Heaps Esquerdistas e Binomiais

Eduardo Furlan Miranda

2024-04-02

Adaptado do material da Prof. Cristina G. Fernandes

Heaps Esquerdistas

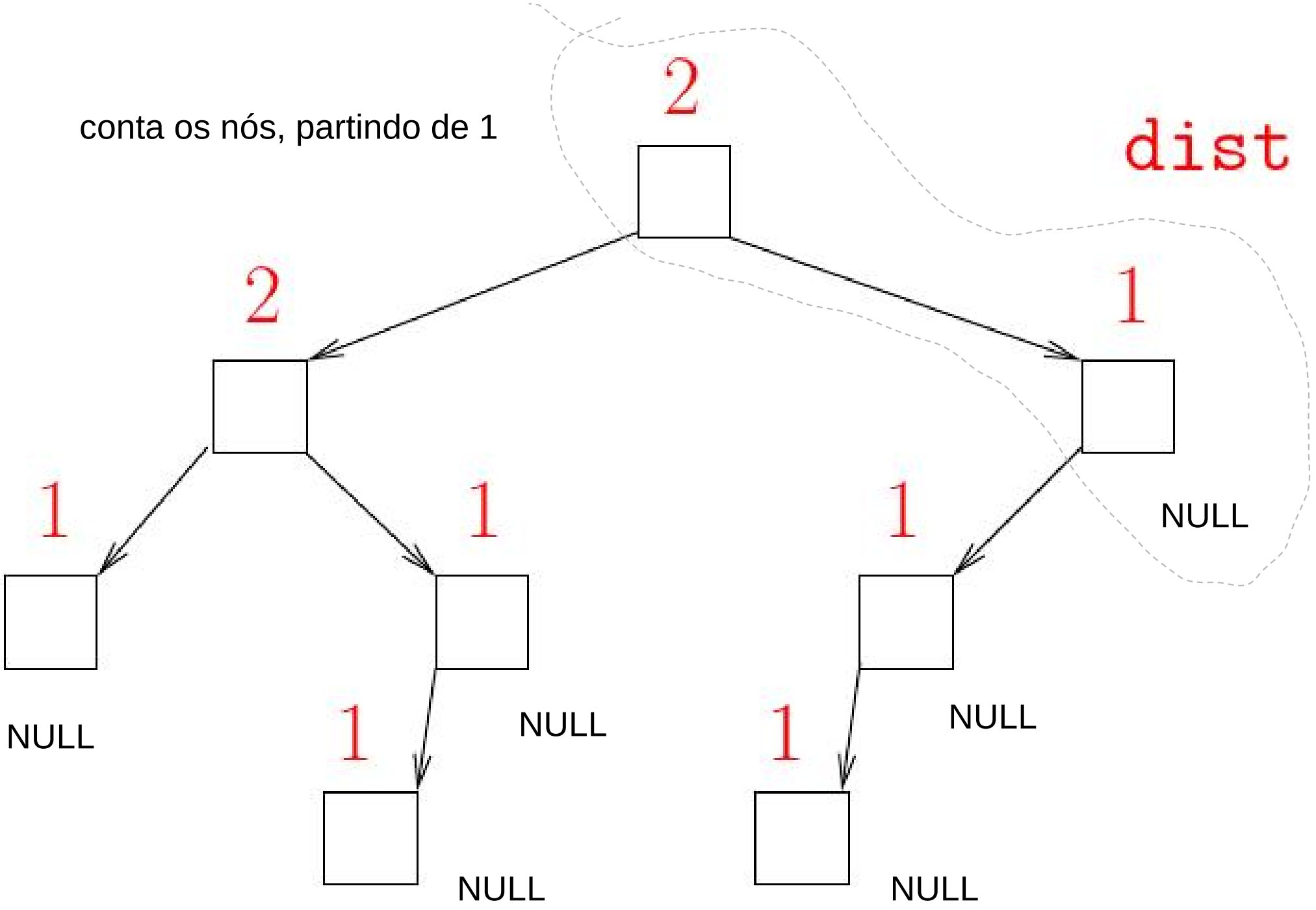
Árvores esquerdistas

- Cada nó x tem 3 campos:
  - **esq[x]**: filho esquerdo de x;
  - **dir[x]**: filho direito de x;
  - **dist[x]**: menor comprimento de um caminho de x a NULL.

```
dist (x)
    se x = NULL então
        devolva 0
    senão
        devolva 1 + min{dist(esq[x]), dist(dir[x])}
```

npl = “Null Path Lenght” = nº de nós entre x e um NULL

conta os nós, partindo de 1

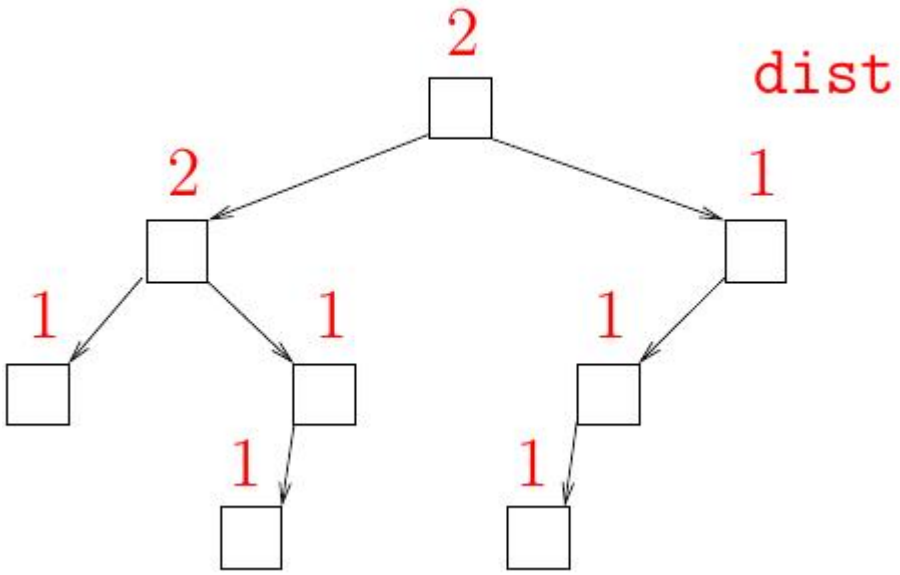


Árvores esquerdistas

- Uma árvore é esquerdista se

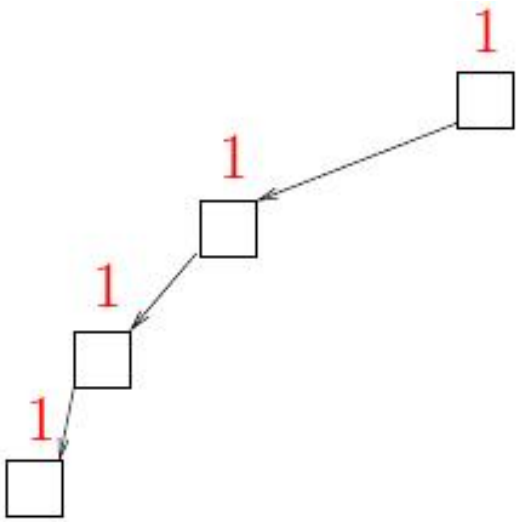
$$dist[esq[x]] \geq dist[dir[x]]$$

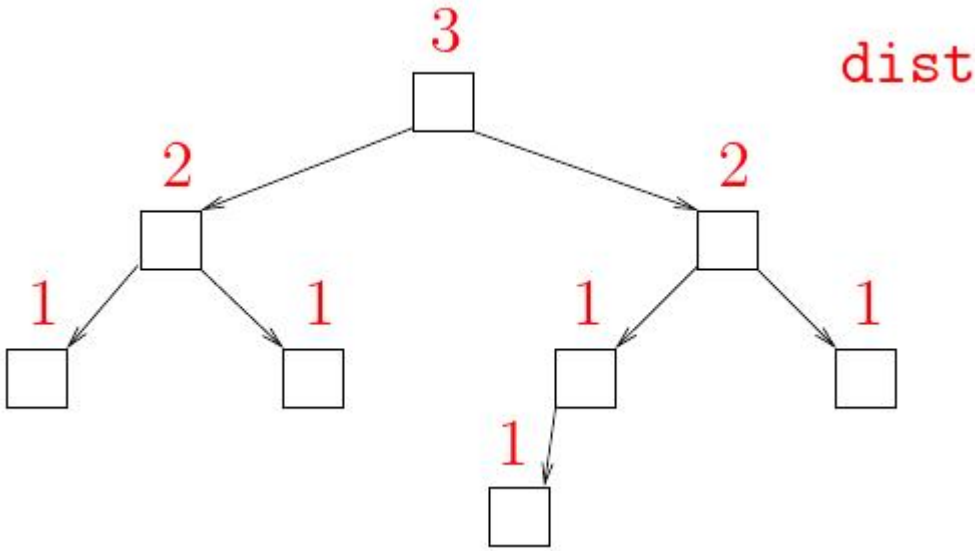
para todo nó x(dist[NULL]=0)



conta os nós até o primeiro NULL

Árvores esquerdistas

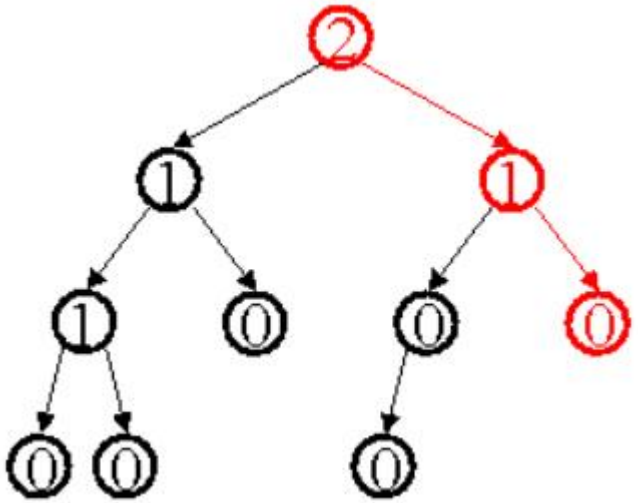




esquerda pode ser igual direita

Caminho direitista (*Right Path*)

- O caminho "direitista" de um nó (por exemplo, a raiz) é obtido seguindo os filhos "direitistas" até que um filho NULL seja alcançado
- dcomp[x] := número de nós no caminho direitista de x
- Se x é um nó de uma árvore esquerdista, então dist[x] = dcomp[x].



(aqui foi adotado o índice inicial 0)

Heap esquerdista

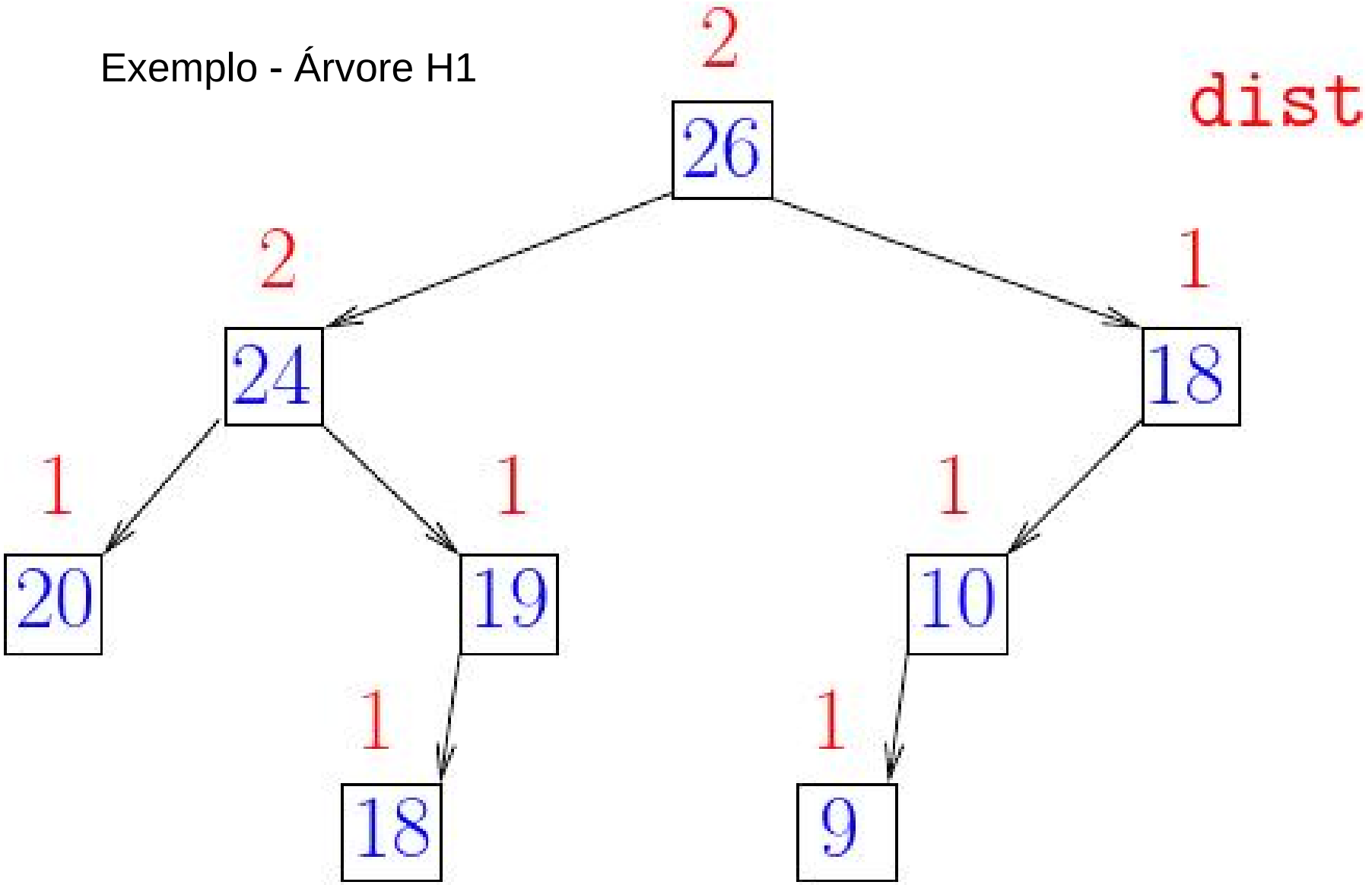
- H := árvore
- raiz[H] := raiz de H
- prior[x] := prioridade do nó x
- pai[x] := pai do nó x
- Um heap esquerdista H é uma árvore esquerdista que satisfaz

$$\text{prior}[\text{pai}[x]] \geq \text{prior}[x]$$

para todo nó $x \neq \text{raiz}[H]$

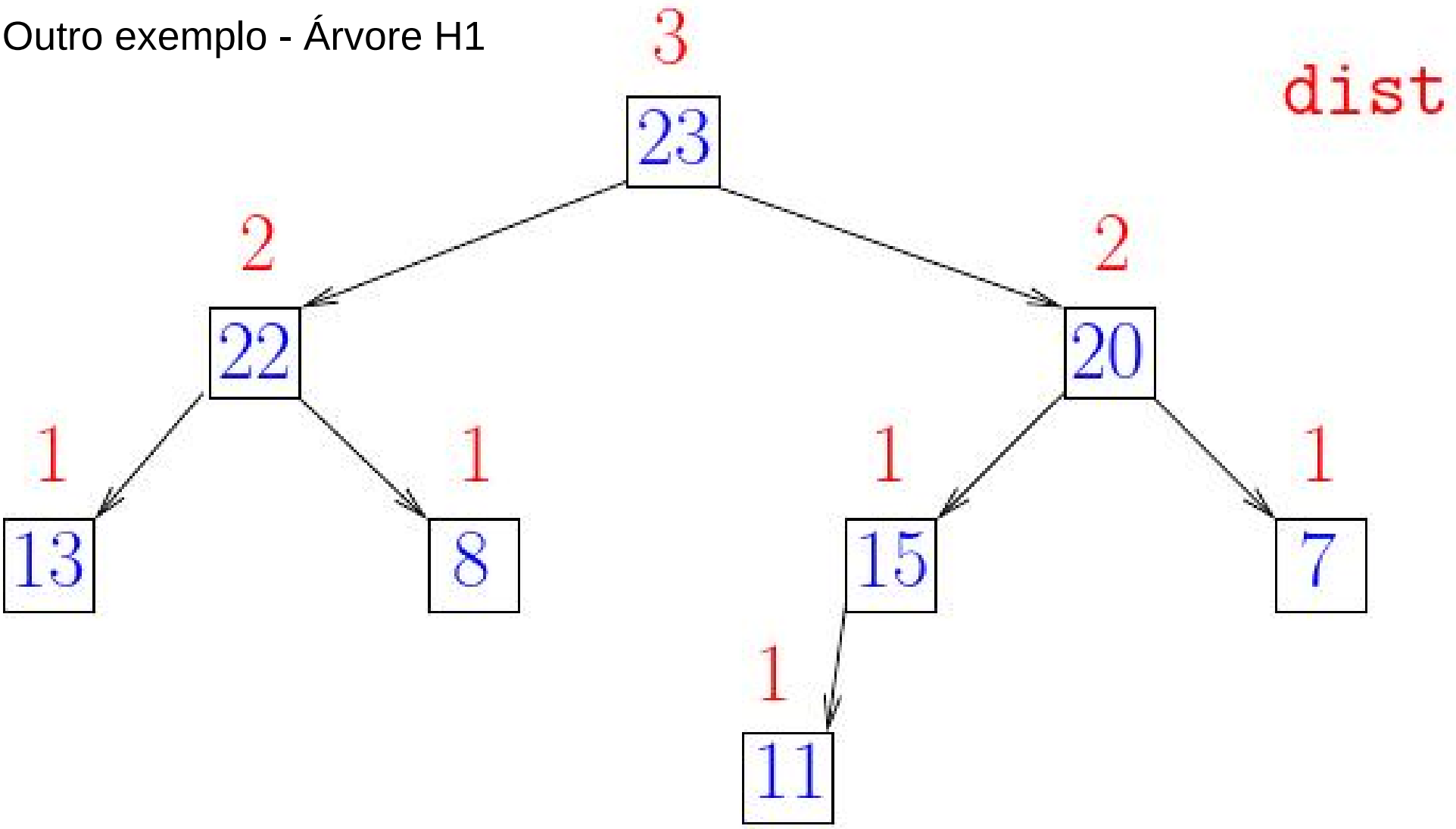
Heap esquerdista

Exemplo - Árvore H1

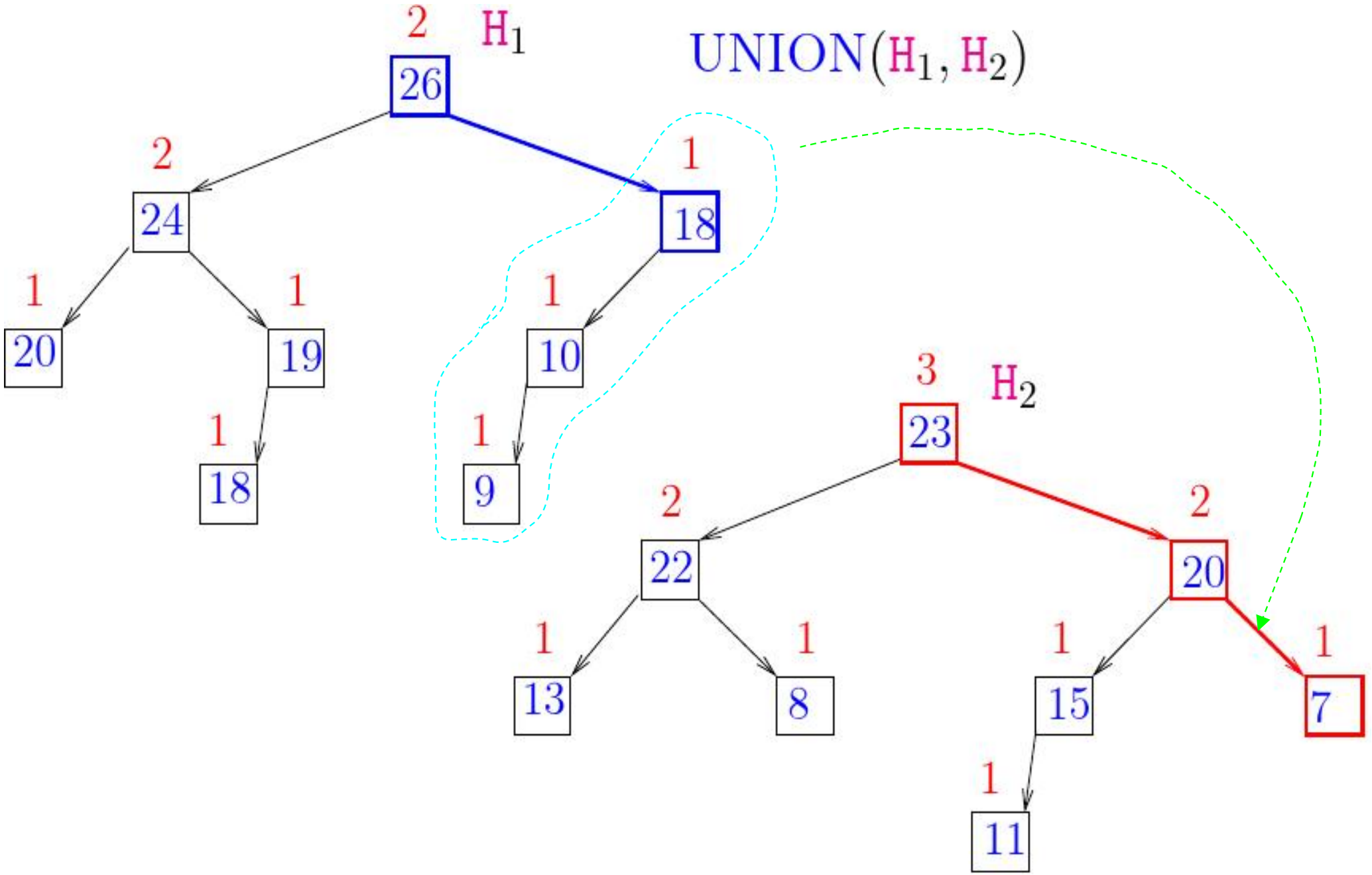


Heap esquerdista

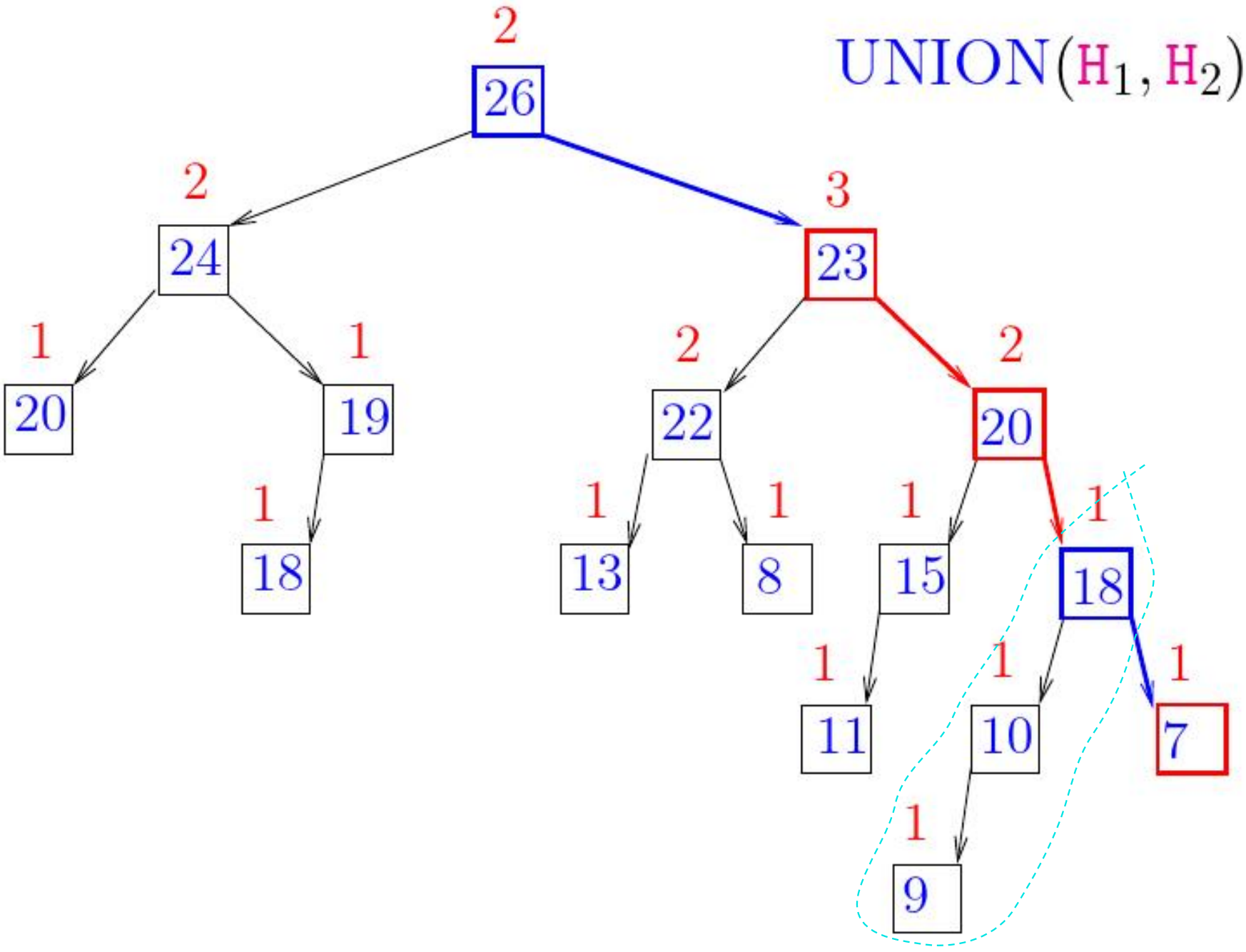
Outro exemplo - Árvore H1



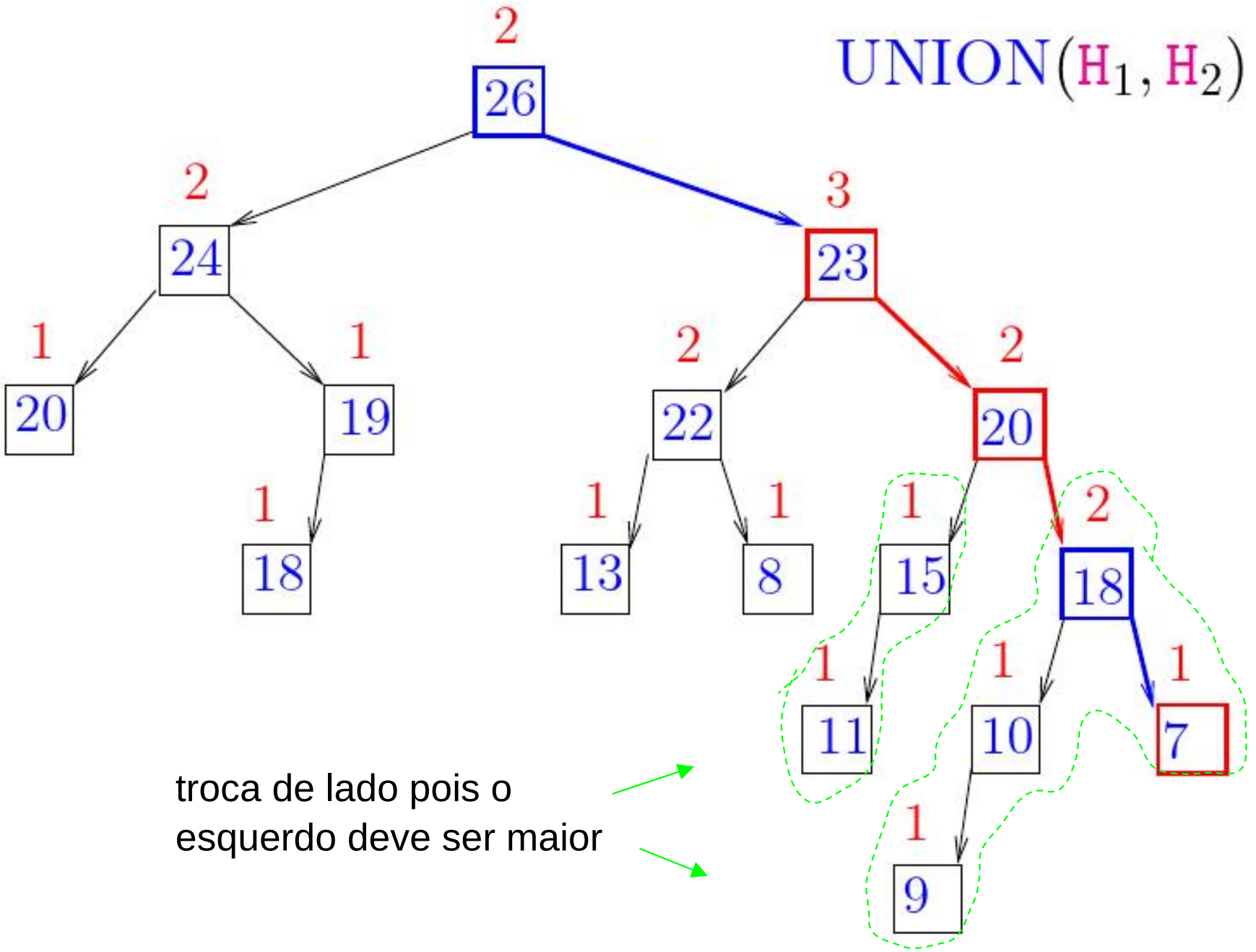
Rotina básica de manipulação



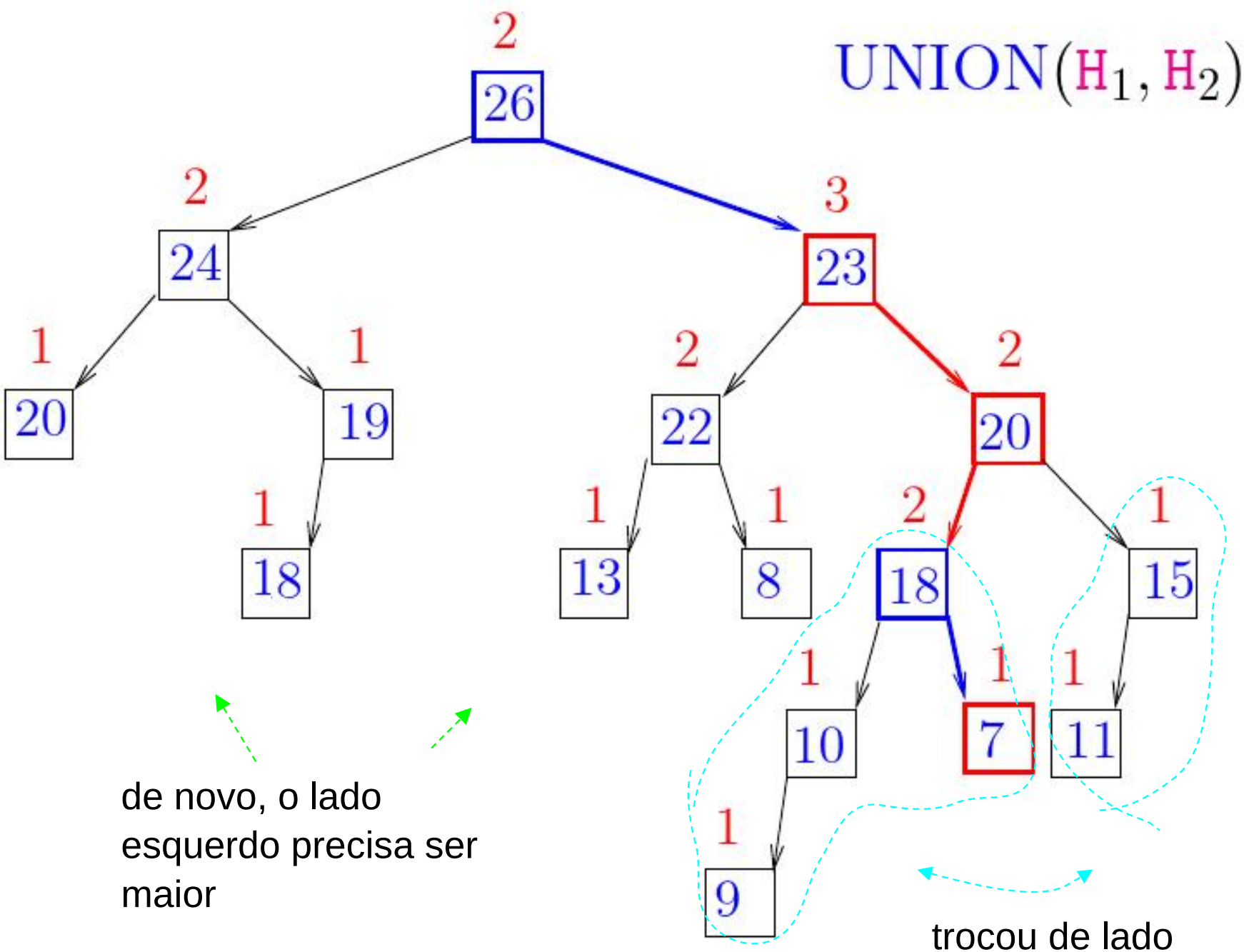
Rotina básica de manipulação

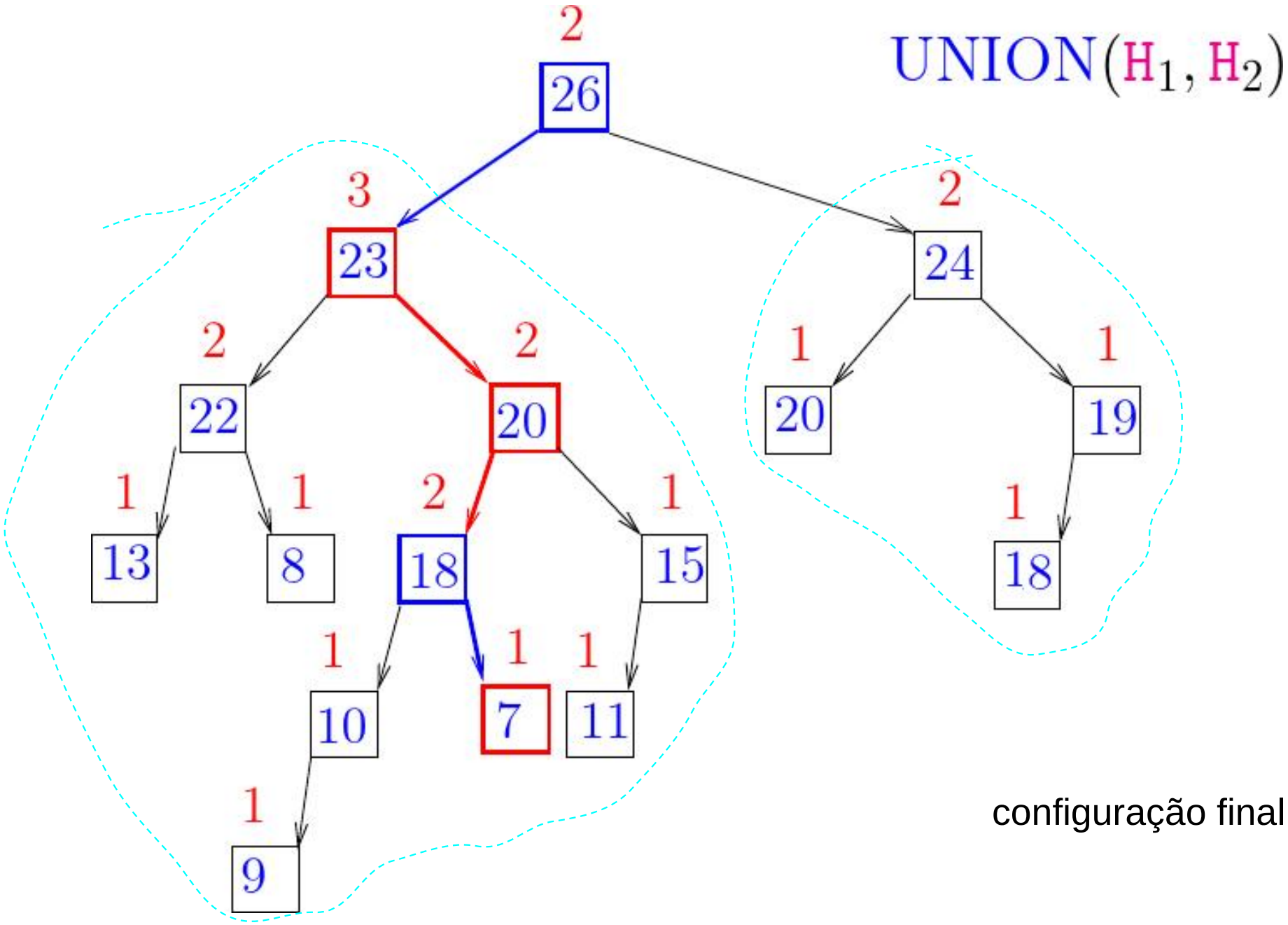


Rotina básica de manipulação



Rotina básica de manipulação





configuração final

Cliente MinPQ: TopM

```
int main(int argc, char * argv) {
  int M = atoi(argv[1]);
  Transaction * t;
  MinPQInit(M + 1);
  while ((t = readT()) != NULL) {
    MinPQInsert(t); /* Mantemos M maiores transacoes na PQ */
    if (MinPQSize() > M) {
      t = MinPQDelMin();
      freeT(t);
    }
  }
  stackInit(); /* Pilha para imprimir da maior para a menor */
  while (!MinPQEmpty()) stackPush(MinPQDelMin());
  while (!stackEmpty()) {
    t = stackPop();
    printT(t);
    freeT(t);
  }
  stackFree();
  MinPQFree();
}
```

Interface para PQ-mínimo

- Arquivo MinPQ.h

```
void MinPQInit(int)       cria uma PQ
void MinPQInsert(Item x) insere x nesta PQ
Item MinPQMin()           devolve um mínimo
Item MinPQDelMin()        remove e devolve um mínimo de PQ
int  MinPQSize()          número de itens
bool MinPQEmpty()         PQ está vazia?
void MinPQFree()          destroi esta PQ
```

A seguir, uma implementação dessa interface, usando árvores esquerdistas

LMinPQ: struct node, Link e newNode

- Cada nó da árvore esquerdista tem quatro campos:

```
typedef struct node * Link;
struct node {
    Transaction * item;
    Link left, right;
    int dist;
};
Link newNode(Transaction * item, Link left,
Link right, int dist) {
    Link p = mallocSafe(sizeof( * p));
    p -> item = item;
    p -> left = left;
    p -> right = right;
    p -> dist = dist;
    return p;
}
```

Arquivo LMinPQ.c: esqueleto

```c
#include "MinPQ.h"

static Link root;
static int n;

void MinPQInit(int max) {...}
void MinPQInsert(Item item) {...}
Item MinPQMin() {...}
Item MinPQDelMin() {...}
bool MinPQEmpty() {...}
int MinPQSize() {...}
void MinPQFree() {...}
static Link merge(Link r1, Link r2) {...}
```

LMinPQ: init(), empty() e size()

```
void MinPQInit(int m) {
    root = NULL;
    n = 0;
}
int MinPQSize() {
    return n;
}
bool MinPQEmpty() {
    return n == 0;
}
```

LMinPQ: insert() e delMin()

```
void MinPQInsert(Item item) {
    Link s = newNode(item, NULL, NULL, 1);
    root = merge(root, s);
    n++;
}
```

LMinPQ: insert() e delMin()

```
void MinPQInsert(Item item) {
    Link s = newNode(item, NULL, NULL, 1);
    root = merge(root, s);
    n++;
}
Item MinPQDelMin() {
    Item item = root -> item;
    Link s = root;
    root = merge(root -> left, root -> right);
    freeNode(s);
    n--;
    return item;
}
```

LMinPQ: merge()

- **merge(r1, r2)** essencialmente intercala as listas encadeadas dos caminhos direitistas de r1 e r2
  - O essencialmente é devido ao fato de ser necessário acertamos os campos **dist** durante a volta da recursão, como é feito mais adiante

```
static Link merge(Link r1, Link r2) {
    if (r1 == null) return r2;
    if (r2 == null) return r1;
    if (less(r2 -> item, r1 -> item)) {
        Link t = r1;
        r1 = r2;
        r2 = t;
    }
    r1 -> right = merge(r1 -> right, r2);
    return r1;
}
```

LMinPQ: merge()

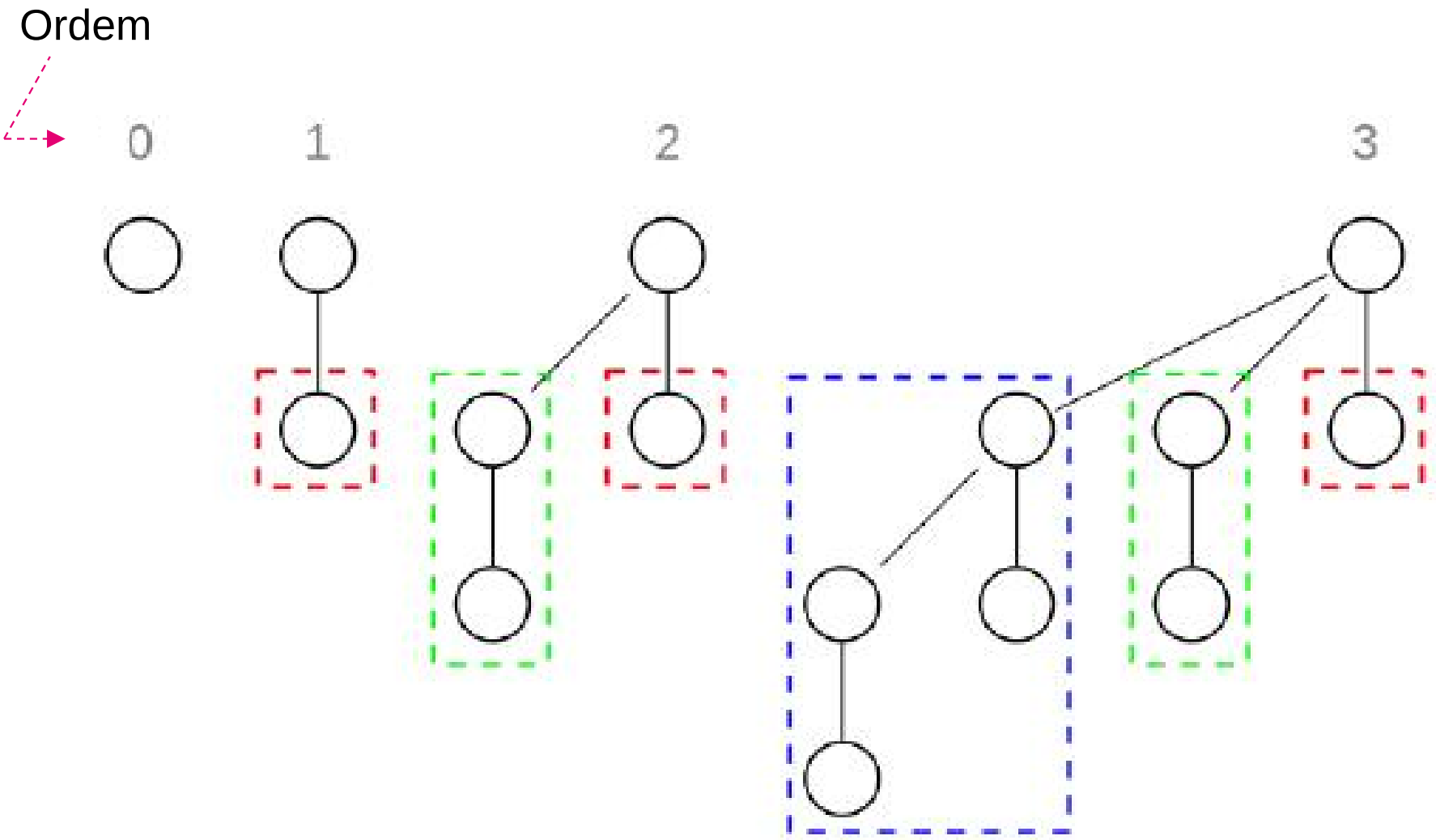
```
static Link merge(Link r1, Link r2) {
        Link t;
        if (r1 == null) return r2;
        if (r2 == null) return r1;
        if (less(r2 -> item, r1 -> item)) {
            t = r1;
            r1 = r2;
            r2 = t;
        }
        /* r1 aponta para o menor item */
        if (r1 -> left == NULL) r1 -> left = r2;
        else {
```

LMinPQ: merge() (continuação)

```
else {
    r1 -> right = merge(r1 -> right, r2);
    if (r1 -> left -> dist < r1 -> right -> dist) {
        t = r1 -> left;
        r1 -> left = r1 -> right;
        r1 -> right = t;
    }
    r1 -> dist = r1 -> right -> dist + 1;
}
return r1;
}
```

Heaps Binomiais

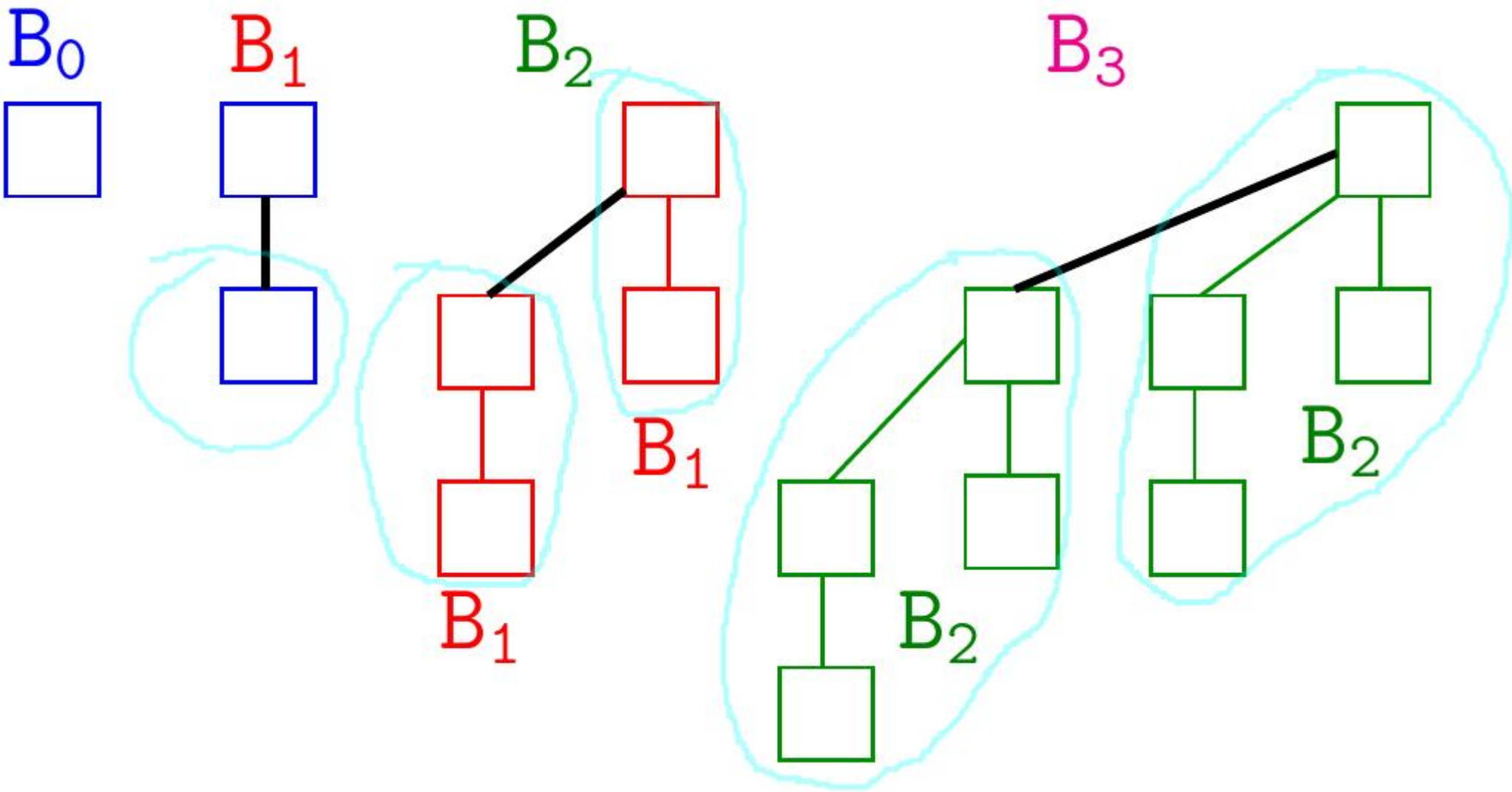
Binomial heaps



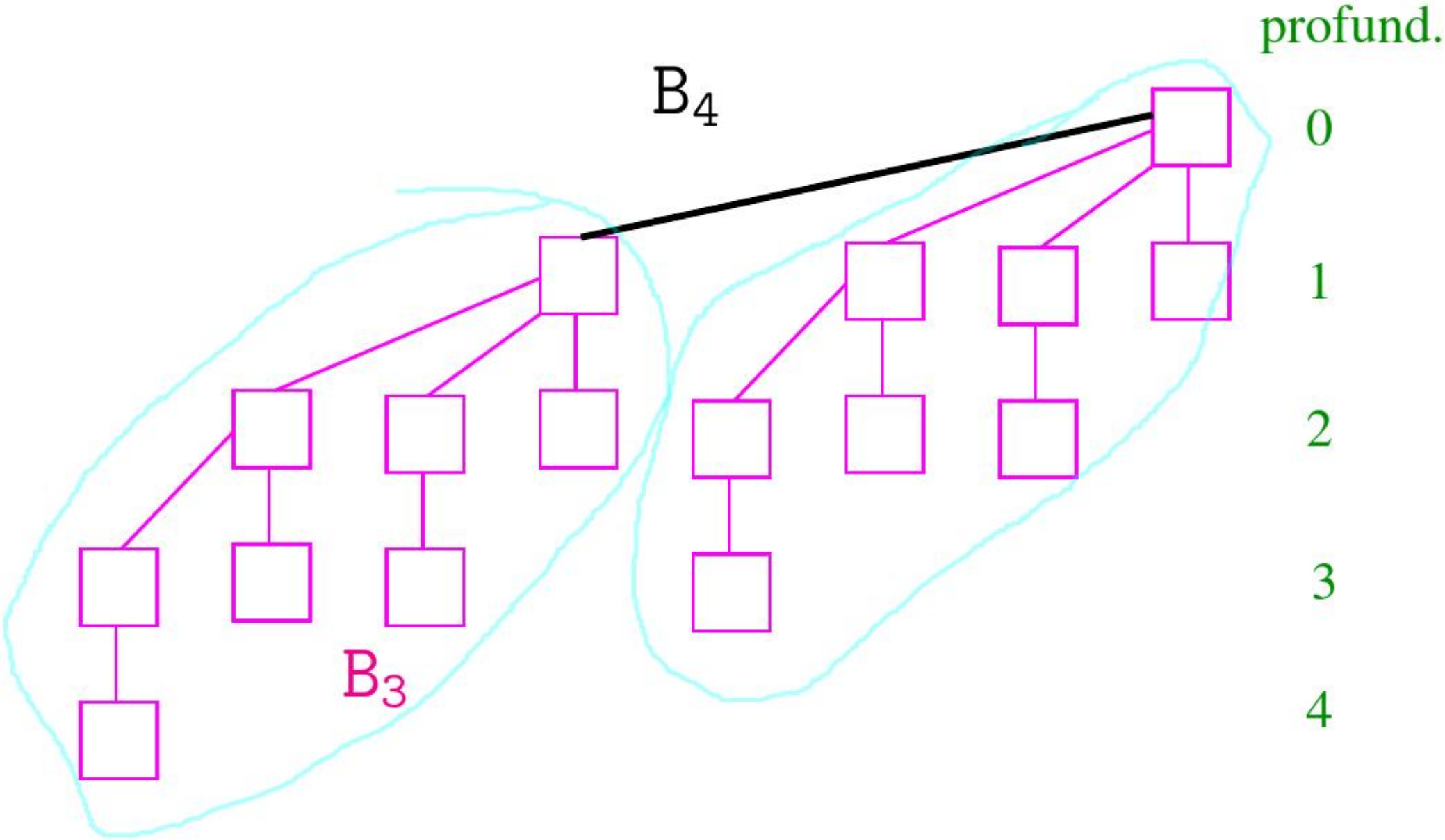
Binomial trees

- Os nós de uma árvore ordenada têm seus filhos ordenados:
  - primeiro, segundo, . . .
- Árvores binomiais são definidas recursivamente:
  - Um nó é a árvore binomial B0 de ordem 0
  - Para k = 1, 2, . . .
    - A árvore binomial B_k de ordem k consiste de duas árvores B_{k-1} ligadas
      - A raiz de uma é o filho mais à esquerda da raiz da outra

Binomial trees



Binomial trees

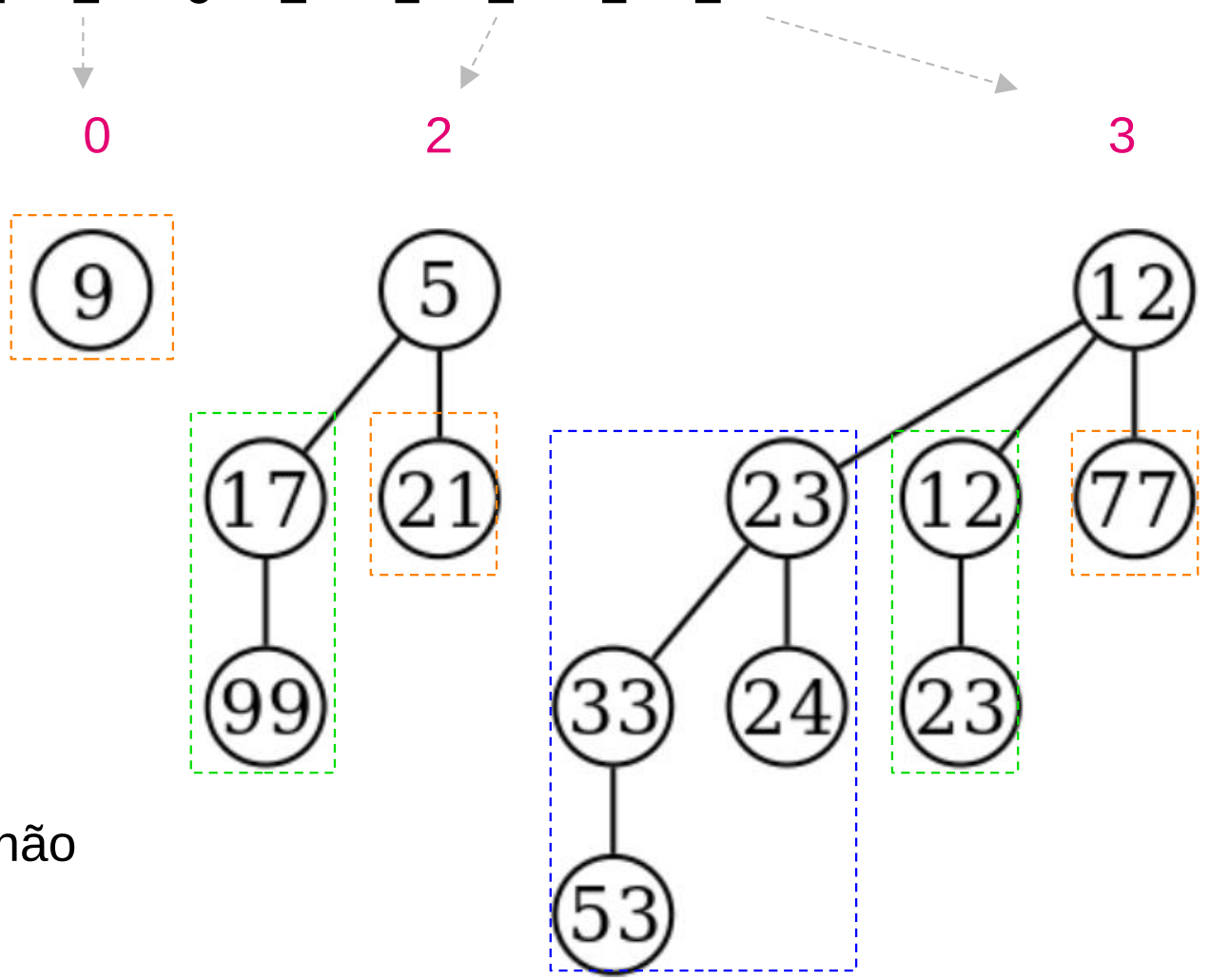


Binomial heap

- Uma *binomial heap* **H** é uma coleção de *binomial trees* que satisfaz as propriedades:
  - Cada *binomial tree* em H é uma "MinPQ" ou "MaxPQ":
    - O valor associado ao item de cada nó é "menor ou igual" ou "maior ou igual" ao valor associado aos seus filhos
  - **H** possui no máximo uma *binomial tree* de cada ordem

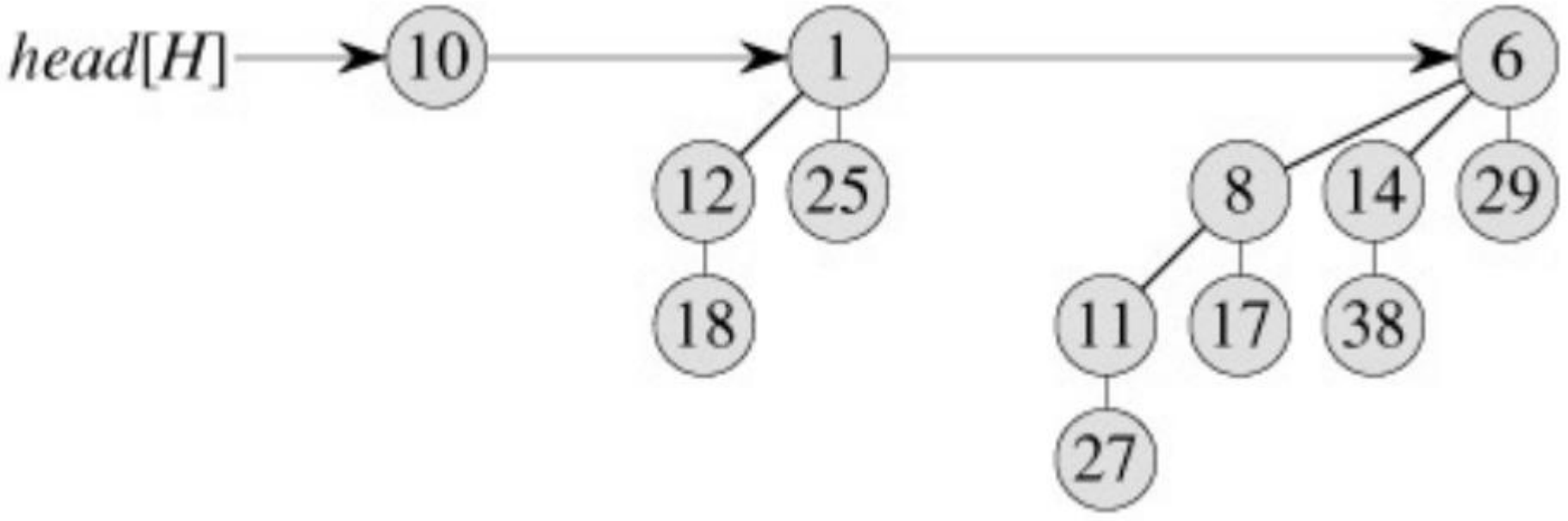
Binomial heap

- Em uma *binomial heap* **H** com **n** itens, os dígitos na representação binária de **n** indicam a ordem das *binomial trees* que formam **H**:
  - $n = 13 = 1 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 + 1 \times 2^3$

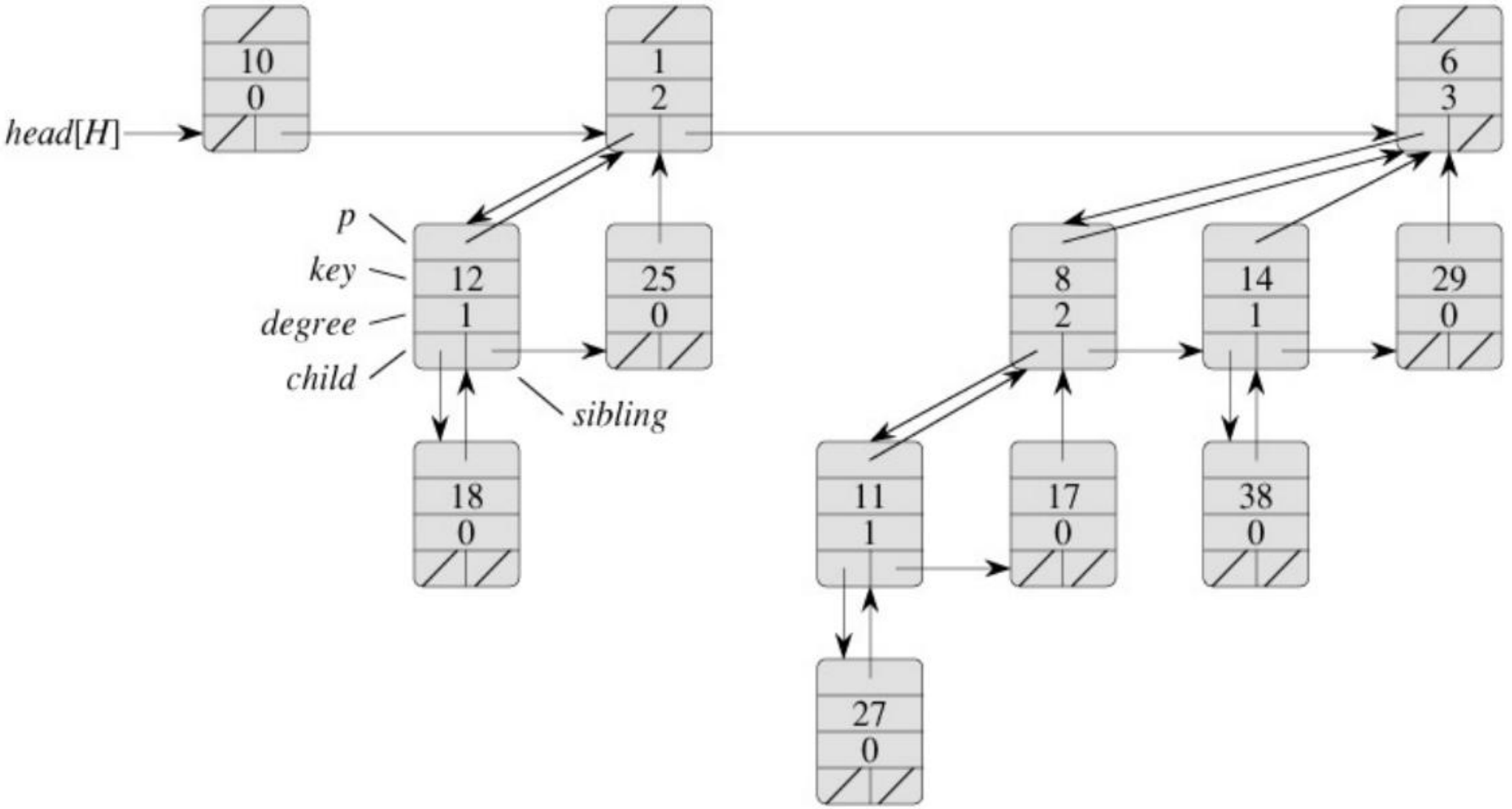


(este exemplo não tem a ordem 1)

Binomial heap: estrutura de dados



Implementação - exemplo

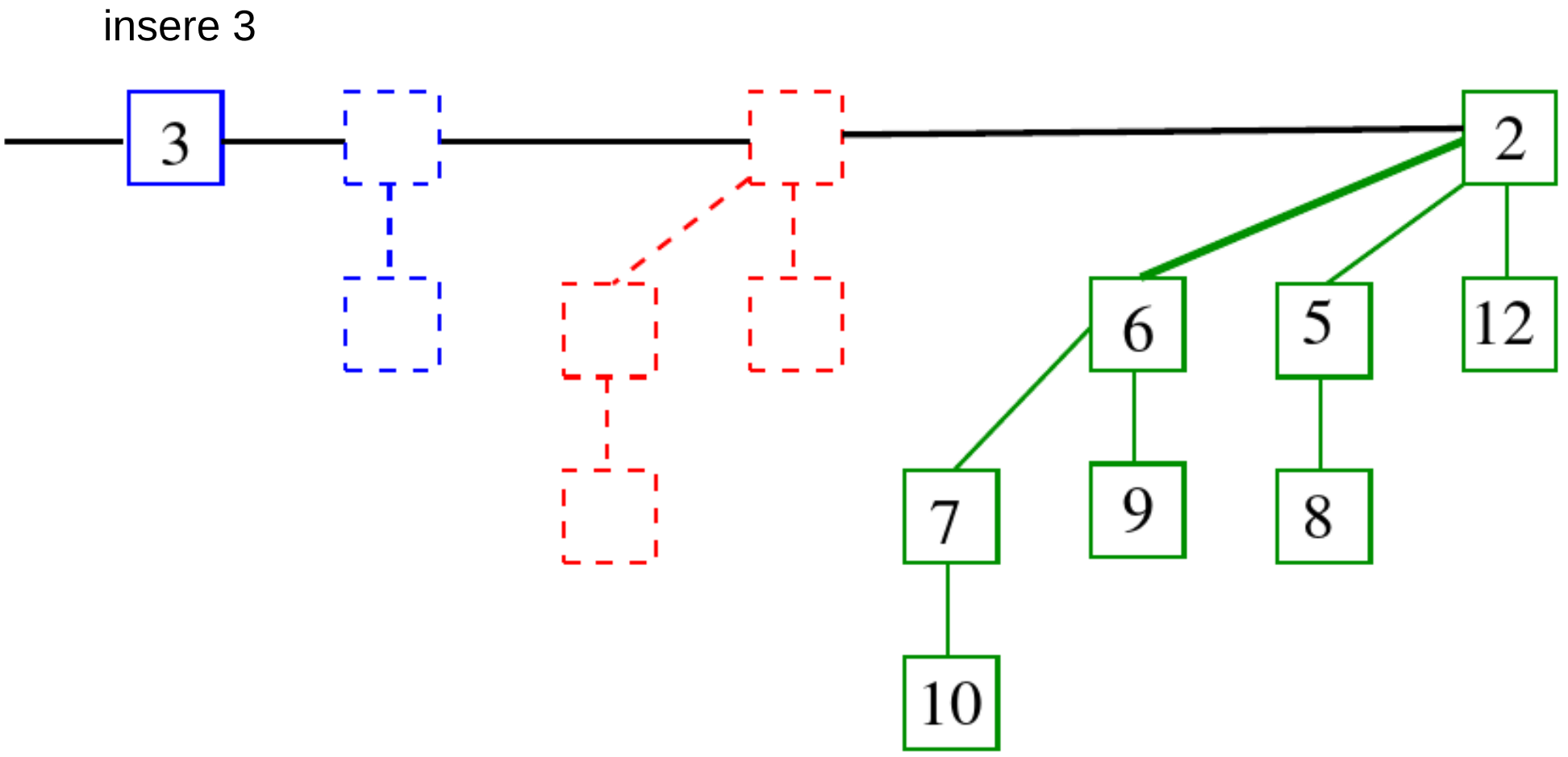


BinomialMinPQ: struct node e Link

- Representação de um nó de uma binomial tree
- A lista de irmão está em ordenada de acordo com o grau dos nós (ordem das árvores)

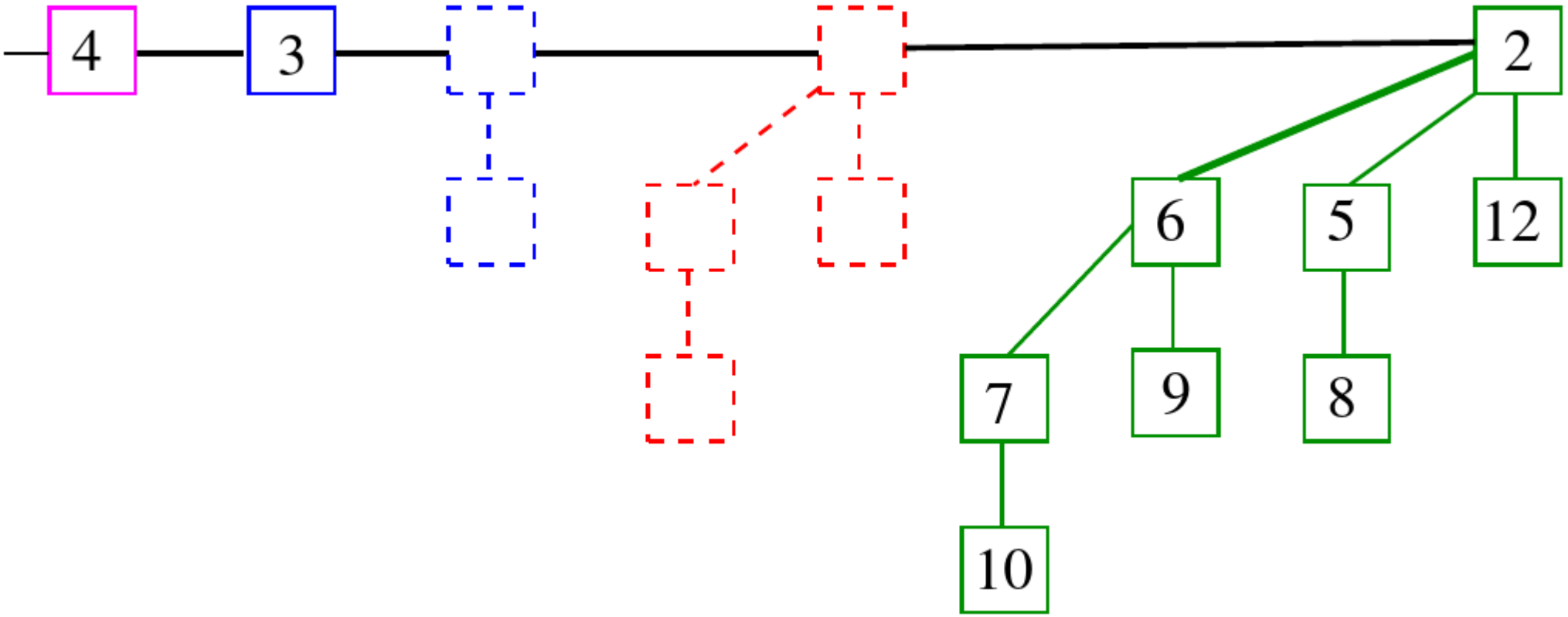
```
typedef struct node * Link;
struct node {
    Item item;
    Link child,   /* filho mais a esquerda */
    Link sibling, /* lista de irmãos */
    Link parent;  /* pai */
    int order;    /* ou grau, degree */
};
```

BinomialMinPQ: insert()

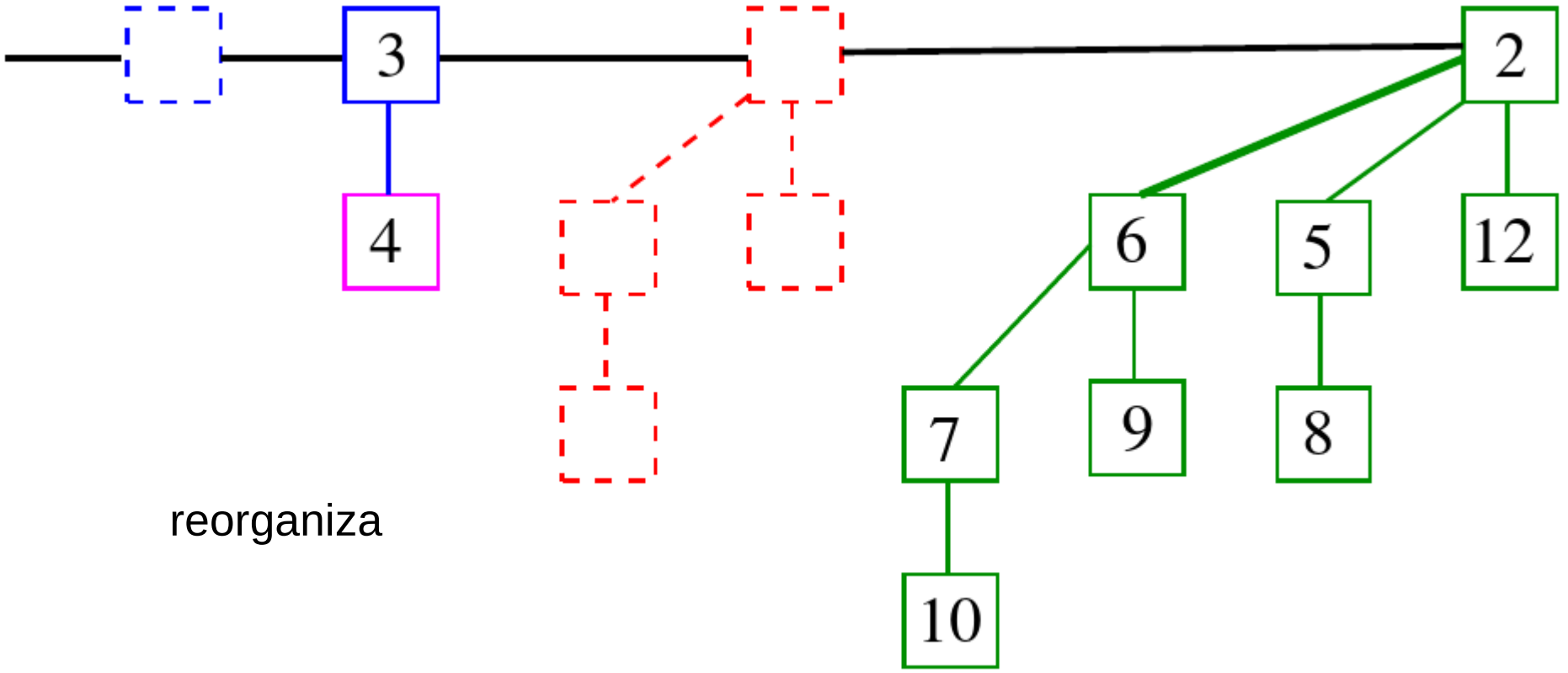


BinomialMinPQ: insert()

insere 4



BinomialMinPQ: insert()



reorganiza

Referências

FERNANDES, C. G. **Árvores esquerdistas e heaps binomiais**. [S. l.]: IME-USP, 2020. Disponível em: https://www.ime.usp.br/~cris/aulas/20_2_323/slides/aula07.pdf.