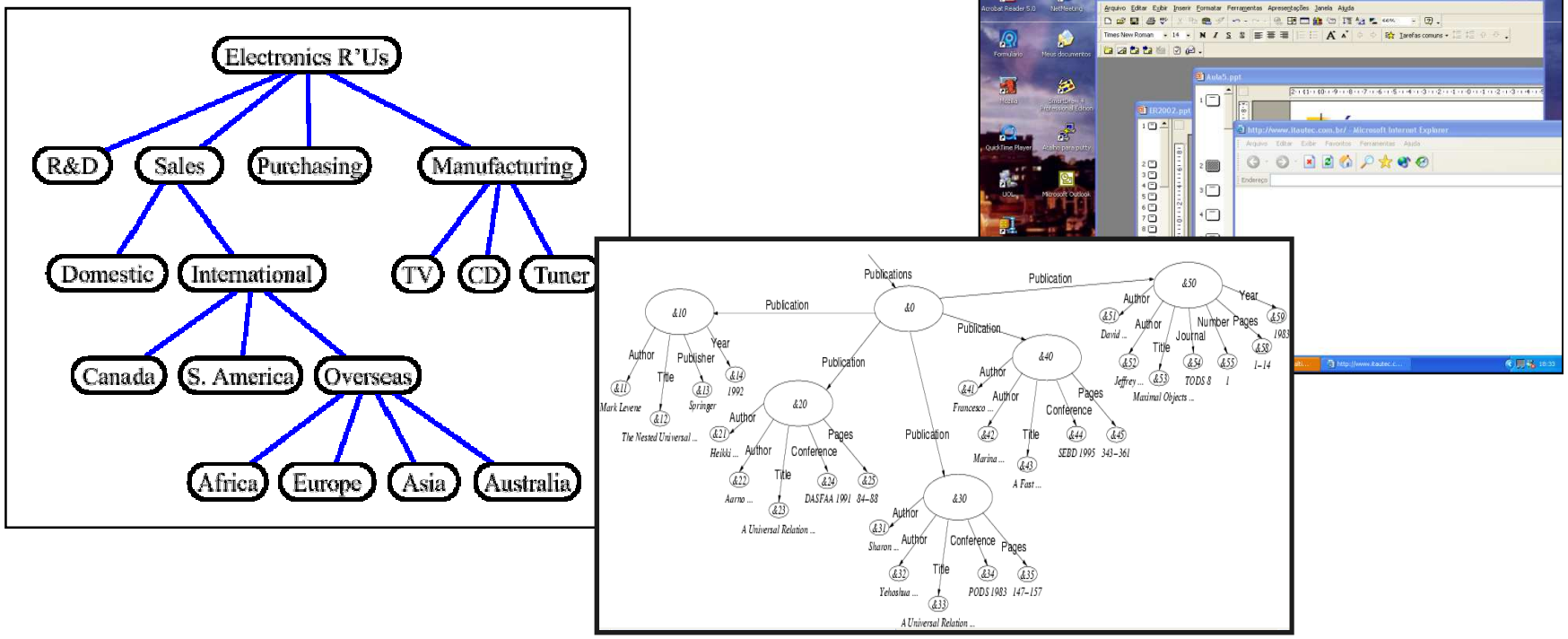
Árvores

Universidade Federal do Amazonas
Departamento de Eletrônica e Computação

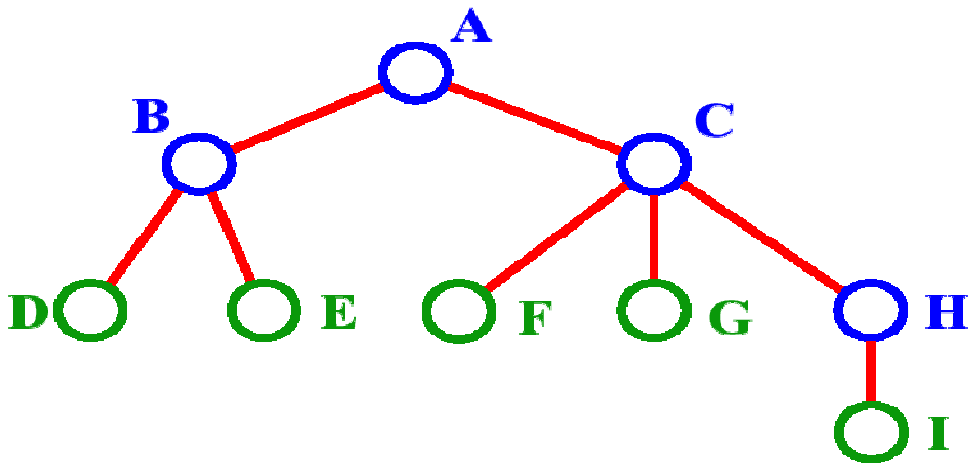
Árvores

- Árvores são conjuntos cujos elementos guardam uma relação hierarquica entre eles



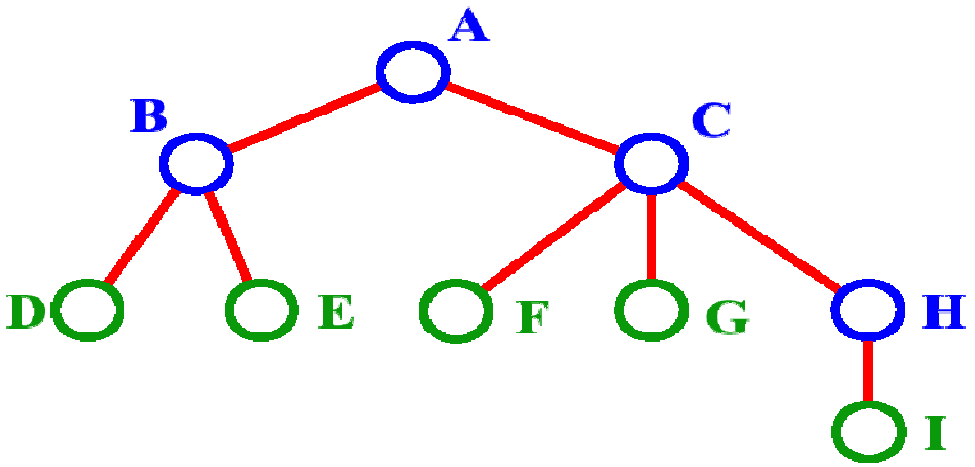
Terminologia (1)

- A é o nodo raiz.
- B é o pai de D e de E.
- D e E são filhos de B.



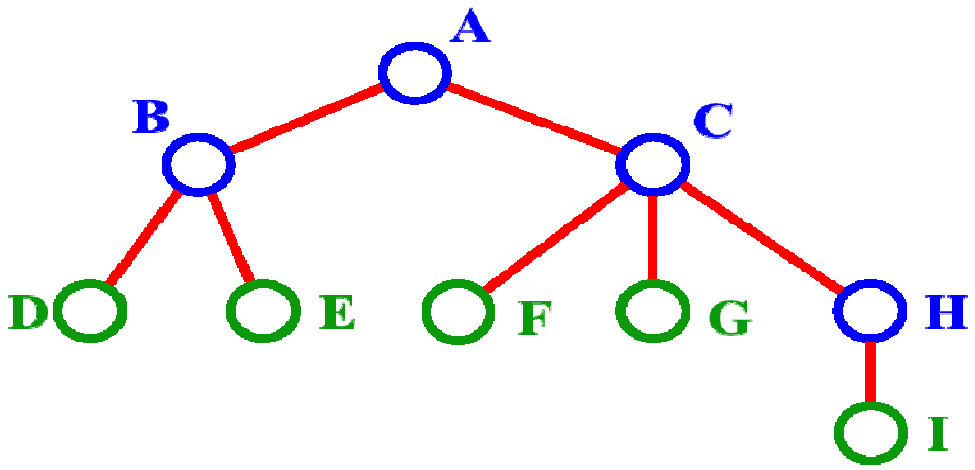
Terminologia (2)

- A é um ancestral de D , de E e de B.
- D,E e B são descendentes de A.
- C é irmão de B



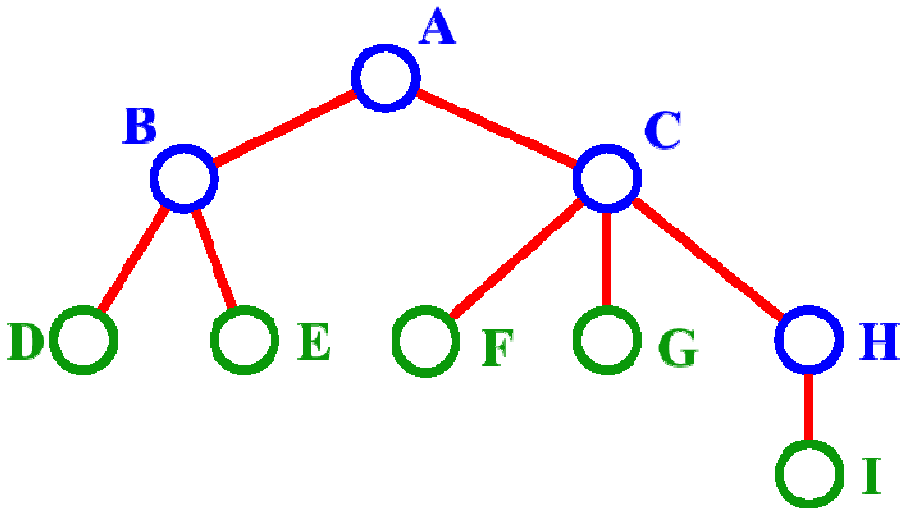
Terminologia (3)

- D, E, F, G, I são folhas.
- A, B, C, H são nodos internos.



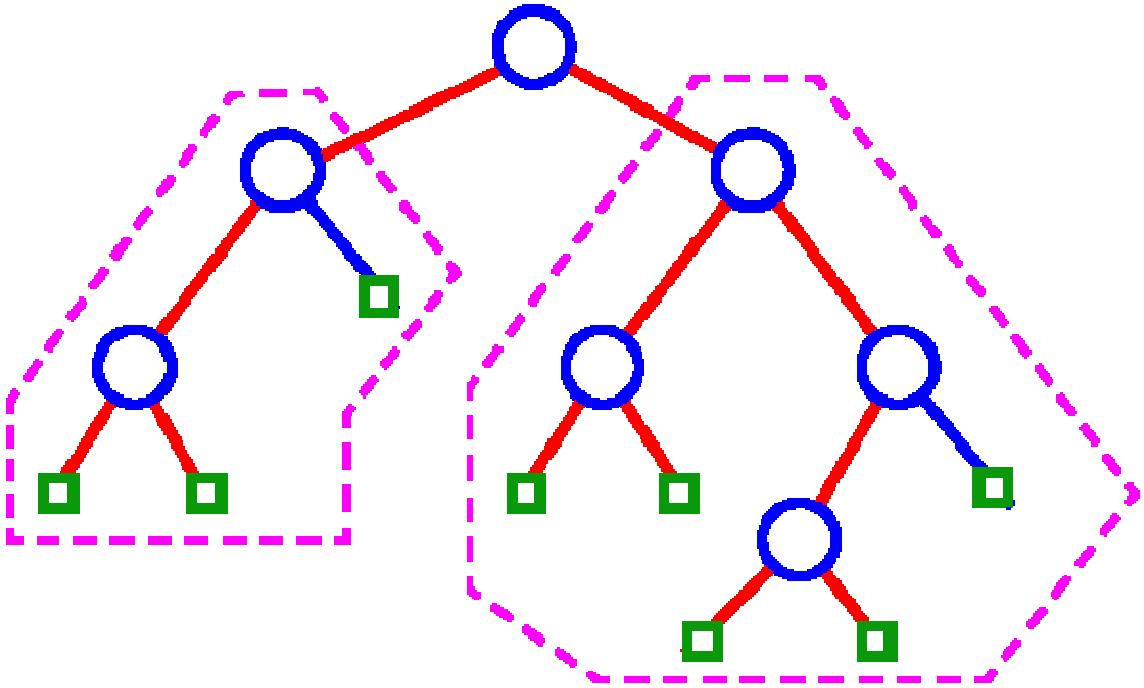
Terminologia (4)

- A profundidade *ou* nível de E é 2
- A altura da árvore é 3
- O grau do nodo *B* é 2



Árvores binárias

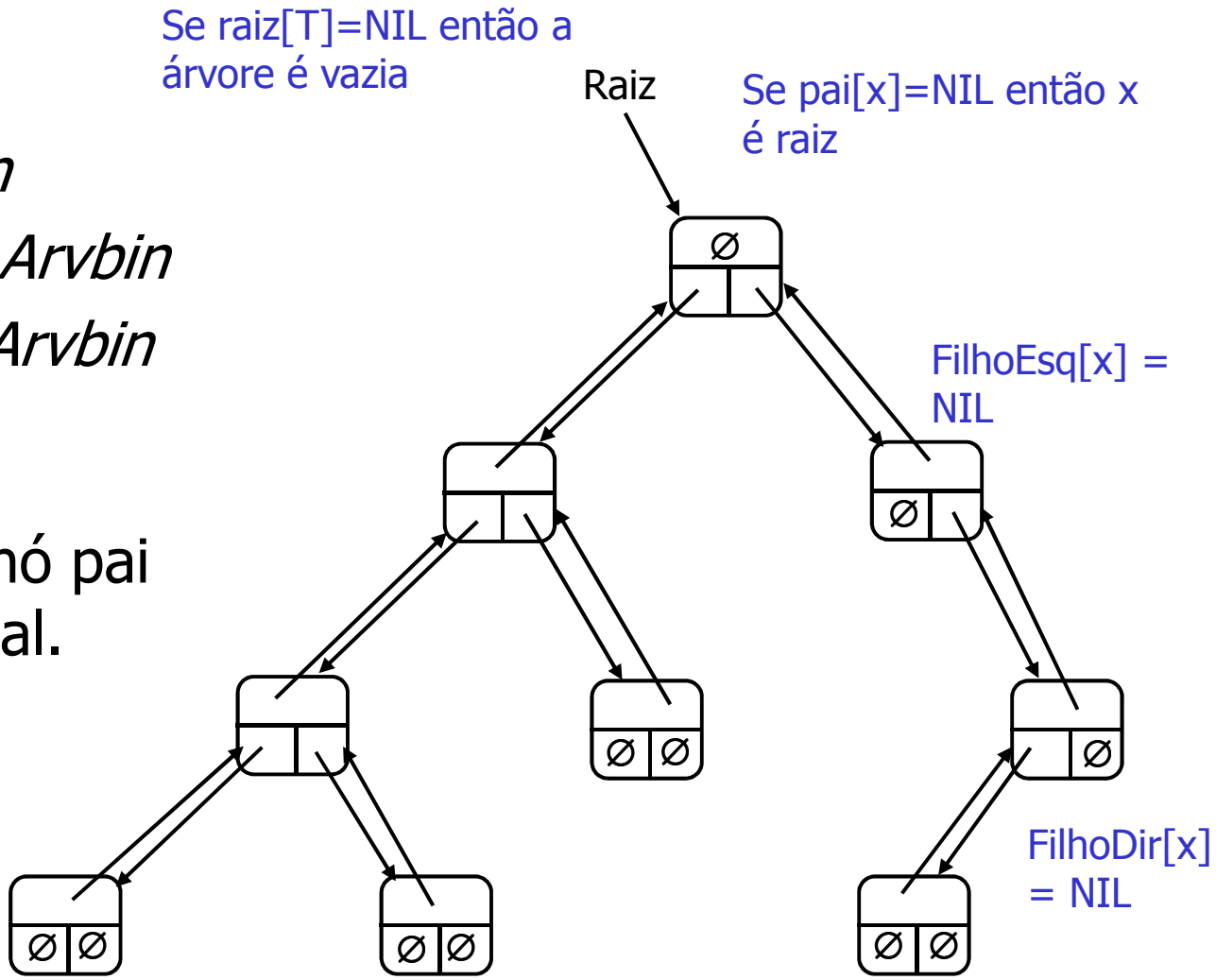
- Árvores cujo grau dos nodos internos é no máximo 2

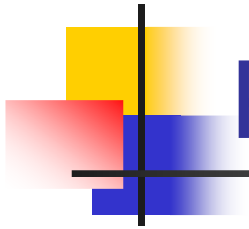


Representação Física

- *ArvBin* :
  - **Pai:** *ArvBin*
  - **FilhoEsq:** *Arvbin*
  - **FilhoDir:** *Arvbin*

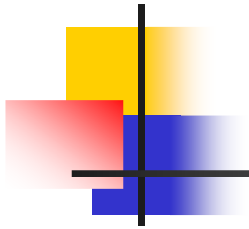
- Ligação entre nó pai e filho bidirecional.
- Não há ciclos!





Representação em C

- Podemos definir um tipo para representar uma árvore binária
  - Vamos considerar que a informação a ser armazenada são valores de caracteres simples
- Que estrutura podemos usar para representar um nó da árvore?
  - Cada nó deve armazenar três informações: a informação propriamente dita, no caso um caractere, e dois ponteiros para as sub-árvores, à esquerda e à direita

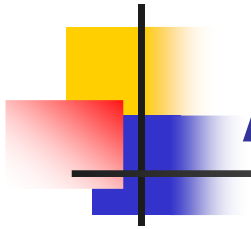


Representação em C

- Os campos *Pai*, *FilhoEsq* e *FilhoDir* são representados por **p**, **left** e **right** resp.

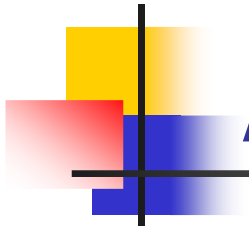
```
typedef struct node {
  char key;
  struct node *p;
  struct node *left;
  struct node *right;
} mynode;
```

A estrutura da árvore como um todo é representada por um ponteiro para o nó raiz (*mynode *root*)



Árvores Binárias: Operações

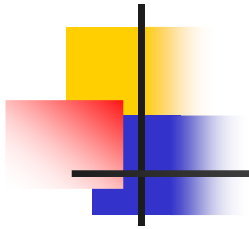
- As árvores binárias admitem operações de conjuntos dinâmicos
  - Search, Minimum, Maximum, Predecessor, Successor, Insert e Delete
- Pode ser usada como um dicionário e também como uma fila de prioridades (fp)
  - fp é uma estrutura de dados para manutenção de um conjunto S de elementos, cada qual com um valor associado chamado **chave** com operações de *insert, maximum, extract-max, increase-key* (exemplo: programar trabalhos em um computador)



Árvores Binárias: Propriedades

- Seja x um nó em uma árvore de pesquisa binária.
  - Se y é um nó na subárvore da esquerda de x, então chave[y] ≤ chave[x]
  - Se y é um nó na subárvore da direita de x, então chave[x] ≤ chave[y]
- Esta propriedade nos permite imprimir todas as chaves em sequência ordenada através de um algoritmo recursivo (**percurso de árvore em ordem**)

Percurso de Árvore em Ordem



- A chave da raiz de uma subárvore é impressa entre os valores de sua subárvore esquerda e aqueles da sua subárvore a direta

```
inorder-tree-walk(x)
if (x!=NIL)
  then inorder-tree-walk(esquerda[x])
    print chave[x]
    inorder-tree-walk(direita[x])
```

Imprime chave

Imprime filho da esquerda

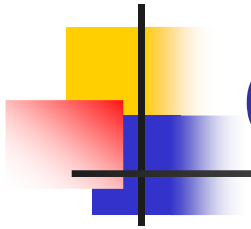
Imprime filho da direita

- Como funciona o percurso de árvore pré-ordem e pós-ordem?

Criando uma árvore binária

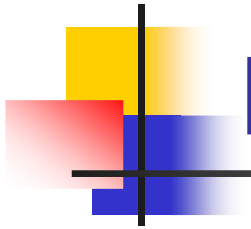
- Para inserir um novo valor **v** em uma árvore binária **T**, utilizamos o seguinte procedimento:

```
TREE-INSERT(T, z)   chave[z]=v, esquerda[z]=NIL, direita[z]=NIL
 1  y ← NIL
 2  x ← raiz[T]
 3  while x ≠ NIL        o ponteiro x traça o caminho
 4     do y ← x          o ponteiro y é mantido como pai
 5        if chave[z] < chave[x]
 6           then x ← esquerda[x]
 7           else x ← direita[x]
 8  p[z] ← y
 9  if y = NIL
10     then raiz[T] ← z          ▷ A árvore T era vazia
11     else if chave[z] < chave[y]
12             then esquerda[y] ← z
13             else direita[y] ← z
```



Criando uma árvore binária em C

```
int insert(mynode *z, char key) {
  mynode *y, *x;
  z = (mynode*) malloc(sizeof(mynode));
  z->key=key; z->left=NULL; z->right=NULL; y = NULL; x=root;
  while (x != NULL) {
    y = x;
    if (z->key < x->key) x = x->left;
    else x = x->right;
  }
  z->p = y;
  if (y==NULL) {root = z; root->left=NULL; root->right=NULL;}
  else if (z->key < y->key) y->left = z;
  else if (z->key > y->key) y->right = z;
  return 0;
}
```



Exercício 1

- Uma árvore binária de pesquisa tem 10 nós. Os nós foram inseridos na seguinte ordem: F C E L G A B I H J
  - Desenhe a respectiva árvore
  - Faça o percurso em ordem da árvore
  - Faça o percurso em pré-ordem da árvore
  - Faça o percurso em pós-ordem da árvore

Árvore Binária: Como pesquisar

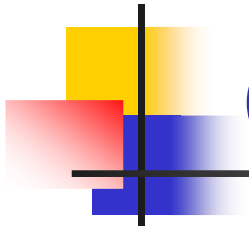
- Procura por um nó **x** com uma determinada chave **k**:

```
TREE-SEARCH(x, k)
1  if x = NIL or k = chave[x]
2     then return x
3  if k < chave[x]
4     then return TREE-SEARCH(esquerda[x], k)
5     else return TREE-SEARCH(direita[x], k)
```

x é um ponteiro para o nó da árvore

retorna um ponteiro para um nó com chave k

Caso o método *tree-search* não encontre o nó, o que ele retorna?



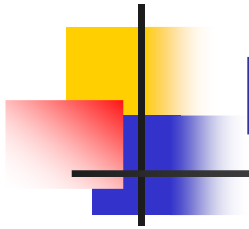
Como pesquisar (em C)

- Qual é o tempo de execução do *tree_search*?

  O(h), onde h é a altura da árvore

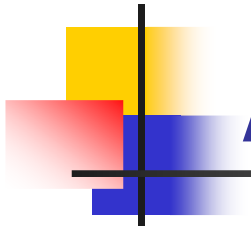
```
mynode* tree_search(mynode *x, int k) {
  if (x == NULL || k == x->key)
    return x;
  if (k < x->key)
    tree_search(x->left, k);
  else
    tree_search(x->right,k);
}
```

Os nós encontrados durante a recursão formam um caminho descendente a partir da raiz



Exercício 2

- Escreva um método para pesquisar em uma árvore binária de forma iterativa



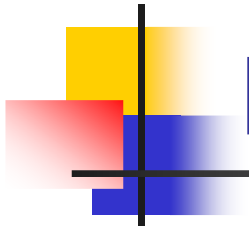
Ávore Binária: Máximo e Mínimo

- O mínimo é obtido seguindo os ponteiros filhos da esquerda desde a raiz até encontrar NIL

```
tree_minimum(x)
while esquerda[x] != NIL
  do x=esquerda[x]
return x
```

- Como encontrar o valor máximo?

```
tree_maximum(x)
while direita[x] != NIL
  do x=direita[x]
return x
```



Exercício 3

- Escreva funções recursivas dos procedimentos *tree_minimum* e *tree_maximum*

Ávore Binária: Sucessor (1)

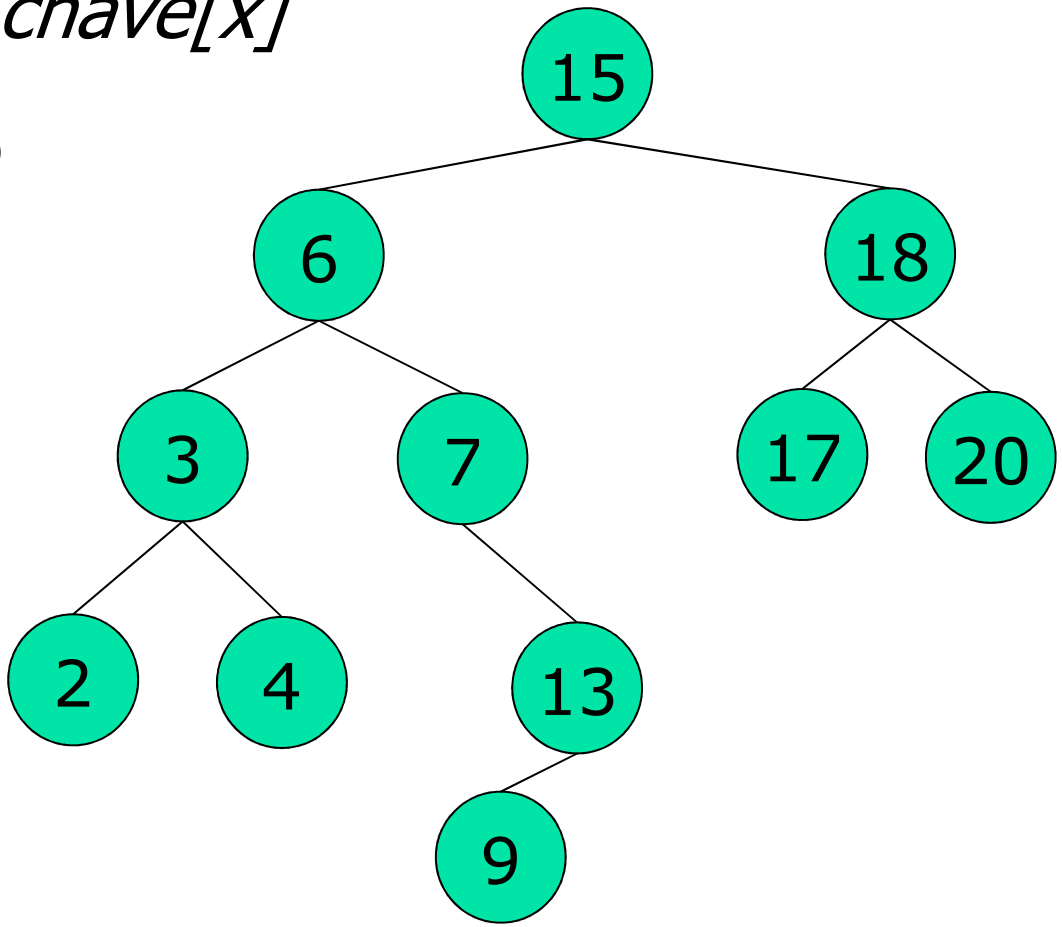
- O sucessor de um nó *x* é o nó com a menor chave maior que *chave[x]*

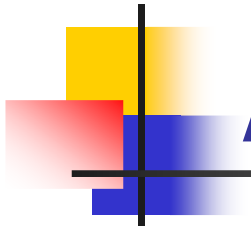
Qual é o sucessor do nó com chave 15?

17

Qual é o sucessor do nó com chave 13?

15



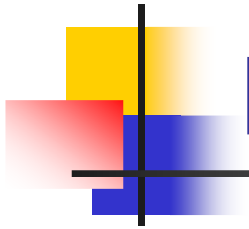


Ávore Binária: Sucessor (2)

- O procedimento retorna o sucessor de um nó x ou NIL se x tem a maior chave da árvore

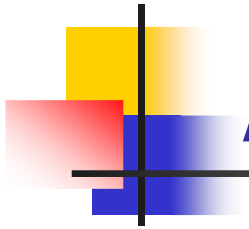
```
tree_successor(x)
  if direita[x] != NIL
    then return tree_minimum(direita[x])
  y=p[x]
  while y!=NIL e x=direita[y]
    do x=y
      y=p[y]
return y
```

- O método *tree_predecessor* é simétrico e é executado no tempo O(h)



Exercício 4

- Escreva o procedimento *tree_predecessor*



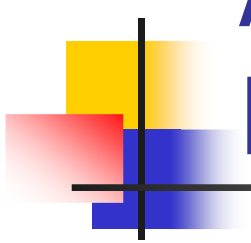
Ávore Binária: Eliminação (1)

- O procedimento toma como argumento um ponteiro para *z*:
  - Se *z* não tem nenhum filho, modificamos seu pai *p[z]* para substituir *z* por *NIL* como seu filho
  - Se o nó tem apenas um único filho, extraímos *z*, criando um novo vínculo entre seu filho e seu pai
  - Se o nó tem dois filhos, extraímos *y*, o sucessor de *z*, que não tem nenhum filho da esquerda e substituímos a chave e os dados satélites de *z* pela chave e pelos dados satélite de *y*

Ávore Binária: Eliminação (2)

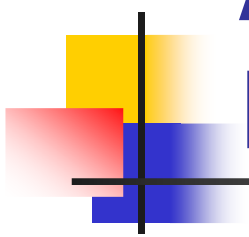
```
TREE-DELETE(T, z)
 1  if esquerda[z] = NIL or direita[z] = NIL
 2     then y ← z    // z tem no máximo um filho
 3     else y ← TREE-SUCCESSOR(z)  // z tem dois filhos
 4  if esquerda[y] ≠ NIL
 5     then x ← esquerda[y]
 6     else x ← direita[y]        // x é o filho não NIL de z
 7  if x ≠ NIL
 8  then p[x] ← p[y]
 9  if p[y] = NIL
10     then raiz[T] ← x
11     else if y = esquerda[p[y]]
12             then esquerda[p[y]] ← x
13             else direita[p[y]] ← x
14   if y ≠ z
15     then chave[z] ← chave[y]
16             copiar dados satélite de y em z
17   return y
```

Árvores Enraizadas com Ramificações Limitadas



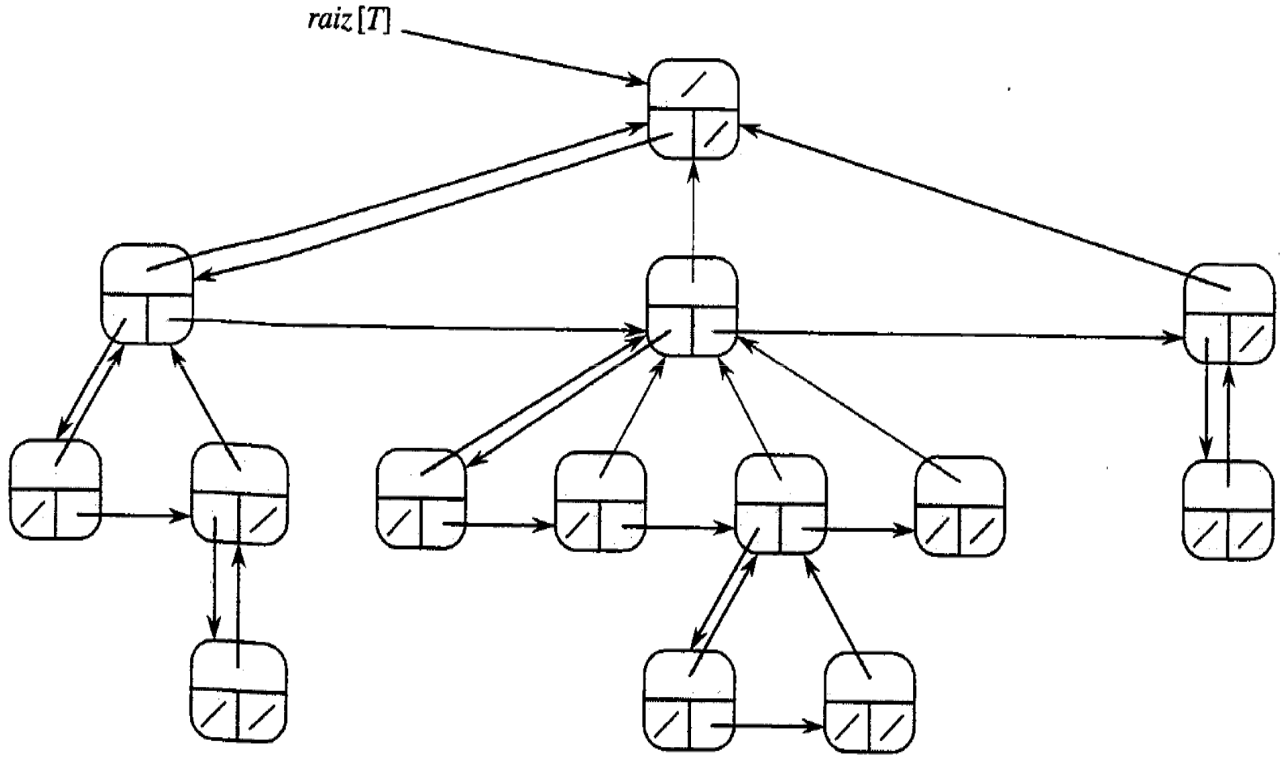
- Árvore binária pode ser estendida a qualquer classe de árvores
  - O número de filhos de cada nó é no máximo alguma constante *k*
  - Substituimos os campos *esquerda* e *direita* por $filho_1$, $filho_2$,..., $filho_k$
- Devemos alocar os campos antecipadamente
  - Podemos desperdiçar uma grande quantidade de memória se não soubermos o número de filhos

Árvores Enraizadas com Ramificações Ilimitadas (1)



- Cada nó x tem campos *p[x]*, *filho da esquerda[x]* e *irmão da direita[x]*
  - *Filho da esquerda[x]* aponta para o filho da extremidade esquerda do nó *x*
  - *Irmão da direita[x]* aponta para o irmão de *x* situado imediatamente a direita
- Se o nó *x* não tem nenhum filho, então o filho da *esquerda[x]=NIL*
- Se o nó *x* é o filho da extremidade direita de seu pai, então irmão direita[x]=NIL

Árvores Enraizadas com Ramificações Ilimitadas (2)



Cada nó *x* tem campos *p[x]* (superior), *filho da esquerda[x]* (inferior esquerdo) e *irmão da direita[x]* (inferior direito)