



UNIVERSIDADE
FEDERAL DE
SERGIPE



DEPARTAMENTO
DE COMPUTAÇÃO

Árvore binária de busca

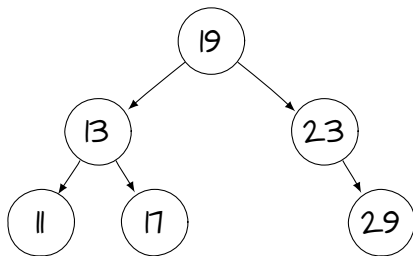
Estruturas de Dados

Bruno Prado

Departamento de Computação / UFS

Introdução

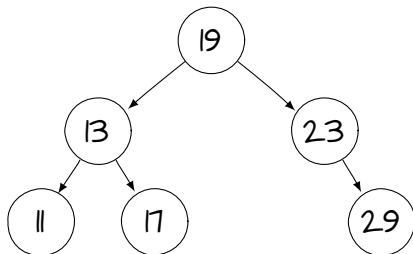
- ▶ O que é uma árvore binária de busca?
 - ▶ É uma árvore enraizada onde cada nó possui uma chave associada para realização da busca
 - ▶ Esquerda: menores ou iguais
 - ▶ Direita: maiores ou iguais



Introdução

- ▶ Propriedades

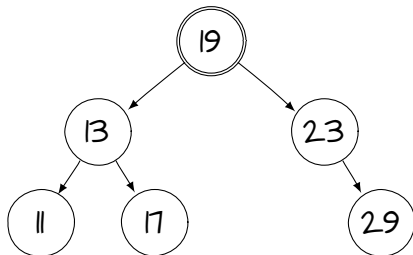
- ▶ A realização do percurso em ordem em uma árvore binária fornece a sequência ordenada dos elementos



Introdução

► Propriedades

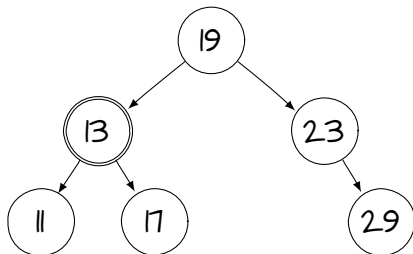
- A realização do percurso em ordem em uma árvore binária fornece a sequência ordenada dos elementos



Introdução

- ▶ Propriedades

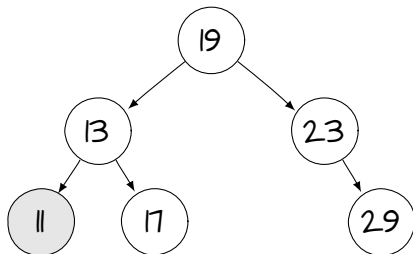
- ▶ A realização do percurso em ordem em uma árvore binária fornece a sequência ordenada dos elementos



Introdução

- ▶ Propriedades

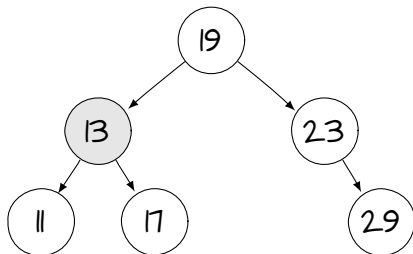
- ▶ A realização do percurso em ordem em uma árvore binária fornece a sequência ordenada dos elementos



Introdução

► Propriedades

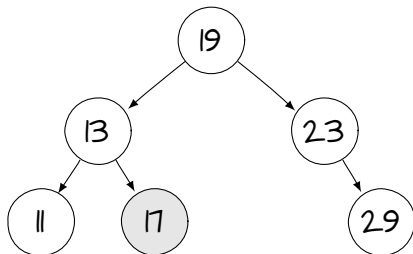
- A realização do percurso em ordem em uma árvore binária fornece a sequência ordenada dos elementos



Introdução

- ▶ Propriedades

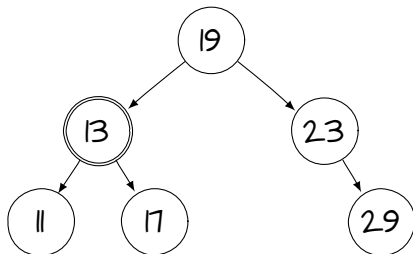
- ▶ A realização do percurso em ordem em uma árvore binária fornece a sequência ordenada dos elementos



Introdução

► Propriedades

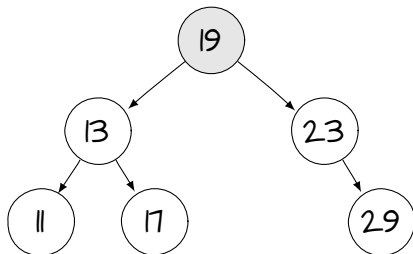
- A realização do percurso em ordem em uma árvore binária fornece a sequência ordenada dos elementos



Introdução

- ▶ Propriedades

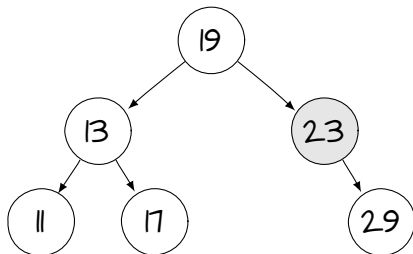
- ▶ A realização do percurso em ordem em uma árvore binária fornece a sequência ordenada dos elementos



Introdução

- ▶ Propriedades

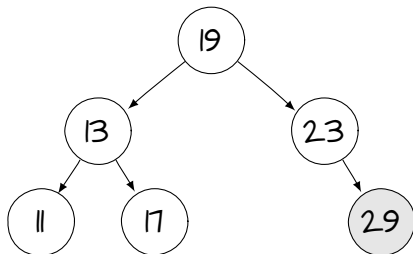
- ▶ A realização do percurso em ordem em uma árvore binária fornece a sequência ordenada dos elementos



Introdução

- ▶ Propriedades

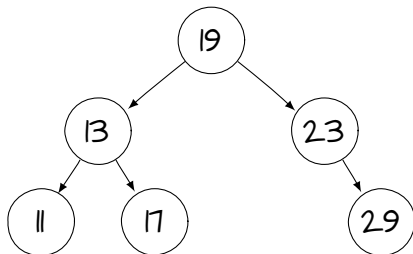
- ▶ A realização do percurso em ordem em uma árvore binária fornece a sequência ordenada dos elementos



Introdução

- ▶ Propriedades

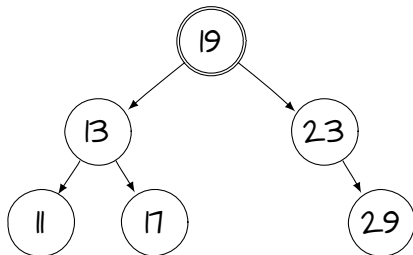
- ▶ Para encontrar os nós mínimo e máximo, basta realizar um percurso até cada extremidade da árvore



Introdução

► Propriedades

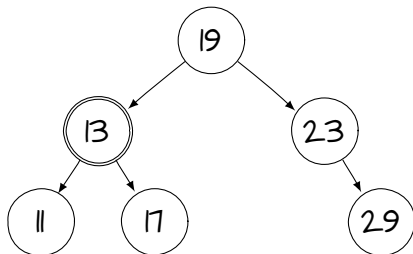
- Para encontrar os nós mínimo e máximo, basta realizar um percurso até cada extremidade da árvore



Introdução

- ▶ Propriedades

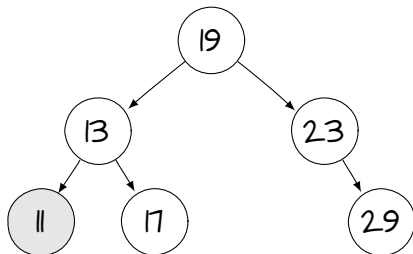
- ▶ Para encontrar os nós mínimo e máximo, basta realizar um percurso até cada extremidade da árvore



Introdução

► Propriedades

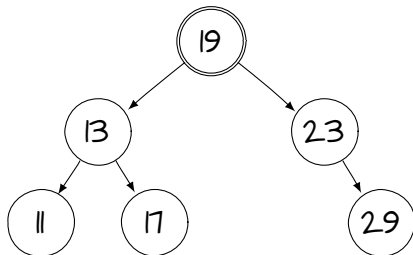
- Para encontrar os nós mínimo e máximo, basta realizar um percurso até cada extremidade da árvore



Introdução

► Propriedades

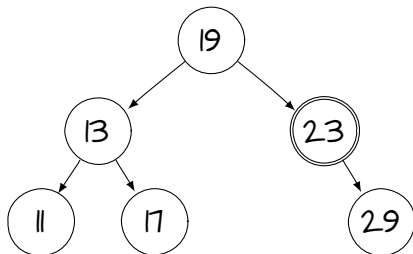
- Para encontrar os nós mínimo e máximo, basta realizar um percurso até cada extremidade da árvore



Introdução

- ▶ Propriedades

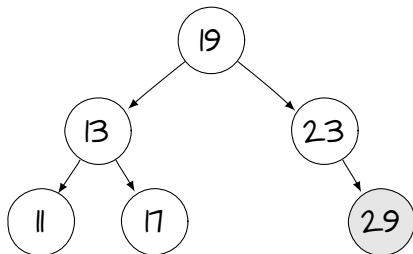
- ▶ Para encontrar os nós mínimo e máximo, basta realizar um percurso até cada extremidade da árvore



Introdução

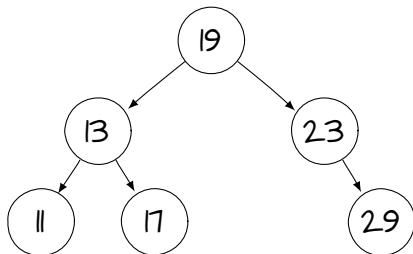
- ▶ Propriedades

- ▶ Para encontrar os nós mínimo e máximo, basta realizar um percurso até cada extremidade da árvore



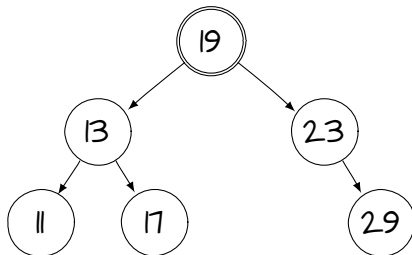
Introdução

- ▶ Propriedades
 - ▶ Particionamento por chave k da árvore em duas subárvores com elementos menores e maiores que 17



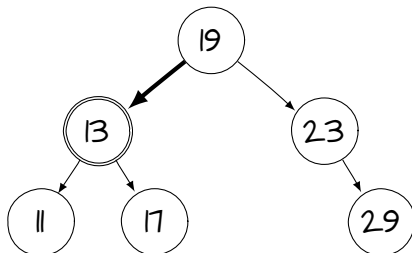
Introdução

- ▶ Propriedades
 - ▶ Particionamento por chave k da árvore em duas subárvores com elementos menores e maiores que 17



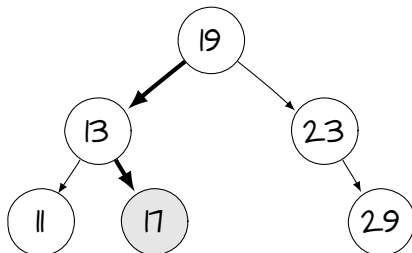
Introdução

- ▶ Propriedades
 - ▶ Particionamento por chave k da árvore em duas subárvores com elementos menores e maiores que 17



Introdução

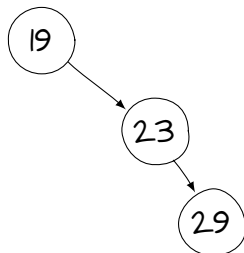
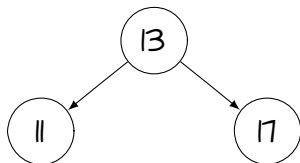
- ▶ Propriedades
 - ▶ Particionamento por chave k da árvore em duas subárvores com elementos menores e maiores que 17



Introdução

► Propriedades

- Particionamento por chave k da árvore em duas subárvores com elementos menores e maiores que 17

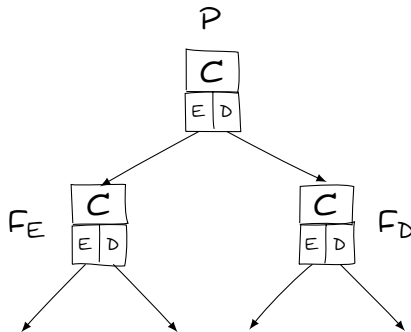


Árvore binária de busca

- ▶ Operações básicas
 - ▶ Busca
 - ▶ Inserção
 - ▶ Remoção

Árvore binária de busca

► Definição da estrutura



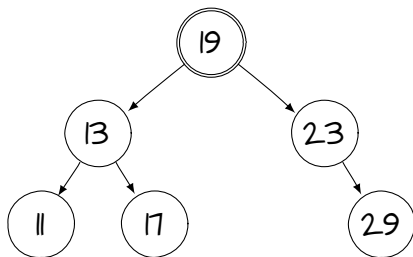
Árvore binária de busca

- ▶ Implementação em C
 - ▶ Estrutura e ponteiros

```
1 // Padrão de tipos por tamanho
2 #include <stdint.h>
3 // Estrutura de nó
4 typedef struct no {
5     // Chave do nó
6     uint32_t C;
7     // Filho da direita
8     struct no* D;
9     // Filho da esquerda
10    struct no* E;
11 } no;
```

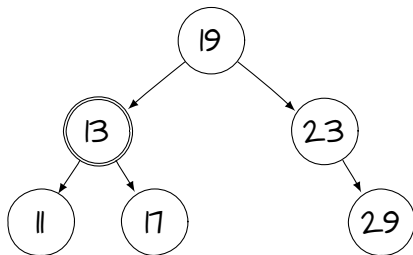
Árvore binária de busca

- ▶ Operação de busca
 - ▶ Parâmetro de chave: 17
 - ▶ A busca tem início pelo elemento raiz da árvore, comparando o valor de sua chave com o parâmetro de busca



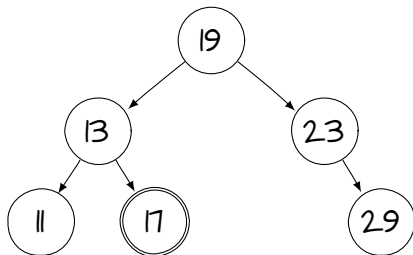
Árvore binária de busca

- ▶ Operação de busca
 - ▶ Parâmetro de chave: 17
 - ▶ Como o resultado da comparação indica que o valor é maior do que o procurado, a busca é aplicada na subárvore esquerda



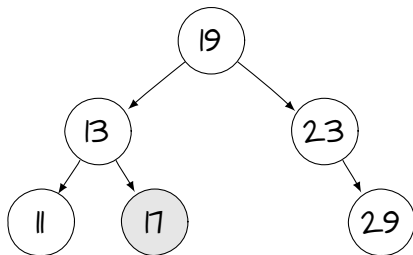
Árvore binária de busca

- ▶ Operação de busca
 - ▶ Parâmetro de chave: 17
 - ▶ Como a comparação da chave do nó da subárvore indica que o parâmetro procurado é maior, a busca é aplicada na subárvore direita



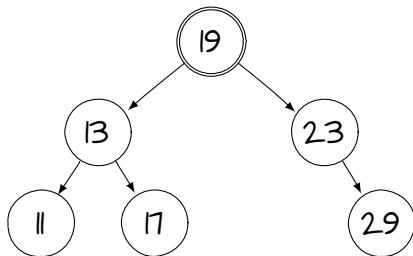
Árvore binária de busca

- ▶ Operação de busca
 - ▶ Parâmetro de chave: 17
 - ▶ A chave do nó é igual ao parâmetro de busca e sua referência é retornada



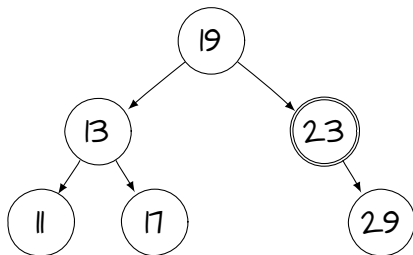
Árvore binária de busca

- ▶ Operação de inserção
 - ▶ Parâmetro de chave: 21
 - ▶ É realizada uma operação de busca utilizando a chave do elemento que será inserido até encontrar uma referência nula



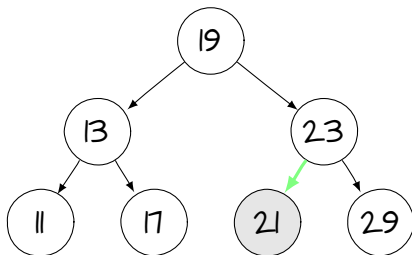
Árvore binária de busca

- ▶ Operação de inserção
 - ▶ Parâmetro de chave: 21
 - ▶ É realizada uma operação de busca utilizando a chave do elemento que será inserido até encontrar uma referência nula



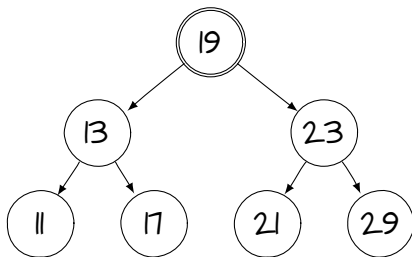
Árvore binária de busca

- ▶ Operação de inserção
 - ▶ Parâmetro de chave: 21
 - ▶ A subárvore esquerda do nó 23 é nula, é feita a alocação do nó para inserção na árvore



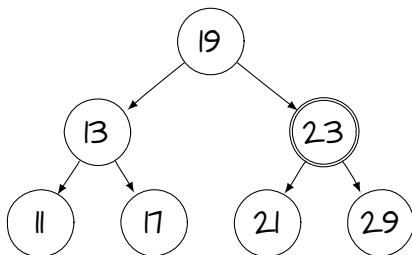
Árvore binária de busca

- ▶ Operação de remoção
 - ▶ Caso 1: o nó removido é uma folha
 - ▶ Parâmetro de chave: 29
 - ▶ É feita a busca pela chave do elemento



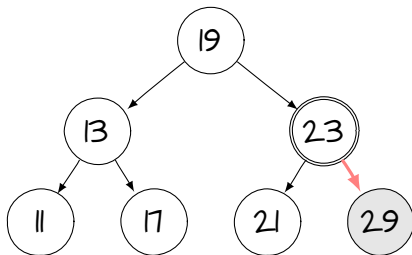
Árvore binária de busca

- ▶ Operação de remoção
 - ▶ Caso 1: o nó removido é uma folha
 - ▶ Parâmetro de chave: 29
 - ▶ É feita a busca pela chave do elemento



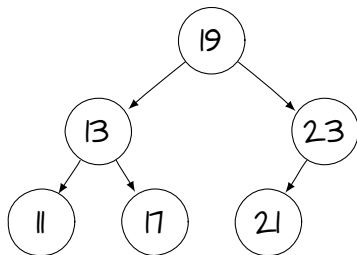
Árvore binária de busca

- ▶ Operação de remoção
 - ▶ Caso 1: o nó removido é uma folha
 - ▶ Parâmetro de chave: 29
 - ▶ É feita a desalocação do elemento e sua referência é anulada para remoção da árvore



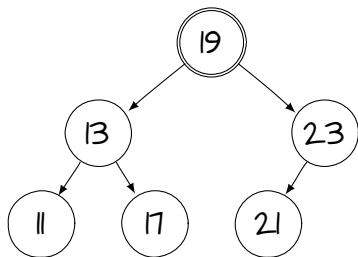
Árvore binária de busca

- ▶ Operação de remoção
 - ▶ Caso 1: o nó removido é uma folha
 - ▶ Parâmetro de chave: 29
 - ▶ É feita a desalocação do elemento e sua referência é anulada para remoção da árvore



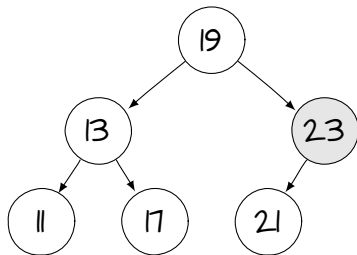
Árvore binária de busca

- ▶ Operação de remoção
 - ▶ Caso 2: o nó removido possui uma subárvore
 - ▶ Parâmetro de chave: 23
 - ▶ É feita a busca pela chave do elemento



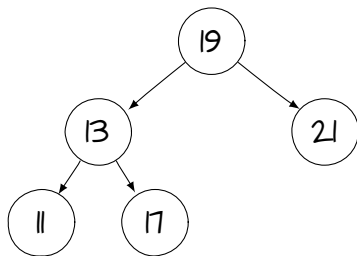
Árvore binária de busca

- ▶ Operação de remoção
 - ▶ Caso 2: o nó removido possui uma subárvore
 - ▶ Parâmetro de chave: 23
 - ▶ O elemento é removido e a referência do nó pai é atualizada para referenciar a subárvore



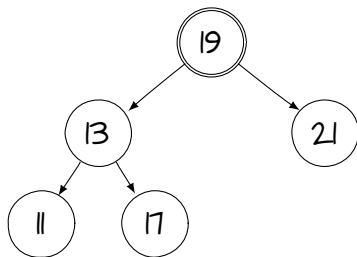
Árvore binária de busca

- ▶ Operação de remoção
 - ▶ Caso 2: o nó removido possui uma subárvore
 - ▶ Parâmetro de chave: 23
 - ▶ O elemento é removido e a referência do nó pai é atualizada para referenciar a subárvore



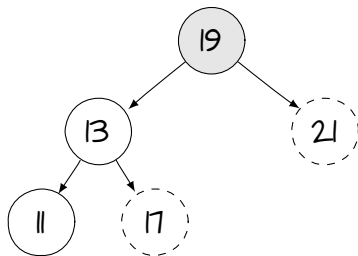
Árvore binária de busca

- ▶ Operação de remoção
 - ▶ Caso 3: o nó removido possui duas subárvores
 - ▶ Parâmetro de chave: 19
 - ▶ É feita a busca pela chave do elemento



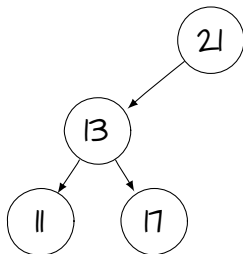
Árvore binária de busca

- ▶ Operação de remoção
 - ▶ Caso 3: o nó removido possui duas subárvores
 - ▶ Parâmetro de chave: 19
 - ▶ O elemento raiz é substituído pelo seu predecessor (17) ou sucessor (21) que será removido após a substituição



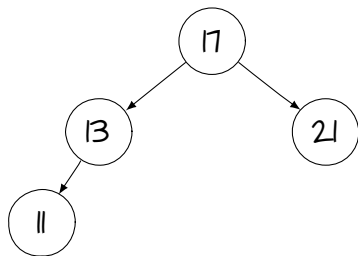
Árvore binária de busca

- ▶ Operação de remoção
 - ▶ Caso 3: o nó removido possui duas subárvores
 - ▶ Parâmetro de chave: 19
 - ▶ É feita a remoção do elemento de chave 21 e o ajuste dos ponteiros da árvore



Árvore binária de busca

- ▶ Operação de remoção
 - ▶ Caso 3: o nó removido possui duas subárvores
 - ▶ Parâmetro de chave: 19
 - ▶ É feita a remoção do elemento de chave 17 e o ajuste dos ponteiros da árvore



Árvore binária de busca

- ▶ Análise de complexidade
 - ▶ Espaço: $\Theta(n)$
 - ▶ Tempo: $\Omega(1)$ e $O(n)$

Árvore binária de busca

- ▶ Análise de complexidade
 - ▶ Espaço: $\Theta(n)$
 - ▶ Tempo: $\Omega(1)$ e $O(n)$

Árvore binária de altura h



$$\log_2 n \leq h \leq n$$

Exemplo

- ▶ Construa uma árvore binária de busca
 - ▶ Insira os elementos com chaves 13, 2, 34, 11, 7, 43 e 9
 - ▶ Realize a remoção dos elementos de chave 11 e 9
 - ▶ Explique as situações de melhor e de pior caso para as operações realizadas neste tipo de árvore

Exercício

- ▶ A empresa de tecnologia Poxim Tech está desenvolvendo um sistema de armazenamento de arquivos baseado em árvore binária de busca
 - ▶ O formato de nome dos arquivos é definido por uma cadeia com 1 até 50 caracteres, composta somente por letras, números e os símbolos '_' e '.'
 - ▶ Cada arquivo possui também informações de permissão de acesso para somente leitura (ro) e escrita e leitura (rw), além do tamanho em bytes
 - ▶ Caso um nome de arquivo repetido seja inserido, é feita a substituição das informações desde que o arquivo permita a escrita (rw)

Exercício

- ▶ Formato de arquivo de entrada
 - ▶ $[\# \text{Número de arquivos}]$
 - ▶ $[\text{Nome}_1] [\text{Tipo}_1] [\text{Tamanho}_1]$
 - ▶ \vdots
 - ▶ $[\text{Nome}_n] [\text{Tipo}_n] [\text{Tamanho}_n]$

```
1 5
2 lista_ed.c_rw_1123
3 senhas.txt_ro_144
4 foto.jpg_rw_8374719
5 documento.doc_rw_64732
6 lista_ed.c_ro_1
```

Exercício

- ▶ Formato de arquivo de saída
 - ▶ Percurso em ordem (*EPD*), pré-ordem (*PED*) e pós-ordem (*EDP*)

```
1  EPD:
2  3_documento.doc_rw_64732_bytes
3  2_foto.jpg_rw_8374719_bytes
4  4_lista_ed.c_ro_1_byte
5  1_senhas.txt_ro_144_bytes
6  PED:
7  4_lista_ed.c_ro_1_byte
8  2_foto.jpg_rw_8374719_bytes
9  3_documento.doc_rw_64732_bytes
10 1_senhas.txt_ro_144_bytes
11 EDP:
12 3_documento.doc_rw_64732_bytes
13 2_foto.jpg_rw_8374719_bytes
14 1_senhas.txt_ro_144_bytes
15 4_lista_ed.c_ro_1_byte
```