MC-202 — Aula 11
Árvore Binárias

Lehilton Pedrosa

Instituto de Computação – Unicamp

Segundo Semestre de 2015

Roteiro

Introdução

Problema

Você e um(a) amigo(a) estão jogando. Ele pensa em um animal de um certo conjunto e você tem que adivinhar que animal é. Você só pode fazer perguntas com respostas sim ou não!

Introdução

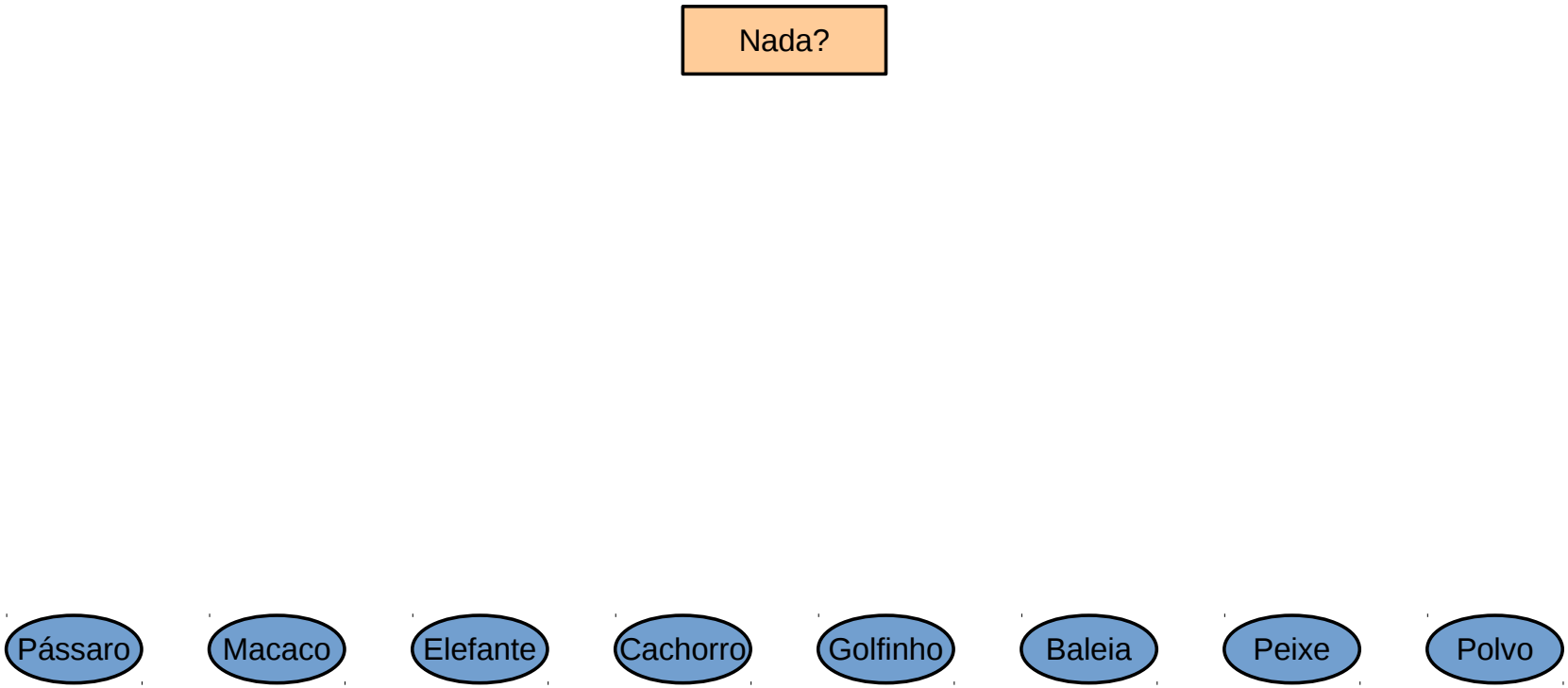
Problema

Você e um(a) amigo(a) estão jogando. Ele pensa em um animal de um certo conjunto e você tem que adivinhar que animal é. Você só pode fazer perguntas com respostas sim ou não!

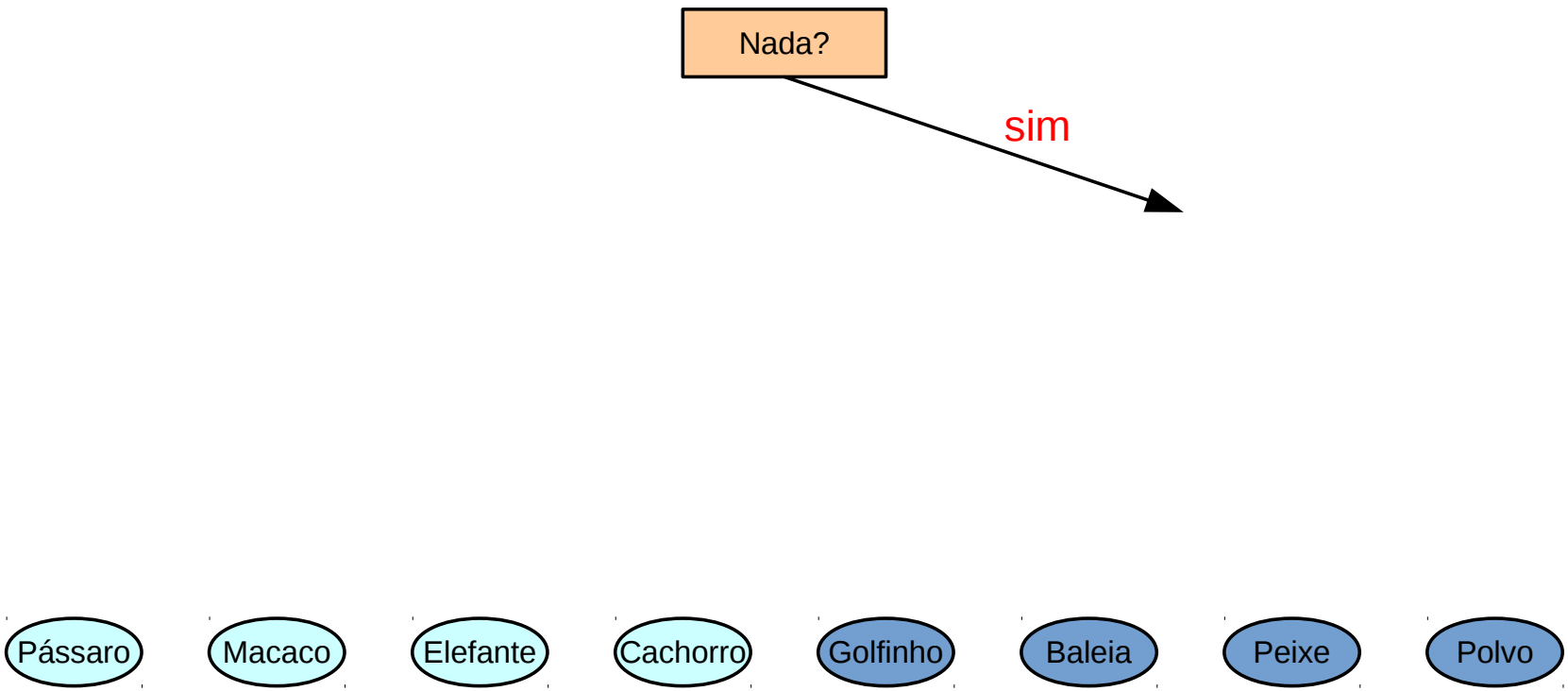
- **Pergunta:** Como fazer o menor número de perguntas?

Adivinhando

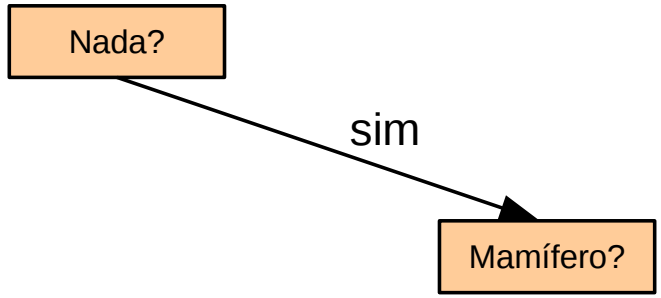
Adivinhando



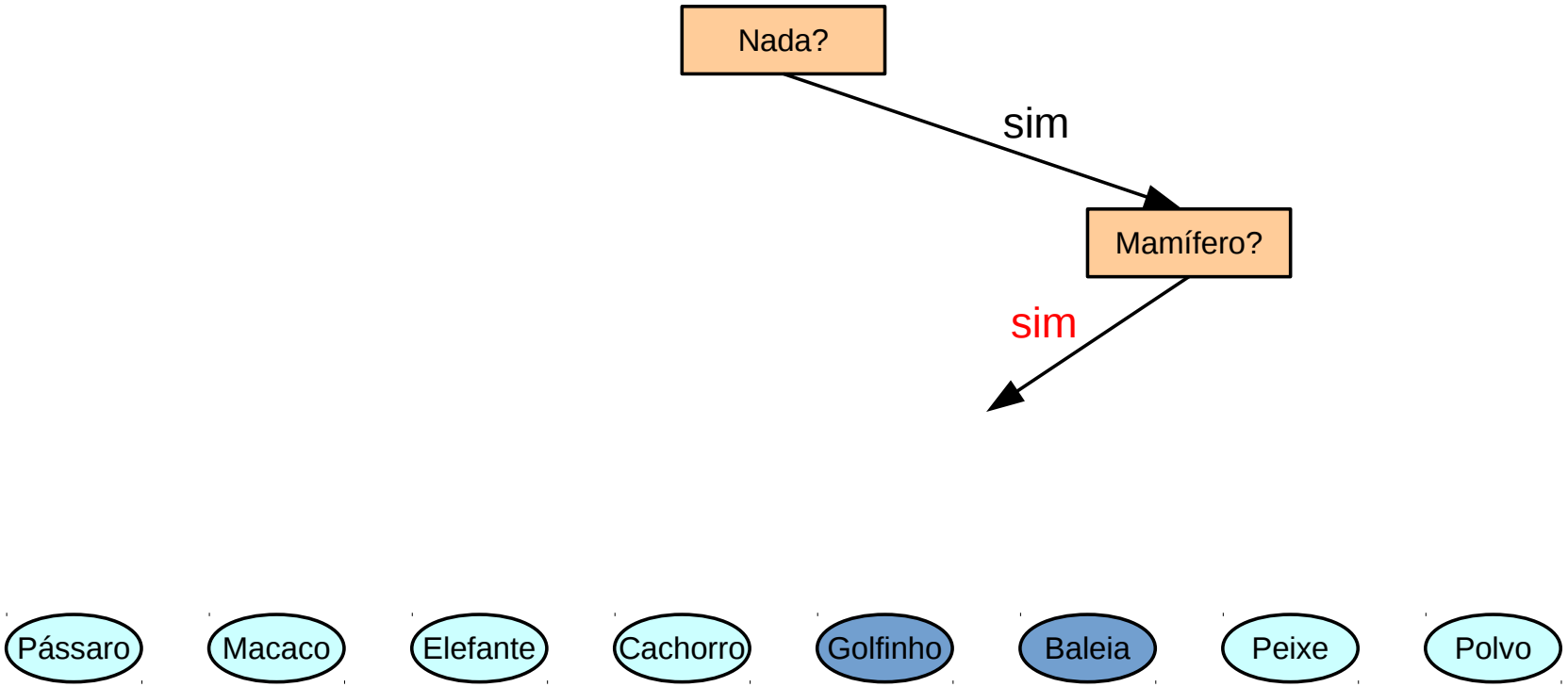
Adivinhando



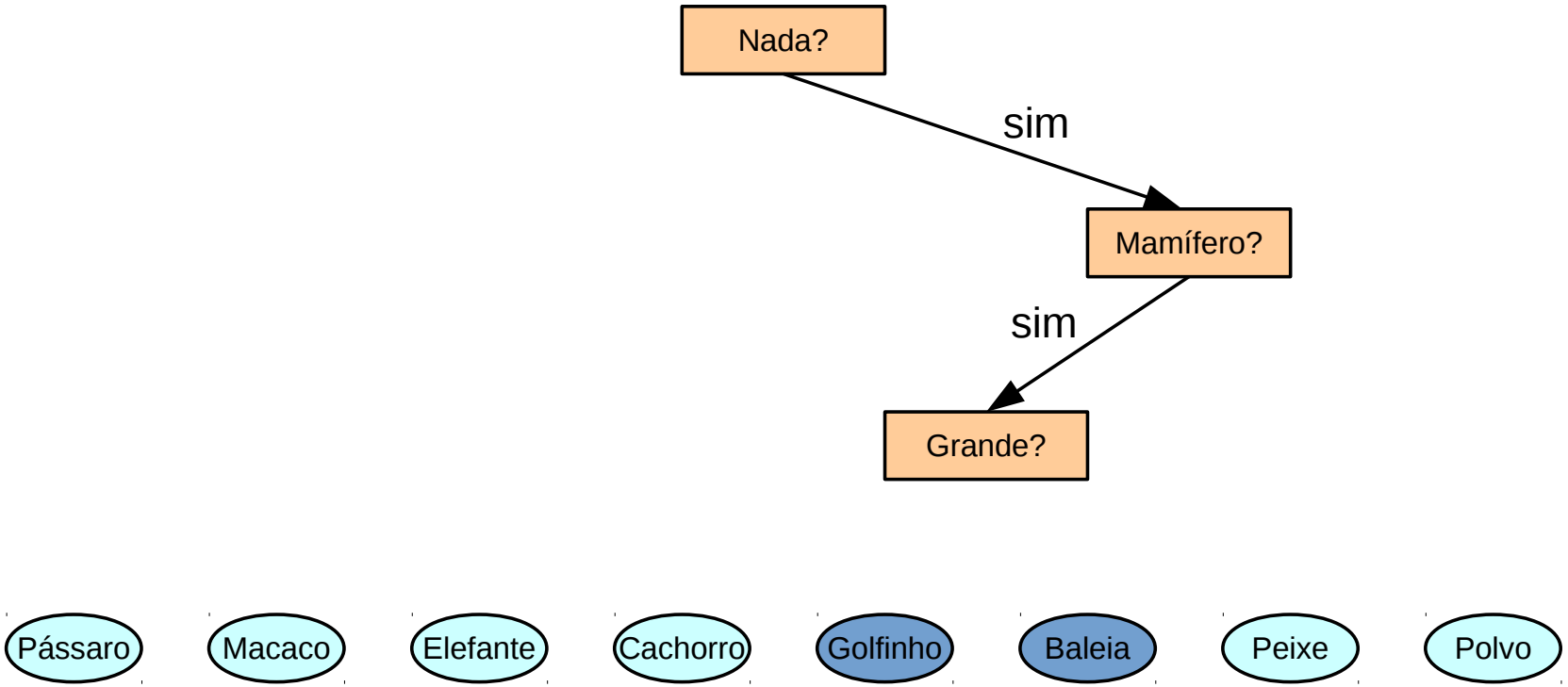
Adivinhando



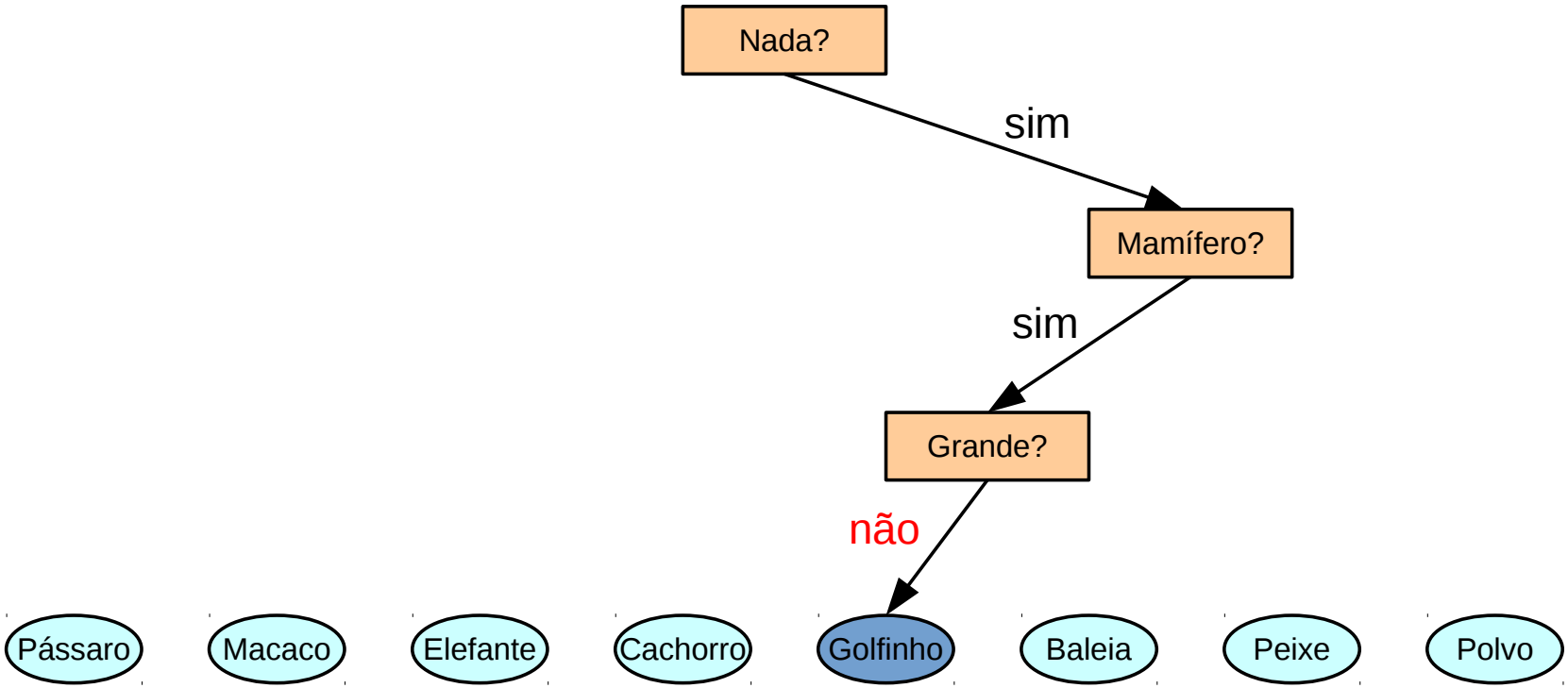
Adivinhando



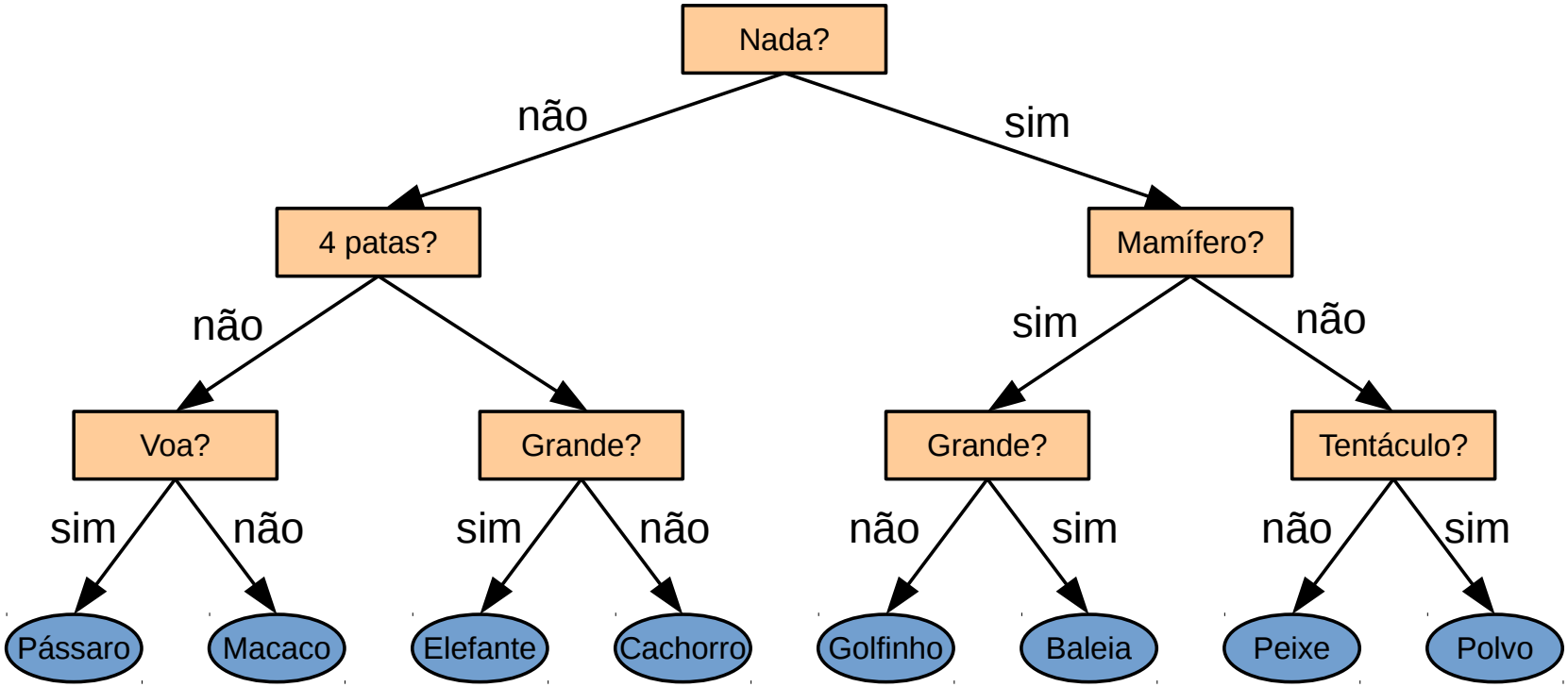
Adivinhando



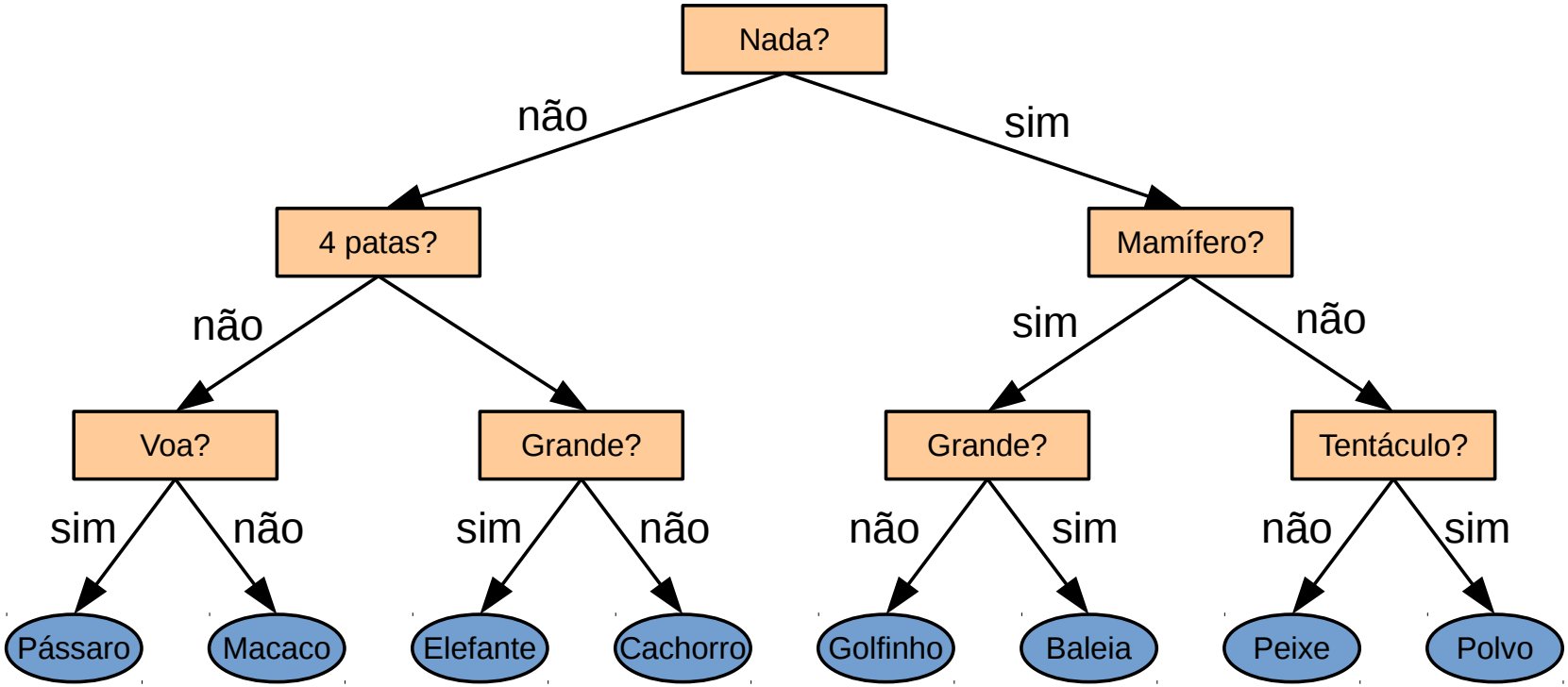
Adivinhando



Adivinhando



Adivinhando



Criamos uma árvore de decisão!

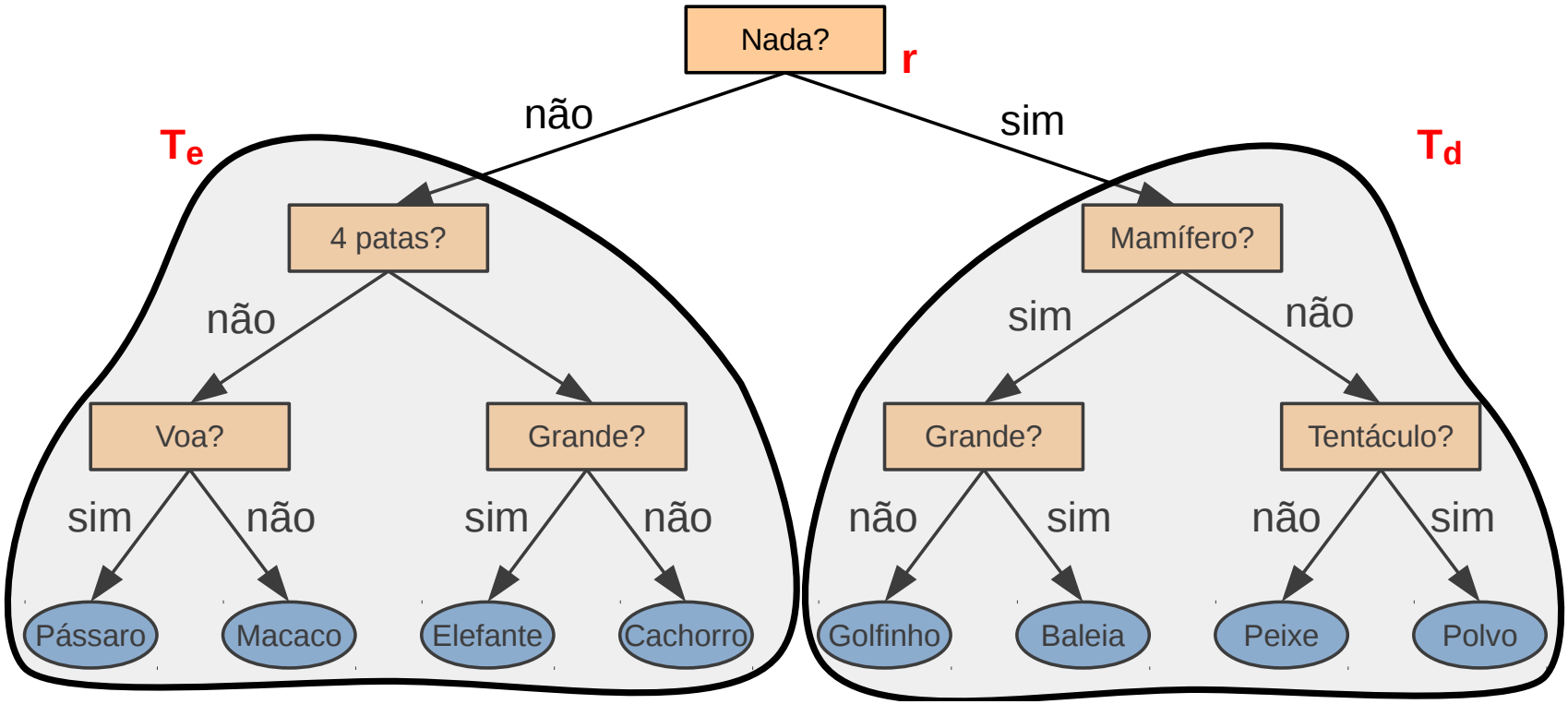
Árvore binária

Definição

Uma árvore binária é um conjunto de nós T tal que ou T é vazio, ou

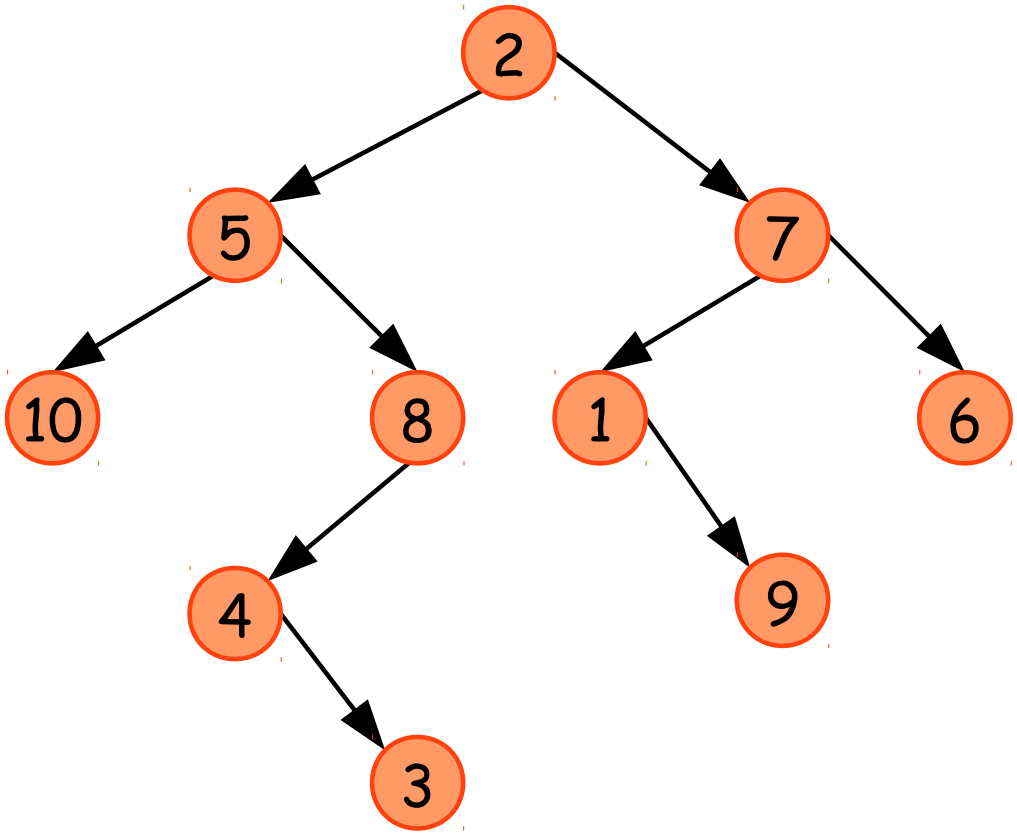
- T contém um elemento especial r, denominado **raiz**
- $T \setminus \{r\}$ é a união de duas subárvores disjuntas, T_e e T_d, denominadas *filho esquerdo* e *filho direito*.

Exemplo com definição



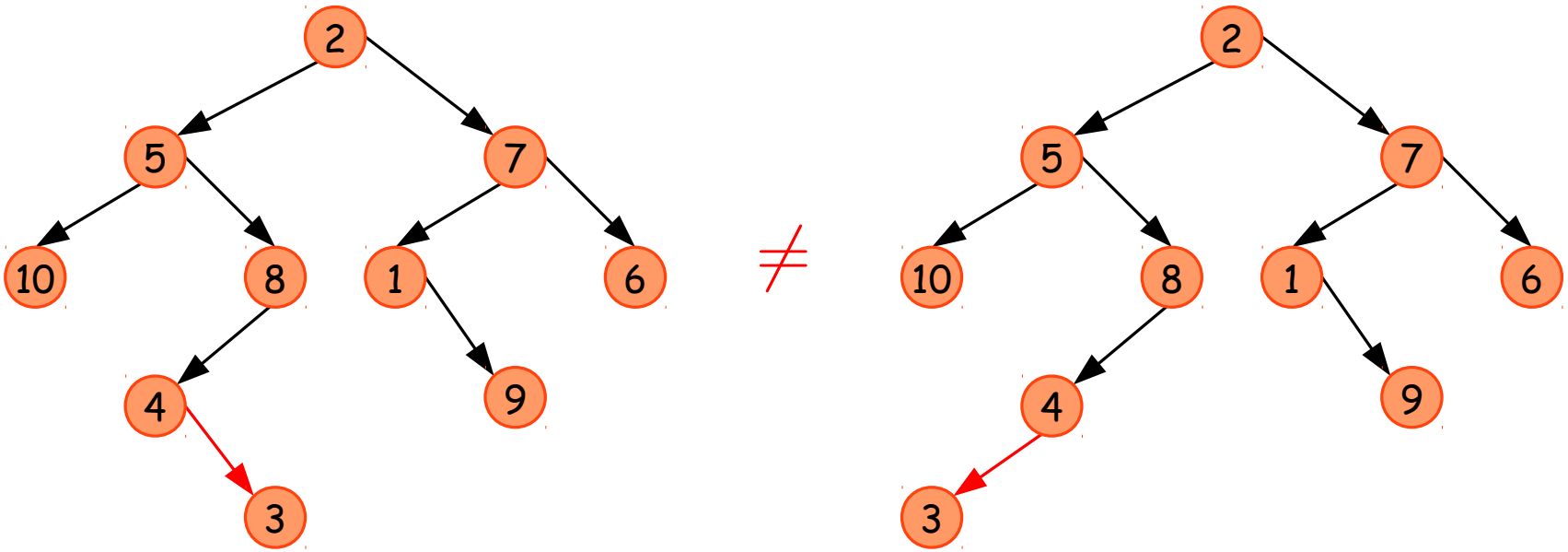
Cada parte!

Uma árvore com números

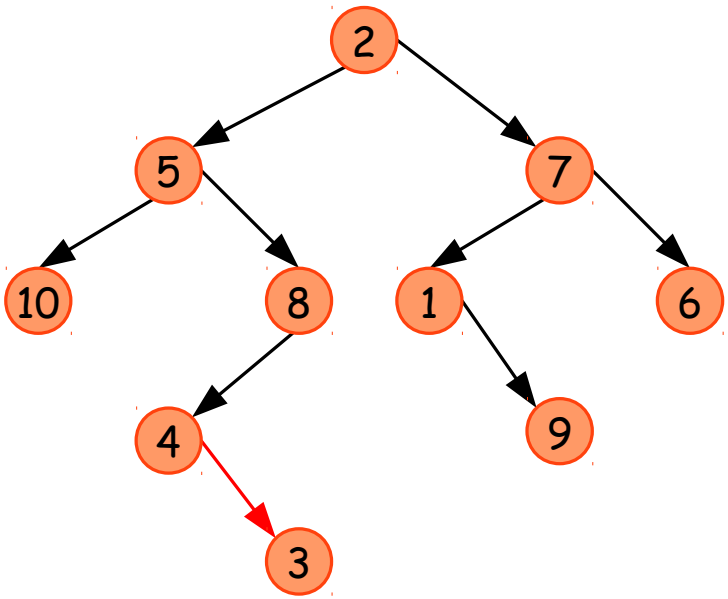


Exemplo mais simples.

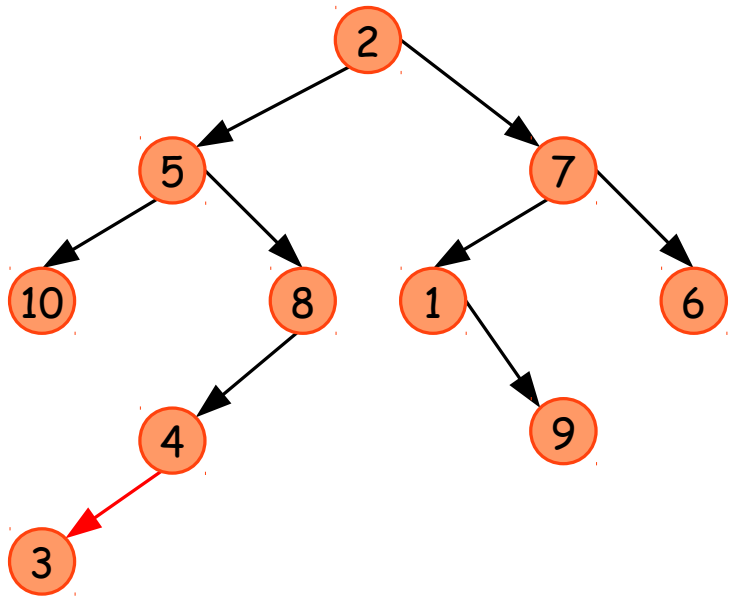
Comparando com atenção



Comparando com atenção



$\neq$



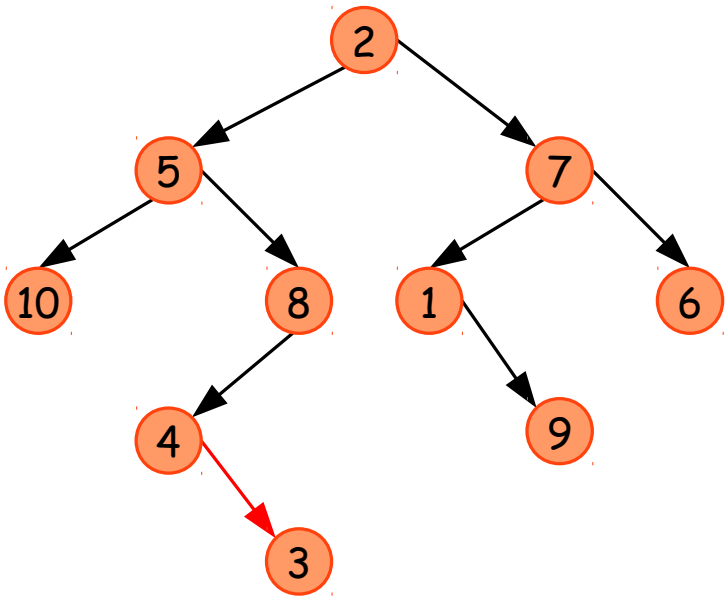
Filhos de 4:

- *esquerdo:* vazio
- *direito:* 3

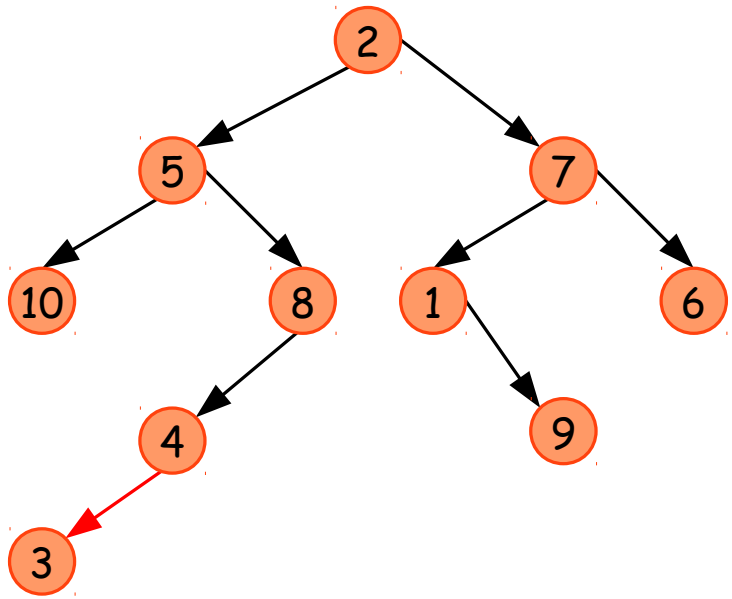
Filhos de 4:

- *esquerdo:* 3
- *direito:* vazio

Comparando com atenção



$\neq$



Filhos de 4:

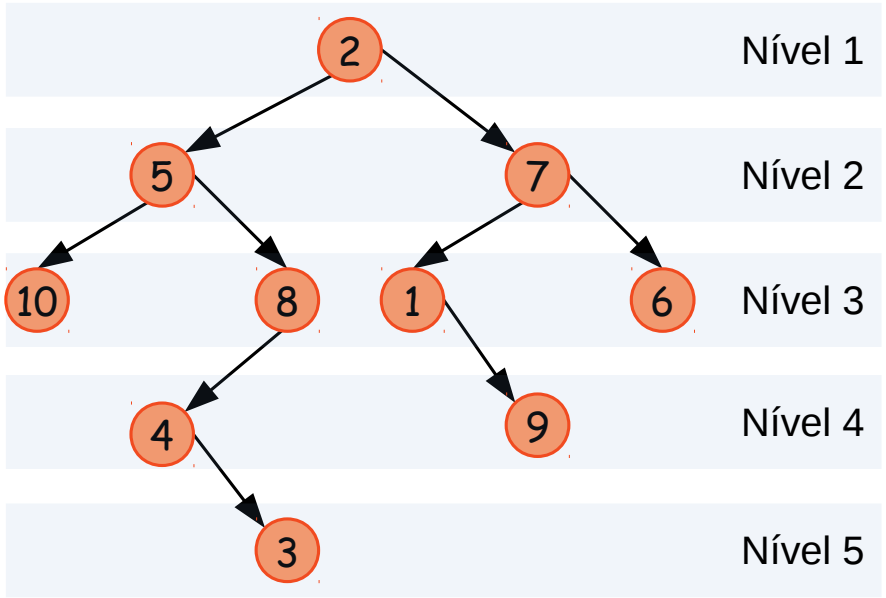
- *esquerdo:* vazio
- *direito:* 3

Filhos de 4:

- *esquerdo:* 3
- *direito:* vazio

Ordem dos filhos é relevante!

Nomenclatura e convenções



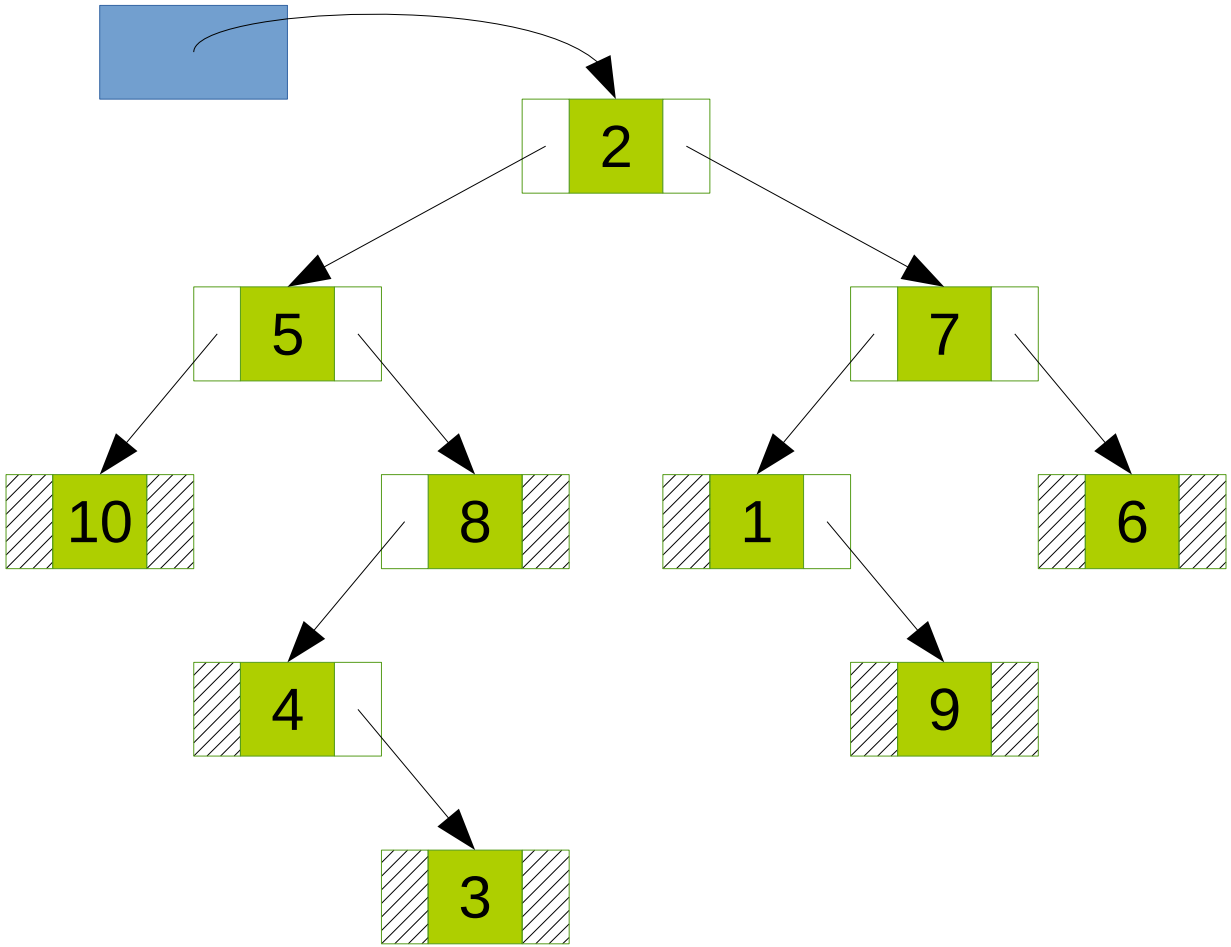
Raiz: 2
Filho esquerdo de 8: 4
Filho direito de 8: (vazio)
Folhas: 10, 6, 9, 3 (só tem filhos vazios)
Nós interno: 2, 5, 7, 8, 1, 4 (tem filhos não vazios)
Descendentes de 7: 1, 6, 9
Ancestrais de 4: 8, 5, 2
Altura: 5 (número de níveis)

Algumas propriedades

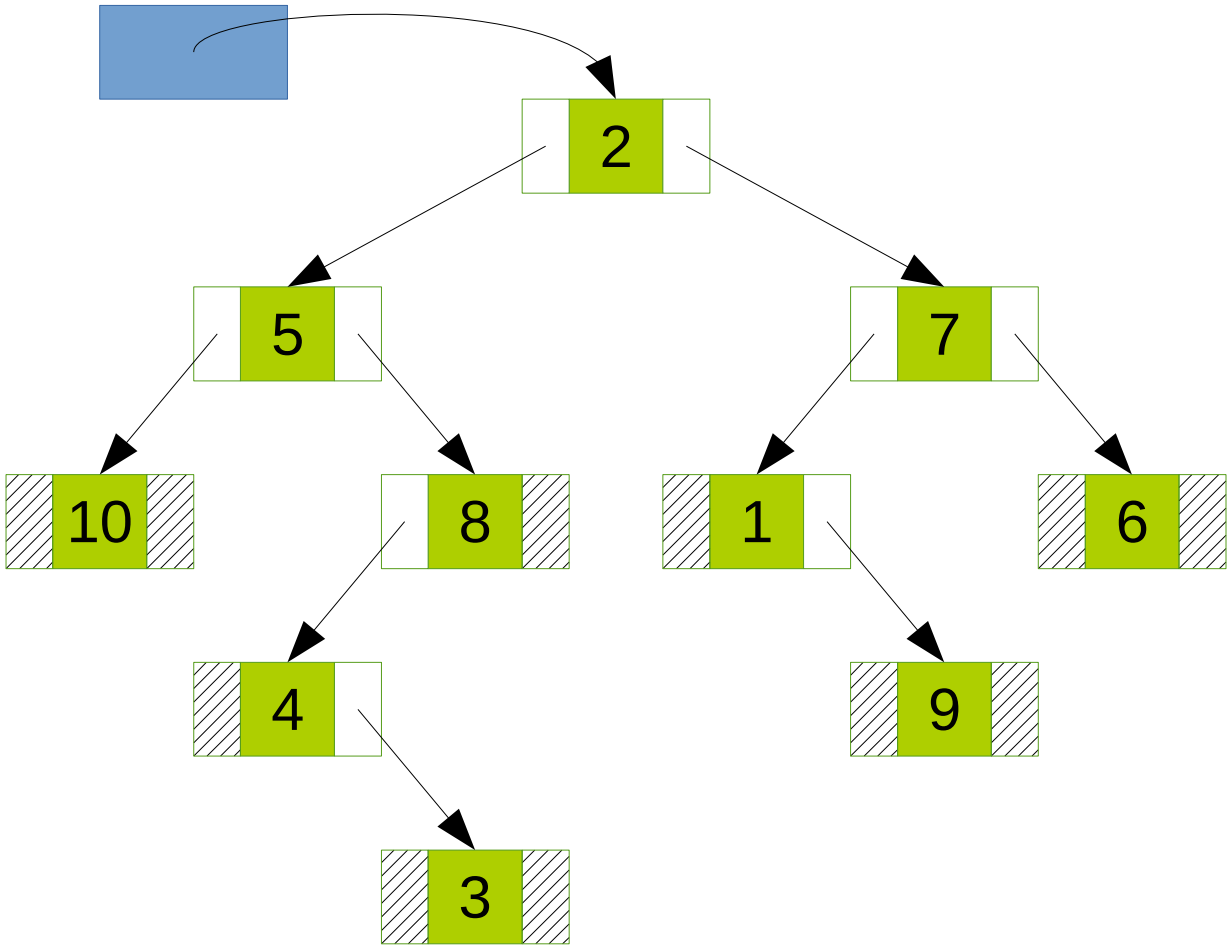
Propriedades

1. Se a altura é h, então a árvore:
   - tem no mínimo h nós
   - tem no máximo $2^h - 1$ nós
2. Se a árvore tem $n \geq 1$ nós, então:
   - a altura é no máximo n
   - a altura é no mínimo $\lceil \log_2(n+1) \rceil$
   - existem exatamente $n+1$ subárvores vazias
   - existem exatamente $n-1$ árvores não vazias

Implementação

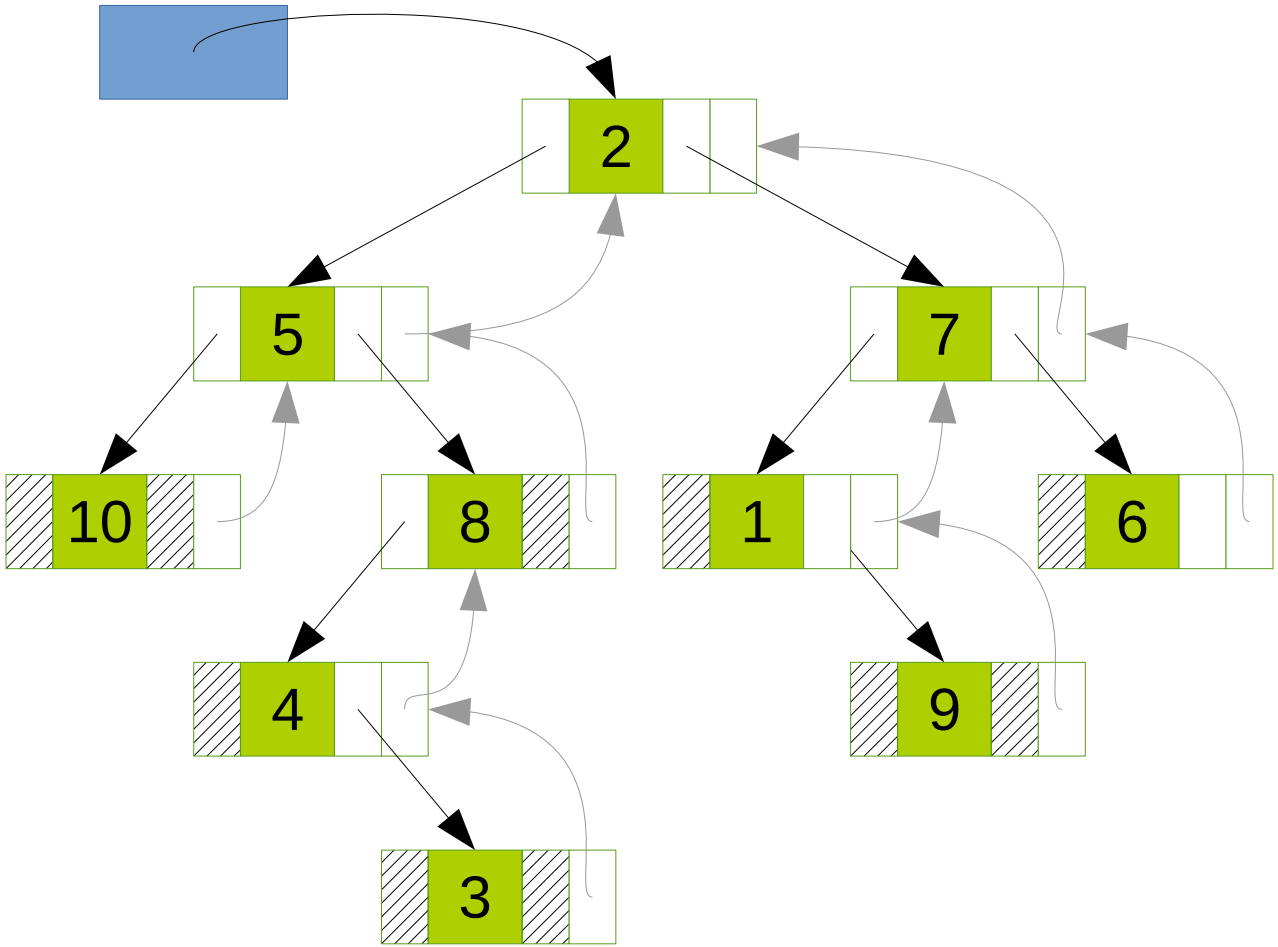


Implementação



E se quisermos saber o pai de um nó?

Implementação com ponteiro para o pai

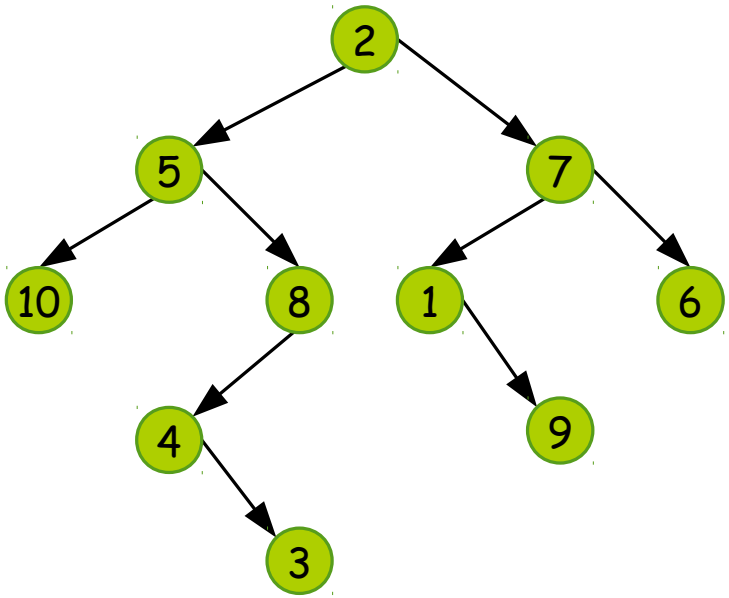


Uma implementação em C

arvbin.h

```
typedef struct NoArv {
    int dado;
    struct NoArv *esq, *dir;
} *NoArv;

NoArv *arv_vazia();
NoArv *criar_arv(NoArv *esq, NoArv *dir, int x);
NoArv *procurar_no(NoArv *arv, int x);
void imprimir(NoArv *arv);
```

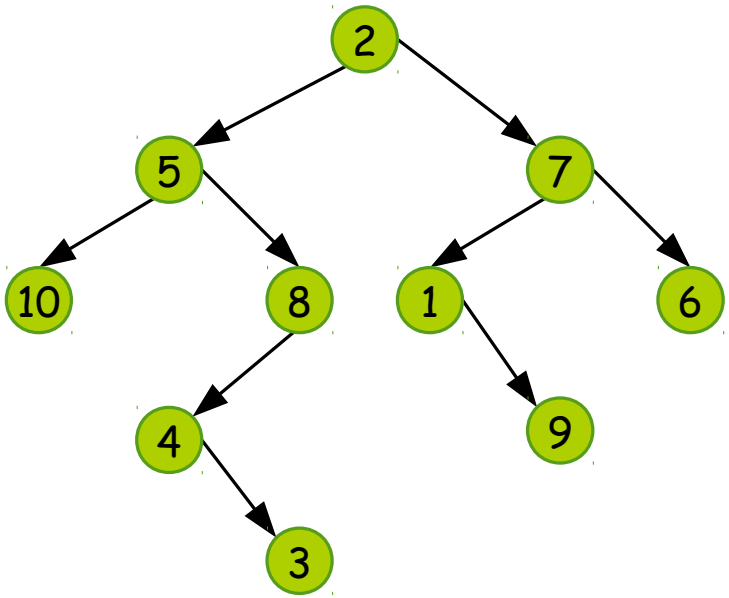
Percorrendo os nós



Percursos

1. **Em profundidade:** percorre subárvores recursivamente

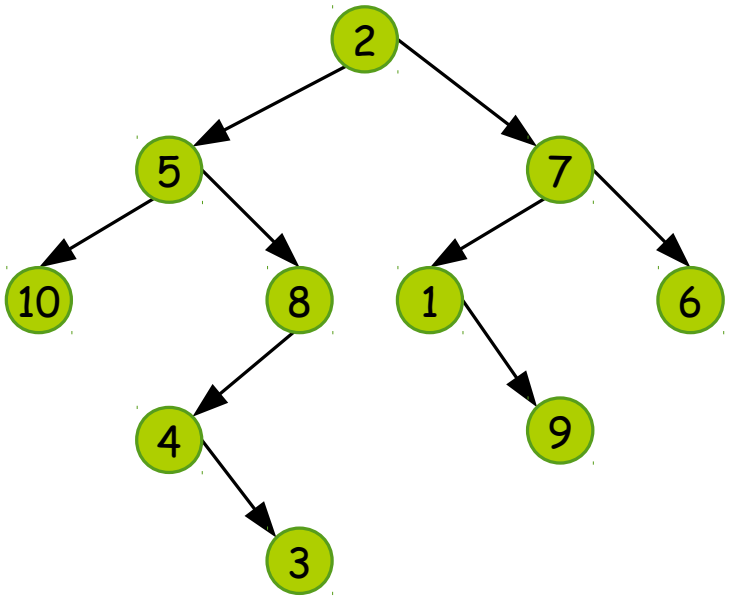
Percorrendo os nós



Percursos

1. **Em profundidade:** percorre subárvores recursivamente
   - Pré-ordem (visita primeiro a raiz):

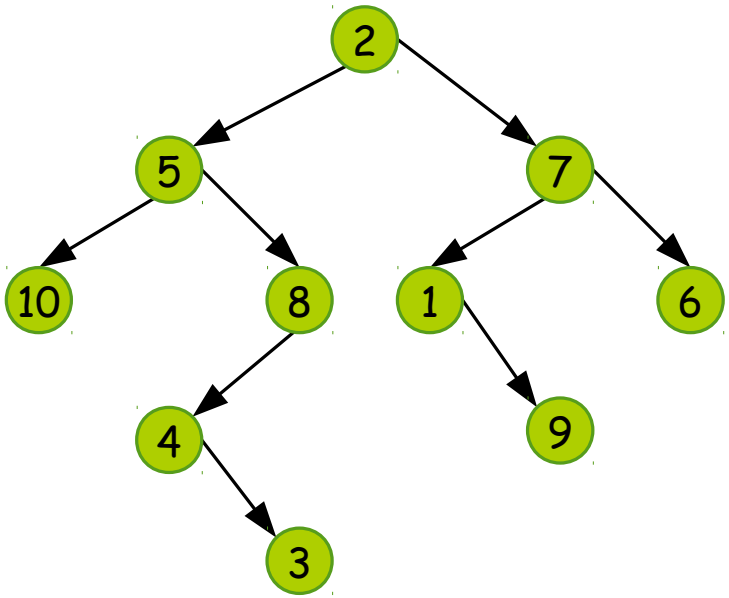
Percorrendo os nós



Percursos

1. **Em profundidade:** percorre subárvores recursivamente
   - Pré-ordem (visita primeiro a raiz): 2, 5, 10, 8, 4, 3, 7, 1, 9, 6

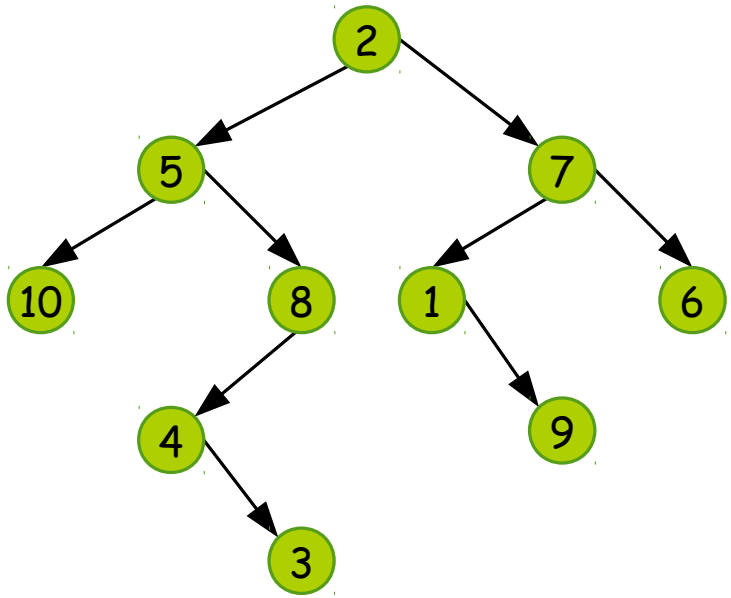
Percorrendo os nós



Percursos

1. **Em profundidade:** percorre subárvores recursivamente
   - Pré-ordem (visita primeiro a raiz): 2, 5, 10, 8, 4, 3, 7, 1, 9, 6
   - Pós-ordem (visita a raiz por último):

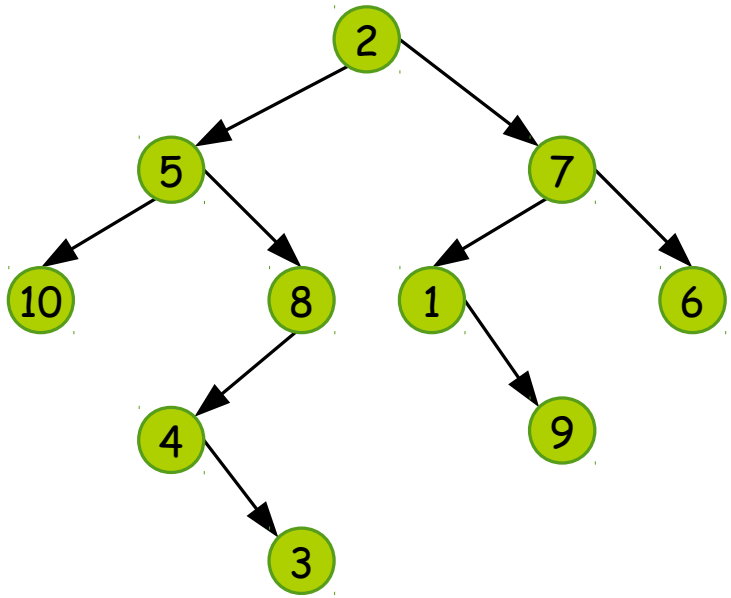
Percorrendo os nós



Percursos

1. **Em profundidade:** percorre subárvores recursivamente
   - Pré-ordem (visita primeiro a raiz): 2, 5, 10, 8, 4, 3, 7, 1, 9, 6
   - Pós-ordem (visita a raiz por último): 10, 3, 4, 8, 5, 9, 1, 6, 7, 2

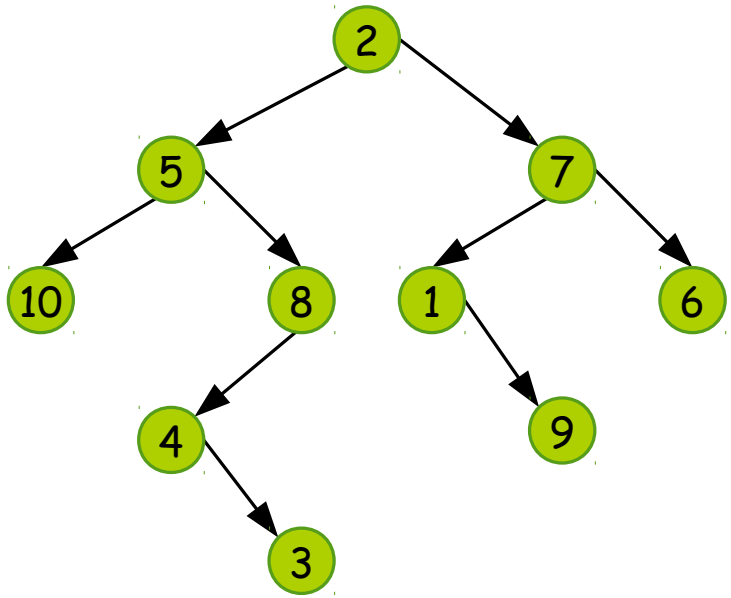
Percorrendo os nós



Percursos

1. **Em profundidade:** percorre subárvores recursivamente
   - Pré-ordem (visita primeiro a raiz): 2, 5, 10, 8, 4, 3, 7, 1, 9, 6
   - Pós-ordem (visita a raiz por último): 10, 3, 4, 8, 5, 9, 1, 6, 7, 2
   - Inordem (visita a raiz no meio):

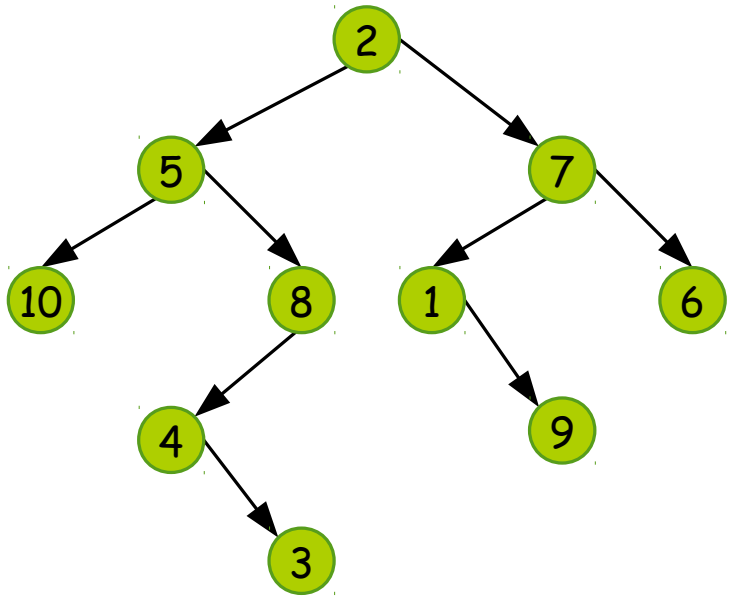
Percorrendo os nós



Percursos

1. **Em profundidade:** percorre subárvores recursivamente
   - Pré-ordem (visita primeiro a raiz): 2, 5, 10, 8, 4, 3, 7, 1, 9, 6
   - Pós-ordem (visita a raiz por último): 10, 3, 4, 8, 5, 9, 1, 6, 7, 2
   - Inordem (visita a raiz no meio): 10, 5, 4, 3, 8, 2, 1, 9, 7, 6

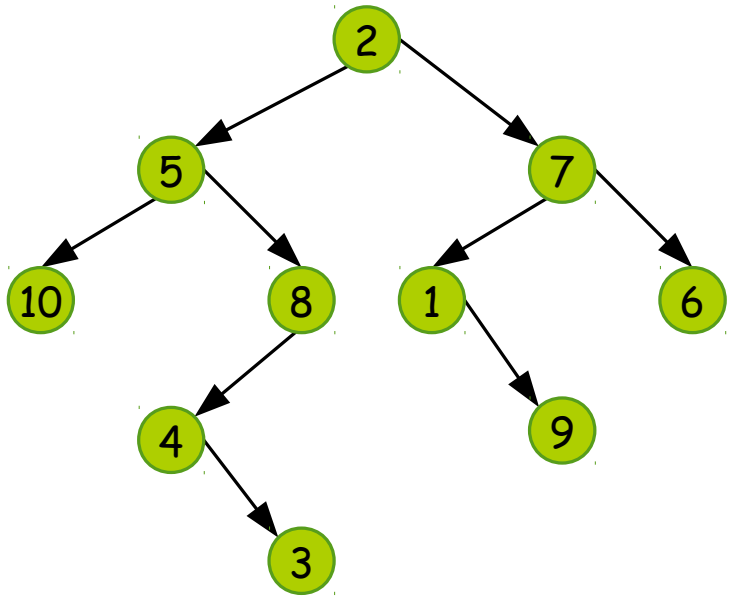
Percorrendo os nós



Percursos

1. **Em profundidade:** percorre subárvores recursivamente
   - Pré-ordem (visita primeiro a raiz): 2, 5, 10, 8, 4, 3, 7, 1, 9, 6
   - Pós-ordem (visita a raiz por último): 10, 3, 4, 8, 5, 9, 1, 6, 7, 2
   - Inordem (visita a raiz no meio): 10, 5, 4, 3, 8, 2, 1, 9, 7, 6
2. **Em largura:** percorre níveis em ordem
   - da esquerda para direita:

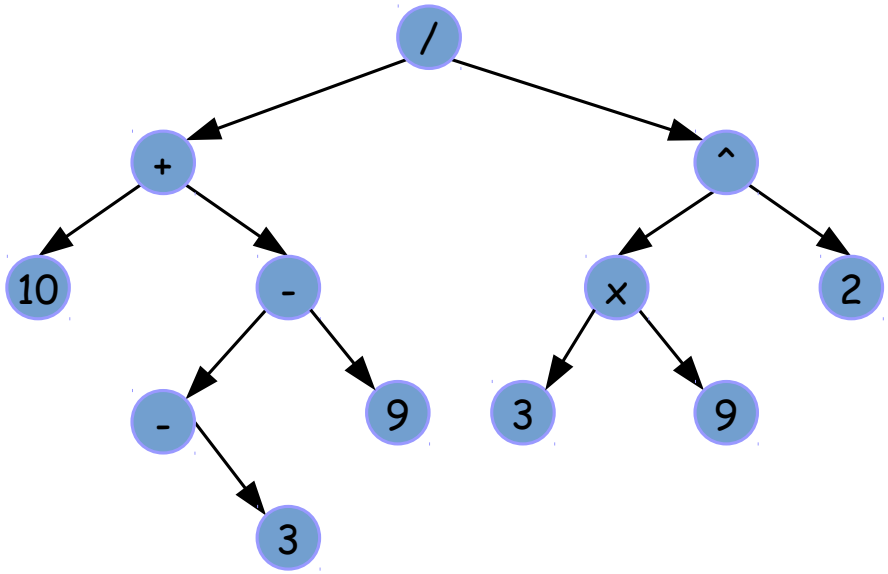
Percorrendo os nós



Percursos

1. **Em profundidade:** percorre subárvores recursivamente
   - Pré-ordem (visita primeiro a raiz): 2, 5, 10, 8, 4, 3, 7, 1, 9, 6
   - Pós-ordem (visita a raiz por último): 10, 3, 4, 8, 5, 9, 1, 6, 7, 2
   - Inordem (visita a raiz no meio): 10, 5, 4, 3, 8, 2, 1, 9, 7, 6
2. **Em largura:** percorre níveis em ordem
   - da esquerda para direita: 2, 5, 7, 10, 8, 1, 6, 4, 9, 3

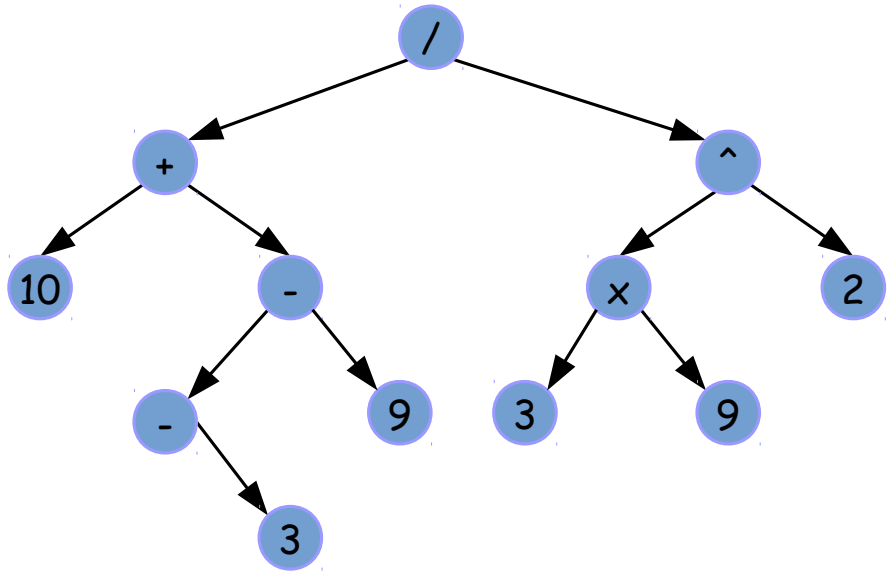
Busca em profundidade e expressões



Ordem

- Pré-fixa: / + 10 - - 3 9 ^ × 3 9 2
- Pós-fixa: 10 3 - 9 - + 3 9 × 2 ^ /
- Infixa: 10 + - 3 - 9 / 3 × 9 ^ 2

Busca em profundidade e expressões



Ordem

- Pré-fixa: / + 10 - - 3 9 ^ × 3 9 2
- Pós-fixa: 10 3 - 9 - + 3 9 × 2 ^ /
- Infixa: 10 + - 3 - 9 / 3 × 9 ^ 2

Existem ambiguidades?

Implementação de busca em profundidade

```
void pre_ordem(NoArv *arv) {
    if (arv) {
        printf("%d ", arv->dado); // visita raiz
        pre_ordem(arv->esq);
        pre_ordem(arv->dir);
    }
}
void pos_ordem(NoArv *arv) {
    if (arv) {
        pos_ordem(arv->esq);
        pos_ordem(arv->dir);
        printf("%d ", arv->dado); // visita raiz
    }
}
void inordem(NoArv *arv) {
    if (arv) {
        inordem(arv->esq);
        printf("%d ", arv->dado); // visita raiz
        inordem(arv->dir);
    }
}
```

Percurso em profundidade com pilha

Como implementar sem usar recursão?

Percurso em profundidade com pilha

Como implementar sem usar recursão?

Pré-ordem

```
void pre_ordem(NoArv *arv) {
    NoPilha *p;
    iniciar_pilha(&p);

    empilhar(&p, arv);
    while(!pilha_vazia(p)) {
        desempilhar(&p, &arv);
        if (arv) {
            empilhar(&p, arv->dir);
            empilhar(&p, arv->esq);
            visita(arv);
        }
    }
    destruir_pilha(&p);
}
```

Percurso em profundidade com pilha

Como implementar sem usar recursão?

Pré-ordem

```
void pre_ordem(NoArv *arv) {
    NoPilha *p;
    iniciar_pilha(&p);

    empilhar(&p, arv);
    while(!pilha_vazia(p)) {
        desempilhar(&p, &arv);
        if (arv) {
            empilhar(&p, arv->dir);
            empilhar(&p, arv->esq);
            visita(arv);
        }
    }
    destruir_pilha(&p);
}
```

Por que empilhamos `arv->dir` primeiro?

Percurso em profundidade com pilha

Como implementar sem usar recursão?

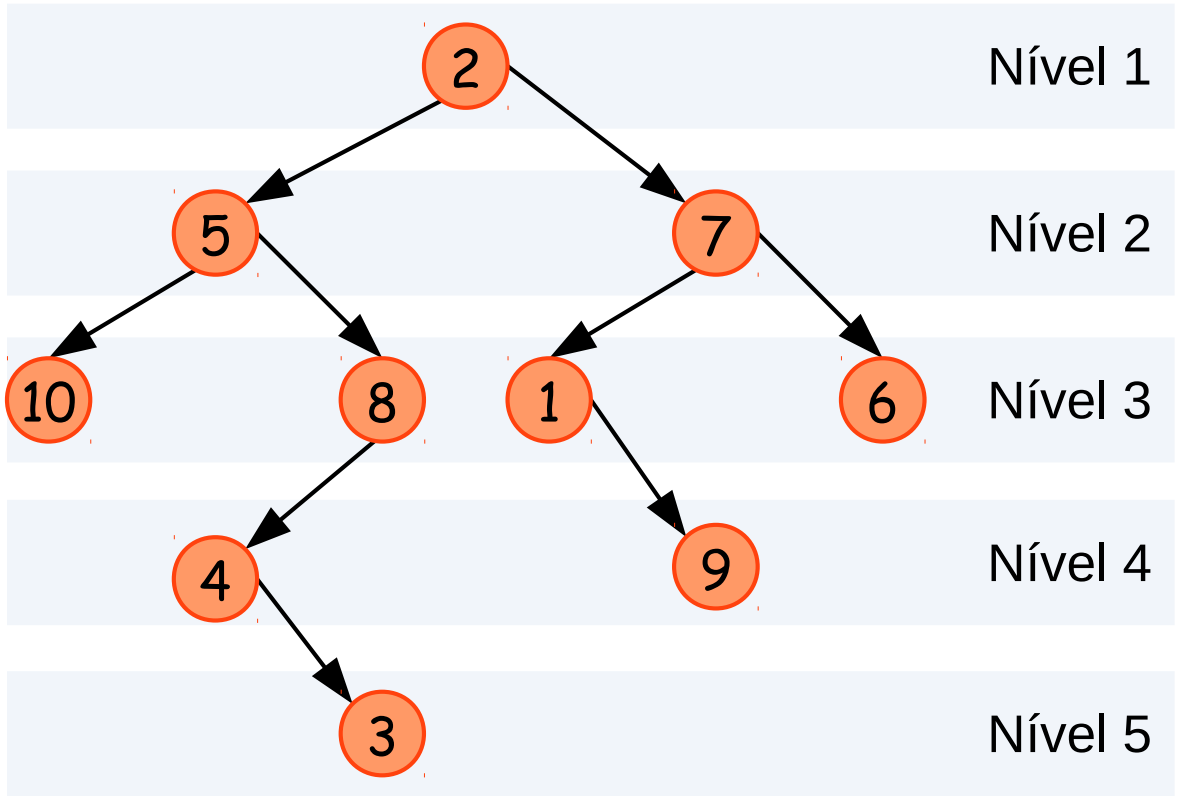
Pré-ordem

```
void pre_ordem(NoArv *arv) {
    NoPilha *p;
    iniciar_pilha(&p);

    empilhar(&p, arv);
    while(!pilha_vazia(p)) {
        desempilhar(&p, &arv);
        if (arv) {
            empilhar(&p, arv->dir);
            empilhar(&p, arv->esq);
            visita(arv);
        }
    }
    destruir_pilha(&p);
}
```

Por que empilhamos `arv->dir` primeiro? Se fosse o contrário?

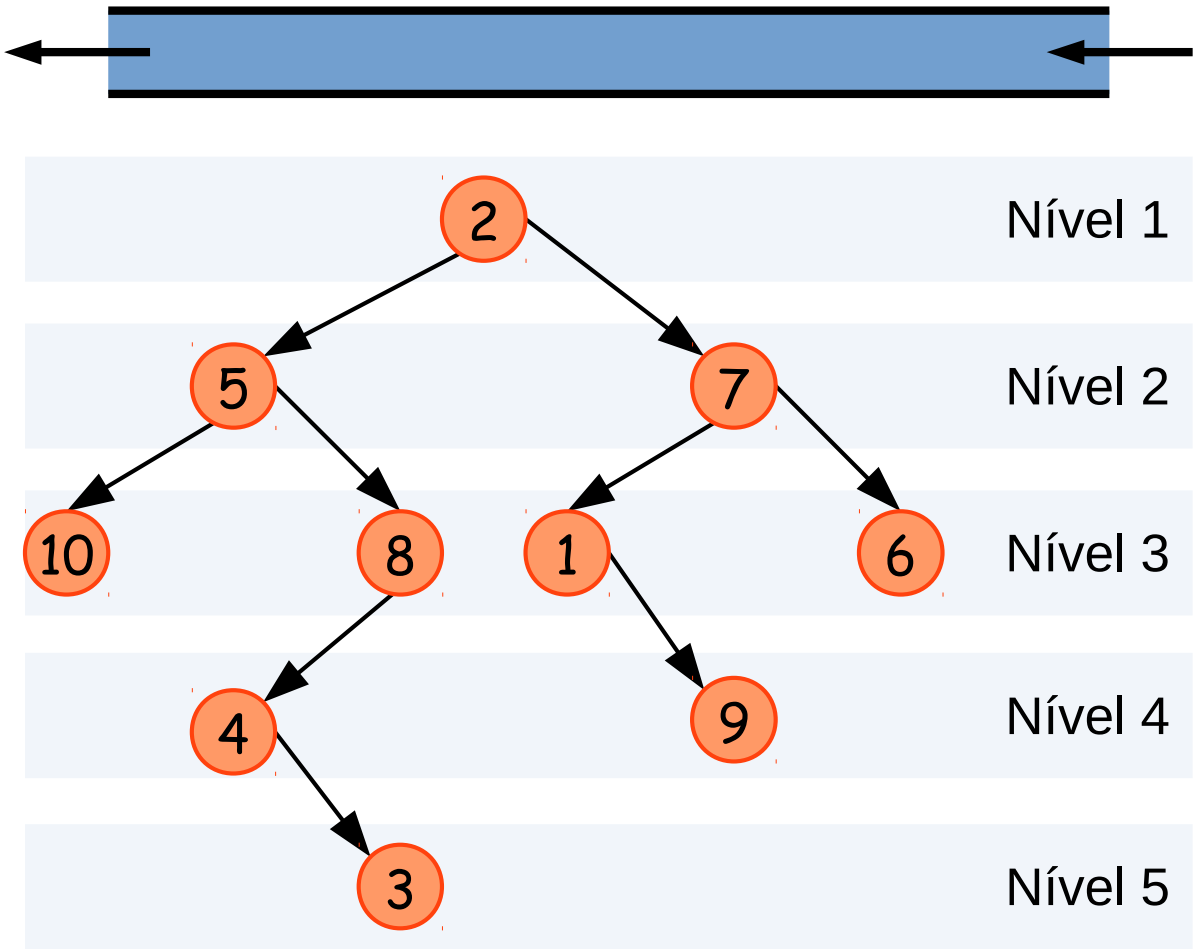
Implementação da busca em largura

Como implementar a busca em largura?



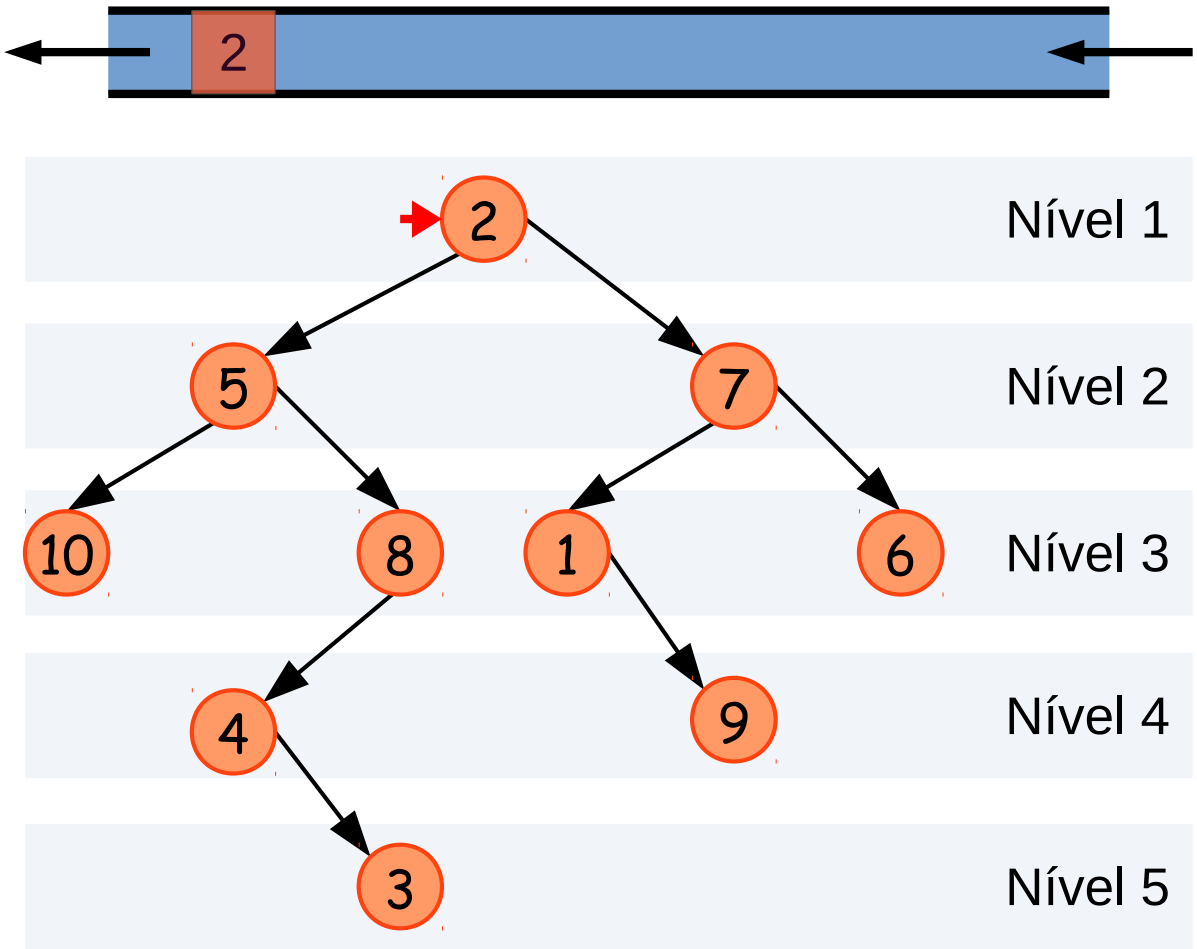
Implementação da busca em largura

Como implementar a busca em largura? Usamos uma fila!



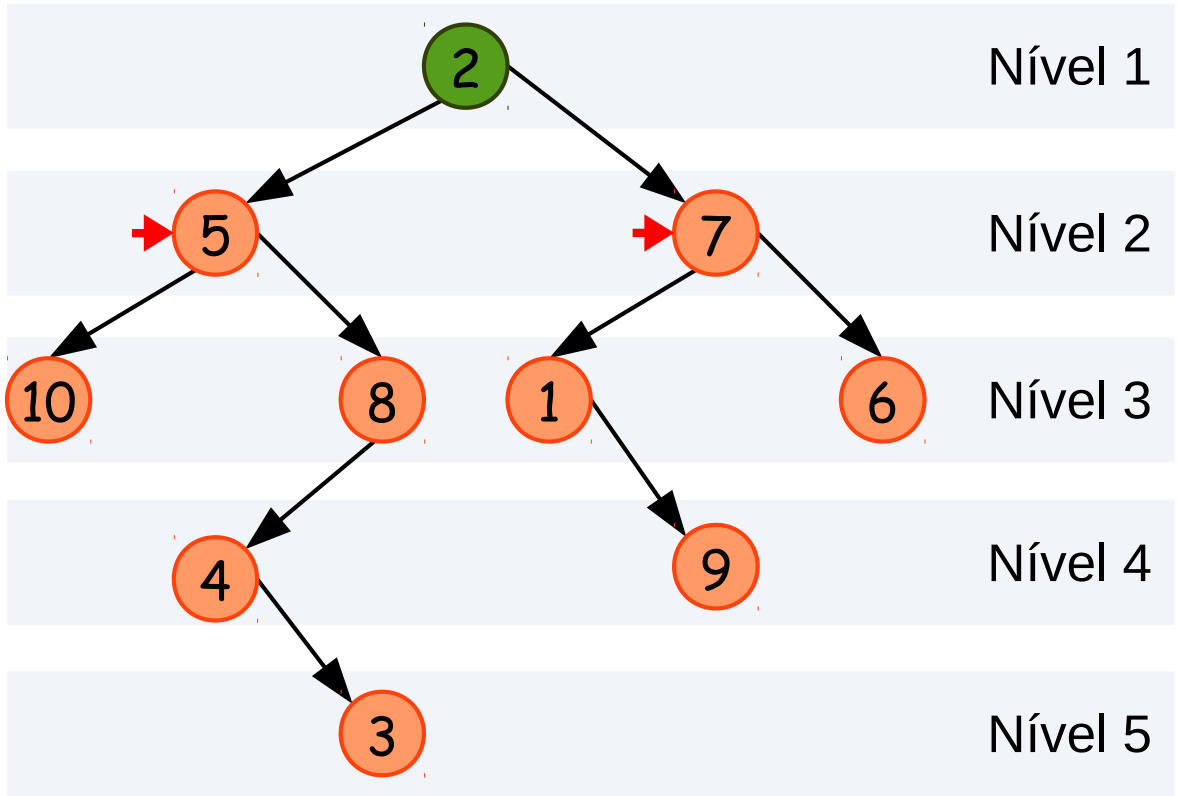
Implementação da busca em largura

Como implementar a busca em largura?



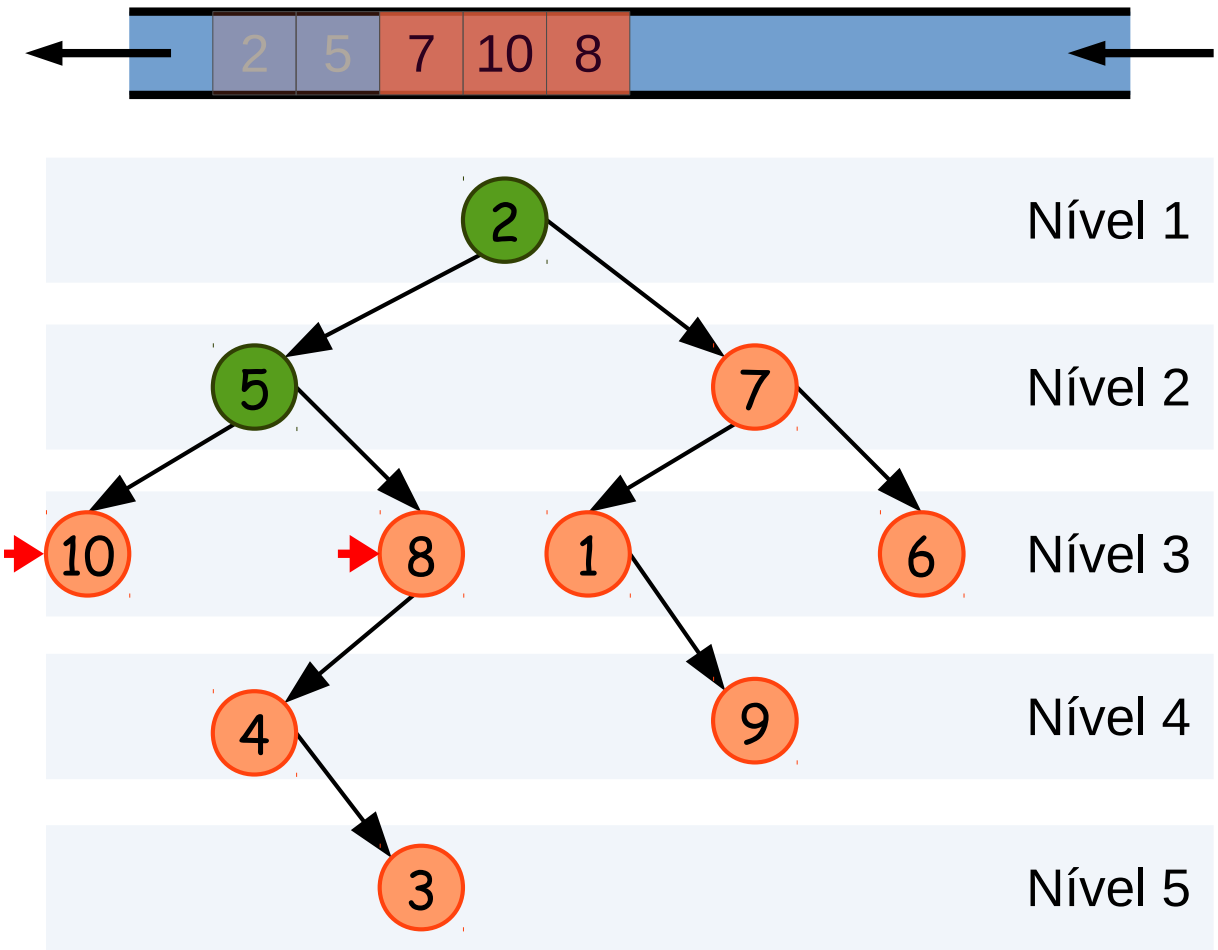
Implementação da busca em largura

Como implementar a busca em largura?



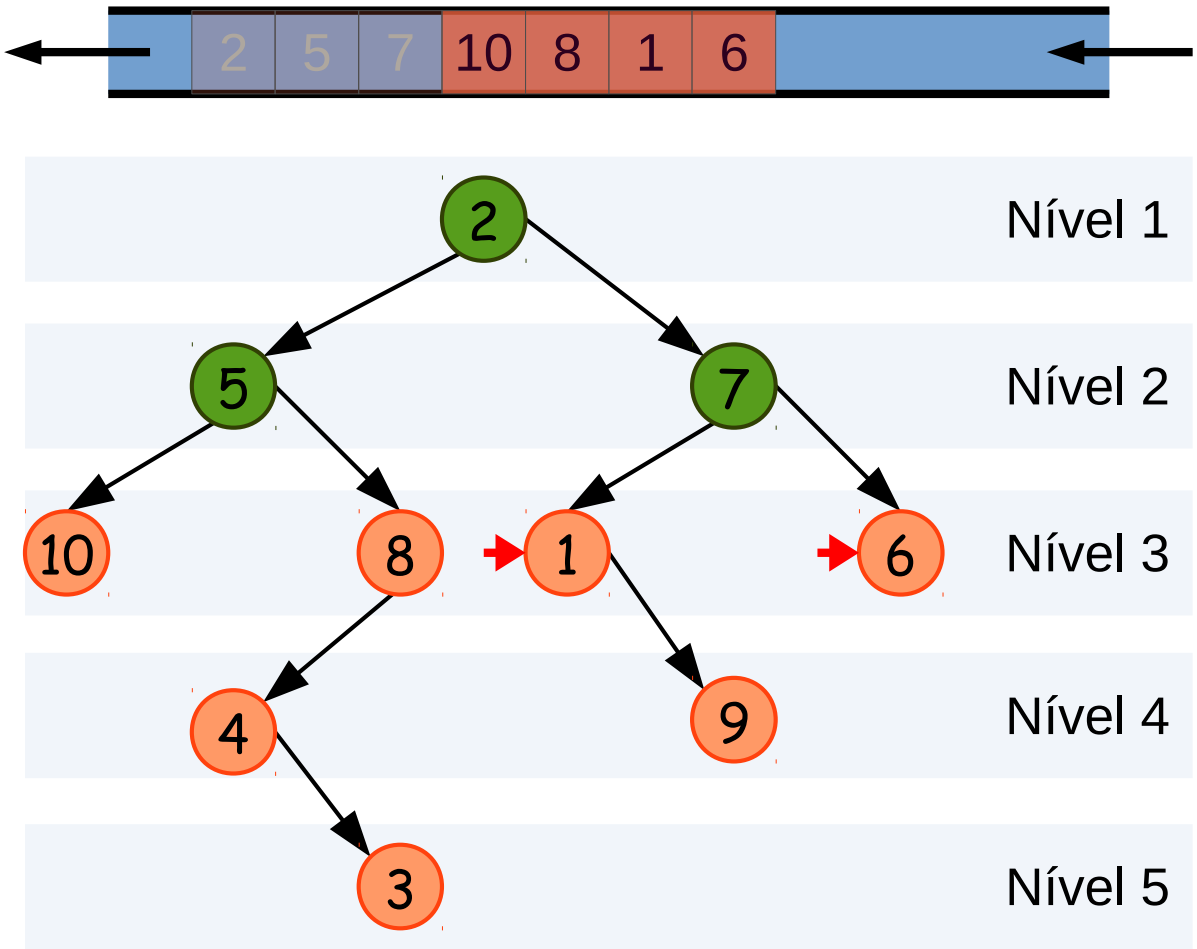
Implementação da busca em largura

Como implementar a busca em largura?



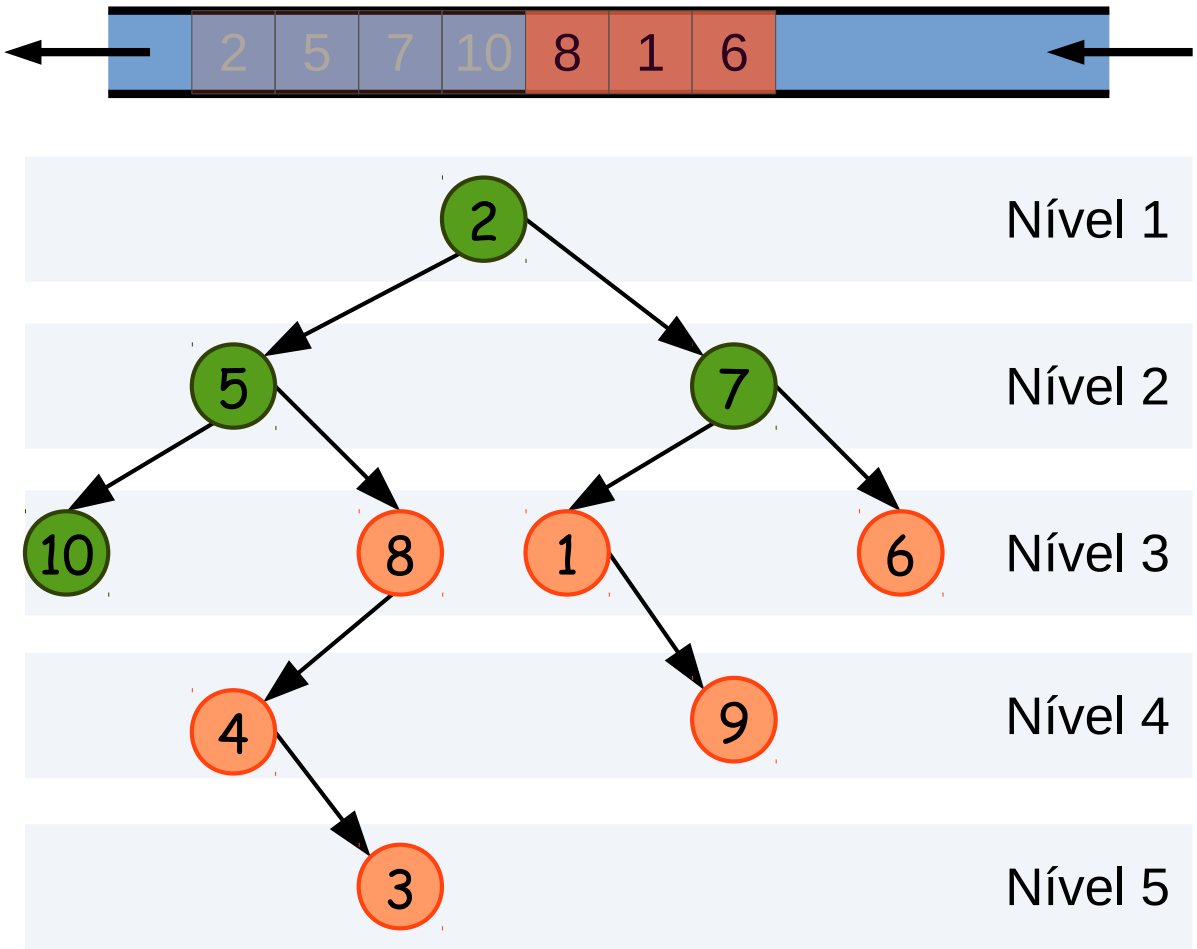
Implementação da busca em largura

Como implementar a busca em largura?



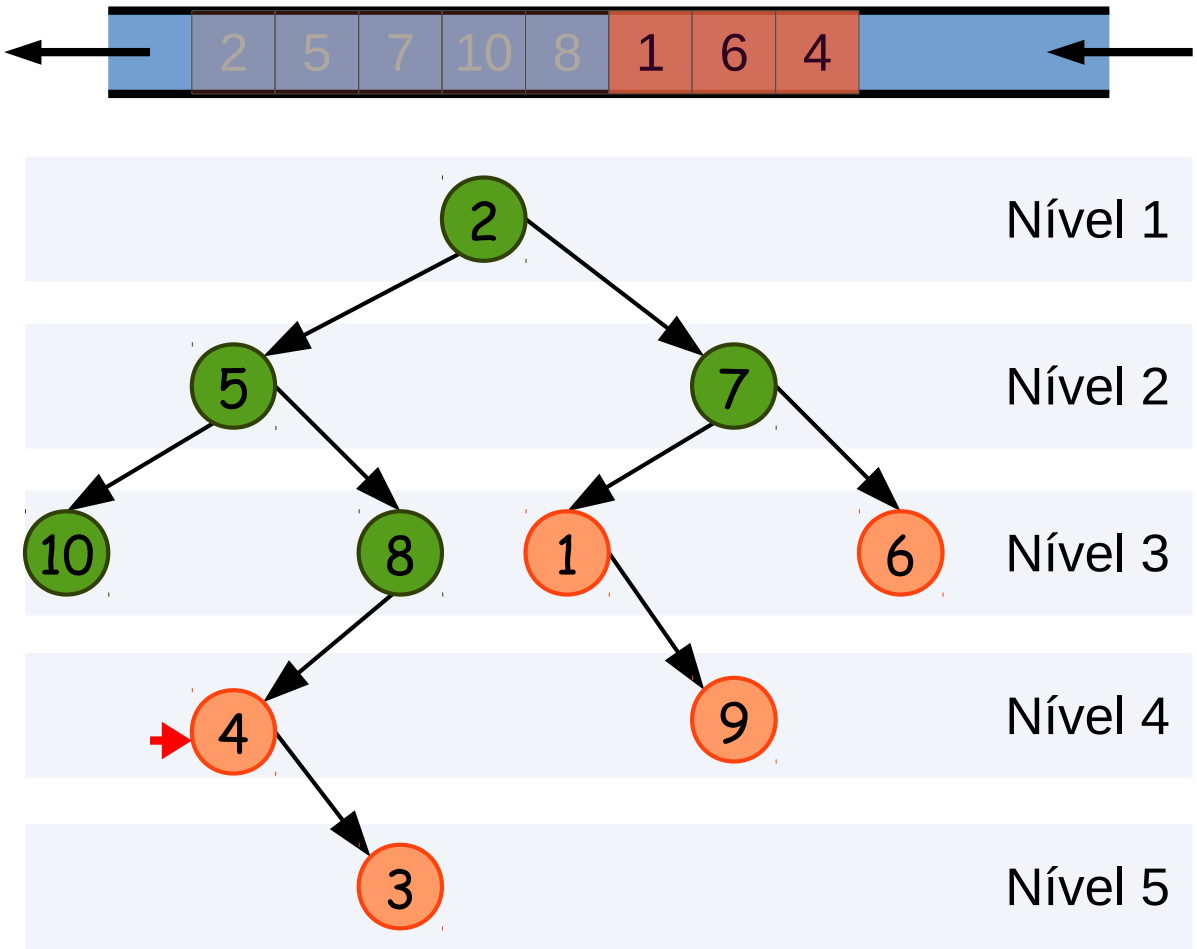
Implementação da busca em largura

Como implementar a busca em largura?



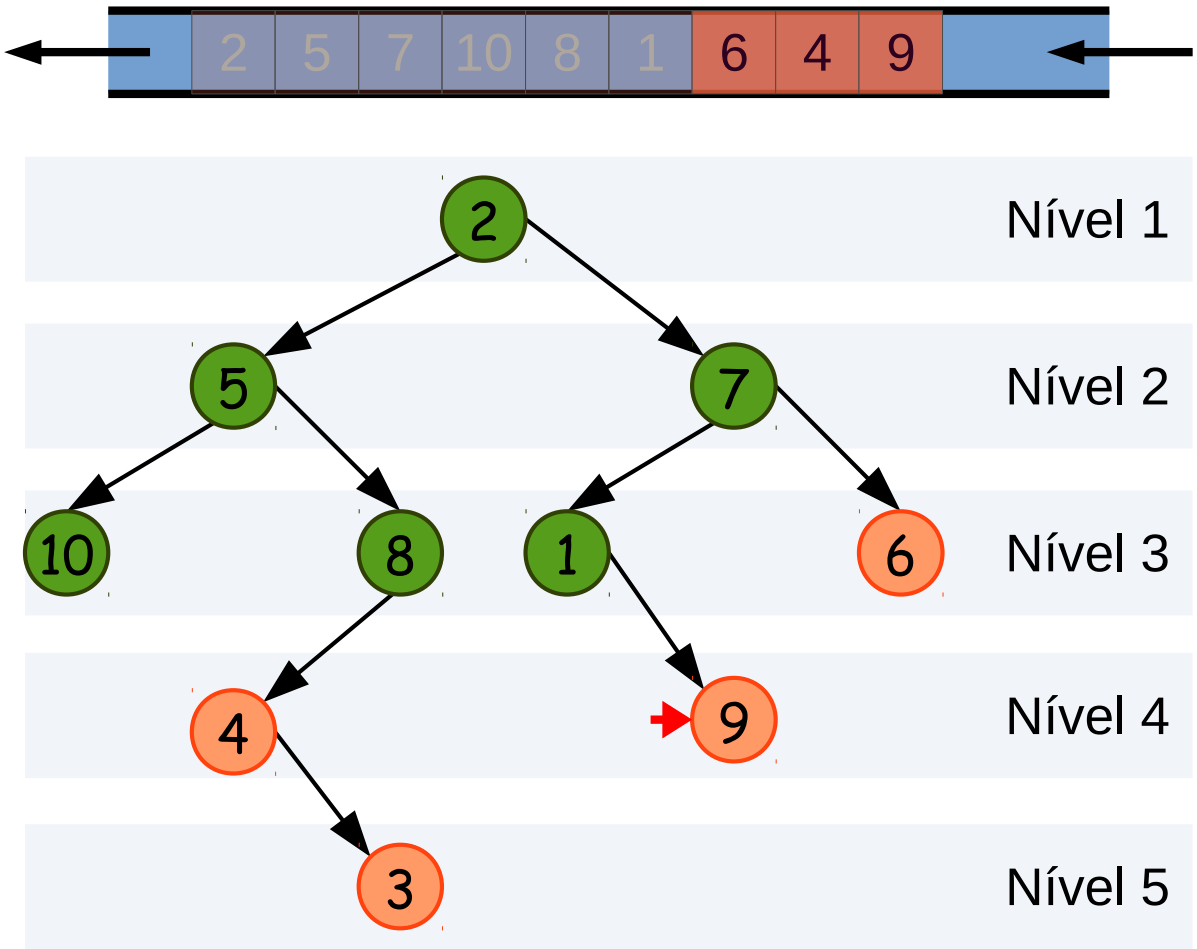
Implementação da busca em largura

Como implementar a busca em largura?



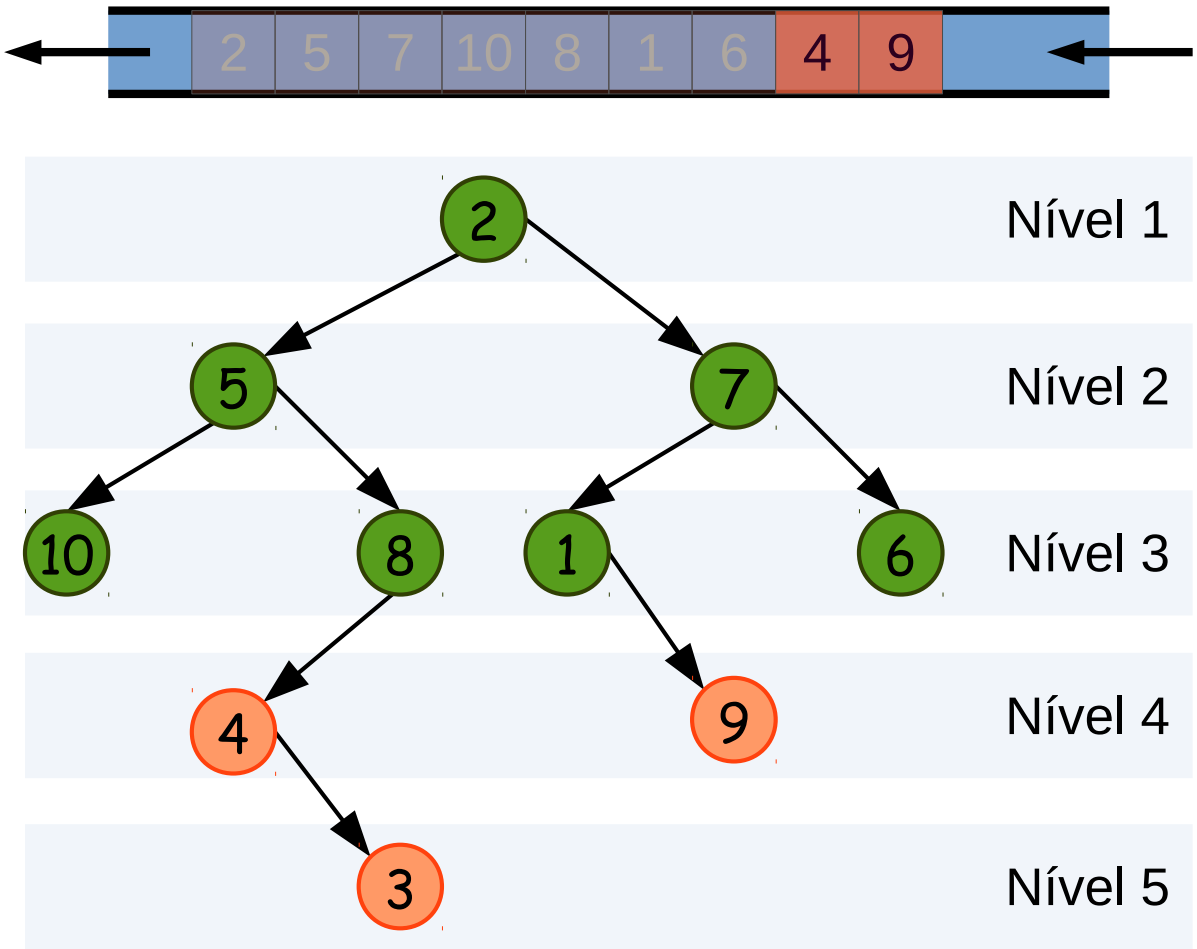
Implementação da busca em largura

Como implementar a busca em largura?



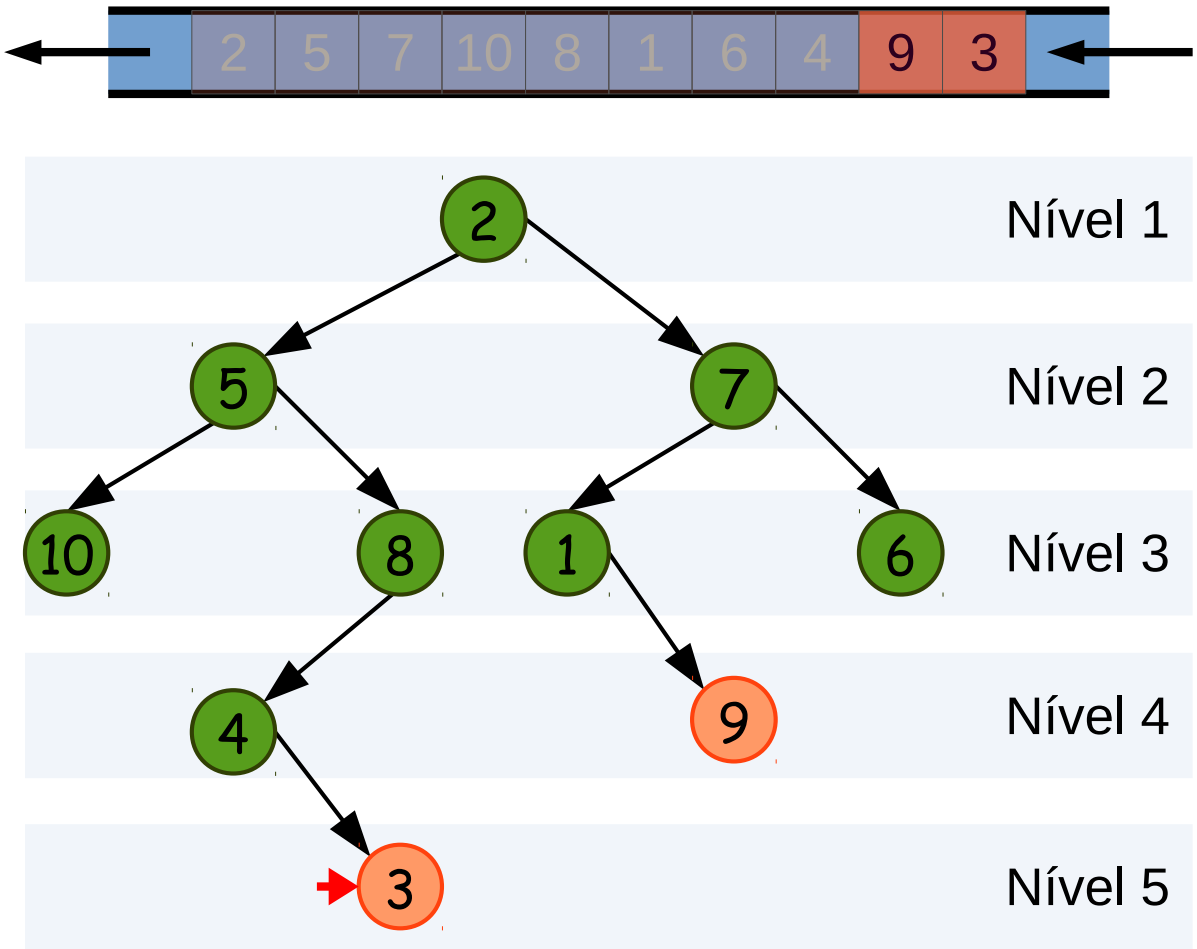
Implementação da busca em largura

Como implementar a busca em largura?



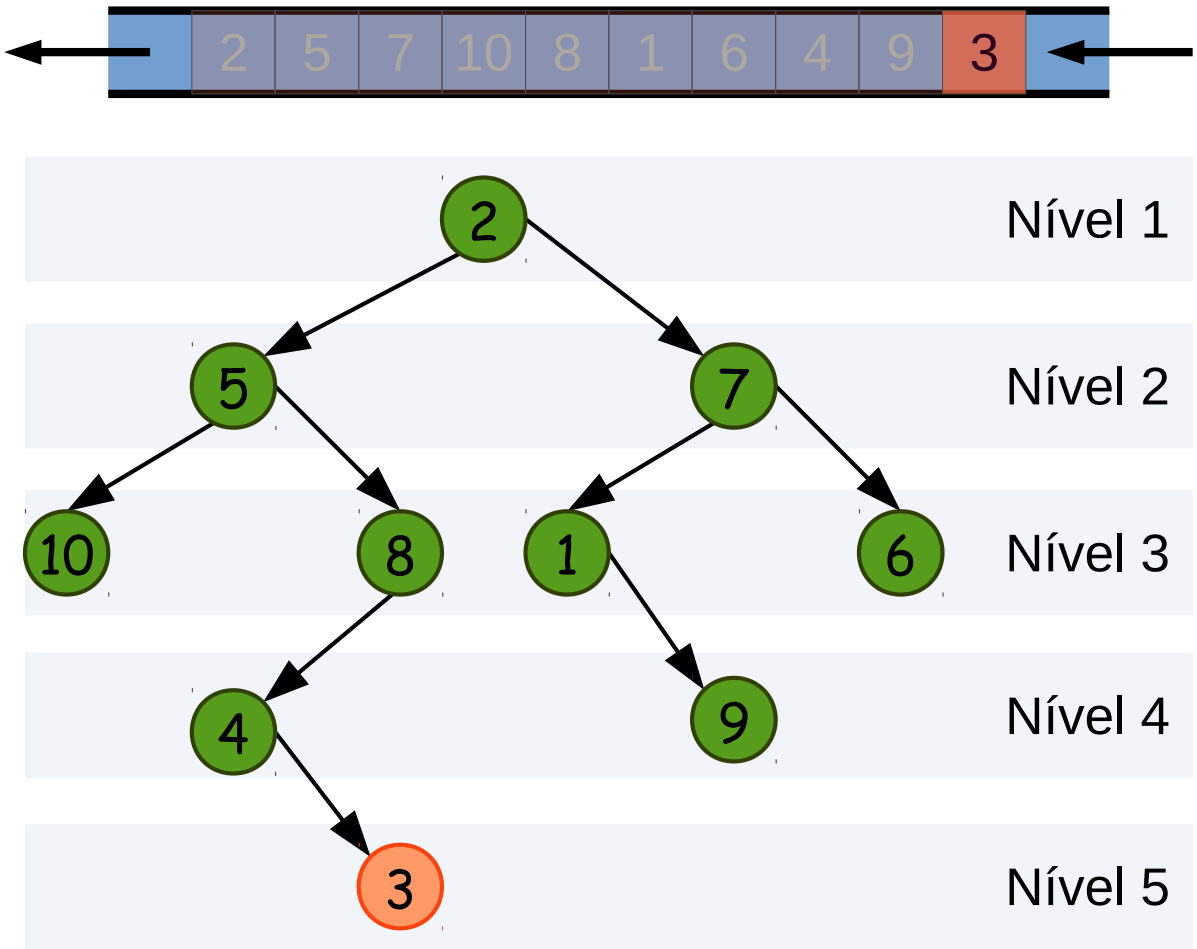
Implementação da busca em largura

Como implementar a busca em largura?



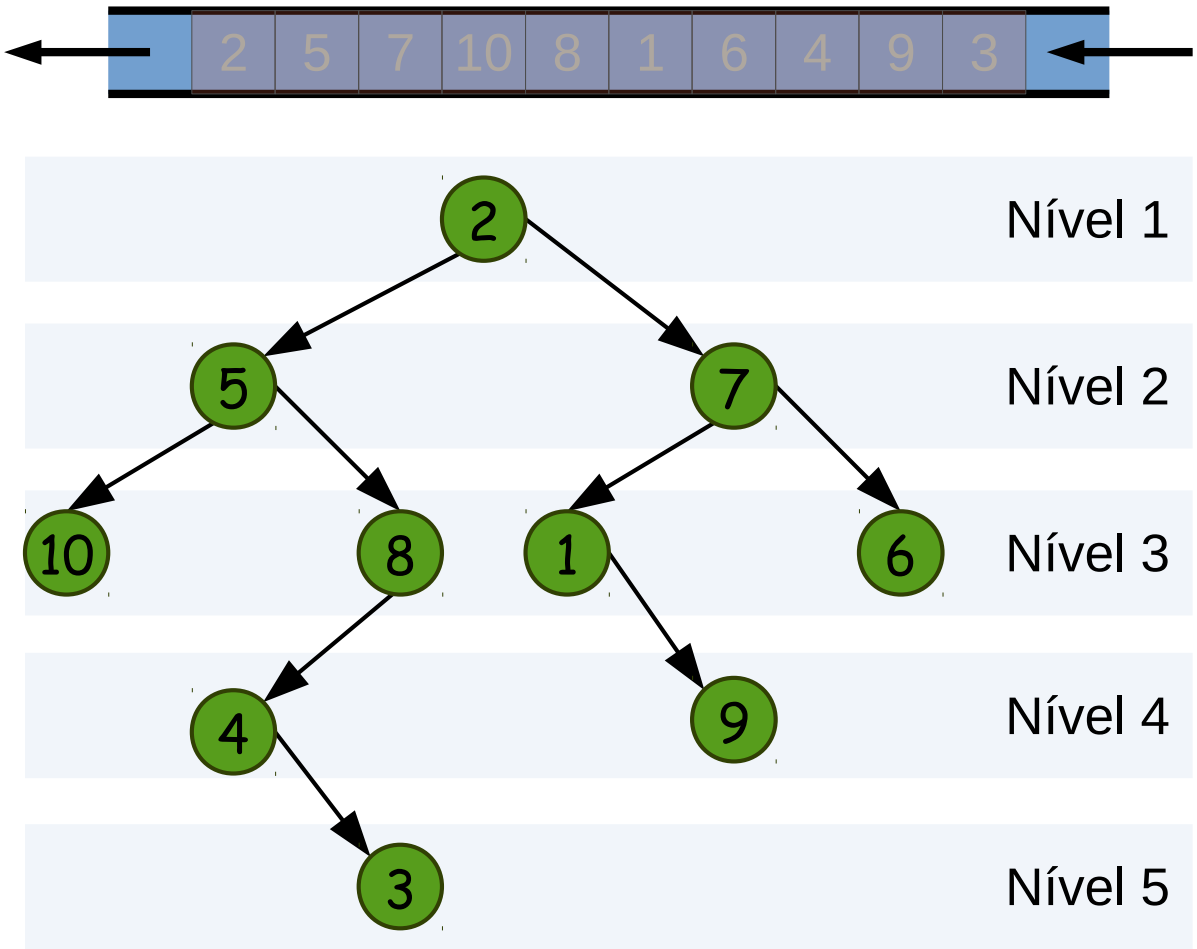
Implementação da busca em largura

Como implementar a busca em largura?



Implementação da busca em largura

Como implementar a busca em largura?



Percurso em largura

Percurso em largura

```
void largura(NoArv *arv) {
    NoFila *p;
    iniciar_fila(&p);

    enfileirar(&p, arv);
    while(!fila_vazia(p)) {
        desenfileirar(&p, &arv);
        if (arv) {
            enfileirar(&f, arv->esq);
            enfileirar(&f, arv->dir);
            visita(arv);
        }
    }
    destruir_fila(&f);
}
```

Percurso em largura

Percurso em largura

```
void largura(NoArv *arv) {
    NoFila *p;
    iniciar_fila(&p);

    enfileirar(&p, arv);
    while(!fila_vazia(p)) {
        desenfileirar(&p, &arv);
        if (arv) {
            enfileirar(&f, arv->esq);
            enfileirar(&f, arv->dir);
            visita(arv);
        }
    }
    destruir_fila(&f);
}
```

Agora enfileiramos `arv->esq` primeiro!

Percurso em largura

Percurso em largura

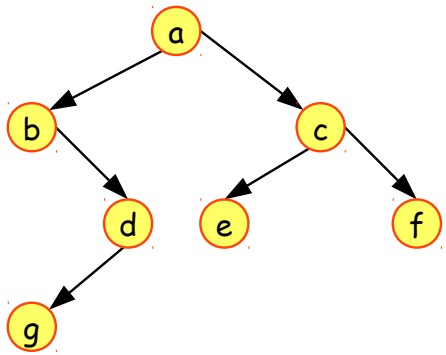
```
void largura(NoArv *arv) {
    NoFila *p;
    iniciar_fila(&p);

    enfileirar(&p, arv);
    while(!fila_vazia(p)) {
        desenfileirar(&p, &arv);
        if (arv) {
            enfileirar(&f, arv->esq);
            enfileirar(&f, arv->dir);
            visita(arv);
        }
    }
    destruir_fila(&f);
}
```

Agora enfileiramos `arv->esq` primeiro! Se fosse o contrário?

Outras representações de árvore binária

Problema

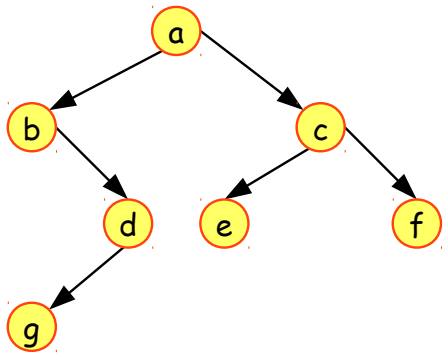
Temos uma árvore e queremos guardar a estrutura em um arquivo de texto para posterior processamento. Como armazenar essa árvore?



Outras representações de árvore binária

Problema

Temos uma árvore e queremos guardar a estrutura em um arquivo de texto para posterior processamento. Como armazenar essa árvore?



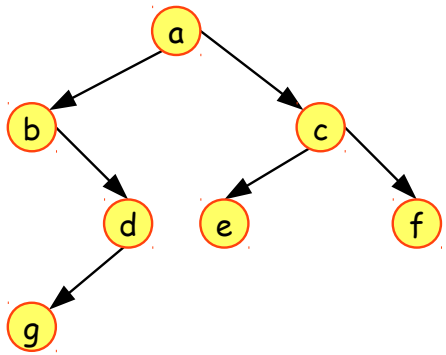
Representações externas

1. Indicadores de subárvores com percurso em profundidade:
   a 1 1, b 0 1, d 1 0, g 0 0, c 1 1, e 0 0, f 0 0

Outras representações de árvore binária

Problema

Temos uma árvore e queremos guardar a estrutura em um arquivo de texto para posterior processamento. Como armazenar essa árvore?

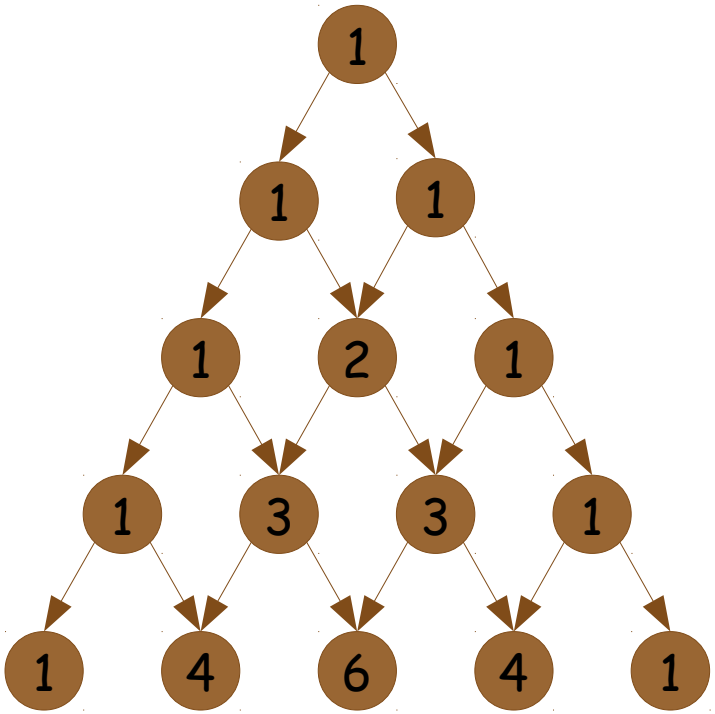


Representações externas

1. Indicadores de subárvores com percurso em profundidade:
   a 1 1, b 0 1, d 1 0, g 0 0, c 1 1, e 0 0, f 0 0
2. Notação com parênteses:
   - pré-ordem: (a(b()(d(g()())()))(c(e()())(f()())))
   - inordem: (()b(((()g())d()))a(((()e())c(()f())))

Exercício 1

Joãozinho quer implementar uma estrutura de dados para representar o triângulo de Pascal até certa altura. Após assistir à aula de árvores binárias, ele decidiu que essa era a estrutura ideal e fez o seguinte desenho:



1. A estrutura que Joãozinho criou é uma árvore binária? Por quê? Se não, sugira uma estrutura mais adequada.
2. Qual o resultado será impresso se as funções para busca em profundidade (pré-ordem, inordem e pós-ordem) e em largura forem chamadas com essa estrutura (tente em uma altura menor)?

Exercício 2 - Recuperando uma árvore

1. Escreva uma função para recuperar uma árvore na memória a partir de uma string com a notação em parênteses em pré-ordem.
2. Você obteve um arquivo em que foram impressos os nós de uma árvore binária no percurso pré-ordem e gostaria de obter a árvore original. Infelizmente não é possível reconstruir essa árvore unicamente. Dê um exemplo que justifique essa afirmação.
3. E se você também tivesse um arquivo com os dados impressos com percurso inordem, você conseguiria obter a árvore original? Que combinações de percursos você precisa?