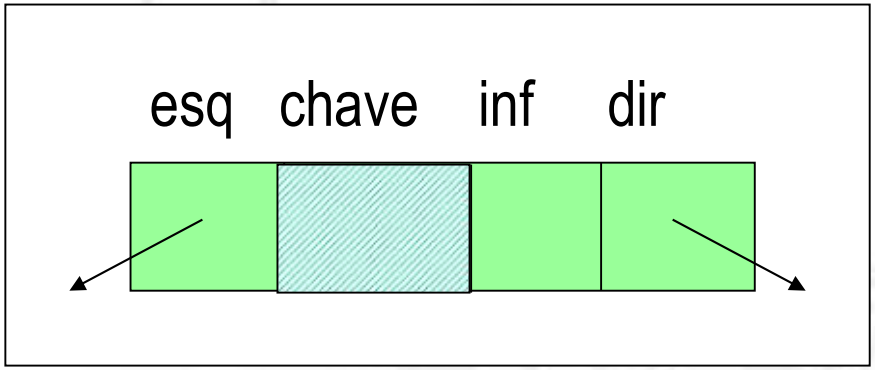
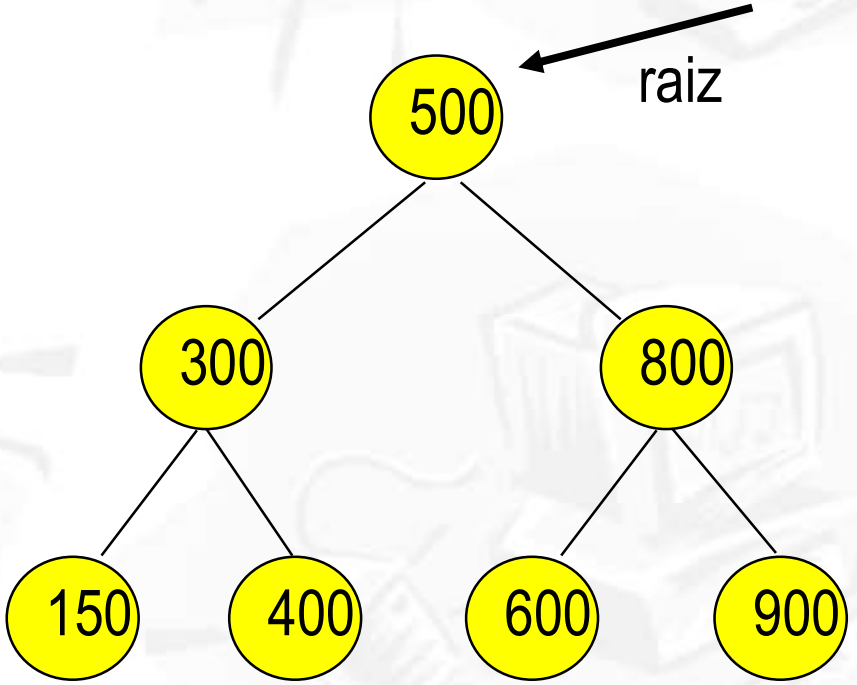
Árvores Binárias de Busca

Definição

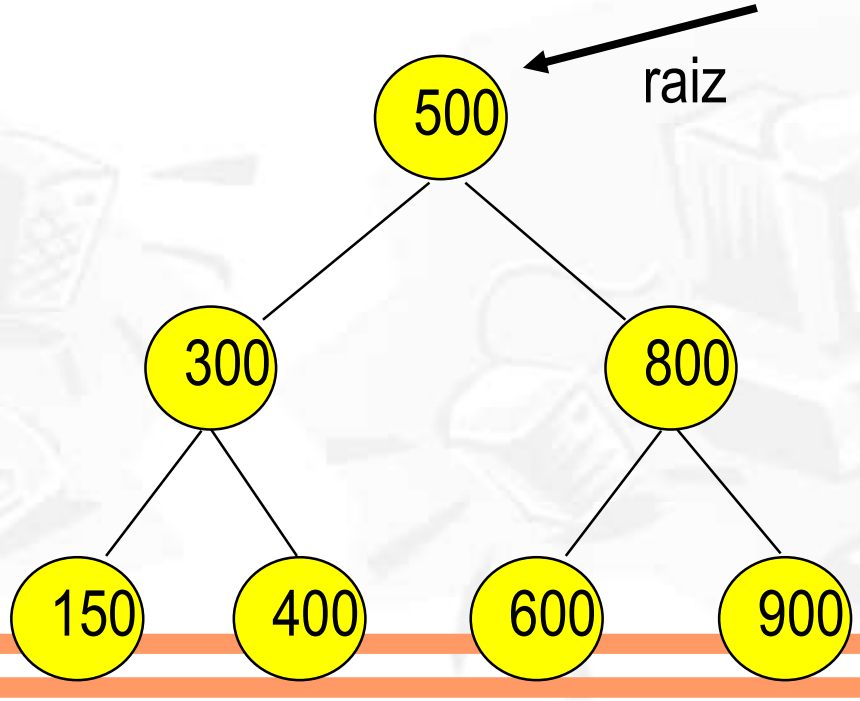
- Uma árvore binária de busca, além da relação hierárquica entre os nós, possuem uma **ordem** entre os nós filhos.
- Ordem é definida pelo campo **chave**





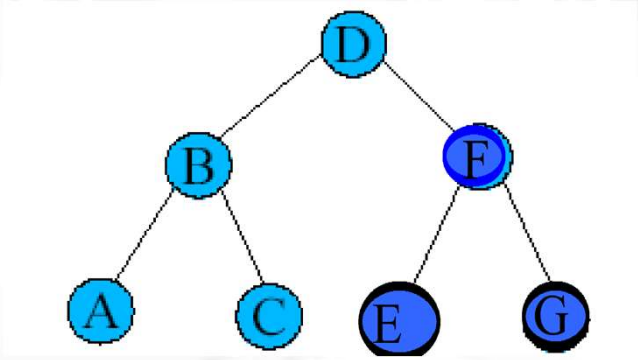
Definição

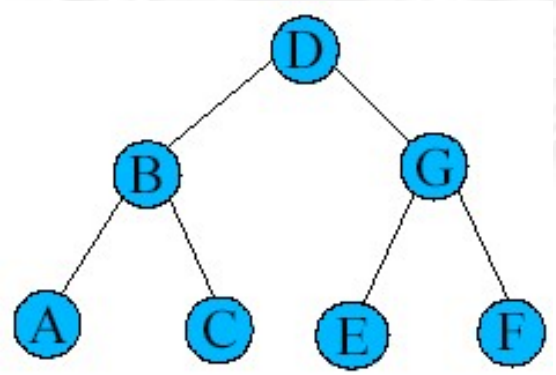
- Uma árvore binária busca, cuja raiz armazena o elemento **R**, é denominada **árvore binária de pesquisa (busca)** se:
  - ✓ todo elemento armazenado na sub - árvore à esquerda é menor **R;**
  - ✓ todo elemento armazenado na sub- árvore à direita é maior do que **R;**
  - ✓ a sub- árvore direita e esquerda também são ABB.



Definição

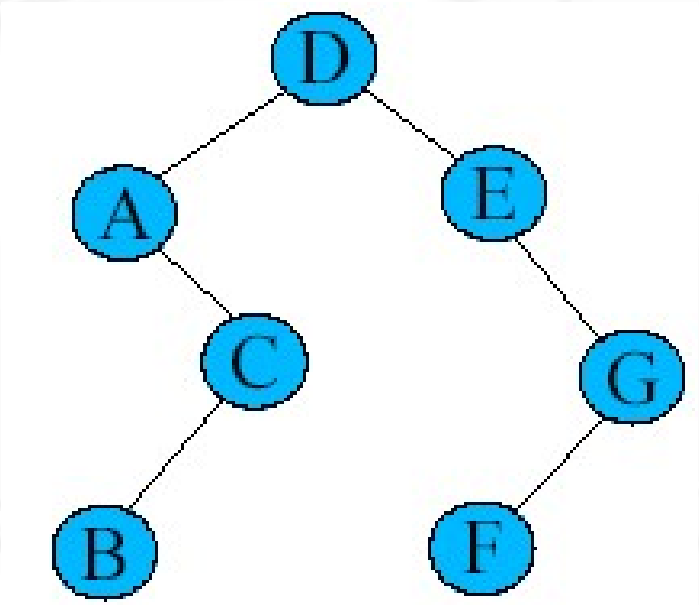
- Exemplo de árvore binária de busca e de árvore binária que não é de busca





Definição

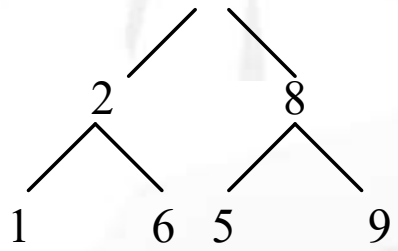
- A árvore abaixo é ABB?



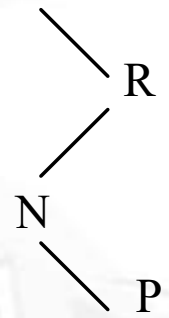
Exemplo

- Quais das árvores a seguir são ABB?

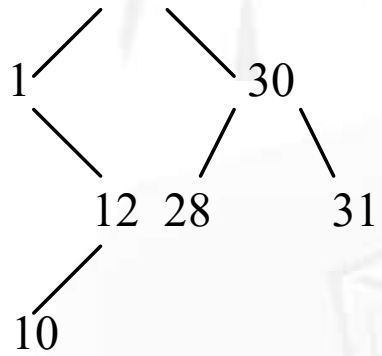
a)



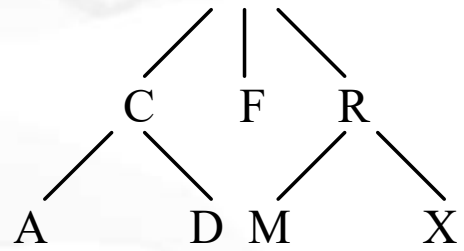
b)



c)



d)



Definição

```
typedef struct arv {
   int info;
   struct arv* esq;
   struct arv*dir;
}TArv;

typedef TArv *PArv;
```

Operações

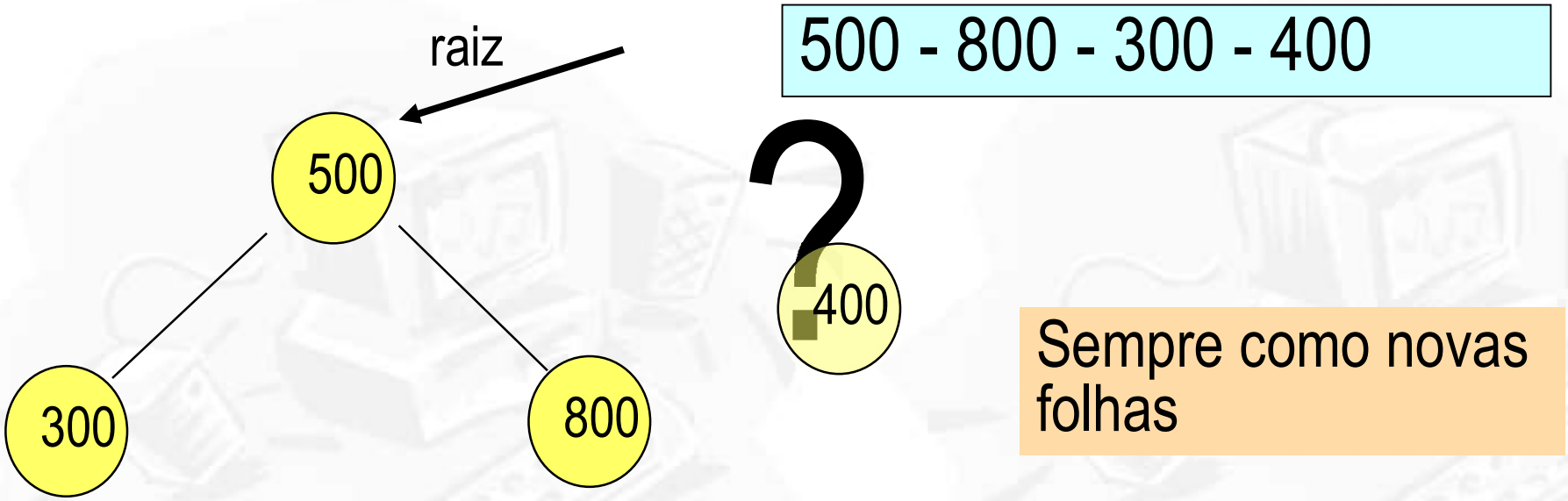
Inserir novo nó

Pesquisar nó

Remover nó

Inserção

- se a árvore for vazia, instala o novo nó na raiz
- se não for vazia, compara a chave com a chave da raiz:
  - se for menor, instala na sub-árvore da esquerda
  - caso contrário, instala na sub-árvore da direita



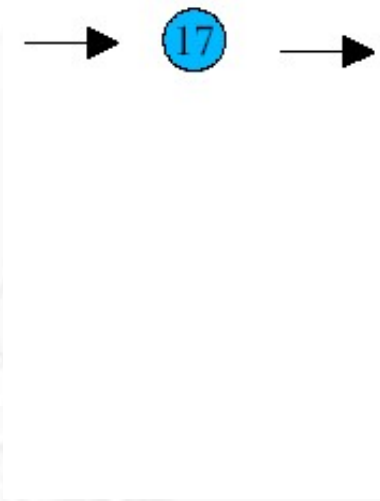
500 - 800 - 300 - 400

Sempre como novas folhas

Exemplo

- Insira os seguintes números numa ABB:

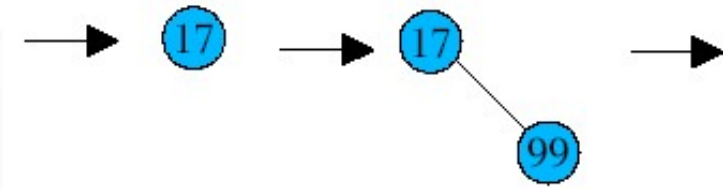
17, 99, 13, 1, 3, 100, 400



Exemplo

- Insira os seguintes números numa ABB:

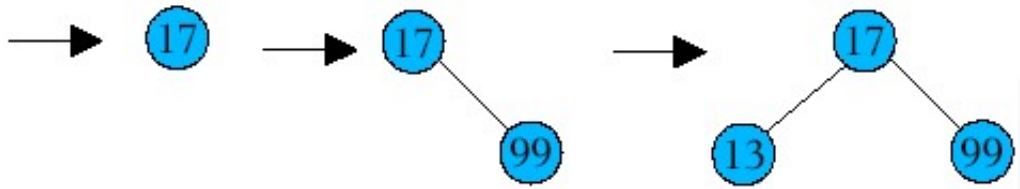
17, 99, 13, 1, 3, 100, 400



Exemplo

- Insira os seguintes números numa ABB:

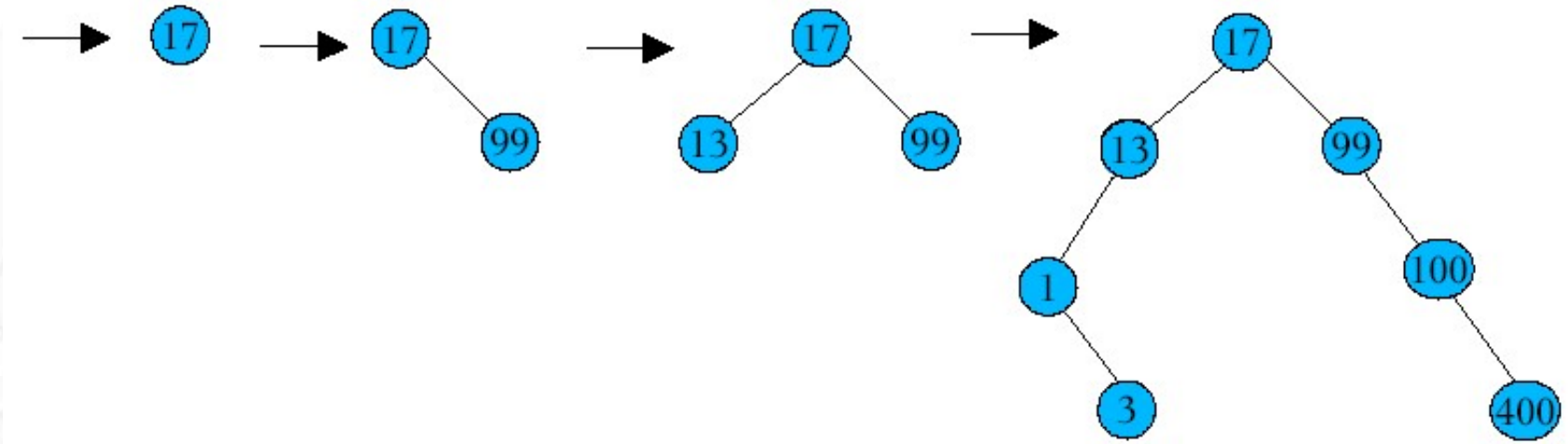
17, 99, 13, 1, 3, 100, 400



Exemplo

- Insira os seguintes números numa ABB:

17, 99, 13, 1, 3, 100, 400



Inserção

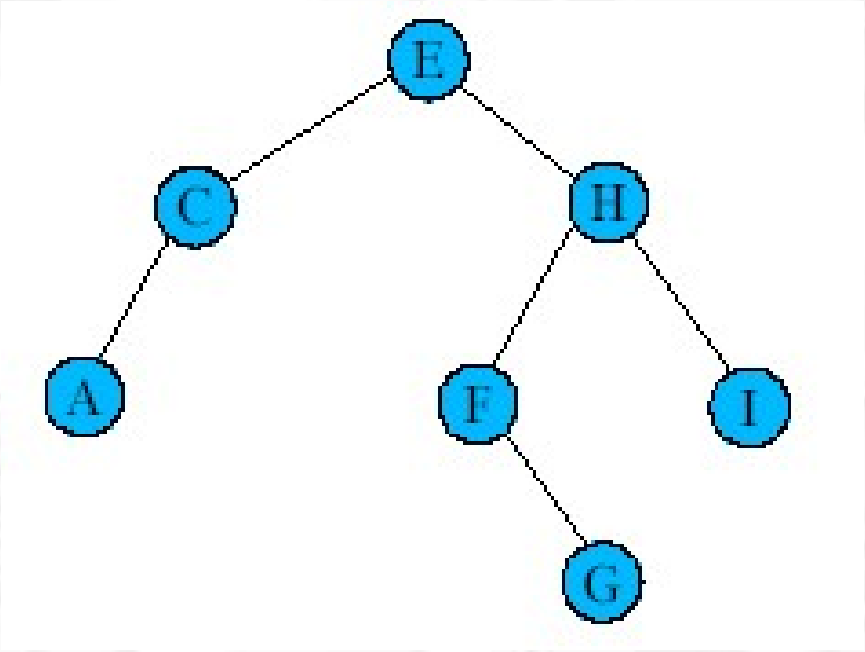
```
Arv insereABB (PArv a, int c) {

if (a == NULL) {

  PArv novo=(PArv)malloc(sizeof(TArv));
  novo->esq = NULL;
  novo->dir = NULL;
  a = novo;
}
else if (c < a->info)
  a->esq = insereABB(a->esq,c);
else
  a->dir = insereABB(a->dir,c);
return(a);
}
```

Exemplo

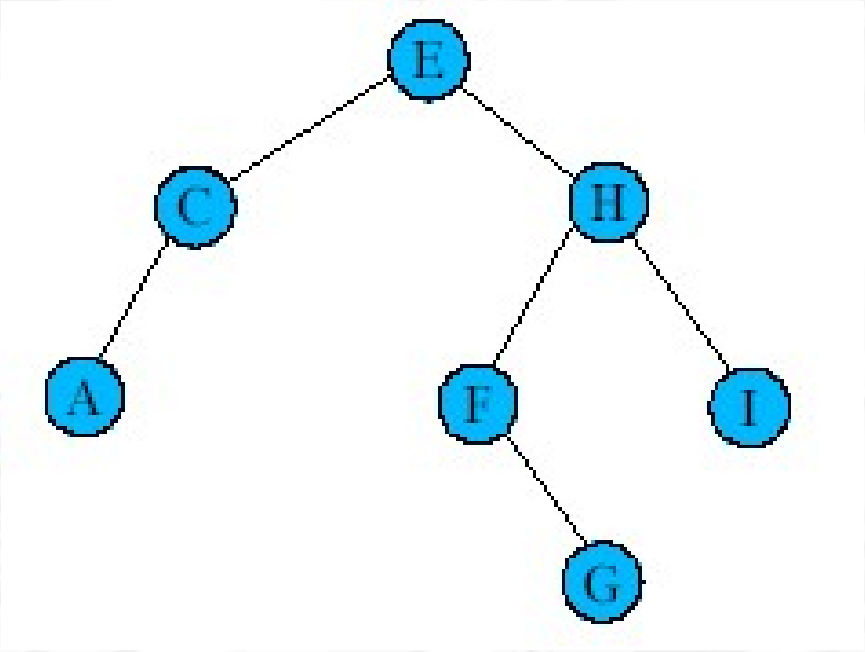
- Como será uma possível sequência de chamada da função de inserção para que seja possível construir a árvore a seguir?



Exemplo

- Como será uma possível sequência de chamada da função de inserção para que seja possível construir a árvore a seguir?

```
PArv a=NULL;
a = insereABB(a,'E');
a = insereABB(a,'C');
a = insereABB(a,'A');
a = insereABB(a,'H');
a = insereABB(a,'F');
a = insereABB(a,'I');
a = insereABB(a,'G');
```



Pesquisa

- Se a árvore for nula, nada a fazer, caso contrário, o processo de pesquisa é o mesmo utilizado para inserção.

Pesquisa

```
PArv buscaABB (PArv a, char c) {

if (a==NULL)
   return NULL;  /*árvore vazia*/
else if(c < a->info)
        return(buscaABB(a->esq,c));
else if (c > a->info)
   return(buscaABB(a->dir,c));
else     return a;

}
```

Caminhamentos

Prefixado à esquerda

visita raiz
percorre sub-árvore esquerda
percorre sub-árvore direita

8 6 4 7 10 11 12

Central à esquerda (infixa)

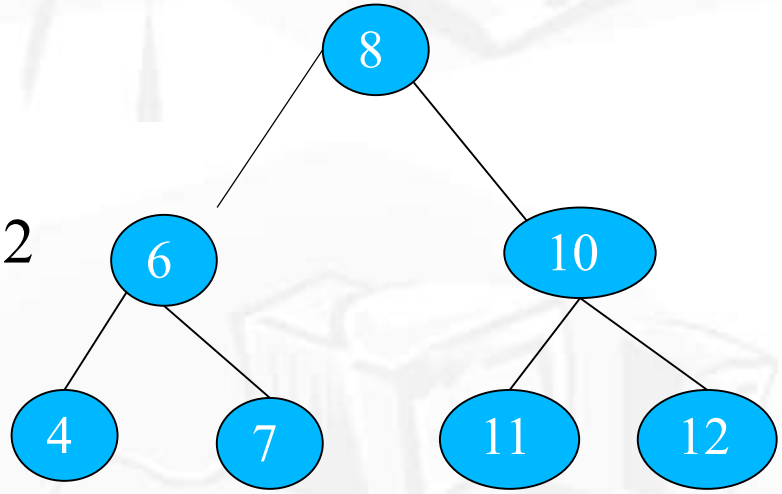
percorre sub-árvore esquerda
visita raiz
percorre sub-árvore direita

4 6 7 8 11 10 12

Pósfixado à esquerda

percorre sub-árvore esquerda
percorre sub-árvore direita
visita raiz

4 7 6 11 12 10 8



Caminhamentos

Prefixado à direita

visita raiz
percorre sub-árvore direita
percorre sub-árvore esquerda

8 10 12 11 6 7 4

Central à direita (infixa)

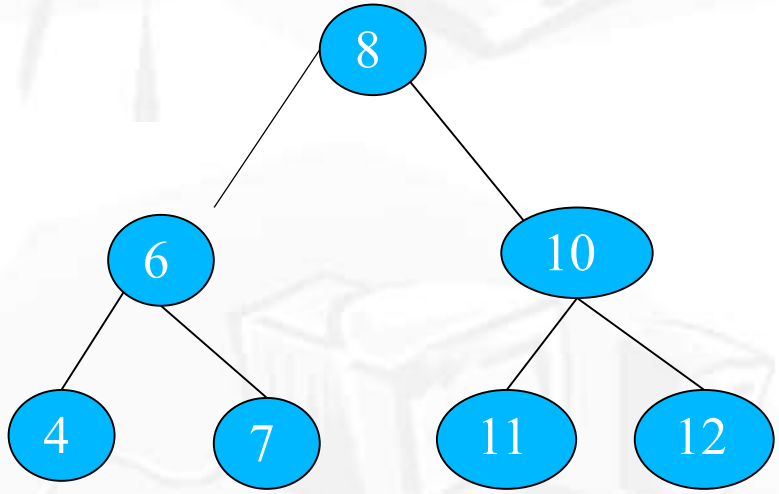
percorre sub-árvore direita
visita raiz
percorre sub-árvore esquerda

12 10 11 8 7 6 4

Pósfixado à direita

percorre sub-árvore direita
percorre sub-árvore esquerda
visita raiz

12 11 10 7 4 6 8



Exercício

- Dadas duas árvores binárias A e B faça uma função que verifique se duas árvores A e B são iguais. Protótipo da função:

```
int iguais(PArv A, PArv B)
```

- Esta função retorna 1 se as árvores recebidas como parâmetros forem iguais e zero caso contrário.

```
int iguais(PArv a, PArv b){
    if (a==NULL && b==NULL)   //as duas árvores são NULL
        return 1;
    if (a==NULL || b==NULL) //somente uma das árvores é NULL
        return 0;
    return((a->info==b->info) && iguais(a->esq,b->esq)
  && iguais(a->dir,b->dir));
}
```

Exercícios

- Escreva uma função que obtenha o menor valor da árvore.

int menor (PArv a)

```
int menor_no(PArv a){
    PArv p;
    for(p=a;p->esq!=NULL;p=p->esq);
    return(p->info);
}
```

Escreva uma função que imprima todas as chaves de uma árvore binária de forma ordenada (do menor para o maior).

```
void imprime_crescente(PArv a) {
   if (a!=NULL)
      {
         imprime_crescente(a->esq); /* mostra sae */
         printf("%d ", a->info); /* mostra raiz */
         imprime_crescente(a->dir); /* mostra sad */
      }
}
```

Imprimir em ordem crescente eh o mesno que imprimir em ordem infixa (esq, raiz, dir)

Exercícios

Escreva uma função que imprime todos os valores nos nós da árvore a que sejam menores que x, em ordem crescente

void showmenor (PArv a, int x);.

```
void menor_que_x(PArv a, int val) {
     if (a==NULL)
        return;
     menor_que_x(a->esq,val);
     if (a->info<val)
        printf("%d ",a->info);
     if (val>a->info)
       menor_que_x(a->dir,val);
}
```

Remoção

- Se o nó a ser retirado de uma árvore for uma folha, basta atualizar o link do seu pai para que não aponte mais para o nó a ser retirado.
- No caso de retirada de um nó que é raiz, deve-se adotar o seguinte procedimento:
  - ✓ a raiz não possui filhos: a solução é trivial;
  - ✓ a raiz possui um único filho: podemos remover o nó raiz, substituindo-o pelo seu nó filho;
  - ✓ a raiz possui dois filhos: não é possível que os dois filhos assumam o lugar do pai.
    - escolhemos o nó que armazena o maior elemento na sub-árvore esquerda;
    - este nó será removido e o elemento armazenado por ele entrará na raiz a ser removida.

Remoção

- São quatro as possibilidades existentes para retirada.
- Considere que o nó a ser retirado pode ser:
  - ✓ Um nó sem descendentes
  - ✓ Um nó sem descendentes na direita mas com descendentes na esquerda.
  - ✓ Um nó sem descendentes na esquerda mas com descendentes na direita.
  - ✓ Um nó com 2 filhos.

Remoção

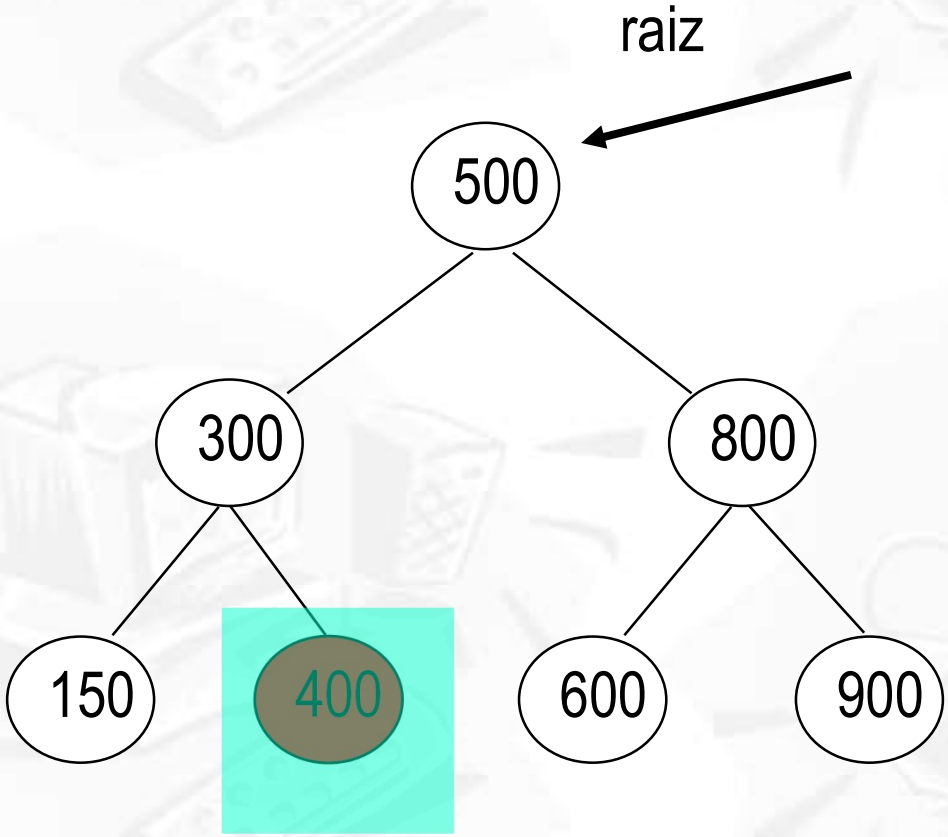
- No programa, isso pode ser representado por 4 testes de seleção (if):

```
if (a-> esq != NULL) && (a-> dir ==NULL) {
//somente descendentes aa esq.
    ... (1 linha)
}
else if (a-> esq == NULL) && (a-> dir !=NULL) {
//somente descendentes aa dir.
    ... (1 linha)
}
else if (a-> esq == NULL) && (a-> dir ==NULL) {
//no sem descendentes.
    ... (1 linha)
}
else {
//no com 2 filhos
    ... (algumas linhas)
}
```

Remoção

nó é uma folha

Excluir 400

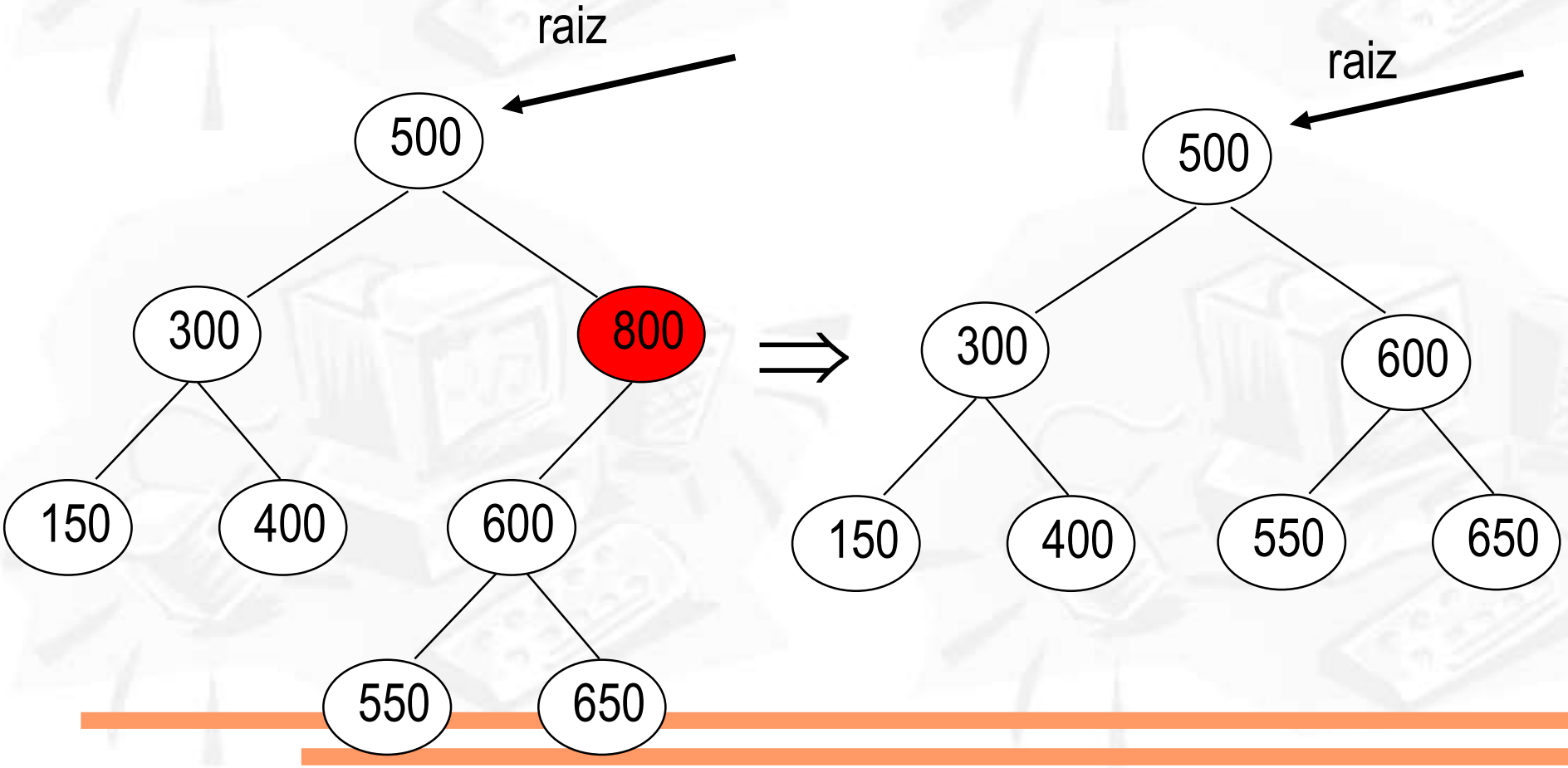


Remoção

o nó tem somente 1 sub-árvore

Excluir 800

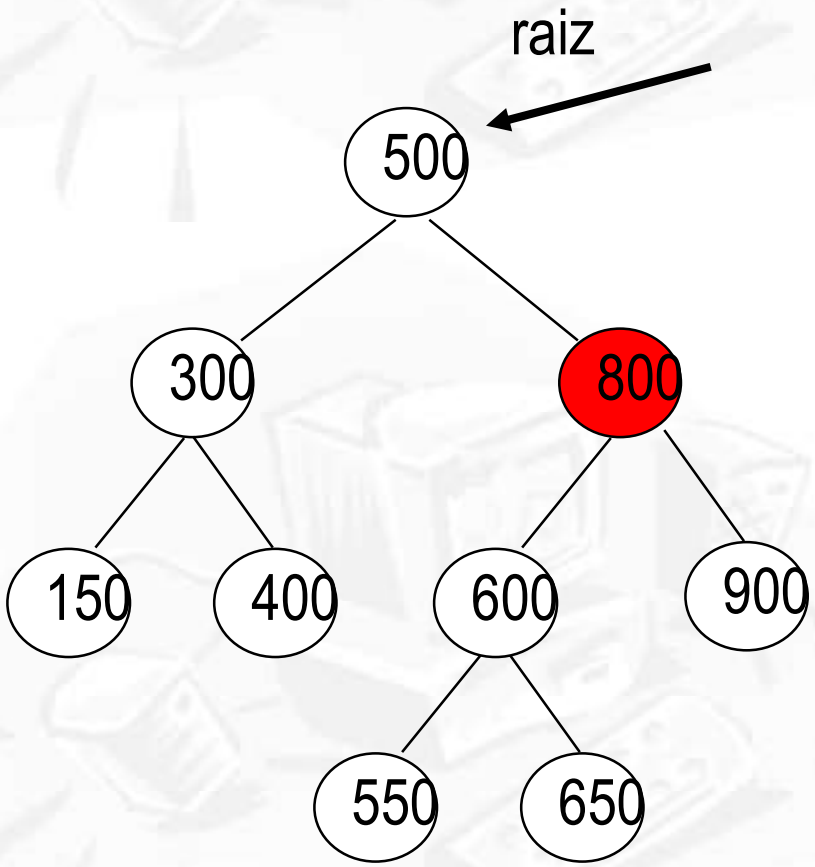
→ raiz da subárvore passa a ocupar o lugar do nodo excluído



Remoção

quando o nó tem 2 sub-árvores

→ reestruturar a árvore



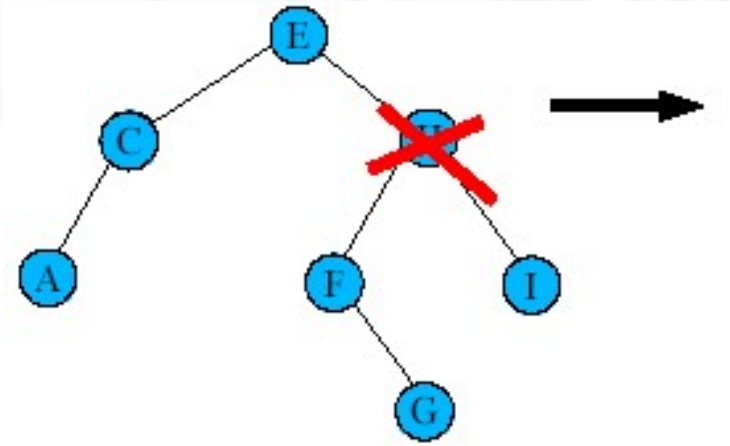
Trocar o valor do nodo a ser removido com:

- valor do nodo que tenha a maior chave da sua subárvore a esquerda

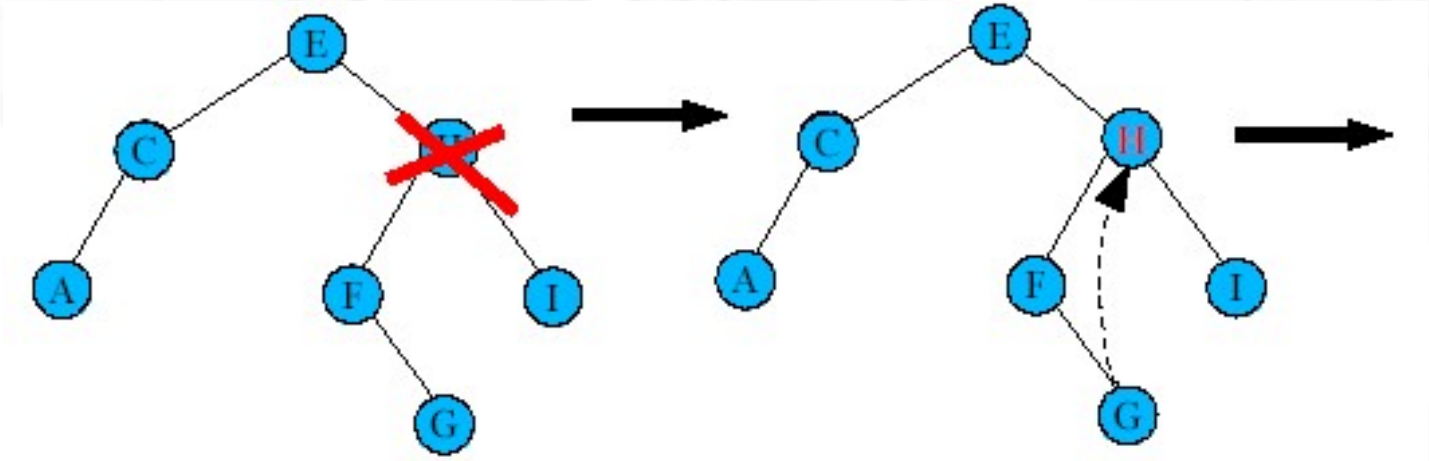
OU

- valor do nodo que tenha a menor chave da sua subárvore a direita

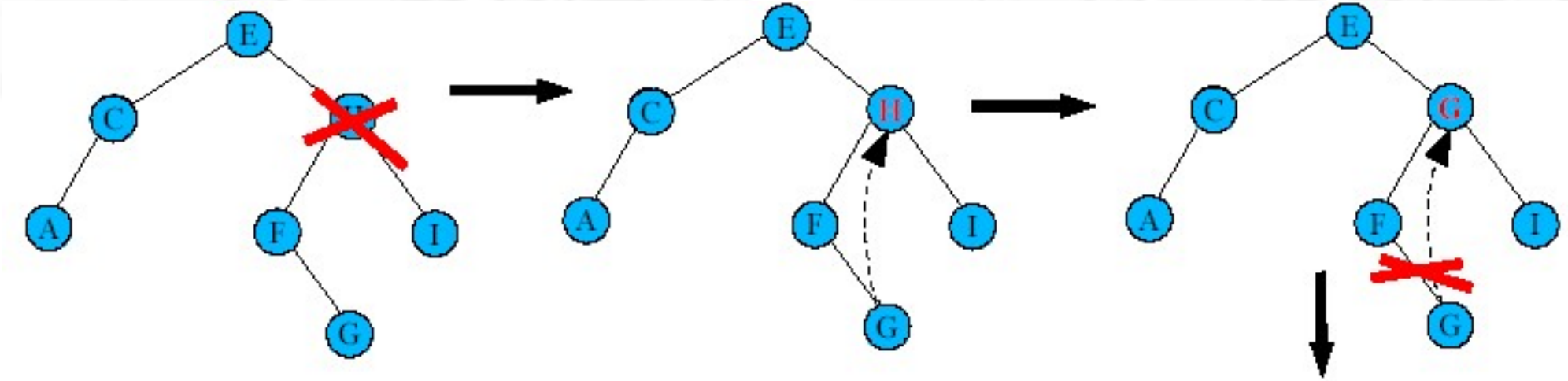
Remoção



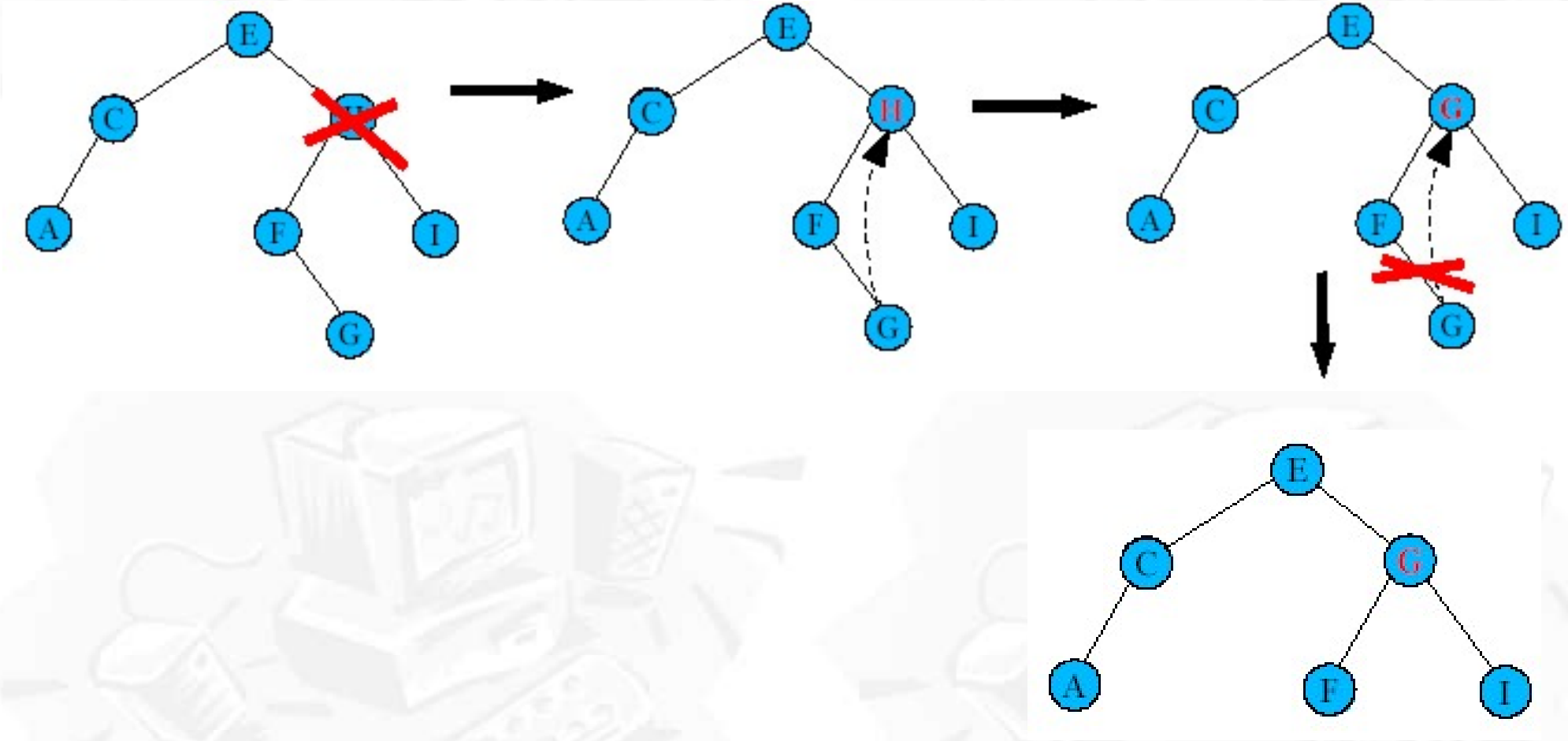
Remoção



Remoção

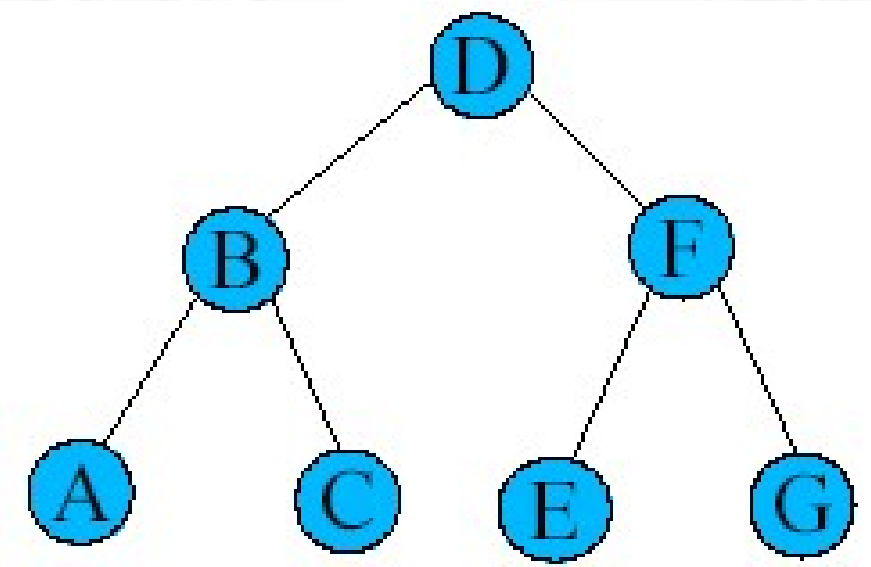


Remoção



Exercício

- Remover o nó D



Nó com 2 filhos

Remoção

```
PArv retira (PArv r, int v) {
   if (r == NULL)
      return NULL; //nao encontrou a
   chave
   else if (r->info > v)
         r->esq = retira(r->esq, v);
   else if (r->info < v)
          r->dir = retira(r->dir, v);
   else { // achou o elemento
          if (r->esq == NULL &&
                r->dir == NULL) {
             // elemento sem filhos
              free (r);
              r = NULL;
           }
          else if (r->esq == NULL) {
            // só tem filho à direita
              PArv t = r;
              r = r->dir;
              free (t);
           } else if (r->dir == NULL) {
              //só tem filho à
   esquerda
                PArv t = r;
                r = r->esq;
                free (t);
           }
else {// tem os dois filhos

        PArv pai = r;
        PArv f = r->esq;
        while (f->dir != NULL) {
           //busca o maior aa esquerda
           pai = f;
          f = f->dir;
        }


  // troca as informações
           r->info = f->info;
           if (pai==r) //quando o pai do maior
valor é o próprio nó raiz que tem que ser
removido
              pai->esq = f->esq;
           else
               pai->dir = f->esq;
                free(f);
             }
    }
    return r;
}
```

FIM