

Estruturas de Dados

Módulo 13 - Árvores



Referências

Waldemar Celes, Renato Cerqueira, José Lucas Rangel,
Introdução a Estruturas de Dados, Editora Campus
(2004)

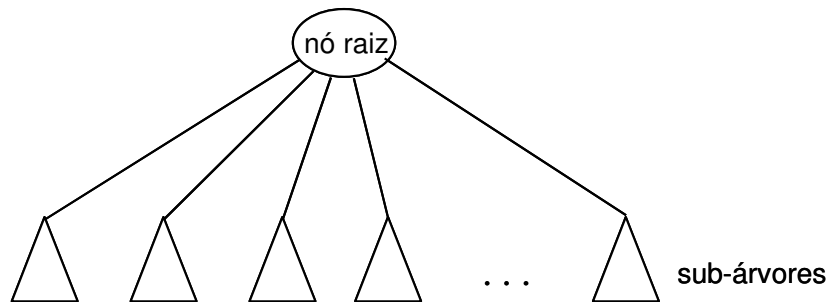
Capítulo 13 – Árvores

Tópicos

- Introdução
- Árvores binárias
 - Representação em C
 - Ordens de percurso em árvores binárias

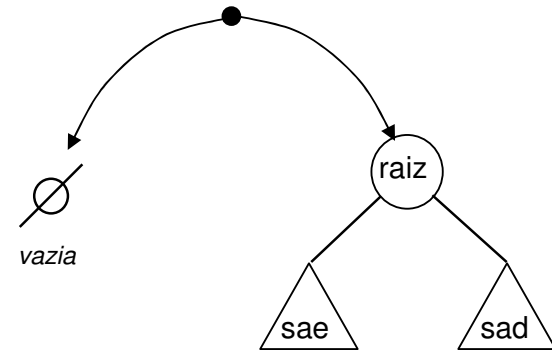
Introdução

- Árvore
 - um conjunto de nós tal que
 - existe um nó r , denominado *raiz*, com zero ou mais sub-árvores, cujas raízes estão ligadas a r
 - os nós raízes destas sub-árvores são os *filhos* de r
 - os *nós internos* da árvore são os nós com filhos
 - as *folhas* ou *nós externos* da árvore são os nós sem filhos



Árvores binárias

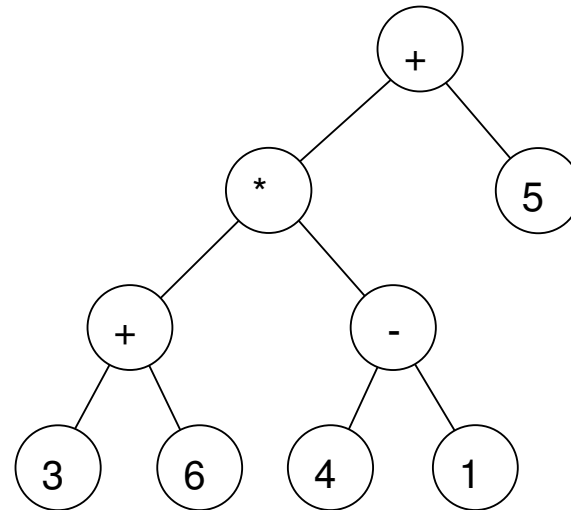
- Árvore binária
 - um árvore em que cada nó tem zero, um ou dois filhos
 - uma árvore binária é:
 - uma árvore vazia; ou
 - um nó raiz com duas sub-árvores:
 - a sub-árvore da direita (sad)
 - a sub-árvore da esquerda (sae)



Árvores binárias

Exemplo

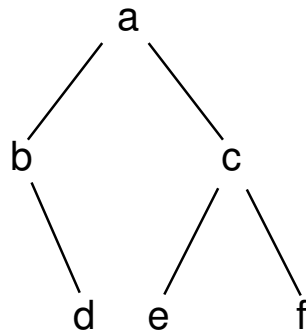
- árvores binárias representando expressões aritméticas:
 - nós folhas representam operandos
 - nós internos operadores
 - exemplo: $(3+6)*(4-1)+5$



Árvores binárias

- Notação textual:
 - a árvore vazia é representada por `<>`
 - árvores não vazias por `<raiz sae sad>`
 - exemplo:

`<a <b <> <d<><>> > <c <e<><>> <f<><>>> > >`



Árvores binárias - Implementação em C

- Representação de uma árvore:
 - através de um ponteiro para o nó raiz
- Representação de um nó da árvore:
 - estrutura em C contendo
 - a informação propriamente dita (exemplo: um caractere)
 - dois ponteiros para as sub-árvores, à esquerda e à direita

```
struct arv {  
    char info;  
    struct arv* esq;  
    struct arv* dir;  
};
```


Árvores binárias - Implementação em C

- Interface do tipo abstrato Árvore Binária: [arv.h](#)

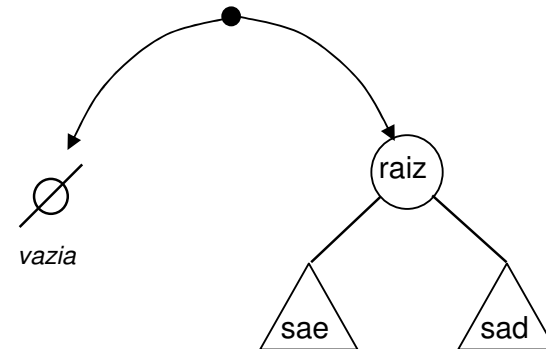
```
typedef struct arv Arv;  
  
Arv* arv_criavazia (void);  
Arv* arv_cria (char c, Arv* e, Arv* d);  
Arv* arv_libera (Arv* a);  
int  arv_vazia (Arv* a);  
int  arv_pertence (Arv* a, char c);  
void arv_imprime (Arv* a);
```

Árvores binárias - Implementação em C

- Implementação das funções:
 - implementação recursiva, em geral
 - usa a definição recursiva da estrutura

Uma árvore binária é:

- uma árvore vazia; ou
- um nó raiz com duas sub-árvores:
 - a sub-árvore da direita (sad)
 - a sub-árvore da esquerda (sae)



Árvores binárias - Implementação em C

- função `arv_criavazia`
 - cria uma árvore vazia

```
Arv* arv_criavazia (void)
{
    return NULL;
}
```

```

struct arv {
char info;
struct arv* esq;
struct arv* dir;
};
typedef struct arv Arv;

Arv* arv_criavazia (void);

```

```

#include <stdio.h>
#include <stdlib.h>
#include "arvore.h"

```

```

Arv* arv_criavazia (void)
{
return NULL;
}

```

```

#include <stdio.h>
#include <stdlib.h>
#include "arvore.h"

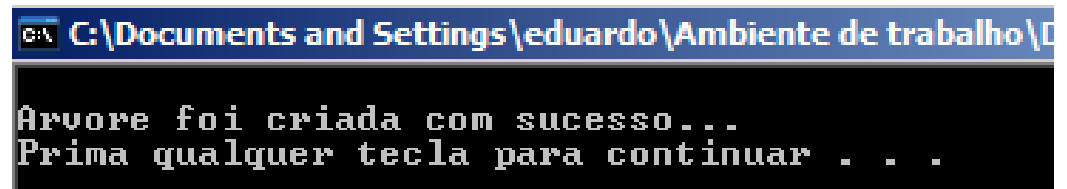
```

```

int main()
{

Arv* a1= arv_criavazia();
if (a1==NULL)
printf ("\nArvore foi criada com sucesso...\n");
system("PAUSE");
return 1;
}

```



```

C:\Documents and Settings\eduardo\Ambiente de trabalho\O
Arvore foi criada com sucesso...
Prima qualquer tecla para continuar . . .

```

Árvores binárias - Implementação em C

- função `arv_cria`
 - cria um nó raiz dadas a informação e as duas sub-árvores, a da esquerda e a da direita
 - retorna o endereço do nó raiz criado

```
Arv* arv_cria (char c, Arv* sae, Arv* sad)
{
    Arv* p=(Arv*)malloc(sizeof(Arv));
    p->info = c;
    p->esq = sae;
    p->dir = sad;
    return p;
}
```

```
struct arv {  
    char info;  
    struct arv* esq;  
    struct arv* dir;  
};  
typedef struct arv Arv;
```

```
Arv* arv_criavazia (void);  
Arv* arv_cria (char c, Arv* e, Arv* d);
```

```
#include <stdio.h>  
#include <stdlib.h>  
#include "arvore.h"
```

```
Arv* arv_criavazia (void)  
{  
    return NULL;  
}
```

```
Arv* arv_cria (char c, Arv* sae, Arv* sad)  
{  
    Arv* p=(Arv*)malloc(sizeof(Arv));  
    p->info = c;  
    p->esq = sae;  
    p->dir = sad;  
    return p;  
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include "arvore.h"

int main()
{

    /* sub-árvore 'd' */
    Arv* a1= arv_cria('d',arv_criavazia(),arv_criavazia());

    /* sub-árvore 'b' */
    Arv* a2= arv_cria('b',arv_criavazia(),a1);

    /* sub-árvore 'e' */
    Arv* a3= arv_cria('e',arv_criavazia(),arv_criavazia());

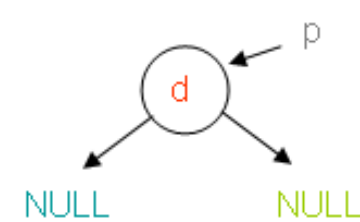
    /* sub-árvore 'f' */
    Arv* a4= arv_cria('f',arv_criavazia(),arv_criavazia());

    /* sub-árvore 'c' */
    Arv* a5= arv_cria('c',a3,a4);

    /* árvore 'a' */
    Arv* a = arv_cria('a',a2,a5 );

    system("PAUSE");
    return 1;
}
```

arv_cria ('d', NULL, NULL)



```
#include <stdio.h>
#include <stdlib.h>
#include "arvore.h"
```

```
int main()
{
```

```
    /* sub-árvore 'd' */
```

```
    Arv* a1= arv_cria('d',arv_criavazia(),arv_criavazia());
```

```
    /* sub-árvore 'b' */
```

```
    Arv* a2= arv_cria('b',arv_criavazia(),a1);
```

```
    /* sub-árvore 'e' */
```

```
    Arv* a3= arv_cria('e',arv_criavazia(),arv_criavazia());
```

```
    /* sub-árvore 'f' */
```

```
    Arv* a4= arv_cria('f',arv_criavazia(),arv_criavazia());
```

```
    /* sub-árvore 'c' */
```

```
    Arv* a5= arv_cria('c',a3,a4);
```

```
    /* árvore 'a' */
```

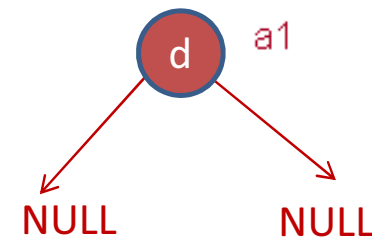
```
    Arv* a = arv_cria('a',a2,a5 );
```

```
    system("PAUSE");
```

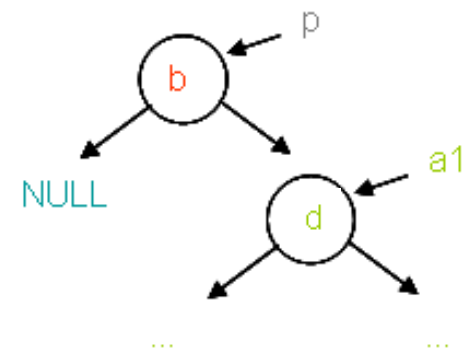
```
    return 1;
```

```
}
```

```
Arv* arv_cria (char c, Arv* sae, Arv* sad)
{
    Arv* p=(Arv*)malloc(sizeof(Arv));
    p->info = c;
    p->esq = sae;
    p->dir = sad;
    return p;
}
```



arv_cria ('b', NULL, a1)



```
#include <stdio.h>
#include <stdlib.h>
#include "arvore.h"
```

```
int main()
{
```

```
Arv* arv_cria (char c, Arv* sae, Arv* sad)
{
    Arv* p=(Arv*)malloc(sizeof(Arv));
    p->info = c;
    p->esq = sae;
    p->dir = sad;
    return p;
}
```

```
/* sub-árvore 'd' */
```

```
Arv* a1= arv_cria('d',arv_criavazia(),arv_criavazia());
```

```
/* sub-árvore 'b' */
```

```
Arv* a2= arv_cria('b',arv_criavazia(),a1);
```

```
/* sub-árvore 'e' */
```

```
Arv* a3= arv_cria('e',arv_criavazia(),arv_criavazia());
```

```
/* sub-árvore 'f' */
```

```
Arv* a4= arv_cria('f',arv_criavazia(),arv_criavazia());
```

```
/* sub-árvore 'c' */
```

```
Arv* a5= arv_cria('c',a3,a4);
```

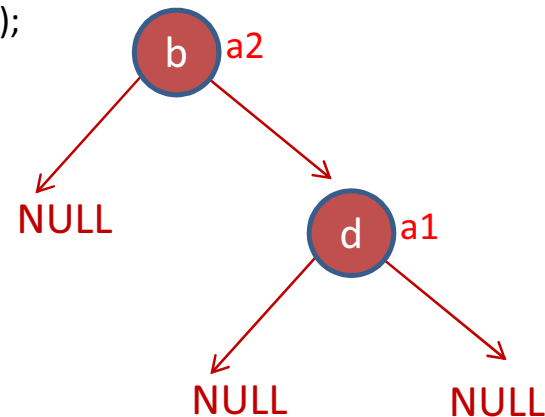
```
/* árvore 'a' */
```

```
Arv* a = arv_cria('a',a2,a5);
```

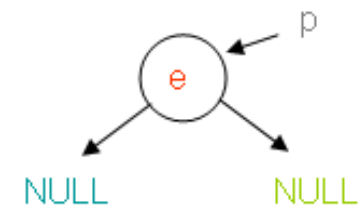
```
system("PAUSE");
```

```
return 1;
```

```
}
```



arv_cria ('e', NULL, NULL)



```
Arv* arv_cria (char c, Arv* sae, Arv* sad)
{
    Arv* p=(Arv*)malloc(sizeof(Arv));
    p->info = c;
    p->esq = sae;
    p->dir = sad;
    return p;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include "arvore.h"
```

```
int main()
{
```

```
    /* sub-árvore 'd' */
```

```
    Arv* a1= arv_cria('d',arv_criavazia(),arv_criavazia());
```

```
    /* sub-árvore 'b' */
```

```
    Arv* a2= arv_cria('b',arv_criavazia(),a1);
```

```
    /* sub-árvore 'e' */
```

```
    Arv* a3= arv_cria('e',arv_criavazia(),arv_criavazia());
```

```
    /* sub-árvore 'f' */
```

```
    Arv* a4= arv_cria('f',arv_criavazia(),arv_criavazia());
```

```
    /* sub-árvore 'c' */
```

```
    Arv* a5= arv_cria('c',a3,a4);
```

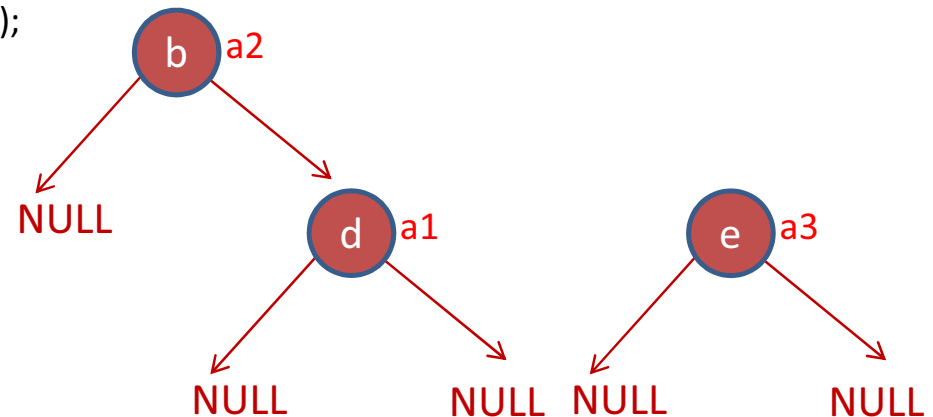
```
    /* árvore 'a' */
```

```
    Arv* a = arv_cria('a',a2,a5 );
```

```
    system("PAUSE");
```

```
    return 1;
```

```
}
```



```
#include <stdio.h>
#include <stdlib.h>
#include "arvore.h"
```

```
int main()
{
```

```
    /* sub-árvore 'd' */
```

```
    Arv* a1= arv_cria('d',arv_criavazia(),arv_criavazia());
```

```
    /* sub-árvore 'b' */
```

```
    Arv* a2= arv_cria('b',arv_criavazia(),a1);
```

```
    /* sub-árvore 'e' */
```

```
    Arv* a3= arv_cria('e',arv_criavazia(),arv_criavazia());
```

```
    /* sub-árvore 'f' */
```

```
    Arv* a4= arv_cria('f',arv_criavazia(),arv_criavazia());
```

```
    /* sub-árvore 'c' */
```

```
    Arv* a5= arv_cria('c',a3,a4);
```

```
    /* árvore 'a' */
```

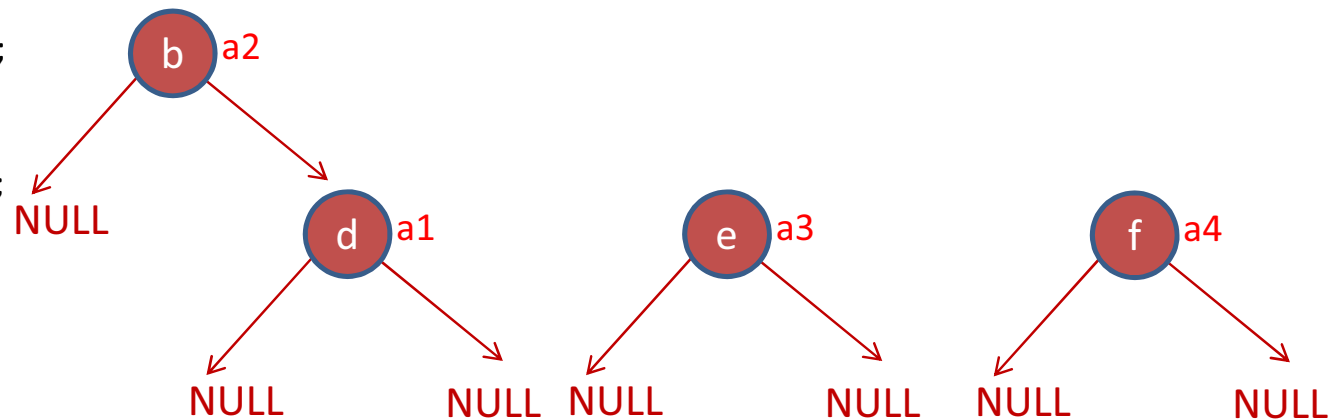
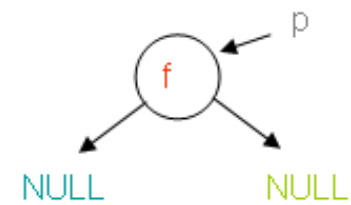
```
    Arv* a = arv_cria('a',a2,a5);
```

```
    system("PAUSE");
    return 1;
```

```
}
```

```
arv_cria ('f', NULL, NULL)
```

```
Arv* arv_cria (char c, Arv* sae, Arv* sad)
{
    Arv* p=(Arv*)malloc(sizeof(Arv));
    p->info = c;
    p->esq = sae;
    p->dir = sad;
    return p;
}
```



```
#include <stdio.h>
#include <stdlib.h>
#include "arvore.h"
```

```
int main()
{
```

```
/* sub-árvore 'd' */
```

```
Arv* a1= arv_cria('d',arv_criavazia(),arv_criavazia());
```

```
/* sub-árvore 'b' */
```

```
Arv* a2= arv_cria('b',arv_criavazia(),a1);
```

```
/* sub-árvore 'e' */
```

```
Arv* a3= arv_cria('e',arv_criavazia(),arv_criavazia());
```

```
/* sub-árvore 'f' */
```

```
Arv* a4= arv_cria('f',arv_criavazia(),arv_criavazia());
```

```
/* sub-árvore 'c' */
```

```
Arv* a5= arv_cria('c',a3,a4);
```

```
/* árvore 'a' */
```

```
Arv* a = arv_cria('a',a2,a5);
```

```
system("PAUSE");
return 1;
```

```
}
```

```
arv_cria ('c', a3, a4)
```

```
Arv* arv_cria (char c, Arv* sae, Arv* sad)
{
```

```
Arv* p=(Arv*)malloc(sizeof(Arv));
```

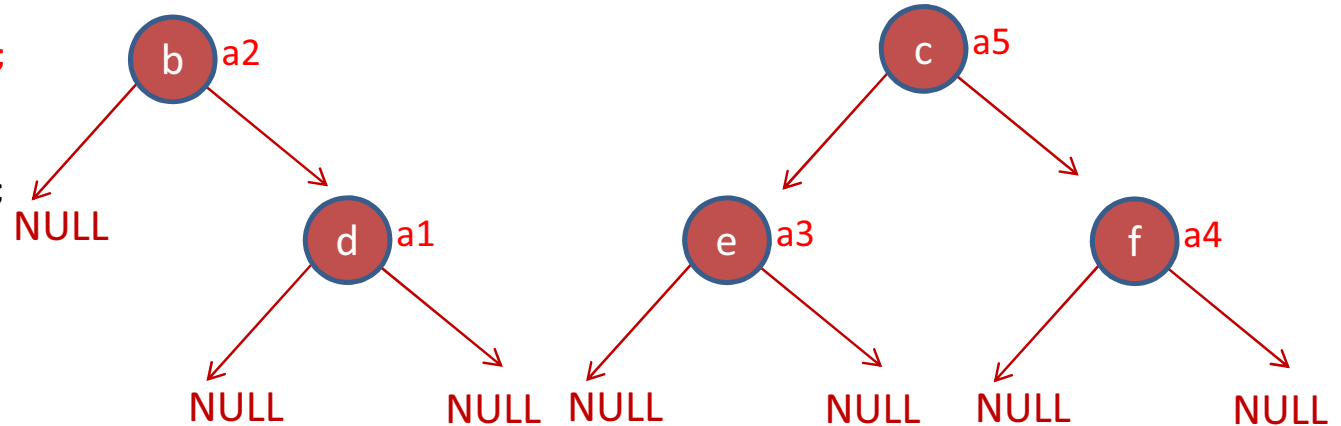
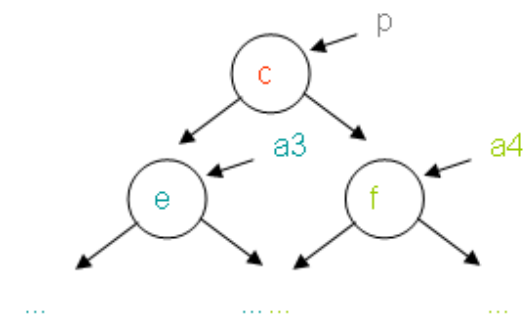
```
p->info = c;
```

```
p->esq = sae;
```

```
p->dir = sad;
```

```
return p;
```

```
}
```



```
#include <stdio.h>
#include <stdlib.h>
#include "arvore.h"
```

```
int main()
{
```

```
/* sub-árvore 'd' */
```

```
Arv* a1= arv_cria('d',arv_criavazia(),arv_criavazia());
```

```
/* sub-árvore 'b' */
```

```
Arv* a2= arv_cria('b',arv_criavazia(),a1);
```

```
/* sub-árvore 'e' */
```

```
Arv* a3= arv_cria('e',arv_criavazia(),arv_criavazia());
```

```
/* sub-árvore 'f' */
```

```
Arv* a4= arv_cria('f',arv_criavazia(),arv_criavazia());
```

```
/* sub-árvore 'c' */
```

```
Arv* a5= arv_cria('c',a3,a4);
```

```
/* árvore 'a' */
```

```
Arv* a = arv_cria('a',a2,a5);
```

NULL

```
system("PAUSE");
return 1;
```

```
}
```

arv_cria ('a', a2, a5)

```
Arv* arv_cria (char c, Arv* sae, Arv* sad)
{
```

```
Arv* p=(Arv*)malloc(sizeof(Arv));
```

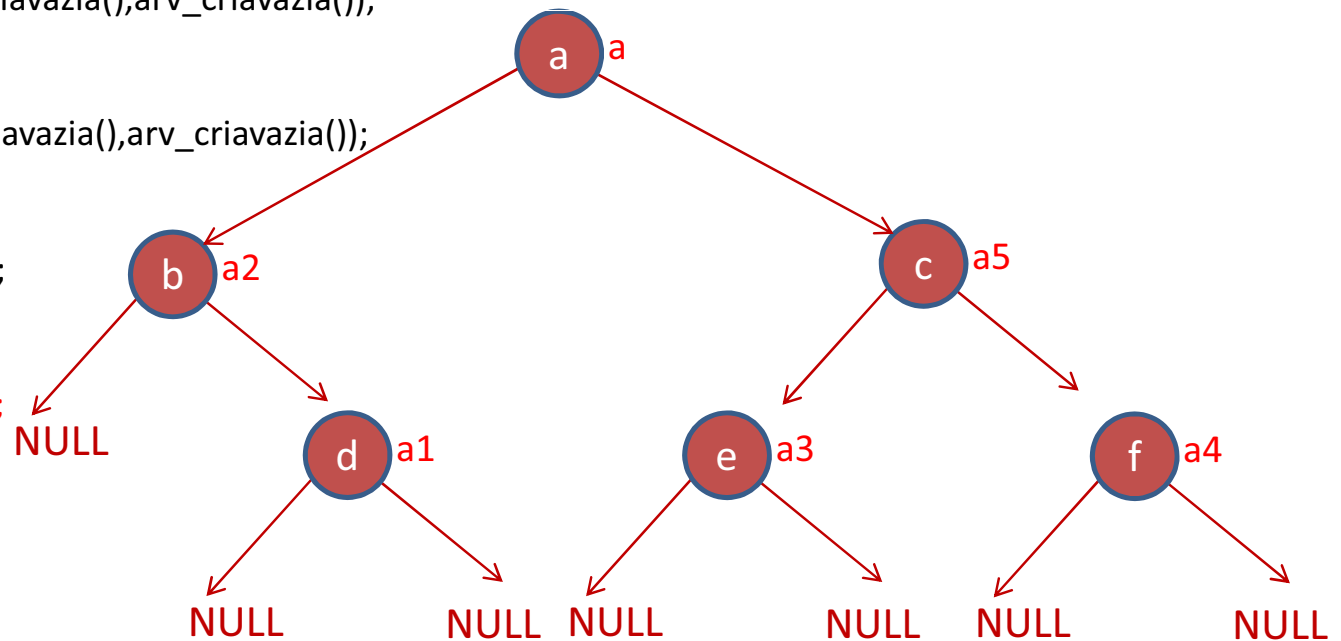
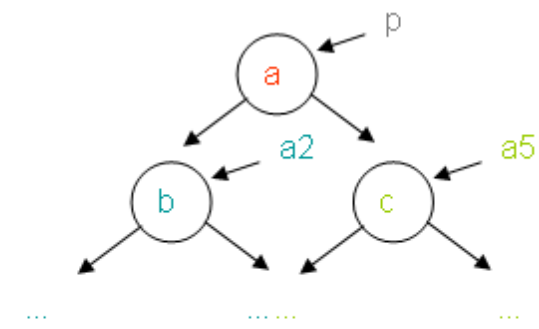
```
p->info = c;
```

```
p->esq = sae;
```

```
p->dir = sad;
```

```
return p;
```

```
}
```



Árvores binárias - Implementação em C

- função `arv_libera`
 - libera memória alocada pela estrutura da árvore
 - as sub-árvores devem ser liberadas antes de se liberar o nó raiz
 - retorna uma árvore vazia, representada por `NULL`

```
Arv* arv_libera (Arv* a){  
    if (!arv_vazia(a)){  
        arv_libera(a->esq);    /* libera sae */  
        arv_libera(a->dir);    /* libera sad */  
        free(a);               /* libera raiz */  
    }  
    return NULL;  
}
```

Árvores binárias - Implementação em C

- função `arv_vazia`
 - indica se uma árvore é ou não vazia

```
int arv_vazia (Arv* a)
{
    return a==NULL;
}
```

```
struct arv {  
    char info;  
    struct arv* esq;  
    struct arv* dir;  
};  
typedef struct arv Arv;
```

```
Arv* arv_criavazia (void);  
Arv* arv_cria (char c, Arv* e, Arv* d);  
Arv* arv_libera (Arv* a);  
int arv_vazia (Arv* a);
```

...

```
int arv_vazia (Arv* a)  
{  
    return a==NULL;  
}
```

```
Arv* arv_libera (Arv* a)  
{  
    if (!arv_vazia(a))  
    {  
        arv_libera(a->esq); /* libera sae */  
        arv_libera(a->dir); /* libera sad */  
        free(a); /* libera raiz */  
    }  
    return NULL;  
}
```



```
#include <stdio.h>
#include <stdlib.h>
#include "arvore.h"

int main()
{

    /* sub-árvore 'd' */
    Arv* a1= arv_cria('d',arv_criavazia(),arv_criavazia());

    /* sub-árvore 'b' */
    Arv* a2= arv_cria('b',arv_criavazia(),a1);

    /* sub-árvore 'e' */
    Arv* a3= arv_cria('e',arv_criavazia(),arv_criavazia());

    /* sub-árvore 'f' */
    Arv* a4= arv_cria('f',arv_criavazia(),arv_criavazia());

    /* sub-árvore 'c' */
    Arv* a5= arv_cria('c',a3,a4);

    /* árvore 'a' */
    Arv* a = arv_cria('a',a2,a5 );

    a = arv_libera (a);

    system("PAUSE");
    return 1;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include "arvore.h"

int main()
{

    /* sub-árvore 'd' */
    Arv* a1= arv_cria('d',arv_criavazia(),arv_criavazia());

    /* sub-árvore 'b' */
    Arv* a2= arv_cria('b',arv_criavazia(),a1);

    /* sub-árvore 'e' */
    Arv* a3= arv_cria('e',arv_criavazia(),arv_criavazia());

    /* sub-árvore 'f' */
    Arv* a4= arv_cria('f',arv_criavazia(),arv_criavazia());

    /* sub-árvore 'c' */
    Arv* a5= arv_cria('c',a3,a4);

    /* árvore 'a' */
    Arv* a = arv_cria('a',a2,a5 );

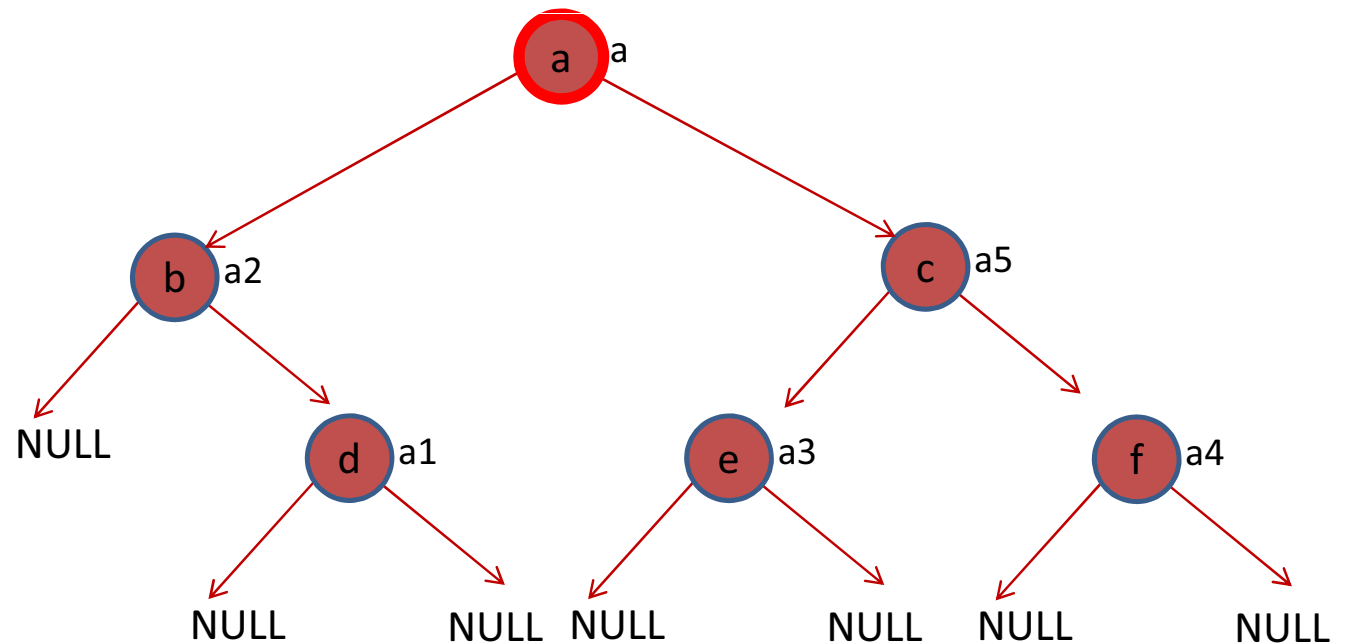
    a = arv_libera (a);

    system("PAUSE");
    return 1;
}
```

```
int arv_vazia (Arv* a)
{
    return a==NULL;
}
```

`a = arv_libera (a);`

```
Arv* arv_libera (Arv* a)
{
    if (!arv_vazia(a))
    {
        arv_libera(a->esq); /* libera sae */
        arv_libera(a->dir); /* libera sad */
        free(a); /* libera raiz */
    }
    return NULL;
}
```



```

int arv_vazia (Arv* a)
{
    return a==NULL;
}

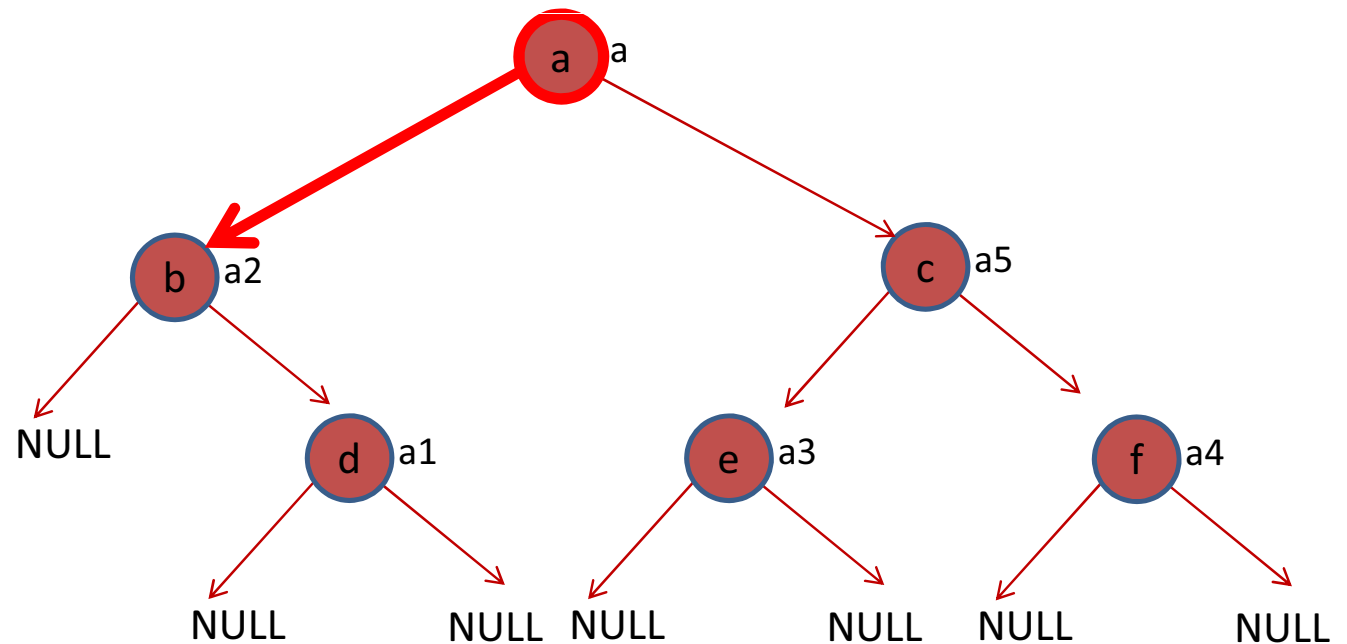
```

`a = arv_libera (a);`

```

Arv* arv_libera (Arv* a)
{
    if (!arv_vazia(a))
    {
        arv_libera(a->esq); /* libera sae */
        arv_libera(a->dir); /* libera sad */
        free(a); /* libera raiz */
    }
    return NULL;
}

```



```

int ar int arv_vazia (Arv* a)
{
    {
    retu return a==NULL;
    }
}

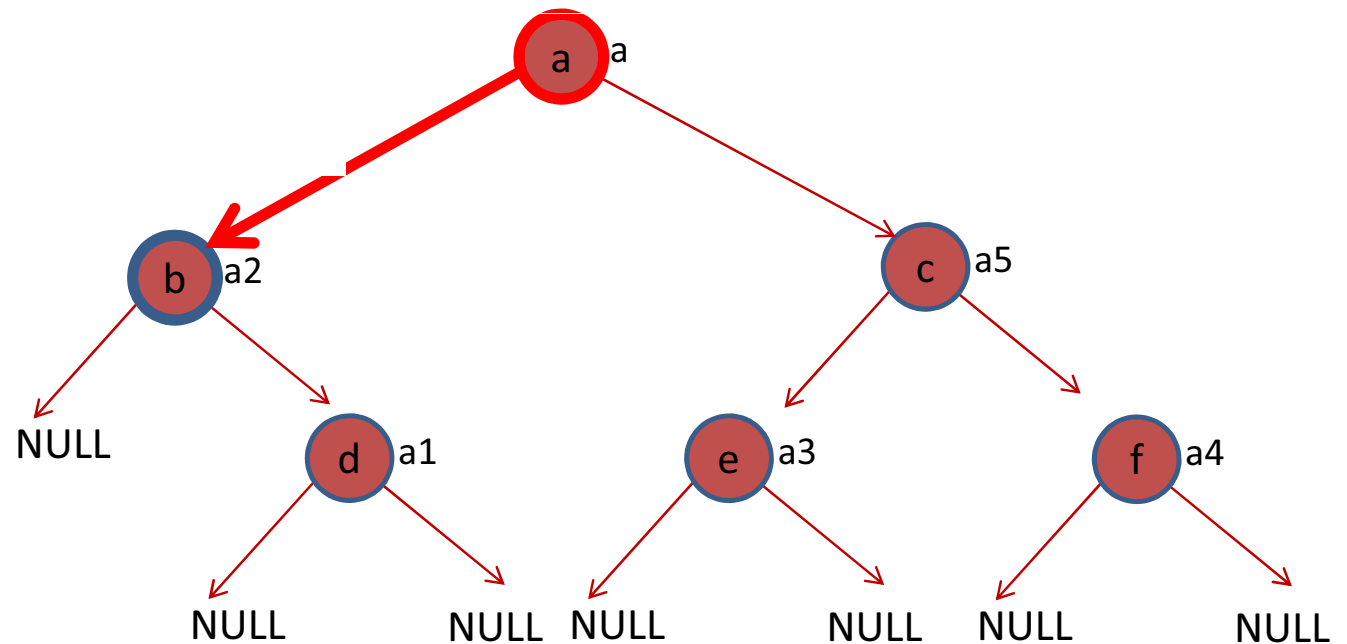
```

a = arv_libera (a2);

```

Arv* Arv* arv_libera (Arv* a)
{
    {
    if (!a if (!arv_vazia(a))
    {
        {
        arv arv_libera(a->esq);
        arv arv_libera(a->dir);
        fre free(a);
        }
    }
    retu return NULL;
    }
}

```



```

int arv_vazia (Arv* a)
{
    return a==NULL;
}

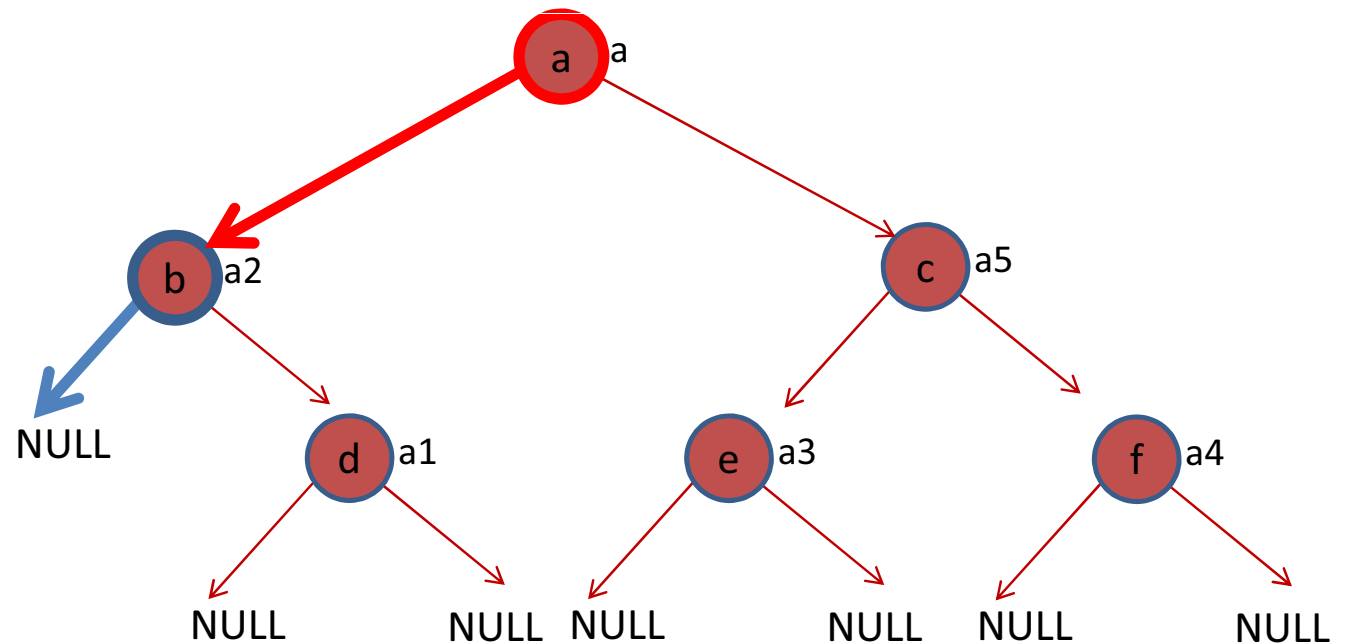
```

`a = arv_libera (a2);`

```

Arv* arv_libera (Arv* a)
{
    if (!arv_vazia(a))
    {
        arv_libera(a->esq);
        arv_libera(a->dir);
        free(a);
    }
    return NULL;
}

```



```

int arv_vazia (Arv* a)
{
    if (a == NULL)
        return a==NULL;
}

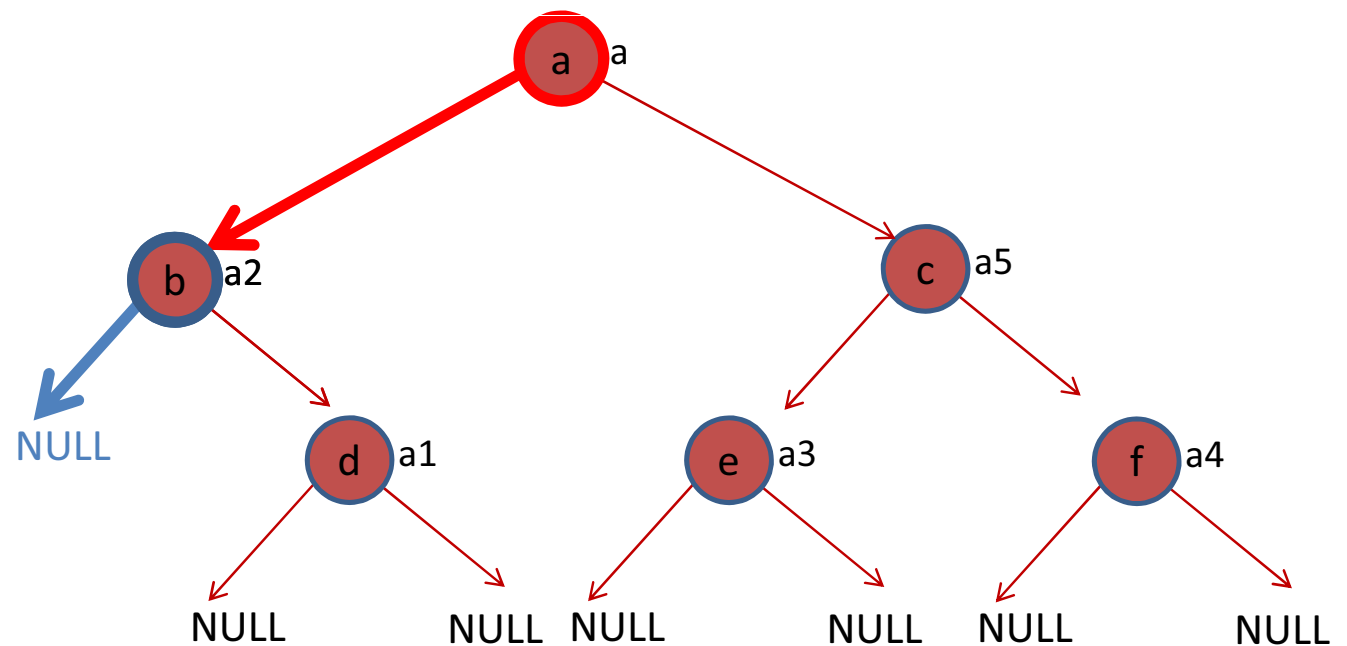
```

a = arv_libera (NULL);

```

Arv* arv_libera (Arv* a)
{
    if (!arv_vazia(a))
    {
        arv_libera(a->esq);
        arv_libera(a->dir);
        free(a);
    }
    return NULL;
}

```



```

int arv_vazia (Arv* a)
{
    if (a == NULL)
        return a==NULL;
}

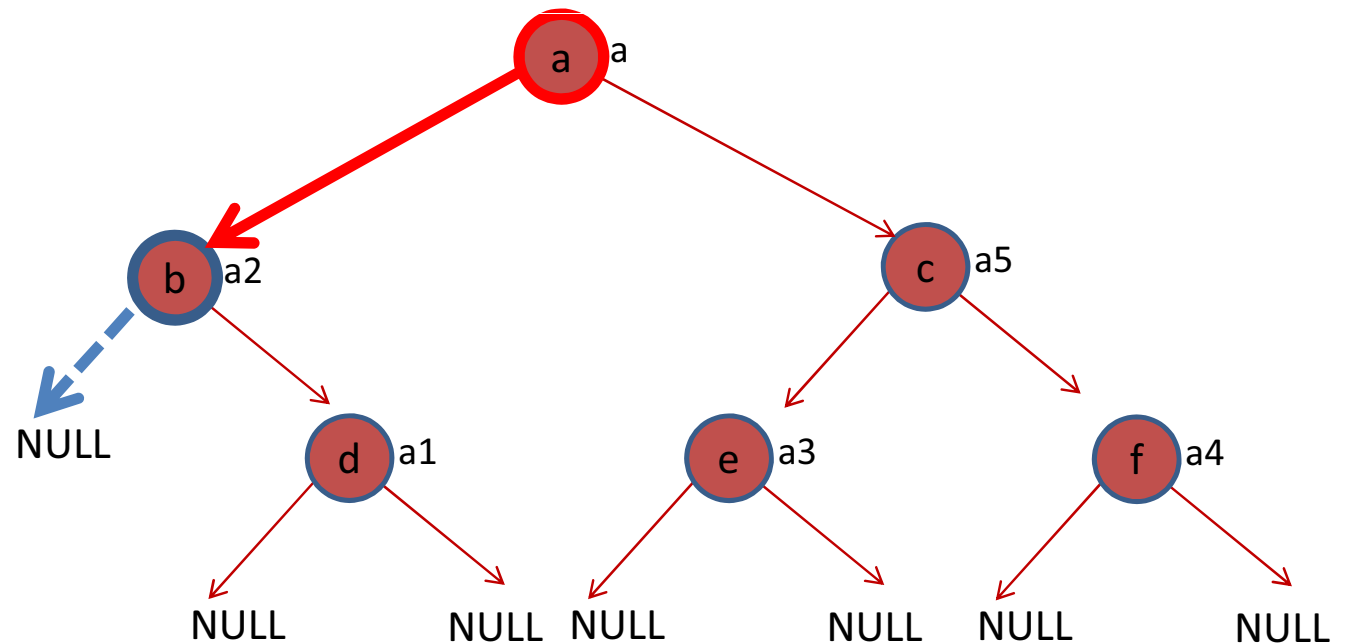
```

`a = arv_libera (NULL);`

```

Arv* arv_libera (Arv* a)
{
    if (!arv_vazia(a))
    {
        arv_libera(a->esq);
        arv_libera(a->dir);
        free(a);
    }
    return NULL;
}

```




```

int arv_vazia (Arv* a)
{
    return a==NULL;
}

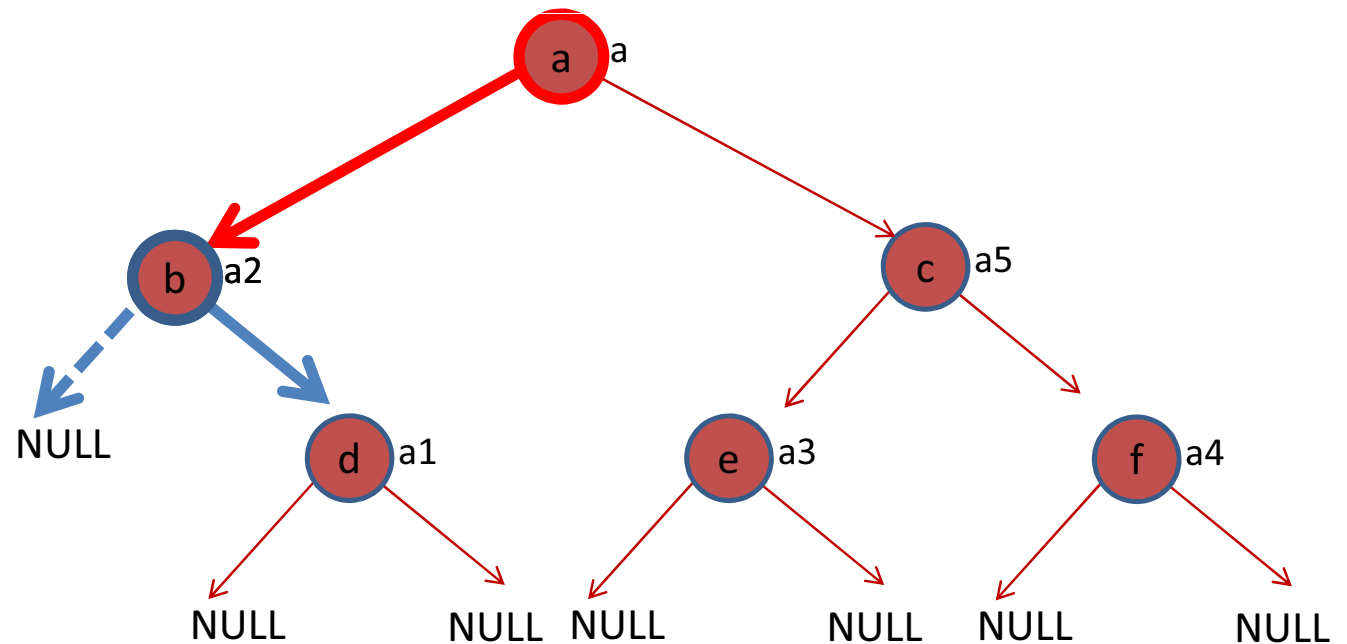
```

a = arv_libera (a2);

```

Arv* arv_libera (Arv* a)
{
    if (!arv_vazia(a))
    {
        arv_libera(a->esq);
        arv_libera(a->dir);
        free(a); /* libera raiz */
    }
    return NULL;
}

```



```

int arv_vazia (Arv* a)
{
    return a==NULL;
}

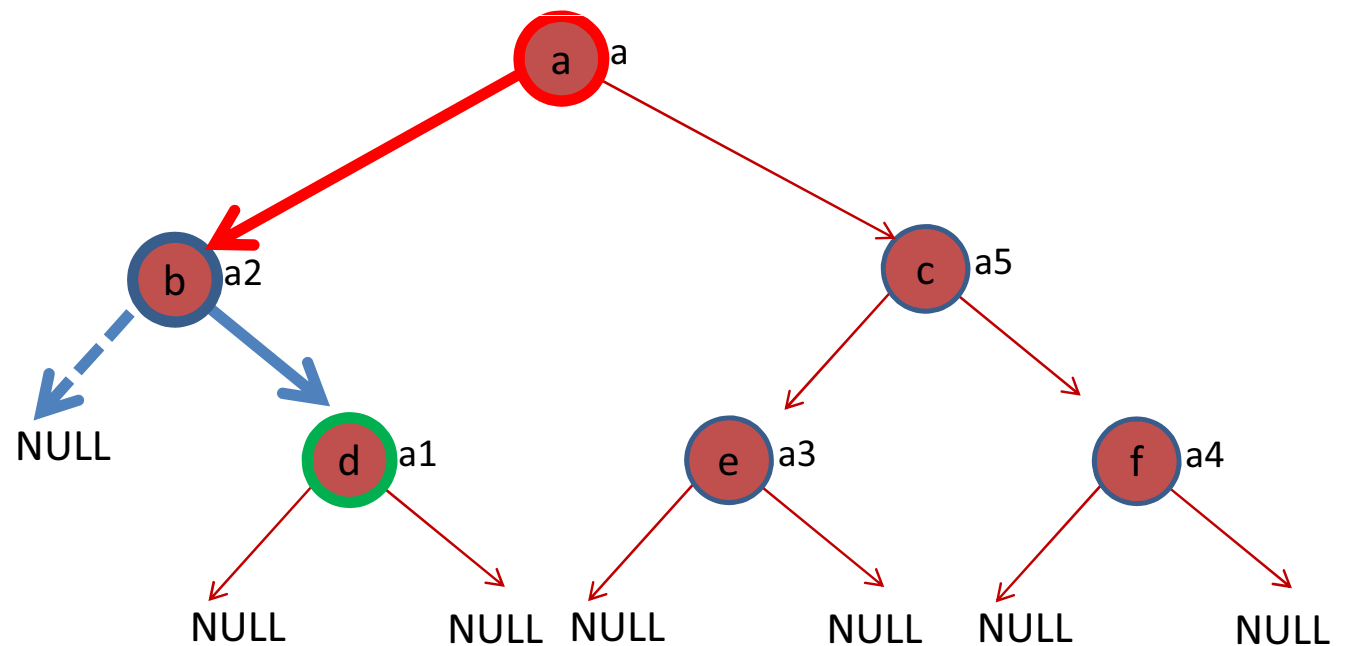
```

a = arv_libera (a1);

```

Arv* arv_libera (Arv* a)
{
    if (!arv_vazia(a))
    {
        arv_libera(a->esq);
        arv_libera(a->dir);
        free(a);
    }
    return NULL;
}

```



```

int arv_vazia (Arv* a)
{
    return a==NULL;
}

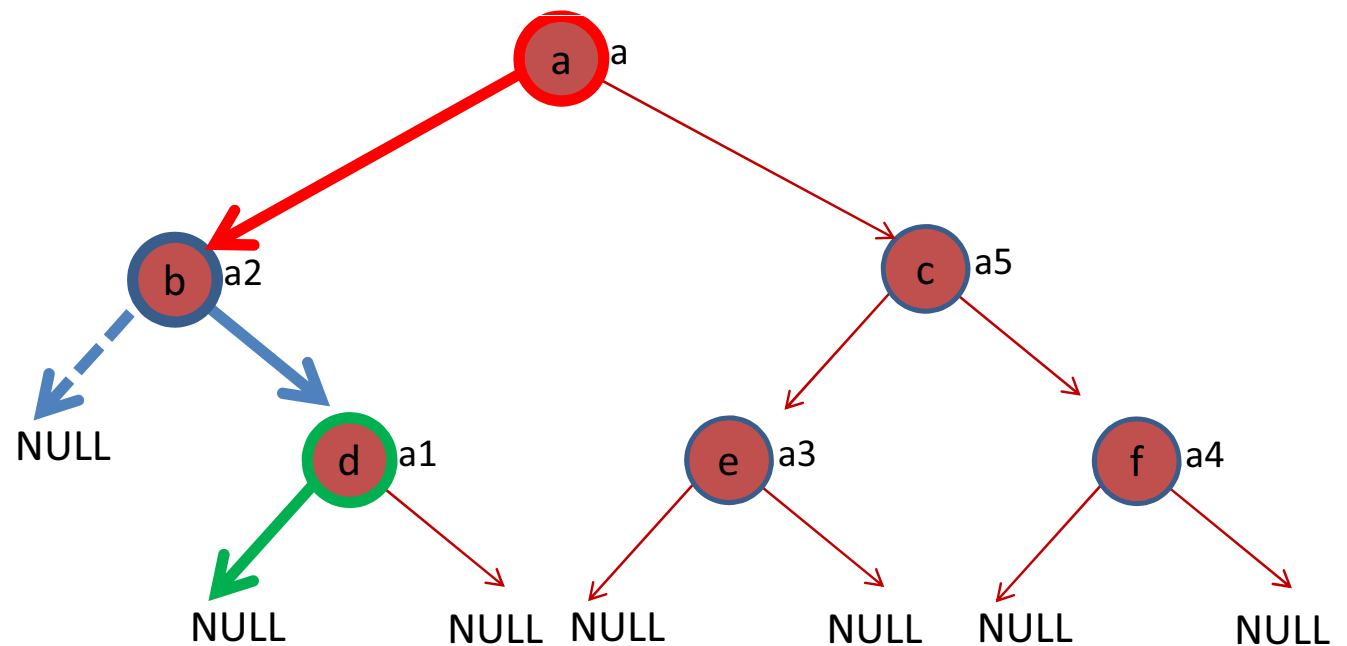
```

a = arv_libera (a1);

```

Arv* arv_libera (Arv* a)
{
    if (!arv_vazia(a))
    {
        arv_libera(a->esq);
        arv_libera(a->dir);
        free(a);
    }
    return NULL;
}

```



```

int arv_vazia (Arv* a)
{
    return a==NULL;
}

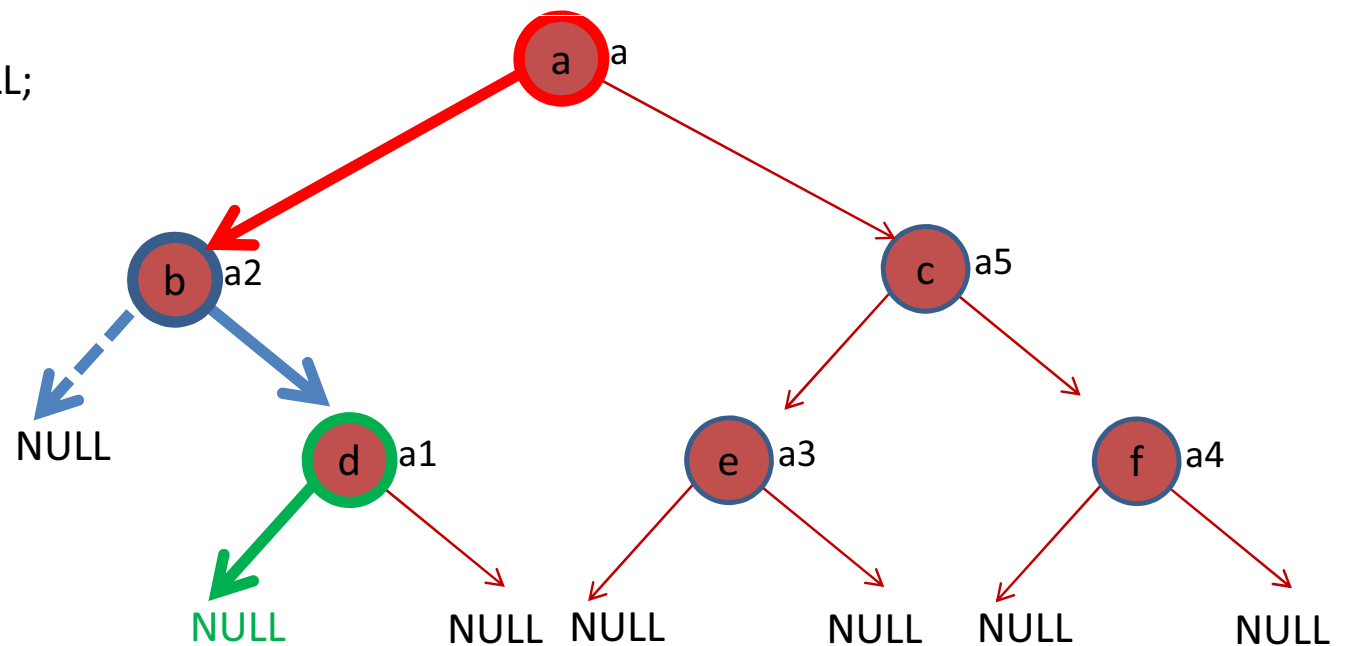
```

`a = arv_libera (NULL);`

```

Arv* arv_libera (Arv* a)
{
    if (!arv_vazia(a))
    {
        arv_libera(a->esq);
        arv_libera(a->dir);
        free(a);
    }
    return NULL;
}

```



```

int arv_vazia (Arv* a)
{
    return a==NULL;
}

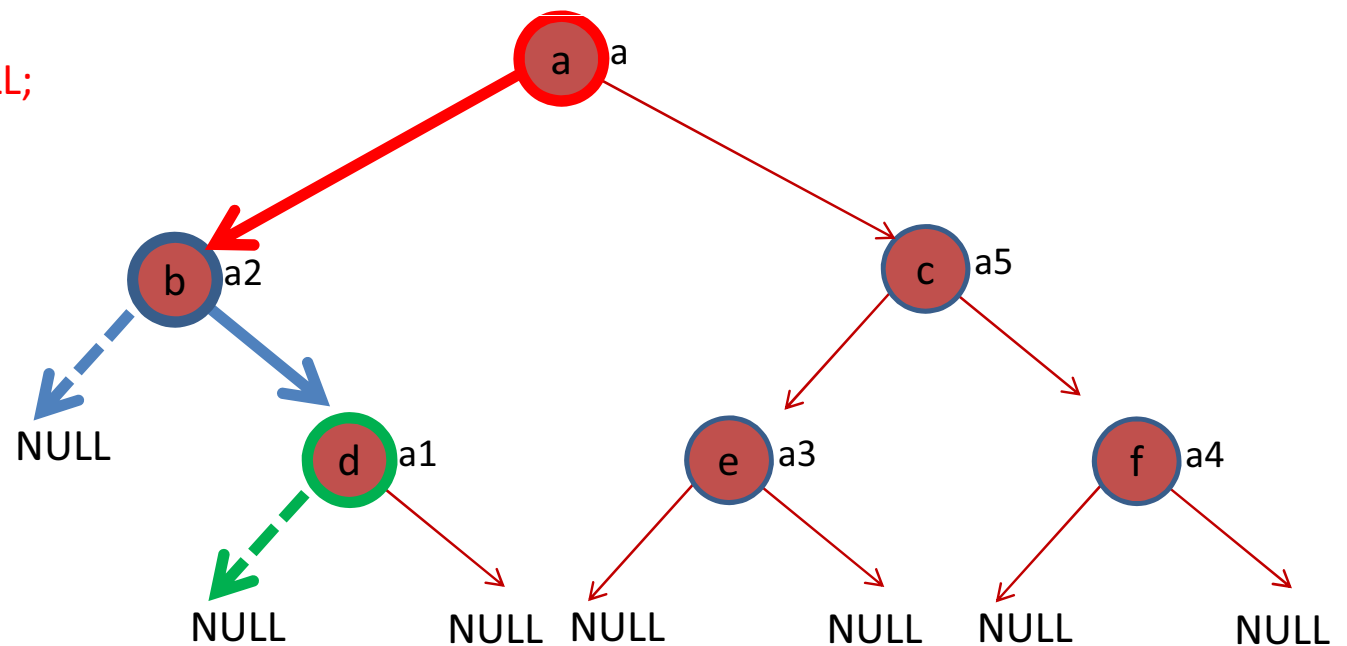
```

`a = arv_libera (NULL);`

```

Arv* arv_libera (Arv* a)
{
    if (!arv_vazia(a))
    {
        arv_libera(a->esq);
        arv_libera(a->dir);
        free(a);
    }
    return NULL;
}

```



```

int arv_vazia (Arv* a)
{
    return a==NULL;
}

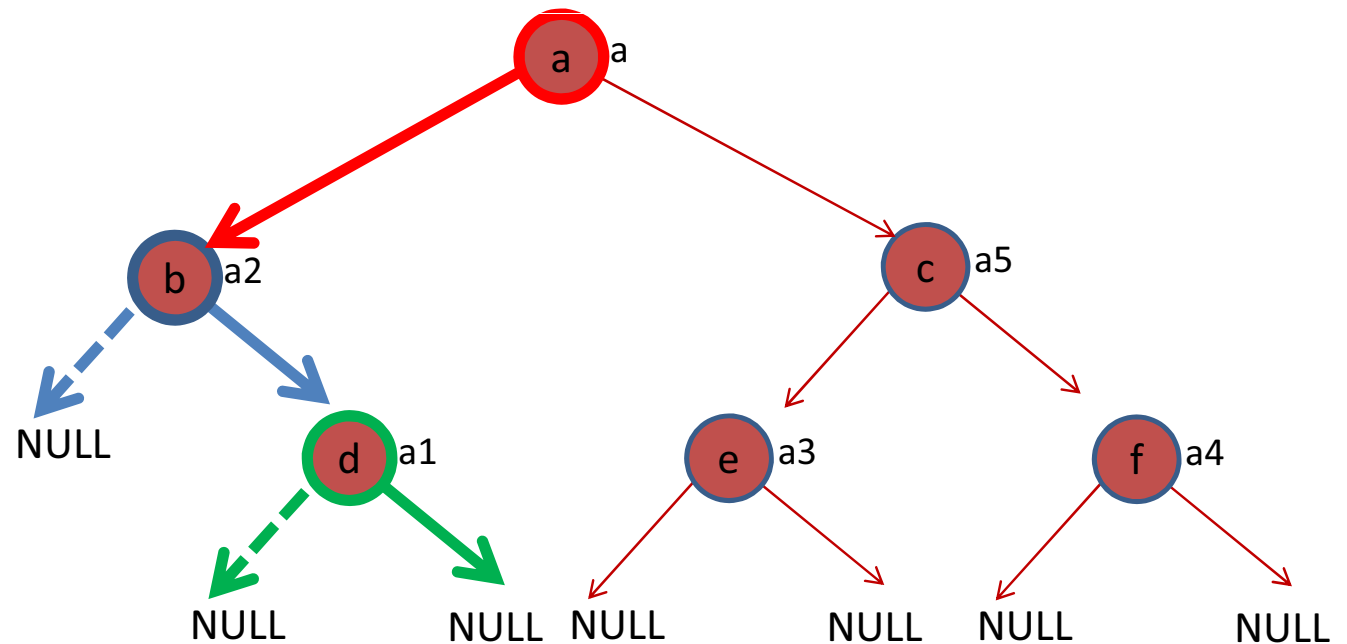
```

a = arv_libera (a1);

```

Arv* arv_libera (Arv* a)
{
    if (!arv_vazia(a))
    {
        arv_libera(a->esq);
        arv_libera(a->dir);
        free(a);
    }
    return NULL;
}

```



```

int arv_vazia (Arv* a)
{
    return a==NULL;
}

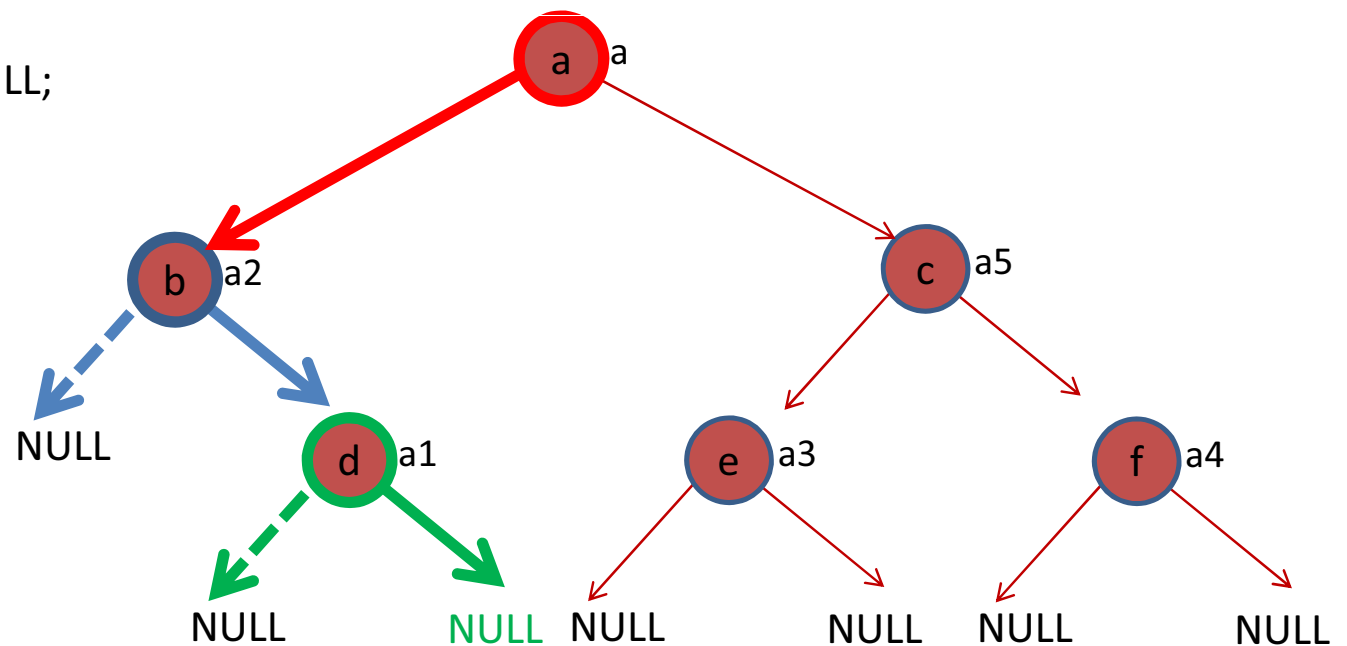
```

`a = arv_libera (NULL);`

```

Arv* arv_libera (Arv* a)
{
    if (!arv_vazia(a))
    {
        arv_libera(a->esq);
        arv_libera(a->dir);
        free(a);
    }
    return NULL;
}

```



```

int arv_vazia (Arv* a)
{
    return a==NULL;
}

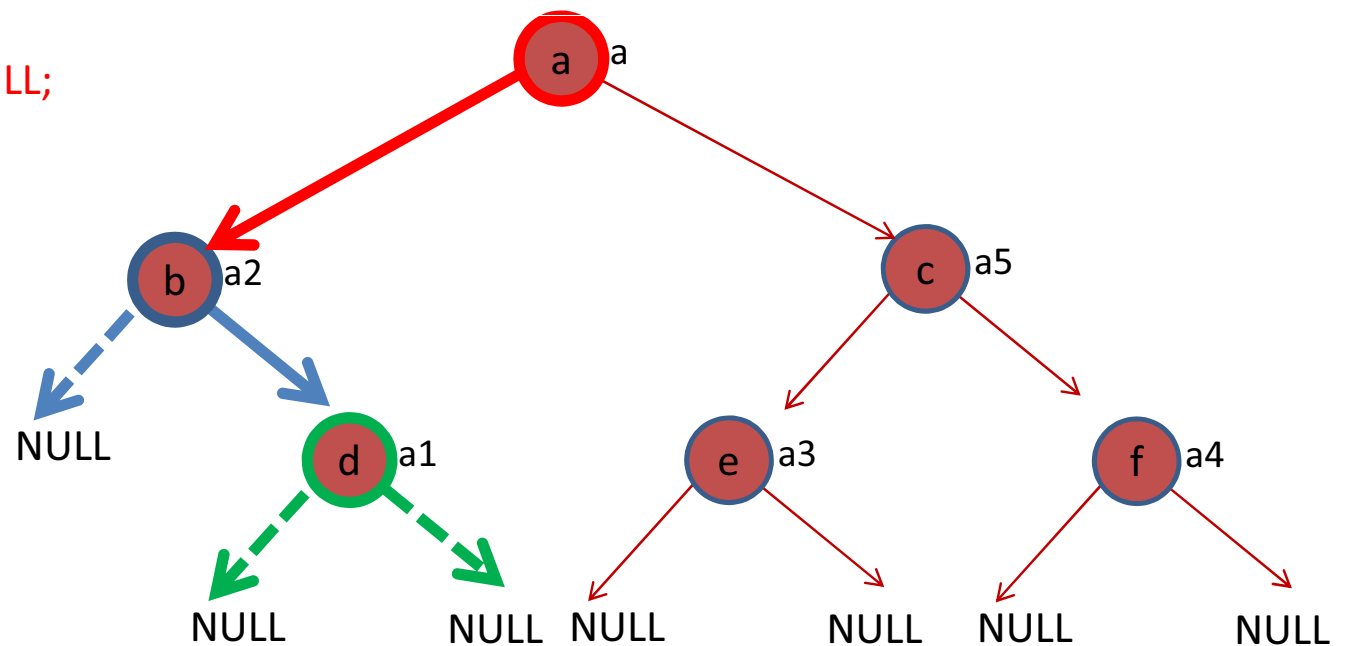
```

a = arv_libera (NULL);

```

Arv* arv_libera (Arv* a)
{
    if (!arv_vazia(a))
    {
        arv_libera(a->esq);
        arv_libera(a->dir);
        free(a);
    }
    return NULL;
}

```




```

int arv_vazia (Arv* a)
{
    return a==NULL;
}

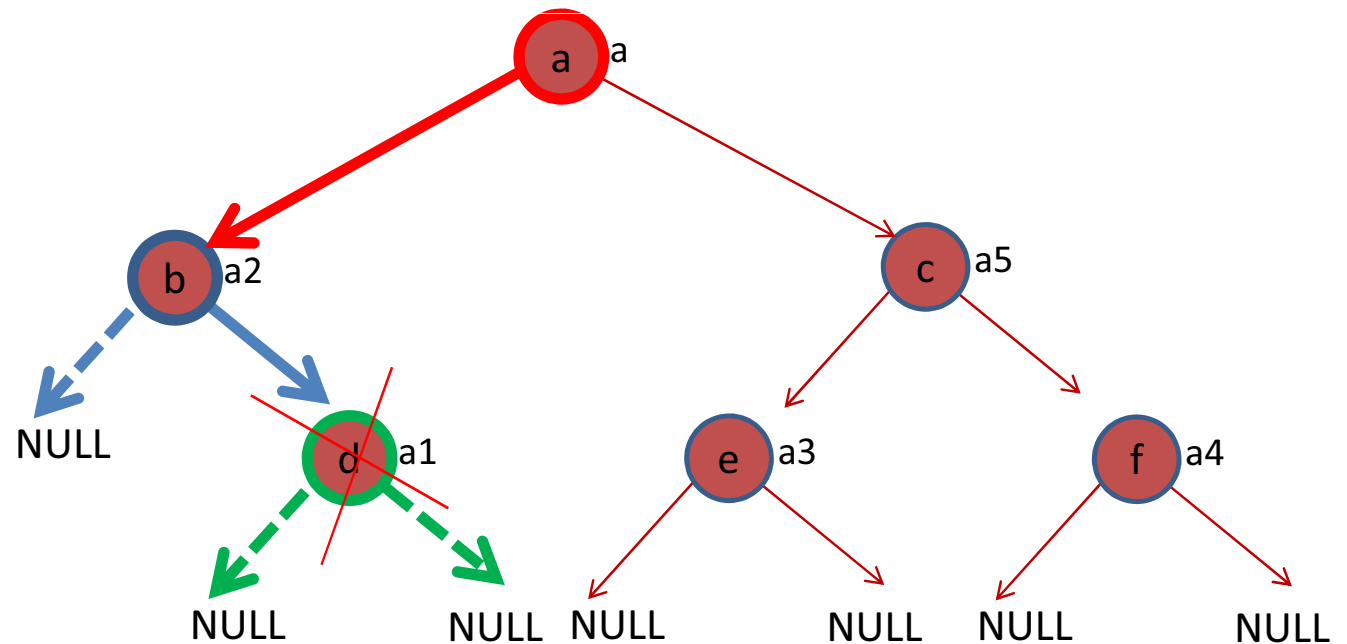
```

a = arv_libera (a1);

```

Arv* arv_libera (Arv* a)
{
    if (!arv_vazia(a))
    {
        arv_libera(a->esq);
        arv_libera(a->dir);
        free(a);
    }
    return NULL;
}

```



```

int arv_vazia (Arv* a)
{
    return a==NULL;
}

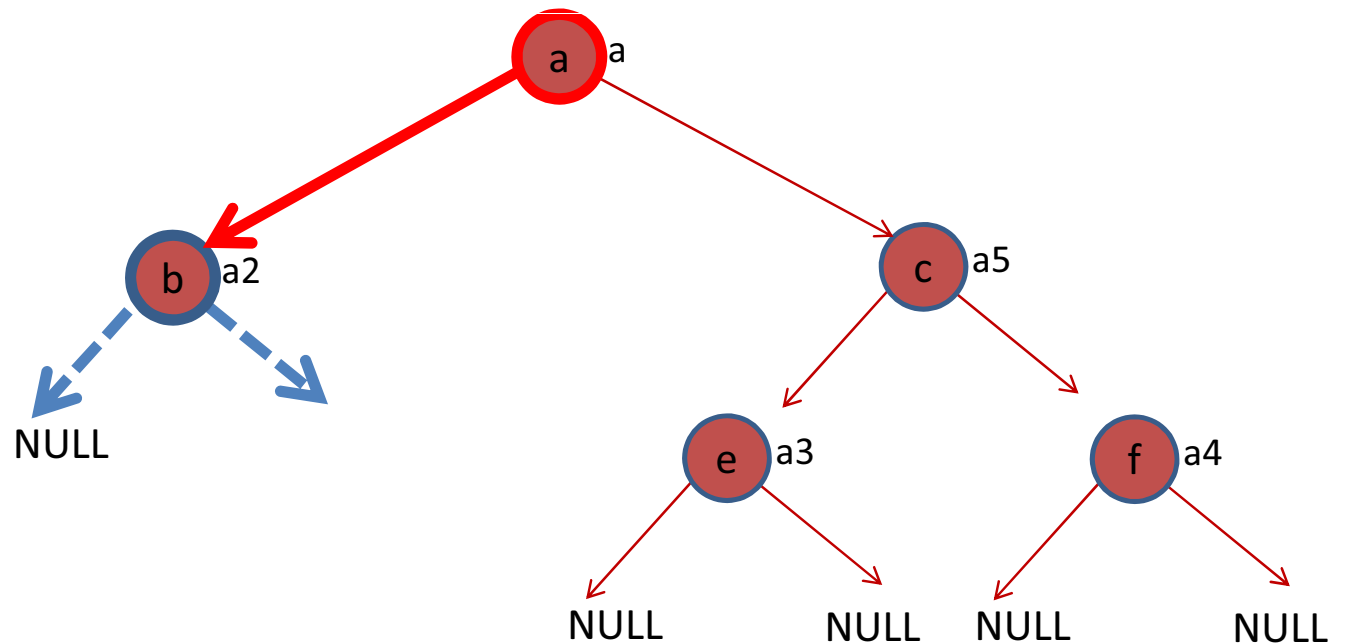
```

a = arv_libera (a1);

```

Arv* arv_libera (Arv* a)
{
    if (!arv_vazia(a))
    {
        arv_libera(a->esq);
        arv_libera(a->dir);
        free(a);
    }
    return NULL;
}

```



```

int arv_vazia (Arv* a)
{
    return a==NULL;
}

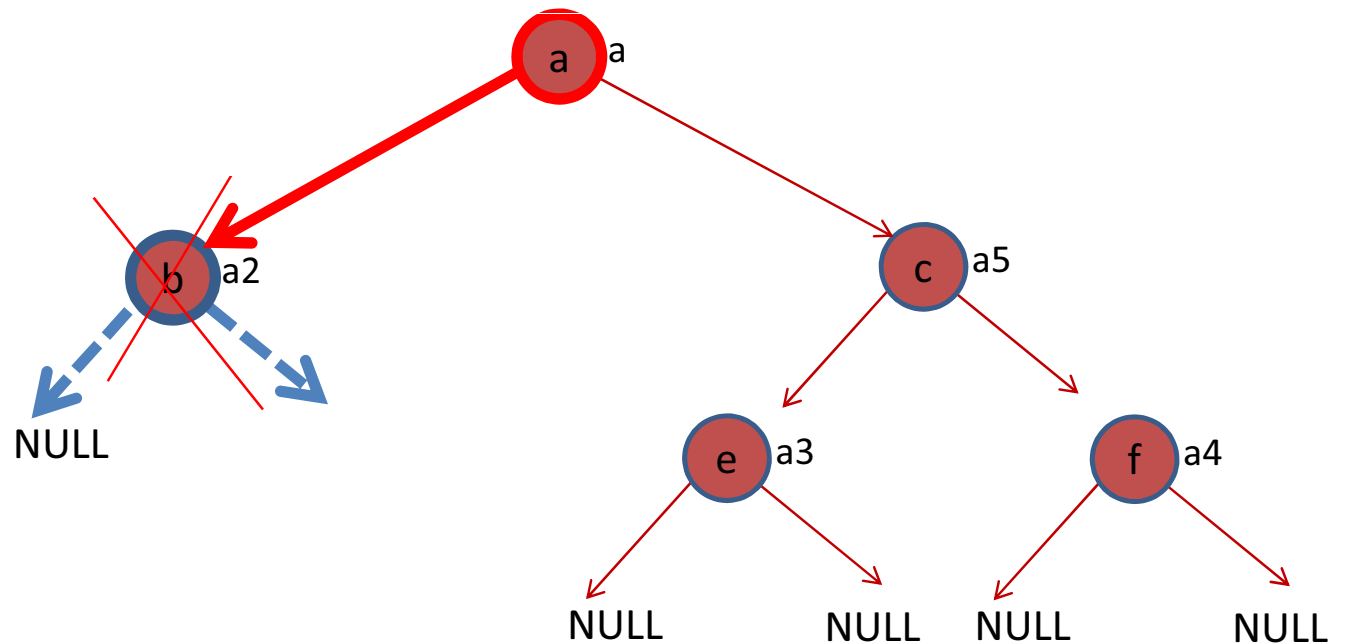
```

a = arv_libera (a2);

```

Arv* arv_libera (Arv* a)
{
    if (!arv_vazia(a))
    {
        arv_libera(a->esq);
        arv_libera(a->dir);
        free(a);
    }
    return NULL;
}

```



```

int : int arv_vazia (Arv* a)
{
    {
    ret return a==NULL;
    }
}

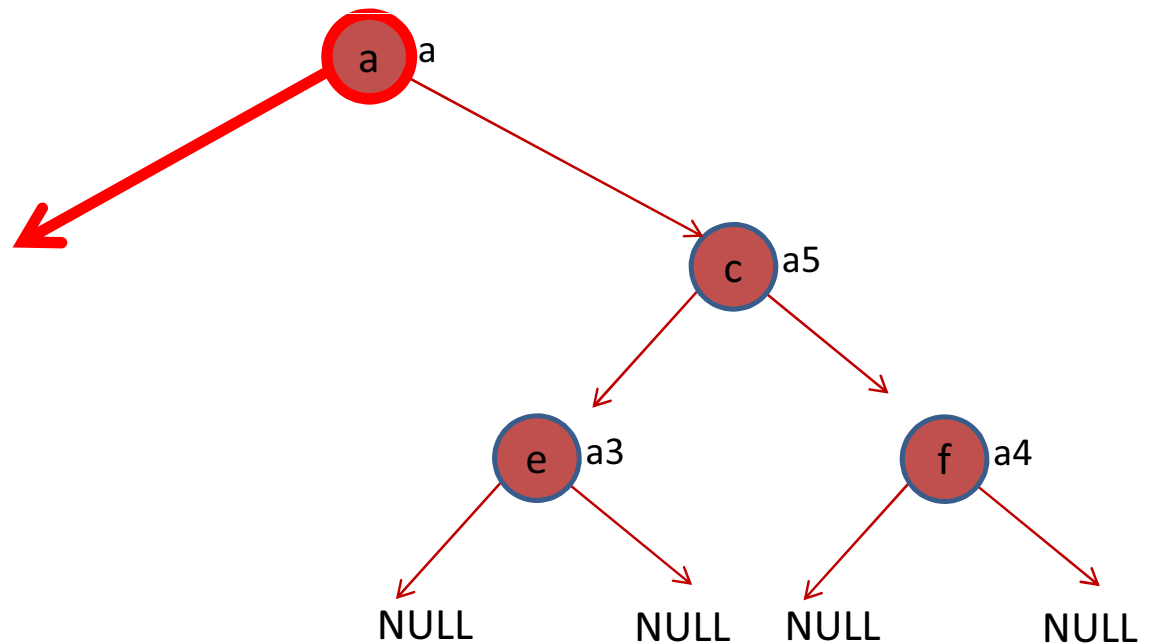
```

```

Arv Arv* arv_libera (Arv* a)
{
    {
    if (! if (!arv_vazia(a))
    {
        a arv_libera(a->esq);
        a arv_libera(a->dir);
        fr free(a);
    }
}
ret return NULL;
}

```

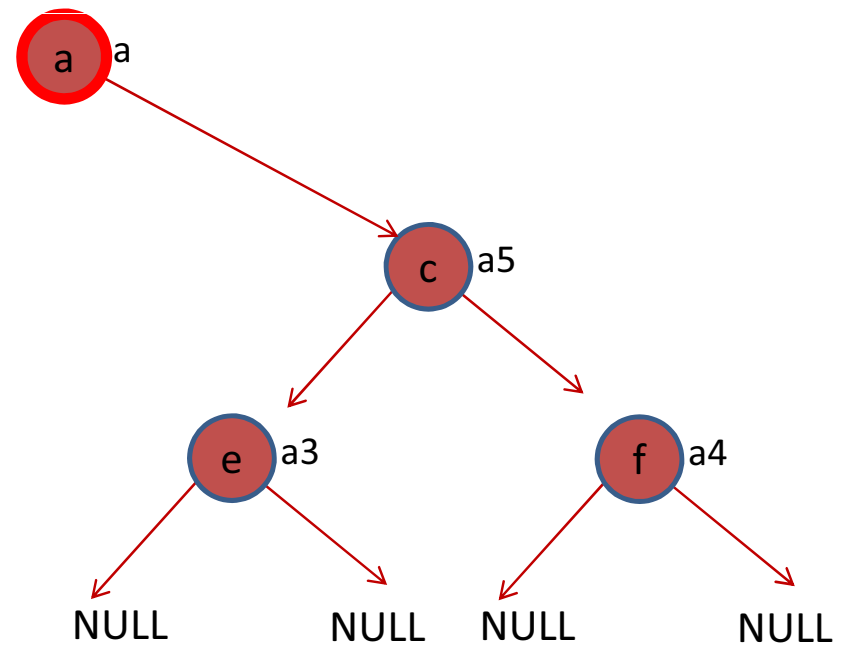
a = arv_libera (a2);



```
int arv_vazia (Arv* a)
{
    return a==NULL;
}
```

```
Arv* arv_libera (Arv* a)
{
    if (!arv_vazia(a))
    {
        arv_libera(a->esq);
        arv_libera(a->dir);
        free(a); /* libera raiz */
    }
    return NULL;
}
```

a = arv_libera (a);



E assim sucessivamente.... até que
toda a árvore seja liberada

Árvores binárias - Implementação em C

- função `arv_pertence`
 - verifica a ocorrência de um caractere `c` em um de nós
 - retorna um valor booleano (1 ou 0) indicando a ocorrência ou não do caractere na árvore

```
int arv_pertence (Arv* a, char c){  
    if (arv_vazia(a))  
        return 0;                /* árvore vazia: não encontrou */  
    else  
        return a->info==c ||  
               arv_pertence(a->esq,c) ||  
               arv_pertence(a->dir,c);  
}
```

```
struct arv {  
    char info;  
    struct arv* esq;  
    struct arv* dir;  
};  
typedef struct arv Arv;
```

```
Arv* arv_criavazia (void);  
Arv* arv_cria (char c, Arv* e, Arv* d);  
Arv* arv_libera (Arv* a);  
int arv_vazia (Arv* a);  
int arv_pertence (Arv* a, char c);
```

```
int arv_vazia (Arv* a)  
{  
    return a==NULL;  
}
```

```
int arv_pertence (Arv* a, char c)  
{  
    if (arv_vazia(a))  
        return 0; /* árvore vazia: não encontrou */  
    else  
        return a->info==c || arv_pertence(a->esq,c) ||  
        arv_pertence(a->dir,c);  
}
```



```

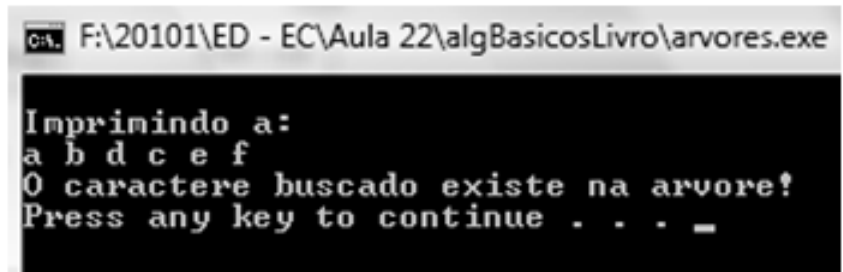
#include <stdio.h>
#include <stdlib.h>
#include "arvore.h"

int main()
{
    Arv* a1= arv_cria('d',arv_criavazia(),arv_criavazia());
    Arv* a2= arv_cria('b',arv_criavazia(),a1);
    Arv* a3= arv_cria('e',arv_criavazia(),arv_criavazia());
    Arv* a4= arv_cria('f',arv_criavazia(),arv_criavazia());
    Arv* a5= arv_cria('c',a3,a4);
    Arv* a = arv_cria('a',a2,a5 );
    printf ("\nImprimindo a: \n");
    arv_imprime (a);

    int x = arv_pertence (a, 'd');
    if (x==0)
        printf ("\nO caractere buscado nao pertence a arvore!\n");
    else
        printf ("\nO caractere buscado existe na arvore!\n");

    a = arv_libera (a);
    system("PAUSE");
    return 1;
}

```



```

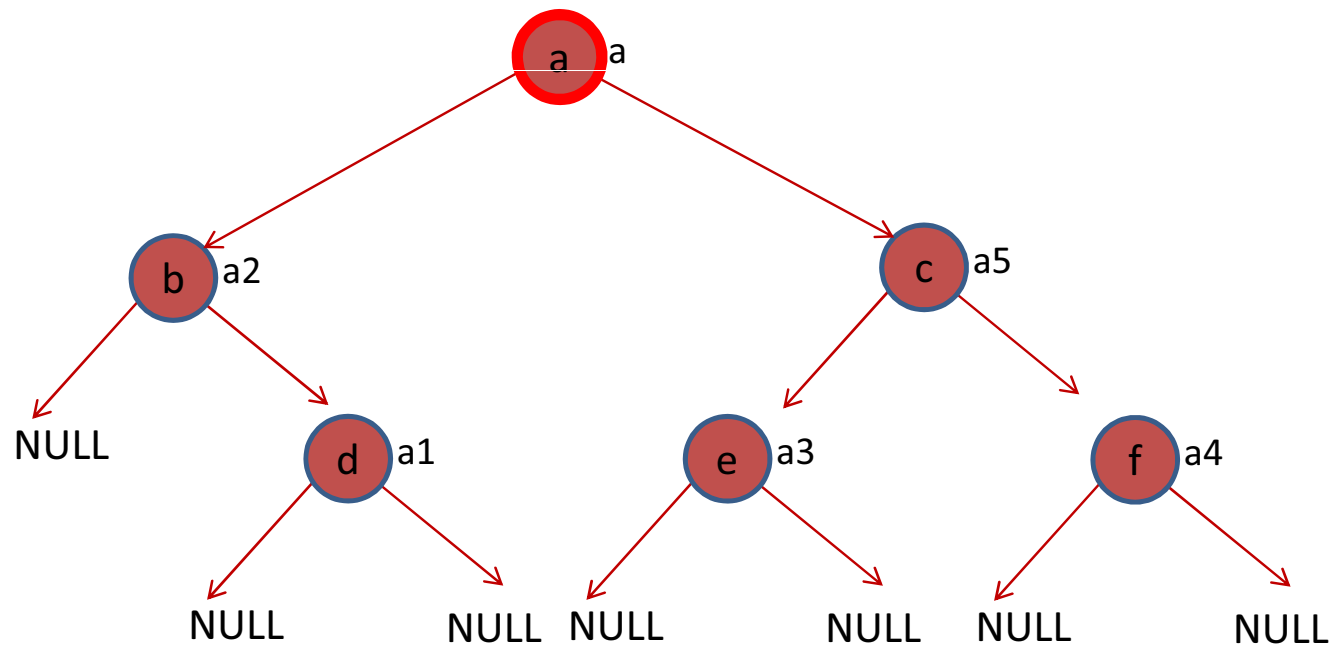
C:\> F:\20101\ED - EC\Aula 22\algBasicosLivro\arvores.exe

Imprimindo a:
a b d c e f
O caractere buscado existe na arvore!
Press any key to continue . . . _

```

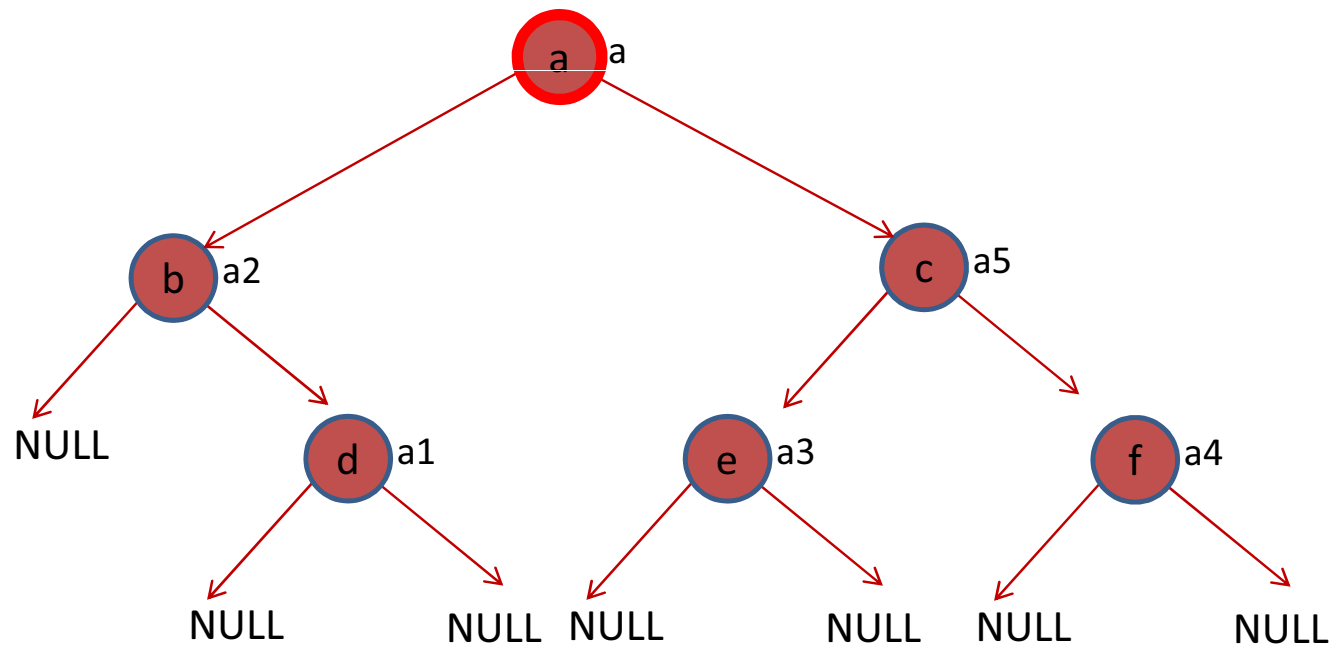
```
int x = arv_pertence (a, 'd');
```

```
int arv_pertence (Arv* a, char c)
{
    if (arv_vazia(a))
        return 0; /* árvore vazia: não encontrou */
    else
        return a->info==c || arv_pertence(a->esq,c) || arv_pertence(a->dir,c);
}
```



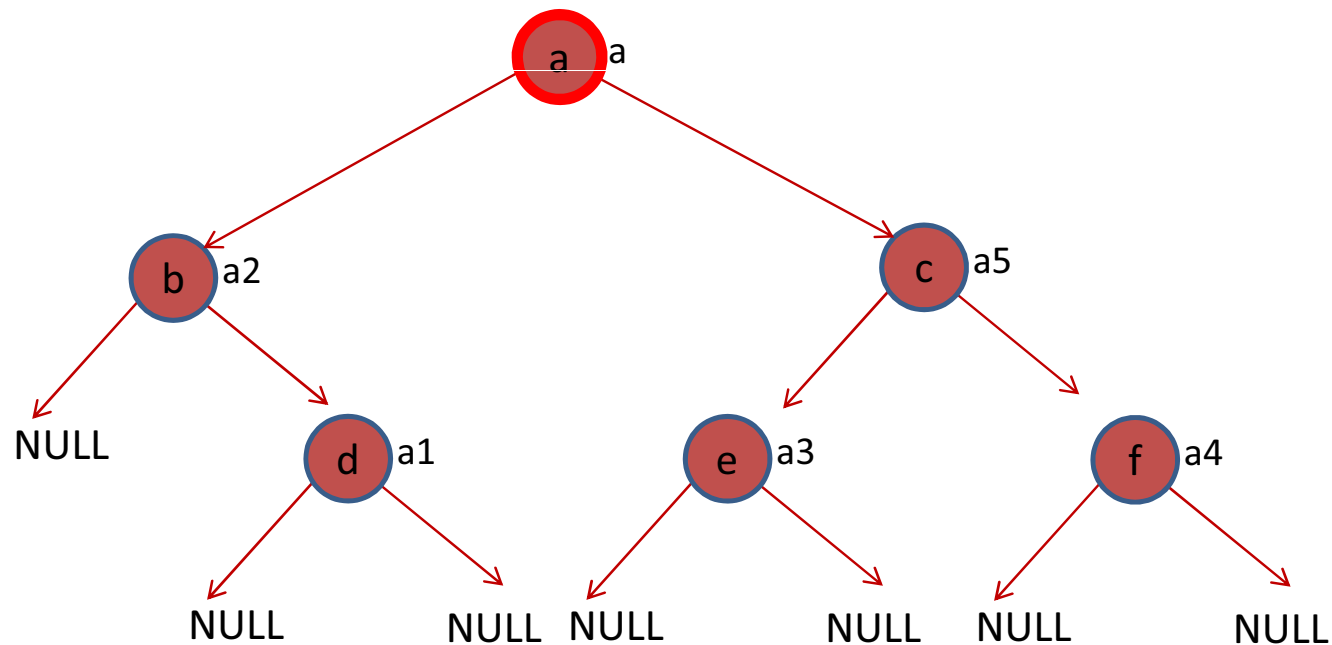
```
int x = arv_pertence (a, 'd');
```

```
int arv_pertence (Arv* a, char c)
{
    if (arv_vazia(a))
        return 0; /* árvore vazia: não encontrou */
    else
        return a->info==c || arv_pertence(a->esq,c) || arv_pertence(a->dir,c);
}
```



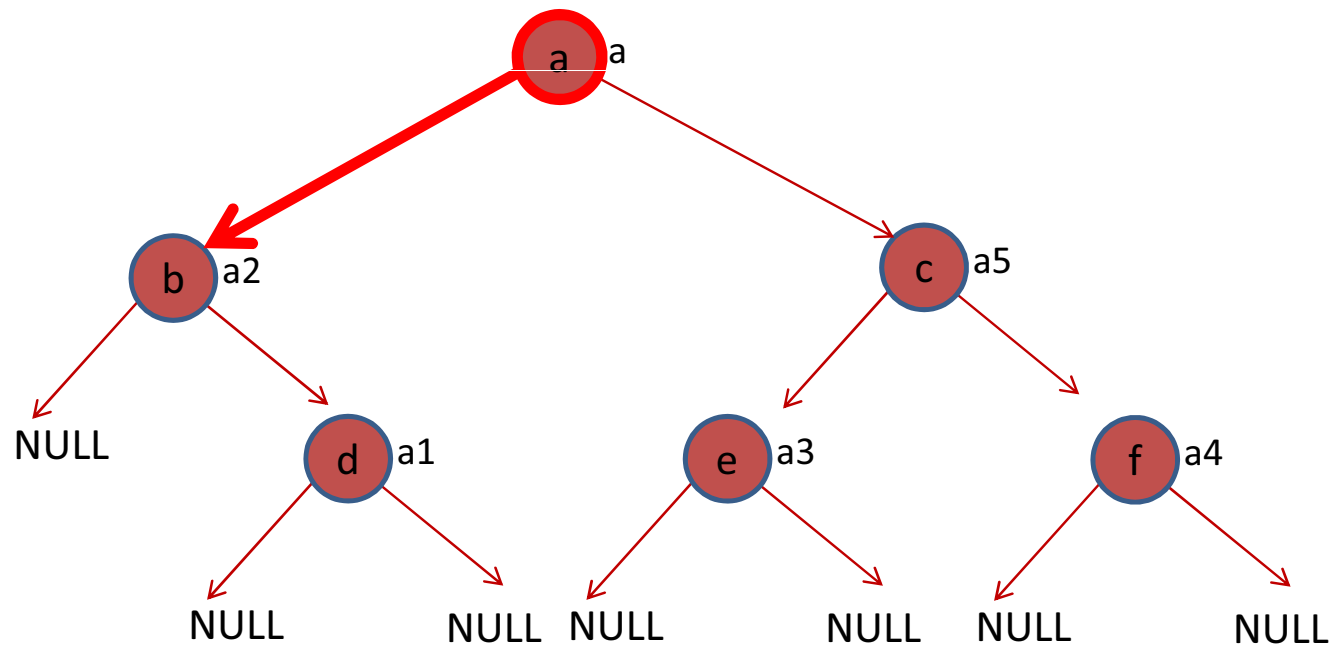
```
int x = arv_pertence (a, 'd');
```

```
int arv_pertence (Arv* a, char c)
{
    if (arv_vazia(a))
        return 0; /* árvore vazia: não encontrou */
    else
        return a->info==c || arv_pertence(a->esq,c) || arv_pertence(a->dir,c);
}
```



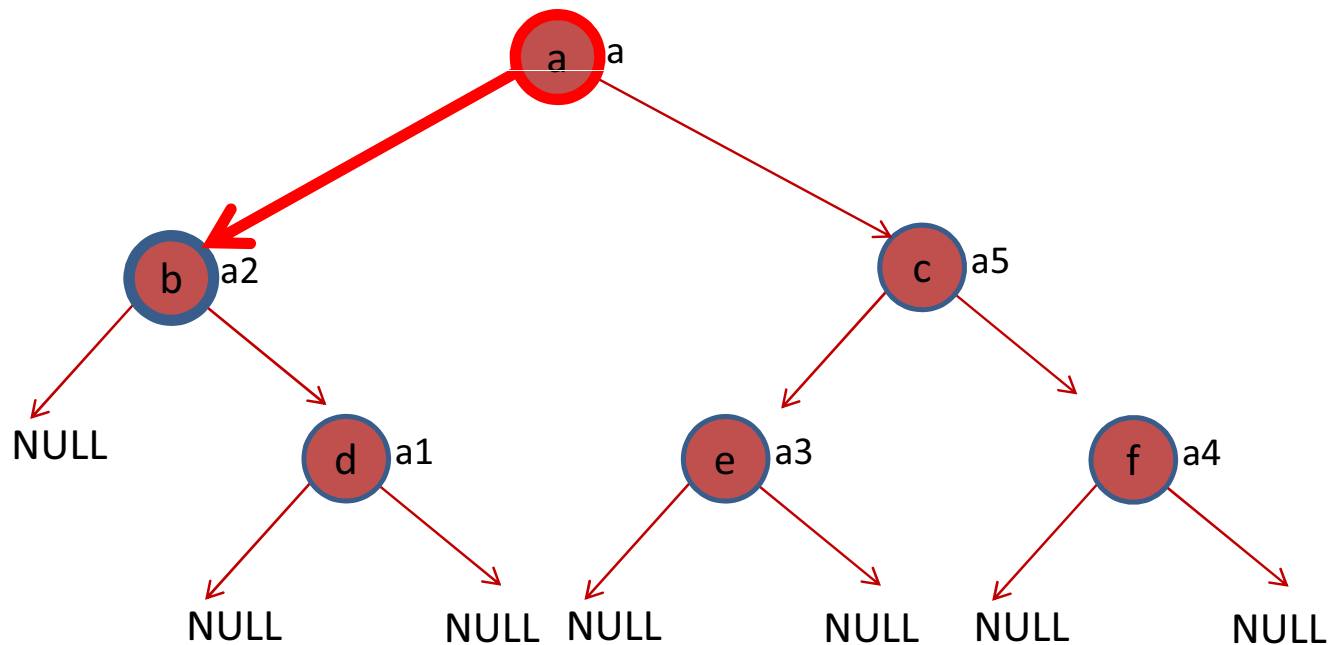
```
int x = arv_pertence (a, 'd');
```

```
int arv_pertence (Arv* a, char c)
{
    if (arv_vazia(a))
        return 0; /* árvore vazia: não encontrou */
    else
        return a->info==c || arv_pertence(a->esq,c) || arv_pertence(a->dir,c);
}
```



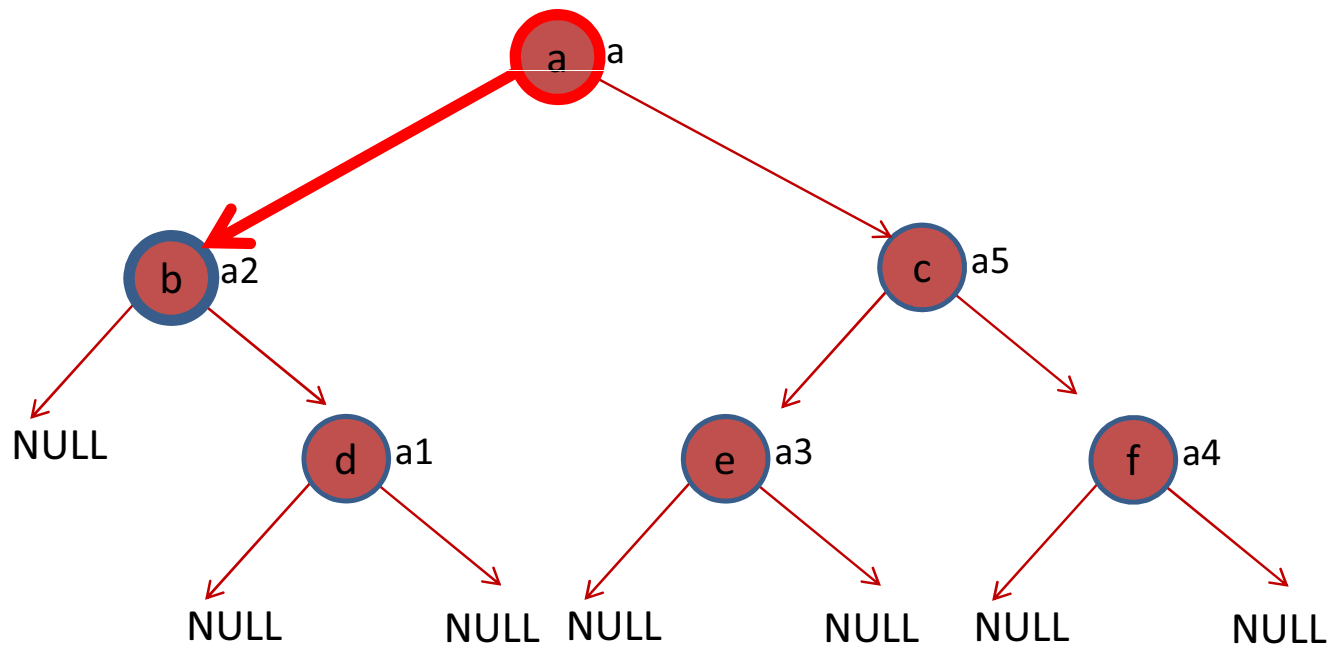
```
int x = arv_pertence (a2, 'd');
```

```
int arv_pertence (Arv* a, char c)
{
    if (arv_vazia(a))
        return 0; /* árvore vazia: não encontrou */
    else
        return a->info==c || arv_pertence(a->esq,c) || arv_pertence(a->dir,c);
}
```



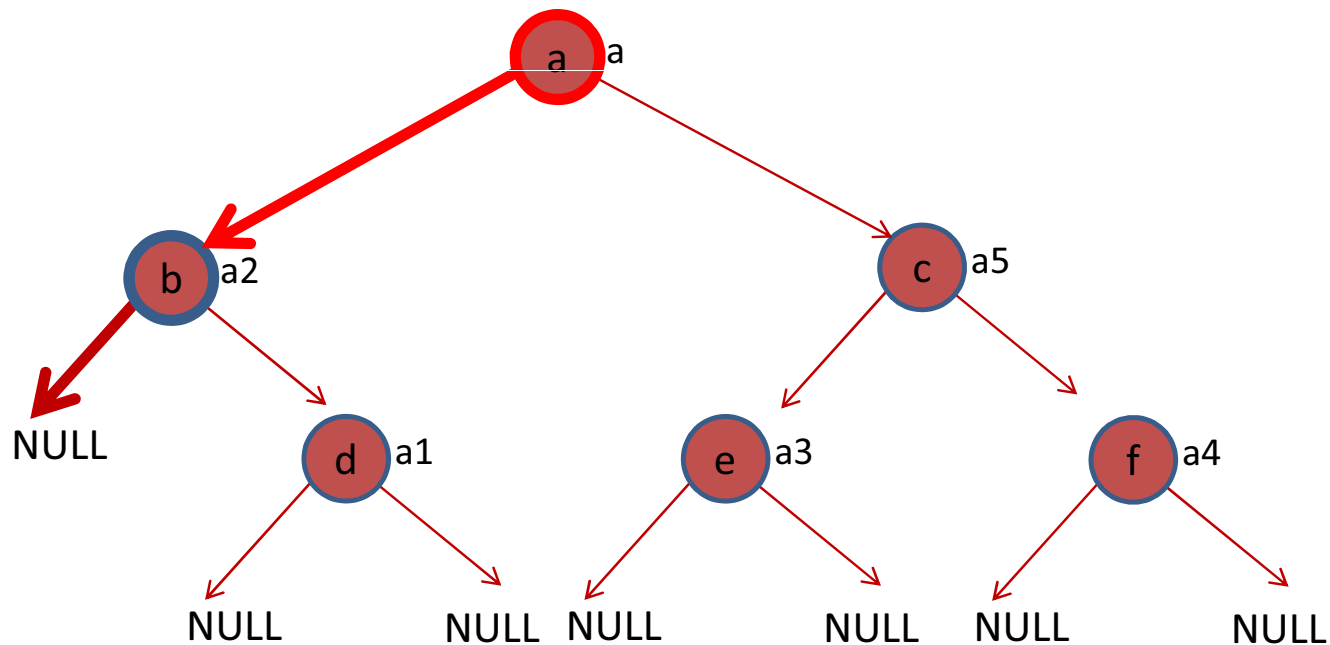
```
int x = arv_pertence (a2, 'd');
```

```
int arv_pertence (Arv* a, char c)
{
    if (arv_vazia(a))
        return 0; /* árvore vazia: não encontrou */
    else
        return a->info==c || arv_pertence(a->esq,c) || arv_pertence(a->dir,c);
}
```



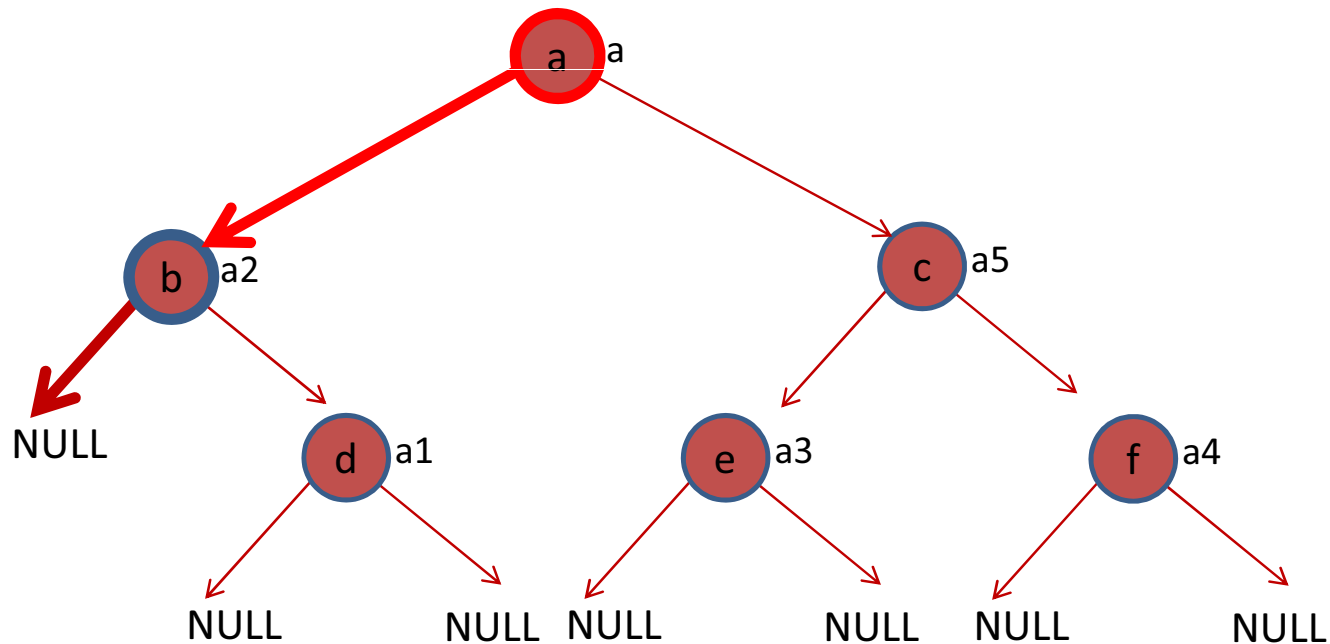
```
int x = arv_pertence (a2, 'd');
```

```
int arv_pertence (Arv* a, char c)
{
    if (arv_vazia(a))
        return 0; /* árvore vazia: não encontrou */
    else
        return a->info==c || arv_pertence(a->esq,c) || arv_pertence(a->dir,c);
}
```



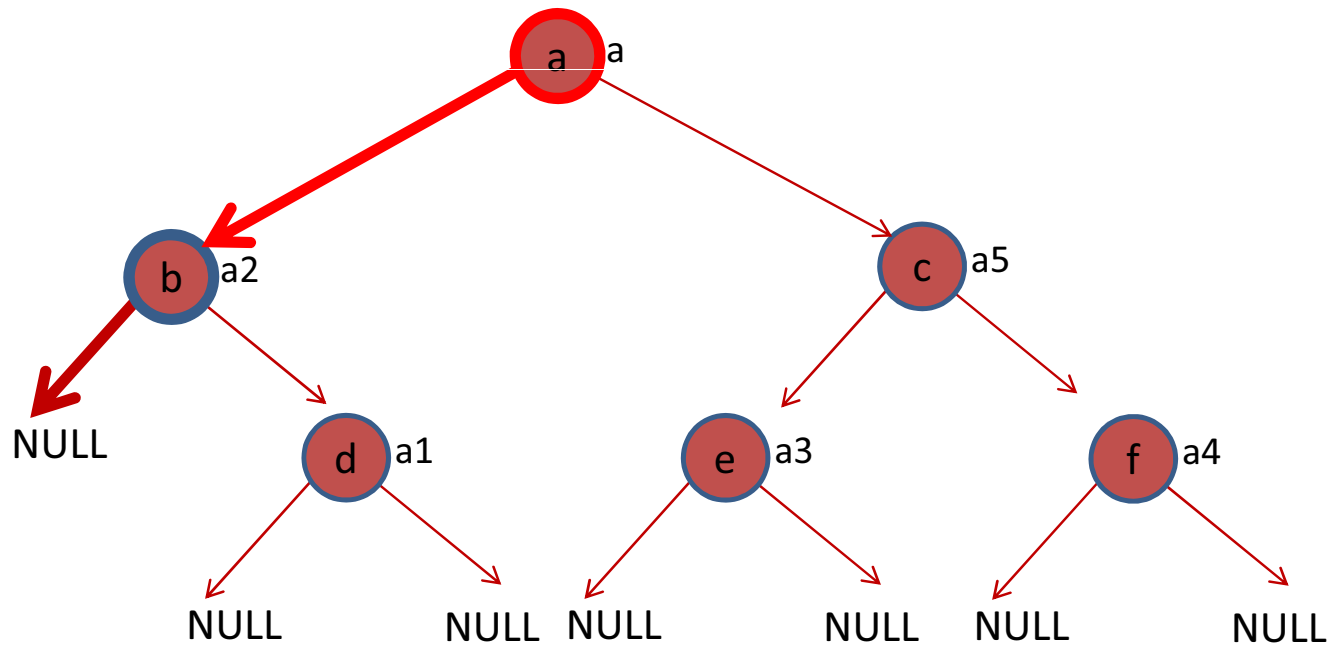

```
int x = arv_pertence (NULL, 'd');
```

```
int arv_pertence (Arv* a, char c)
{
    if (arv_vazia(a))
        return 0; /* árvore vazia: não encontrou */
    else
        return a->info==c || arv_pertence(a->esq,c) || arv_pertence(a->dir,c);
}
```



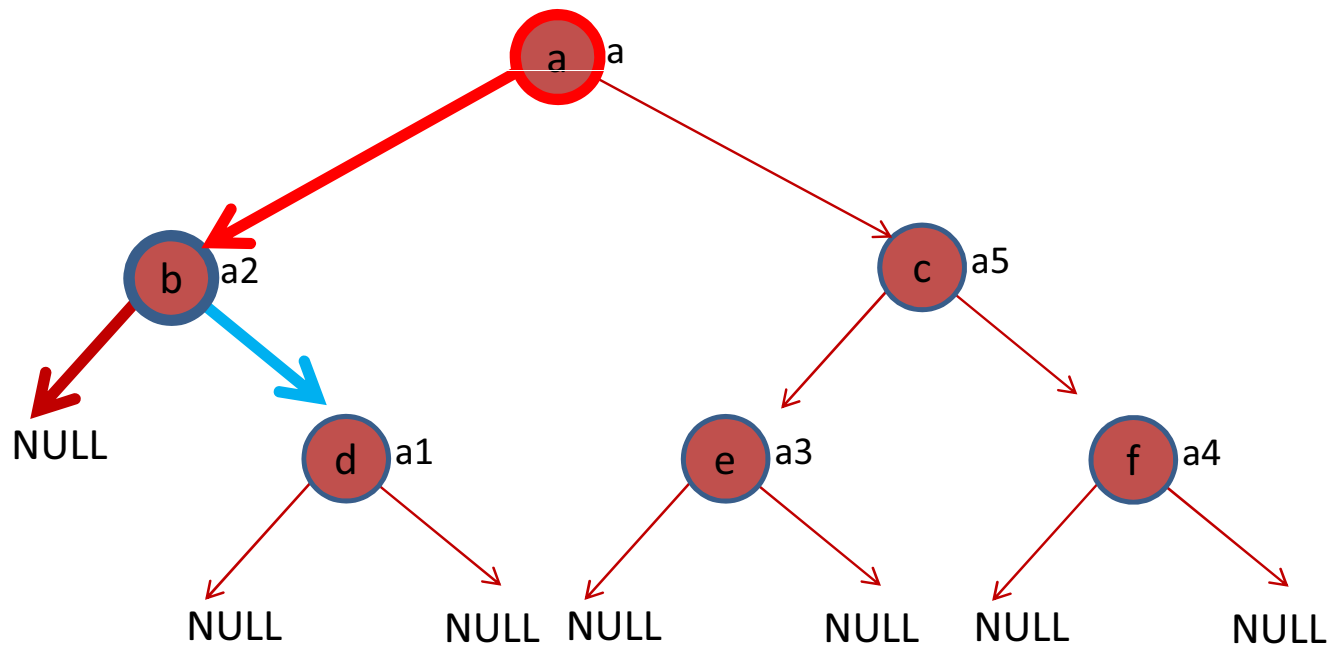
```
int x = arv_pertence (NULL, 'd');
```

```
int arv_pertence (Arv* a, char c)
{
    if (arv_vazia(a))
        return 0; /* árvore vazia: não encontrou */
    else
        return a->info==c || arv_pertence(a->esq,c) || arv_pertence(a->dir,c);
}
```



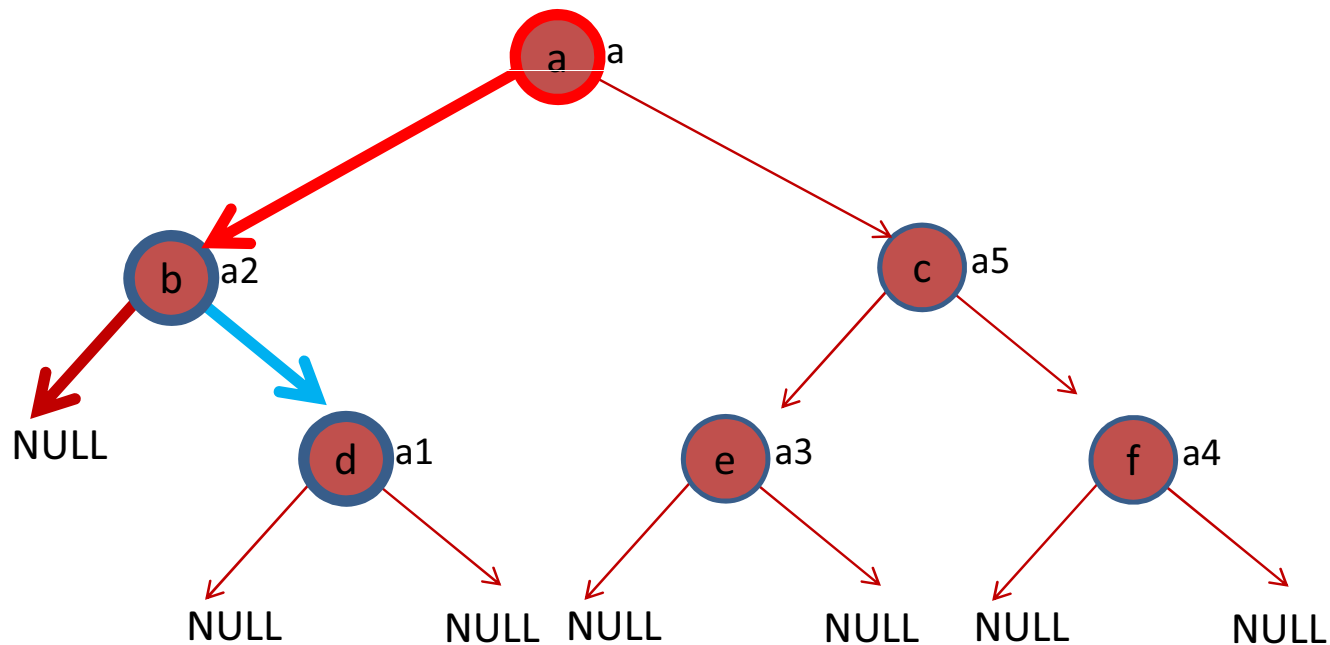
`int x = arv_pertence (a2, 'd');`

```
int arv_pertence (Arv* a, char c)
{
    if (arv_vazia(a))
        return 0; /* árvore vazia: não encontrou */
    else
        return a->info==c || arv_pertence(a->esq,c) || arv_pertence(a->dir,c);
}
```



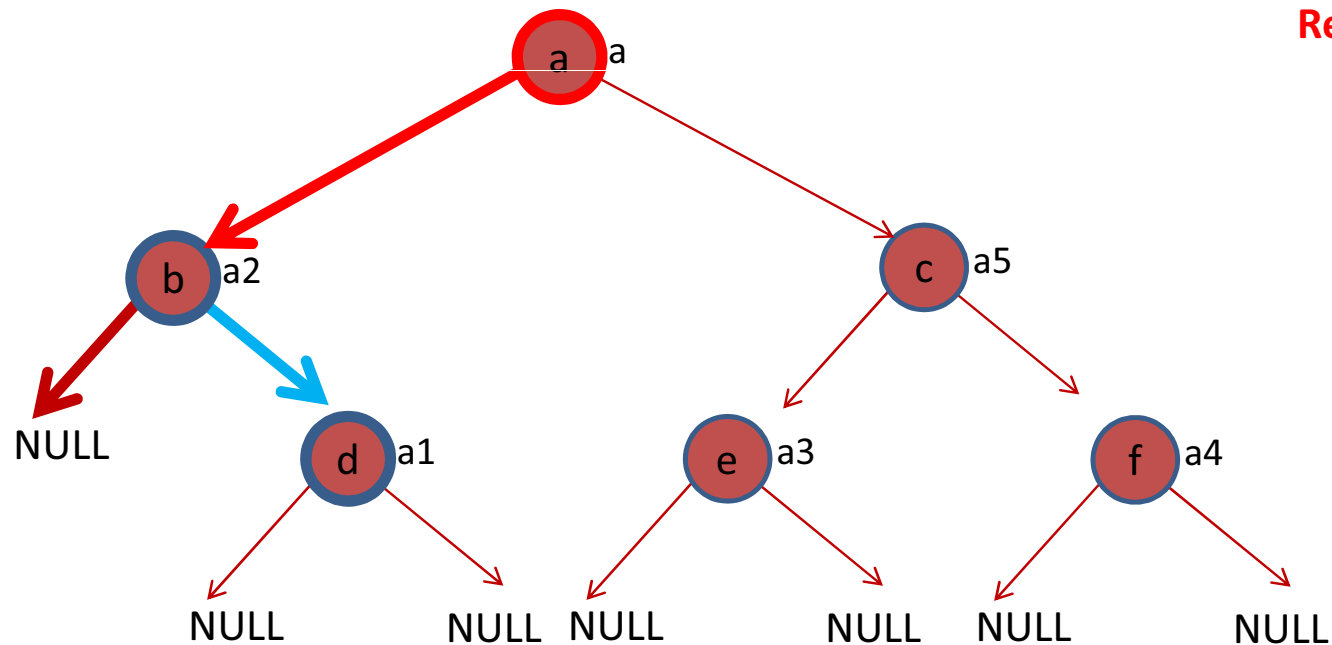
```
int x = arv_pertence (a1, 'd');
```

```
int arv_pertence (Arv* a, char c)
{
    if (arv_vazia(a))
        return 0; /* árvore vazia: não encontrou */
    else
        return a->info==c || arv_pertence(a->esq,c) || arv_pertence(a->dir,c);
}
```



`int x = arv_pertence (a1, 'd');`

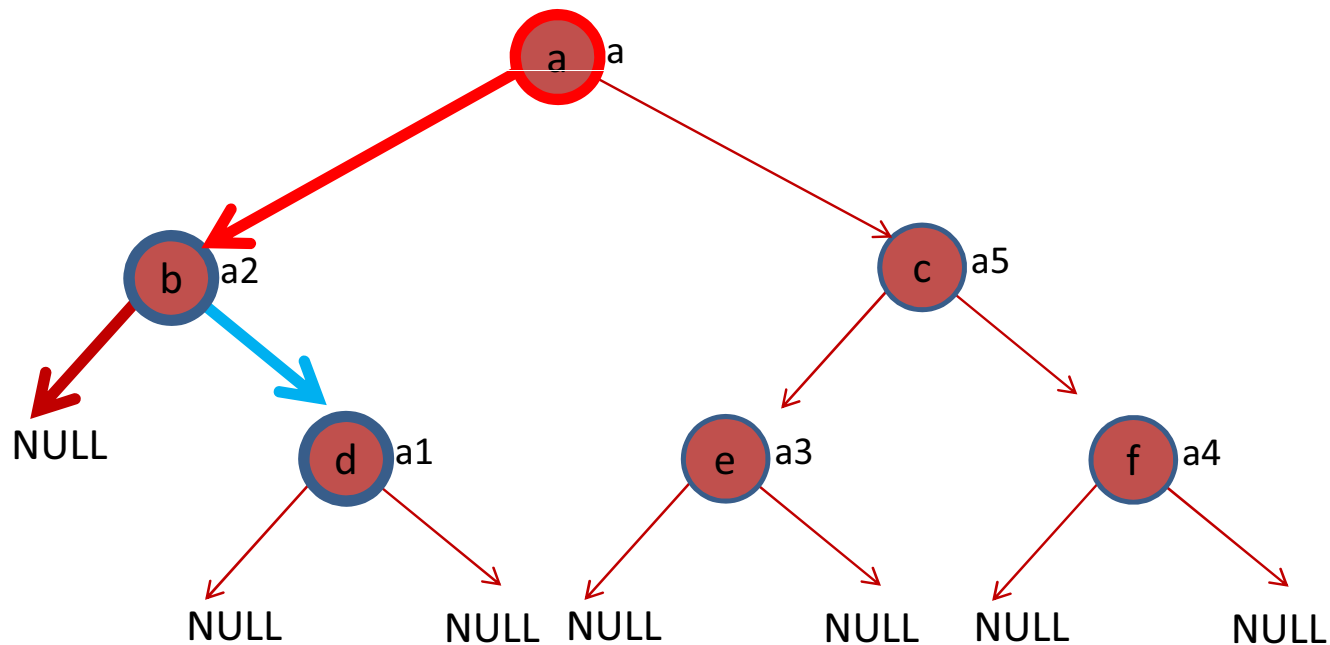
```
int arv_pertence (Arv* a, char c)
{
    if (arv_vazia(a))
        return 0; /* árvore vazia: não encontrou */
    else
        return a->info==c || arv_pertence(a->esq,c) || arv_pertence(a->dir,c);
}
```



```
int x = arv_pertence (a2, 'd');
```

```
int arv_pertence (Arv* a, char c)
{
    if (arv_vazia(a))
        return 0; /* árvore vazia: não encontrou */
    else
        return a->info==c || arv_pertence(a->esq,c) || arv_pertence(a->dir,c);
}
```

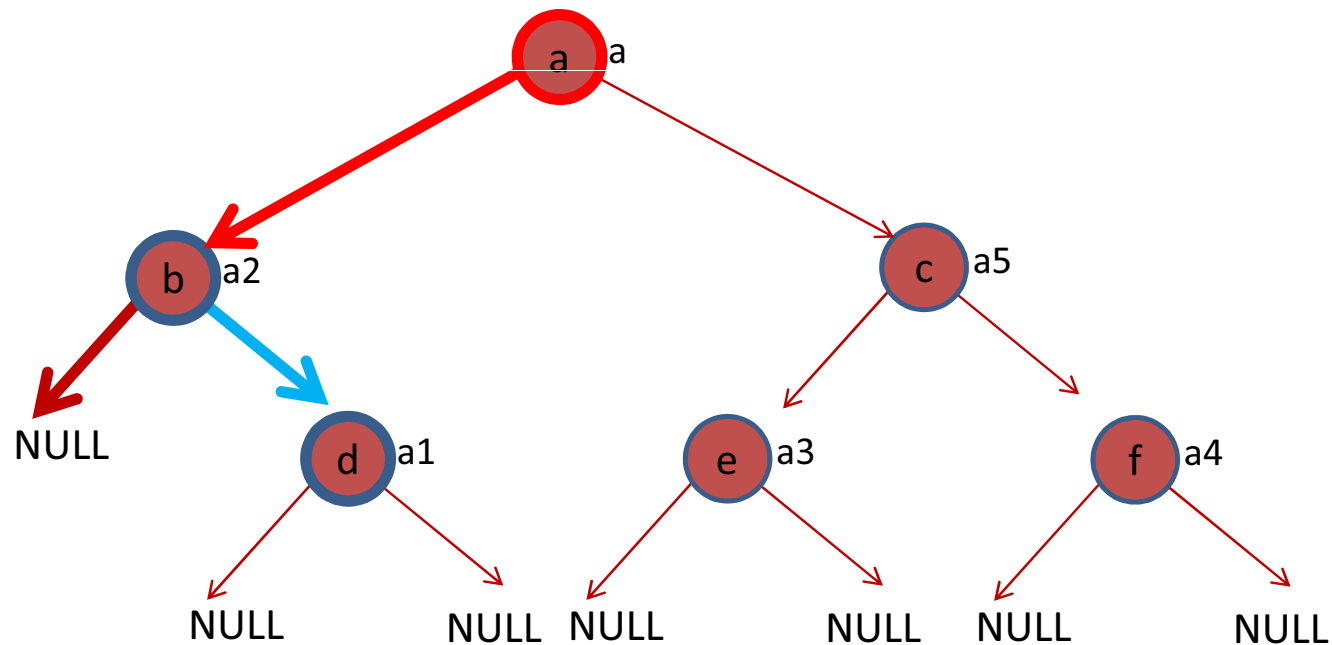
Retorna 1



```
int x = arv_pertence (a, 'd');
```

```
int arv_pertence (Arv* a, char c)
{
  if (arv_vazia(a))
    return 0; /* árvore vazia: não encontrou */
  else
    return a->info==c || arv_pertence(a->esq,c) || arv_pertence(a->dir,c);
}
```

Retorna 1 para o main



Árvores binárias - Implementação em C

- função arv_imprime
 - percorre recursivamente a árvore, visitando todos os nós e imprimindo sua informação

```
void arv_imprime (Arv* a)
{
    if (!arv_vazia(a)){
        printf("%c ", a->info);           /* mostra raiz */
        arv_imprime(a->esq);              /* mostra sae */
        arv_imprime(a->dir);              /* mostra sad */
    }
}
```



```
struct arv {  
    char info;  
    struct arv* esq;  
    struct arv* dir;  
};  
typedef struct arv Arv;
```

```
Arv* arv_criavazia (void);  
Arv* arv_cria (char c, Arv* e, Arv* d);  
Arv* arv_libera (Arv* a);  
int arv_vazia (Arv* a);  
int arv_pertence (Arv* a, char c);  
void arv_imprime (Arv* a);
```

```
void arv_imprime (Arv* a)  
{  
    if (!arv_vazia(a))  
    {  
        printf("%c ", a->info); /* mostra raiz */  
        arv_imprime(a->esq); /* mostra sae */  
        arv_imprime(a->dir); /* mostra sad */  
    }  
}
```

```

#i #include <stdio.h>
#i #include <stdlib.h>
#i #include "arvore.h"


in int main()
{
    Arv* a1= arv_cria('d',arv_criavazia(),arv_criavazia());
    Arv* a2= arv_cria('b',arv_criavazia(),a1);
    Arv* a3= arv_cria('e',arv_criavazia(),arv_criavazia());
    Arv* a4= arv_cria('f',arv_criavazia(),arv_criavazia());
    Arv* a5= arv_cria('c',a3,a4);
    Arv* a = arv_cria('a',a2,a5 );

    arv_imprime (a);

    a = arv_libera (a);

    system("PAUSE");
    return 1;
}

```



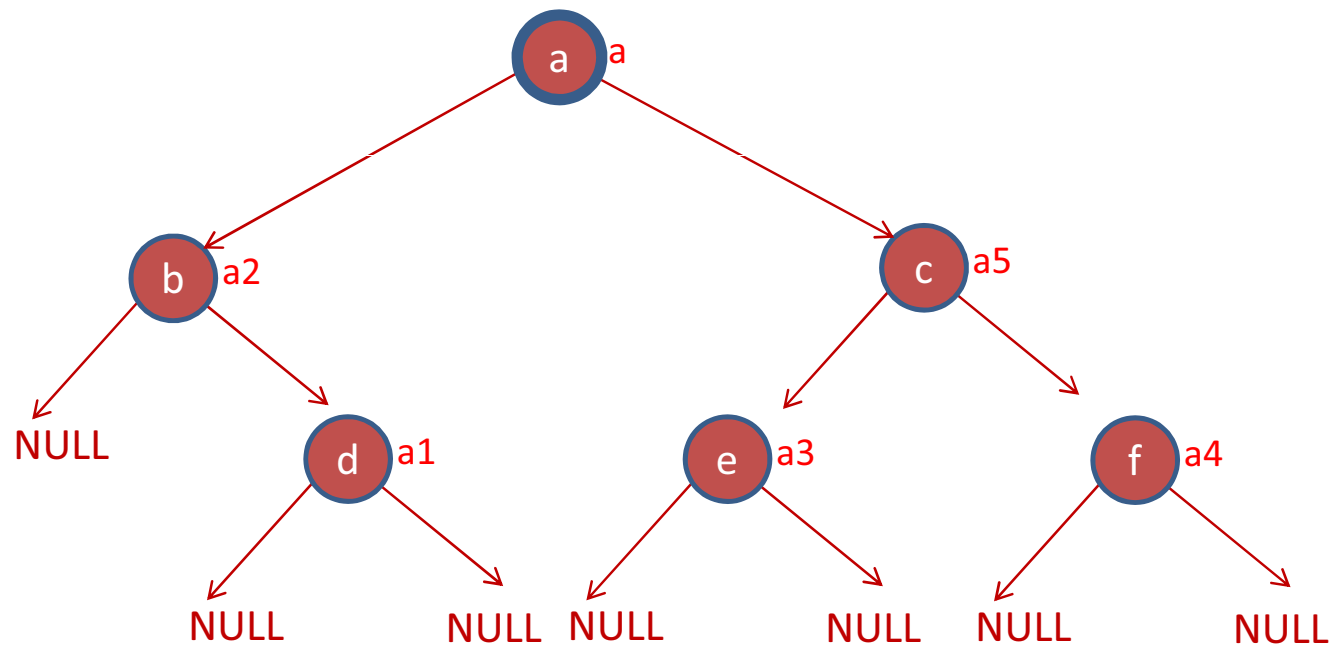
```

C:\Documents and Settings\eduardo\Ambiente de trabalho\DEISE\20101\ED -
a b d c e f Prima qualquer tecla para continuar . . .

```

```
void arv_imprime (Arv* a)
{
    if (!arv_vazia(a))
    {
        printf("%c ", a->info);
        arv_imprime(a->esq);
        arv_imprime(a->dir);
    }
}
```

arv_imprime (a);

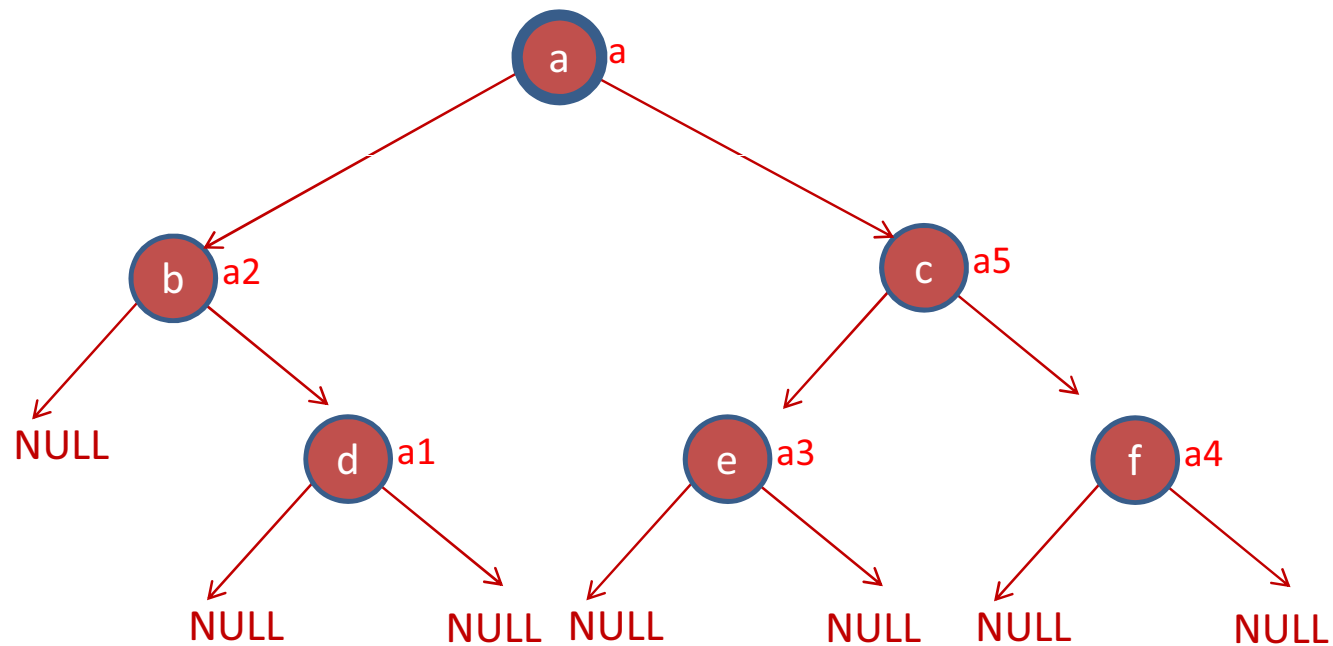


```

void arv_imprime (Arv* a)
{
  if (!arv_vazia(a))
  {
    printf("%c ", a->info);
    arv_imprime(a->esq);
    arv_imprime(a->dir);
  }
}

```

arv_imprime (a);



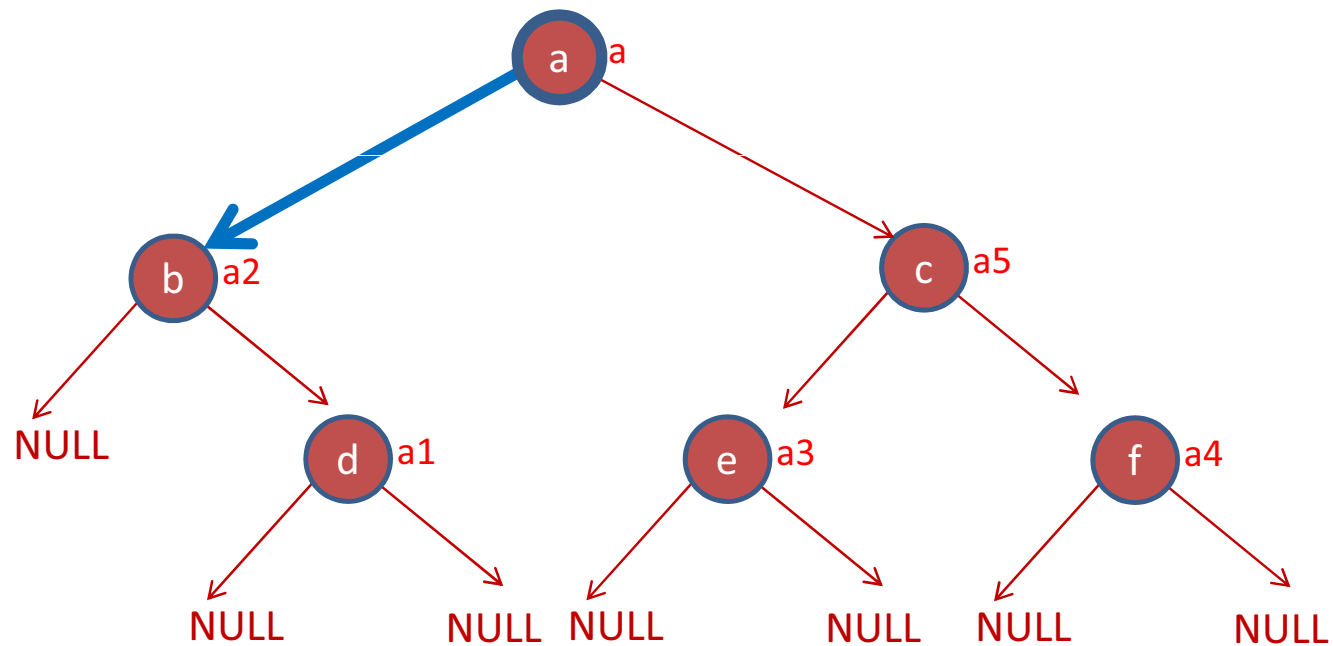
a

```

void arv_imprime (Arv* a)
{
  if (!arv_vazia(a))
  {
    printf("%c ", a->info);
    arv_imprime(a->esq);
    arv_imprime(a->dir);
  }
}

```

arv_imprime (a);



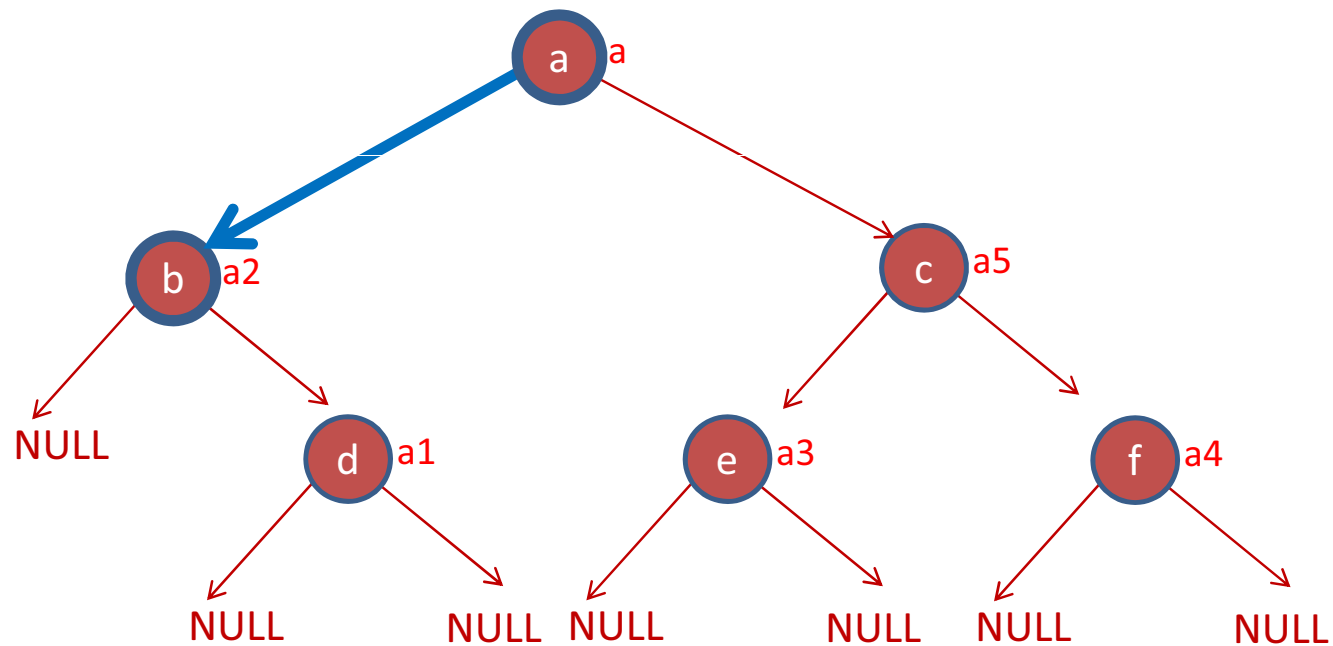
a

```

void arv_imprime (Arv* a)
{
    if (!arv_vazia(a))
    {
        printf("%c ", a->info);
        arv_imprime(a->esq);
        arv_imprime(a->dir);
    }
}

```

arv_imprime (a2);



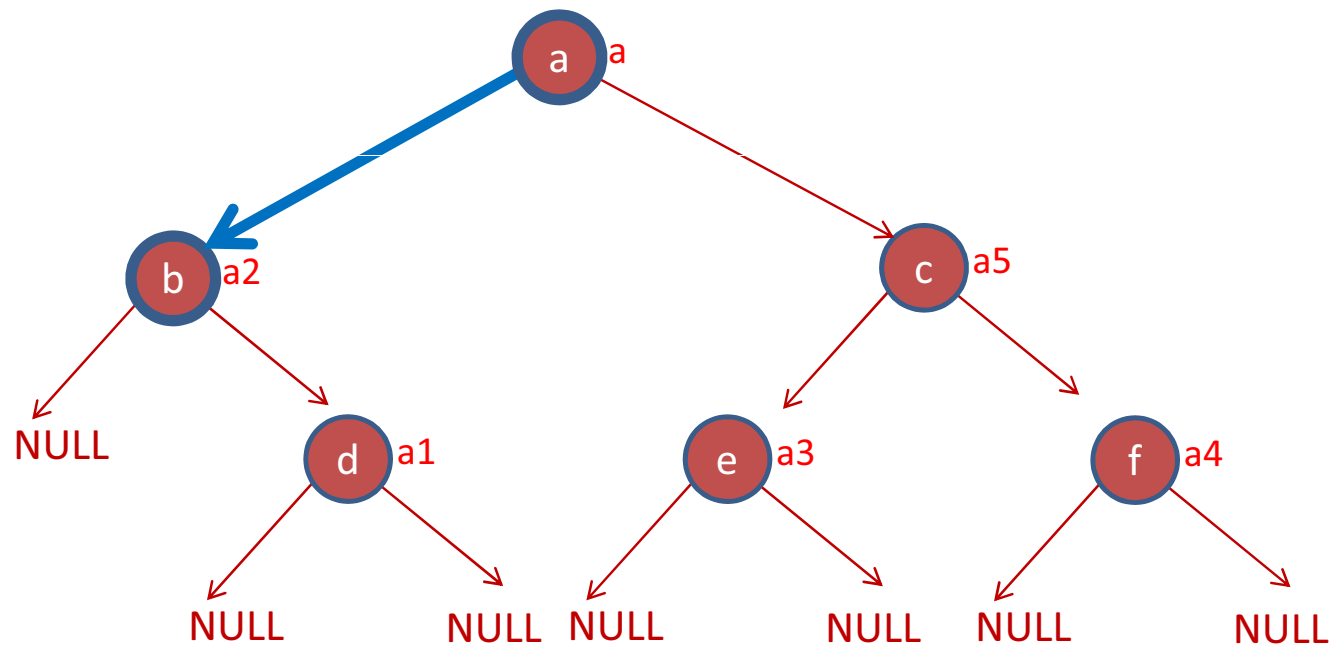
a

```

void arv_imprime (Arv* a)
{
    if (!arv_vazia(a))
    {
        printf("%c ", a->info);
        arv_imprime(a->esq);
        arv_imprime(a->dir);
    }
}

```

arv_imprime (a2);



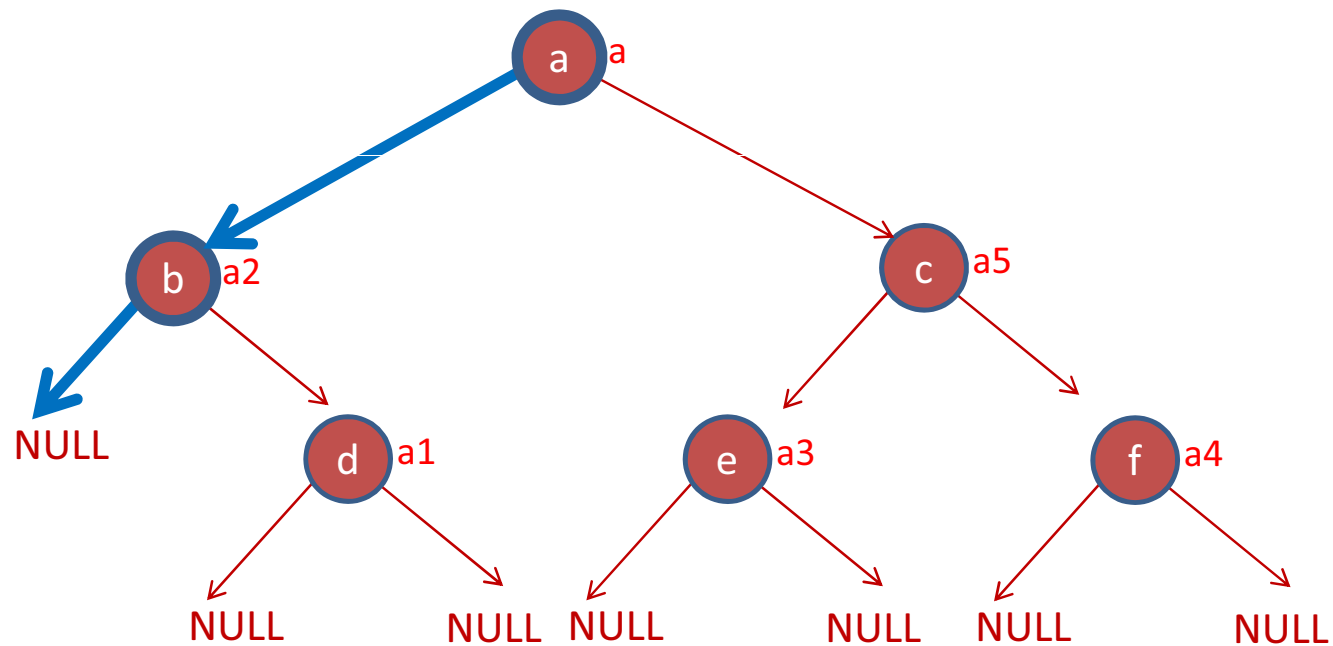
a b

```

void arv_imprime (Arv* a)
{
    if (!arv_vazia(a))
    {
        printf("%c ", a->info);
        arv_imprime(a->esq);
        arv_imprime(a->dir);
    }
}

```

arv_imprime (a2);



a b

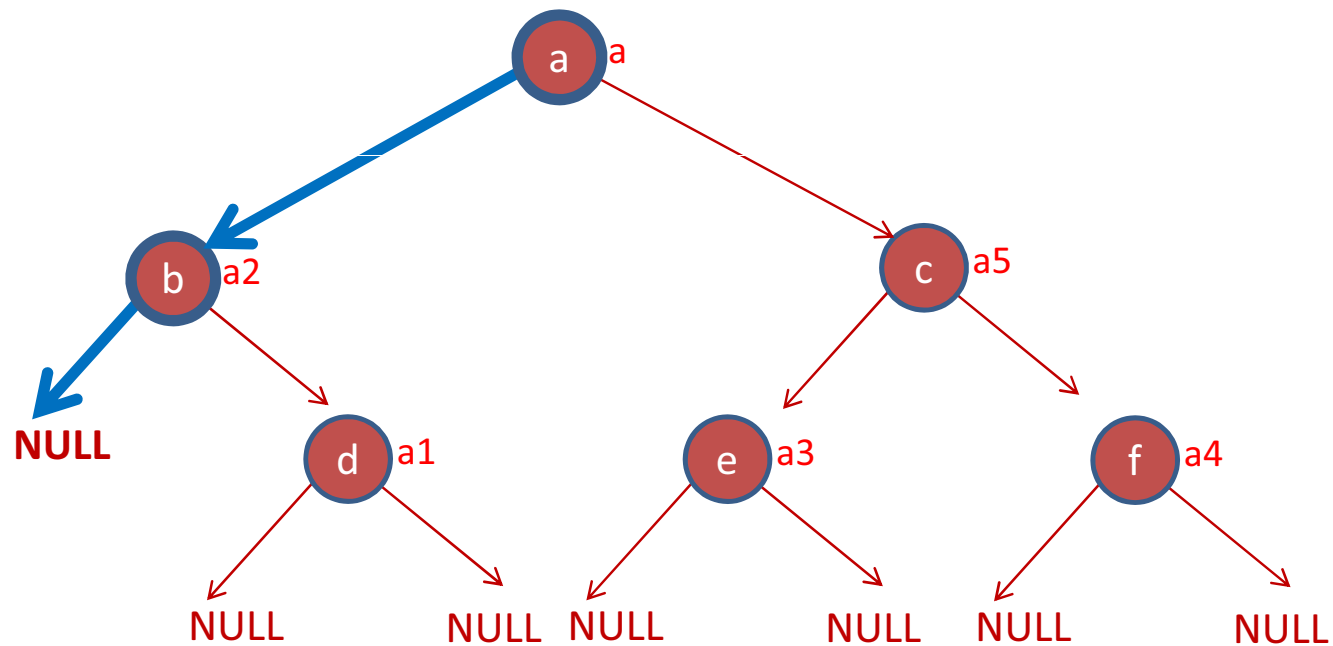

```

void arv_imprime (Arv* a)
{
    if (!arv_vazia(a))
    {
        printf("%c ", a->info);
        arv_imprime(a->esq);
        arv_imprime(a->dir);
    }
}

```

arv_imprime (NULL);

Retorna



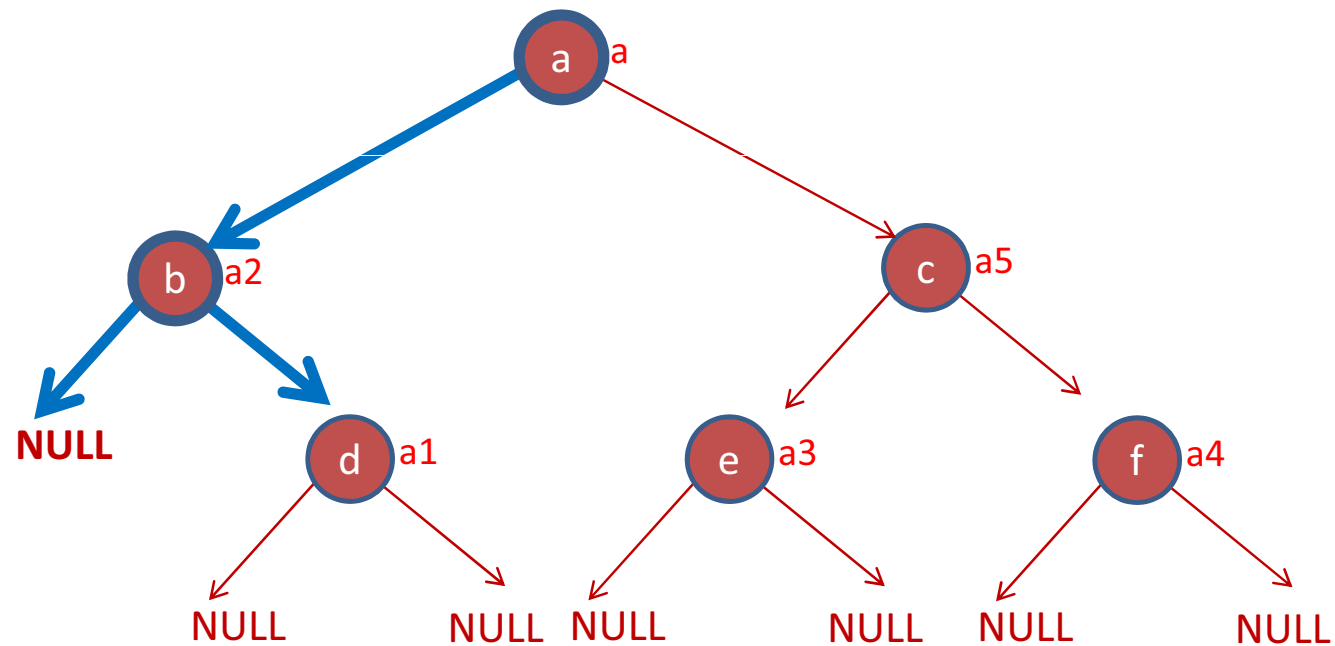
a b

```

void arv_imprime (Arv* a)
{
    if (!arv_vazia(a))
    {
        printf("%c ", a->info);
        arv_imprime(a->esq);
        arv_imprime(a->dir);
    }
}

```

arv_imprime (a2);



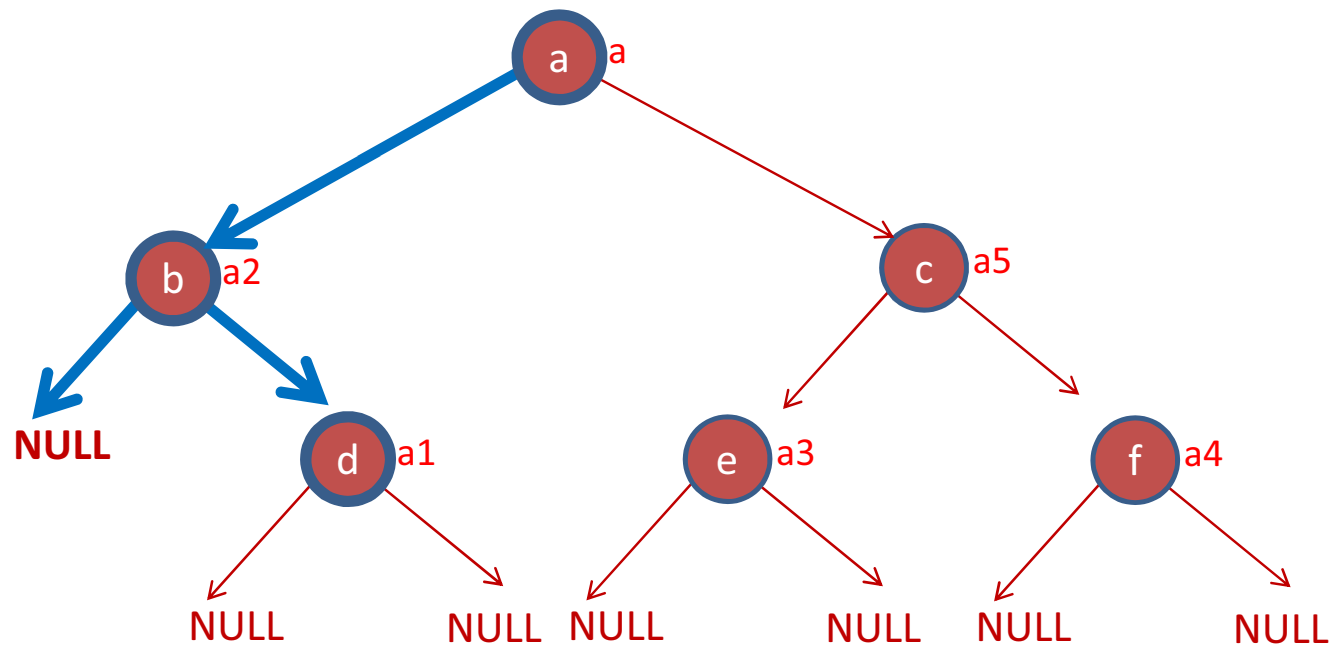
a b

```

void arv_imprime (Arv* a)
{
    if (!arv_vazia(a))
    {
        printf("%c ", a->info);
        arv_imprime(a->esq);
        arv_imprime(a->dir);
    }
}

```

arv_imprime (a1);



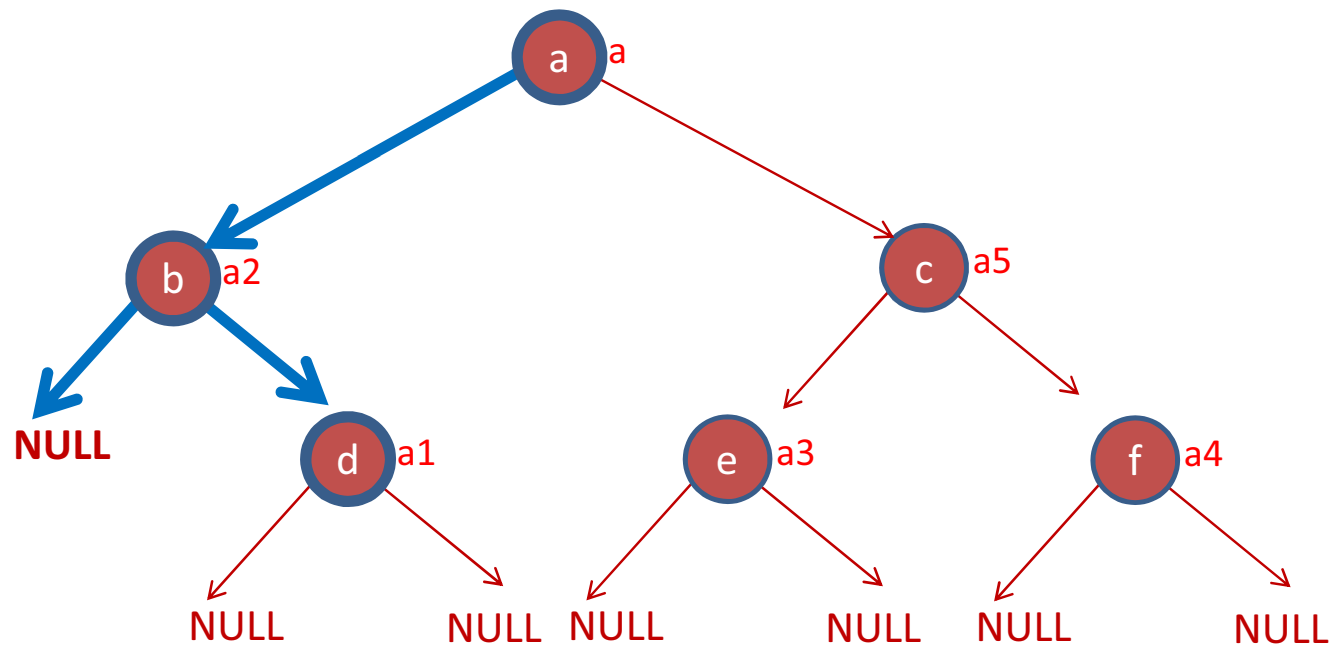
a b

```

void arv_imprime (Arv* a)
{
    if (!arv_vazia(a))
    {
        printf("%c ", a->info);
        arv_imprime(a->esq);
        arv_imprime(a->dir);
    }
}

```

arv_imprime (a1);



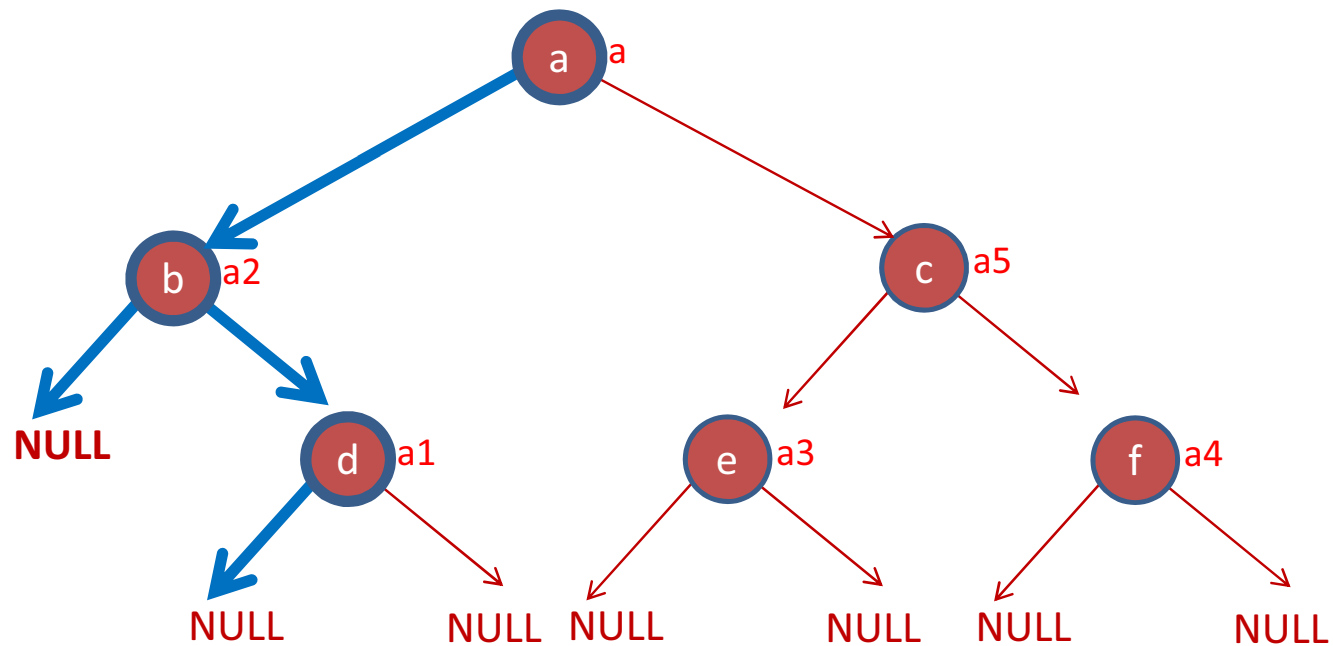
a b d

```

void arv_imprime (Arv* a)
{
    if (!arv_vazia(a))
    {
        printf("%c ", a->info);
        arv_imprime(a->esq);
        arv_imprime(a->dir);
    }
}

```

arv_imprime (a1);



a b d

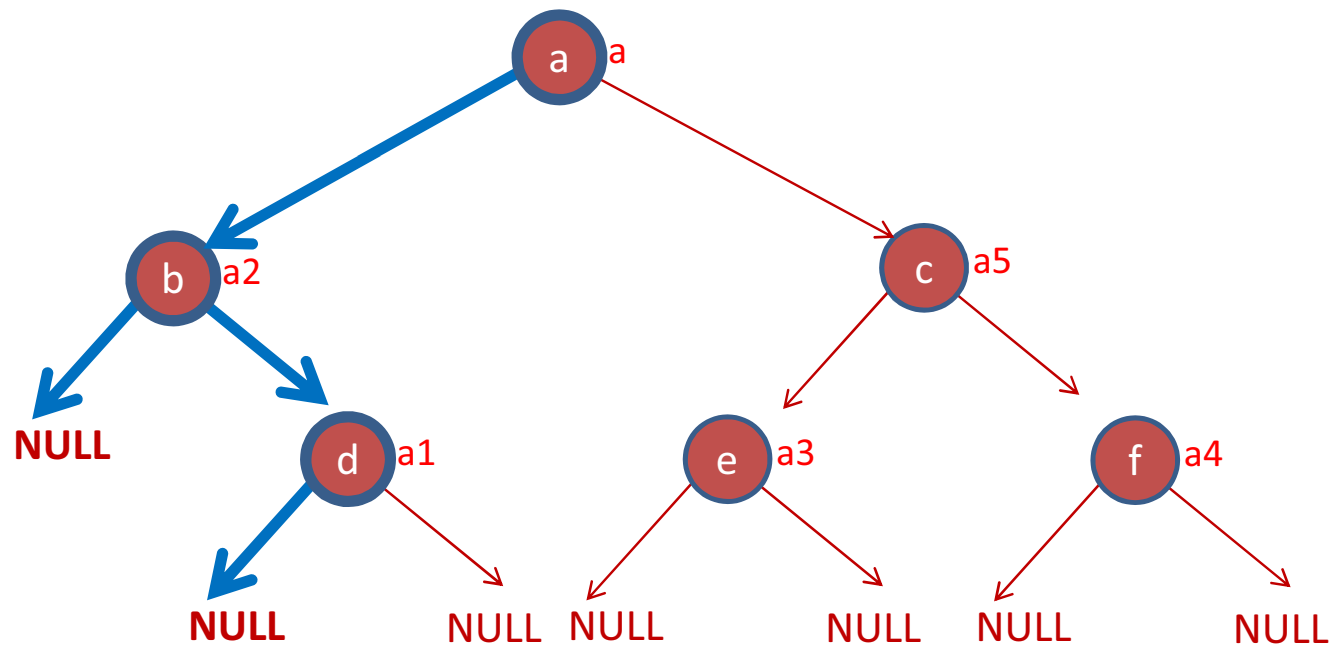
```

void arv_imprime (Arv* a)
{
    if (!arv_vazia(a))
    {
        printf("%c ", a->info);
        arv_imprime(a->esq);
        arv_imprime(a->dir);
    }
}

```

arv_imprime (NULL);

Retorna



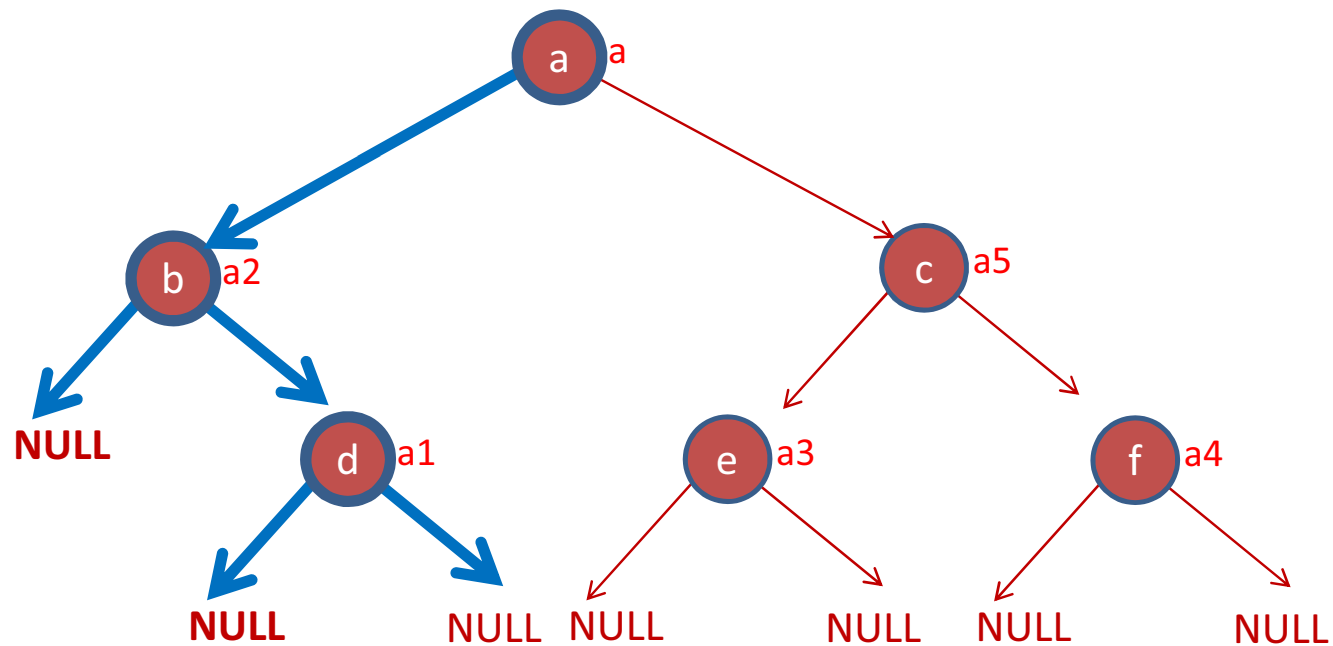
a b d

```

void arv_imprime (Arv* a)
{
    if (!arv_vazia(a))
    {
        printf("%c ", a->info);
        arv_imprime(a->esq);
        arv_imprime(a->dir);
    }
}

```

arv_imprime (a1);



a b d

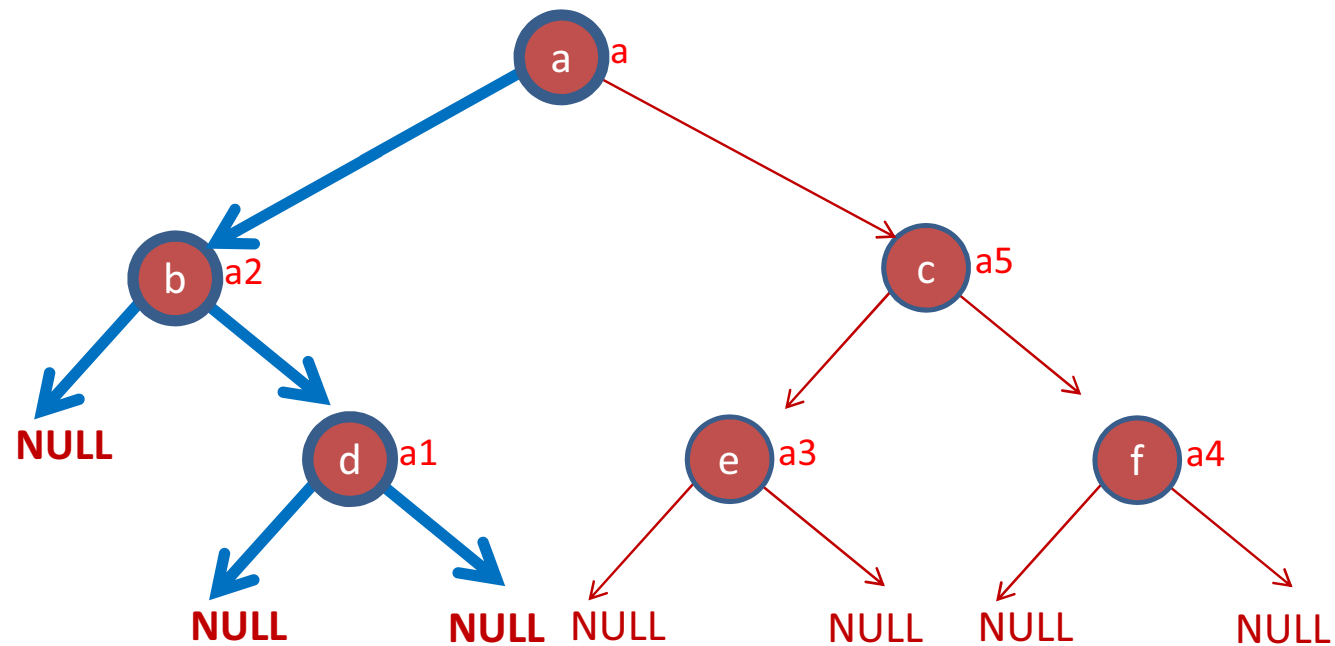
```

void arv_imprime (Arv* a)
{
    if (!arv_vazia(a))
    {
        printf("%c ", a->info);
        arv_imprime(a->esq);
        arv_imprime(a->dir);
    }
}

```

arv_imprime (NULL);

Retorna



a b d

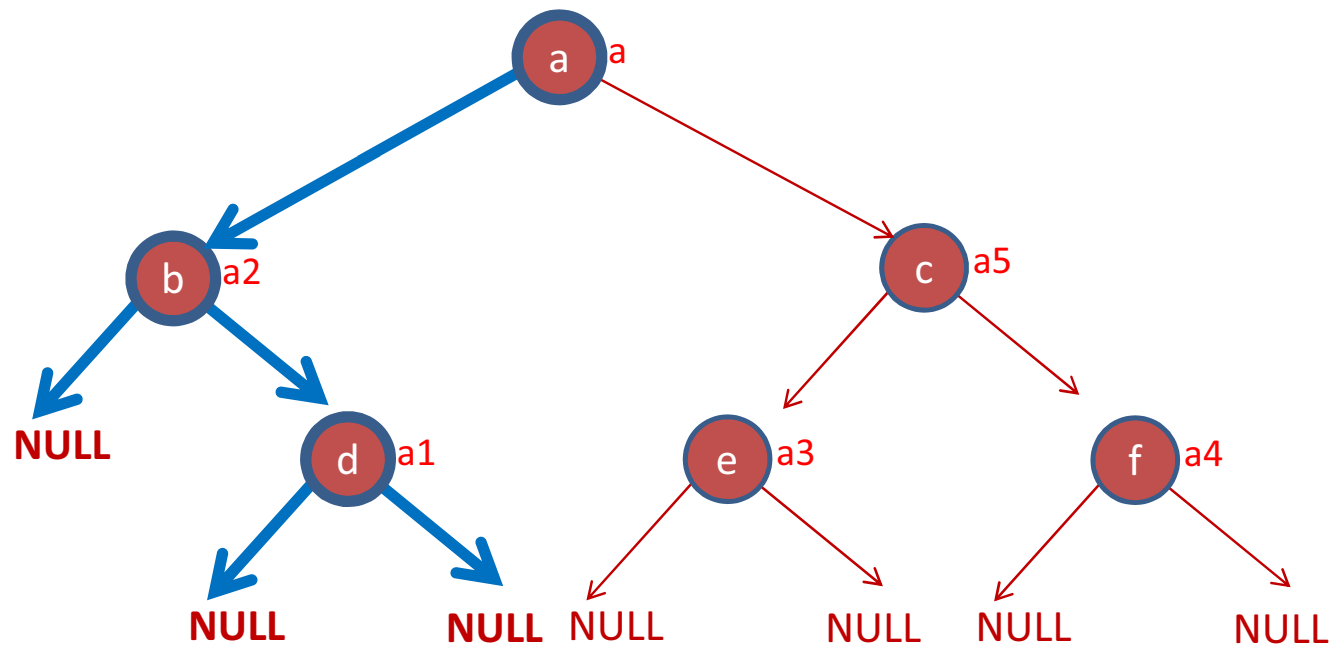

```

void arv_imprime (Arv* a)
{
    if (!arv_vazia(a))
    {
        printf("%c ", a->info);
        arv_imprime(a->esq);
        arv_imprime(a->dir);
    }
}

```

arv_imprime (a1);

Retorna



a b d

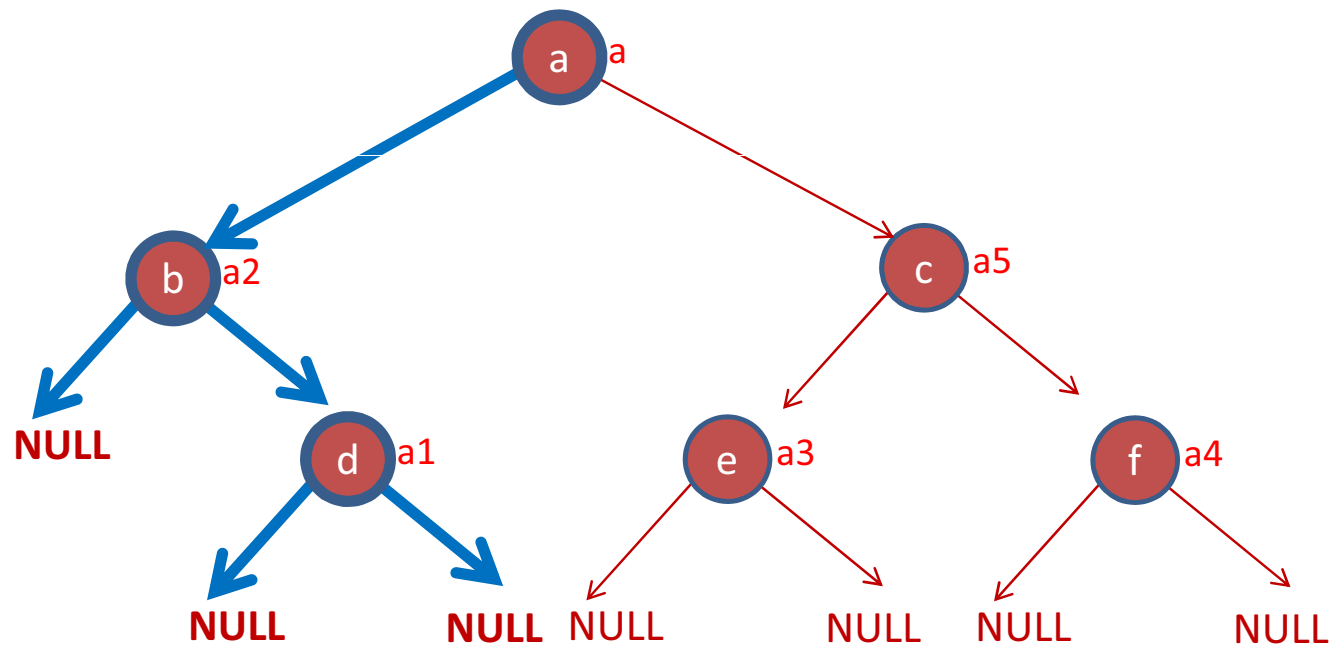
```

void arv_imprime (Arv* a)
{
    if (!arv_vazia(a))
    {
        printf("%c ", a->info);
        arv_imprime(a->esq);
        arv_imprime(a->dir);
    }
}

```

arv_imprime (a2);

Retorna



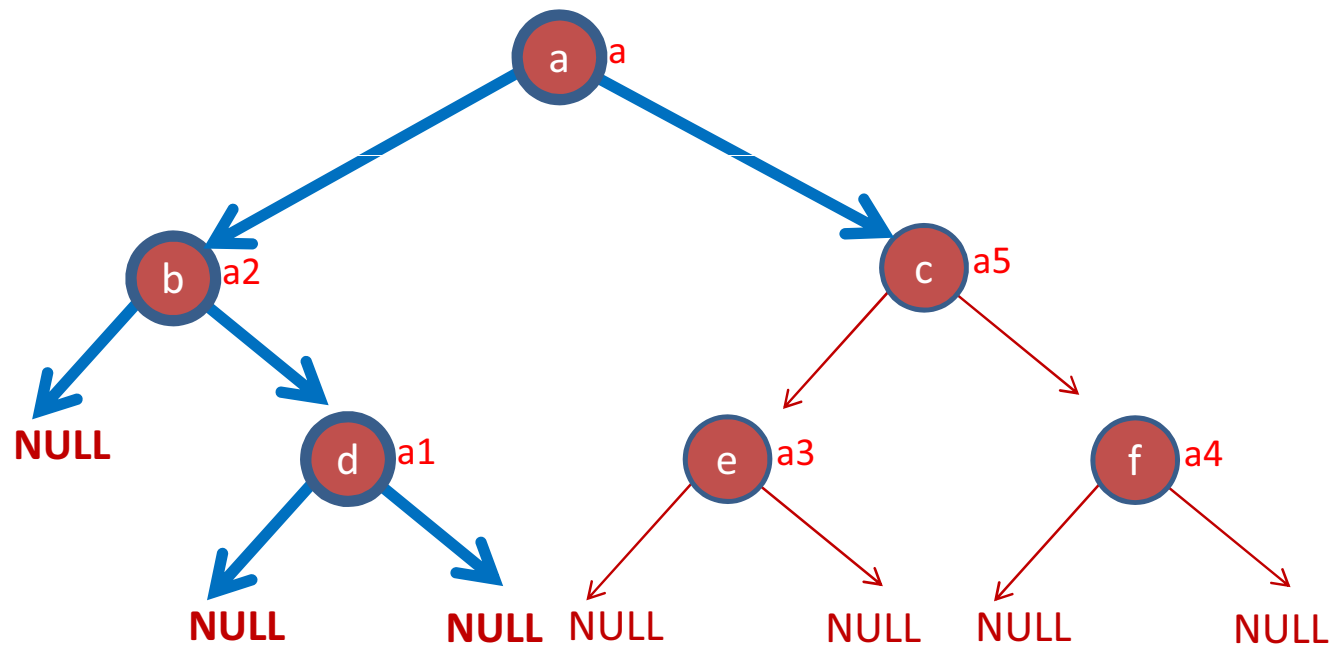
a b d

```

void arv_imprime (Arv* a)
{
  if (!arv_vazia(a))
  {
    printf("%c ", a->info);
    arv_imprime(a->esq);
    arv_imprime(a->dir);
  }
}

```

arv_imprime (a);



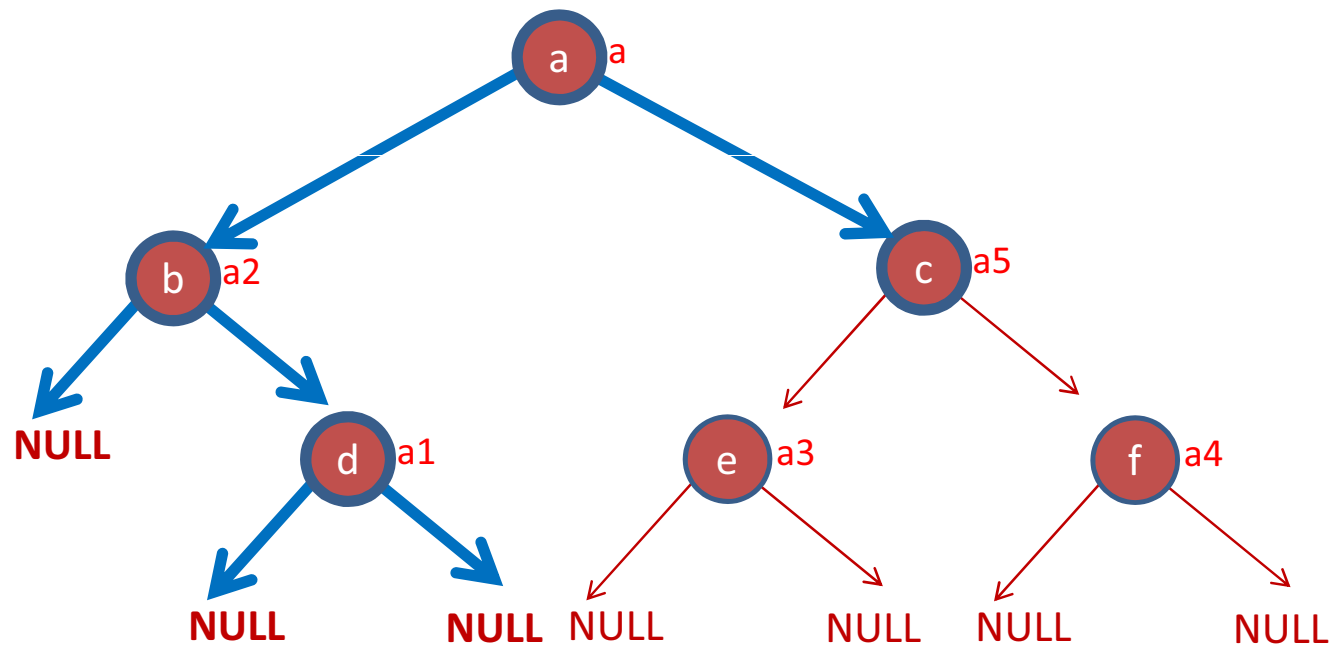
a b d

```

void arv_imprime (Arv* a)
{
    if (!arv_vazia(a))
    {
        printf("%c ", a->info);
        arv_imprime(a->esq);
        arv_imprime(a->dir);
    }
}

```

arv_imprime (a5);



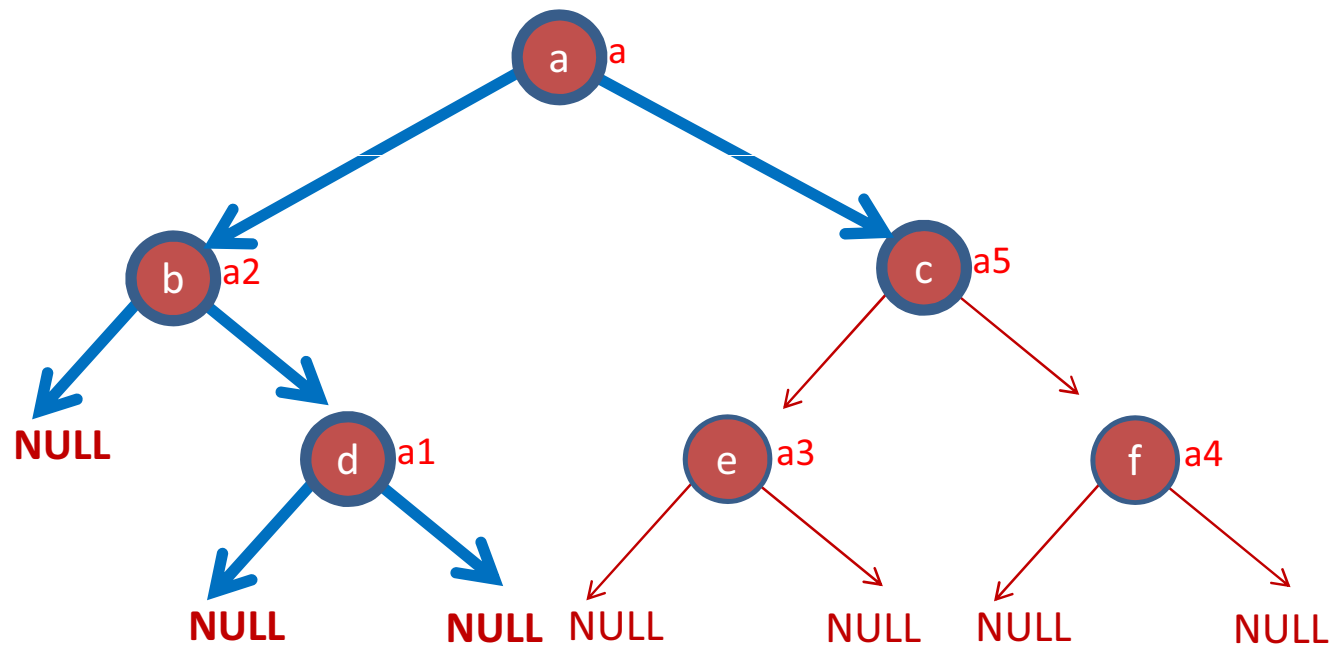
a b d

```

void arv_imprime (Arv* a)
{
    if (!arv_vazia(a))
    {
        printf("%c ", a->info);
        arv_imprime(a->esq);
        arv_imprime(a->dir);
    }
}

```

arv_imprime (a5);



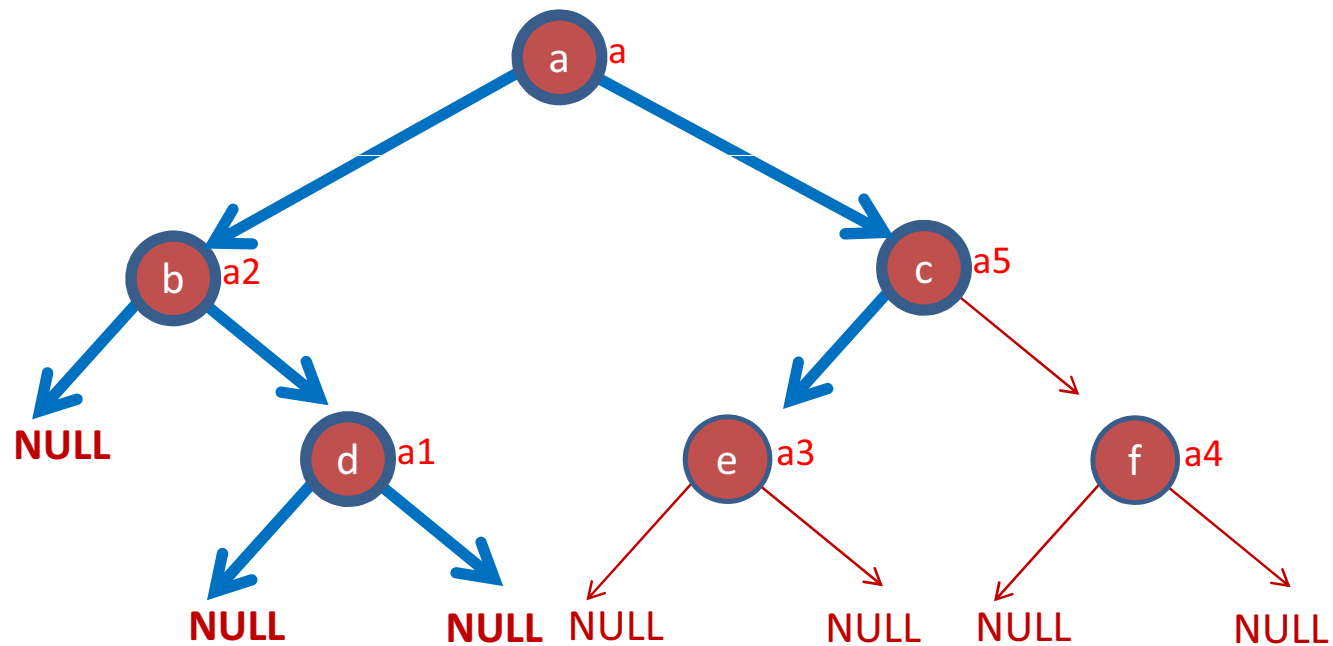
a b d c

```

void arv_imprime (Arv* a)
{
    if (!arv_vazia(a))
    {
        printf("%c ", a->info);
        arv_imprime(a->esq);
        arv_imprime(a->dir);
    }
}

```

arv_imprime (a5);



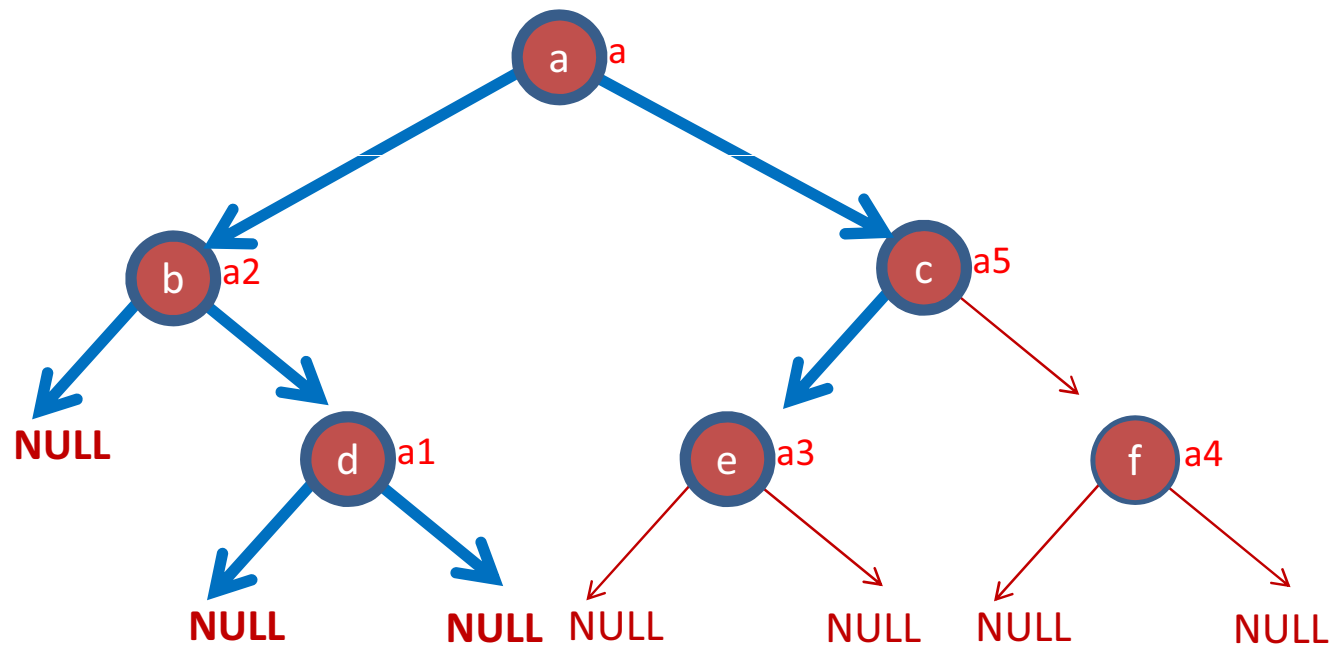
a b d c

```

void arv_imprime (Arv* a)
{
    if (!arv_vazia(a))
    {
        printf("%c ", a->info);
        arv_imprime(a->esq);
        arv_imprime(a->dir);
    }
}

```

arv_imprime (a3);



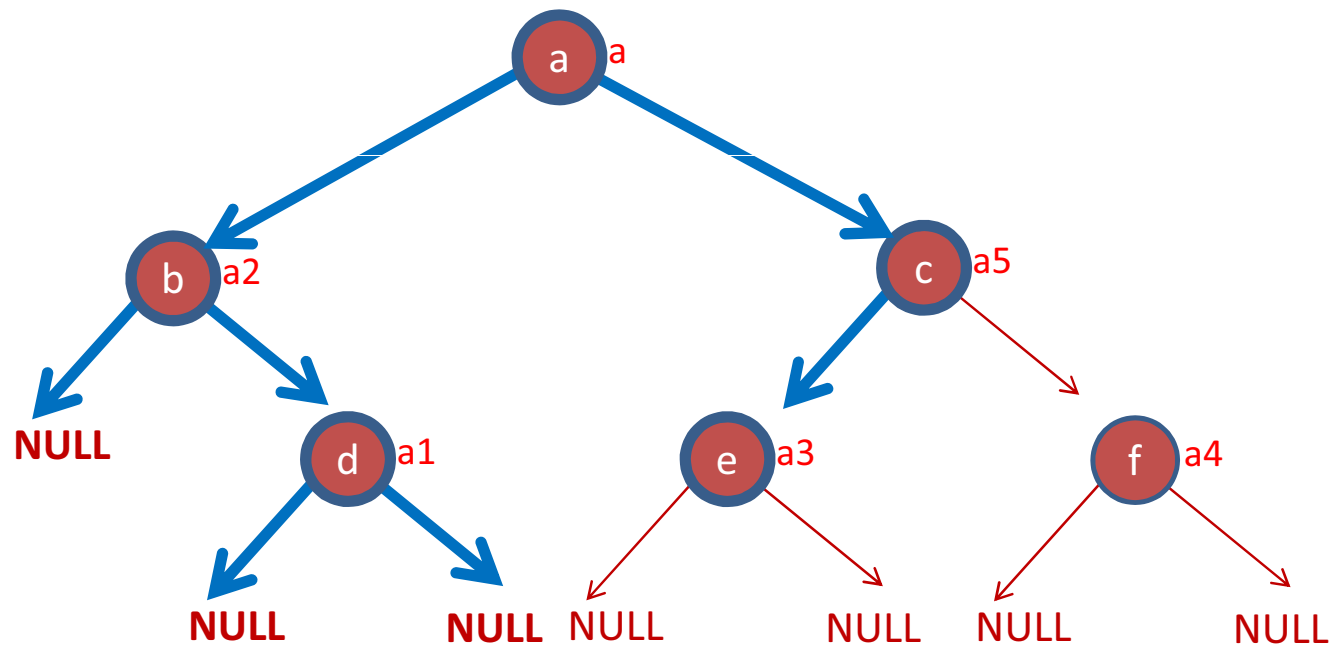
a b d c

```

void arv_imprime (Arv* a)
{
    if (!arv_vazia(a))
    {
        printf("%c ", a->info);
        arv_imprime(a->esq);
        arv_imprime(a->dir);
    }
}

```

arv_imprime (a3);



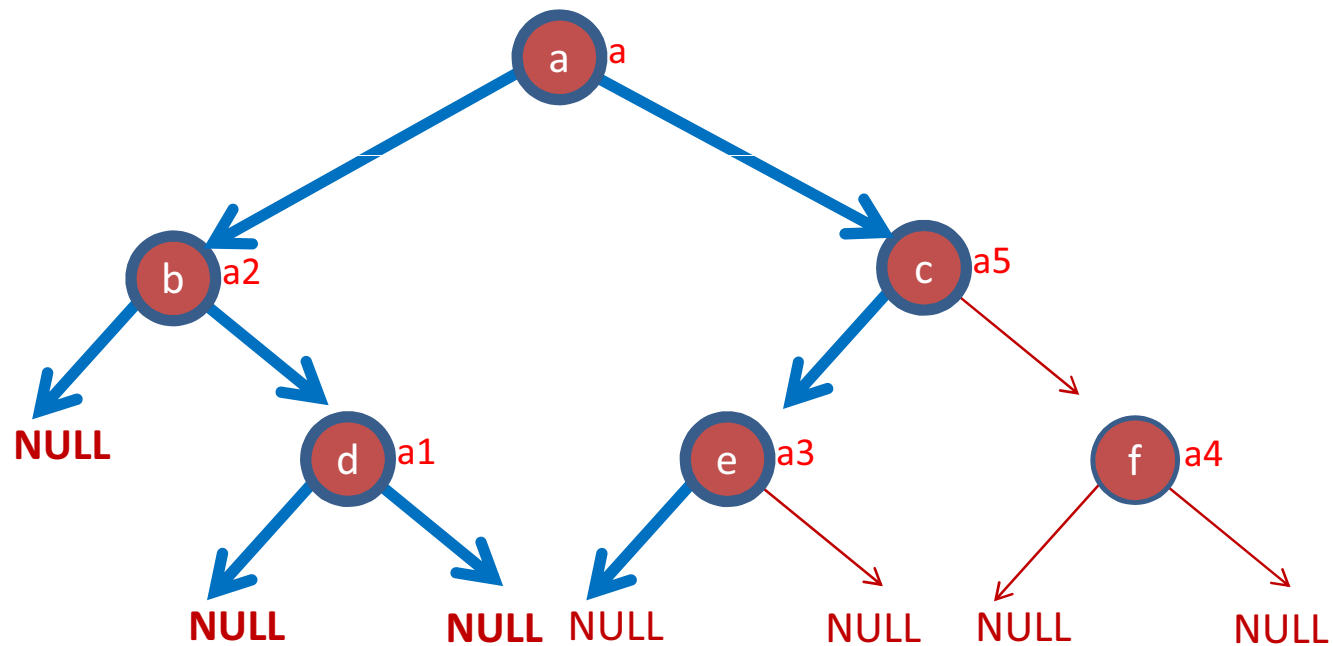
a b d c e


```

void arv_imprime (Arv* a)
{
    if (!arv_vazia(a))
    {
        printf("%c ", a->info);
        arv_imprime(a->esq);
        arv_imprime(a->dir);
    }
}

```

arv_imprime (a3);



a b d c e

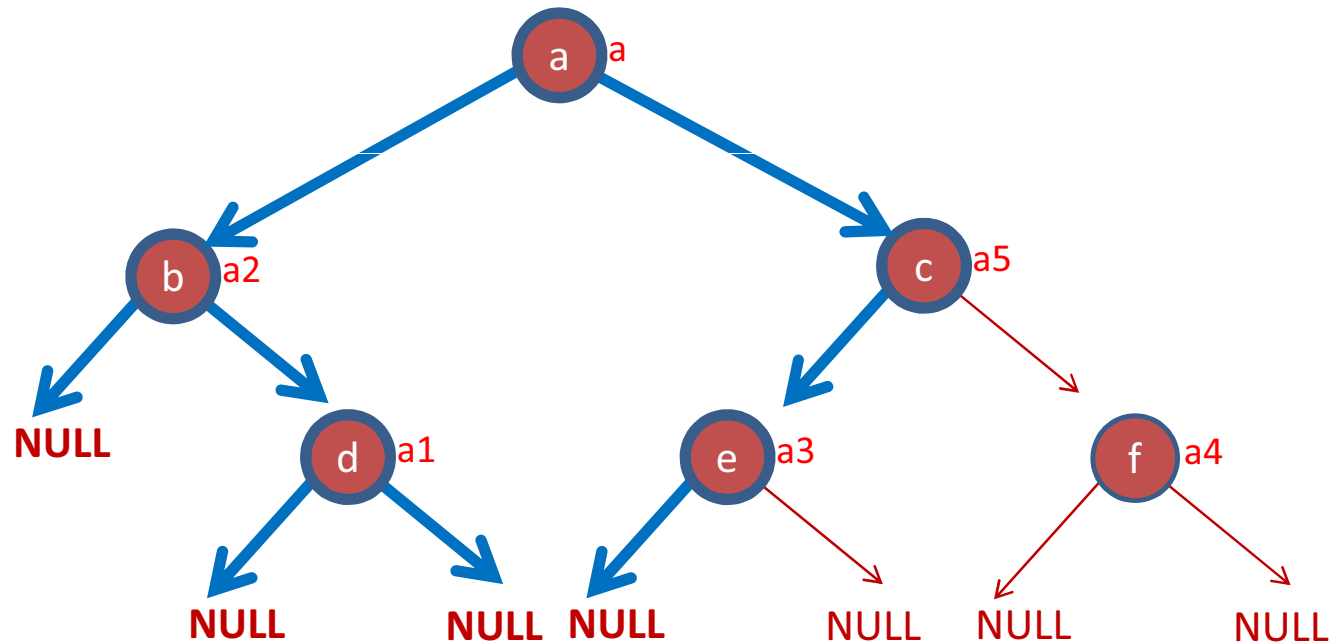
```

void arv_imprime (Arv* a)
{
    if (!arv_vazia(a))
    {
        printf("%c ", a->info);
        arv_imprime(a->esq);
        arv_imprime(a->dir);
    }
}

```

arv_imprime (NULL);

Retorna



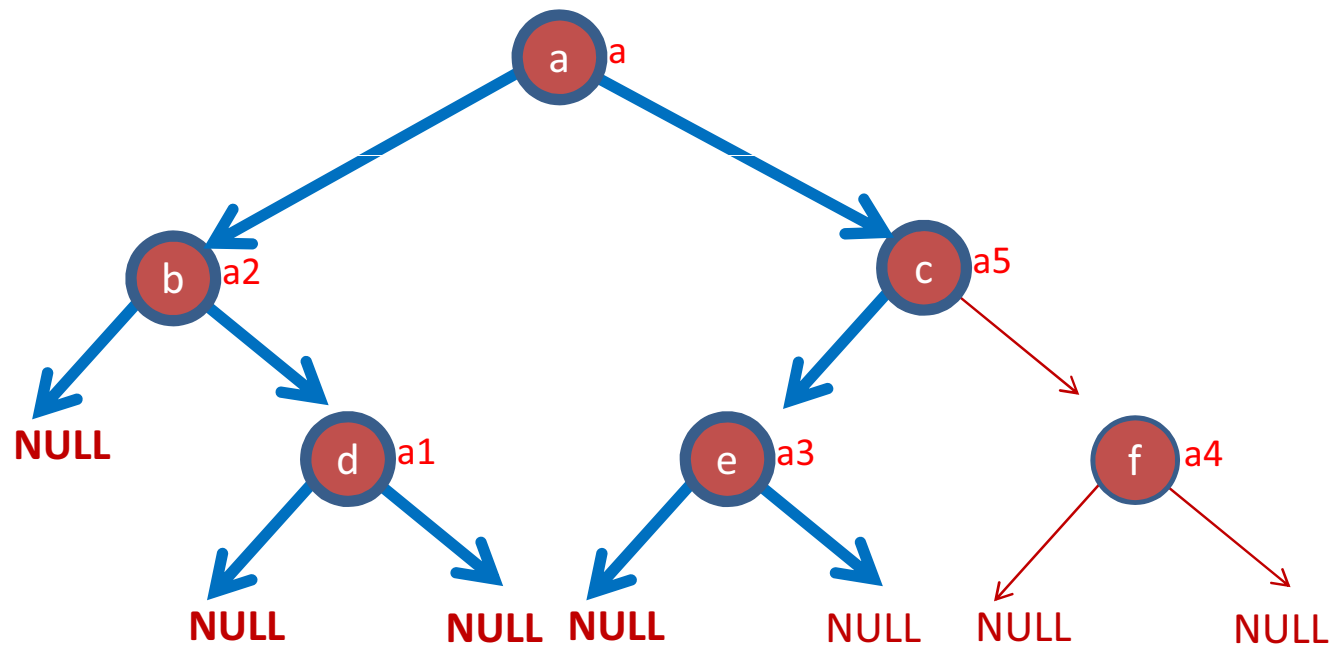
a b d c e

```

void arv_imprime (Arv* a)
{
    if (!arv_vazia(a))
    {
        printf("%c ", a->info);
        arv_imprime(a->esq);
        arv_imprime(a->dir);
    }
}

```

arv_imprime (a3);



a b d e

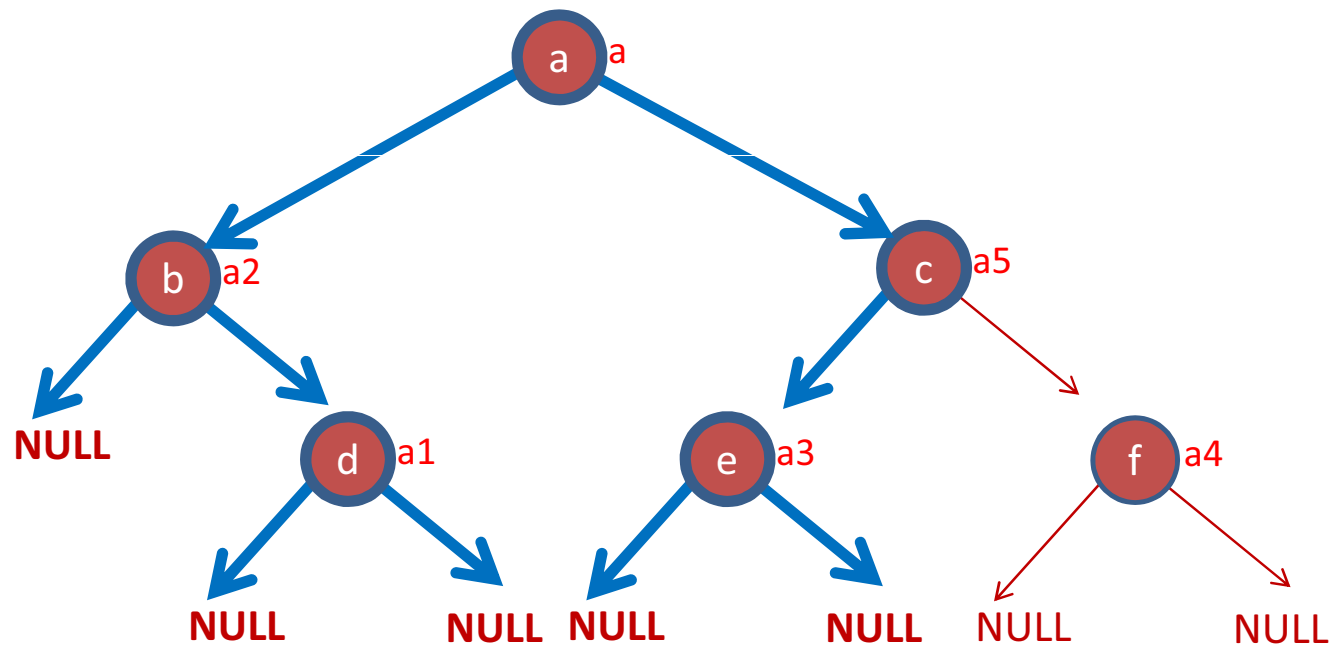
```

void arv_imprime (Arv* a)
{
    if (!arv_vazia(a))
    {
        printf("%c ", a->info);
        arv_imprime(a->esq);
        arv_imprime(a->dir);
    }
}

```

arv_imprime (NULL);

Retorna



a b d c e

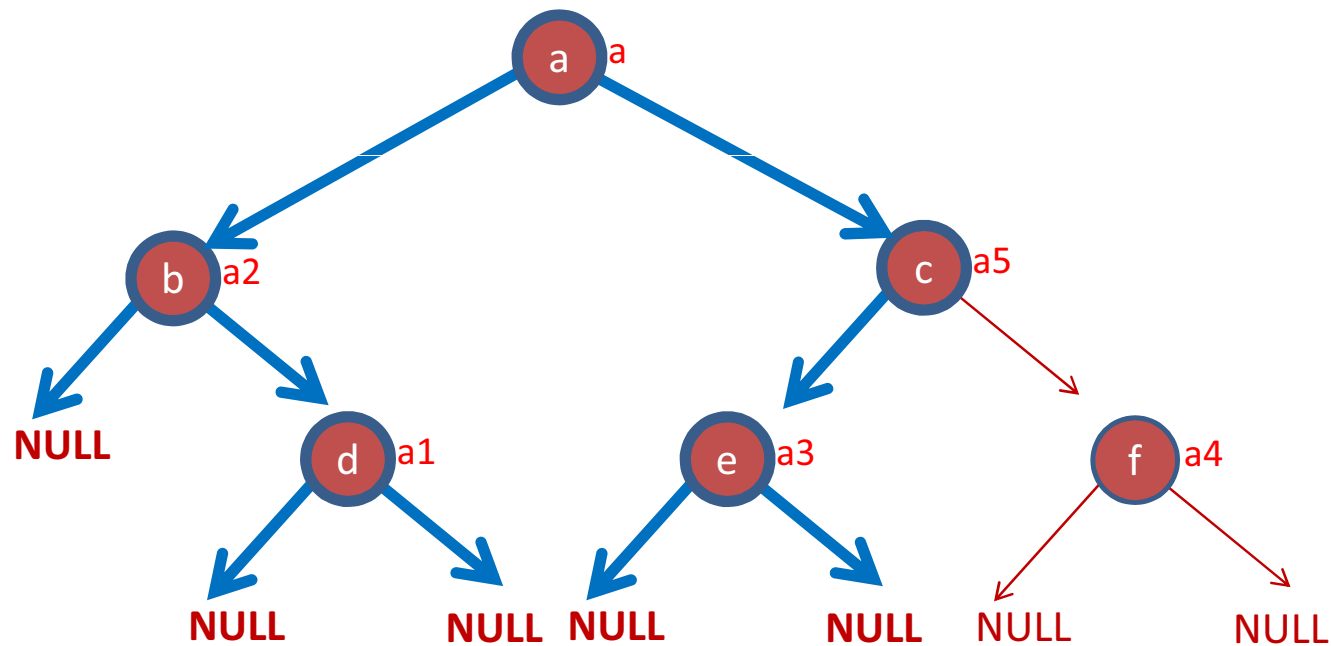
```

void arv_imprime (Arv* a)
{
    if (!arv_vazia(a))
    {
        printf("%c ", a->info);
        arv_imprime(a->esq);
        arv_imprime(a->dir);
    }
}

```

arv_imprime (a3);

Retorna



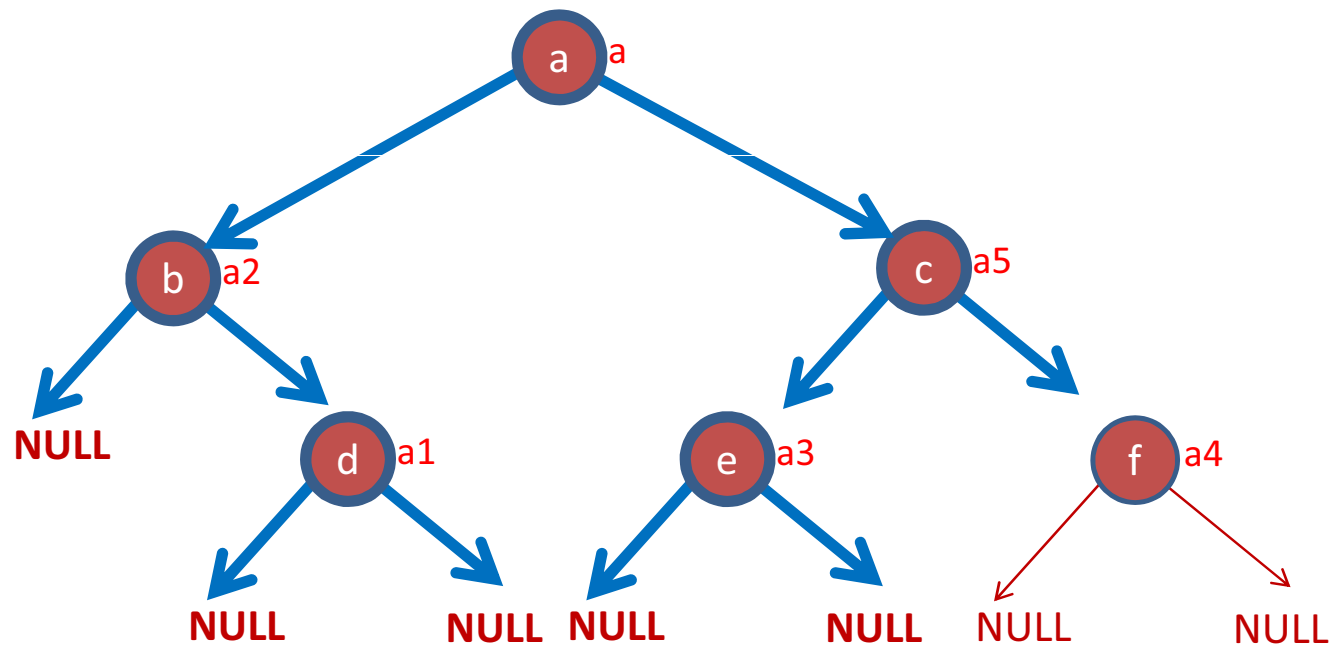
a b d c e

```

void arv_imprime (Arv* a)
{
    if (!arv_vazia(a))
    {
        printf("%c ", a->info);
        arv_imprime(a->esq);
        arv_imprime(a->dir);
    }
}

```

arv_imprime (a5);



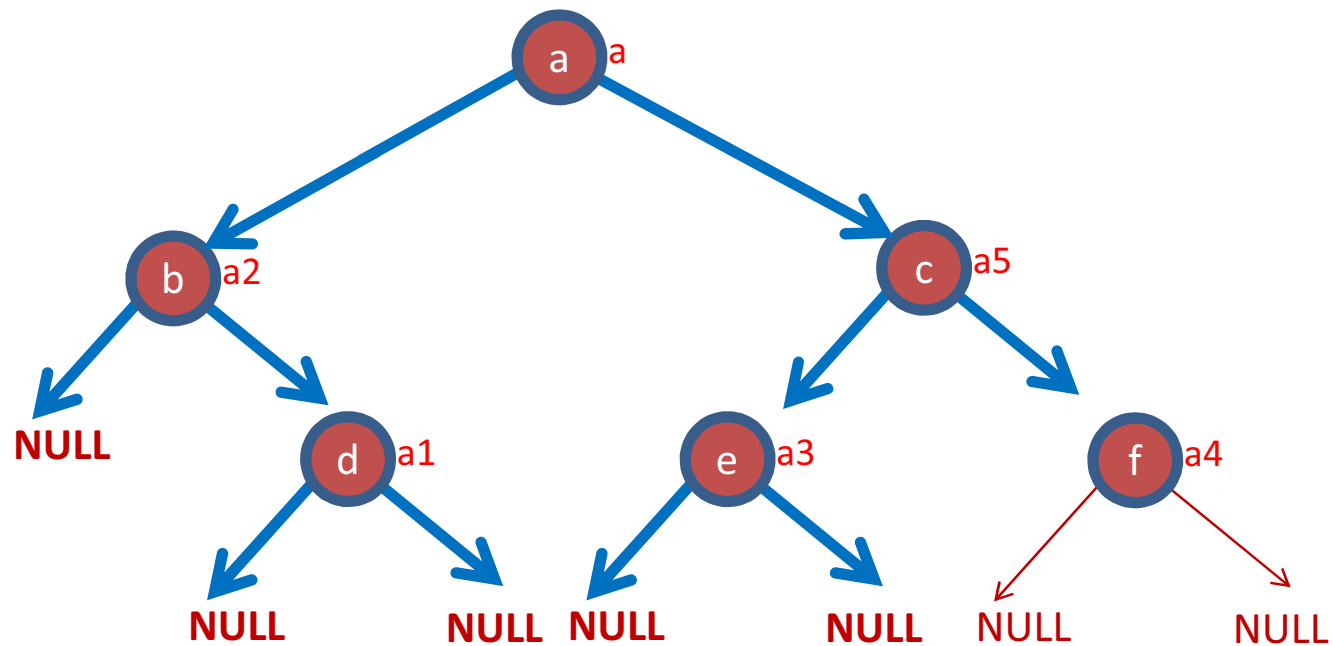
a b d c e

```

void arv_imprime (Arv* a)
{
    if (!arv_vazia(a))
    {
        printf("%c ", a->info);
        arv_imprime(a->esq);
        arv_imprime(a->dir);
    }
}

```

arv_imprime (a4);



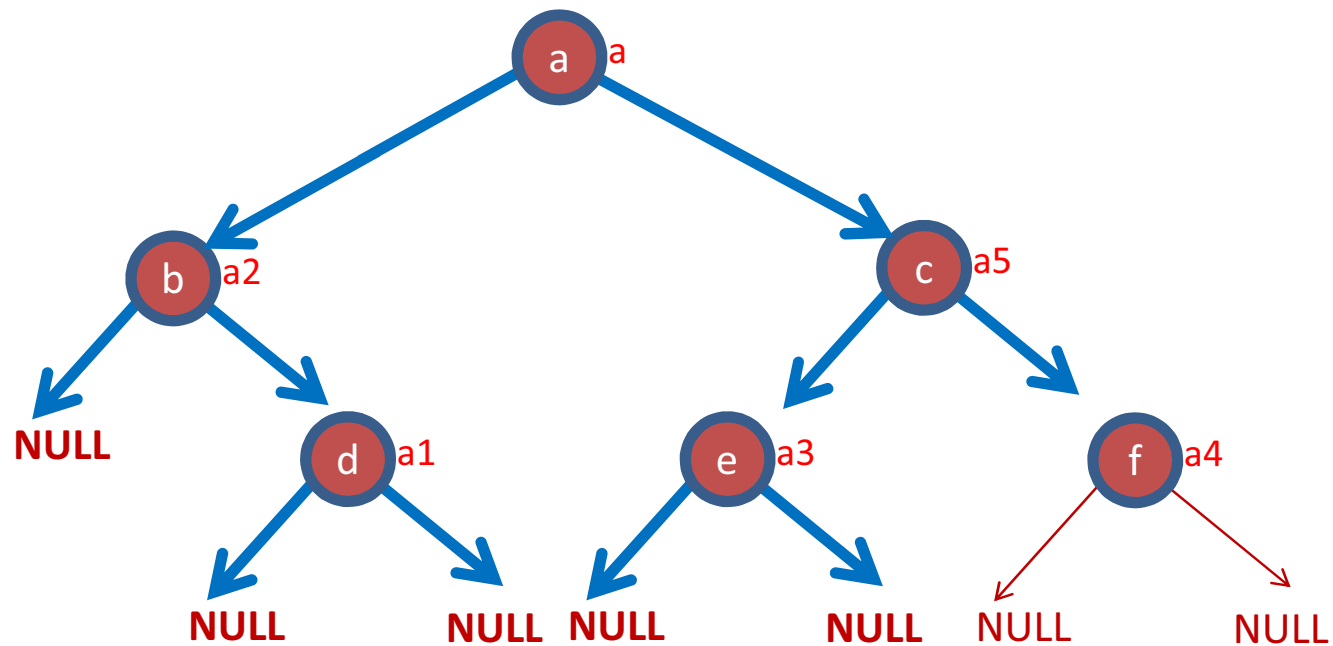
a b d c e

```

void arv_imprime (Arv* a)
{
    if (!arv_vazia(a))
    {
        printf("%c ", a->info);
        arv_imprime(a->esq);
        arv_imprime(a->dir);
    }
}

```

arv_imprime (a4);



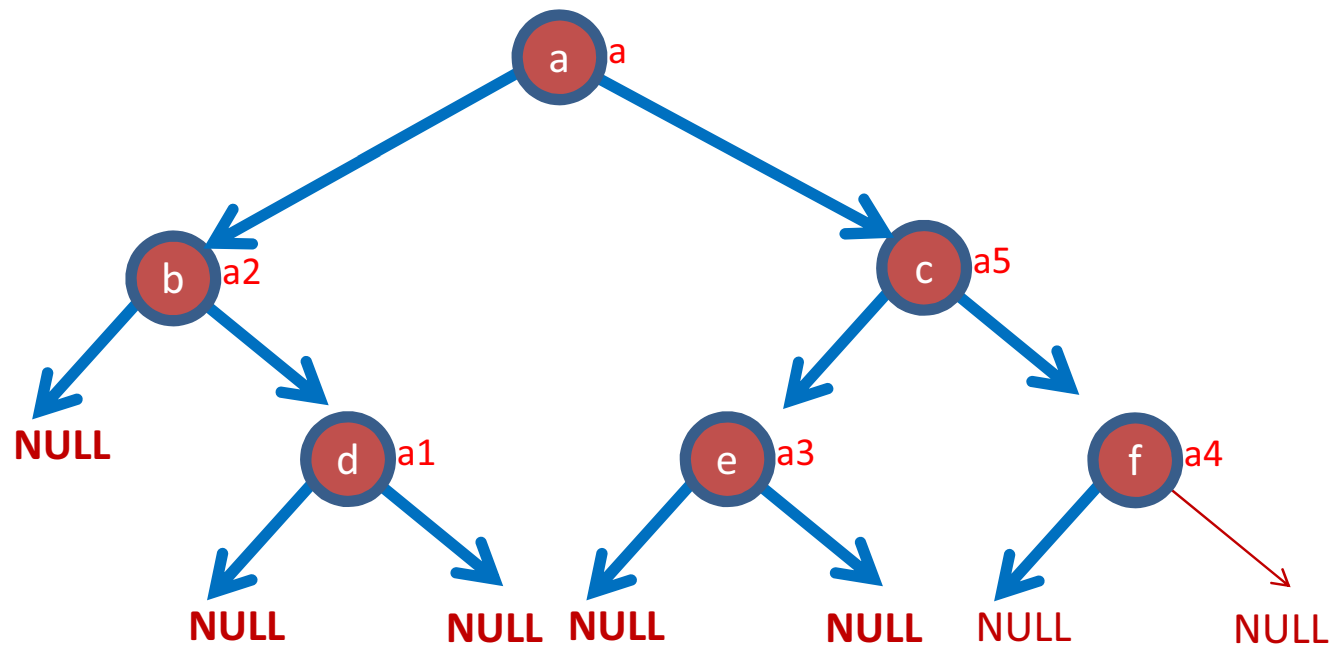
a b d c e f


```

void arv_imprime (Arv* a)
{
    if (!arv_vazia(a))
    {
        printf("%c ", a->info);
        arv_imprime(a->esq);
        arv_imprime(a->dir);
    }
}

```

arv_imprime (a4);



a b d c e f

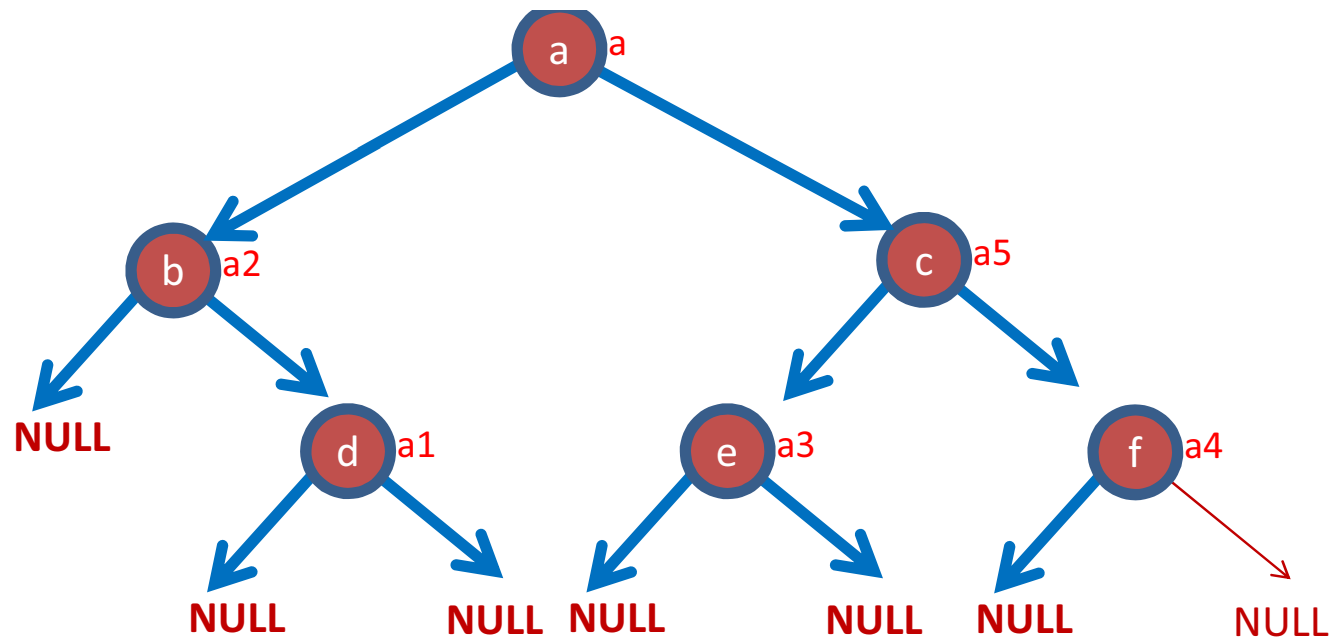
```

void arvore_imprime (Arv* a)
{
    if (!arvore_vazia(a))
    {
        printf("%c ", a->info);
        arvore_imprime(a->esq);
        arvore_imprime(a->dir);
    }
}

```

arvore_imprime (NULL);

Retorna



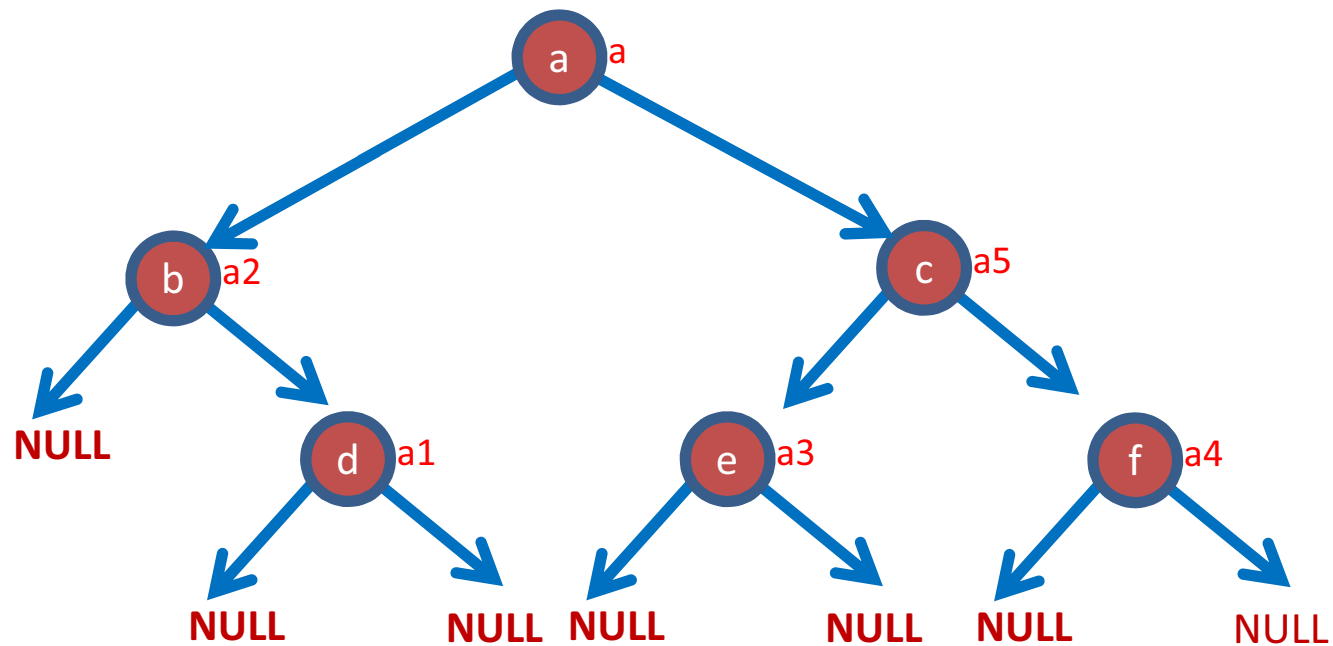
a b d c e f

```

void arv_imprime (Arv* a)
{
    if (!arv_vazia(a))
    {
        printf("%c ", a->info);
        arv_imprime(a->esq);
        arv_imprime(a->dir);
    }
}

```

arv_imprime (a4);



a b d c e f

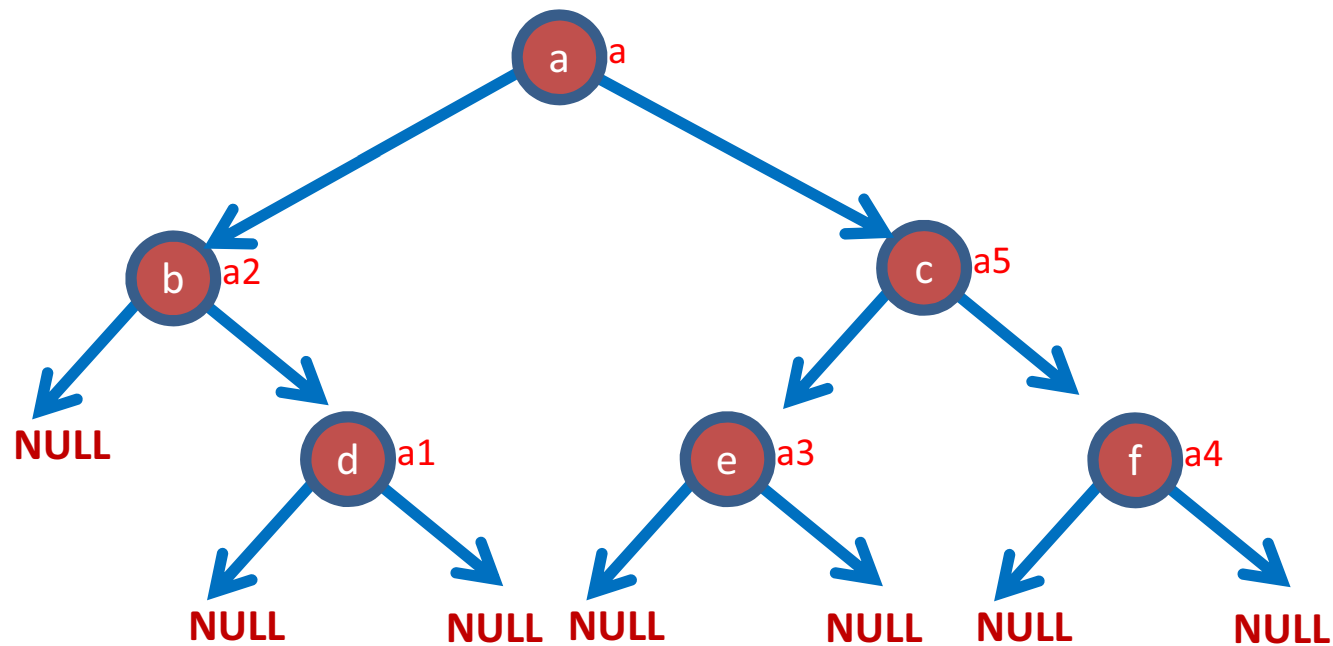
```

void arv_imprime (Arv* a)
{
    if (!arv_vazia(a))
    {
        printf("%c ", a->info);
        arv_imprime(a->esq);
        arv_imprime(a->dir);
    }
}

```

arv_imprime (NULL);

Retorna



a b d c e f

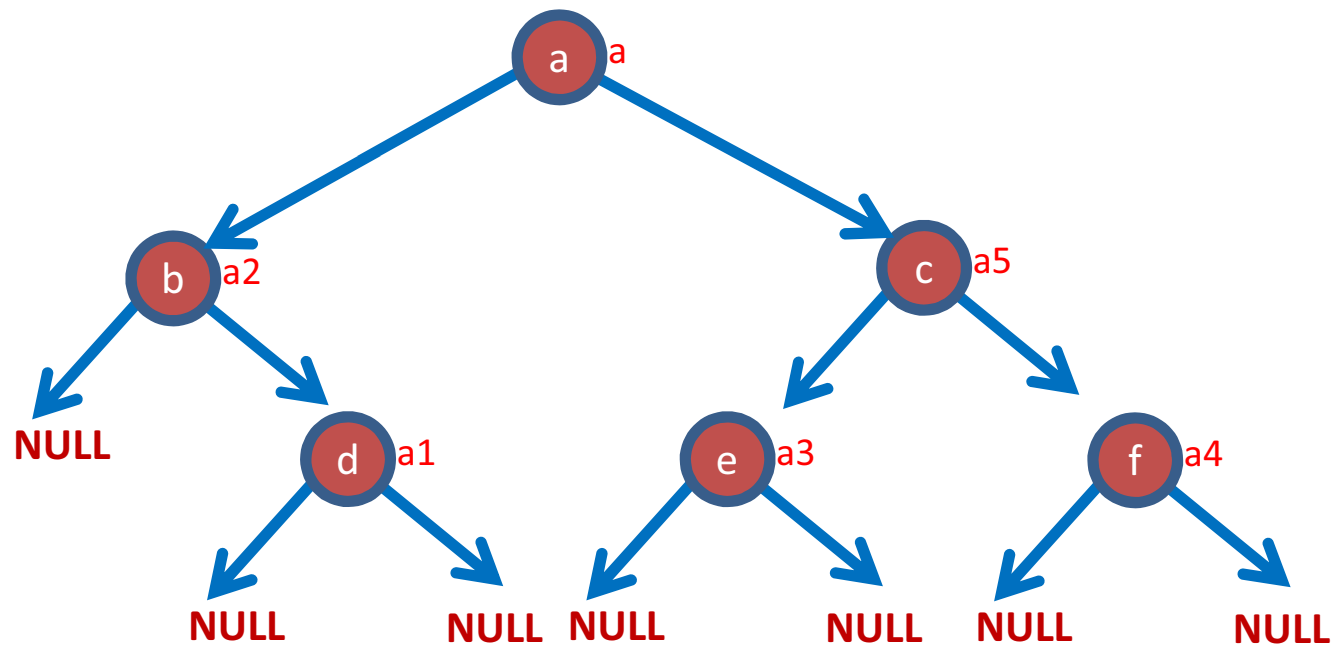
```

void arv_imprime (Arv* a)
{
    if (!arv_vazia(a))
    {
        printf("%c ", a->info);
        arv_imprime(a->esq);
        arv_imprime(a->dir);
    }
}

```

arv_imprime (a4);

Retorna



a b d c e f

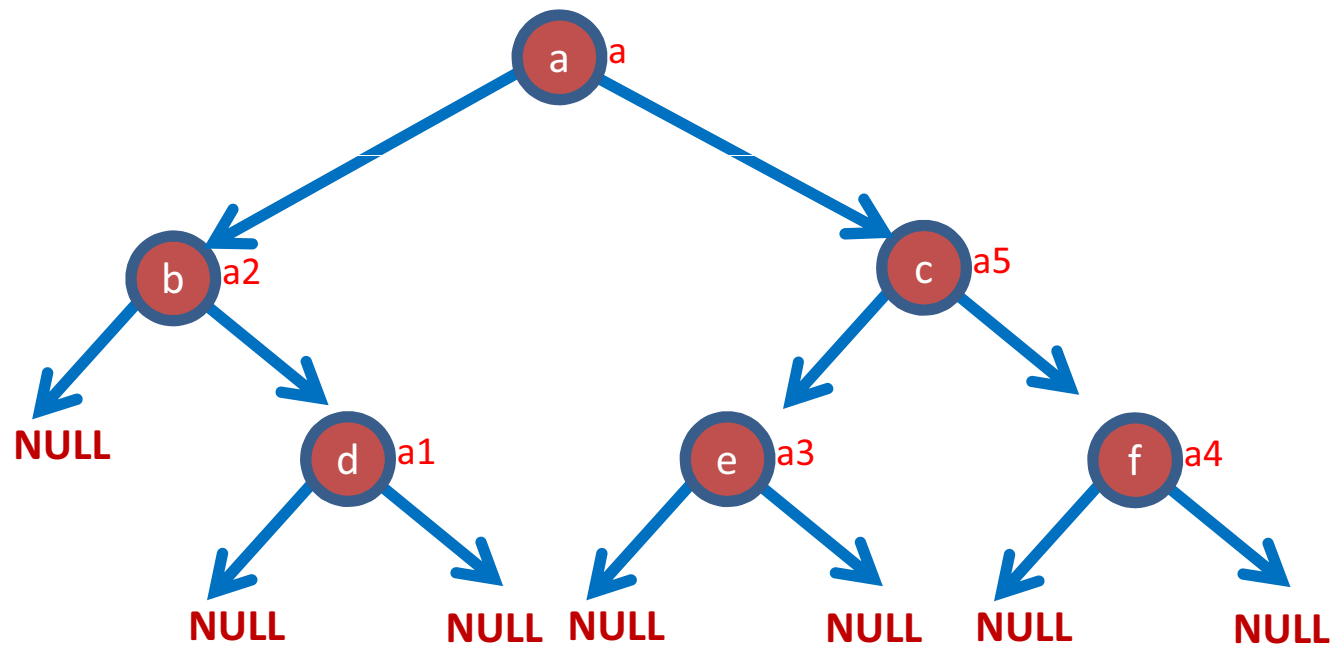
```

void arv_imprime (Arv* a)
{
    if (!arv_vazia(a))
    {
        printf("%c ", a->info);
        arv_imprime(a->esq);
        arv_imprime(a->dir);
    }
}

```

arv_imprime (a5);

Retorna



a b d c e f

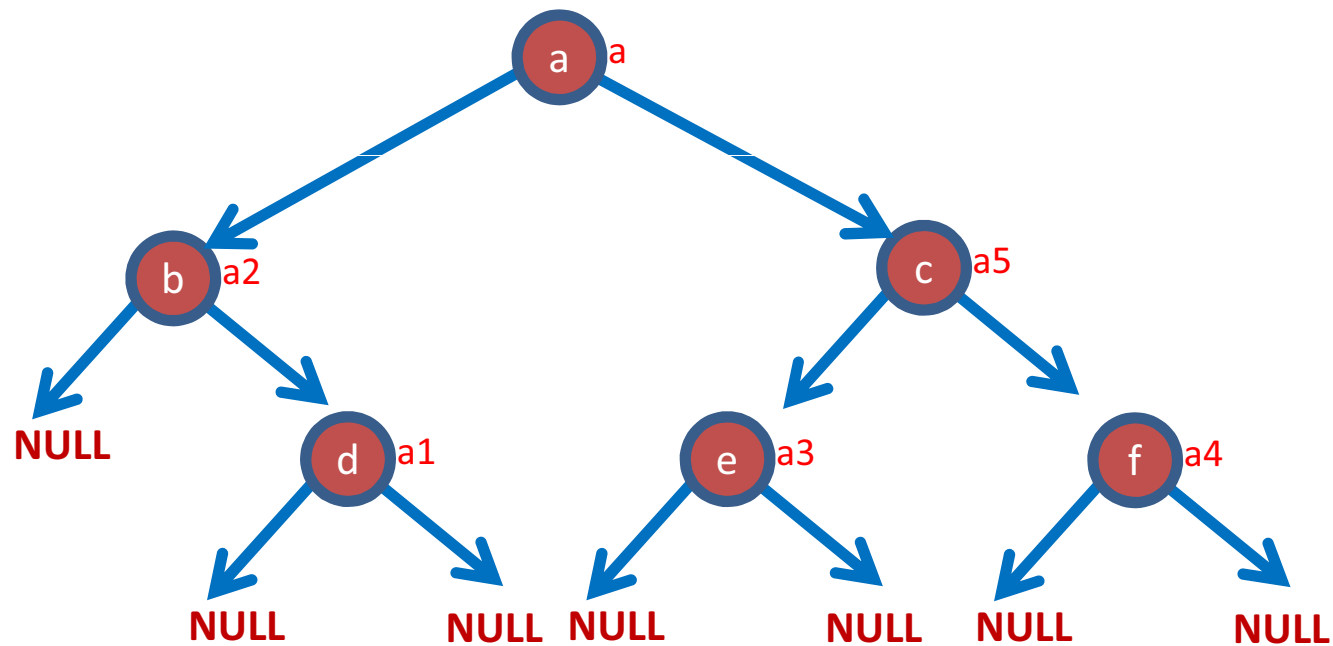
```

void arv_imprime (Arv* a)
{
  if (!arv_vazia(a))
  {
    printf("%c ", a->info);
    arv_imprime(a->esq);
    arv_imprime(a->dir);
  }
}

```

arv_imprime (a);

Retorna para o main

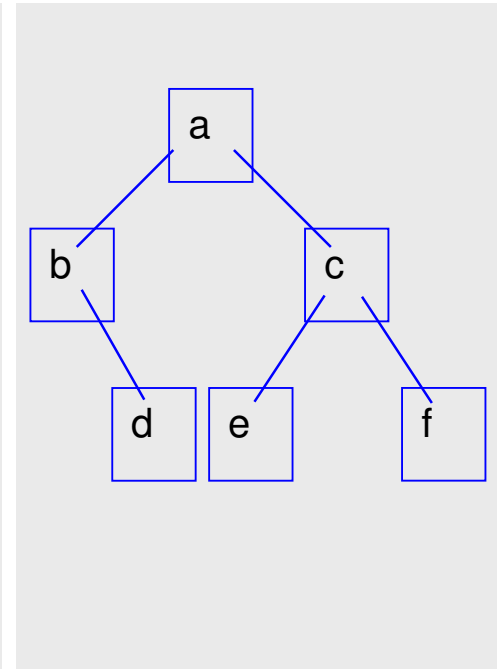


a b d c e f

Árvores binárias - Implementação em C

Exemplo: <a <b <> <d <><> > <c <e <><> > <f <><> > > >

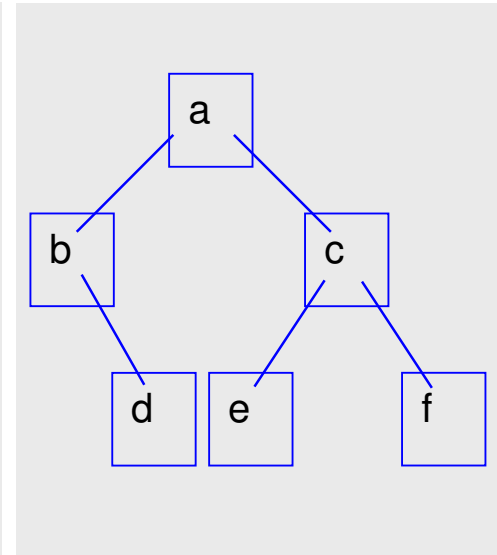
```
/* sub-árvore 'd' */
Arv* a1= arv_cria('d',arv_criavazia(),arv_criavazia());
/* sub-árvore 'b' */
Arv* a2= arv_cria('b',arv_criavazia(),a1);
/* sub-árvore 'e' */
Arv* a3= arv_cria('e',arv_criavazia(),arv_criavazia());
/* sub-árvore 'f' */
Arv* a4= arv_cria('f',arv_criavazia(),arv_criavazia());
/* sub-árvore 'c' */
Arv* a5= arv_cria('c',a3,a4);
/* árvore 'a' */
Arv* a = arv_cria('a',a2,a5 );
```



Árvores binárias - Implementação em C

Exemplo: <a <b <> <d <><> > > <c <e <><> > <f <><> > > >

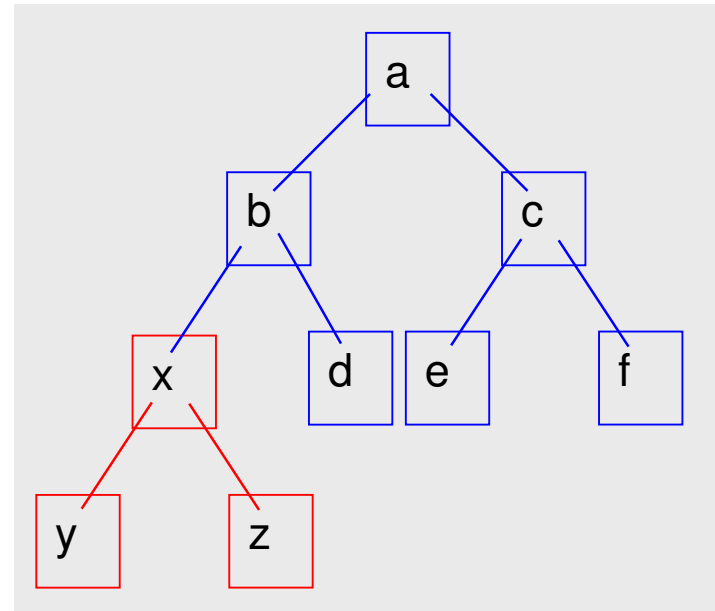
```
Arv* a = arv_cria('a',  
    arv_cria('b',  
        arv_criavazia(),  
        arv_cria('d', arv_criavazia(), arv_criavazia())  
    ),  
    arv_cria('c',  
        arv_cria('e', arv_criavazia(), arv_criavazia()),  
        arv_cria('f', arv_criavazia(), arv_criavazia())  
    )  
);
```



Árvores binárias - Implementação em C

- Exemplo - acrescenta nós

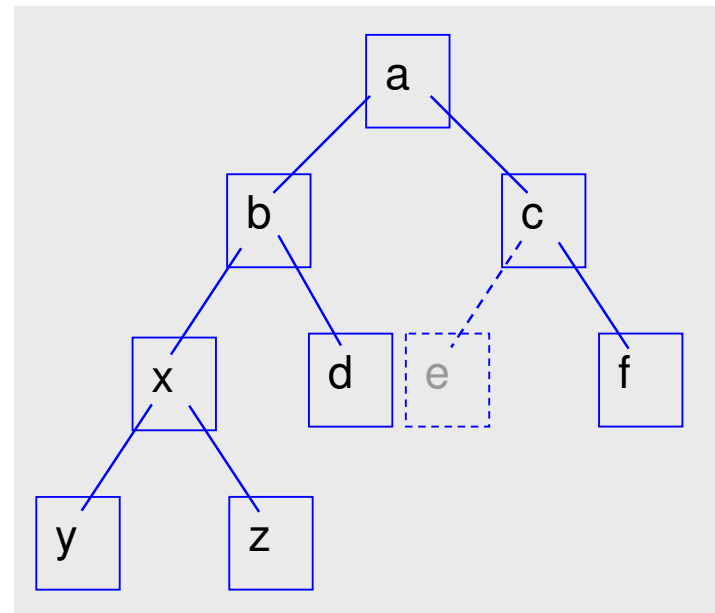
```
a->esq->esq =  
    arv_cria('x',  
            arv_cria('y',  
                    arv_criavazia(),  
                    arv_criavazia()),  
            arv_cria('z',  
                    arv_criavazia(),  
                    arv_criavazia())  
    );
```



Árvores binárias - Implementação em C

- Exemplo - libera nós

```
a->dir->esq =  
    arv_libera(a->dir->esq);
```



Árvores binárias - Ordens de percurso

Ordens de percurso:

pré-ordem:

trata *raiz*, percorre *sae*, percorre *sad*

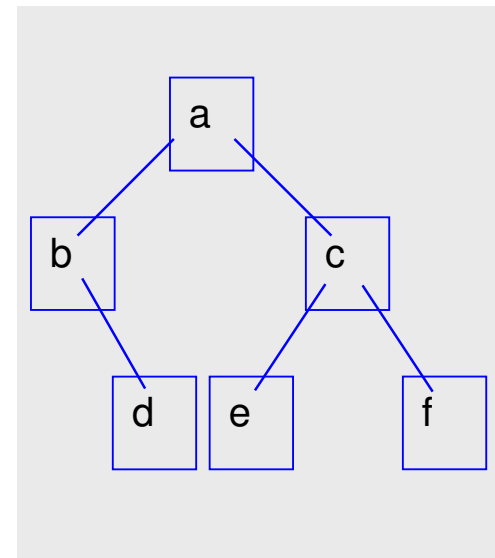
exemplo: a b d c e f

ordem simétrica:

- percorre *sae*, trata *raiz*, percorre *sad*
- exemplo: b d a e c f

— *pós-ordem:*

- percorre *sae*, percorre *sad*, trata *raiz*
- exemplo: d b e f c a



Ordens de Percurso

Eduardo Piveta

Introdução

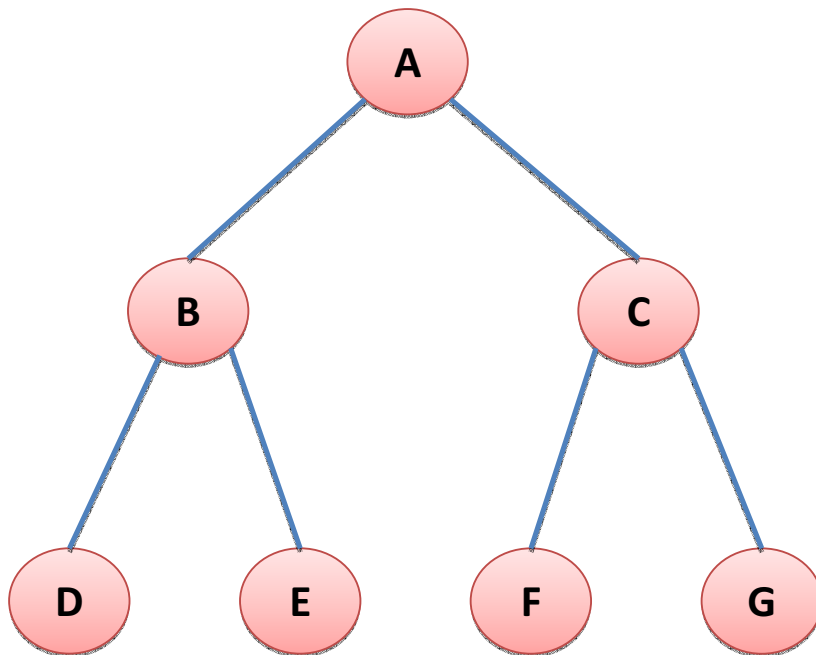
- Existem diversas ordens possíveis para percorrermos uma árvore.
- Nesta apresentação, veremos as mais utilizadas, que são:
 - pré-ordem
 - ordem simétrica
 - pós ordem

Ordens de percurso

- Pré-ordem:
 - Primeiro é tratada a raiz, percorre a SAE, percorre a SAD
- Ordem simétrica:
 - Percorre SAE, trata a raiz, percorre a SAD
- Pós-ordem:
 - Percorre SAE, percorre a SAD, trata a raiz

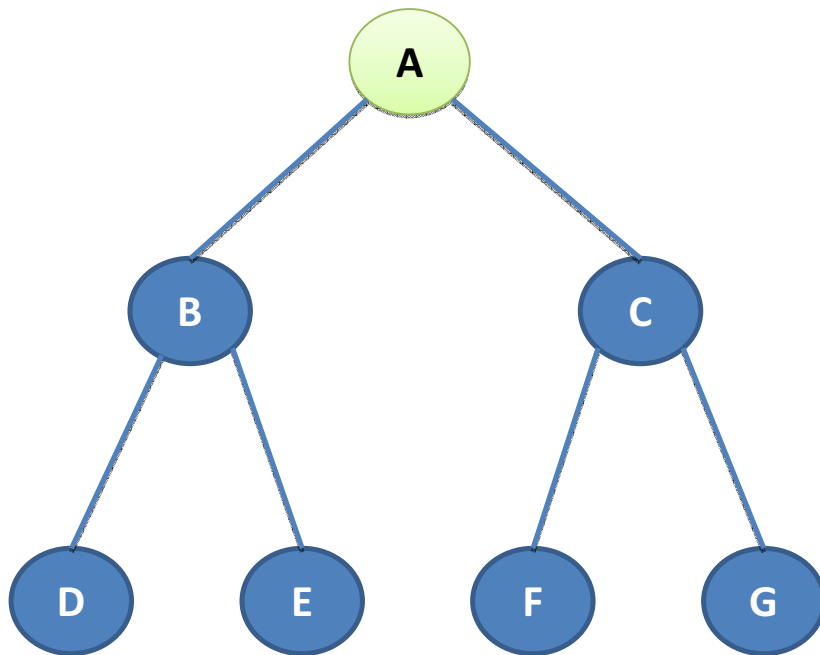
Exemplo – Pré-ordem

Pré-ordem: RAIZ(arv), SAE(arv), SAD(arv)



Exemplo – Pré-ordem

Pré-ordem: RAIZ(arv), SAE(arv), SAD(arv)

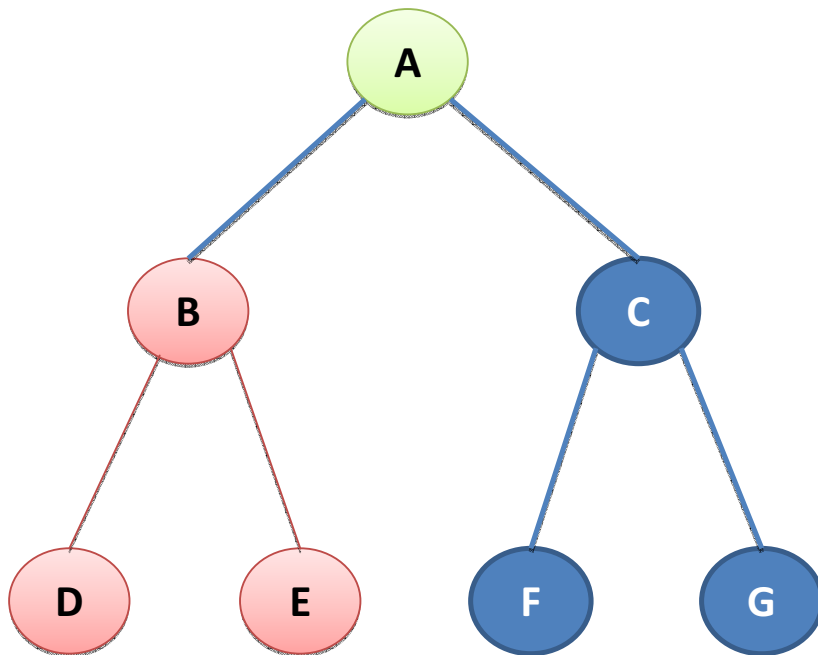


Percurso:

A

Exemplo – Pré-ordem

Pré-ordem: RAIZ(arv), SAE(arv), SAD(arv)

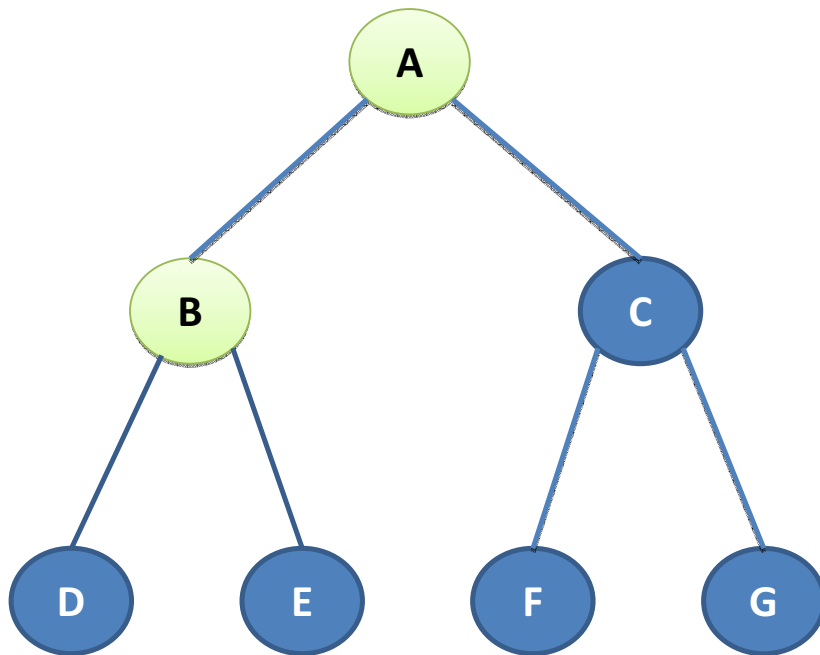


Percurso:

A

Exemplo – Pré-ordem

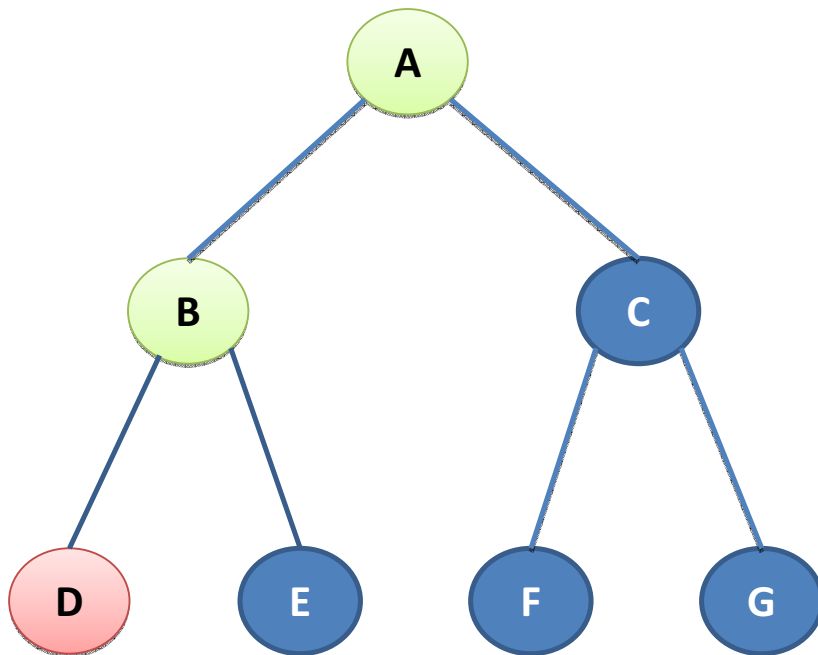
Pré-ordem: RAIZ(arv), SAE(arv), SAD(arv)



Percurso:
A B

Exemplo – Pré-ordem

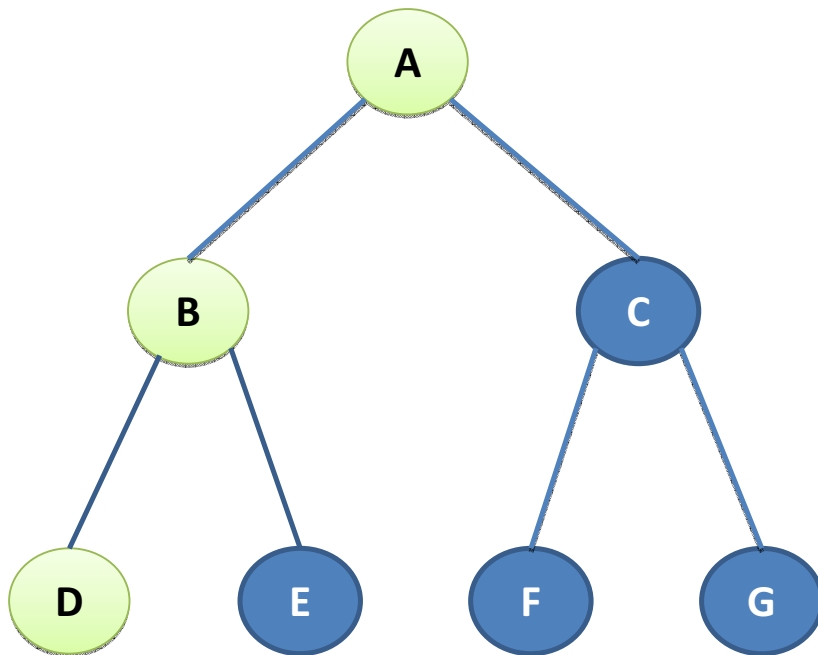
Pré-ordem: RAIZ(arv), SAE(arv), SAD(arv)



Percurso:
A B

Exemplo – Pré-ordem

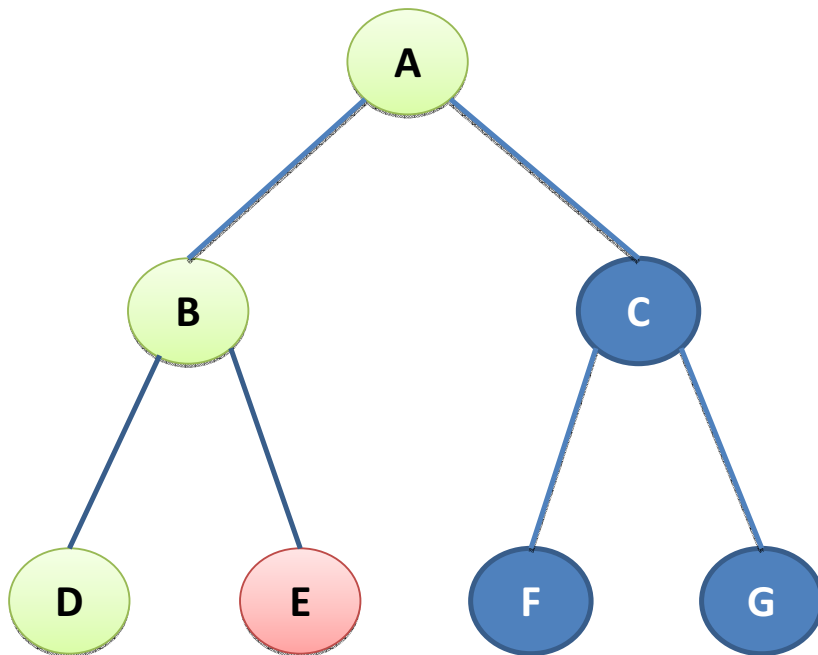
Pré-ordem: RAIZ(arv), SAE(arv), SAD(arv)



Percurso:
A B D

Exemplo – Pré-ordem

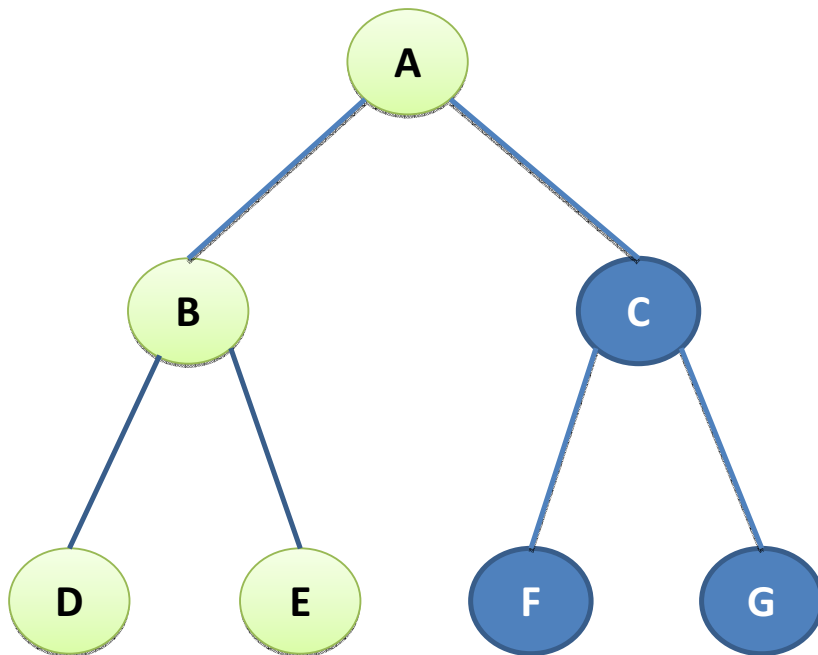
Pré-ordem: RAIZ(arv), SAE(arv), SAD(arv)



Percurso:
A B D

Exemplo – Pré-ordem

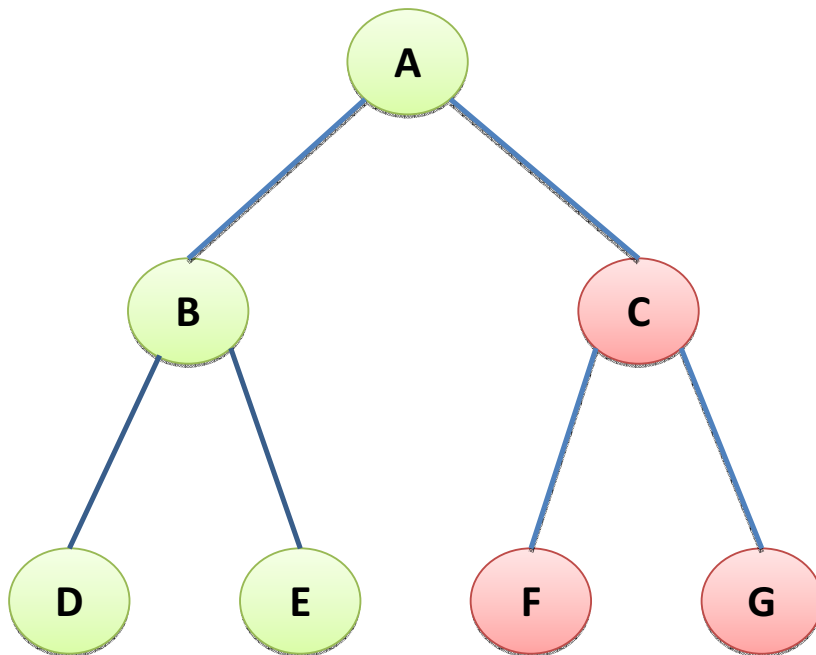
Pré-ordem: RAIZ(arv), SAE(arv), SAD(arv)



Percurso:
A B D E

Exemplo – Pré-ordem

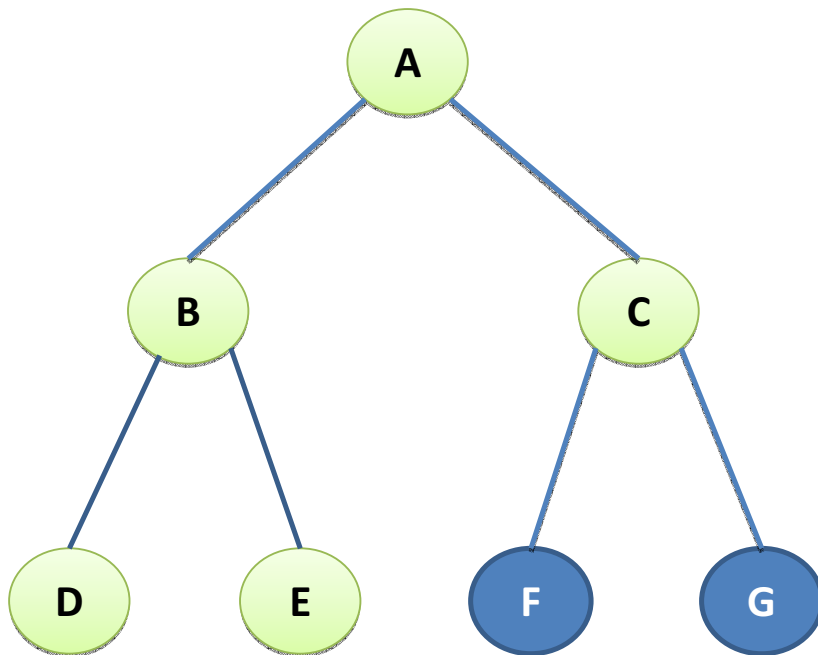
Pré-ordem: RAIZ(arv), SAE(arv), SAD(arv)



Percurso:
A B D E

Exemplo – Pré-ordem

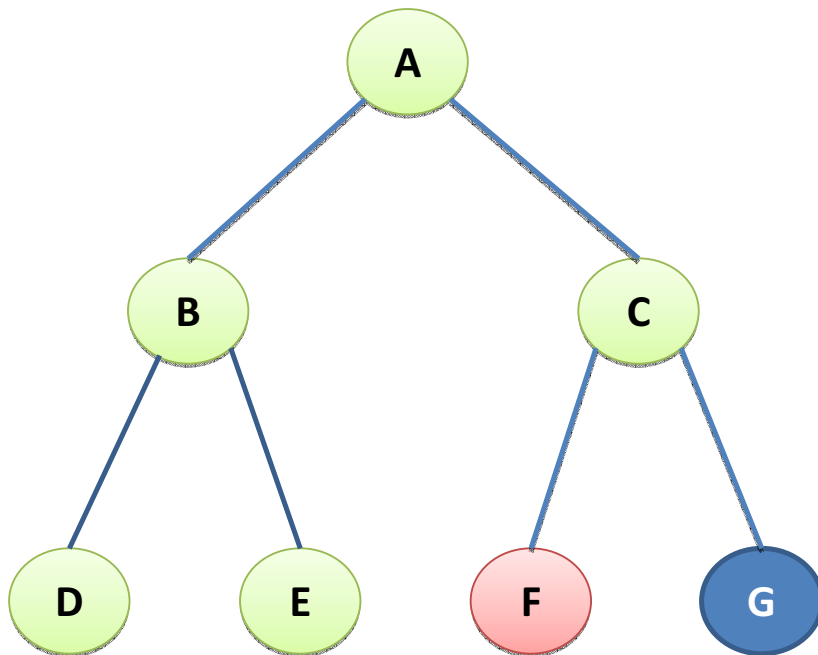
Pré-ordem: RAIZ(arv), SAE(arv), SAD(arv)



Percurso:
A B D E C

Exemplo – Pré-ordem

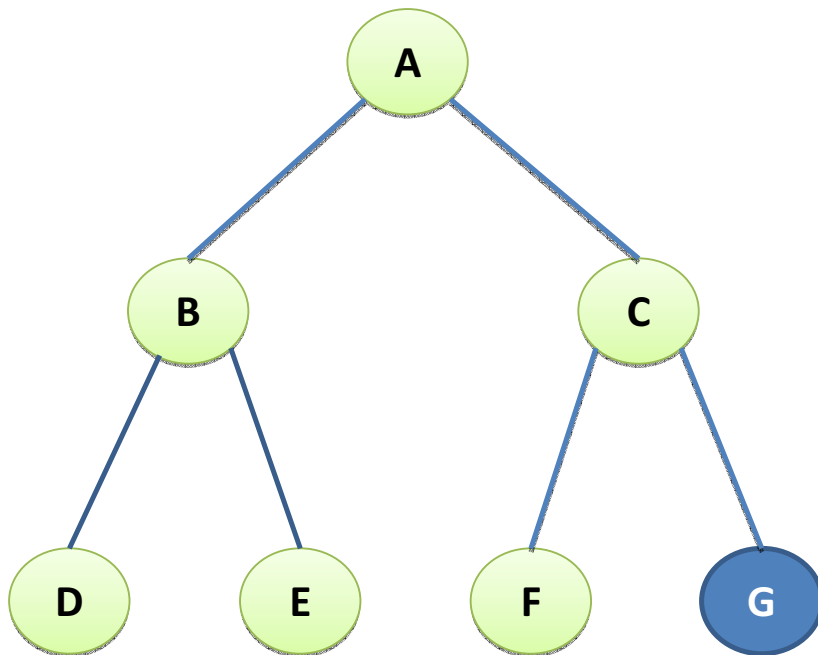
Pré-ordem: RAIZ(arv), SAE(arv), SAD(arv)



Percurso:
A B D E C

Exemplo – Pré-ordem

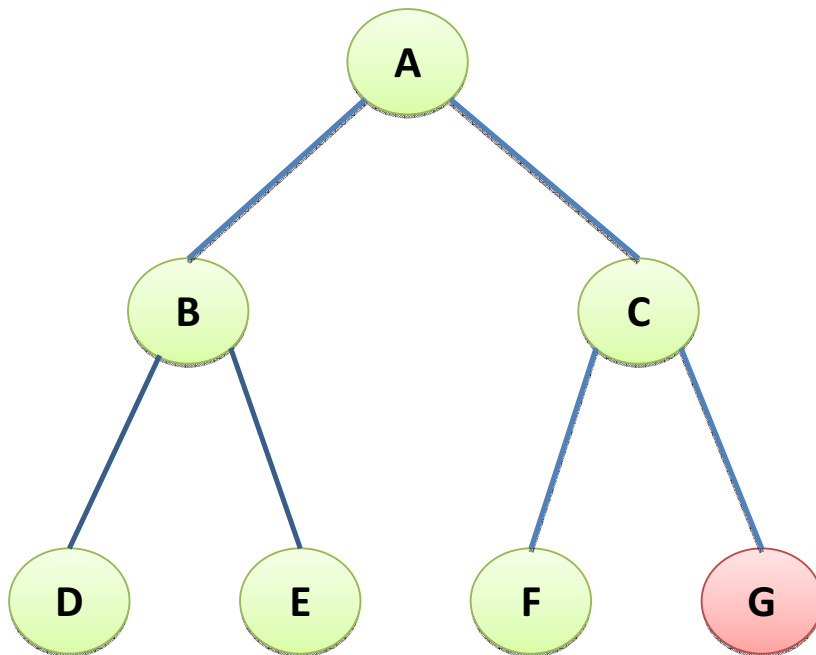
Pré-ordem: RAIZ(arv), SAE(arv), SAD(arv)



Percurso:
A B D E C F

Exemplo – Pré-ordem

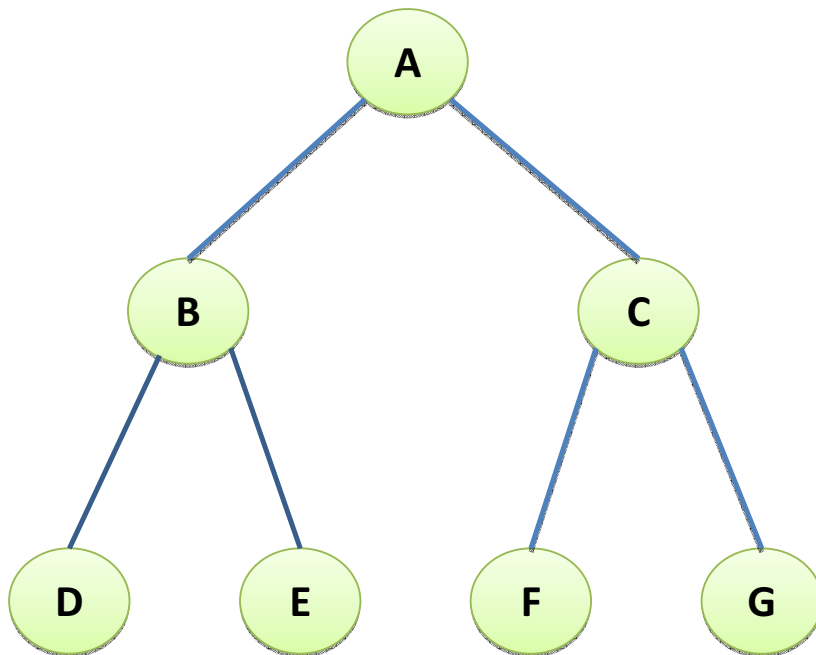
Pré-ordem: RAIZ(arv), SAE(arv), SAD(arv)



Percurso:
A B D E C F

Exemplo – Pré-ordem

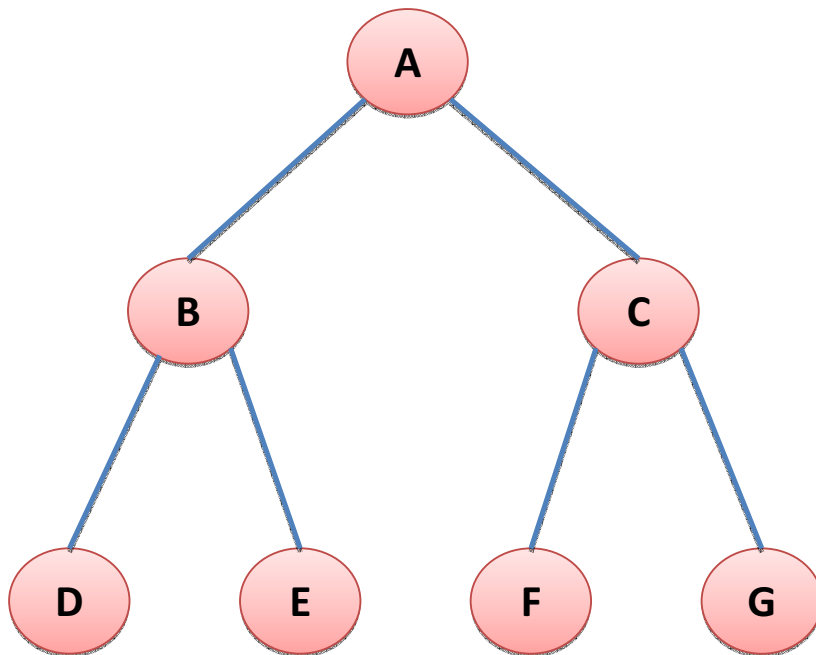
Pré-ordem: RAIZ(arv), SAE(arv), SAD(arv)



Percurso:
A B D E C F G

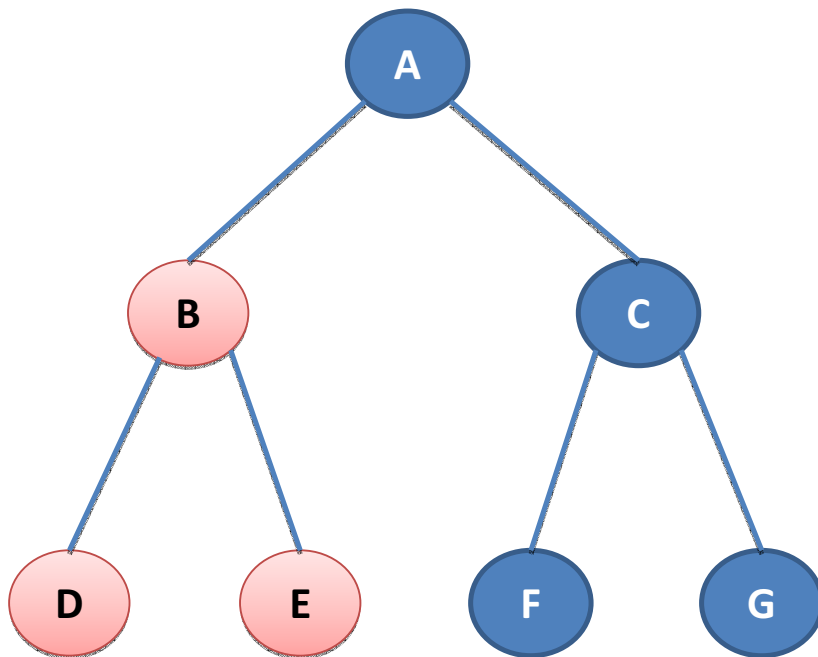
Exemplo – Ordem simétrica

Ordem simétrica: SAE(arv), RAIZ(arv), SAD(arv)



Exemplo – Ordem simétrica

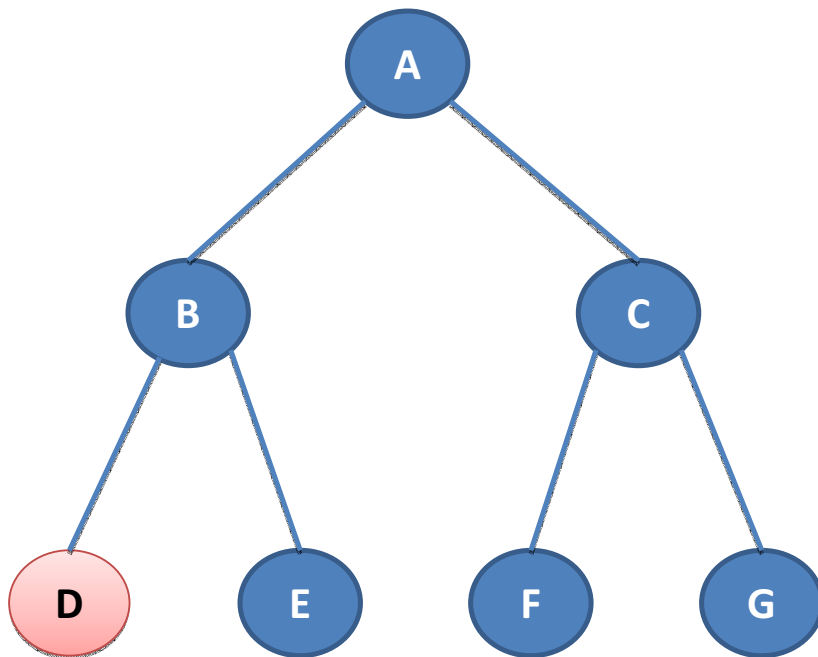
Ordem simétrica: SAE(arv), RAIZ(arv), SAD(arv)



Percurso:

Exemplo – Ordem simétrica

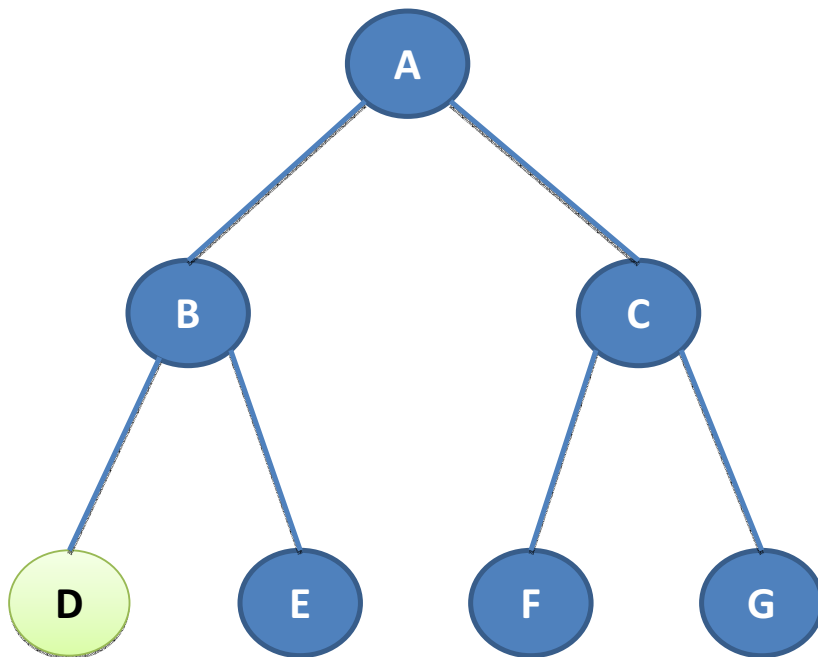
Ordem simétrica: SAE(arv), RAIZ(arv), SAD(arv)



Percurso:

Exemplo – Ordem simétrica

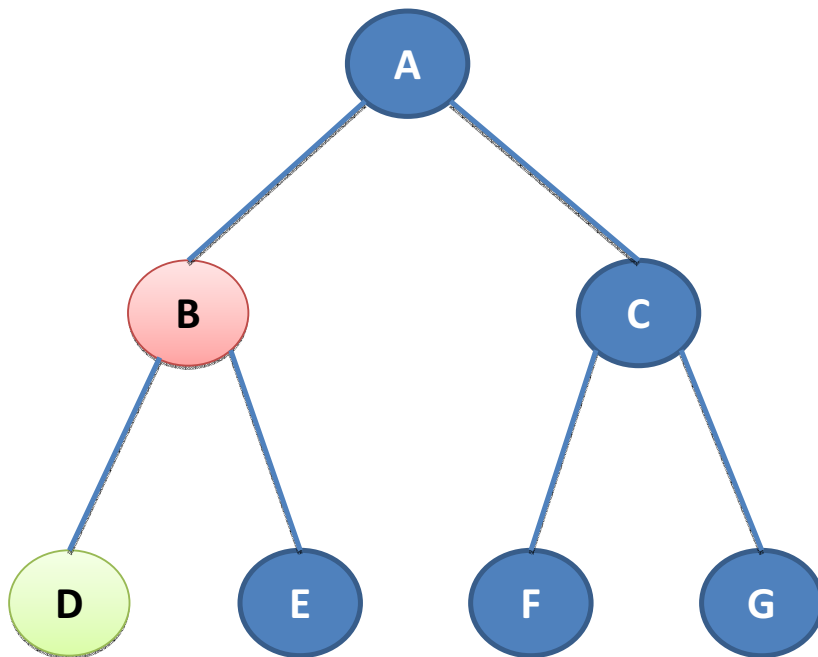
Ordem simétrica: SAE(arv), RAIZ(arv), SAD(arv)



Percurso:
D

Exemplo – Ordem simétrica

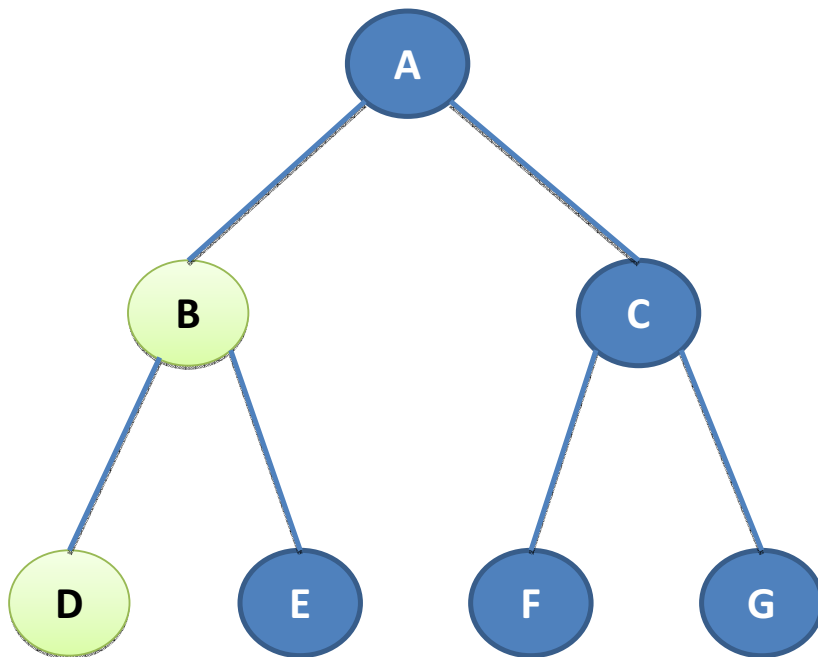
Ordem simétrica: SAE(arv), RAIZ(arv), SAD(arv)



Percurso:
D

Exemplo – Ordem simétrica

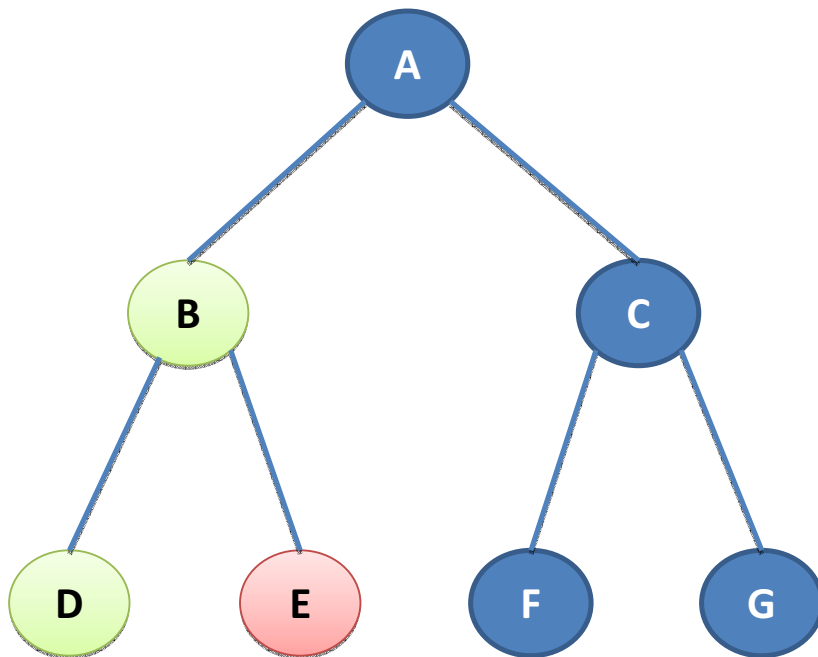
Ordem simétrica: SAE(arv), RAIZ(arv), SAD(arv)



Percurso:
D B

Exemplo – Ordem simétrica

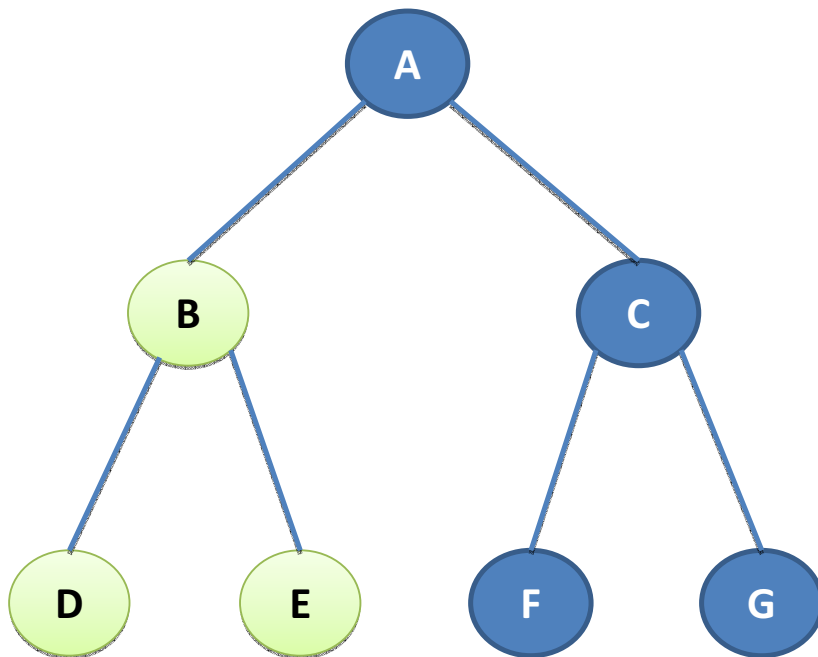
Ordem simétrica: SAE(arv), RAIZ(arv), SAD(arv)



Percurso:
D B

Exemplo – Ordem simétrica

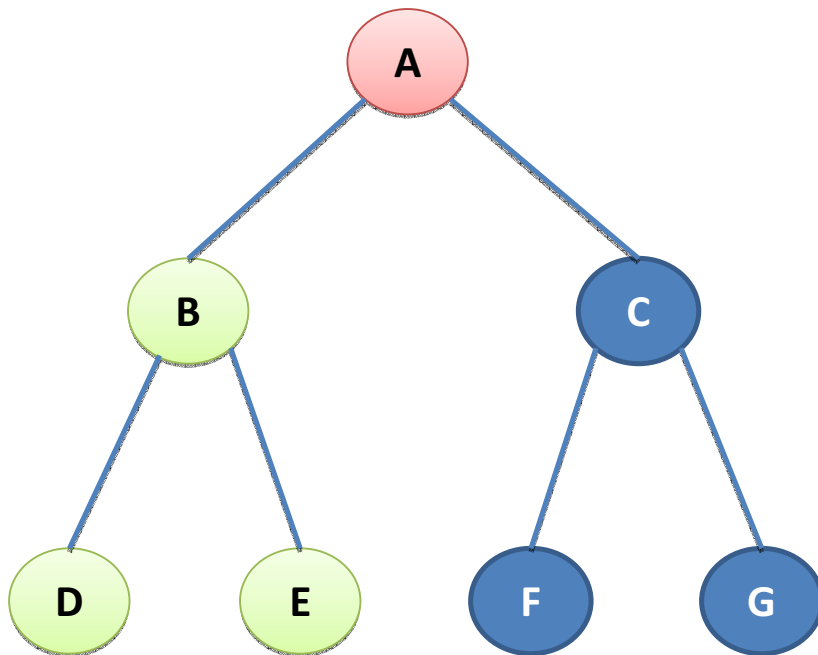
Ordem simétrica: SAE(arv), RAIZ(arv), SAD(arv)



Percurso:
D B E

Exemplo – Ordem simétrica

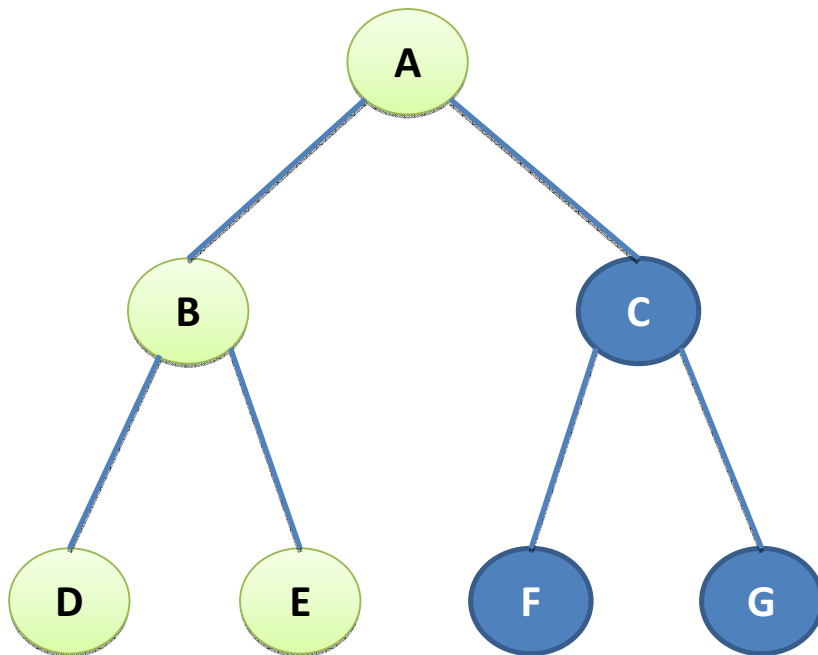
Ordem simétrica: SAE(arv), RAIZ(arv), SAD(arv)



Percurso:
D B E

Exemplo – Ordem simétrica

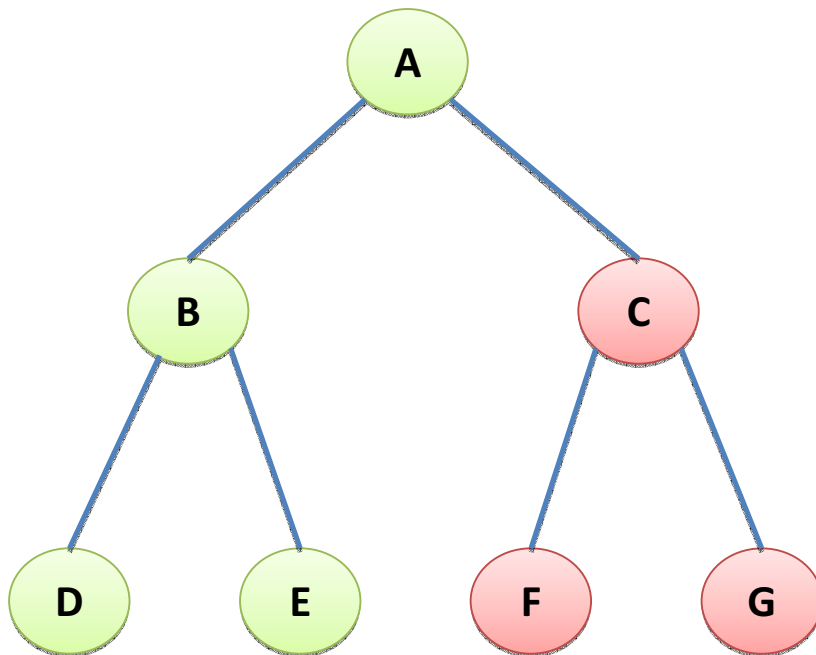
Ordem simétrica: SAE(arv), RAIZ(arv), SAD(arv)



Percurso:
D B E A

Exemplo – Ordem simétrica

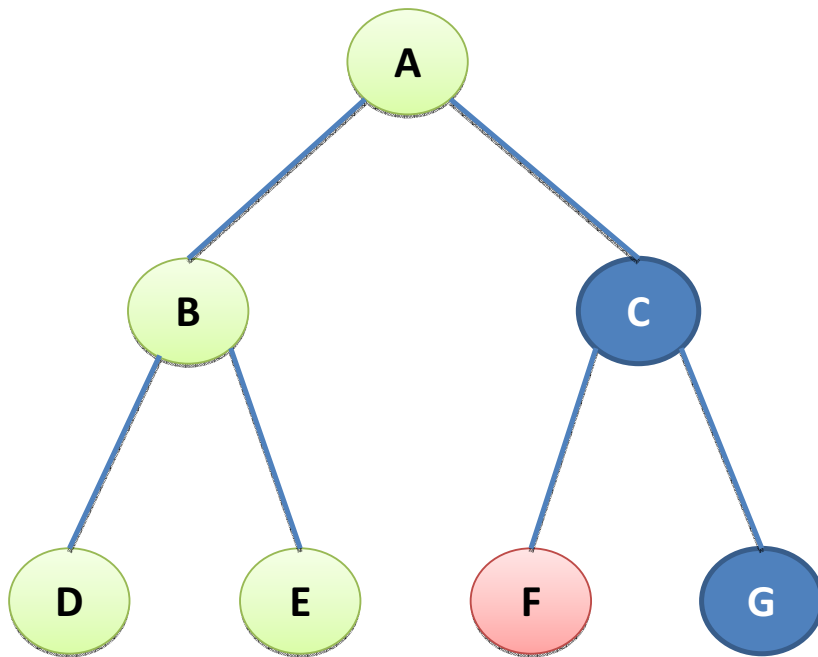
Ordem simétrica: SAE(arv), RAIZ(arv), SAD(arv)



Percurso:
D B E A

Exemplo – Ordem simétrica

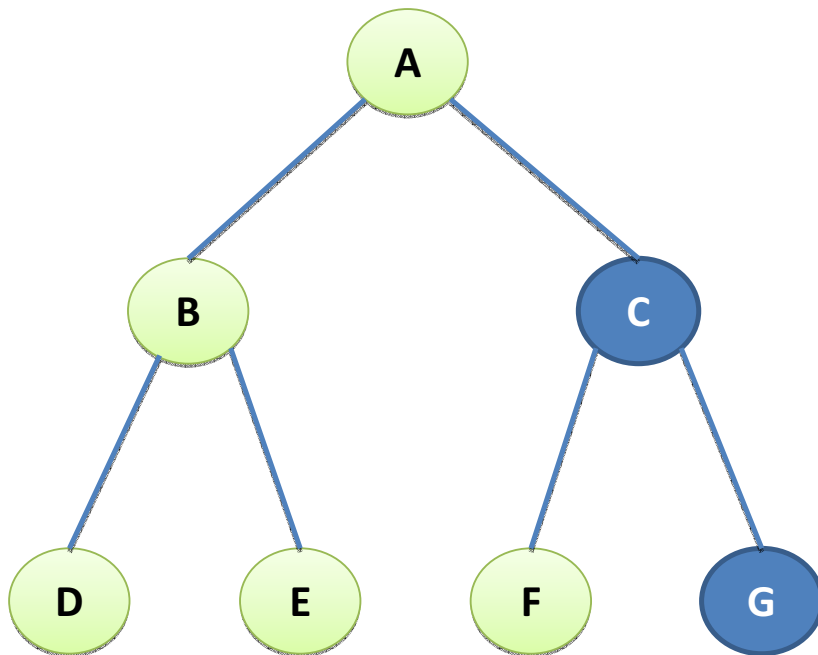
Ordem simétrica: SAE(arv), RAIZ(arv), SAD(arv)



Percurso:
D B E A

Exemplo – Ordem simétrica

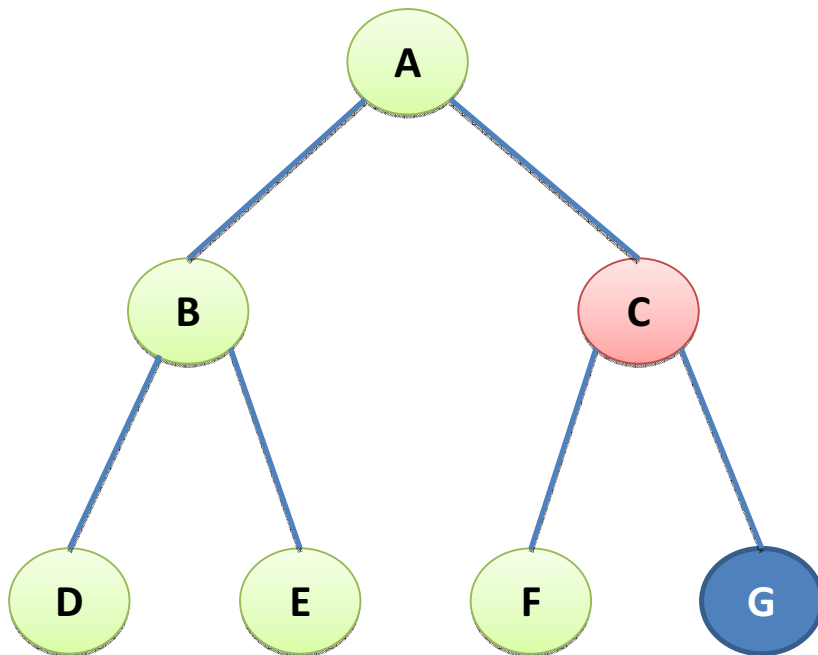
Ordem simétrica: SAE(arv), RAIZ(arv), SAD(arv)



Percurso:
D B E A F

Exemplo – Ordem simétrica

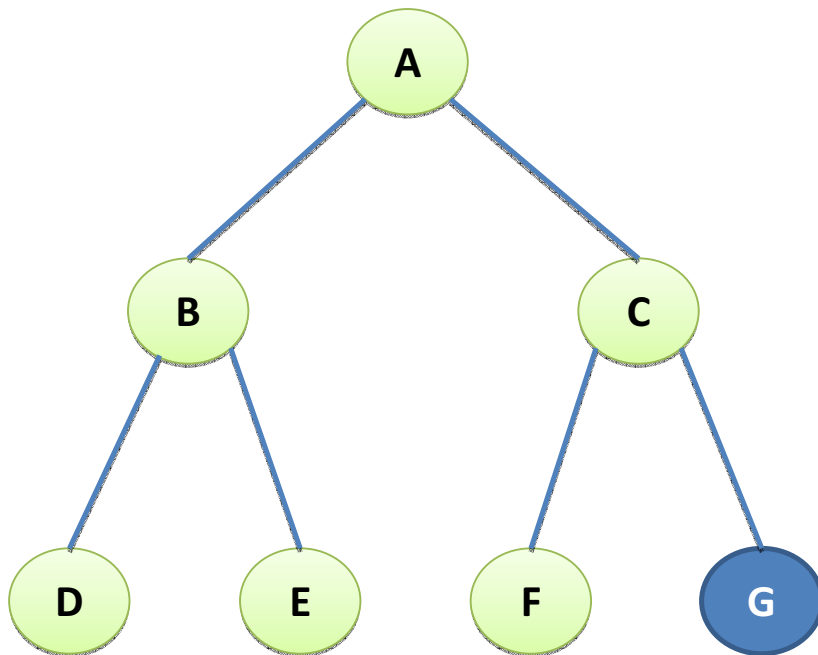
Ordem simétrica: SAE(arv), RAIZ(arv), SAD(arv)



Percurso:
D B E A F

Exemplo – Ordem simétrica

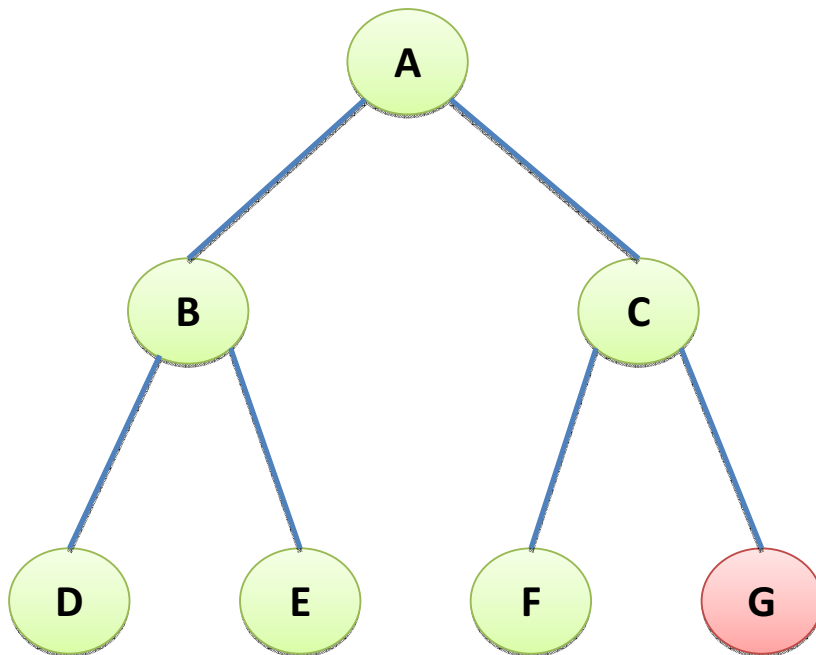
Ordem simétrica: SAE(arv), RAIZ(arv), SAD(arv)



Percurso:
D B E A F C

Exemplo – Ordem simétrica

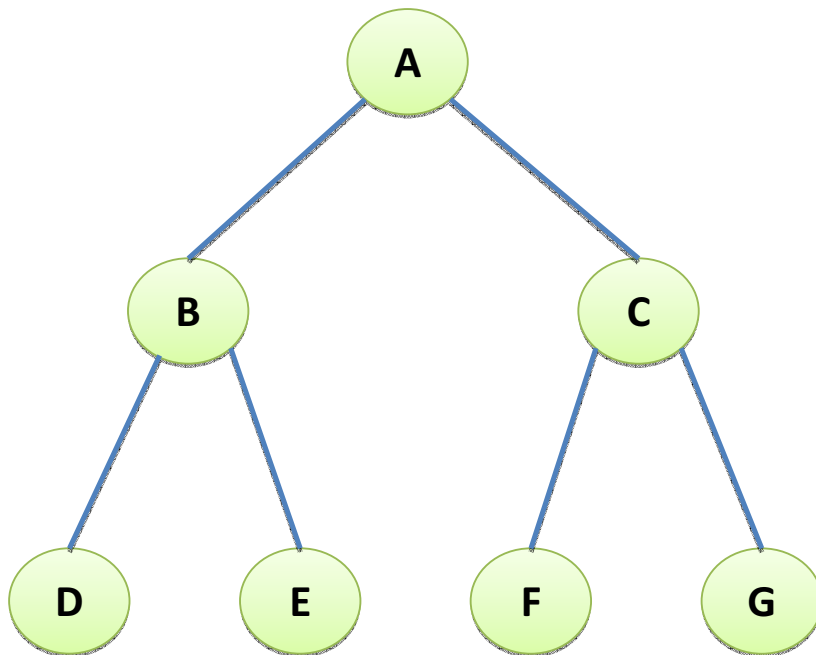
Ordem simétrica: SAE(arv), RAIZ(arv), SAD(arv)



Percurso:
D B E A F C

Exemplo – Ordem simétrica

Ordem simétrica: SAE(arv), RAIZ(arv), SAD(arv)

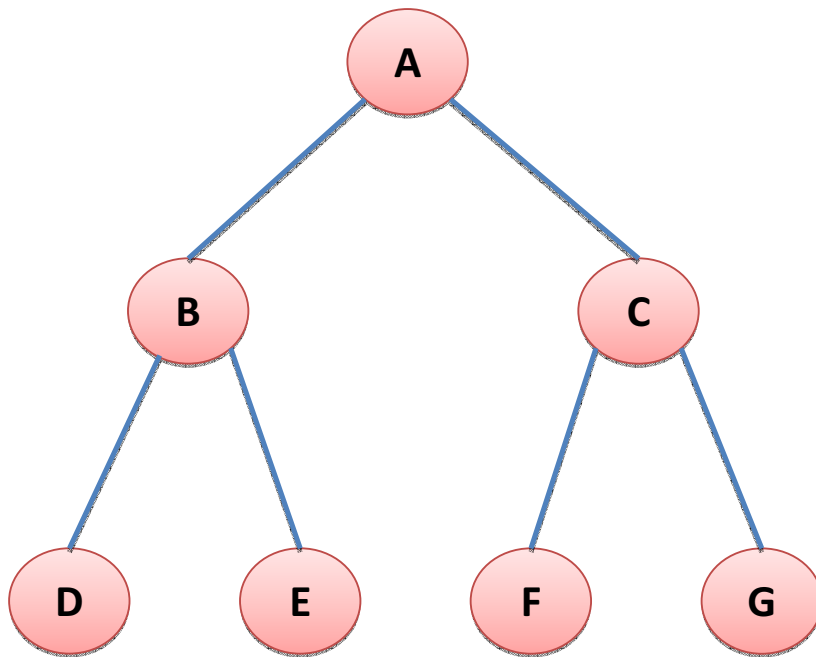


Percurso:

D B E A F C G

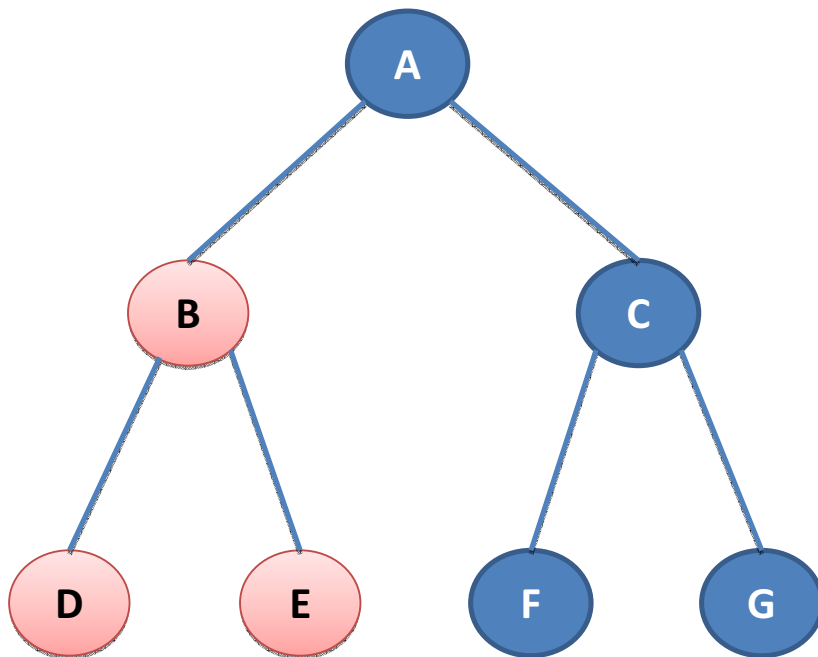
Exemplo – Pós-ordem

Pós-ordem: SAE(arv), SAD(arv), RAIZ(arv)



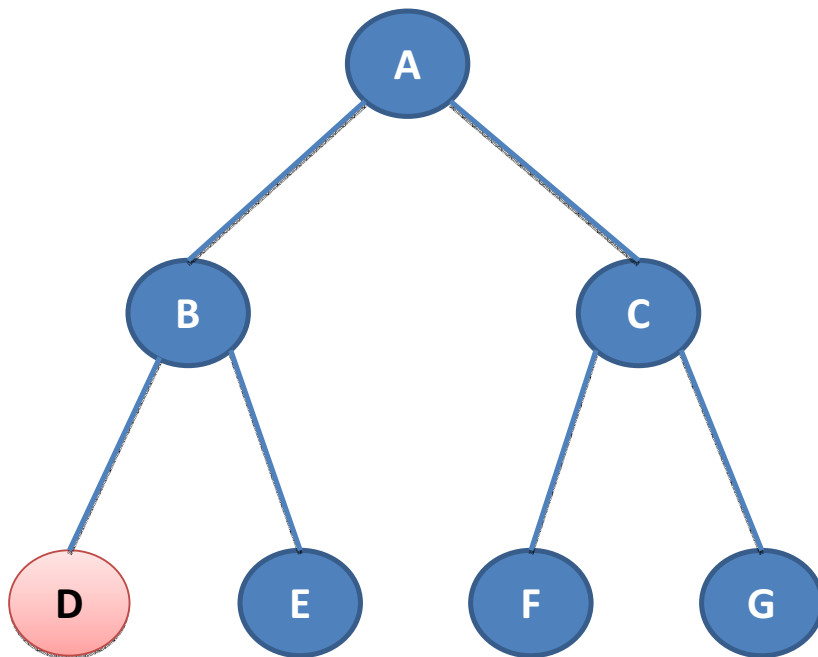
Exemplo – Pós-ordem

Pós-ordem: SAE(arv), SAD(arv), RAIZ(arv)



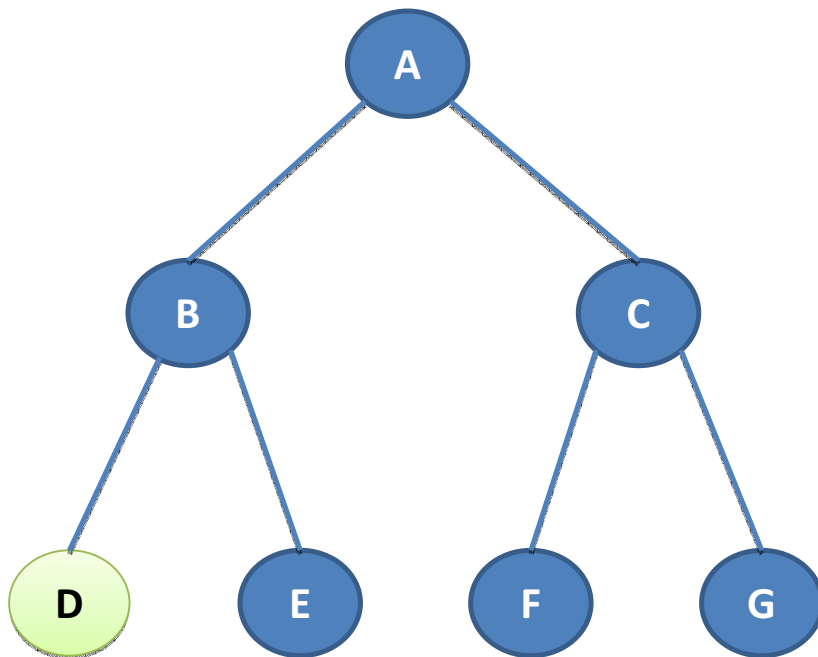
Exemplo – Pós-ordem

Pós-ordem: SAE(arv), SAD(arv), RAIZ(arv)



Exemplo – Pós-ordem

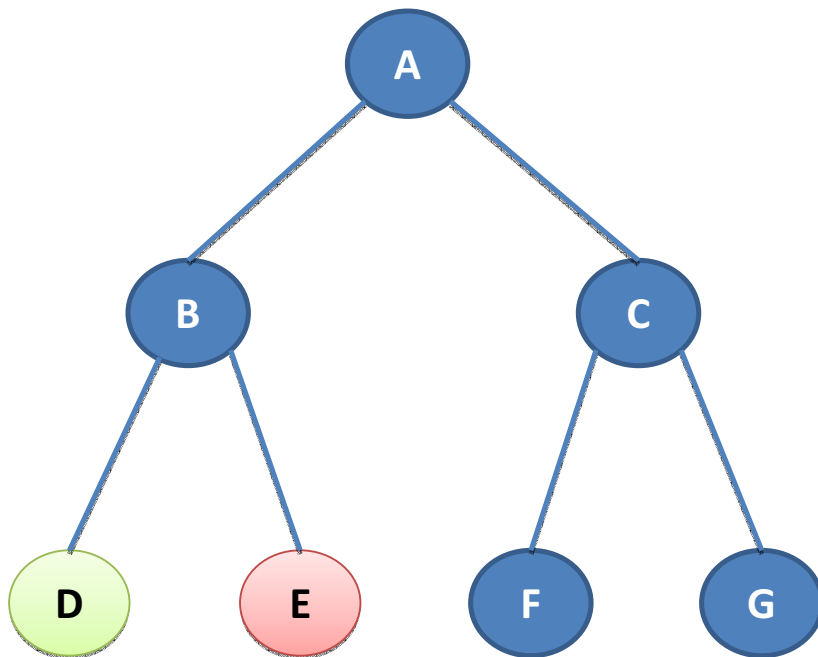
Pós-ordem: SAE(arv), SAD(arv), RAIZ(arv)



Percurso:
D

Exemplo – Pós-ordem

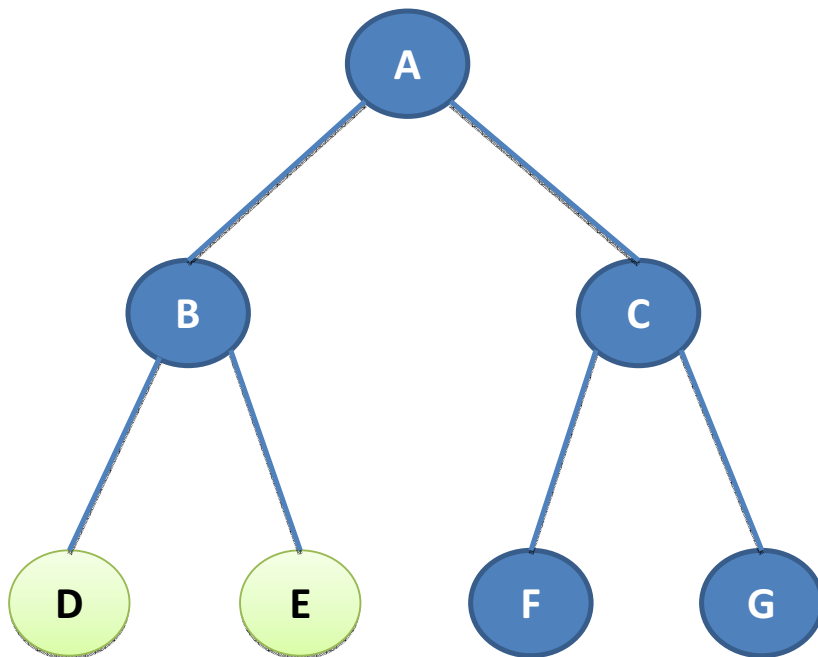
Pós-ordem: SAE(arv), SAD(arv), RAIZ(arv)



Percurso:
D

Exemplo – Pós-ordem

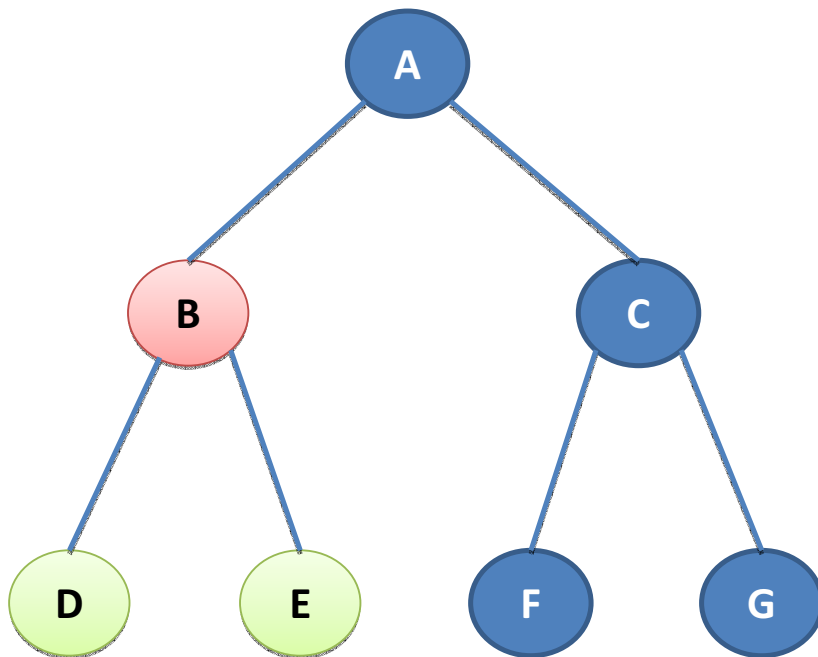
Pós-ordem: SAE(arv), SAD(arv), RAIZ(arv)



Percurso:
D E

Exemplo – Pós-ordem

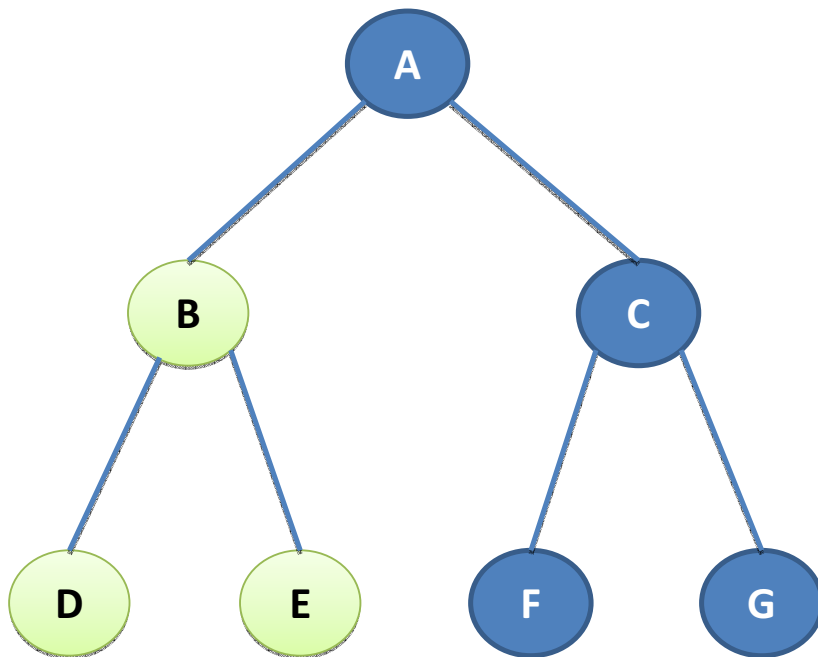
Pós-ordem: SAE(arv), SAD(arv), RAIZ(arv)



Percurso:
D E

Exemplo – Pós-ordem

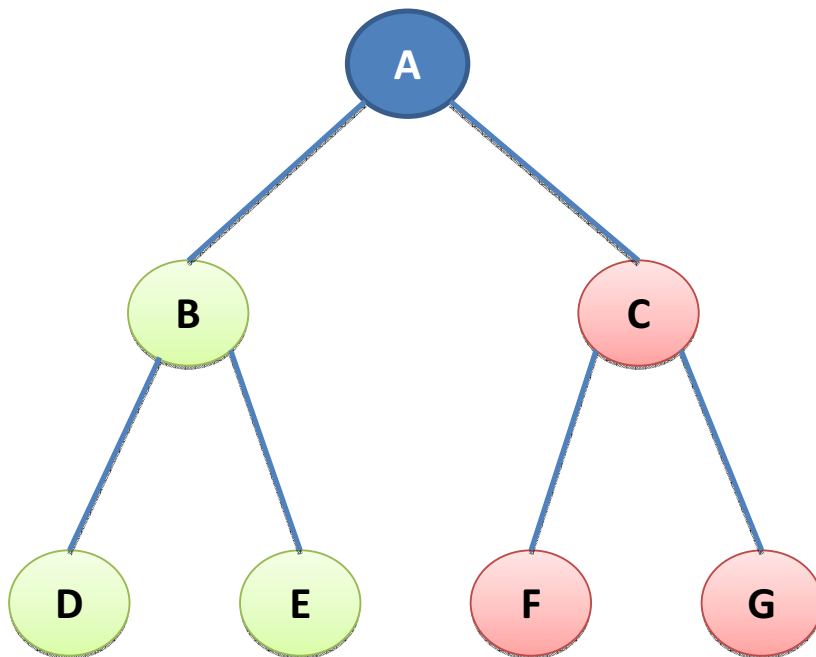
Pós-ordem: SAE(arv), SAD(arv), RAIZ(arv)



Percurso:
D E B

Exemplo – Pós-ordem

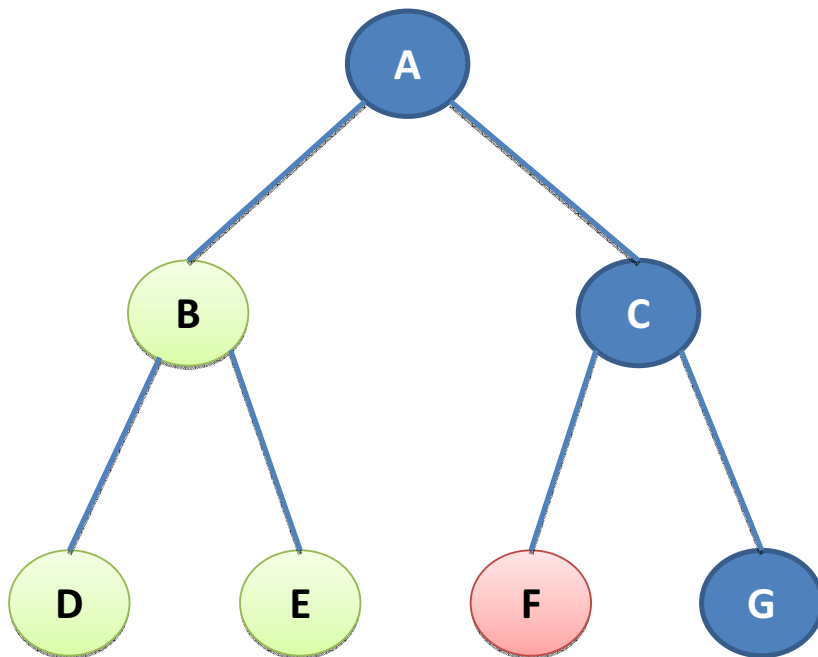
Pós-ordem: SAE(arv), SAD(arv), RAIZ(arv)



Percurso:
D E B

Exemplo – Pós-ordem

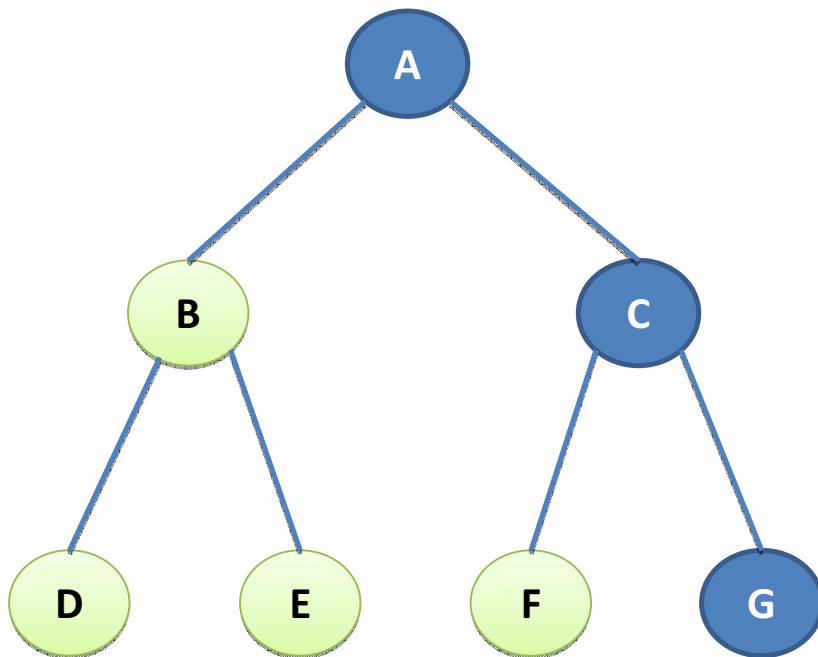
Pós-ordem: SAE(arv), SAD(arv), RAIZ(arv)



Percurso:
D E B

Exemplo – Pós-ordem

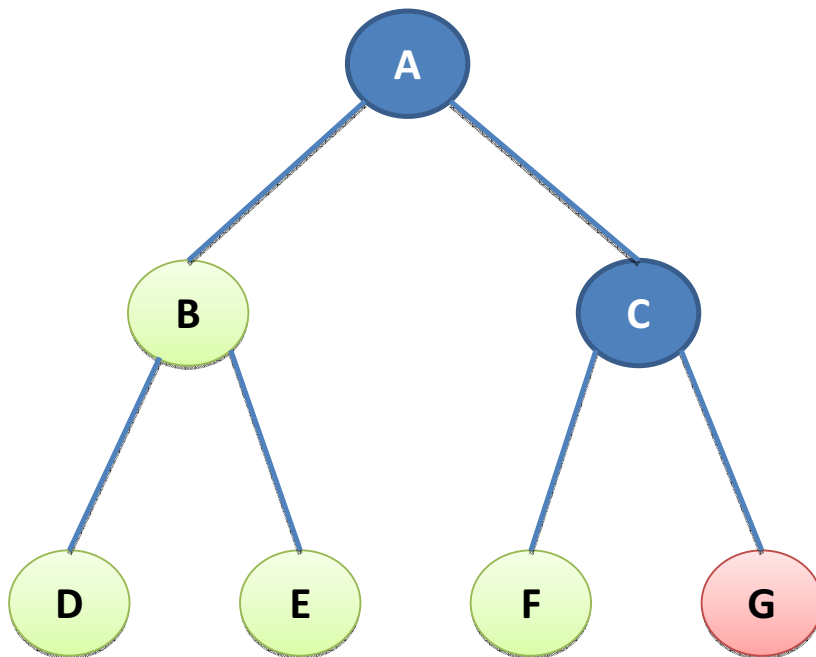
Pós-ordem: SAE(arv), SAD(arv), RAIZ(arv)



Percurso:
D E B F

Exemplo – Pós-ordem

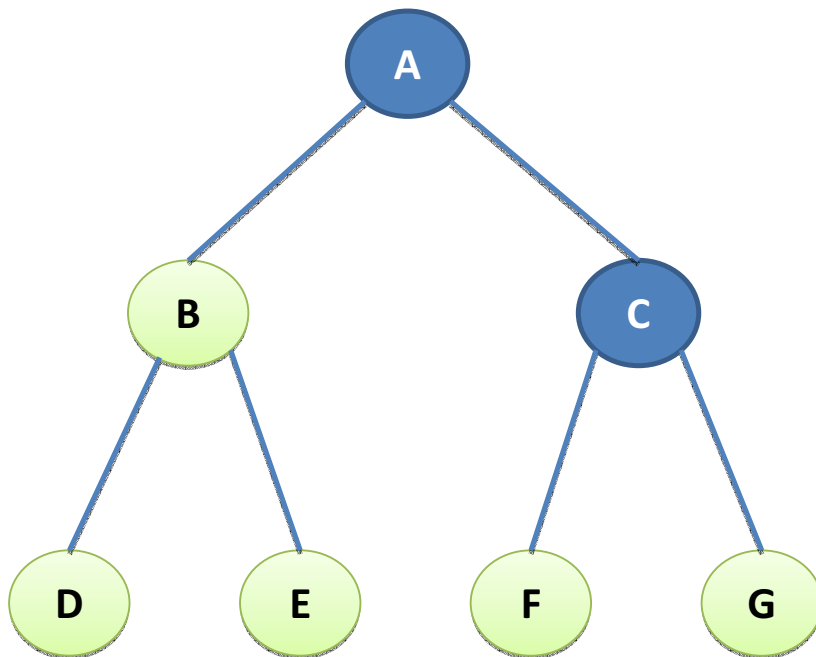
Pós-ordem: SAE(arv), SAD(arv), RAIZ(arv)



Percurso:
D E B F

Exemplo – Pós-ordem

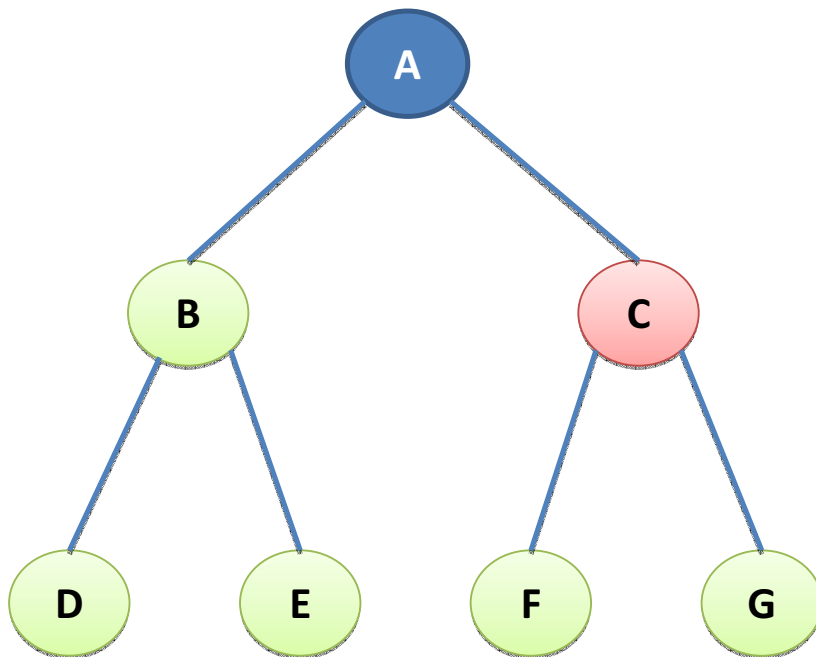
Pós-ordem: SAE(arv), SAD(arv), RAIZ(arv)



Percurso:
D E B F G

Exemplo – Pós-ordem

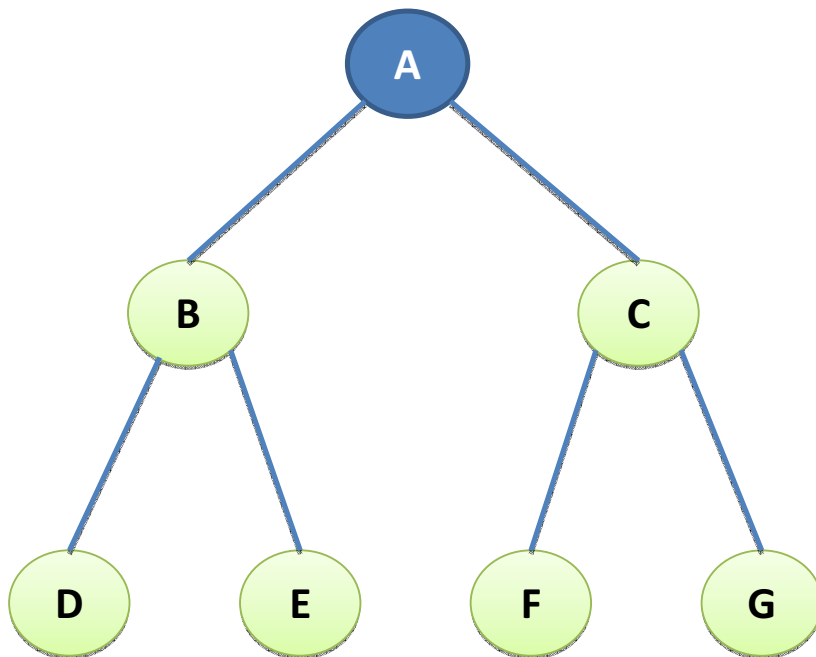
Pós-ordem: SAE(arv), SAD(arv), RAIZ(arv)



Percurso:
D E B F G

Exemplo – Pós-ordem

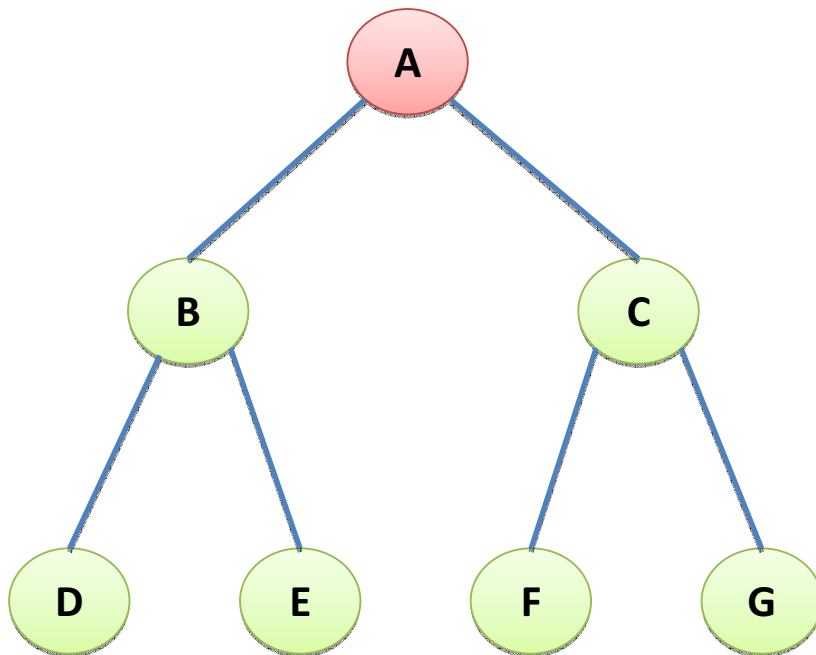
Pós-ordem: SAE(arv), SAD(arv), RAIZ(arv)



Percurso:
D E B F G C

Exemplo – Pós-ordem

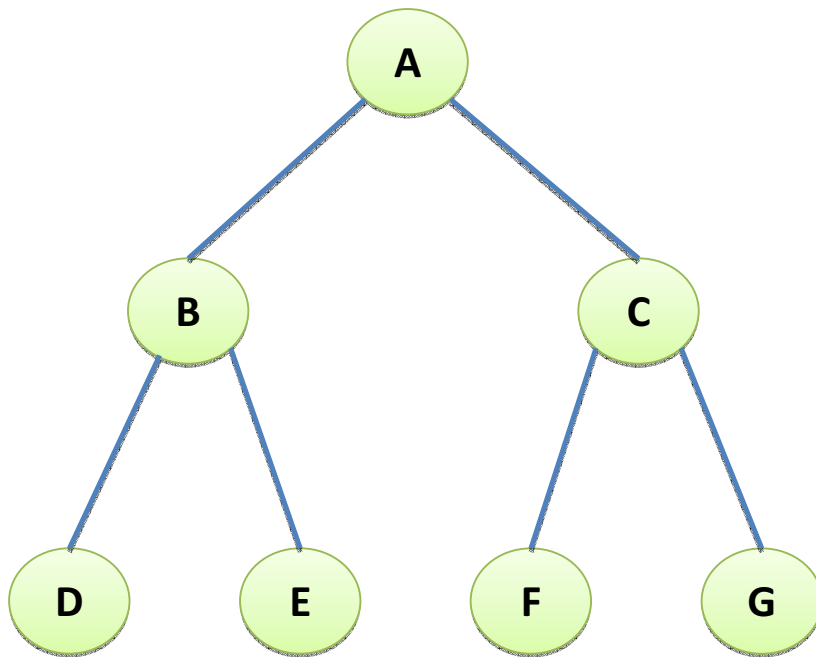
Pós-ordem: SAE(arv), SAD(arv), RAIZ(arv)



Percurso:
D E B F G C

Exemplo – Pós-ordem

Pós-ordem: SAE(arv), SAD(arv), RAIZ(arv)



Percurso:
D E B F G C A