Melhores momentos

AULA 20

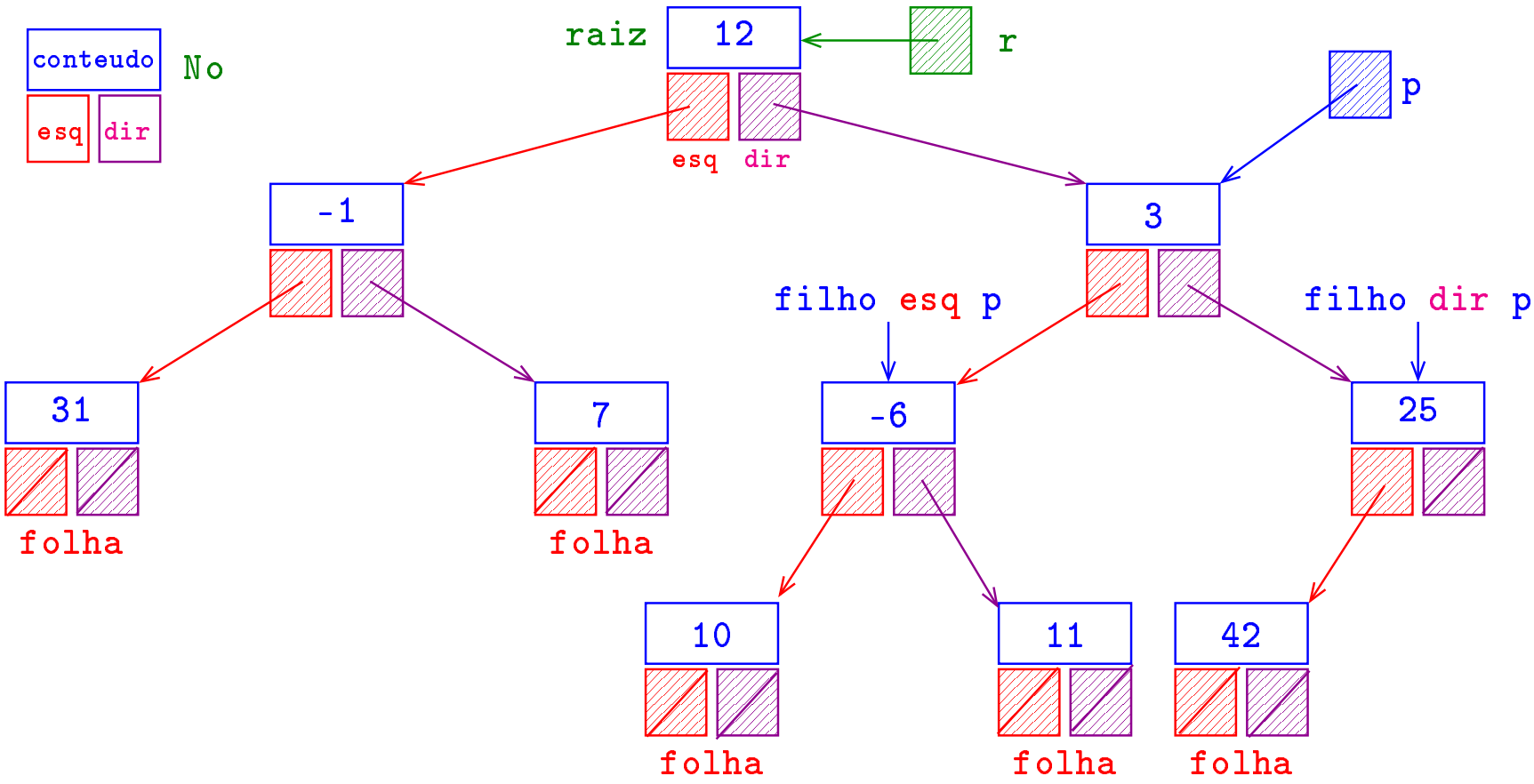
Árvore binárias

Uma **árvore binária** (= *binary tree*) é um conjunto de nós/células que satisfaz certas condições.

Cada nó tem três campos:

```
typedef struct celula Celula;
struct celula {
   int conteudo; /* tipo devia ser Item*/
   Celula *esq;
   Celula *dir;
};
typedef Celula No;
No x, y;
```

Ilustração de uma árvore binária

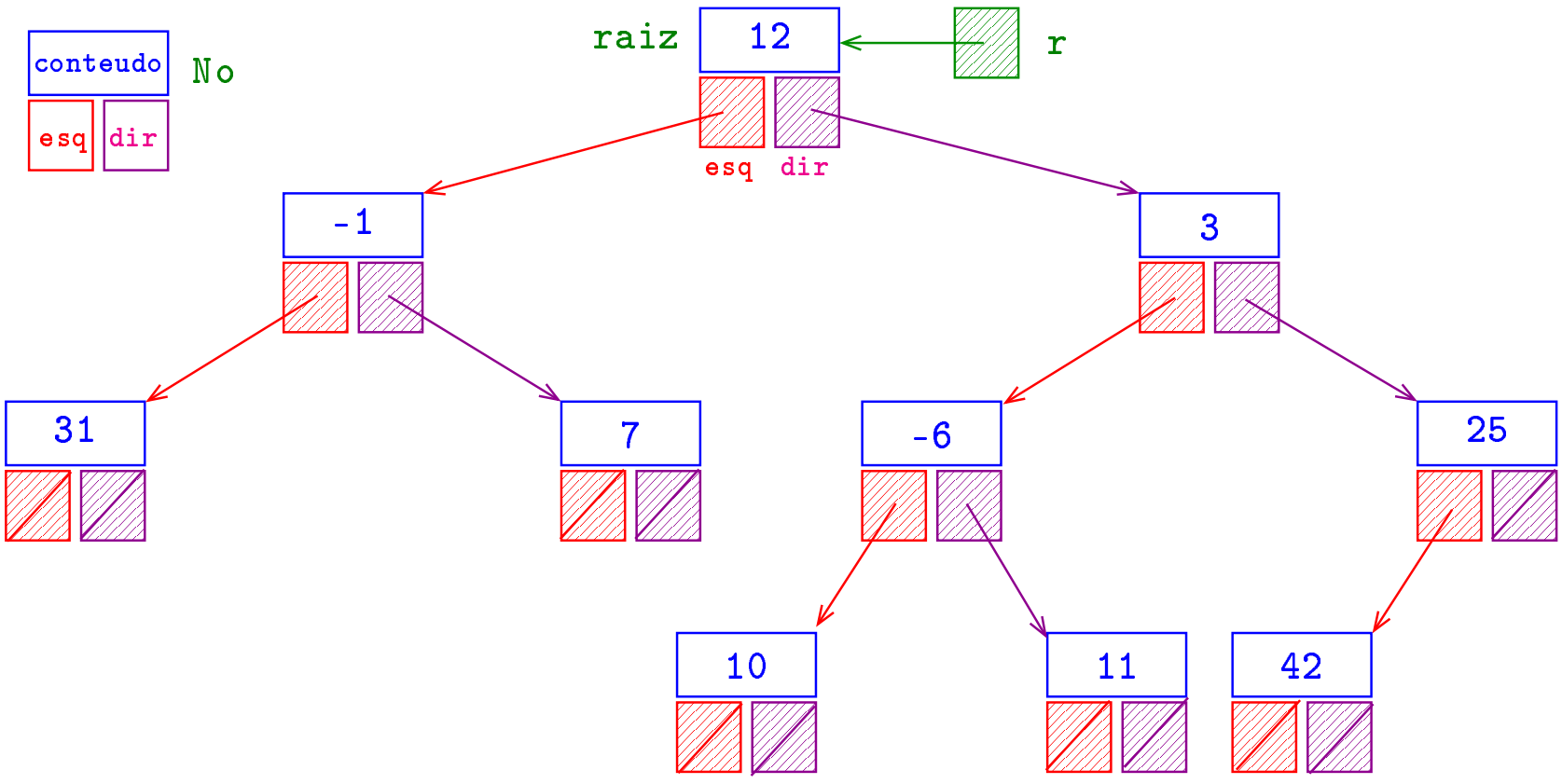


Maneiras de varrer uma árvore

Existem várias maneiras de percorrermos uma árvore binária. Talvez as mais tradicionais sejam:

- *inorder traversal*: esquerda-raiz-direita (e-r-d);
- *preorder traversal*: raiz-esquerda-direita (r-e-d);
- *posorder traversal*: esquerda-direita-raiz (e-d-r);

Ilustração de percursos em árvores binárias



in-ordem (e-r-d): 31 -1 7 12 10 -6 11 3 42 25

pré-ordem (r-e-d): 12 -1 31 7 3 -6 10 11 25 42

pós-ordem (e-d-r): 31 7 -1 10 11 -6 42 25 3 12

AULA 21

Primeiro nó esquerda-raiz-direita

Recebe a raiz `r` de uma árvore binária não vazia e retorna o primeiro nó na ordem e-r-d

Primeiro nó esquerda-raiz-direita

Recebe a raiz r de uma árvore binária não vazia e retorna o primeiro nó na ordem e-r-d

```
No *primeiro(Arvore r)
{
   while (r->esq != NULL)
       r = r->esq;
   return r;
}
```

Altura

A **altura** de `p` é o número de passos do mais longo caminho que leva de `p` até uma folha.

A altura de uma árvore é a altura da sua raiz.
Altura da árvore vazia é -1.

Altura

A **altura** de p é o número de passos do mais longo caminho que leva de p até uma folha.

A altura de uma árvore é a altura da sua raiz.
Altura da árvore vazia é -1.

```
#define MAX(a,b) ((a) > (b)?  (a):  (b))
int altura(Arvore r) {
   if (r == NULL) return -1;
   else {
        int he = altura(r->esq);
        int hd = altura(r->dir);
        return MAX(he,hd) + 1;
   }
{
```

Árvores balanceadas

A altura de uma árvore com n nós é um número entre $\lg(n)$ e n.

Uma árvore binária é **balanceada** (ou **equilibrada**) se, em cada um de seus nós, as subárvores esquerda e direita tiverem *aproximadamente* a mesma altura.

Árvores balanceadas têm altura *próxima* de $\lg(n)$.

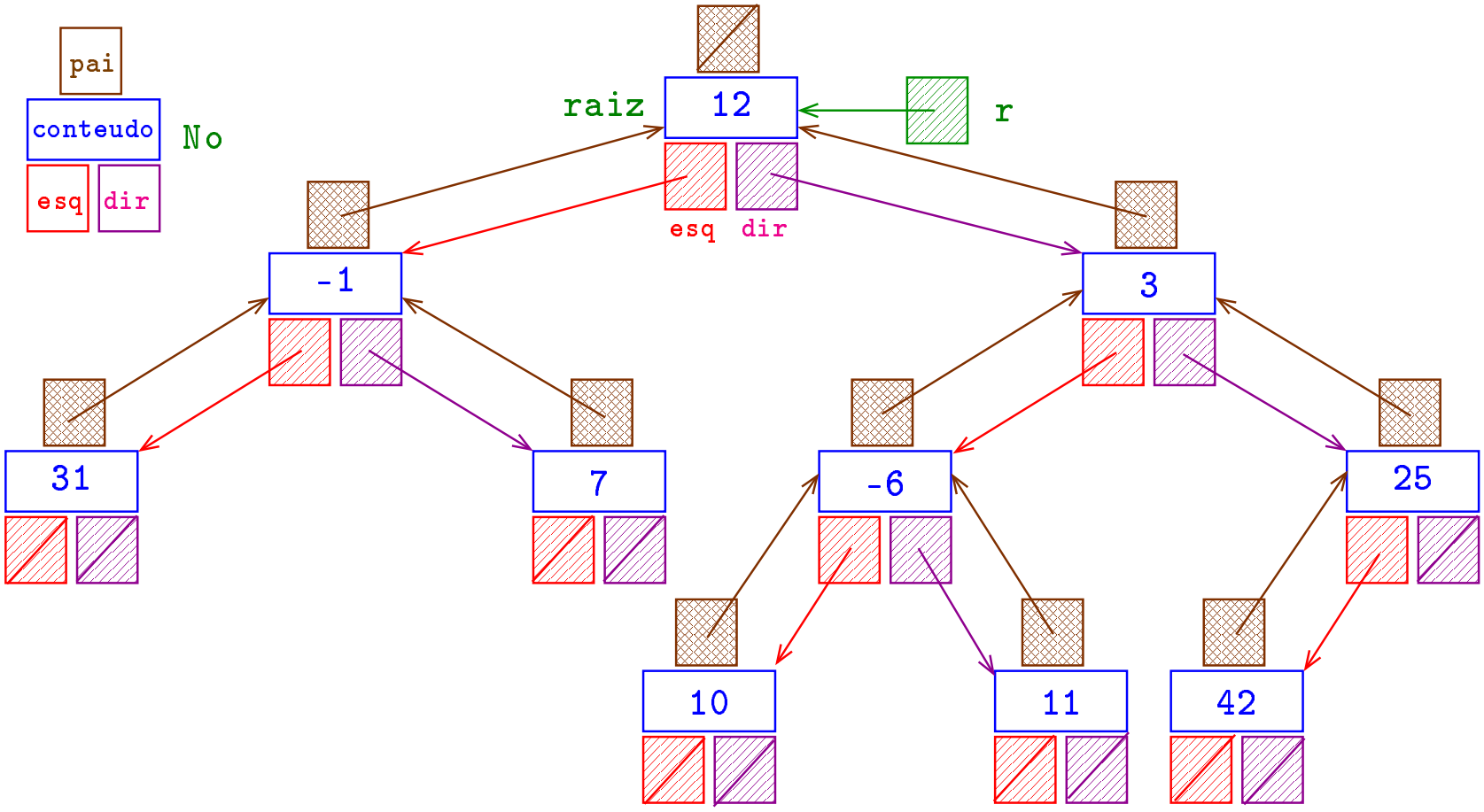
O consumo de tempo dos algoritmos que manipulam árvores binárias dependem frequentemente da altura da árvore.

Nós com campo pai

Em algumas aplicações é conveniente ter acesso imediato ao pai de qualquer nó.

```
typedef struct celula Celula;
struct celula {
   int conteudo; /* tipo devia ser Item*/
   Celula *pai;
   Celula *esq;
   Celula *dir;
};
typedef Celula No;
typedef No *Arvore;
```

Ilustração de nós com campo pai



Sucessor e predecessor

Recebe o endereço p de um nó de uma árvore binária não vazia e retorna o seu **sucessor** na ordem e-r-d.

Sucessor e predecessor

Recebe o endereço p de um nó de uma árvore binária não vazia e retorna o seu **sucessor** na ordem e-r-d.

```
No *sucessor(No *p) {
   if (p->dir != NULL) {
       No *q= p->dir;
       while (q->esq != NULL) q = q->esq;
       return q;
   }
   while (p->pai!=NULL && p->pai->dir==p)
       p = p->pai;
   return p->pai;
}
```

Exercício: função que retorna o **predecessor**.

Mais exercícios

Escreva uma função que recebe o endereço `p`
de um nó de uma árvore binária e

1. devolve o número de nós da árvore.
2. devolve o maior número que aparece na árvore
   ($-\infty$ se a árvore está vazia).
3. espelha a árvore.
4. devolve uma cópia da árvore.
5. libera todos os nós da árvore.

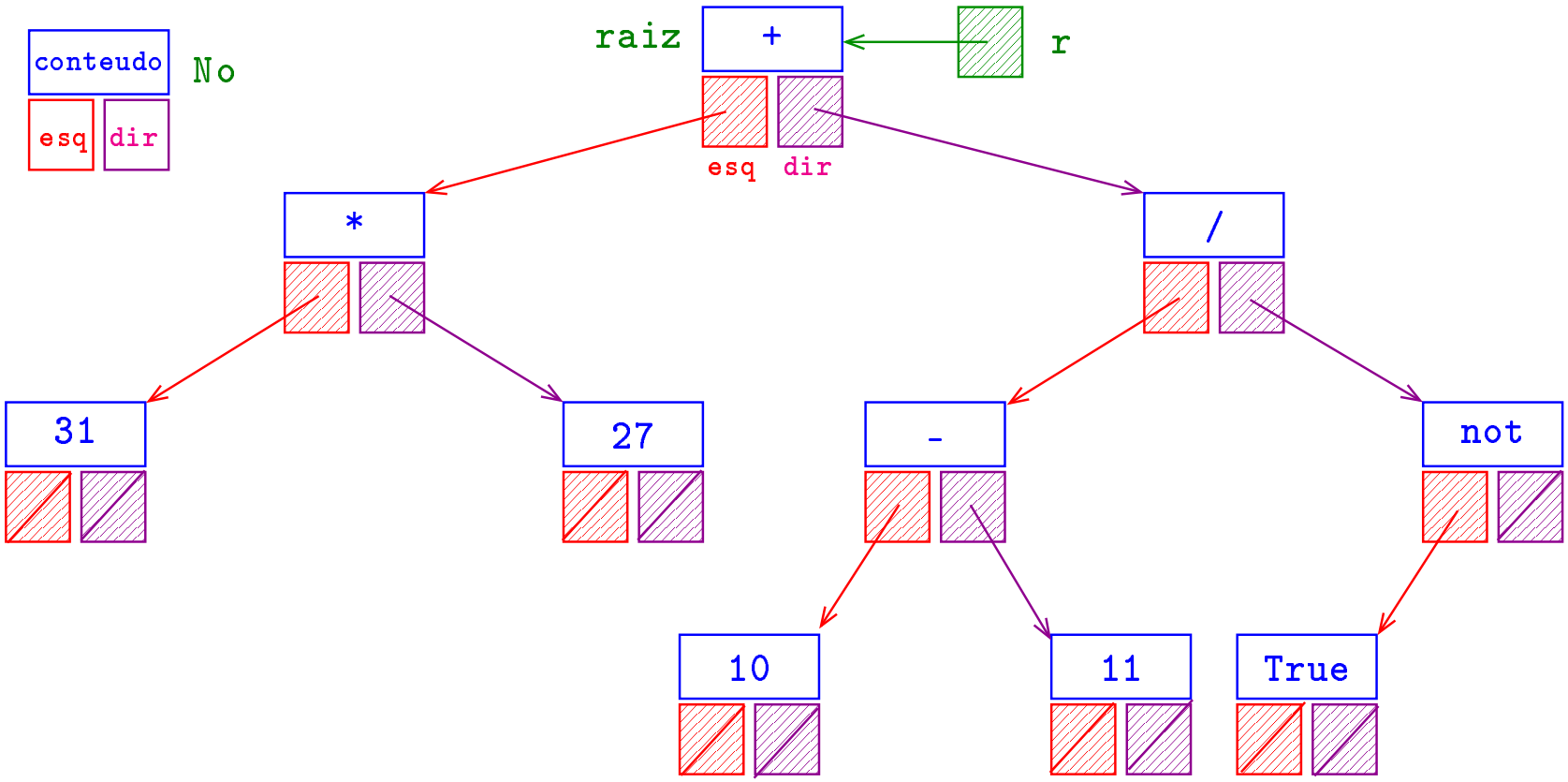
Mais exercícios

Escreva uma função que recebe o endereço `p`
de um nó de uma árvore binária e

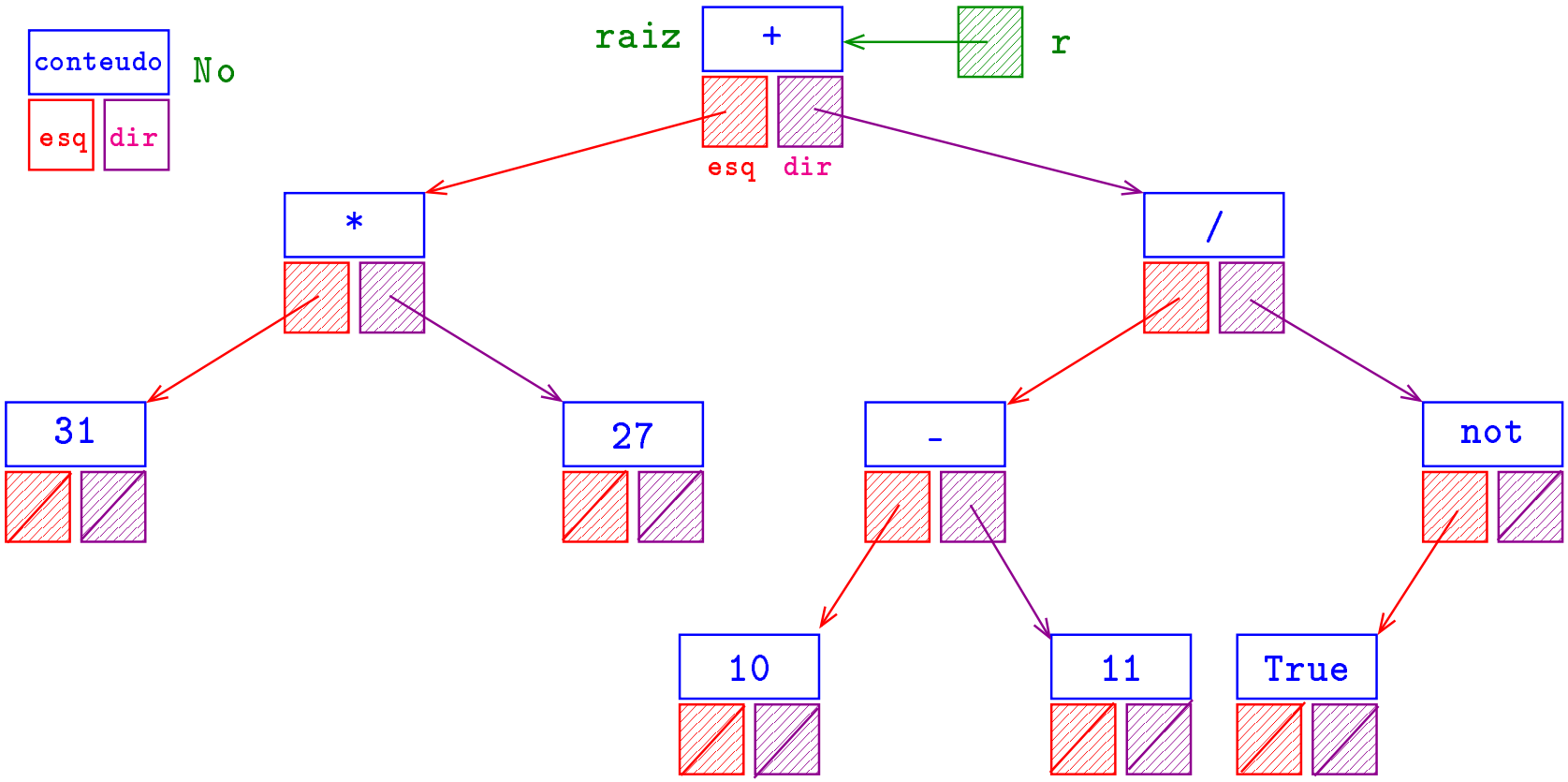
1. devolve o número de nós da árvore.
2. devolve o maior número que aparece na árvore
   ($-\infty$ se a árvore está vazia).
3. espelha a árvore.
4. devolve uma cópia da árvore.
5. libera todos os nós da árvore.

Vamos fazer um agora? Qual?

Árvores com expressões



Árvores com expressões



Dada uma expressão aritmética, como construir a árvore binária que representa esta expressão?