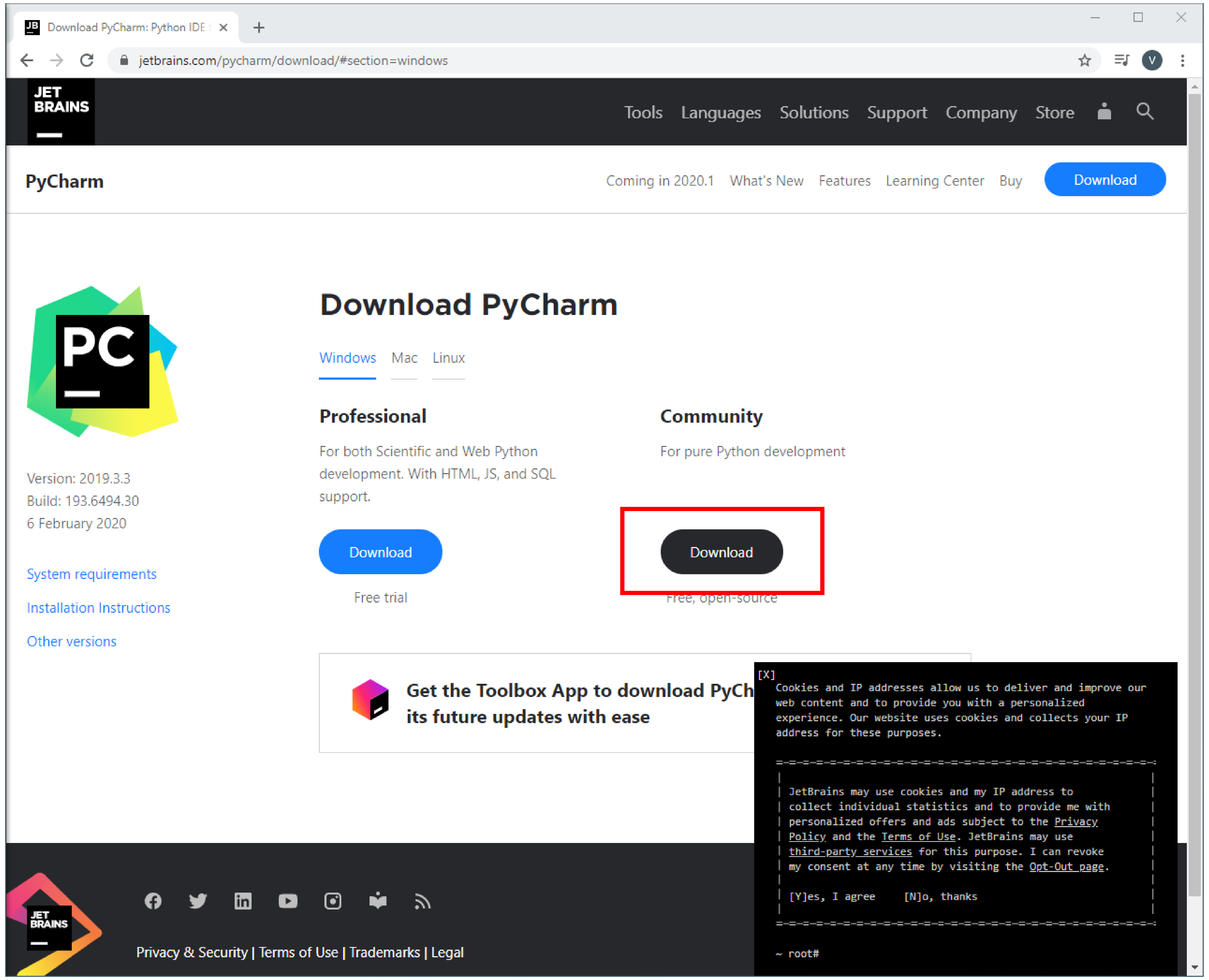
Instalando e configurando o PyCharm

- O **PyCharm** é um dos editores (ou IDEs) mais usados para a programação de aplicações em Python. Para baixar e instalar, primeiro acesse a página https://www.jetbrains.com/pycharm/ e clique em “Download”.



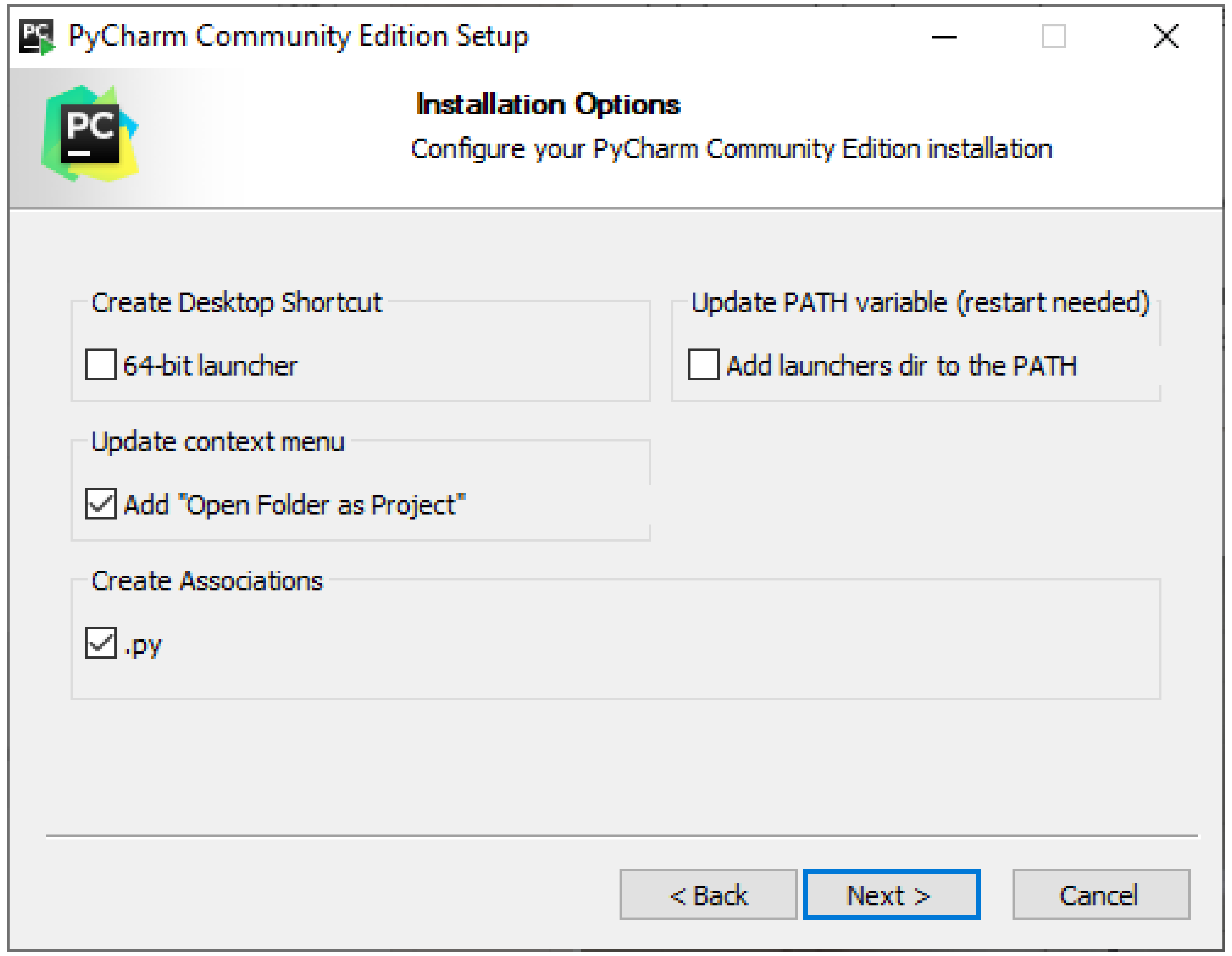
Instalando e configurando o PyCharm

- Escolha o seu sistema operacional (Windows, Mac ou Linux - vamos trabalhar com Windows) e baixe a versão “Community”. O download deve começar automaticamente.



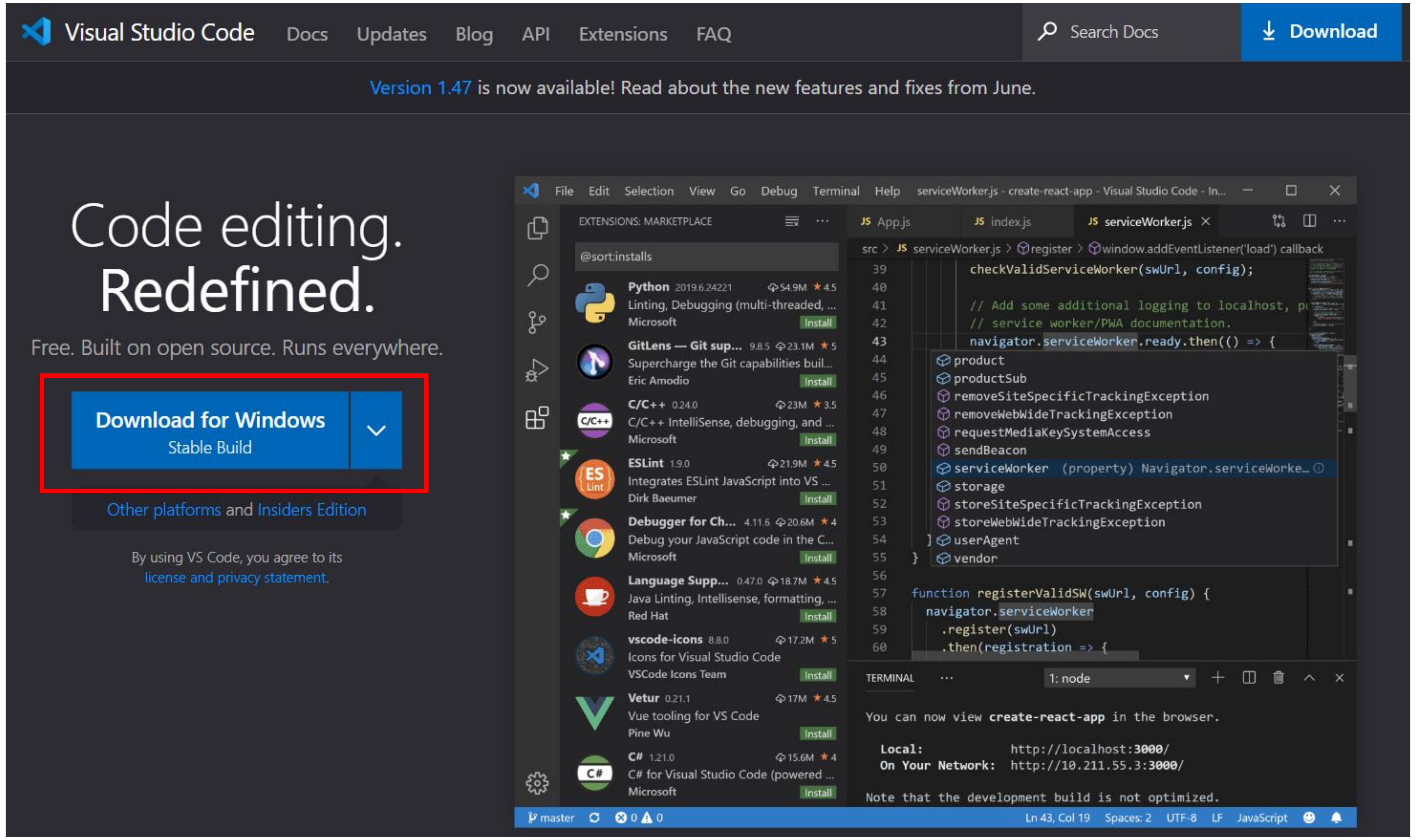
Instalando e configurando o PyCharm

- Abra o instalador do PyCharm e clique em “Next”;
- Na tela de escolha do caminho de instalação, clique em “Next”;
- Na tela seguinte, marque as opções indicadas na imagem abaixo e clique em “Next”;
- Na tela seguinte, clique em “Install” para começar a instalação.



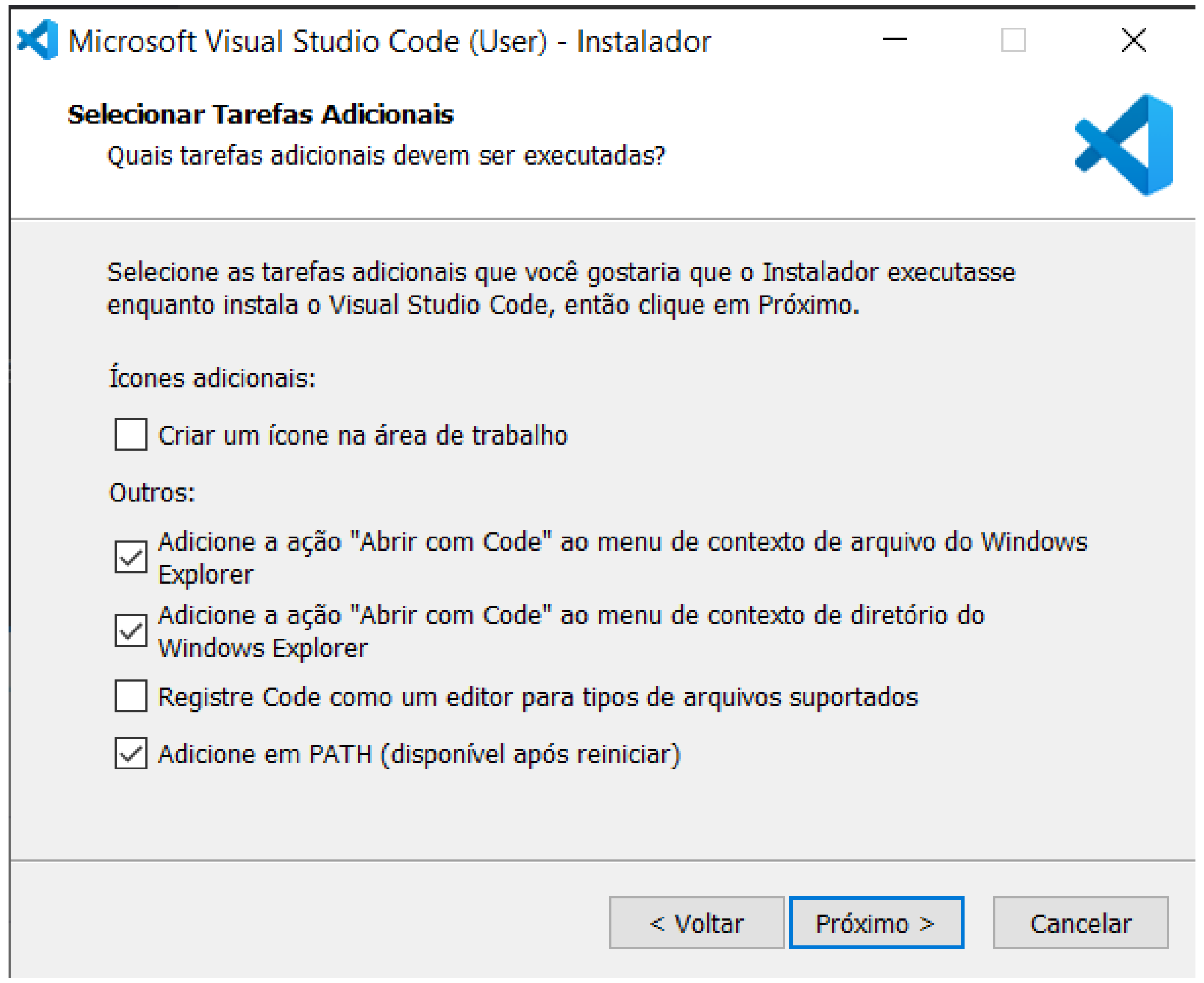
Instalando e configurando o VSCode

- O **VSCode** é um dos IDEs recentes mais famosos para basicamente qualquer linguagem de programação. Possui inúmeras extensões que facilitam e customizam o editor para a necessidade de cada programador. Para baixar e instalar, primeiro acesse a página https://code.visualstudio.com/ e clique em “Download for Windows”. Clicando na seta à direita existem opções para MAC e Linux.



Instalando e configurando o VSCode

- Abra o instalador, e após aceitar o acordo de licença e clicar em “Próximo”, clique em “Próximo” novamente até chegar na tela “Selecionar Tarefas Adicionais”
- Marque as opções abaixo e clique em “Próximo” e depois em “Instalar”

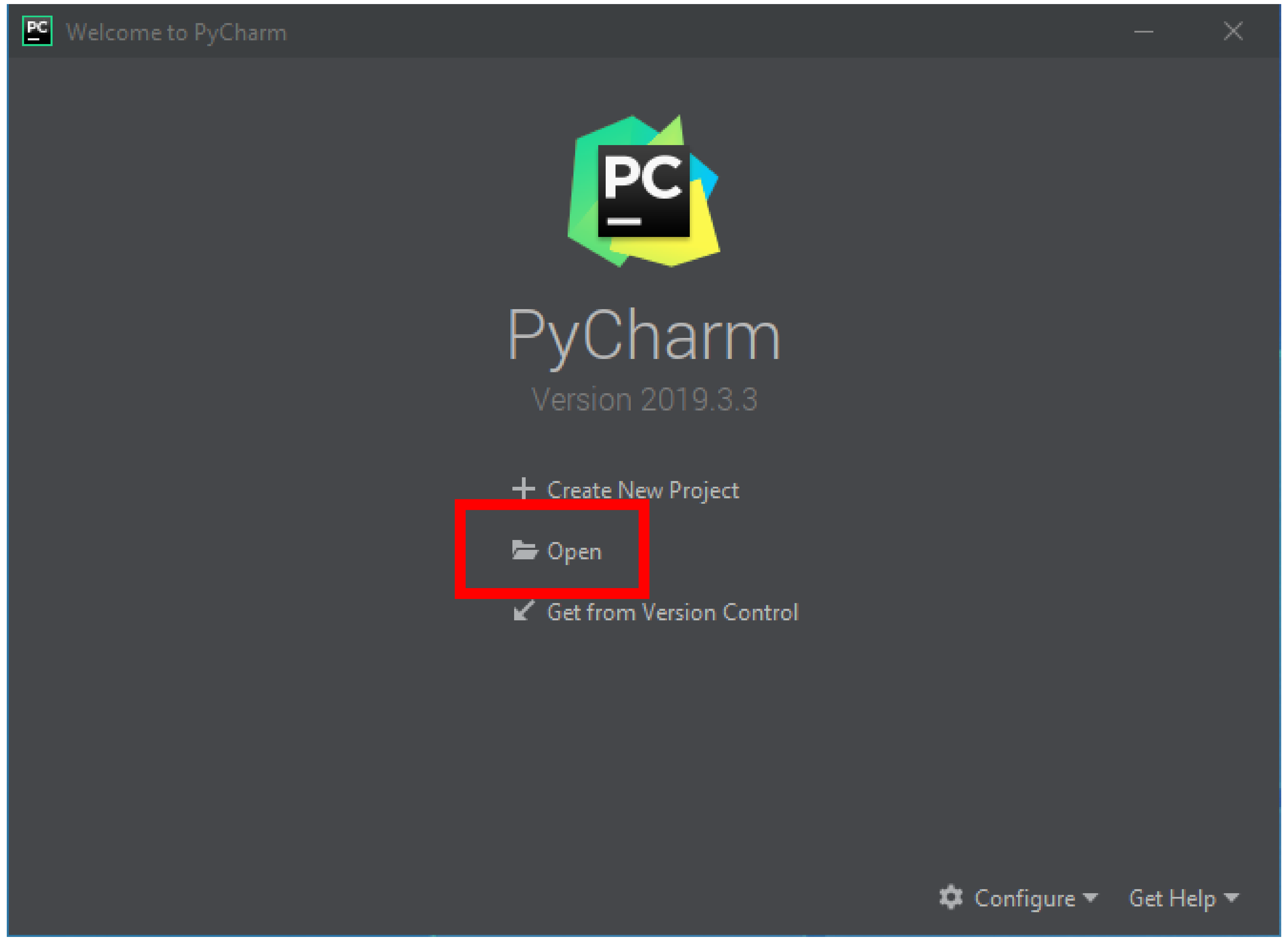


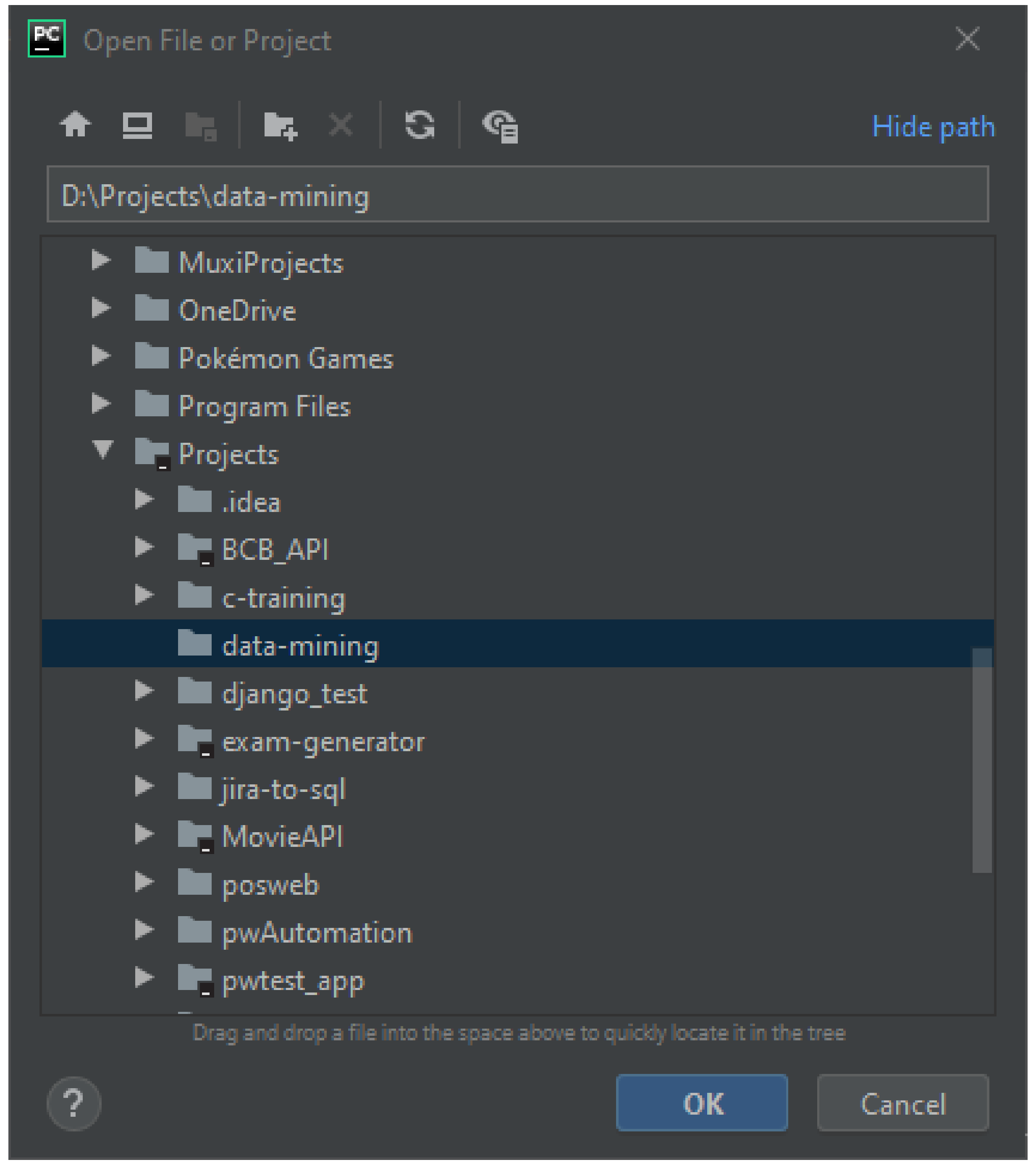
Ibmec
PROTAGONISTAS PARA O MUNDO

Utilizando o PyCharm

Iniciando o PyCharm

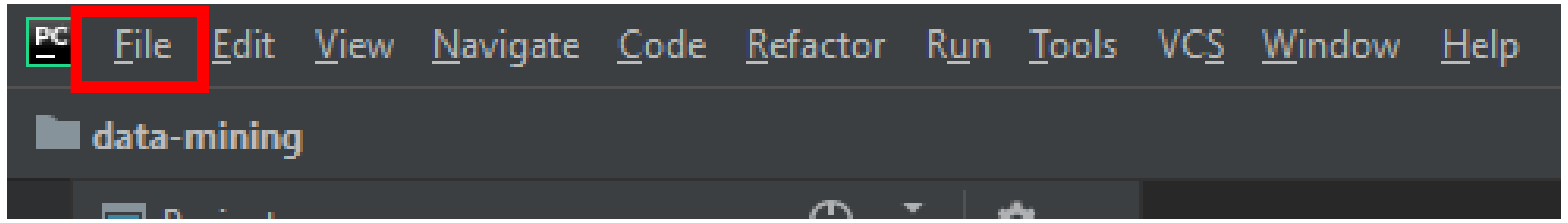
- **Se essa é a primeira vez que abre o PyCharm:**
  - Garanta que você possui uma pasta com o projeto desejado (p.ex., **D:\Projetos\data-mining**);
  - Abra o PyCharm, e na tela inicial clique em **Open**. Selecione a pasta que você criou e clique em **Ok**;

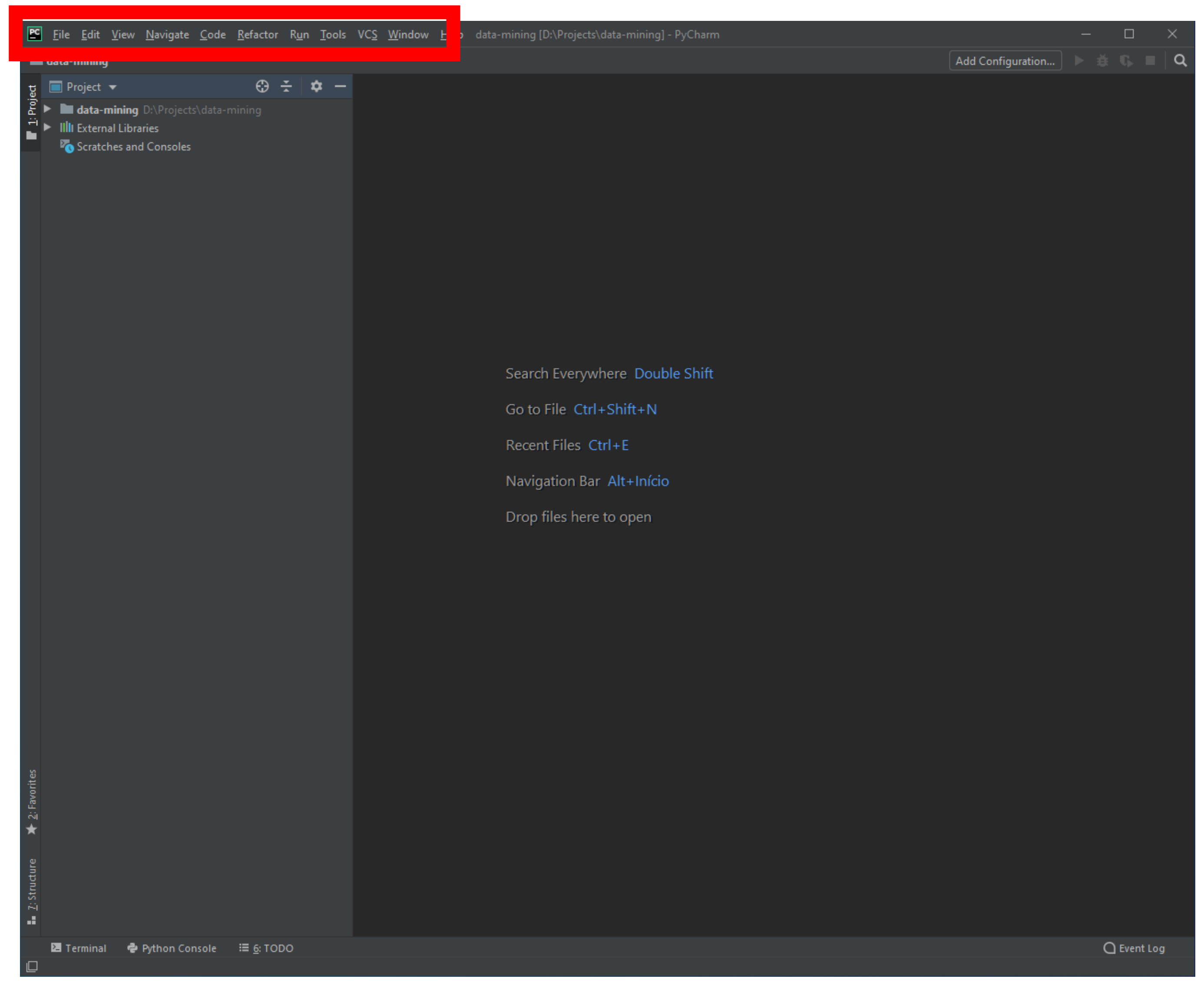


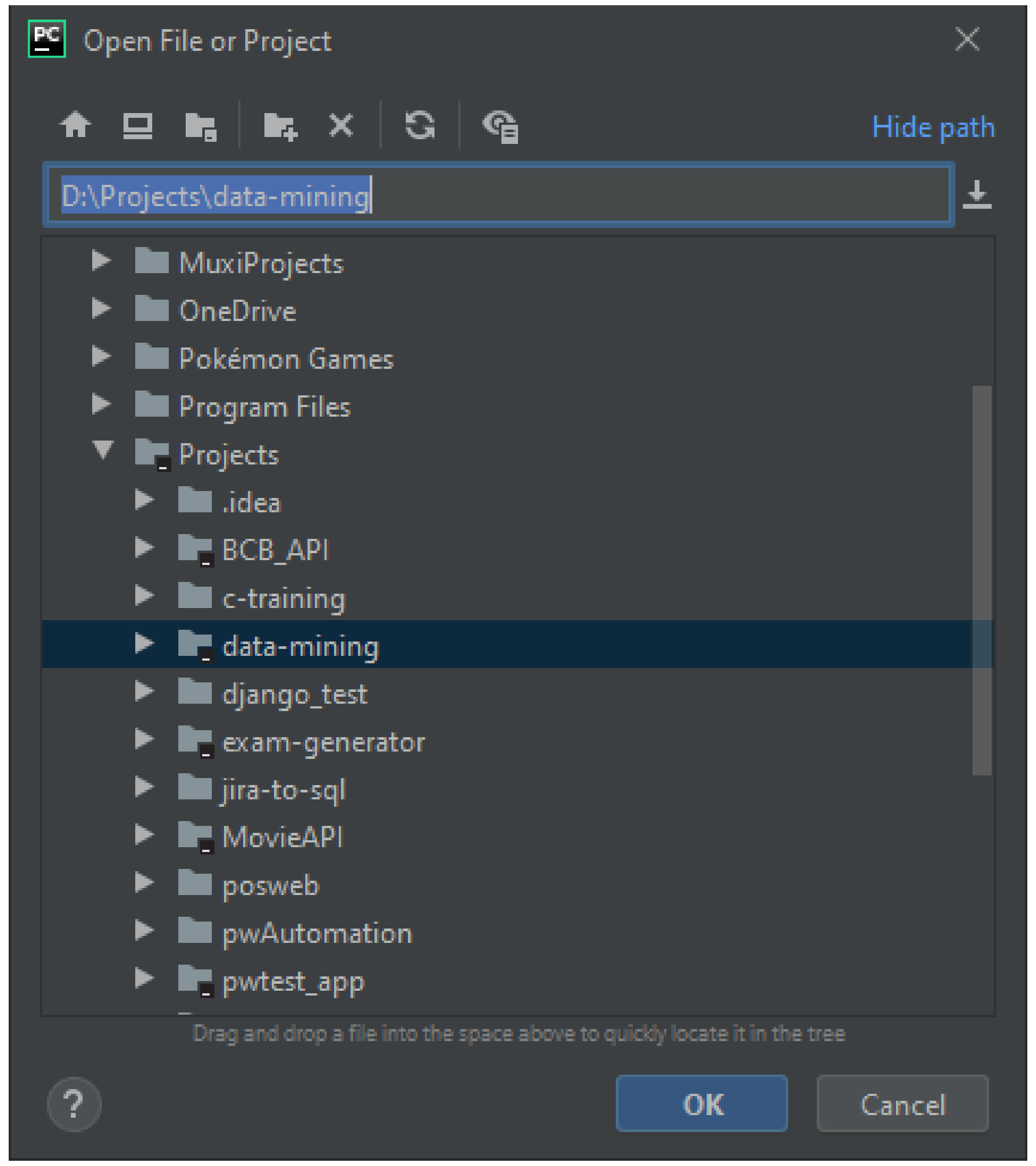


Iniciando o PyCharm

- **Se você já abriu o PyCharm antes:**
  - Garanta que você possui uma pasta com o projeto desejado (p.ex., **C:\Projetos\data-mining**);
  - Abra o PyCharm, e na tela que abrir clique em **File > Open**. Selecione a pasta que você criou e clique em **Ok**;





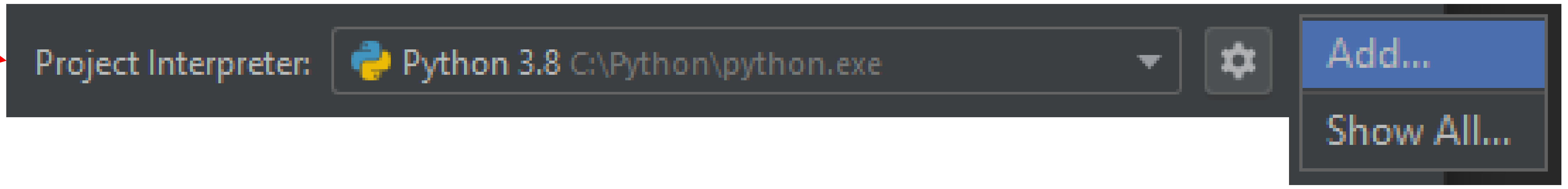


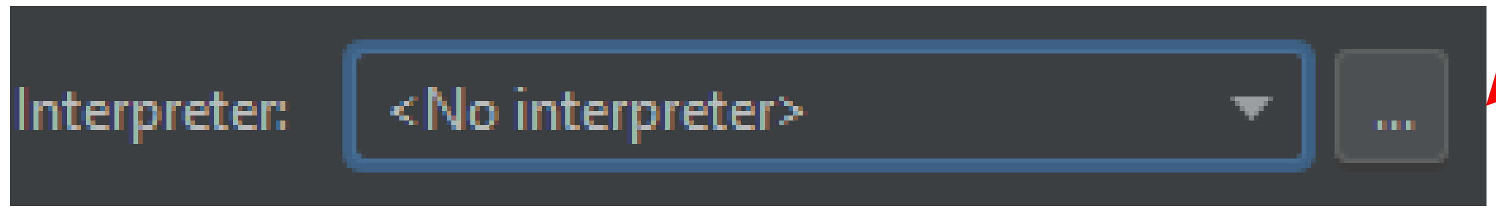
Algumas configurações do PyCharm

- **Mudar o tema para escuro:**
  - Clique em **File > Settings**;
  - Na janela que aparecer, procure por **Appearance & Behavior > Appearance**. No campo **Theme**, selecione o tema desejado (minha sugestão é o **Darcula**);
  - Clique em **Ok** para fechar a tela de configurações.

Algumas configurações do PyCharm

- **Configurar o interpretador de Python:**
  - Clique em **File > Settings**;
  - Na janela que aparecer, procure por **Project: <nome-do-projeto> > Project Interpreter**. No campo **Project Interpreter**, verifique se o Python informado é o instalado (no meu caso, **C:\Python\python.exe**). Caso não seja, clique na engrenagem e em **Add...**;
  - Na nova janela, clique em **System Interpreter**, e no campo **Interpreter** clique nos três pontos à direita para indicar o local que o seu Python está instalado. Aperte **Ok** até voltar à tela de **Settings**;
  - Novamente no campo **Project Interpreter**, altere o Python para refletir o que está instalado. Em seguida clique em **Ok**.



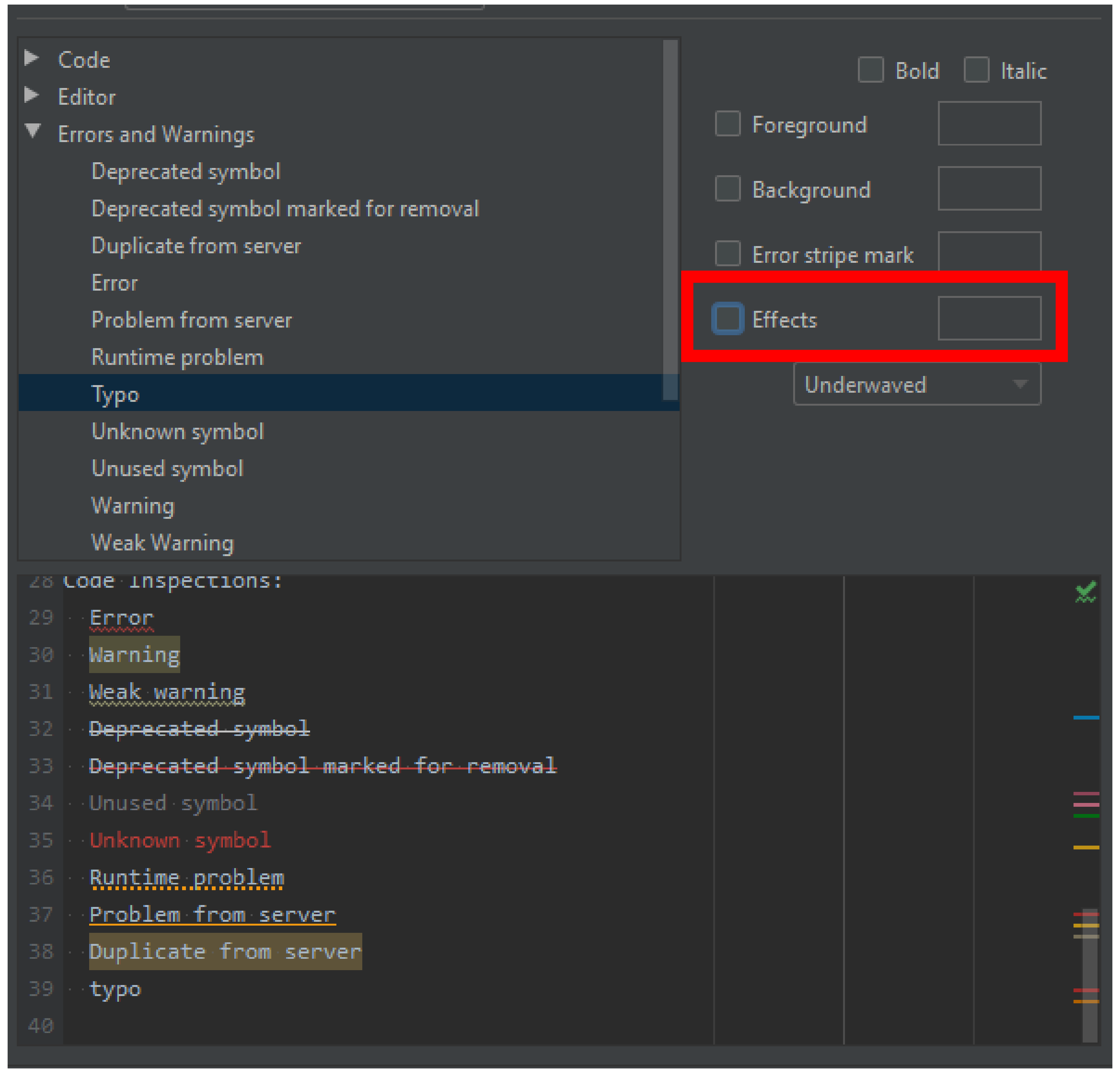


Algumas configurações do PyCharm

- **Desativar inspeção ortográfica:**
  - Clique em **File > Settings**;
  - Na janela que aparecer, procure por **Editor > Spelling**. No campo **Bundled dictionaries**, desmarque todas as opções;
  - Clique em **Ok** para sair das configurações.

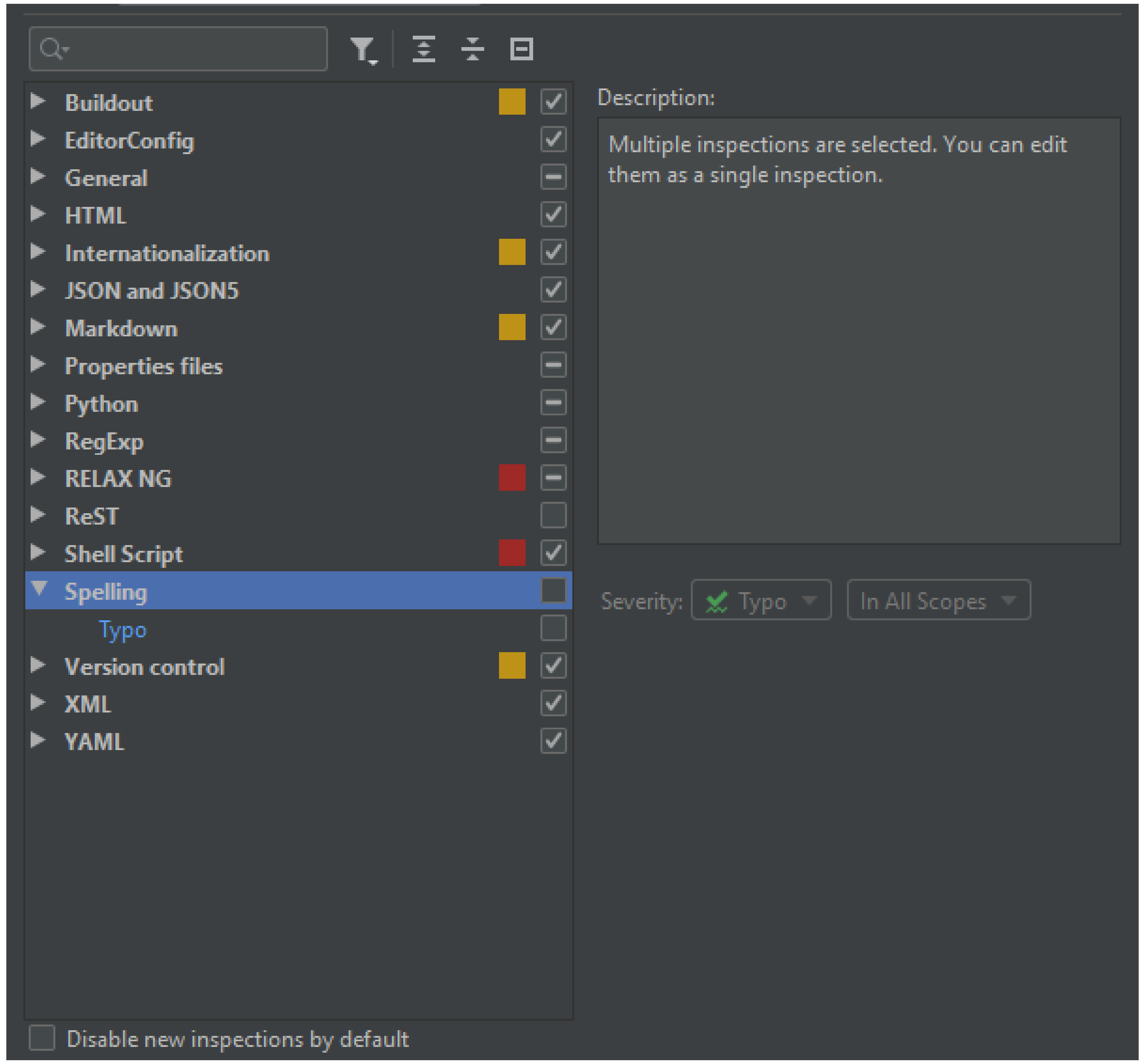
Algumas configurações do PyCharm

- **Desmarcar esquema de cores para *typos*:**
  - Clique em **File > Settings**;
  - Na janela que aparecer, procure por **Editor > Color Scheme > General**. No campo **Errors and Warnings > Typo**, desmarque a opção **Effects**;
  - Clique em **Ok** para sair das configurações.



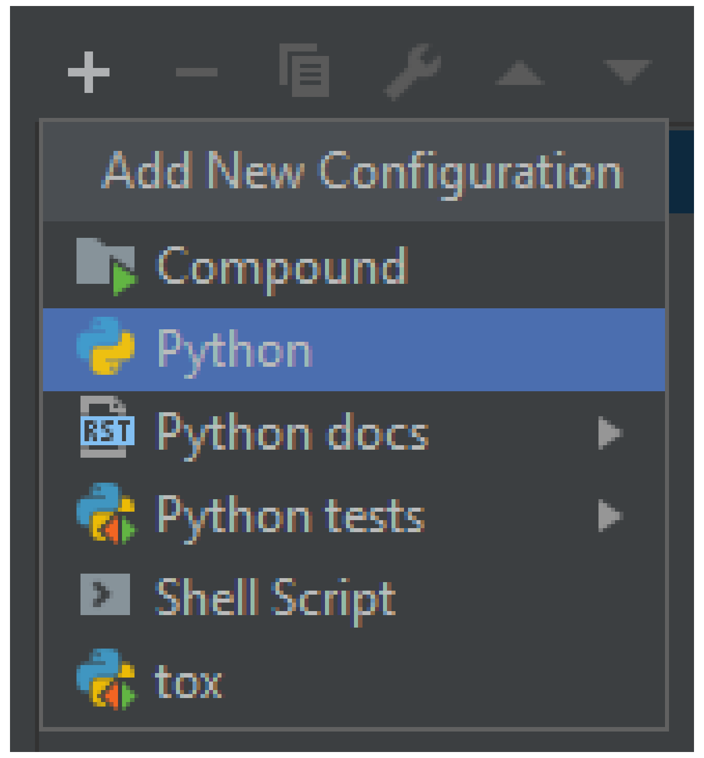
Algumas configurações do PyCharm

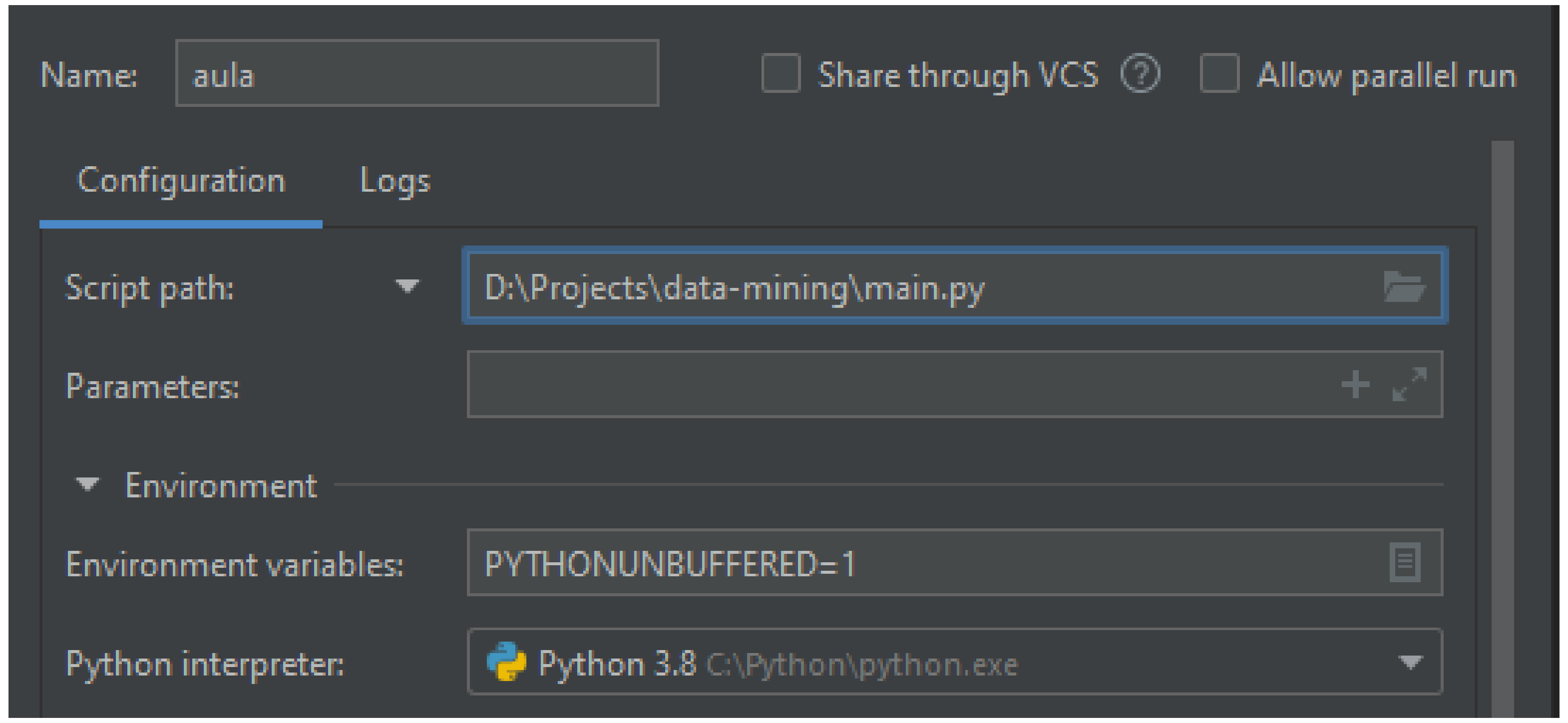
- **Ajustar os destaques de inspeções:**
  - Clique em **File > Settings**;
  - Na janela que aparecer, procure por **Editor > Inspections**. Desmarque o campo **Spelling**;
  - Na seção **Python**, recomendo *desmarcar* algumas opções para simplificar o aprendizado:
    - *Boolean variable check can be simplified;*
    - *Chained comparisons can be simplified;*
    - *Comparison with None performed with equality operators.*
  - Clique em **Ok** para sair das configurações.



Algumas configurações do PyCharm

- **Configurar uma determinada execução de código:**
  - Garanta que as configurações do slide **Configurar o interpretador de Python** foram executadas;
  - Aperte **Alt + Shift + F10** > **Edit Configurations**, ou na barra de tarefas clique em **Run > Edit Configurations**;
  - Na janela que aparecer, no canto superior esquerdo clique no botão de **+ > Python**;
  - Dê um nome para a execução (p.ex., **aula**) e em **Script path**, informe o caminho do arquivo que você deseja executar;
  - Certifique-se que o **Python interpreter** é o configurado anteriormente, clique em **Apply** e em seguida em **Close**;
  - Para rodar o arquivo, é só usar o atalho **Shift + F10**.



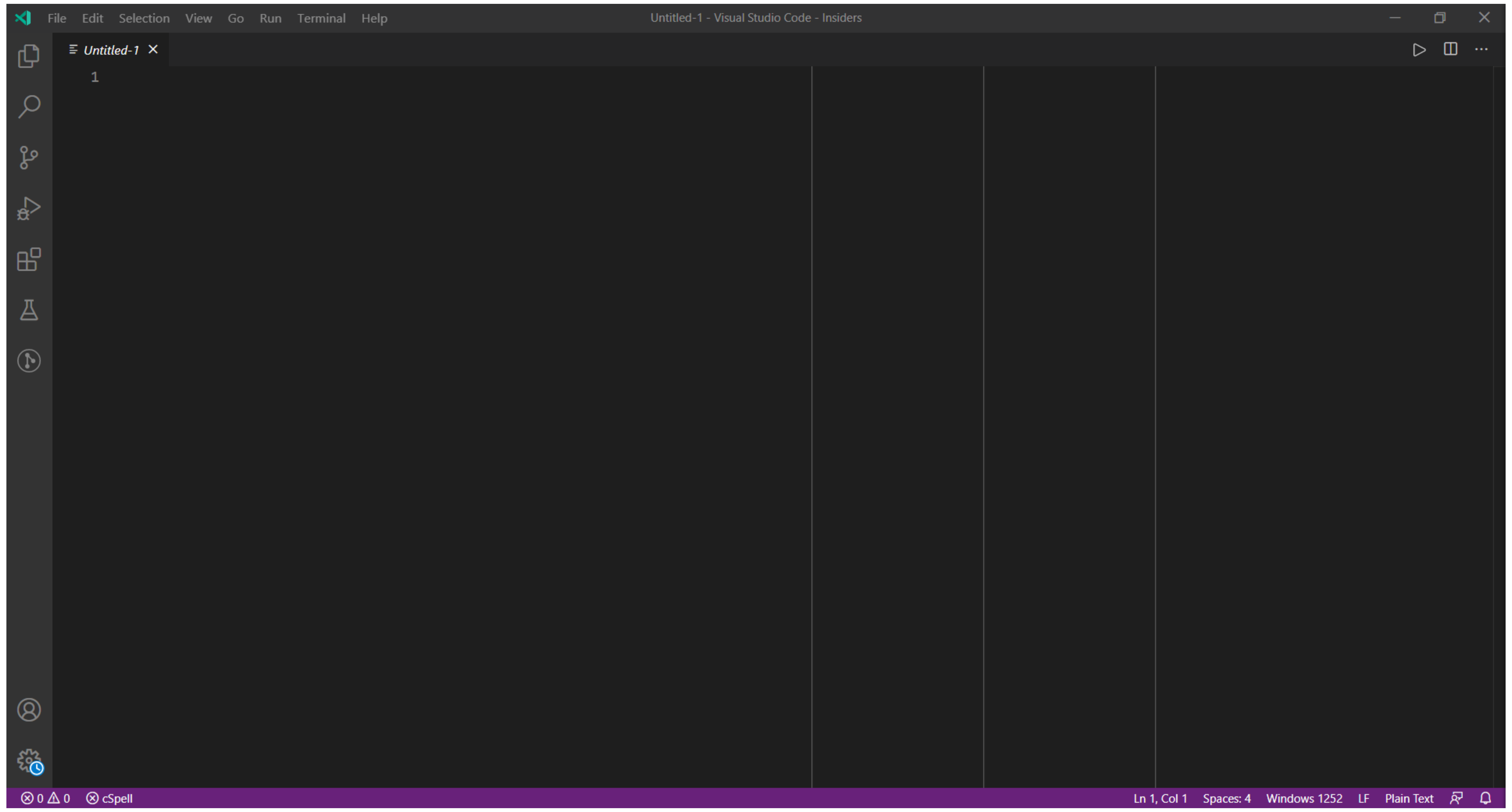


Utilizando o VSCode

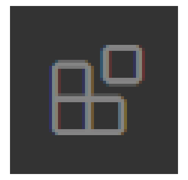
Ibmec
PROTAGONISTAS PARA O MUNDO

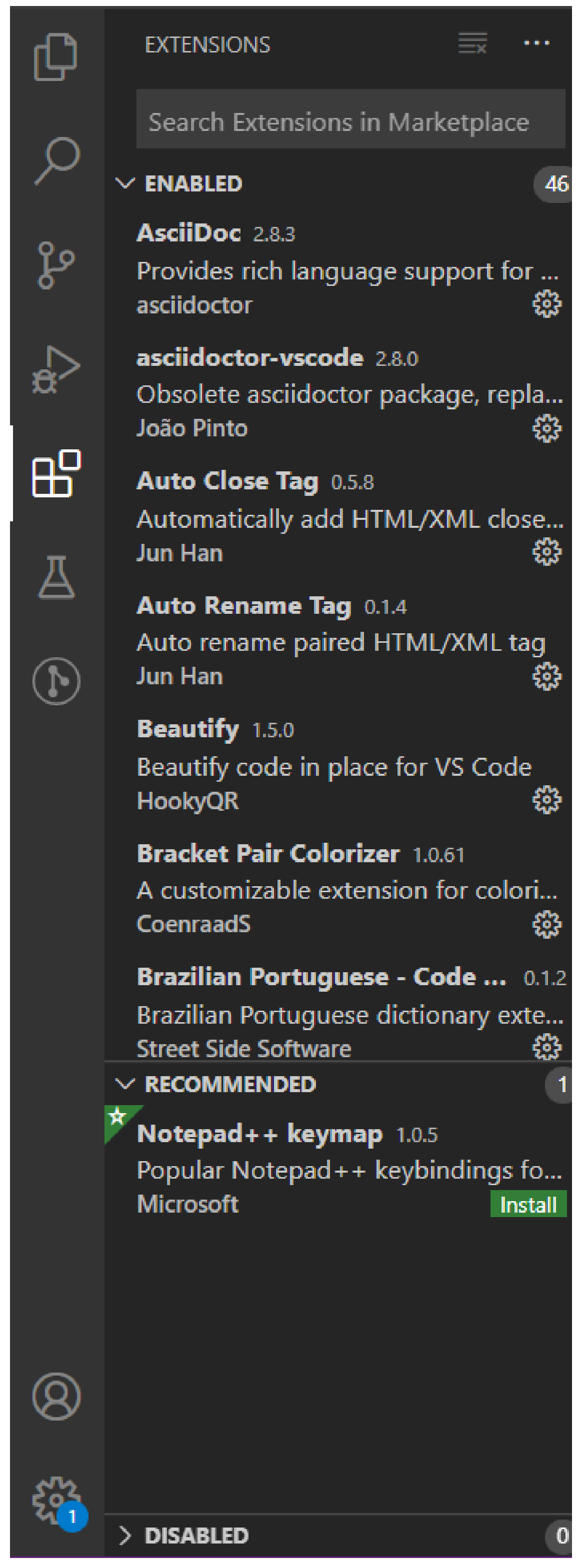
Iniciando o VSCode

- **Se essa é a primeira vez que abre o VSCode:**
  - Garanta que você possui uma pasta com o projeto desejado (p.ex., **D:\Projetos\algoritmos**);
  - Abra o VSCode, e na tela inicial clique em **File > Open Folder**. Selecione a pasta que você criou e clique em **Selecionar Pasta**.



Iniciando o VSCode

- **Instalando extensões:**
  - Boa parte do atrativo no VSCode é a possibilidade de instalar extensões e customizar a experiência de uso;
  - Para instalar extensões, vá na coluna à esquerda e clique no ícone  ;
  - Uma barra de extensões vai aparecer, e você poderá buscar as extensões desejadas e instalá-las;
  - Abaixo seguem algumas sugestões de extensões para o curso:
    - Beautify
    - Bracket Pair Colorizer
    - Brazilian Portuguese - Code Spell Checker
    - C/C++
    - Code Runner
    - GitLens - Git supercharged
    - Git History Diff
    - Git History
    - Gitconfig Syntax
    - Markdownlint
    - Partial Diff
    - Python
    - Visual Studio IntelliCode
    - vscode-python-docstring
    - AsciiDoc



Iniciando o VSCode

- **Atalhos interessantes no VSCode:**
  - Ctrl + K, Ctrl + O: Abre uma pasta
  - Ctrl + D: Quando uma palavra estiver selecionada, seleciona todas as palavras no arquivo
  - Ctrl + F: Procura por uma palavra ou sentença no arquivo
  - Ctrl + Shift + F: Procura por uma palavra ou sentença em todos os arquivos da pasta
  - Ctrl + H: Substitui uma palavra ou sentença por outra em todo o arquivo
  - Alt + ↓ ou Alt + ↑: Move a linha inteira para baixo ou para cima
  - Shift + Alt + ↓ ou Shift + Alt + ↑: Copia a linha inteira para baixo ou para cima
  - Ctrl + Alt + ↓ ou Ctrl + Alt + ↑: Incluí um ou mais cursores nas linhas abaixo ou acima

Ibmec
PROTAGONISTAS PARA O MUNDO

Iniciando o VSCode

- **Atalhos interessantes no VSCode:**
  - F12: Vai para a declaração de uma variável ou função
  - F12 F12: Mostra todos os usos de uma variável ou função
  - Alt + → ou Alt + ←: Retrocede ou avança na última posição do cursor
  - Ctrl + S: Salva um arquivo
  - Ctrl + P: Abre um arquivo diretamente
  - Ctrl + ,: Abre o menu de configurações
  - Ctrl + Shift + P: Abre uma dropdown com opções de configuração
  - Ctrl + ‘: Abre a janela do terminal

Ibmec
PROTAGONISTAS PARA O MUNDO

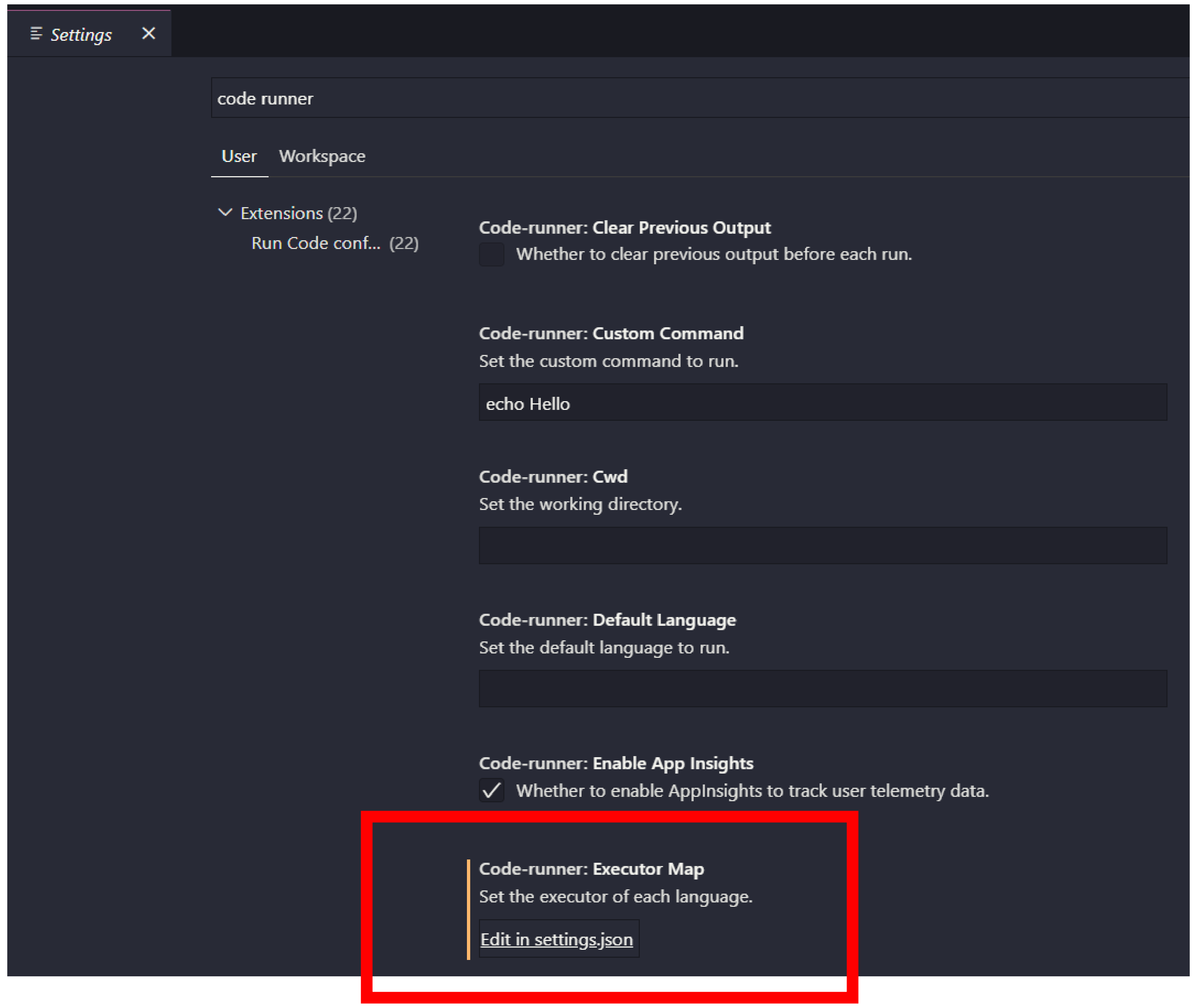
Iniciando o VSCode

- **Atalhos interessantes no VSCode:**
  - Ctrl + G: Vai para uma linha específica do arquivo
  - Ctrl +]: Divide a tela em duas
  - Ctrl + 1 (ou 2, 3, 4): Seleciona um painel específico
  - Alt + Shift + 0: Alterna entre divisão na horizontal ou na vertical
  - Ctrl + F4 ou Ctrl + W: Fecha o arquivo selecionado (se for um painel selecionado e ele estiver vazio, fecha o painel)
  - Ctrl + ;: Comenta a(s) linha(s) selecionada(s)

Iniciando o VSCode

- **Executando um código Python no VSCode com a extensão *Code Runner*:**
  - Essa extensão permite a execução do código através de um atalho
  - Com a extensão instalada, use o atalho **Ctrl + ,** ou vá em **File > Preferences > Settings**;
  - Na barra de pesquisa, digite “Code Runner”. Procure pela caixa abaixo e clique em “Edit in settings.json”;
  - Inclua as seguintes linhas (edite o endereço para o caminho em que o Python está instalado):

```
"python.pythonPath": "C:\\Python\\python.exe",
"code-runner.executorMap": {
    "python": "\"C:\\Python\\python.exe\""
},
"code-runner.runInTerminal": true,
```



  - Para rodar, na janela do código use o atalho **Ctrl + Alt + N**.

Configurando o pylint

PROTAGONISTAS PARA O MUNDO

Configurando o pylint

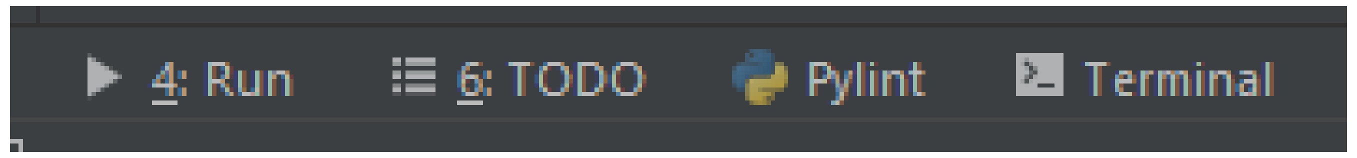
- O pylint é um pacote do Python que auxilia na adequação do código aos padrões de programação da comunidade Python
- Para instalar o pylint...
  - ...para MacOS veja [aqui]
  - ...para Windows veja [aqui]
- Para executar o pylint basta entrar com o comando **pylint <caminho**>, com o caminho do arquivo que deseja rodar o linter.

```
C:\Users\vmachado>pylint D:\Victor\Pessoal\IBMEC\2020.2\ALG\Aulas\teste.py
************* Module teste
D:\Victor\Pessoal\IBMEC\2020.2\ALG\Aulas\teste.py:1:0: C0114: Missing module docstring (missing-module-docstring)

-----------------------------------
Your code has been rated at 0.00/10
```

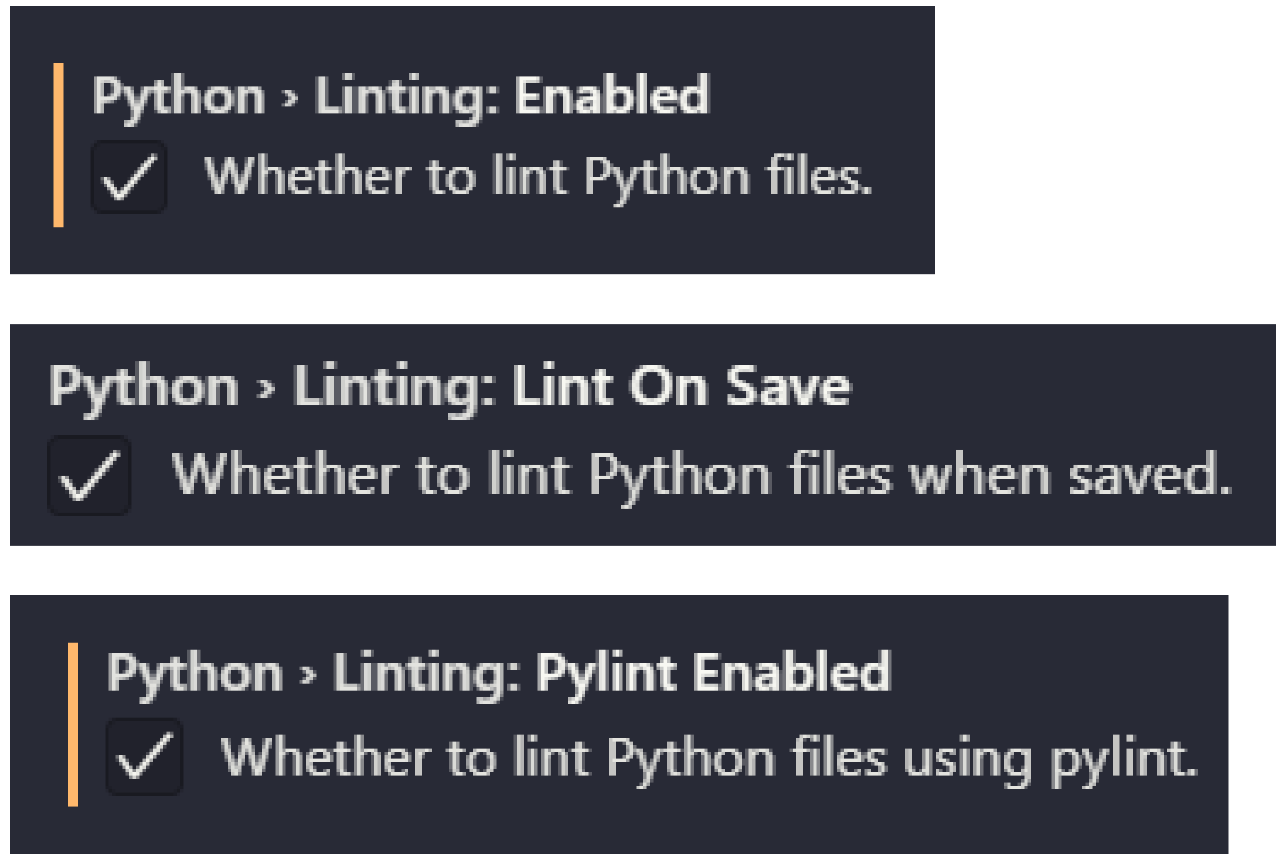
Configurando o pylint

- O pylint ainda pode ser rodado direto no terminal do VSCode ou do PyCharm, basta abrir o terminar e seguir os mesmos passos mencionados anteriormente. No entanto, os ambos os IDEs fornecem meios de utilizar o pylint diretamente durante a implementação do código.
- **Usando o pylint no PyCharm:**
  - O PyCharm já possui a inspeção automática do editor, após configurar o interpretador. Para identificar as inspeções vá em **File > Settings** e na janela clique em **Editor** e depois em **Inspections**;
  - O PyCharm também possui um plugin. Em **File > Settings**, clique em **Plugins** e busque por **Pylint**. Instale e reinicie o IDE. Sempre que abrir um arquivo Python o IDE vai incluir a opção “Pylint” no canto inferior esquerdo. Para executar basta clicar na opção e depois em “Run”.



Configurando o pylint

- **Usando o pylint no VSCode:**
    - Com a extensão “Python” instalada no VSCode e o pacote pylint instalado, abra o menu de configurações (use o atalho **Ctrl + ,** ou vá em **File > Preferences > Settings**) e procure por python linting;
    - Marque as seguintes opções:
        - Python > Linting: Enabled
        - Python > Linting: Lint On Save
        - Python > Linting: Pylint Enabled
    - Para rodar o pylint sem ser pelo terminal, use o atalho **Ctrl + Shift + P** e em seguida digite “Run Linting” e aperte Enter. O IDE vai marcar no código os problemas.



Instalando pacotes pelo PyPI

Instalando pacotes pelo PyPI

- Uma das grandes vantagens ao se programar em Python é ter à disposição uma gama de pacotes e bibliotecas disponíveis pela própria comunidade, que desenvolve novas funcionalidades e distribui online, na maioria das vezes de forma gratuita.
- Um dos locais mais confiáveis e mais simples de se obter um novo pacote é através do PyPI, um repositório oficial de pacotes da linguagem, mantido pela própria organização que mantém o Python.
- O PyPI é acessado utilizando uma ferramenta chamada **pip**, que é instalada automaticamente ao se instalar o interpretador de Python. Portanto, a instalação de novos pacotes é muito fácil de se realizar, com apenas uma linha de comando.

Instalando pacotes pelo PyPI

- Instalando pacotes via pip no Windows:
  - Clique no botão do Windows ;
  - Digite a caixa de pesquisa **prompt de comando**, e abra o programa;
  - No terminal entre com o seguinte comando:

```
C:\Users\vmachado>C:\Python\python.exe -m pip install pylint
```

  - Lembre-se de substituir o caminho do Python para o caminho instalado no seu computador, e altere **pylint** para o pacote escolhido.

Instalando pacotes pelo PyPI

- A instalação do Python no MacOS não costuma vir com o pip. Caso não tenha vindo, tente fazer os passos abaixo primeiro:
  - Abra o terminal (pasta **Applications > Utilities > Terminal**);
  - Entre com os seguintes comandos:

```
curl https://bootstrap.pypa.io/get-pip.py > get-pip.py
sudo python get-pip.py
```

- Para instalar um pacote via pip no MacOS:
  - Usando o mesmo terminal usado para instalar o pip, entre com o seguinte comando:

```
sudo pip install <nome_do_pacote>
```

- Veja o vídeo abaixo para mais detalhes:
  - https://www.youtube.com/watch?v=yBdZZGPpYxg

Ibmec
PROTAGONISTAS PARA O MUNDO

Algoritmos

Introdução

- Um algoritmo é um processo sistemático para a resolução de um problema. O desenvolvimento de algoritmos é particularmente importante para problemas a serem solucionados em um computador, pela própria natureza do instrumento utilizado.
- Um algoritmo computa uma saída, o resultado do problema, a partir de uma entrada, as informações inicialmente conhecidas e que permitem encontrar a solução do problema. Durante o processo de computação o algoritmo manipula dados, gerados a partir da sua entrada.
- O estudo de estruturas de dados não pode ser desvinculado de seus aspectos algorítmicos. A escolha correta da estrutura adequada a cada caso depende diretamente do conhecimento de algoritmos para manipular a estrutura de maneira eficiente.

Ibmec
PROTAGONISTAS PARA O MUNDO

Apresentação dos algoritmos

- As convenções seguintes serão utilizadas com respeito à linguagem:
  - O início e o final de cada bloco são determinados por endentação, isto é, pela posição da margem esquerda. Se uma certa linha do algoritmo inicia um bloco, ele se estende até a última linha seguinte, cuja margem esquerda se localiza mais à direita do que a primeira do bloco;
  - A declaração de atribuição é indicada pelo símbolo :=;
  - As declarações seguintes são empregadas com significado semelhante ao usual:

```
se... então
se... então... senão
enquanto... faça
para... faça
pare
```

  - Variáveis simples, vetores, matrizes e registro são considerados como tradicionalmente em linguagens de programação. Os elementos de vetores e matrizes são identificados por índices entre colchetes.

```
para i := 1, ..., |__n/2__|
    temp := S[i]
    S[i] := S[n - i + 1]
    S[n - i + 1] := temp
```

Recursividade

- Um tipo especial de procedimento será utilizado, algumas vezes, ao longo do curso. É aquele que contém, em sua descrição, uma ou mais chamadas a si mesmo. Um procedimento dessa natureza é denominado **recursivo**.
- Naturalmente, todo procedimento, recursivo ou não, deve possuir pelo menos uma chamada proveniente de um local exterior a ele. Essa chamada é denominada **externa**.
- Um procedimento não recursivo é, pois, aquele em que todas as chamadas são externas.

Recursividade

- O exemplo clássico mais simples de recursividade é o cálculo do fatorial de um inteiro $n \geq 0$:

```
função fat(i)
    fat(i) := se i <= 1 então 1 senão i * fat(i - 1)
```

```
fat[0] := 1
para j := 1, ..., n faça
    fat[j] := j * fat[j - 1]
```

- Um exemplo conhecido, onde a solução recursiva é natural e intuitiva, é o do **[Problema da Torre de Hanói](#)**.

Recursividade

- A solução do problema é descrita a seguir. Naturalmente, para $n > 1$, o pino-trabalho deve ser utilizado como área de armazenamento temporário. O raciocínio utilizado para resolver o problema é semelhante ao de uma prova matemática por indução. Suponha que se saiba como resolver o problema até $n - 1$ discos, $n > 1$, de forma recursiva. A extensão para n discos pode ser obtida pela realização dos seguintes passos:
  - Resolver o problema da Torre de Hanói para os $n - 1$ discos do topo do pino-origem A, supondo que o pino-destino seja C e o trabalho seja B;
  - Mover o n-ésimo pino (maior de todos) de A para B;
  - Resolver o problema da Torre de Hanói para os $n - 1$ discos localizados no pino C, suposto origem, considerando os pinos A e B como trabalho e destino, respectivamente.

```
procedimento hanoi(n, A, B, C)
se n > 0 então
    hanoi(n - 1, A, C, B)
    mover o disco do topo de A para B
    hanoi(n - 1, C, B, A)
```

Complexidade de algoritmos

- Conforme já mencionado, uma característica muito importante de qualquer algoritmo é o seu tempo de execução. Naturalmente, é possível determiná-lo através de métodos empíricos, isto é, obter o tempo de execução através da execução propriamente dita do algoritmo, considerando-se entradas diversas.
- Ao contrário do método empírico, o método analítico visa aferir o tempo de execução de forma independente do computador utilizado, da linguagem e dos compiladores empregados e das condições locais de processamento.

Complexidade de algoritmos

- As seguintes simplificações serão introduzidas para o modelo proposto:
  - Suponha que a quantidade de dados a serem manipulados pelo algoritmo seja suficientemente grande. Somente o comportamento assintótico será avaliado.
  - Não serão consideradas constantes aditivas ou multiplicativas na expressão matemática obtida. Isto é, a expressão matemática obtida será válida, a menos de tais constantes.
- O processo de execução de um algoritmo pode ser dividido em etapas elementares, denominadas *passos*. Cada passo consiste na execução de um número fixo de operações básicas cujos tempos de execução são considerados constantes.

```
para i := 1, ..., |__n/2__|
    temp := S[i]
    S[i] := S[n - i + 1]
    S[n - i + 1] := temp
```

Ibmec
PROTAGONISTAS PARA O MUNDO

Complexidade de algoritmos

- Como exemplos adicionais, considere os problemas de determinar as matrizes soma C e produto D de duas matrizes dadas.

```
para i := 1, ..., n faça
    para j := 1, ..., n faça
        C[i][j] := A[i][j] + B[i][j]
```

```
para i:= 1, ..., n faça
    para j := 1, ..., n faça
        C[i][j] := 0
        para k := 1, ..., n faça
            C[i][j] := C[i][j] + A[i][k] * B[k][j]
```

Ibmec
PROTAGONISTAS PARA O MUNDO

Complexidade de algoritmos

- Noção de complexidade:
  - Seja A um algoritmo, $\{E_1, \ldots, E_m\}$, o conjunto de todas as entradas possíveis de A. Denote por t_i o número de passos efetuados por A, quando a entrada for E_i. Definem-se, com p_i sendo a probabilidade de ocorrência da entrada E_i:
    - Complexidade do pior caso: $\max_{E_{i \in E}}\{t_i\}$;
    - Complexidade do melhor caso: $min_{E_{i \in E}}\{t_i\}$;
    - Complexidade do caso médio: $\sum_{1 \leq i \leq m}(p_i \times t_i)$.
  - As complexidades têm por objetivo avaliar a eficiência de tempo ou espaço. A complexidade de tempo de pior caso corresponde ao número de passos que o algoritmo efetua no seu pior caso de execução, isto é, para a entrada mais desfavorável. De certa forma, a complexidade de pior caso é a mais importante das três mencionadas.

A Notação O

- Quando se considera o número de passos efetuados por um algoritmo, podem-se desprezar constantes aditivas ou multiplicativas.
- Por exemplo, um valor de número de passos igual a 3n será aproximado para n.
- Além disso, como o interesse é restrito a valores assintóticos, termos de menor grau também podem ser desprezados. Assim, um valor de número de passos igual a $n^2 + n$ será aproximado para n^2. O valor $6n^3 + 4n - 9$ será transformado em n^3.
- Torna-se útil, portanto, descrever operadores matemáticos que sejam capazes de representar situações como essas. A notação O será utilizada com essa finalidade.

Ibmec
PROTAGONISTAS PARA O MUNDO

A Notação O

- Sejam f, h funções reais positivas de variável inteira n. Diz-se que f é $O(h)$, escrevendo-se $f = O(h)$, quando existir uma constante $c > 0$ e um valor inteiro n_o, tal que:

$$n > n_o \Rightarrow f(n) \leq c \times h(n)$$

- Ou seja, a função h atua como um limite superior para valores assintóticos da função f. Em seguida são apresentados alguns exemplos da notação O.

$$f = n^2 - 1 \Rightarrow f = O(n^2)$$

$$f = n^3 - 1 \Rightarrow f = O(n^3)$$

$$f = 403 \Rightarrow f = O(1)$$

$$f = 5 + 2 \log n + 3 \log^2 n \Rightarrow f = O(\log^2 n)$$

Listas Lineares

PROTAGONISTAS PARA O MUNDO

Introdução

- Dentre as estruturas de dados não primitivas, as listas lineares são as de manipulação mais simples. Iremos discutir seus algoritmos e estruturas de armazenamento.
- Uma lista linear agrupa informações referentes a um conjunto de elementos que, de alguma forma, se relacionam entre si. Ela pode se constituir, por exemplo, de informações sobre os funcionários de uma empresa, sobre notas de compras, itens de estoque, notas de alunos, etc.
- Uma *lista linear*, ou *tabela*, é então um conjunto de $n \geq 0$ nós L[1], L[2], ..., L[n] tais que suas propriedades estruturais decorrem, unicamente, da posição relativa dos nós dentro da sequência linear.

Ibmec
PROTAGONISTAS PARA O MUNDO

Introdução

- As operações mais frequentes em listas são a *busca*, a *inclusão* e a *remoção* de um determinado elemento, o que, aliás, ocorre na maioria das estruturas de dados. Tais operações podem ser consideradas como básicas e, por essa razão, é necessário que os algoritmos que as implementem sejam eficientes.
- Outras operações também são relevantes, porém não serão estudadas a fundo neste curso:
  - Alteração de um elemento da lista;
  - Combinação de duas ou mais listas lineares.
  - Ordenação dos nós segundo um determinado campo;
  - Determinação do primeiro (ou do último) nó da lista;
  - etc.

Introdução

- As operações mais frequentes em listas são a *busca*, a *inclusão* e a *remoção* de um determinado elemento, o que, aliás, ocorre na maioria das estruturas de dados. Tais operações podem ser consideradas como básicas e, por essa razão, é necessário que os algoritmos que as implementem sejam eficientes.
- Outras operações também são relevantes, porém não serão estudadas a fundo neste curso:
  - Alteração de um elemento da lista;
  - Combinação de duas ou mais listas lineares.
  - Ordenação dos nós segundo um determinado campo;
  - Determinação do primeiro (ou do último) nó da lista;
  - etc.

Ibmec
PROTAGONISTAS PARA O MUNDO

Introdução

- Casos particulares de listas são de especial interesse:
  - Se as inserções e remoções são permitidas apenas nas extremidades da lista, ela recebe o nome de **deque** (uma abreviatura do inglês *double ended queue*);
  - Se as inserções e as remoções são realizadas somente em um extremo, a lista é denominada **pilha**;
  - A lista é denominada **fila** no caso em que as inserções são realizadas em um extremo e remoções em outro.

Introdução

- O tipo de armazenamento de uma lista linear pode ser classificado de acordo com a posição relativa na memória de dois nós consecutivos na lista:
  - Quando dois nós consecutivos na lista são alocados contiguamente na memória do computador, a lista é classificada como de *alocação sequencial de memória*;
  - Quando dois nós consecutivos não são alocados contiguamente, a lista é classificada como de alocação encadeada de memória.
- A escolha de um ou outro tipo depende essencialmente das operações que serão executadas sobre a lista, do número de listas envolvidas na operação, bem como de características particulares.

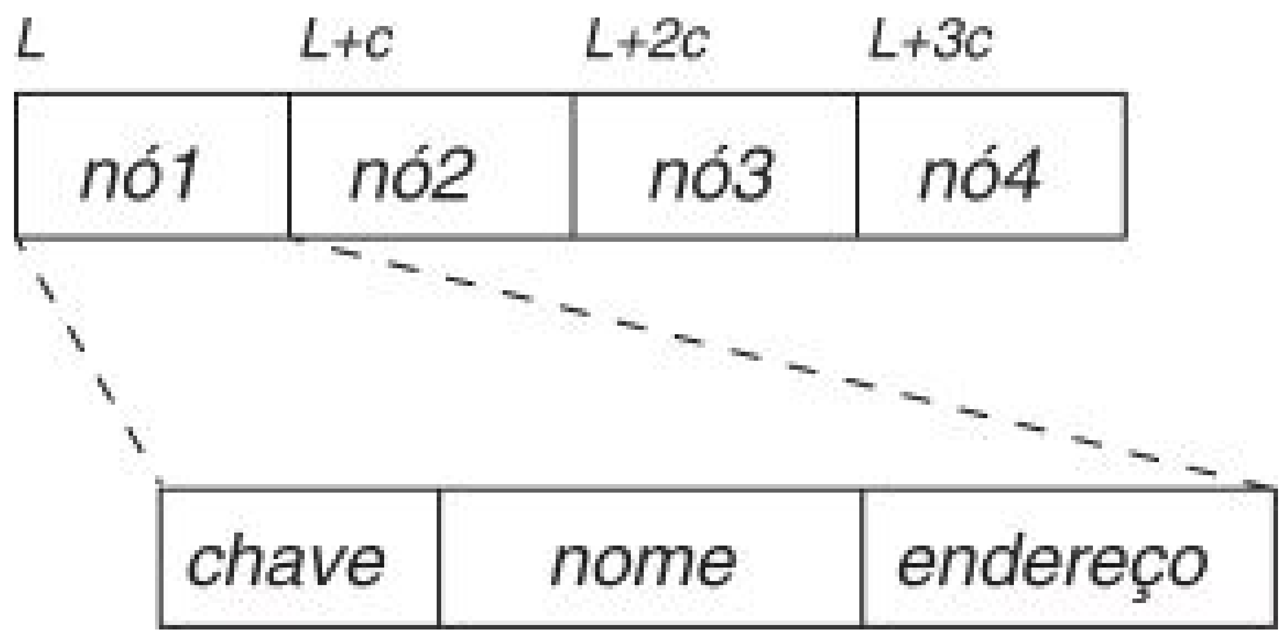
Ibmec
PROTAGONISTAS PARA O MUNDO

Alocação sequencial

- A maneira mais simples de se manter uma lista linear na memória do computador é colocar seus nós em posições contíguas.
- O armazenamento sequencial é particularmente atraente no caso de filas e pilhas porque, nessas estruturas, as operações básicas podem ser implementadas de forma bastante eficiente.
- Esse tratamento pode, contudo, se tornar oneroso em termos de memória quando se empregam diversas estruturas simultaneamente. Nesse caso, a utilização ou não do armazenamento sequencial dependeria de um estudo cuidadoso das opções existentes.

Listas lineares em alocação sequencial

- Seja uma lista linear. Cada nó é formado por *campos*, que armazenam as características distintas dos elementos da lista. Além disso, cada nó da lista possui, geralmente, um identificador, denominado *chave*. A chave, quando presente, se constitui em um dos campos do nó. Os nós podem se encontrar ordenados, ou não, segundo os valores de suas chaves. No primeiro caso a lista é denominada *ordenada*, e *não ordenada* no caso contrário.



Listas lineares em alocação sequencial

- Observe que, para cada elemento da tabela referenciado na busca, o algoritmo realiza dois testes.
- A complexidade de pior caso é $O(n)$.

```
função busca1(x)
    i := 1
    busca1 := 0
    enquanto i <= n faça
        se L[i].chave = x então
            % chave encontrada
            busca1 := i
            i := n + 1
        senão
            % pesquisa prossegue
            i:= i + 1
```

Ibmec
PROTAGONISTAS PARA O MUNDO

Listas lineares em alocação sequencial

- Quando a lista está ordenada, pode-se tirar proveito desse fato. Se o número procurado não pertence à lista, não há necessidade de percorrê-la até o final.
- Podemos criar um novo nó ao final da lista para evitar a dupla comparação do algoritmo anterior. No entanto, a complexidade de pior caso ainda é $O(n)$.

```
função busca-ord(x)
    L[n + 1].chave := x
    i := 1
    enquanto L[i].chave < x faça
        i := i + 1
    se i = n + 1 ou L[i].chave != x então
        busca-ord := 0
    senão
        busca-ord := i
```

Listas lineares em alocação sequencial

- Ainda no caso das listas ordenadas, um algoritmo diverso e bem mais eficiente pode ser apresentado: a *busca binária*. Em tabelas, o primeiro nó pesquisado é o que se encontra no meio; se a comparação não é positiva, metade da tabela pode ser abandonada na busca, uma vez que o valor procurado se encontra ou na metade inferior, ou na superior. Esse procedimento, aplicado recursivamente, esgota a tabela.

```
função busca-bin(x)
    inf := 1
    sup := n
    busca-bin := 0
    enquanto inf <= sup faça
        % índice a ser buscado
        meio := |__(inf + sup) / 2__|
        se L[meio].chave = x então
            % elemento encontrado
            busca-bin := meio
            inf := sup + 1
        senão se L[meio].chave < x então
            inf := meio + 1
        senão
            sup := meio - 1
```

Listas lineares em alocação sequencial

- A complexidade do algoritmo de busca binária pode ser avaliada da seguinte forma. O pior caso acontece quando o elemento procurado é o último a ser encontrado, ou mesmo não é encontrado. Na primeira iteração, a dimensão da tabela é n, e algumas operações são realizadas para situar o valor procurado. Na segunda, a dimensão se reduz a $\lfloor n/2 \rfloor$, e assim sucessivamente. Ao final, a dimensão da tabela é 1. Então, no pior caso:
  - 1ª iteração: a dimensão da tabela é n;
  - 2ª iteração: a dimensão da tabela é $\lfloor n/2 \rfloor$;
  - 3ª iteração: a dimensão da tabela é $\left\lfloor \lfloor n/2 \rfloor / 2 \right\rfloor$;
  - mª iteração: a dimensão da tabela é 1.
- Portanto, o número máximo de iterações é $1 + \log_2 n$, tendo, portanto, complexidade $O(\log_2 n)$.

Ibmec
PROTAGONISTAS PARA O MUNDO

Listas lineares em alocação sequencial

- Ambas as operações de inserção e remoção utilizam o procedimento de busca. No primeiro caso, o objetivo é evitar chaves repetidas e, no segundo, a necessidade de localizar o elemento a ser removido.
- Os dois algoritmos a seguir consideram tabelas não ordenadas. A memória pressuposta disponível tem M posições. Devem-se levar em conta as hipóteses de se tentar fazer inserções numa lista que já ocupa M posições (situação conhecida como **overflow**), bem como a tentativa de remoção de um elemento de uma lista vazia (**underflow**). A atitude a ser tomada em cada um desses casos depende do problema tratado.
- Ambos as operações possuem complexidade $O(n)$, apesar do algoritmo de remoção ser mais lento que o de inserção.

Listas lineares em alocação sequencial

Algoritmo 11. Inserção de um nó na lista L

```
se n < M então
    se busca(x) = 0 então
        L[n + 1] := novo-valor
        n := n + 1
    senão
        "elemento já existe na tabela"
senão
    "overflow"
```

Algoritmo 12. Remoção de um nó na lista L

```
se n != 0 então
    indice := busca(x)
    se indice != 0 então
        valor-recuperado := L[indice]
        para i := indice, n - 1 faça
            L[i] := L[i + 1]
        n := n - 1
    senão "elemento não se encontra na tabela"
senão
    "underflow"
```

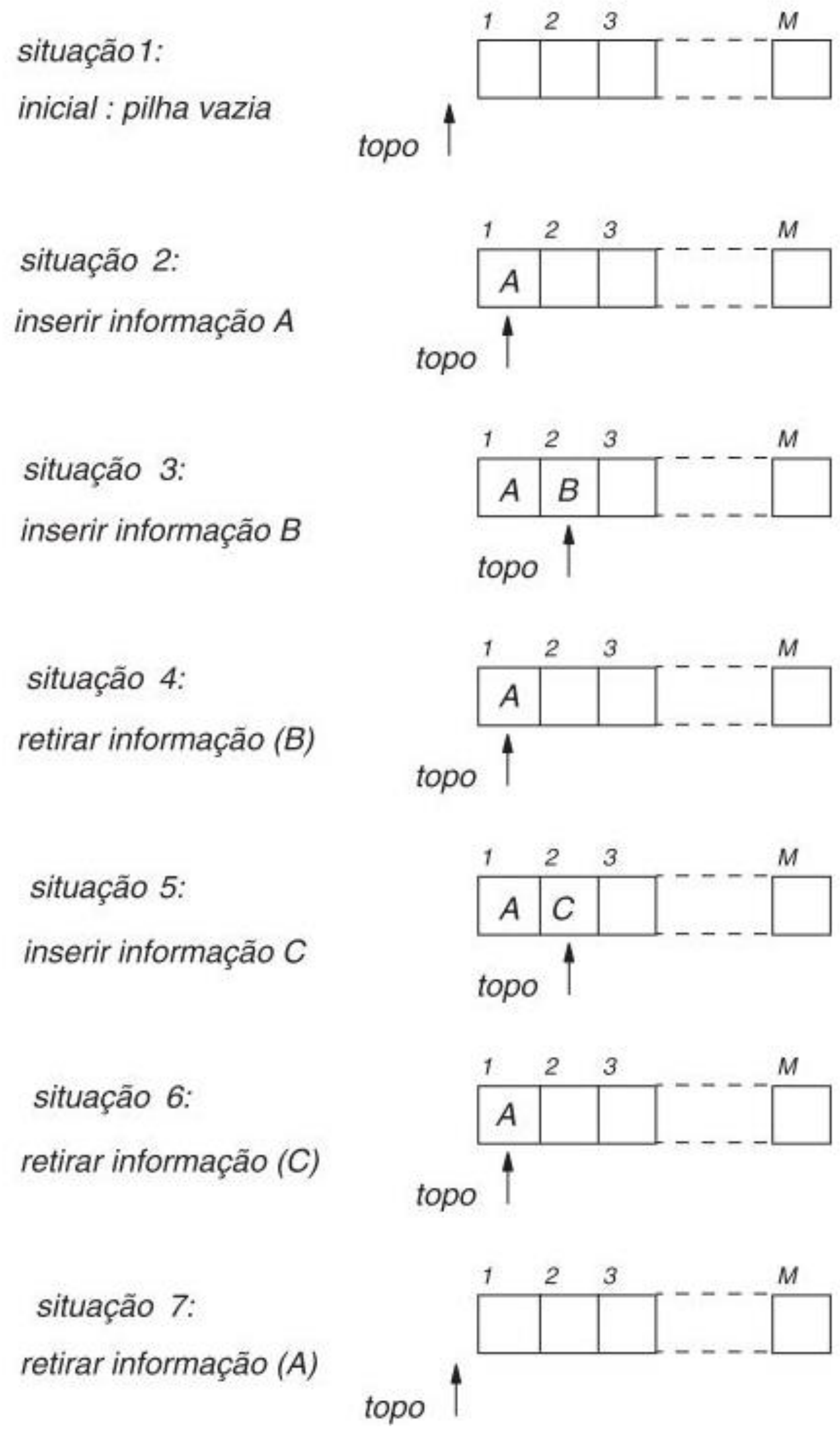
PROTAGONISTAS PARA O MUNDO

Pilhas e Filas

- Em geral, o armazenamento sequencial de listas é empregado quando as estruturas, ao longo do tempo, sofrem poucas remoções e inserções. Em casos particulares de listas, esse armazenamento também é empregado. Nessa caso, a situação favorável é aquela em que inserções e remoções não acarretam movimentação de nós, o que ocorre se os elementos a serem inseridos e removidos estão em posições especiais, como a primeira ou a última posição. Deques, pilhas e filas satisfazem tais condições.
- Na alocação sequencial de listas genéricas, considera-se sempre a primeira posição da lista no endereço 1 da memória disponível. Uma alternativa a essa estratégia consiste na utilização de indicadores especiais, denominados **ponteiros**, para o acesso a posições selecionadas.

Ibmec
PROTAGONISTAS PARA O MUNDO

Pilhas e Filas



Algoritmo 13. Inserção na pilha P

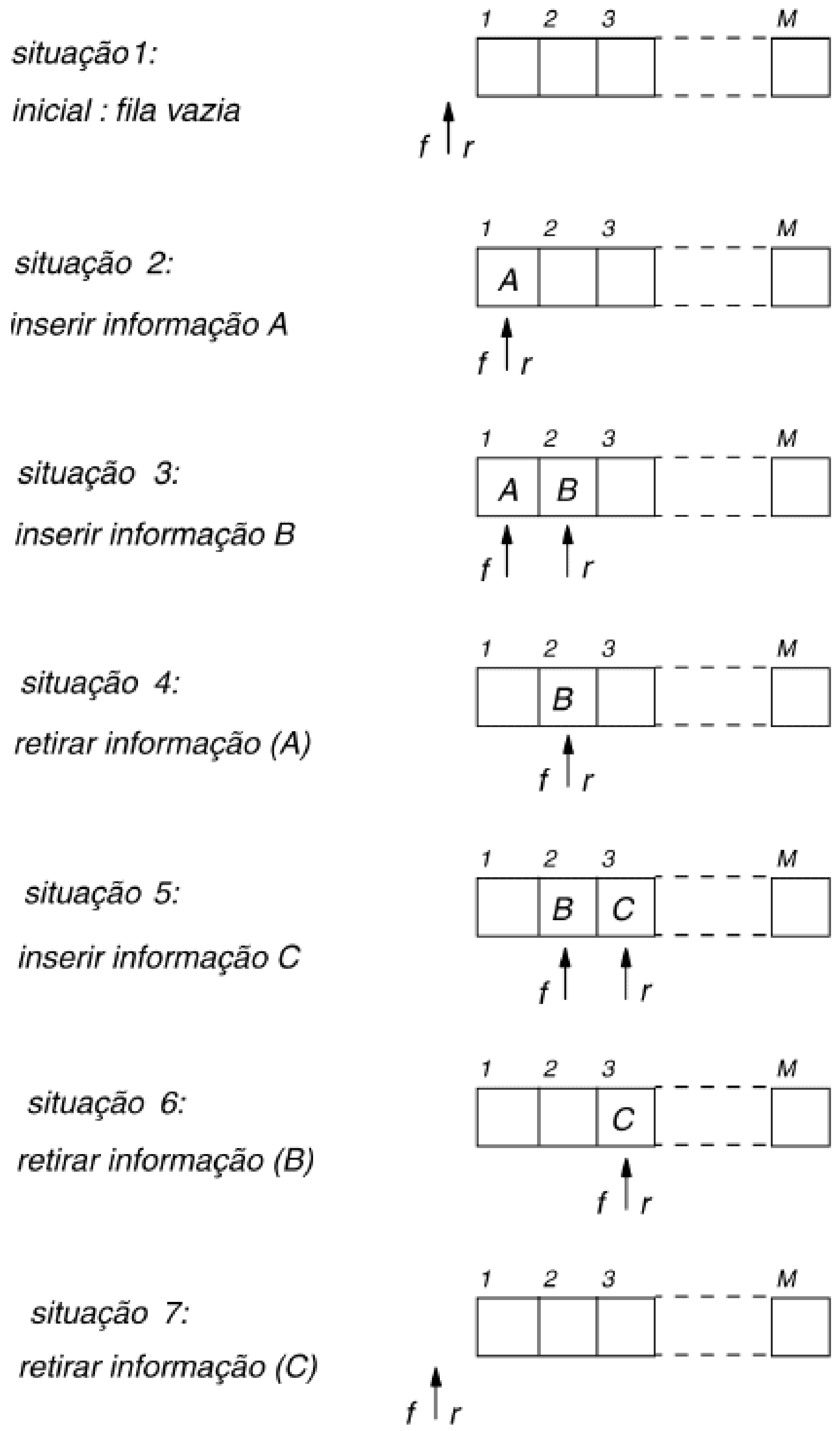
```
se topo != M então
    topo := topo + 1
    P[topo] := novo-valor
senão
    "overflow"
```

Algoritmo 14. Remoção da pilha P

```
Se topo != 0 então
    valor-recuperado := P[topo]
    topo := topo - 1
senão
    "underflow"
```

Ibmec
PROTAGONISTAS PARA O MUNDO

Pilhas e Filas



Algoritmo 15. Inserção na fila F

```
prov := r mod M + 1
se prov != f então
    r := prov
    F[r] := novo-valor
    se f = 0 então
        f := 1
senão
    "overflow"
```

Algoritmo 16. Remoção da fila F

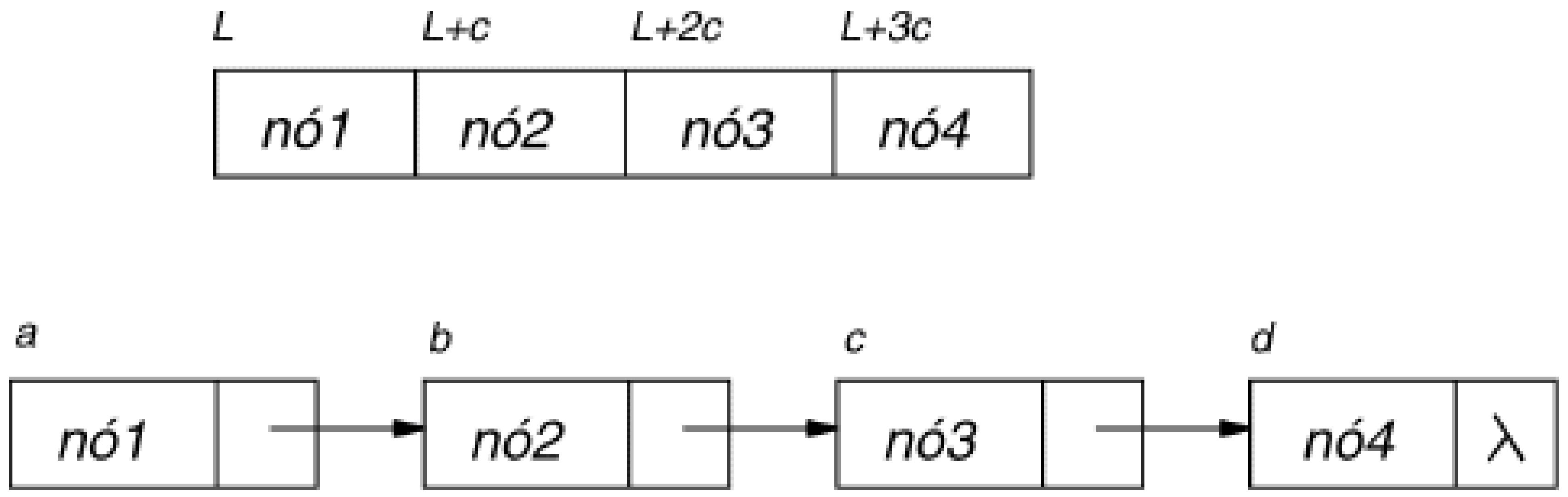
```
se f != 0 então
    valor-recuperado := F[f]
    se f = r então
        f := 0
        r := 0
    senão
        f := f mod M + 1
senão
    "underflow"
```

Alocação encadeada

- O desempenho dos algoritmos que implementam operações realizadas em listas com alocação sequencial, mesmo sendo estes muito simples, pode ser bastante fraco. E mais, quando está prevista a utilização concomitante de mais de duas listas a gerência de memória se torna mais complexa.
- Nesses casos se justifica a utilização da alocação encadeada, também conhecida por alocação dinâmica, uma vez que posições de memórias são alocadas (ou desalocadas) na medida em que são necessárias (ou dispensadas).
- Os nós de uma lista encontram-se então aleatoriamente dispostos na memória e são interligados por ponteiros, que indicam a posição do próximo elemento da tabela. É necessário o acréscimo de um campo a cada nó, justamente o que indica o endereço do próximo nó da lista.

Alocação encadeada

- Há vantagens e desvantagens associadas a cada tipo de alocação. Estas, entretanto, só podem ser precisamente medidas ao se conhecerem as operações envolvidas na aplicação desejada.
- De maneira geral pode-se afirmar que a alocação encadeada, a despeito de um gasto de memória maior em virtude da necessidade de um novo campo no nó (o campo do ponteiro), é mais conveniente quando o problema inclui o tratamento de mais uma lista.



PROTAGONISTAS PARA O MUNDO

Listas simplesmente encadeadas

- Qualquer estrutura, inclusive listas, que seja armazenada em alocação encadeada requer o uso de um ponteiro que indique o endereço de seu primeiro nó. O percurso de uma lista é feito então a partir desse ponteiro. A ideia consiste em seguir consecutivamente pelos endereços existentes no campo que indica o próximo nó, da mesma forma que na alocação sequencial se acrescentava uma unidade ao índice do percurso.

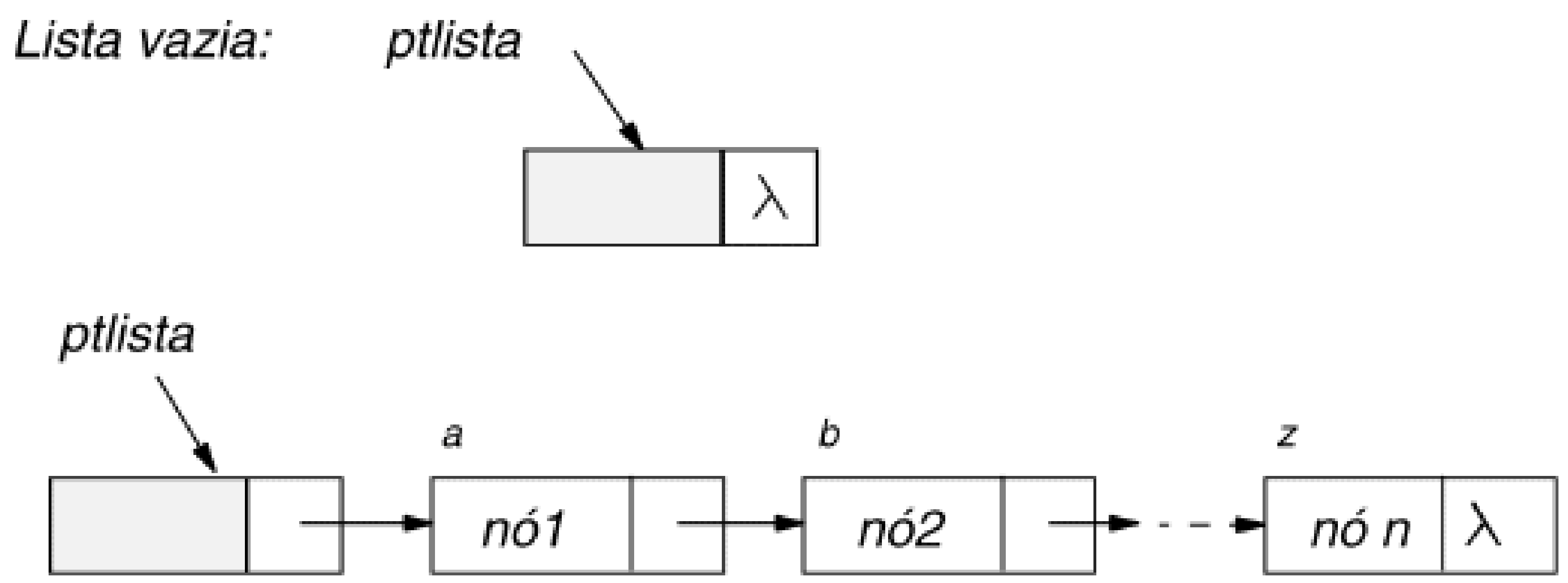
Algoritmo 17. Impressão da lista apontada por ptlista

```
pont := ptlista
enquanto pont != lambda faça
    imprimir(pont*.info)
    pont := pont*.prox
```

PROTAGONISTAS PARA O MUNDO

Listas simplesmente encadeadas

- O algoritmo de busca em listas encadeadas possui mais restrições, além da eficiência: por exemplo, a existência de um ponteiro indicando o primeiro nó da lista obriga os algoritmos de inserção e remoção a apresentarem testes especiais para verificar se o nó desejado é o primeiro da lista.
- Isto pode ser resolvido por uma pequena variação na estrutura de armazenamento: a criação de um nó especial, chamado **nó-cabeça**, nunca removido, que passa a ser o nó indicado pelo ponteiro de início de lista.



PROTAGONISTAS PARA O MUNDO

Listas simplesmente encadeadas

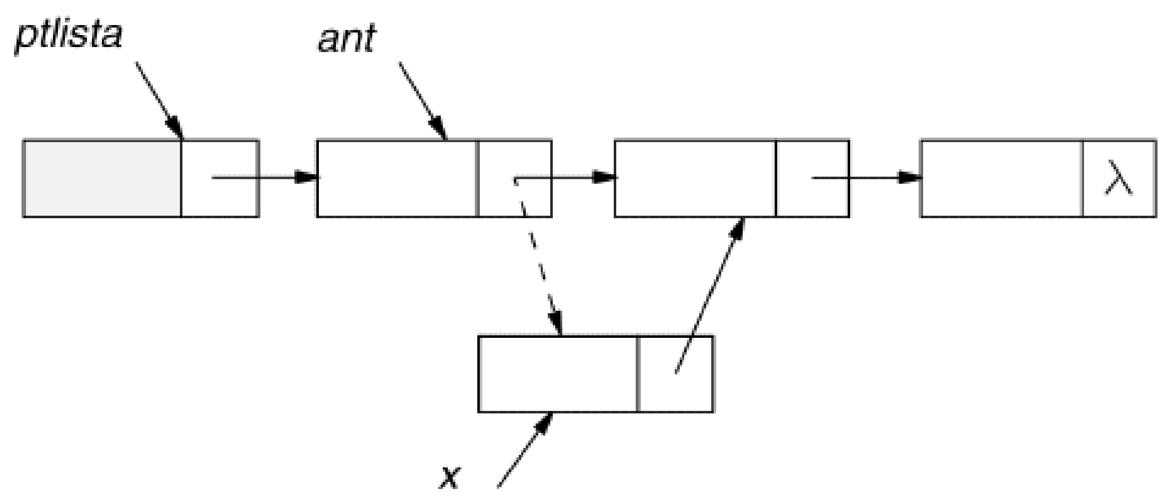
Algoritmo 18. Busca em uma lista ordenada

```
procedimento busca-enc(x, ant, pont)
    ant := ptlista
    pont := lambda
    % ponteiro de percurso
    ptr := ptlista*.prox
    enquanto ptr != lambda faça
        se ptr*.chave < x então
            % atualiza ant e ptr
            ant := ptr
            ptr := ptr*.prox
        senão se ptr*.chave = x então
            % chave encontrada
            pont := ptr
            ptr := lambda
```

PROTAGONISTAS PARA O MUNDO

Listas simplesmente encadeadas

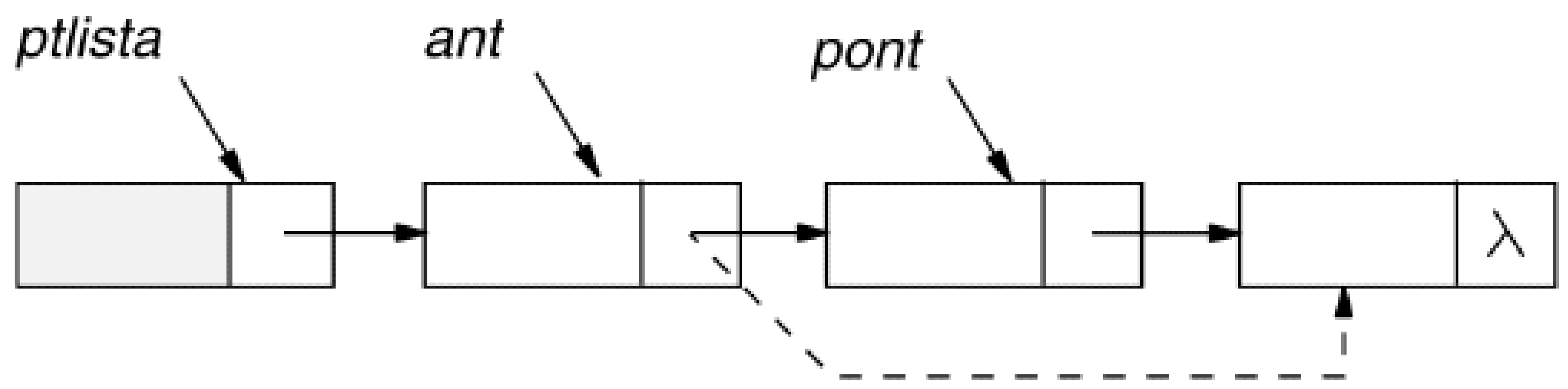
- Após a realização da busca, as operações de inserção e remoção em uma lista encadeada são triviais.
- Há três fases a serem cumpridas: a ocupação do espaço na memória, o acesso ao campo de informações e o acerto da estrutura.



Algoritmo 19. Inserção de um nó após o nó apontado por busca-enc(x, ant, pont)

```
se pont = lambda então
    % solicitar nó
    ocupar(pt)
    % inicializar nó
    pt*.info := novo-valor
    % acertar lista
    pt*.chave := x
    pt*.prox := ant*.prox
    ant*.prox := pt
senão
    "elemento já está na tabela"
```

Listas simplesmente encadeadas



Algoritmo 20. Remoção do nó apontado por pont na lista

```
busca-enc(x, ant, pont)
se pont != lambda então
    % acertar lista
    ant*.prox := pont*.prox
    % utilizar nó
    valor-recuperado := pont*.info
    % devolver nó
    desocupar(pont)
senão
    "nó não se encontra na tabela"
```

Ibmec
PROTAGONISTAS PARA O MUNDO

Pilhas e Filas encadeadas

- Como casos particulares, algumas modificações são necessárias para implementar operações eficientes em pilhas e filas. No caso de pilhas, as operações são muito simples. Considerando-se listas simplesmente encadeadas (sem nó-cabeça), o topo da pilha é o primeiro nó da lista, apontado por uma variável ponteiro $topo$. Se a pilha estiver vazia então $topo = lambda$. Filas exigem duas variáveis do tipo ponteiro: $inicio$, que aponta para o primeiro nó da lista, e fim, que aponta para o último. Na fila vazia, ambos apontam para $lambda$. Os algoritmos que se seguem implementam essas operações.

Pilhas e Filas encadeadas

Algoritmo 21. Inserção na pilha

```
% solicitar nó
ocupar(pt)
% inicializar nó
pt*.info := novo-valor
pt*.prox := topo
% acertar pilha
topo := pt
```

Algoritmo 22. Remoção da pilha

```
se topo != lambda então
    % acertar pilha
    pt := topo
    topo := topo*.prox
    % utilizar nó
    valor-recuperado := pt*.info
    % devolver nó
    desocupar(pt)
senão
    "underflow"
```

Pilhas e Filas encadeadas

Algoritmo 23. Inserção na fila

```
% solicitar nó
ocupar(pt)
% inicializar nó
pt*.info := novo-valor
pt*.prox := lambda
% acertar fila
se fim != lambda então
    fim*.prox := pt
senão inicio := pt
fim := pt
```

Algoritmo 24. Remoção da fila

```
se inicio != lambda então
    pt := inicio
    % acertar fila
    inicio := inicio*.prox
    se inicio = lambda então
        fim := lambda
    % utilizar nó
    valor-recuperado := pt*.info
    % devolver nó
    desocupar(pt)
senão
    "underflow"
```

Árvores

Introdução

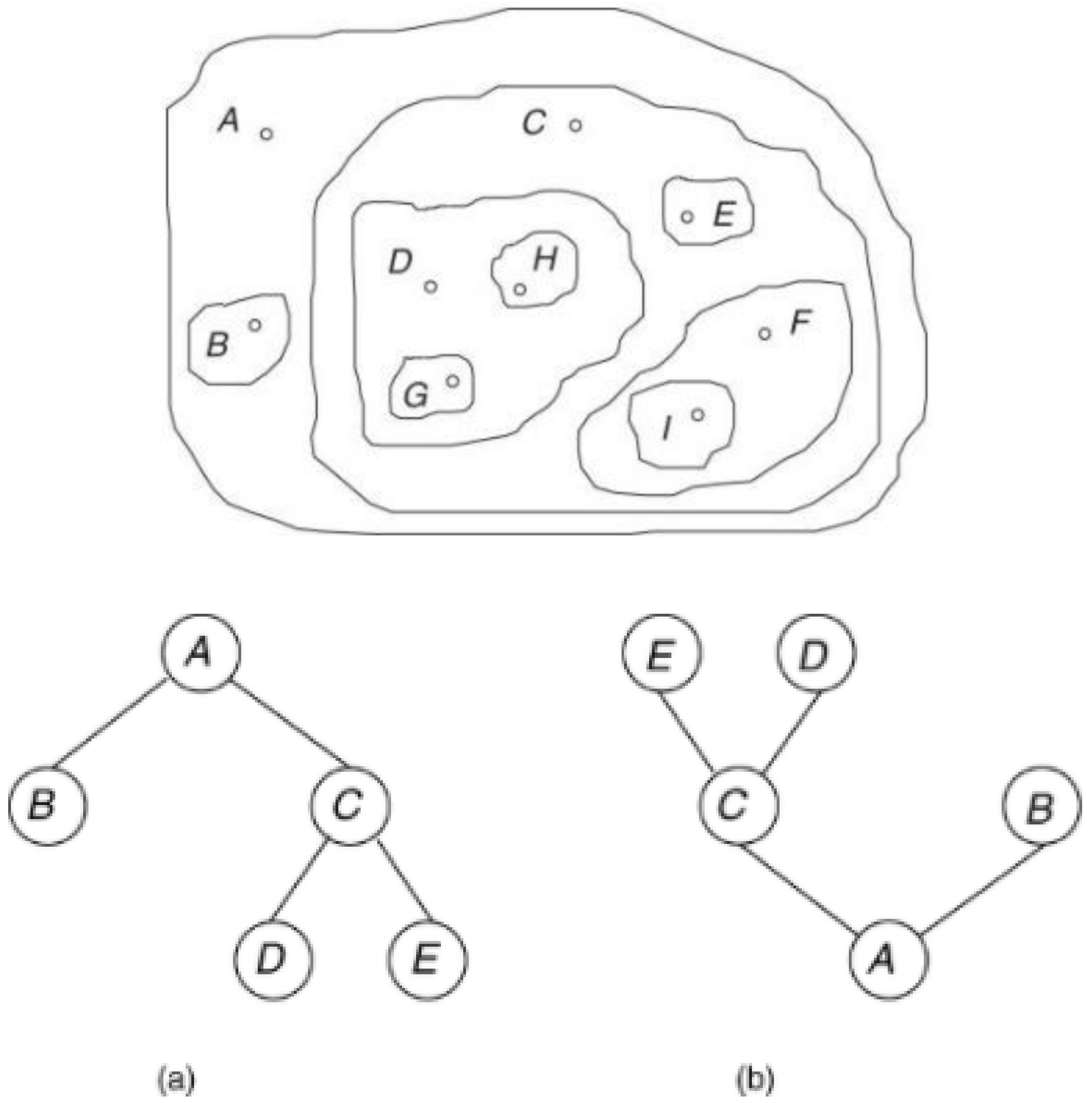
- Em diversas aplicações necessita-se de estruturas mais complexas do que as puramente sequenciais, examinadas no capítulo anterior. Entre essas, destacam-se as árvores, por existirem inúmeros problemas práticos que podem ser modelados através delas. Além disso, as árvores, em geral, admitem um tratamento computacional simples e eficiente. Isso não pode ser dito de estruturas mais gerais do que as árvores, como os grafos, por exemplo.
- Neste capítulo são apresentados os conceitos iniciais relativos às árvores, bem como os algoritmos para sua manipulação computacional básica.

Definições e representações básicas

- Uma **árvore enraizada T**, ou simplesmente **árvore**, é um conjunto finito de elementos denominados *nós* ou *vértices* tais que:
  - $T = \emptyset$, e a árvore é dita *vazia*; ou
  - Existe um nó especial chamado *raiz* de $T\big(r(T)\big)$; os restantes constituem um único conjunto vazio ou são divididos em $m \geq 1$ conjuntos disjuntos não vazios, as subárvores de $r(T)$, ou simplesmente **subárvores**, cada qual, por sua vez, uma árvore.
- Uma floresta é um conjunto de árvores. Se v é um nó de T, a notação $T(v)$ indica a subárvore de T com raiz v.

Ibmec
PROTAGONISTAS PARA O MUNDO

Definições e representações básicas



Definições e representações básicas

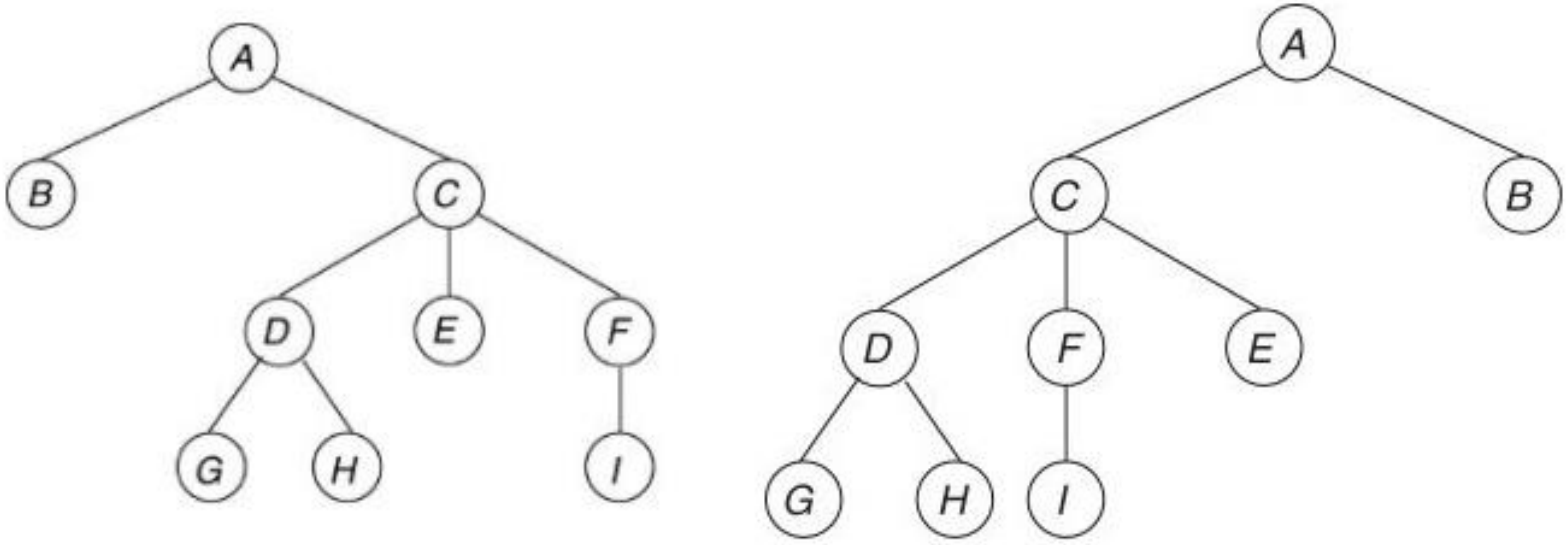
- Seja v o nó raiz da subárvore $T(v)$ de T. Os nós raízes $w_1, w_2, \dots, w_j$ das subárvores de $T(v)$ são chamados **filhos** de v; v é chamado pai de $w_1, w_2, \dots, w_j$. Os nós $w_1, w_2, \dots, w_j$ são **irmãos**. Se z é filho de w_1, então w_2 é **tio** de z e v é **avô** de z. O número de filhos de um nó é chamado de **grau de saída** desse nó. Se x pertence à subárvore $T(v)$, x é **descendente** de v, e v é **ancestral** de x. Nesse caso, sendo x diferente de v, x é **descendente próprio** de v, e v é **ancestral próprio** de x.
- Um nó que não possui descendentes próprios é chamado de **folha**. Toda árvore com $n > 1$ nós possui no mínimo 1 e no máximo $n - 1$ folhas. Um nó não folha é dito **interior**.

Definições e representações básicas

- Uma sequência de nós distintos $v_1, v_2, \dots, v_k$, tal que existe sempre entre nós consecutivos (v_1 e v_2, v_2 e v_3, ..., v_{k-1} e v_k) a relação "é filho de" ou "é pai de", é denominada **caminho da árvore**. Diz-se que v_1 alcança v_k e vice-versa. Um caminho de k vértices é obtido pela sequência de $k - 1$ pares da relação. O valor $k - 1$ é o comprimento do caminho. **Nível de um nó** v é o número de nós do caminho da raiz até o nó v. O nível da raiz é, portanto, igual a 1. A **altura** de um nó v é o número de nós do maior caminho de v até um de seus descendentes. As folhas têm altura 1. A altura da árvore T é igual ao nível máximo de seus nós. Representa-se a altura de T por $h(T)$, enquanto $h(v)$ é a altura da subárvore de raiz v.

Definições e representações básicas

- Uma **árvore ordenada** é aquela na qual os filhos de cada nó estão ordenados. Assume-se que tal ordenação se desenvolva da esquerda para a direita. Assim, as árvores da figura abaixo são distintas se consideradas como ordenadas. Contudo, elas podem se tornar coincidentes mediante uma reordenação de nós irmãos.



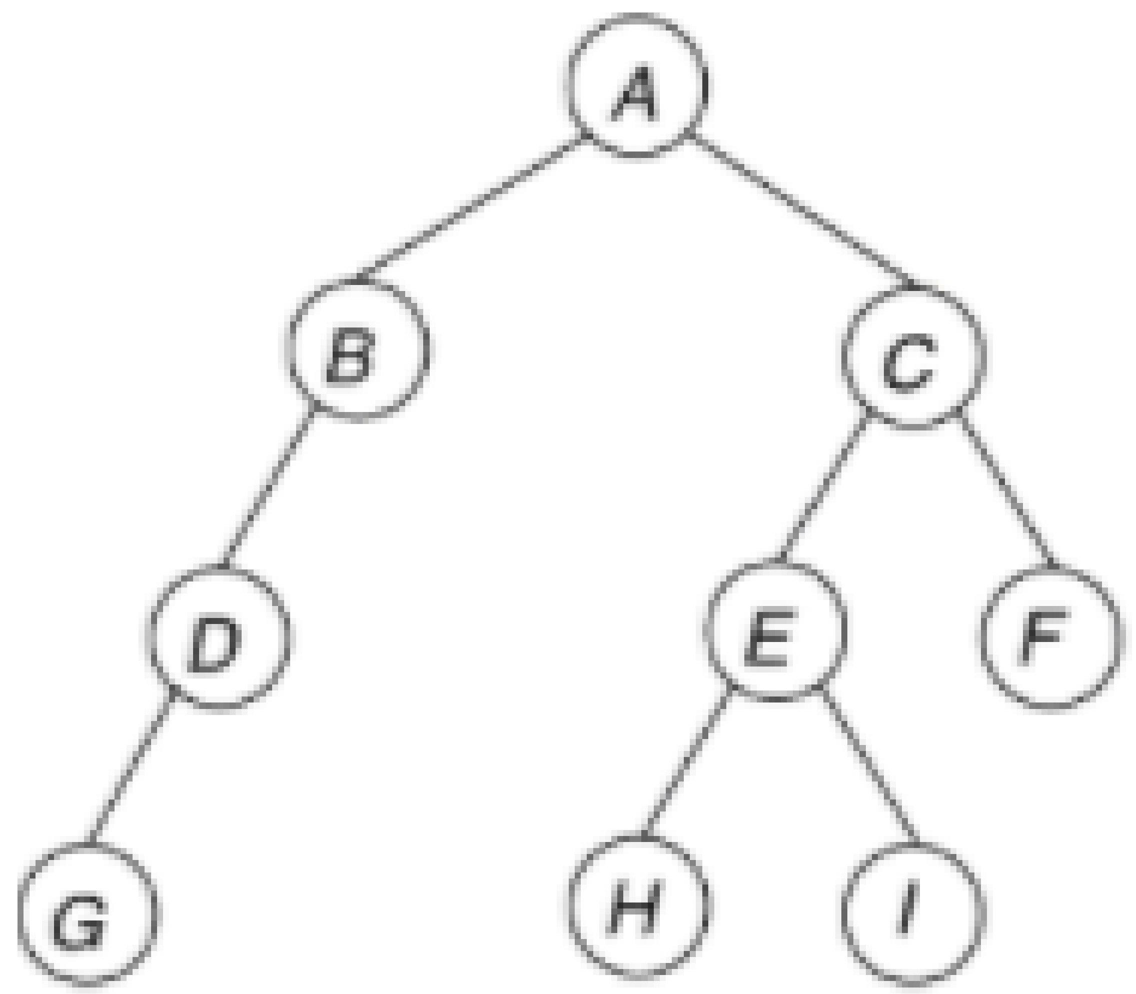
Árvores binárias

- Conforme já mencionado, as árvores constituem as estruturas não sequenciais com maior aplicação em computação. Dentre as árvores, as binárias são, sem dúvida, as mais comuns.
- Uma **árvore binária** T é um conjunto finito de elementos denominados nós ou vértices, tal que:
  - $T = \emptyset$, e a árvore é dita *vazia*; ou
  - Existe um nó especial chamado *raiz* de $T\big(r(T)\big)$; e os restantes podem ser divididos em dois subconjuntos disjuntos, $T_E\big(r(T)\big)$ e $T_D\big(r(T)\big)$, a subárvore esquerda e a direita da raiz, respectivamente, as quais são também árvores binárias.

Ibmec
PROTAGONISTAS PARA O MUNDO

Árvores binárias

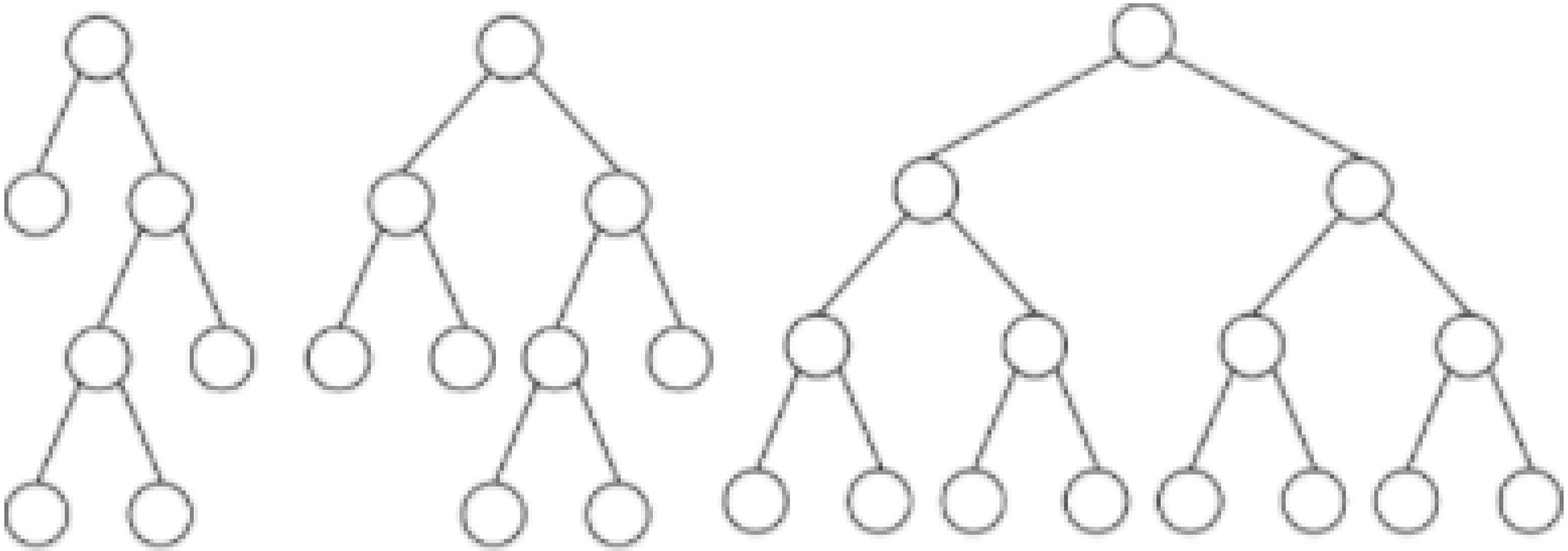
- A raiz da subárvore esquerda (direita) de um nó v, se existir, é denominada **filho esquerdo (direito)** de v. Naturalmente, o esquerdo pode existir sem o direito e vice-versa. Analogamente à seção anterior, a notação $T(v)$ indica a (sub) árvore binária, cuja raiz é v e cujas subárvores esquerda e direita de T são $T_E(v)$ e $T_D(v)$, respectivamente.



Árvores binárias

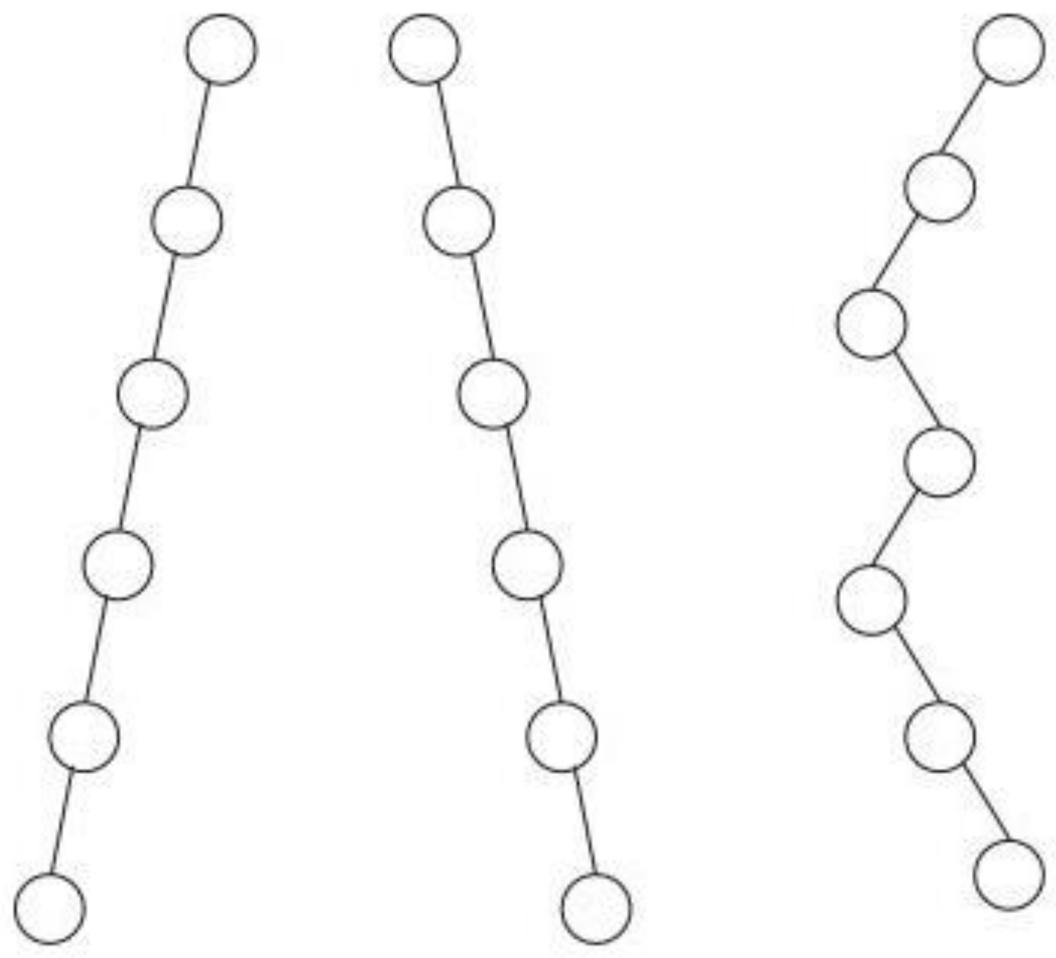
- Toda árvore binária com n nós possui exatamente $n+1$ subárvores vazias entre suas subárvores esquerdas e direitas. Por exemplo, a árvore da figura anterior possui 9 nós e 10 subárvores vazias: as subárvores esquerda e direita dos nós F, G, H, I e as subárvores direitas de B e D.
- Em seguida, são introduzidos alguns tipos especiais de árvores binárias:
  - Uma *árvore estritamente binária* é uma árvore binária em que cada nó possui 0 ou 2 filhos;
  - Uma *árvore binária completa* é aquela que apresenta a seguinte propriedade: se v é um nó tal que alguma subárvore de v é vazia, então v se localiza ou no último (maior) ou no penúltimo nível da árvore;
  - Uma *árvore binária cheia* é aquela em que, se v é um nó com alguma de suas subárvores vazias, então v se localiza no último nível. Segue-se que toda árvore binária cheia é completa e estritamente binária.

Árvores binárias



Árvores binárias

- A relação entre a altura de uma árvore binária e o seu número de nós é um dado importante para várias aplicações. Para um valor fixo de n, indagar-se-ia quais são as árvore binárias que possuem altura h máxima e mínima. A resposta ao primeiro problema é imediata. A árvore binária que possui altura máxima é aquela cujos nós interiores possuem exatamente uma subárvore vazia. Essas árvores são denominadas **zigue-zague**. Naturalmente, a altura de uma árvore zigue-zague é igual a n.



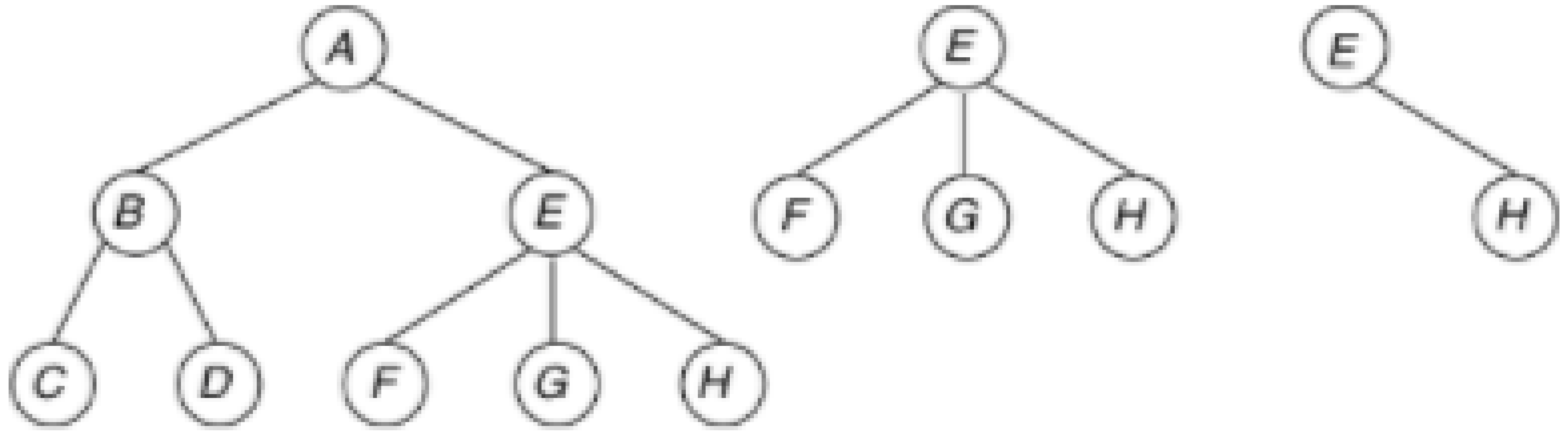
Altura de árvores binárias completas

- Seja T uma árvore binária completa com $n > 0$ nós. Então T possui altura h mínima.
- A altura h mínima é dada por $h = 1 + \lfloor \log n \rfloor$, cuja demonstração, por indução, é dada abaixo:
  - Se $n = 1$, então $h = 1 + \lfloor \log n \rfloor = 1$, correto. Quando $n > 1$, suponha o resultado verdadeiro para todas as árvores binárias completas com até $n - 1$ nós. Seja T' a árvore obtida de T pela remoção de todos os nós, em número de k, do último nível. Logo, T' é uma árvore cheia com $n' = n - k$ nós. Pela hipótese de indução, $h(T') = 1 + \lfloor \log n' \rfloor$. Como T' é cheia, $n' = 2^m - 1$, para algum inteiro $m > 0$. Isto é, $h(T') = m$. Além disso, $1 \leq k \leq n' + 1$. Assim:

$$h(T) = 1 + h(T') = 1 + m = 1 + \log(n' + 1) = 1 + \lfloor \log(n' + k) \rfloor = 1 + \lfloor \log n \rfloor$$

Subárvore e subárvore parcial

- Seja T uma árvore (ou uma árvore binária) e v um nó de T. Seja $T(v)$ a subárvore de T de raiz v, e S um conjunto de nós $T(v)$ tal que $T(v) - S$ é uma árvore. A árvore $T' = T(v) - S$ é chamada **subárvore parcial** de raiz v. Observe, por exemplo, a árvore T abaixo, à esquerda. A árvore ao centro é subárvore de T de raiz E, enquanto a árvore à direita é uma subárvore parcial de T de raiz E, porém não é subárvore de T. Observe que a diferença entre uma subárvore de raiz v e uma subárvore parcial de raiz v é que a primeira contém obrigatoriamente todos os descendentes de v, enquanto a segunda, não necessariamente.

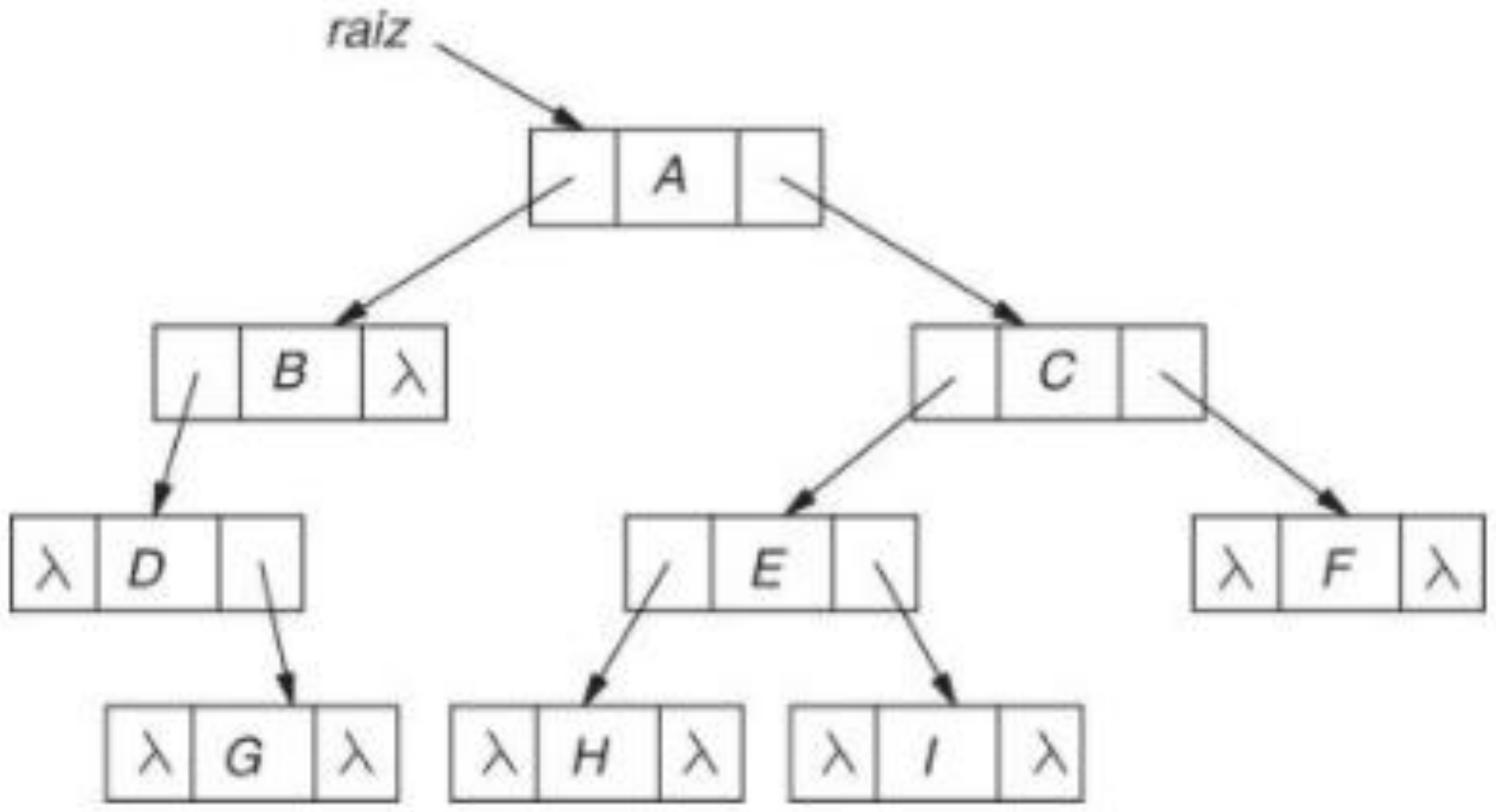


Armazenamento de árvores

- O armazenamento de árvores pode utilizar alocação sequencial ou encadeada. As vantagens e desvantagens de uma e outra já foram discutidas. Sendo a árvore uma estrutura mais complexa do que listas lineares, as vantagens na utilização da alocação encadeada prevalecem.
- Não é difícil observar que a estrutura de armazenamento para árvores deve conter, em cada nó, ponteiros para seus filhos. A disposição mais econômica consiste em limitar o número de filhos a dois, exatamente o caso de árvores binárias. Note que o número de subárvores vazias cresce com o aumento do parâmetro m das árvores m-árias. Para um dado valor de n, a árvore binária é aquela que minimiza o número de ponteiros necessários.

Armazenamento de árvores

- O armazenamento de uma árvore binária surge naturalmente de sua definição. Cada nó deve possuir dois campos de ponteiros, **esq** e **dir**, que apontam para as suas subárvores esquerda e direita, respectivamente. O ponteiro **ptraiz** indica a raiz da árvore. Da mesma forma que na alocação encadeada de listas lineares, a memória é inicialmente considerada uma lista de espaço disponível. Os campos do nó da árvore que contém as informações pertinentes ao problema serão aqui representados como um só campo de nome **info**.
- A figura abaixo ilustra a estrutura de ponteiros usada no armazenamento de uma árvore binária.



Percurso em árvores binárias

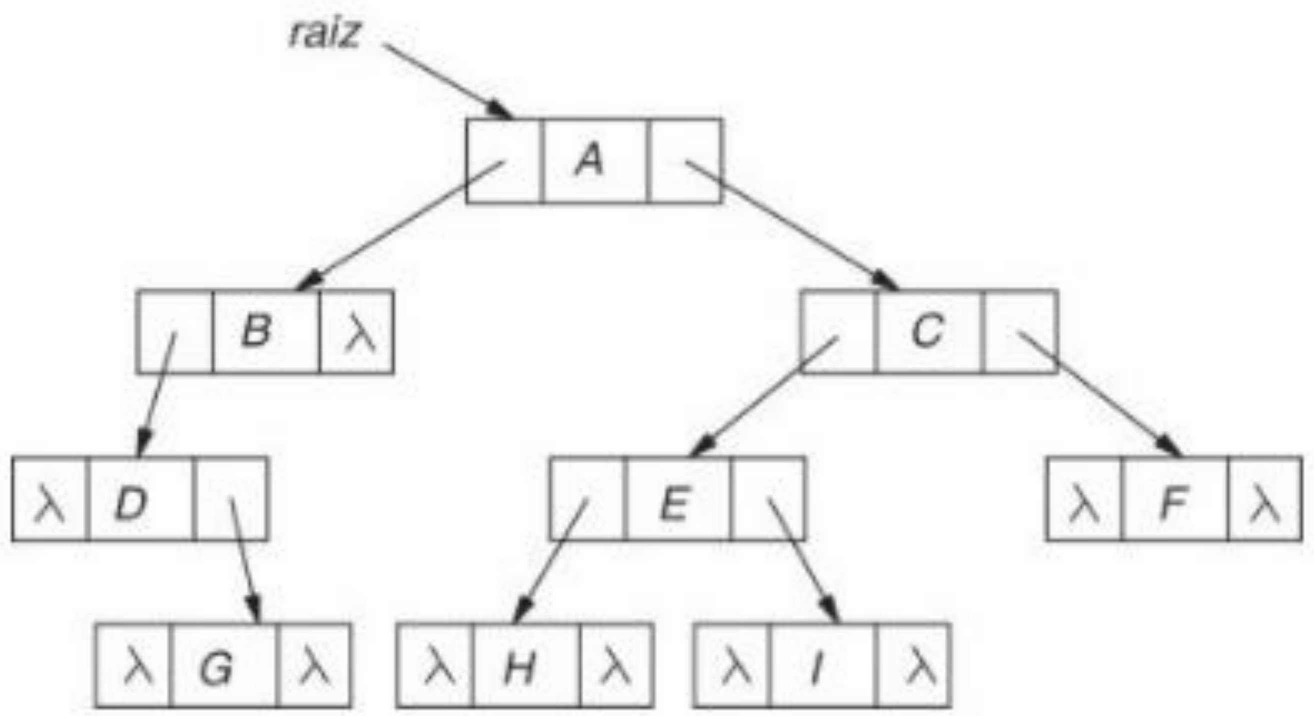
- Por **percurso** entende-se uma visita sistemática a cada um de seus nós; esta é uma das operações básicas relativas à manipulação de árvores. Uma árvore é, essencialmente, uma estrutura não sequencial. Por isso mesmo, ela pode ser utilizada em aplicações que demandem acesso direto. Contudo, mesmo nesse tipo de aplicação é imprescindível conhecer métodos eficientes para percorrer toda a estrutura. Por exemplo, para listar o conteúdo de um arquivo é necessário utilizar algoritmos para percurso.
- Para percorrer uma árvore deve-se, então, visitar cada um de seus nós. Visitar um nó significa operar, de alguma forma, com a informação a ele relativa. Em geral, percorrer uma árvore significa visitar seus nós exatamente uma vez. Contudo, no processo de percorrer a árvore pode ser necessário passar várias vezes por alguns de seus nós sem visitá-los. A seguir são discutidas as ideias principais nas quais se baseiam alguns dos algoritmos de percurso em árvore.

Percurso em árvores binárias

- Um dos passos de qualquer algoritmo de percurso é visitar a raiz v de cada subárvore da árvore T. Além disso, pode-se assumir que o algoritmo opere de tal forma que o percurso de T seja uma composição de percursos de suas subárvores.
- Nesse caso, poder-se-iam se identificar, no percurso de T, os percursos de suas subárvores em forma contígua. Esses percursos correspondem, no algoritmo, às operações de **percorrer subárvores esquerda** e **direita** de v, para cada nó v de T. Essas três operações (visitar e percorrer subárvores esquerda e direita) compõem um algoritmo.
- Cada um desses percursos pode ser mais ou menos adequado a um problema de aplicação dado. São apresentados a seguir três percursos diversos.

Percurso em árvores binárias

- O **percurso em pré-ordem** segue recursivamente os seguintes passos, para cada subárvore da árvore:
  - Visitar a raiz;
  - Percorrer sua subárvore esquerda, em pré-ordem;
  - Percorrer sua subárvore direita, em pré-ordem.



- Para a árvore da figura ao lado, o percurso em pré-ordem para impressão de nós fornece a seguinte saída: A B D G C E H I F.

Algoritmo 25. Percurso em pré-ordem

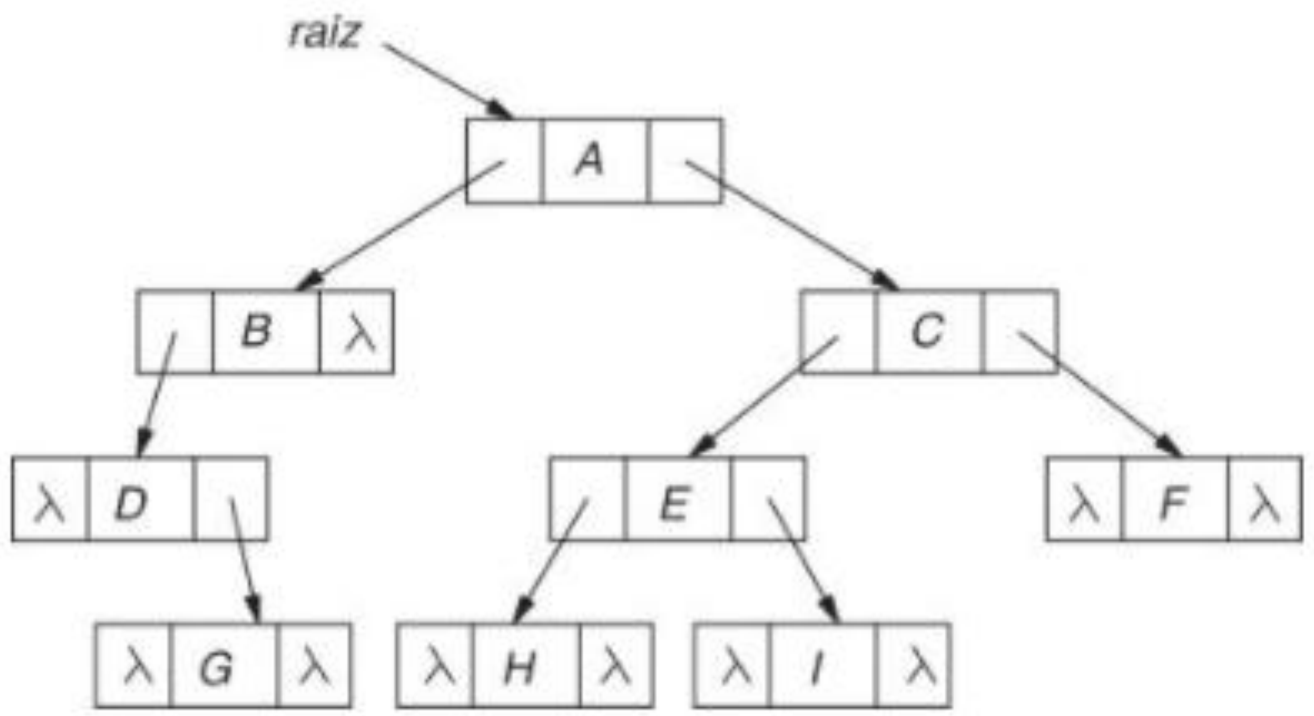
```
procedimento pre(pt)
    visita(pt)
    se pt*.esq != lambda então pre(pt*.esq)
    se pt*.dir != lambda então pre(pt*.dir)

se prtaiz != lambda então pre(ptraiz)
```

Ibmec
PROTAGONISTAS PARA O MUNDO

Percurso em árvores binárias

- O **percurso em ordem simétrica** é composto dos passos seguintes, para cada uma de suas subárvores:
  - Percorrer sua subárvore esquerda, em ordem simétrica;
  - Visitar a raiz;
  - Percorrer sua subárvore direita, em ordem simétrica.



- Para a árvore da figura ao lado, o percurso em ordem simétrica para impressão de nós fornece a seguinte saída: D G B A H E I C F.

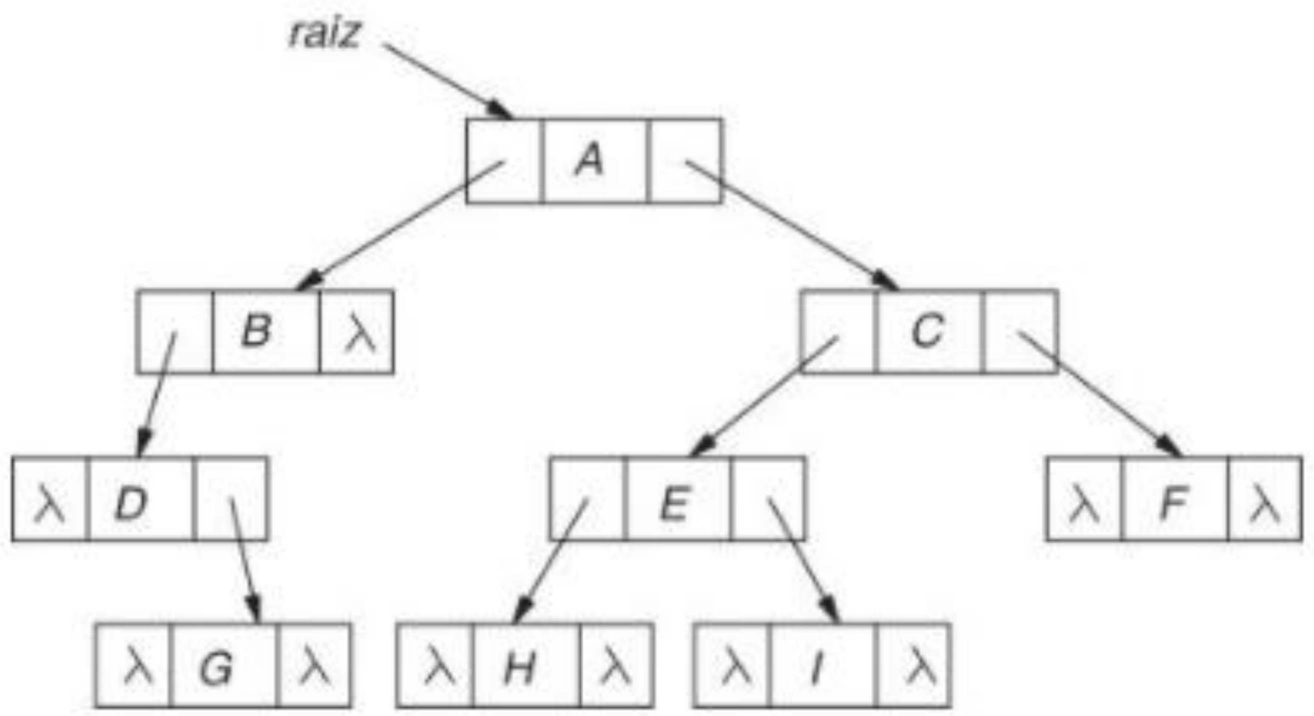
Algoritmo 26. Percurso em ordem simétrica

```
procedimento simet(pt)
    se pt*.esq != lambda então simet(pt*.esq)
    visita(pt)
    se pt*.dir != lambda então simet(pt*.dir)

se prtaiz != lambda então simet(ptraiz)
```

Percurso em árvores binárias

- O **percurso em pós-ordem** é composto dos passos seguintes, para cada uma de suas subárvores:
  - Percorrer sua subárvore esquerda, em pós-ordem;
  - Percorrer sua subárvore direita, em pós-ordem.
  - Visitar a raiz;
- Para a árvore da figura ao lado, o percurso em ordem simétrica para impressão de nós fornece a seguinte saída: G D B H I E F C A.



Algoritmo 27. Percurso em pós-ordem

```
procedimento pos(pt)
    se pt*.esq != lambda então pos(pt*.esq)
    se pt*.dir != lambda então pos(pt*.dir)
    visita(pt)

se prtaiz != lambda então pos(ptraiz)
```

Cálculo da altura dos nós

- O cálculo da altura de todos os nós de uma árvore binária é uma aplicação do percurso em pós-ordem.
- A altura das folhas, pela própria definição, é 1. Para os outros nós, por exemplo v, é necessário conhecer o comprimento do maior caminho de v até um de seus descendentes. Isto equivale dizer que a altura de v deve ser calculada após a visita a seus descendentes.
- O algoritmo a seguir mostra a implementação do procedimento **visita(pt)**, que executa a tarefa de determinar a altura do nó apontado por **pt**. Considera-se **altura** um campo do nó da árvore. As variáveis auxiliares **alt1** e **alt2** armazenam, respectivamente, as alturas das subárvores esquerda e direita do nó em questão. A altura desejada corresponderá à maior altura dentre as de suas duas subárvores incrementada de um.

Cálculo da altura dos nós

Algoritmo 28. Cálculo da altura de um nó da árvore binária

```
procedimento visita(pt)
    se pt*.esq != lambda então
        alt1 := (pt*.esq)*.altura
    senão alt1 := 0

    se pt*.dir != lambda então
        alt2 := (pt*.dir)*.altura
    senão alt2 := 0

    se alt1 > alt2 então
        pt*.altura := alt1 + 1
    senão pt*.altura := alt2 + 1
```

Árvores Binárias de Busca

PROTAGONISTAS PARA O MUNDO

Introdução

- Dado um conjunto de elementos, onde cada um é identificado por uma chave, o objetivo é localizar nesse conjunto o elemento correspondente a uma chave específica procurada.
- No presente capítulo será visto um método de solução que emprega a árvore binária como estrutura na qual se processa a busca. Ou seja, os elementos do conjunto são previamente distribuídos pelos nós de uma árvore de forma conveniente. A localização da chave desejada é então obtida através de um caminhamento apropriado na árvore.
- É importante ressaltar, mais uma vez, a relevância desse problema na área de computação, em especial nas aplicações não numéricas. Sem dúvida, a operação de busca é uma das mais frequentemente realizadas.

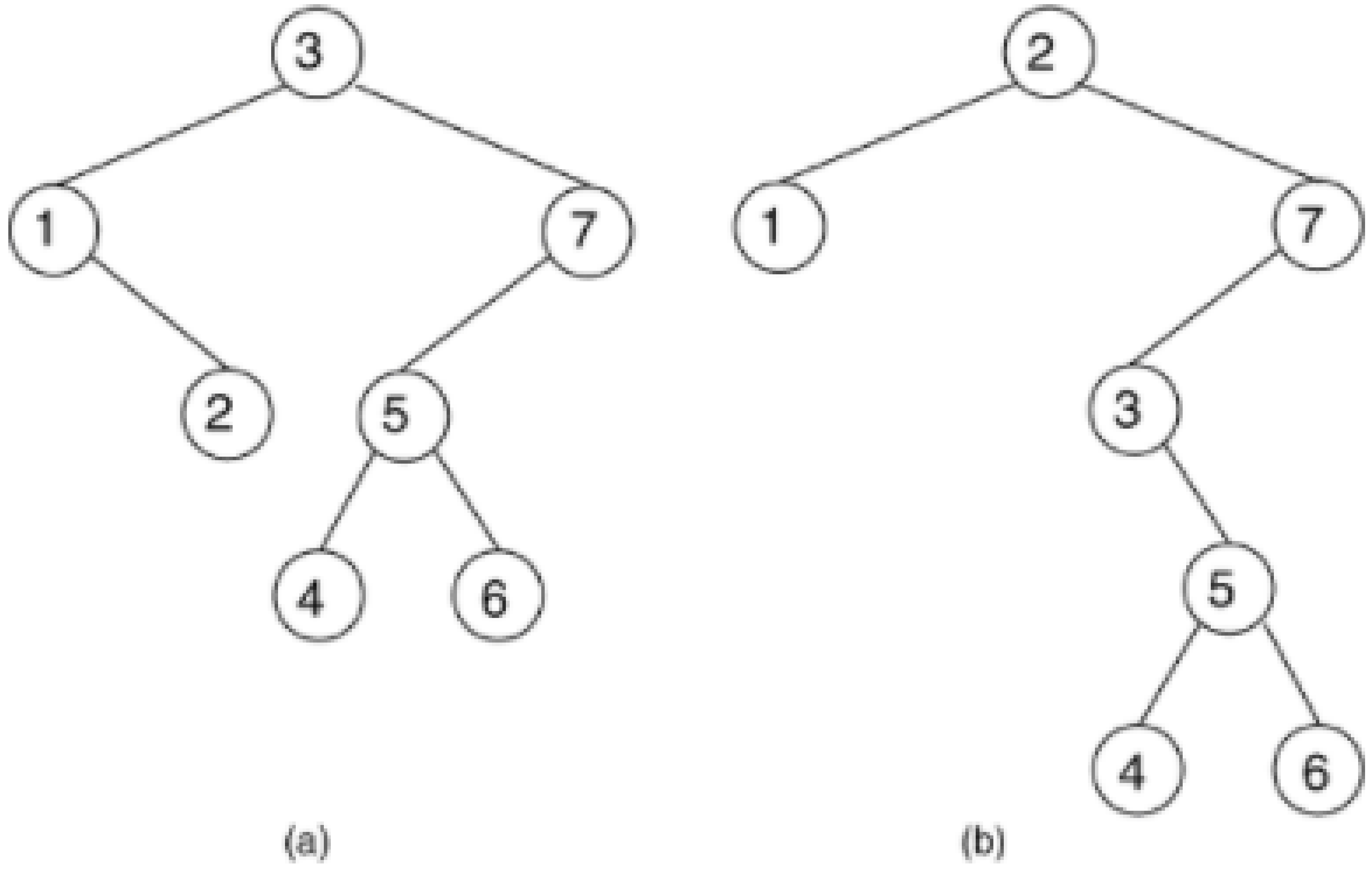
Conceitos básicos

- Seja $S = \{s_1, \ldots, s_n\}$ o conjunto de chaves satisfazendo $s_1 < \cdots < s_n$. Seja x um valor dado. O objetivo é verificar se $x \in S$ ou não. Em caso positivo, localizar x em S, isto é, determinar o índice j tal que $x = s_j$.
- Para resolver esse problema, emprega-se uma árvore binária rotulada T, com as seguintes características:
  - T possui n nós. Cada nó v corresponde a uma chave distinta $s_j \in S$ e possui como rótulo o valor $rt(v) = s_j$;
  - Seja um nó v de T. Seja também v_1, pertencente à subárvore esquerda de v. Então $rt(v_1) < rt(v)$. Analogamente, se v_2 pertence à subárvore direita de v, $rt(v_2) > rt(v)$.

Ibmec
PROTAGONISTAS PARA O MUNDO

Conceitos básicos

- A árvore T denomina-se **árvore binária de busca** para S. Naturalmente, se $|S| > 1$, existem várias árvores de busca para S. A figura abaixo ilustra duas dessas árvores para o conjunto $\{1, 2, 3, 4, 5, 6, 7\}$.



Busca

- O algoritmo seguinte implementa a ideia. Suponha que a árvore esteja armazenada da forma habitual, isto é, para cada nó v, esq e dir designam os campos que armazenam ponteiros para os filhos esquerdo e direito de v, respectivamente. A raiz da árvore é apontada por $ptraiz$. A variável f designa a natureza final da busca. Tem-se, então:
  - $f = 0$, se a árvore é vazia;
  - $f = 1$, se $x \in S$. Nesse caso, pt aponta para o nó procurado;
  - $f > 1$, se $x \notin S$.

Busca

Algoritmo 29. Busca em árvore binária de busca

```
procedimento busca-arvore(x, pt, f)
    se pt = lambda então f := 0
    senão se x = pt*.chave então f := 1
    senão se x < pt*.chave então
        se pt*.esq = lambda então f:= 2
        senão
            pt := pt*.esq
            busca-arvore(x, pt, f)
    senão
        se pt*.dir = lambda então f := 3
        senão
            pt := pt*.dir
            busca-arvore(x, pt, f)

pt := ptraiz
busca-arvore(x, pt, f)
```

Inserção

- Para resolver o problema de inserção de nós na árvore de busca T, utiliza-se também o procedimento *busca-arvore*. Seja x o valor da chave que se deseja inserir em T e $novo - valor$ a informação associada a x. A ideia inicial é verificar se $x \in \ S$. Em caso positivo, trata-se de uma chave duplicata e a inserção não pode ser realizada. Se $x \notin S$, a chave de valor x será o rótulo de algum novo nó w, situado à esquerda ou à direita de v, para $f = 2$ ou $f = 3$, respectivamente, de acordo com o procedimento *busca-arvore*. O algoritmo seguinte descreve o processo.

Ibmec
PROTAGONISTAS PARA O MUNDO

Inserção

Algoritmo 30. Inserção em árvore binária de busca

```
pt := ptraiz
busca-arvore(x, pt, f)
se f = 1 então
    "inserção inválida"
senão
    ocupar(pt1)
    pt1*.chave := x
    pt1*.info := novo-valor
    pt1*.esq := lambda
    pt1*.dir := lambda
    se f = 0 então
        ptraiz := pt1
    senão se f = 2 então
        pt*.esq := pt1
    senão pt*.dir := pt1
```

Ibmec
PROTAGONISTAS PARA O MUNDO

Árvores Balanceadas

PROTAGONISTAS PARA O MUNDO

Introdução

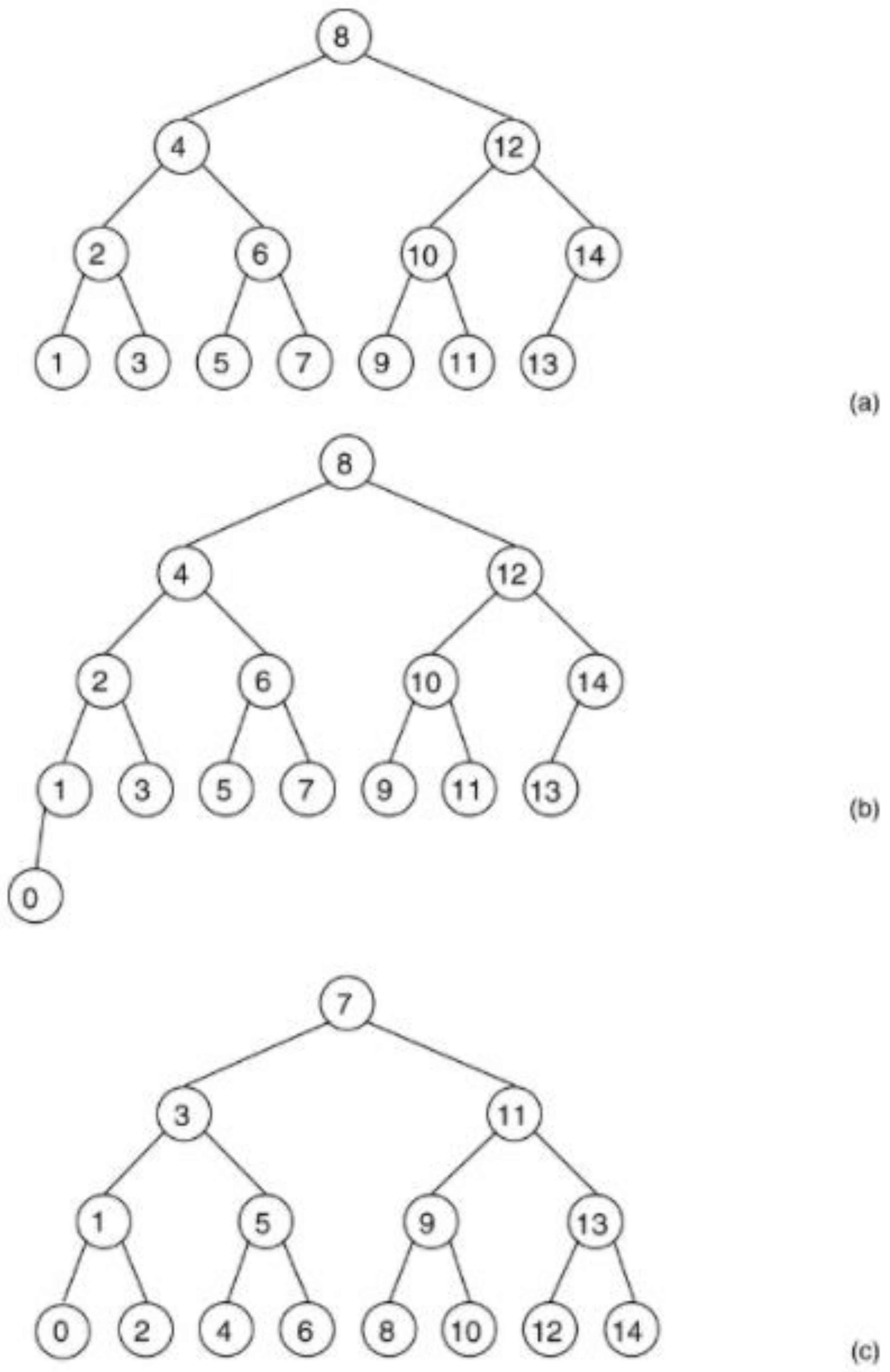
- Um aspecto fundamental do estudo de árvores de busca é, naturalmente, o custo de acesso a uma chave desejada. Com o intuito de minimizar esse custo, foram desenvolvidas as árvores binárias de busca e de partilha ótimas. Ambas, porém, se restringem a aplicações estáticas. Isto é, após um certo número de inserções e remoções as árvores deixam de ser ótimas. Além disso, a complexidade da árvore de partilha ótima é muito elevada.
- Para estrutura em que as probabilidades de acesso são idênticas entre si, há uma alternativa. A ideia é manter o custo de acesso na mesma ordem de grandeza de uma árvore ótima, ou seja, $O(\log n)$. Esse custo deve se manter ao longo de toda utilização da estrutura, inclusive após inclusões e remoções. Para alcançar essa finalidade, a estrutura deve ser alterada, periodicamente, de forma a se moldar aos novos dados, mantendo o custo em $O(\log n)$. Uma estrutura que opera com essas características é denominada **balanceada**.

Ibmec
PROTAGONISTAS PARA O MUNDO

O Conceito de Balanceamento

- As árvores completas são aquelas que minimizam o número de comparações efetuadas no pior caso para uma busca com chaves de probabilidades de ocorrência idênticas.
- Do ponto de vista das aplicações dinâmicas, contudo, o uso de árvores completas é, em geral, desaconselhável. Em um caso extremo, ela pode inclusive degenerar-se em uma lista.
- Para contornar esse problema, uma ideia seria aplicar um algoritmo que tornasse a árvore novamente completa, tão logo tal característica fosse perdida após uma inclusão ou exclusão. A dificuldade reside em como efetuar essa operação de forma ótima.

O Conceito de Balanceamento



O Conceito de Balanceamento

- Para efetuar essas transformações usuais de árvores binárias é necessário percorrer todos os nós da árvore. Isso implica que o algoritmo de restabelecimento da estrutura requer, pelo menos, $O(n)$ passos.
- Naturalmente, este custo é considerado excessivo, considerando que operações como inserção ou remoção seriam efetuadas em $O(\log n)$ passos. Por esse motivo, as árvores completas (e a busca binária) não são recomendadas para aplicações que requeiram estruturas dinâmicas.
- Além disso, é desejável que esta propriedade se estenda a todas as subárvores: cada subárvore que contém m nós deve possuir altura igual a $O(\log m)$. Uma árvore que satisfaça essa condição é denominada **balanceada**.

Ibmec
PROTAGONISTAS PARA O MUNDO

Árvores AVL

- O nome dessa estrutura deriva dos seus criadores, Adelson Velsky e Landis. Uma árvore binária *T* é denominada AVL quando, para qualquer nó de *T*, as alturas de suas duas subárvores, esquerda e direita, diferem em módulo de até uma unidade. Nesse caso, *v* é um nó **regulado**. Em contrapartida, um nó que não satisfaça essa condição de altura é denominado **desregulado**, e uma árvore que contenha um nó nessas condições é também chamada **desregulada**. Naturalmente, toda árvore completa é AVL, mas não necessariamente vale a recíproca.
- Em outras palavras, uma árvore AVL é tal que, nas inserções e remoções, procura-se executar uma rotina de balanceamento tal que as alturas das subárvores esquerda e direita tenham alturas bem próximas.

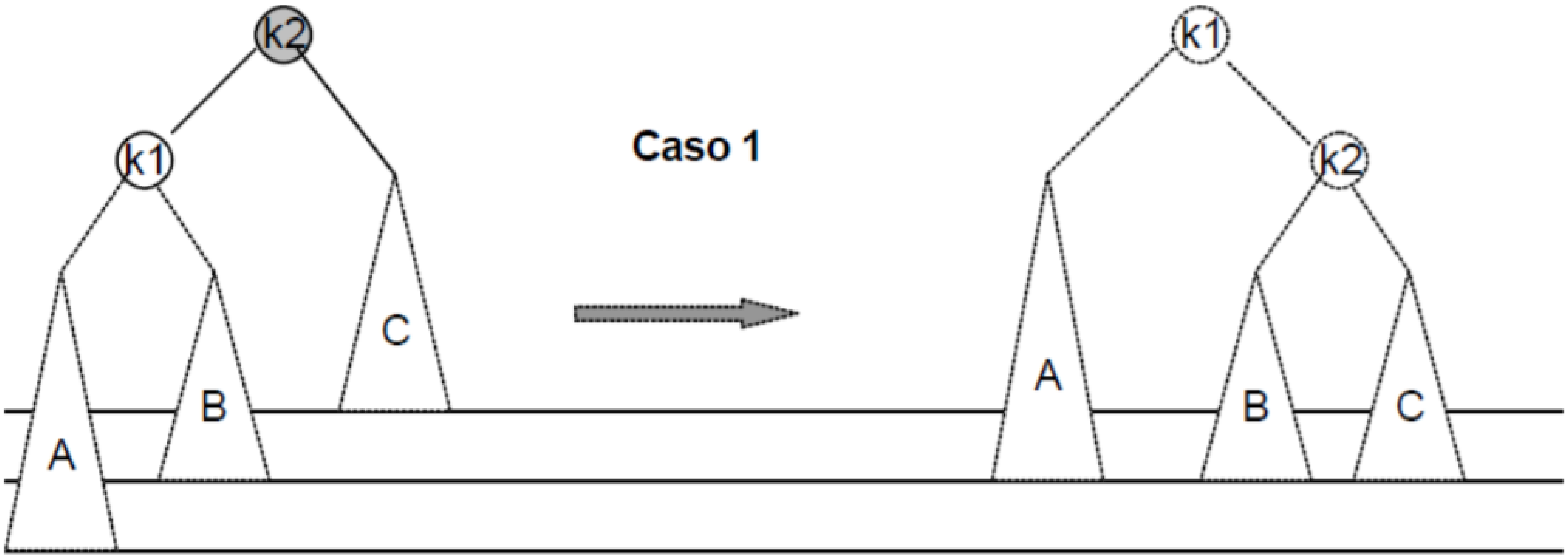
Ibmec
PROTAGONISTAS PARA O MUNDO

Árvores AVL

- Em uma árvore AVL, cada nó recebe um rótulo adicional que indica o seu fator de balanceamento. Esse rótulo pode ter os valores -1, 0 e 1. Se o fator ficar abaixo de -1 ou acima de 1, então a árvore precisa ser balanceada.
- No processo de inserção dos nós podem ser realizadas quatro tipos possíveis de transformações na árvore, para obter o seu balanceamento:
  - Rotação direita
  - Rotação esquerda
  - Rotação dupla direita
  - Rotação dupla esquerda

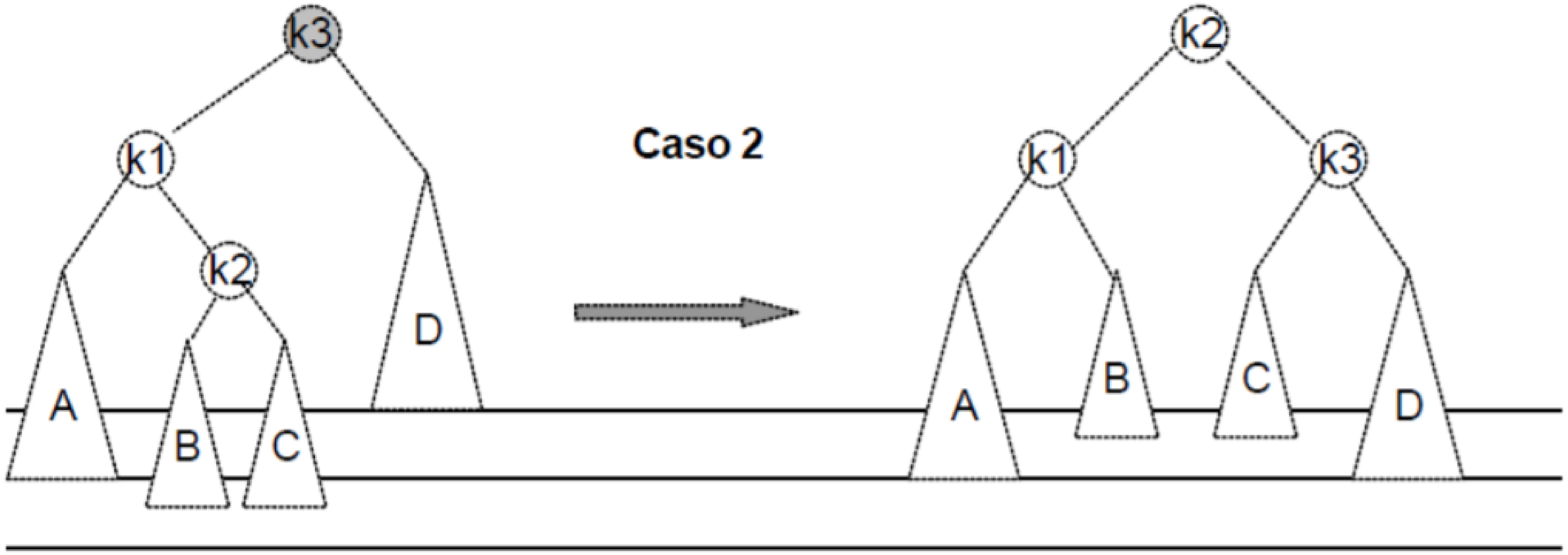
Árvores AVL

- Rotação simples
  - k2 é o nó mais profundo onde falha o equilíbrio
  - Subárvore esquerda está 2 níveis abaixo da direita



Árvores AVL

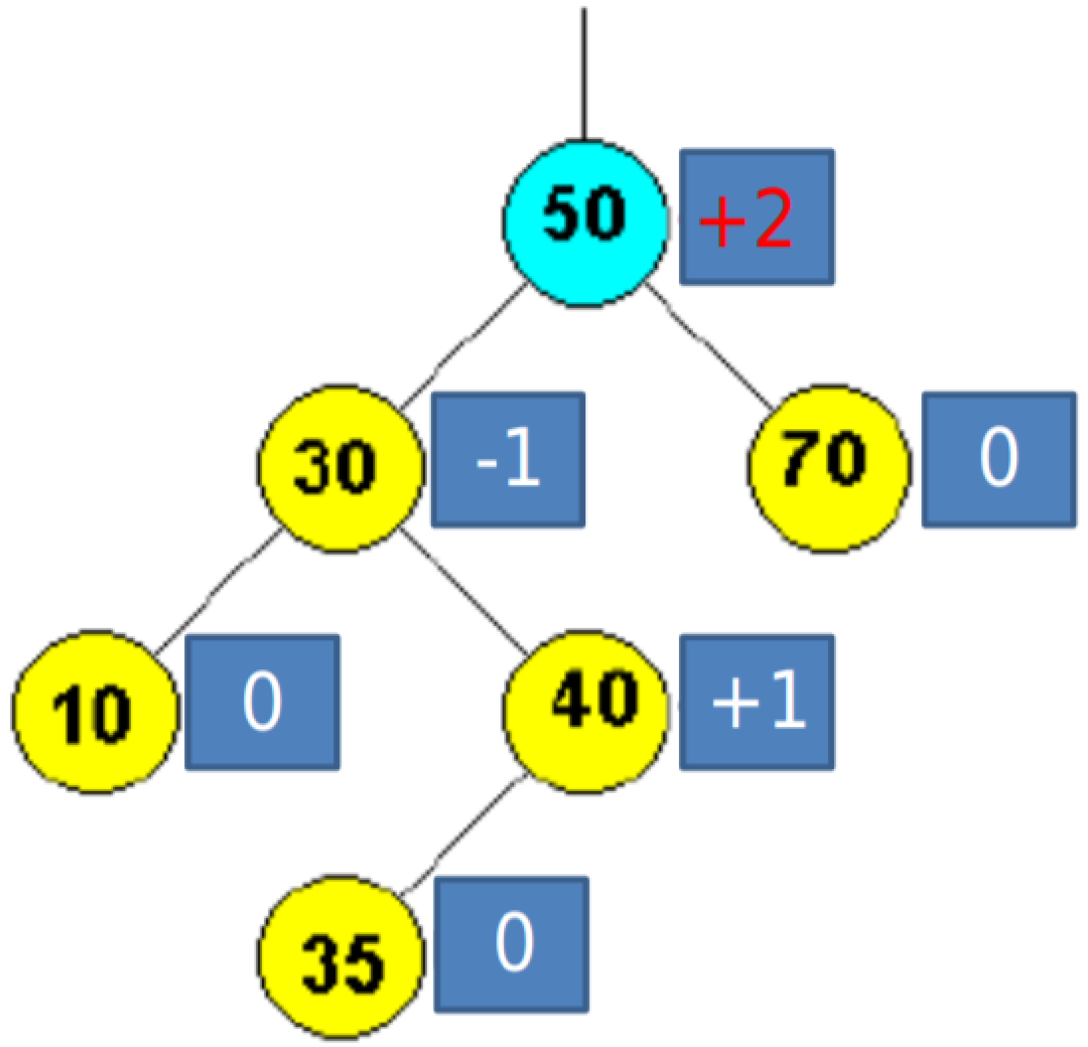
- Rotação dupla
  - Uma das subárvores B ou C está 2 níveis abaixo de D
  - k2, a chave intermédia, fica na raiz
  - Posições de k1, k3 e subárvores completamente determinadas pela rotação



Fator de balanceamento

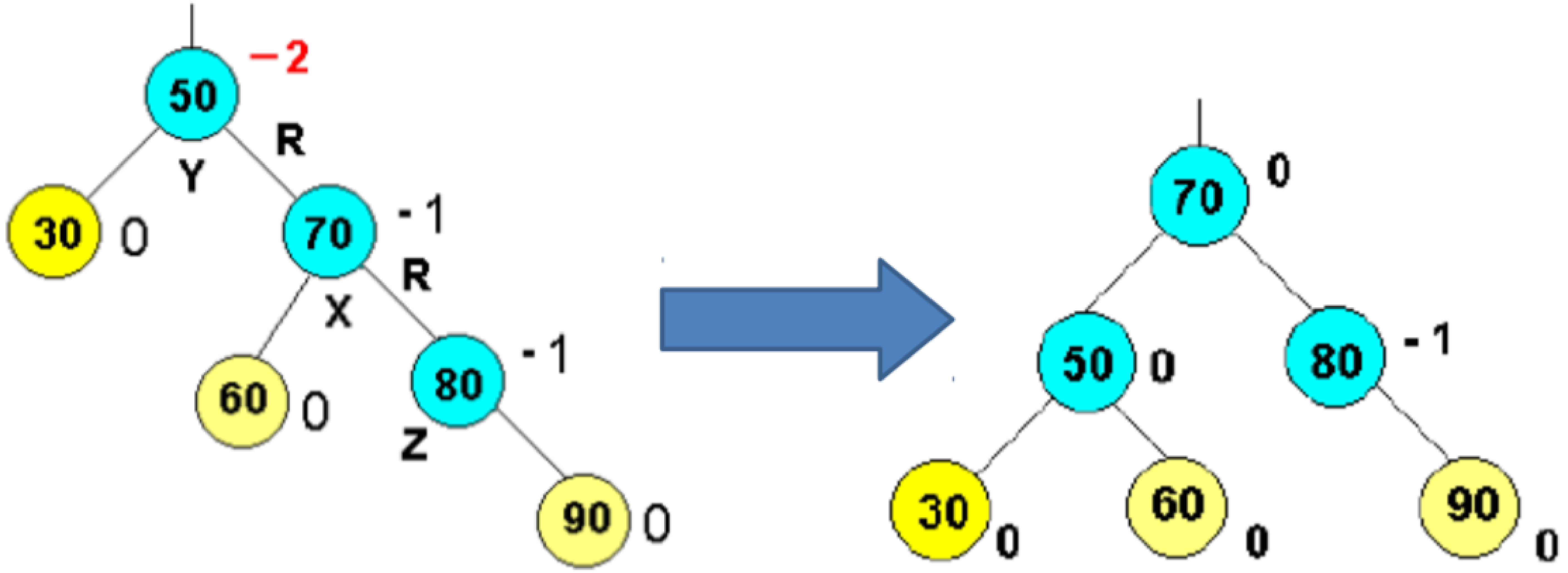
- Coeficiente que serve como referência para verificar se uma árvore AVL está ou não balanceada.
- O fator é calculado nó a nó e leva em consideração a diferença das alturas das subárvores da direita e da esquerda

$$FB = h_e - h_d$$



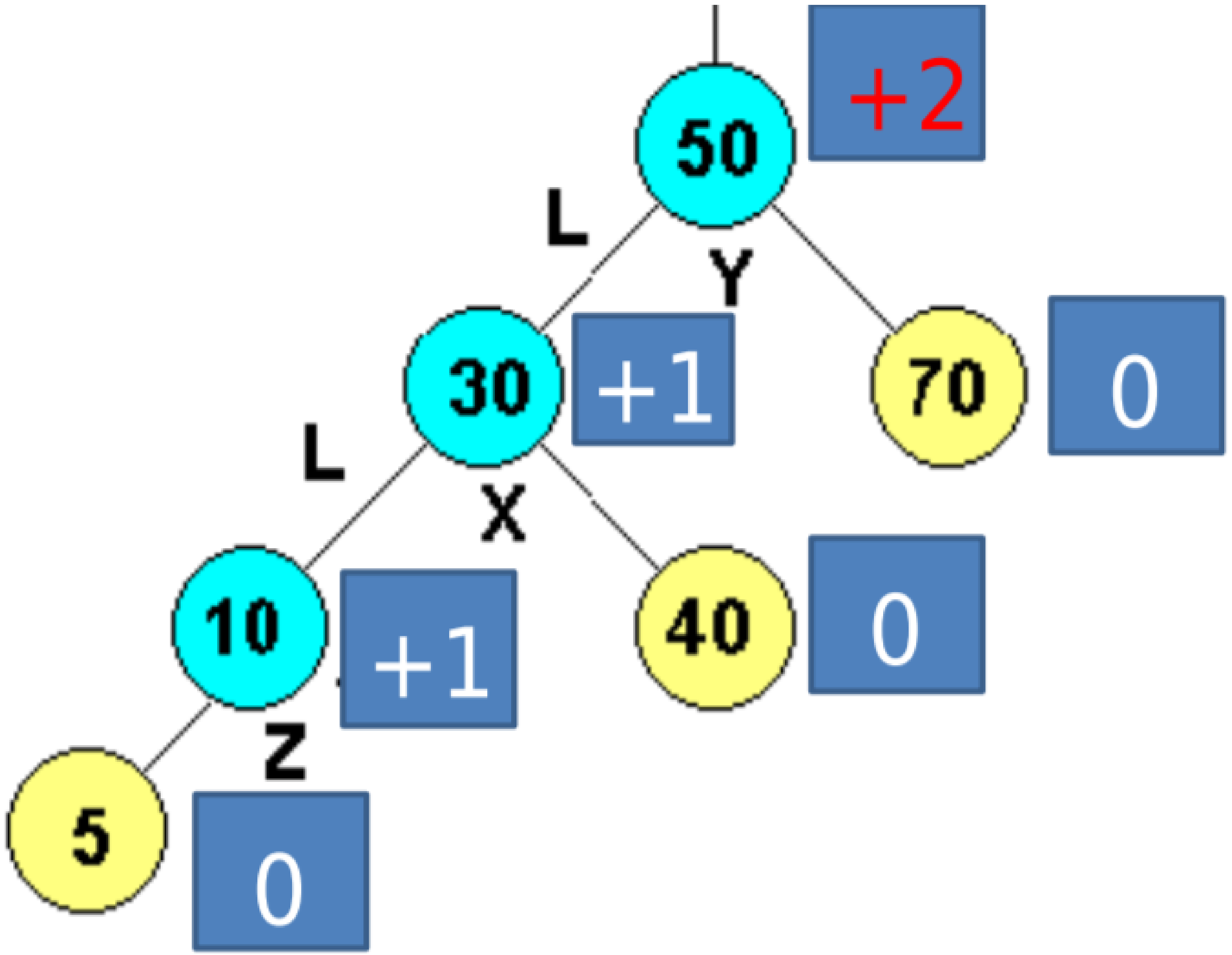
Quando balancear?

- Sempre que existir um fator de balanceamento superior a +1 ou inferior a -1
- Caso exista mais de um nó que se encaixe neste perfil deve-se sempre balancear o nó com o nível mais alto.
- Como balancear? Utilizando um dos processos de transformação (rotações simples ou duplas, à esquerda ou à direita).

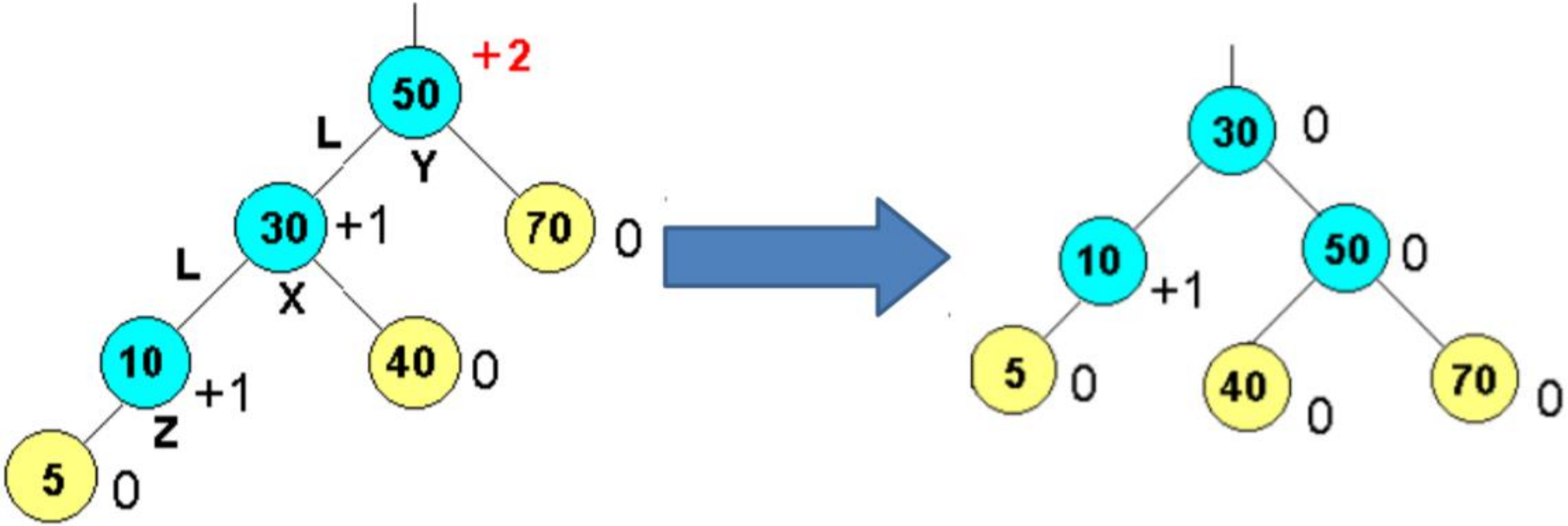


PROTAGONISTAS PARA O MUNDO

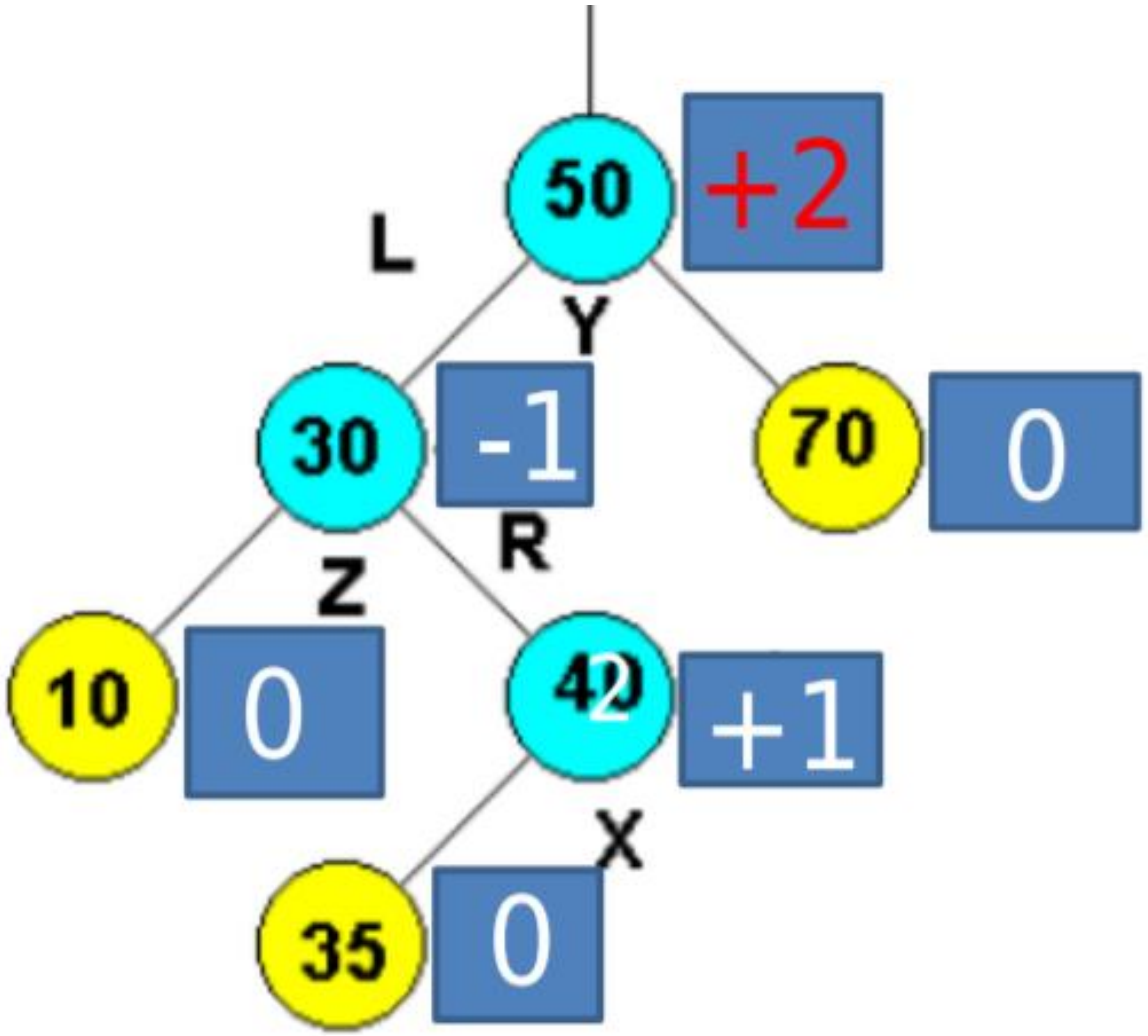
Como balancear?



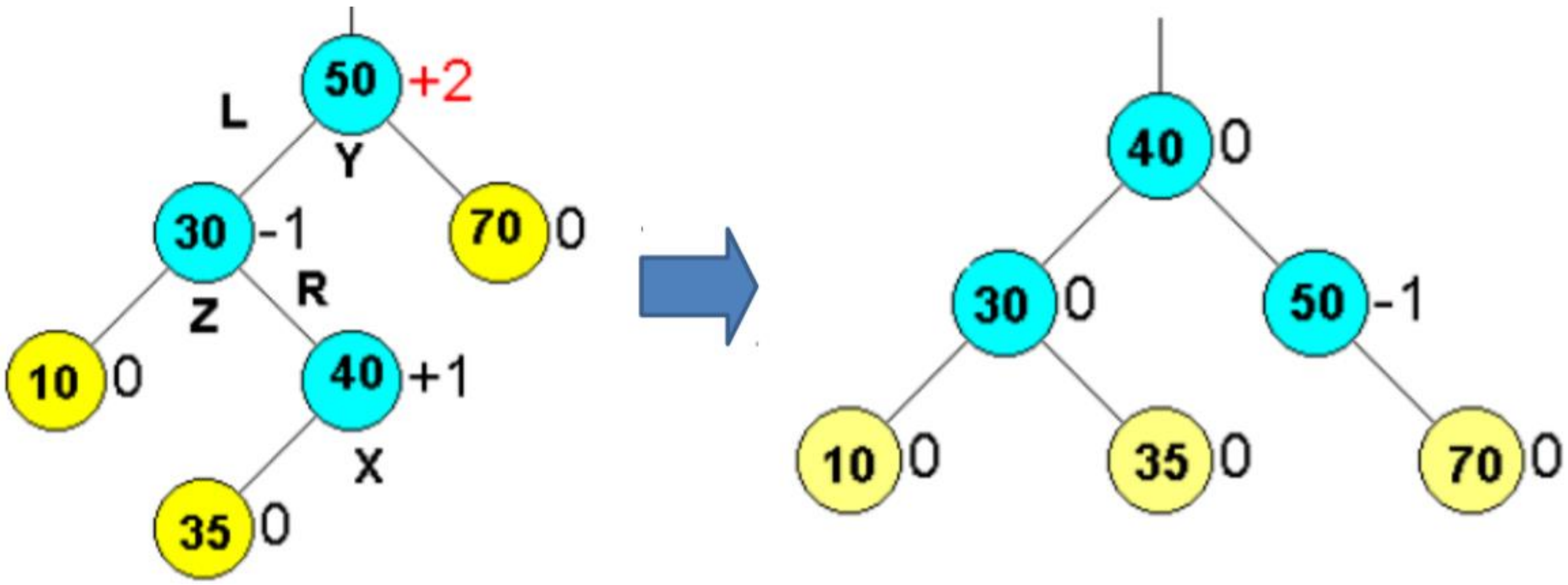
Como balancear?



Como balancear?

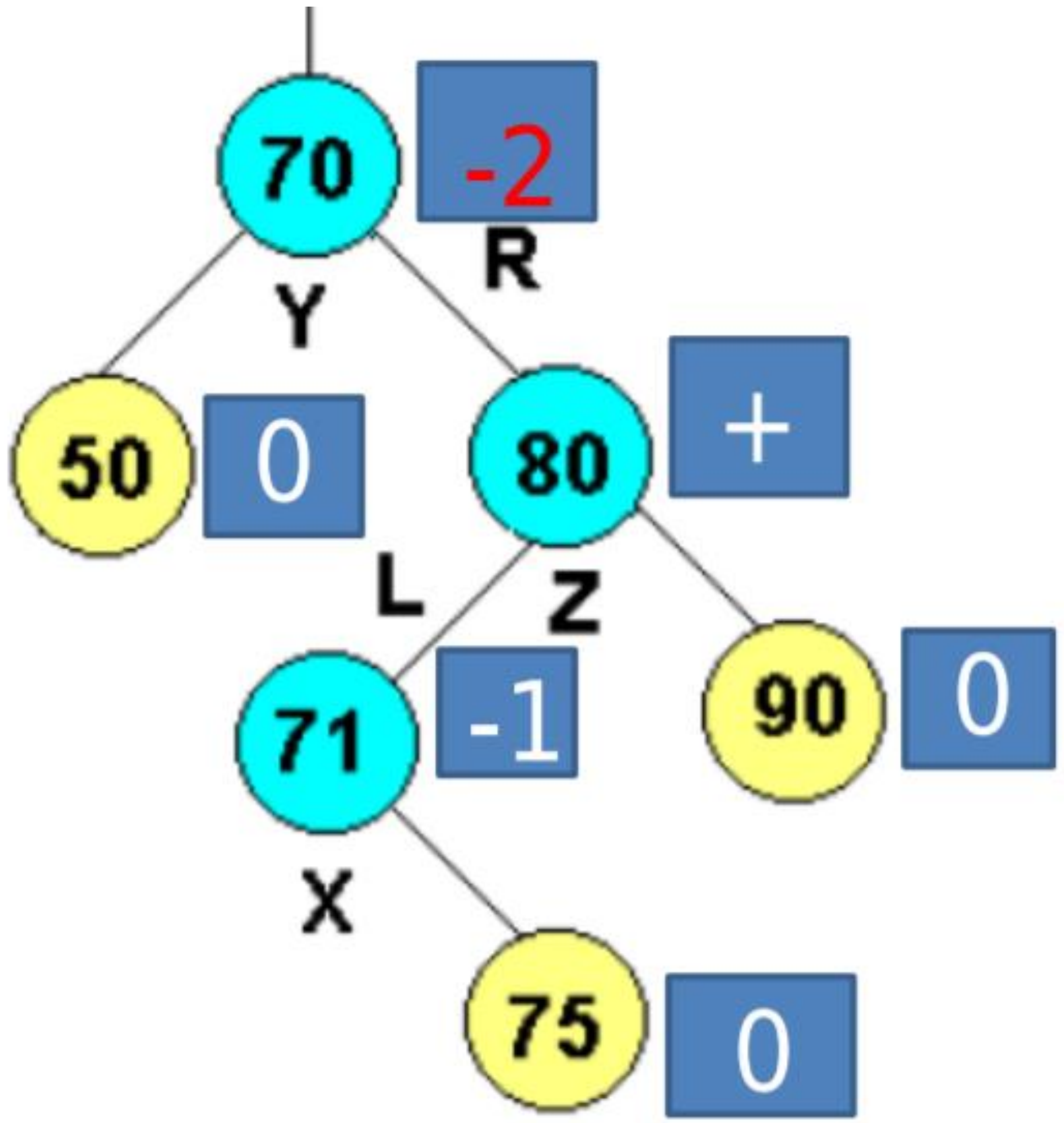


Como balancear?

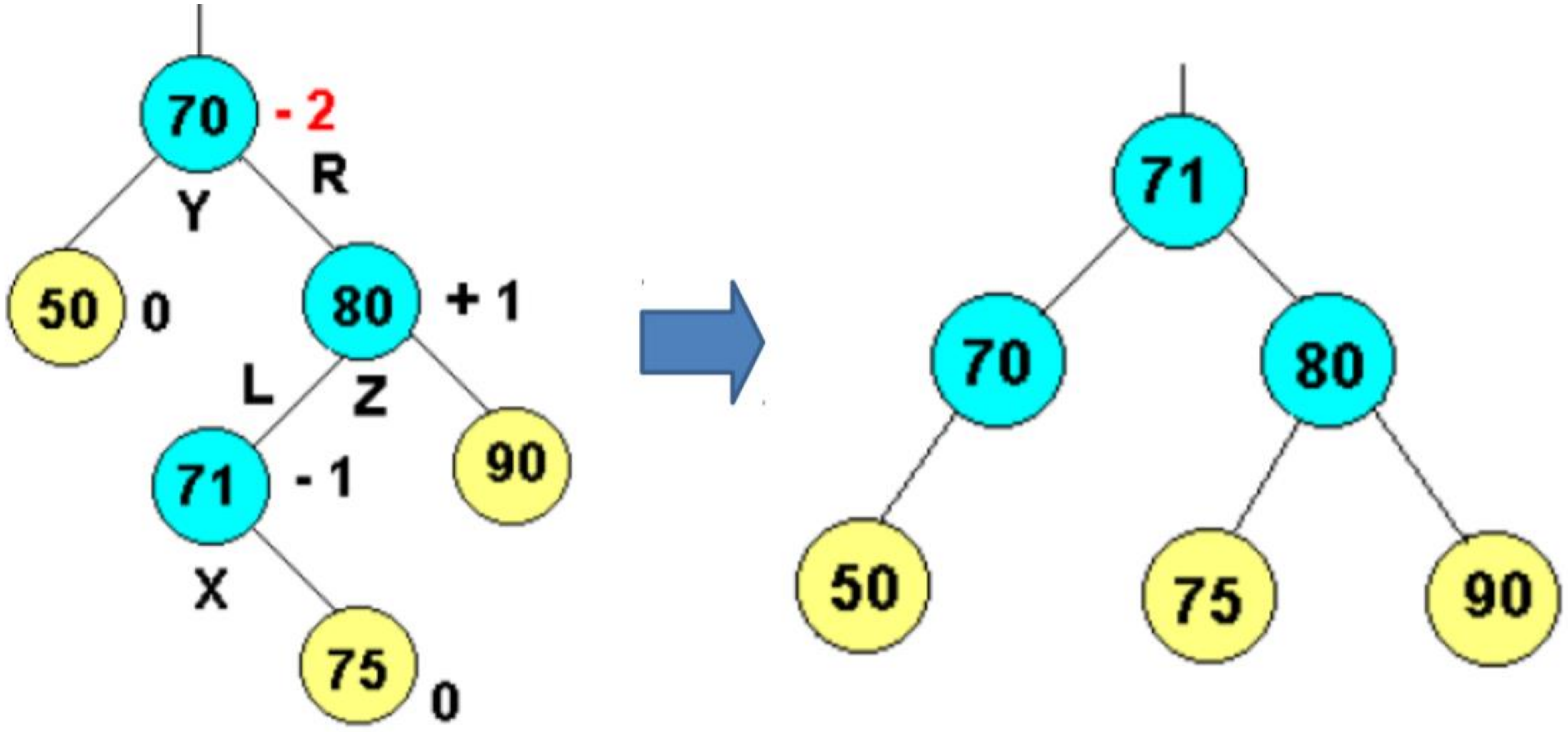


PROTAGONISTAS PARA O MUNDO

Como balancear?



Como balancear?



PROTAGONISTAS PARA O MUNDO

Algoritmo

- Inserir nó com chave X numa árvore A
  - Recursivamente, inserir na subárvore conveniente de A, SA;
  - Se a altura de SA não se modifica, terminar;
  - Se a altura de SA é modificada: se ocorre desequilíbrio em A, fazer as rotações necessárias para reequilibrar.
- Comparação das alturas
  - Para evitar o cálculo repetido de alturas de subárvores, pode-se manter em cada nó o resultado da comparação das alturas das subárvores.

Ibmec
PROTAGONISTAS PARA O MUNDO

Importante

- Na inserção, caso a árvore AVL esteja desbalanceada, basta 1 operação de rotação para rebalanceá-la.
- Na remoção, caso a árvore esteja desbalanceada, pode ser necessário até $\log n$ operações de rotação.

OBRIGADO!

www.ibmec.br

/ibmec

ibmec

@ibmec_oficial

ibmec

PROTAGONISTAS PARA O MUNDO