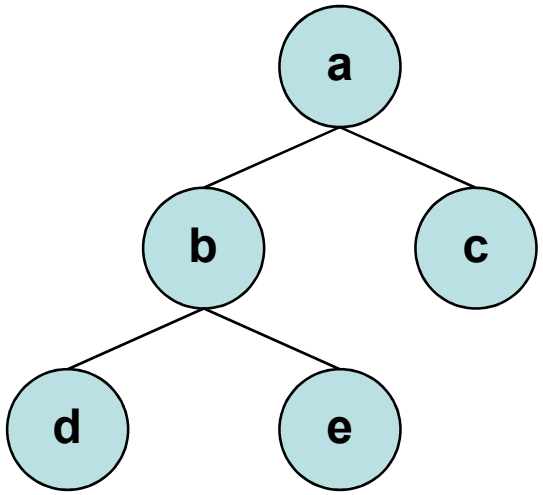
```
int brother(ARV_BIN_SEQ *t, int p) {
   if (father(t, p) != -1) /*Se não for a raiz*/
      if (isleft(t, p))
         return right(t,father(t, p));
      else
         return t->nodes[t->nodes[p].father].left;
   return -1;
}

int isleft(ARV_BIN_SEQ *t, int p) {
   int q = father(t, p);
   if (q == -1) /*Se for a raiz*/
      return (0);
   if (left(t,q) == p)
      return (1);
   return (0);
}

int isright(ARV_BIN_SEQ *t, int p) {
   if (father(t, p) != -1)
      return (!isleft(t,p));
   return (0);
}
```

Árvores Binárias

Existem formas de armazenamento estático alternativas à apresentada. Por exemplo, vamos considerar a árvore binária abaixo.

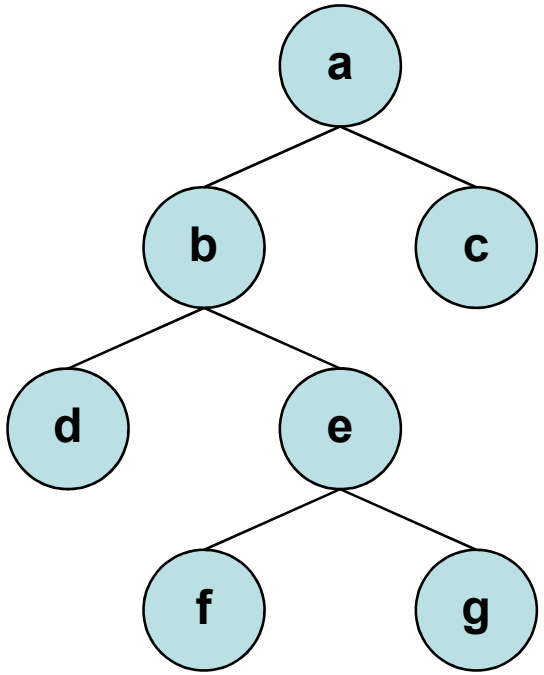


Esta Árvore poderia ser armazenada em um vetor da seguinte forma:

a	b	c	d	e

Árvores Binárias

Se a árvore anterior passasse a ter a seguinte configuração:



Como ficaria o armazenamento no vetor?

Assim

a	b	c	d	e	f	g

?

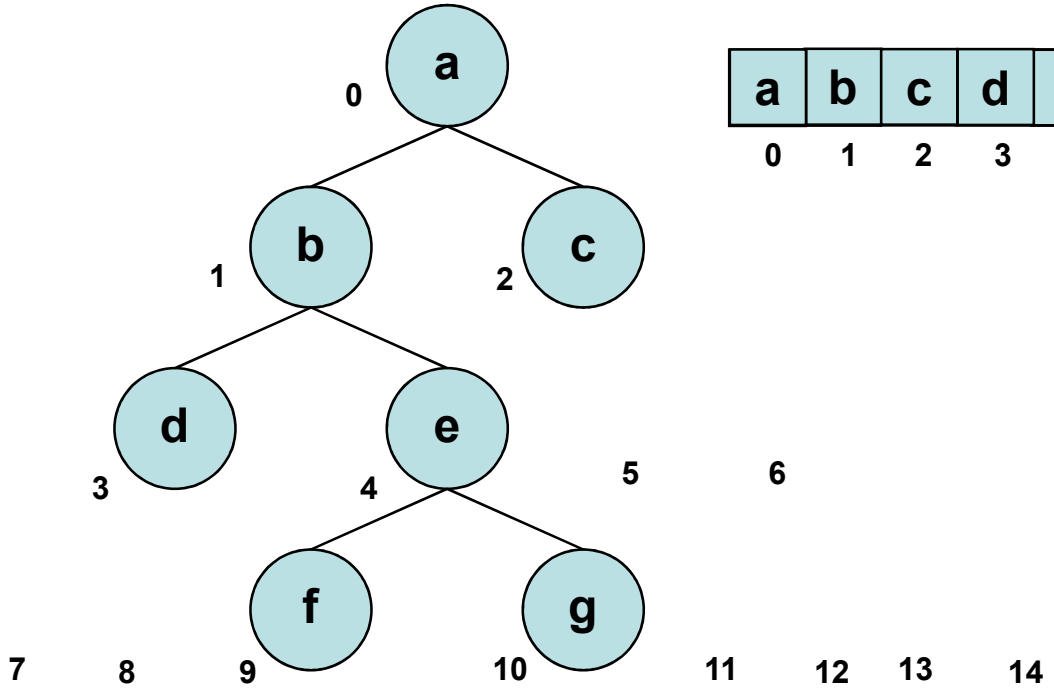
E se fosse assim:

a	b	c	d	e					f	g				

Vocês identificam alguma vantagem?

Árvores Binárias

Árvore:



Representação:

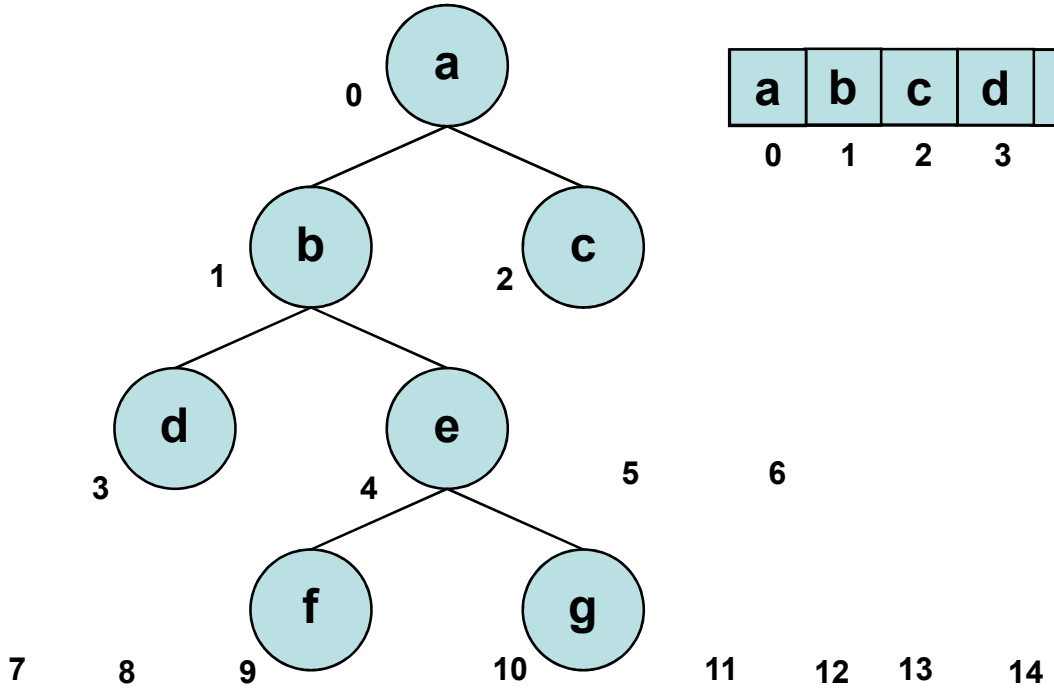
a	b	c	d	e					f	g				
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

A raiz está aonde?

No índice 0 (zero).

Árvores Binárias

Árvore:



Representação:

a	b	c	d	e					f	g				
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

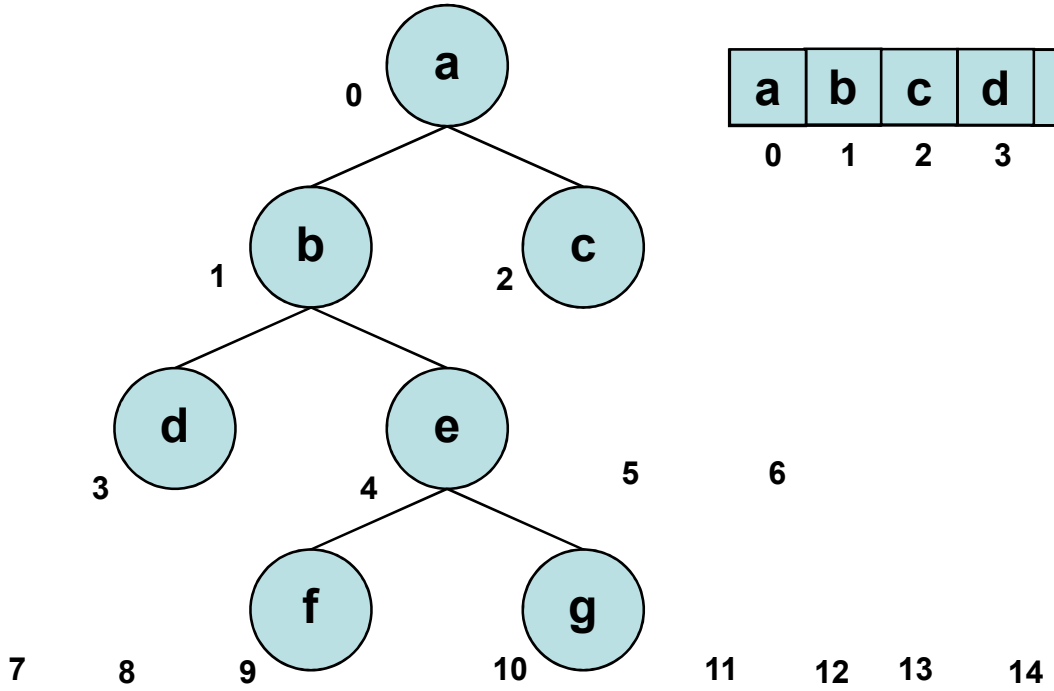
Como saber quem é o pai de um nó?

(p-1)/2.

p é índice do vetor que contém o nó.

Árvores Binárias

Árvore:



Representação:

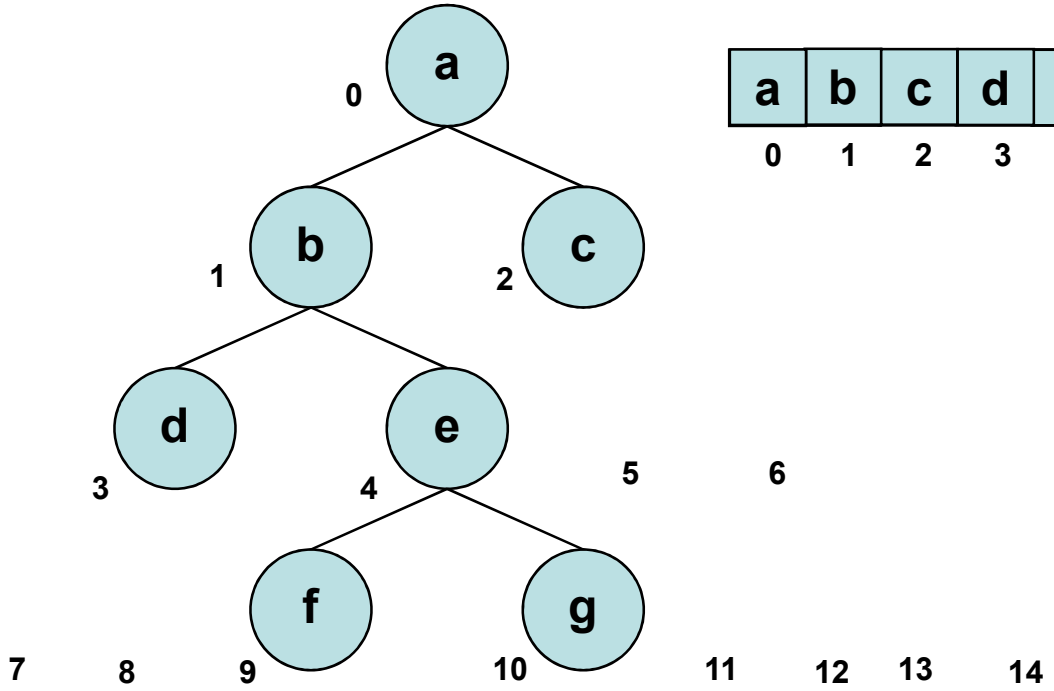
a	b	c	d	e					f	g				
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Como saber quem é o filho esquerdo de um nó?

2*p+1

Árvores Binárias

Árvore:



Representação:

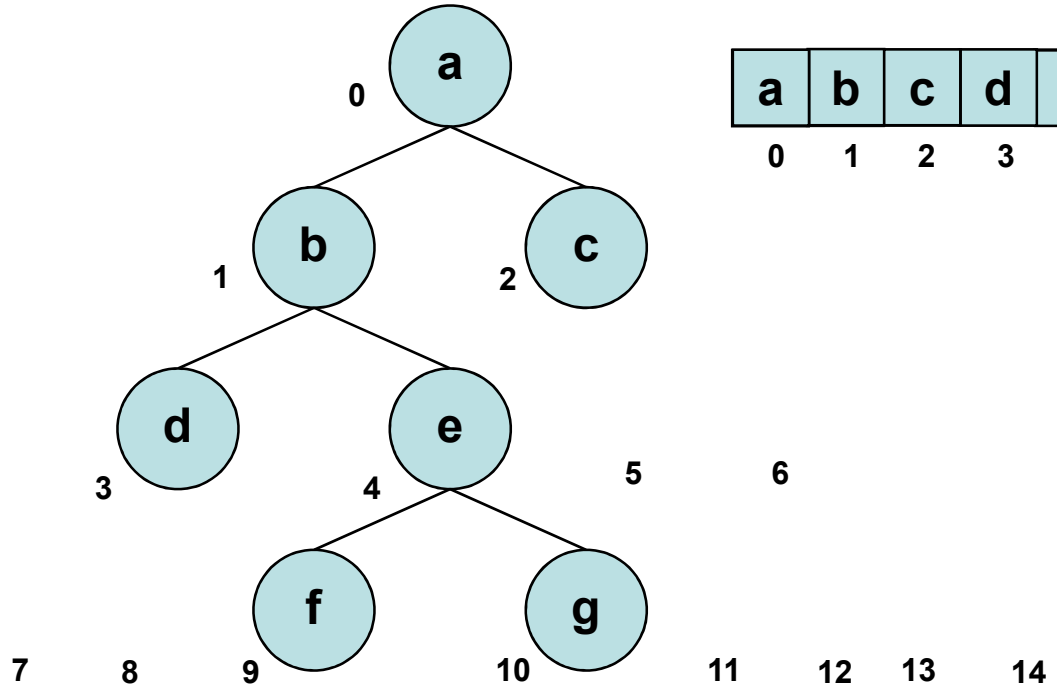
a	b	c	d	e					f	g				
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Como saber quem é o filho direito de um nó?

2*p+2

Árvores Binárias

Árvore:



Representação:

a	b	c	d	e					f	g				
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Como saber onde está o irmão direito de um nó?

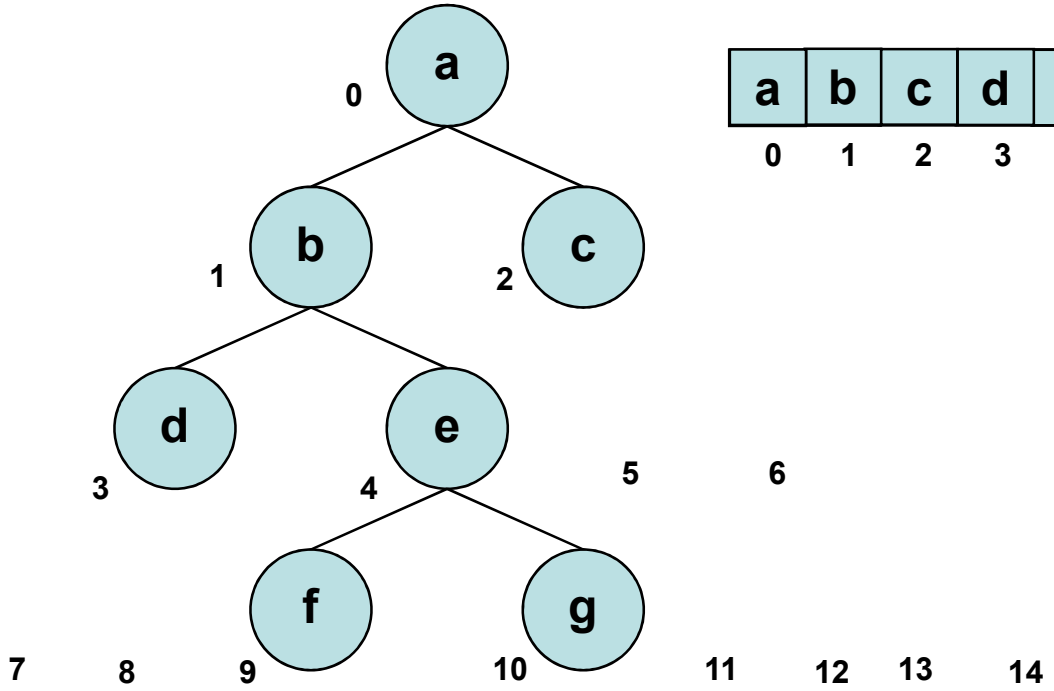
Primeiro, devemos saber se ele é um filho esquerdo.

Como saber?

É só verificar se p é impar.

Árvores Binárias

Árvore:



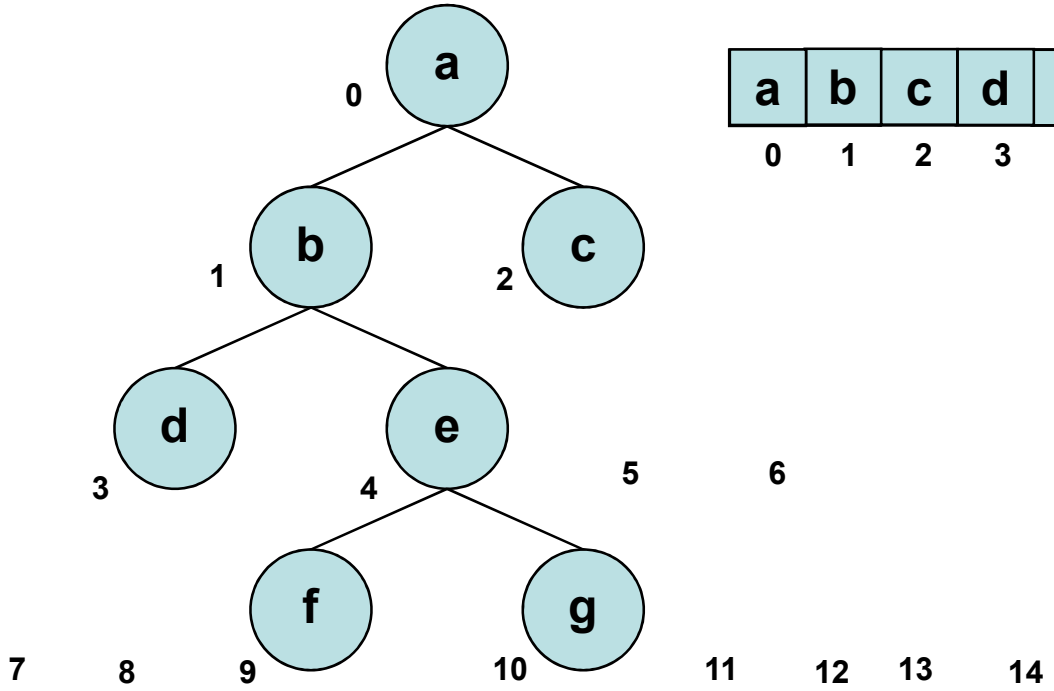
Representação:

a	b	c	d	e					f	g				
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Se um nó for um filho a esquerda, para saber onde está o seu irmão direito basta somar uma unidade a p.

Árvores Binárias

Árvore:



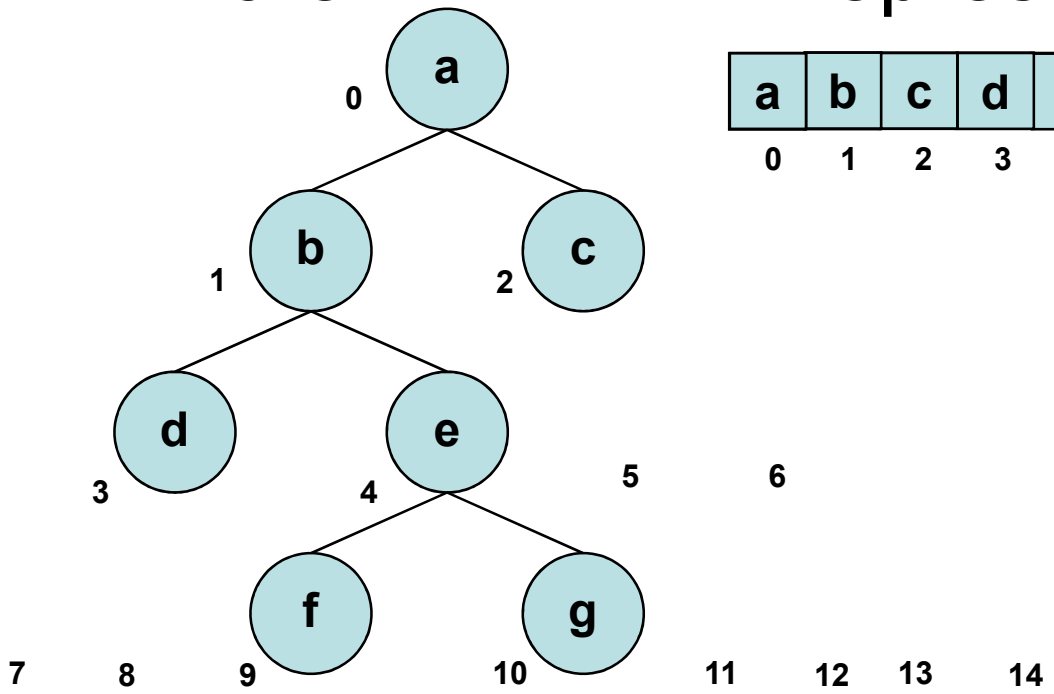
Representação:

a	b	c	d	e					f	g				
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

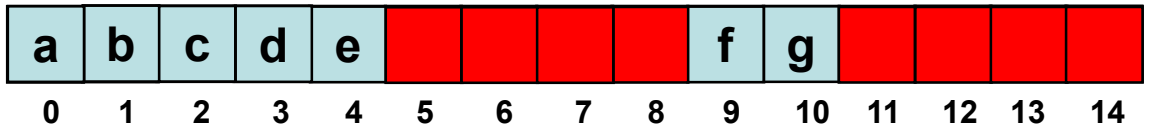
De forma análoga determinamos se um nó é um filho à direita e onde está o seu irmão esquerdo.

Árvores Binárias

Árvore:



Representação:



Como saber se d não tem filho esquerdo e/ou direito?

Com a inserção de um campo adicional em cada nó, que identificará se este está ou não ocupado. Assim, ao computar onde está o filho direito de d (3*2+2=8) verifica-se o valor deste campo adicional.

Árvores Binárias

Esta forma de representação é denominada representação implícita em vetores de árvores binárias.

Para armazenar uma árvore binária completa com profundidade igual a d será necessário um vetor com $2^{d+1} - 1$.

A implementação da representação implícita em vetores de árvores binárias é uma boa proposta de exercício.

Contudo, visando otimizar a utilização da memória, temos a possibilidade de armazenar uma árvore binária dinamicamente.

UNIVASF

Árvores Binárias

Para tal precisaremos de uma estrutura para os nós da seguinte forma:

```
typedef struct node
{
    int info;
    struct node *left;
    struct node *right;
    struct node *father;
} NODE;
```

Uma árvore binária encadeada seria defina como:

```
typedef NODE * ARV_BIN_ENC;
```

Árvores Binárias - Alocação Encadeada

Com base no que foi visto implemente as operações que compõem o TAD ARV_BIN_ENC.

```
/*Definição a ARV_BIN_ENC*/
typedef struct node
{
      int info;
      struct node *left;
      struct node *right;
      struct node *father;
} NODE;
typedef NODE * ARV_BIN_ENC;
void maketree(ARV_BIN_ENC *, int);
void setleft(ARV_BIN_ENC, int);
void setright(ARV_BIN_ENC, int);
int info(ARV_BIN_ENC);
ARV_BIN_ENC left(ARV_BIN_ENC);
ARV_BIN_ENC right(ARV_BIN_ENC);
ARV_BIN_ENC father(ARV_BIN_ENC);
ARV_BIN_ENC brother(ARV_BIN_ENC);
int isleft(ARV_BIN_ENC);
int isright(ARV_BIN_ENC);
```

```
void maketree(ARV_BIN_ENC *t, int x)
{
   *t  = (ARV_BIN_ENC) malloc (sizeof (NODE));
   if (!(*t))
   {
      printf("Erro! Nao existe memoria disponivel!");
      exit (1);
   }
   (*t)->info = x;
   (*t)->left = NULL;
   (*t)->right = NULL;
   (*t)->father = NULL;
}
```

```
void setleft(ARV_BIN_ENC t, int x)
{
   t->left  = (ARV_BIN_ENC) malloc (sizeof (NODE));
   if (!(t->left))
   {
      printf("Erro! Nao existe memoria disponivel!");
      exit (1);
   }
   t->left->info = x;
   t->left->left = NULL;
   t->left->right = NULL;
   t->left->father = t;
}
```

```
void setright(ARV_BIN_ENC t, int x)
{
   t->right  = (ARV_BIN_ENC) malloc (sizeof (NODE));
   if (!(t->right))
   {
      printf("Erro! Nao existe memoria disponivel!");
      exit (1);
   }
   t->right->info = x;
   t->right->left = NULL;
   t->right->right = NULL;
   t->right->father = t;
}
```

```
int info(ARV_BIN_ENC t)
{
  return t->info; /*Se a arvore for vazia?*/
}

ARV_BIN_ENC left(ARV_BIN_ENC t)
{
  return t->left;
}

ARV_BIN_ENC right(ARV_BIN_ENC t)
{
  return t->right;
}

ARV_BIN_ENC father(ARV_BIN_ENC t)
{
  return t->father;
}
```

```
ARV_BIN_ENC brother(ARV_BIN_ENC t) {
   if (father(t)) /*Se não for a raiz*/
      if (isleft(t))
         return right(father(t));
      else
         return left(father(t));
   return NULL;
}

int isleft(ARV_BIN_ENC t) {
   NODE *q = father(t);
   if (!q)
      return (0);
   if (left(q) == t)
      return (1);
   return (0);
}

int isright(ARV_BIN_ENC t) {
   if (father(t))
      return (!isleft(t));
   return (0);
}
```

Árvores Binárias - Aplicações

Uma árvore binária é uma estrutura de dados útil quando precisam ser tomadas decisões bidirecionais em cada ponto de um processo.

Por exemplo, suponha que precisemos encontrar todas as repetições numa lista de números. Uma maneira de fazer isto é comparar cada número com todos que o precedem.

Entretanto, isso envolve um grande número de comparações.

O número de comparações pode ser reduzido usando-se uma árvore binária.

O primeiro número na lista é colocado num nó estabelecido como a raiz de uma árvore binária com as subárvores esquerda e direita vazias.

Árvores Binárias - Aplicações

Cada número sucessivo na lista é, então, comparado ao número na raiz. Se coincidirem, teremos uma repetição.

Se for menor, examinaremos a subárvore esquerda; se for maior, examinaremos a subárvore direita.

Se a subárvore estiver vazia, o número não será repetido e será colocado num novo nó nesta posição na árvore.

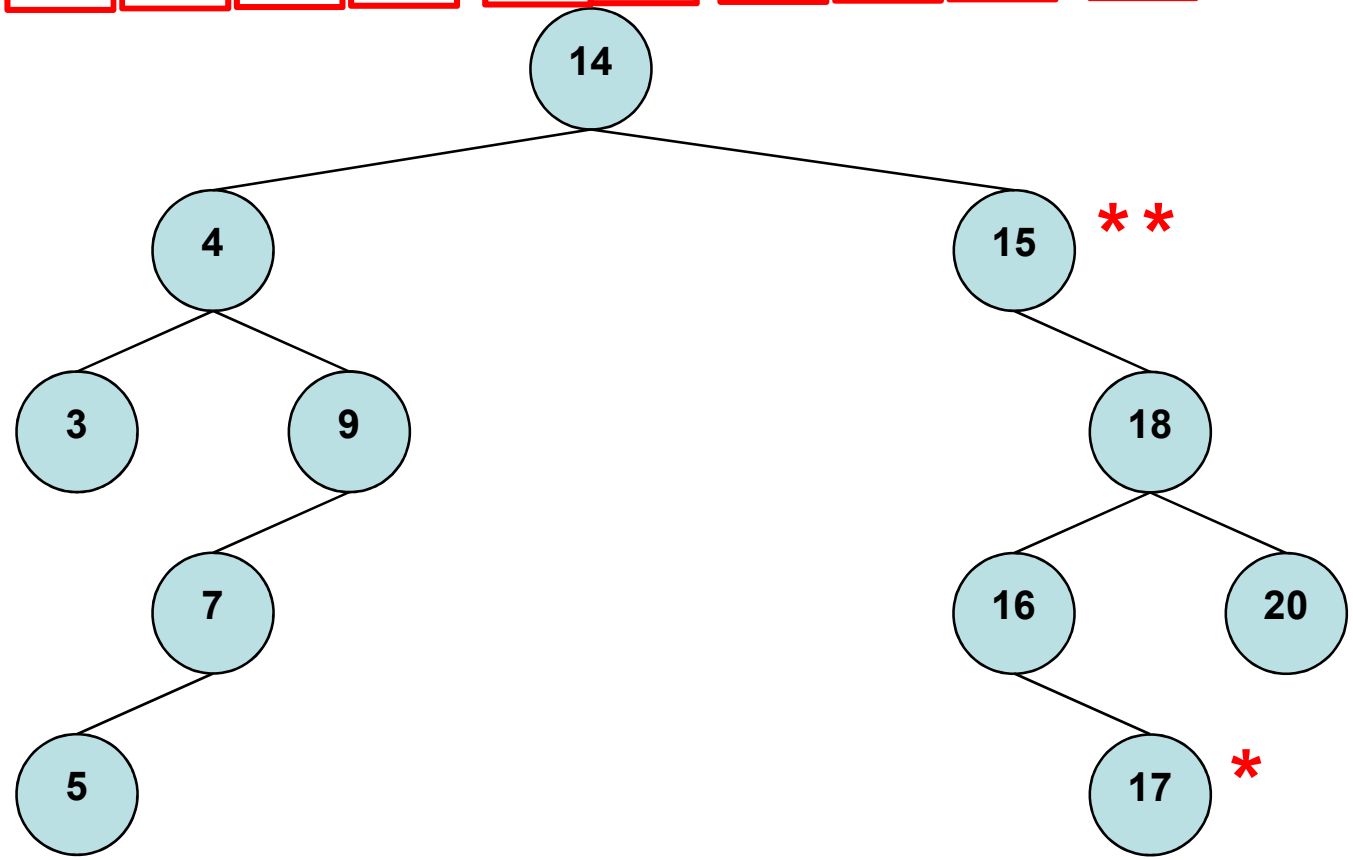
Se a subárvore não estiver vazia, compararemos o número ao conteúdo da raiz da subárvore e o processo inteiro será repetido com a subárvore.

UNIVASF

Árvores Binárias - Aplicações

Veja a seguir um exemplo.

14	4	15	3	15	9	18	7	15	16	5	17	17	20



Árvores Binárias - Aplicações

Como outra aplicação das árvores binárias, temos o método de representar uma expressão aritmética contendo operandos e operadores binários por uma árvore estritamente binária.

Onde a raiz da árvore estritamente binária conterá um operador que deve ser aplicado aos resultados das avaliações das expressões representadas pelas subárvores esquerda e direita.

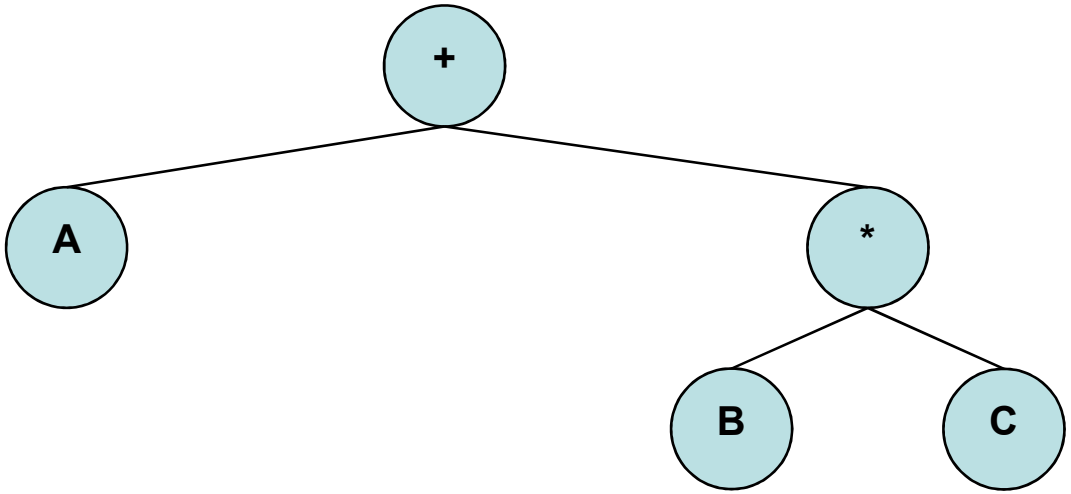
Um nó representando um operador é um nó que não é folha, enquanto um nó representando um operando é um folha.

UNIVASF

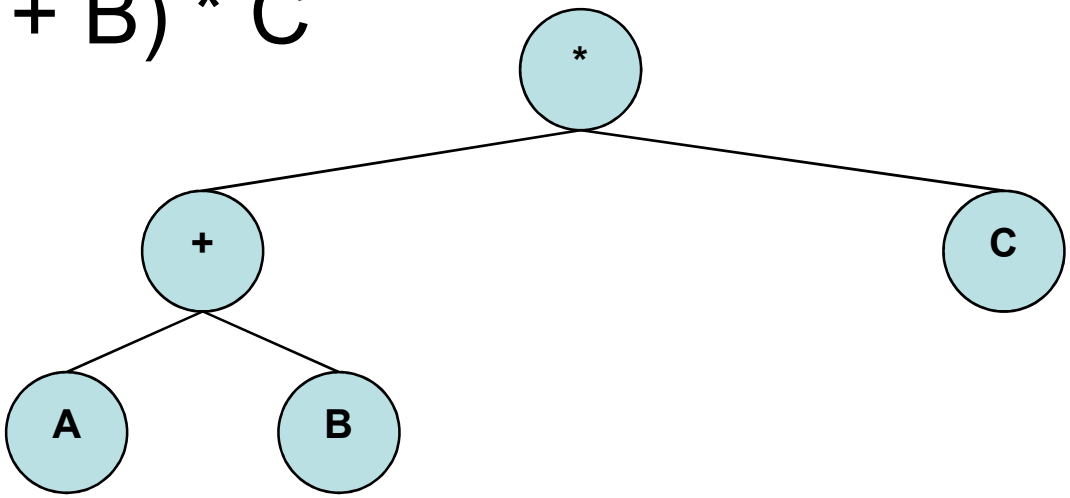
Árvores Binárias - Aplicações

Vejamos alguns exemplos:

A + B * C

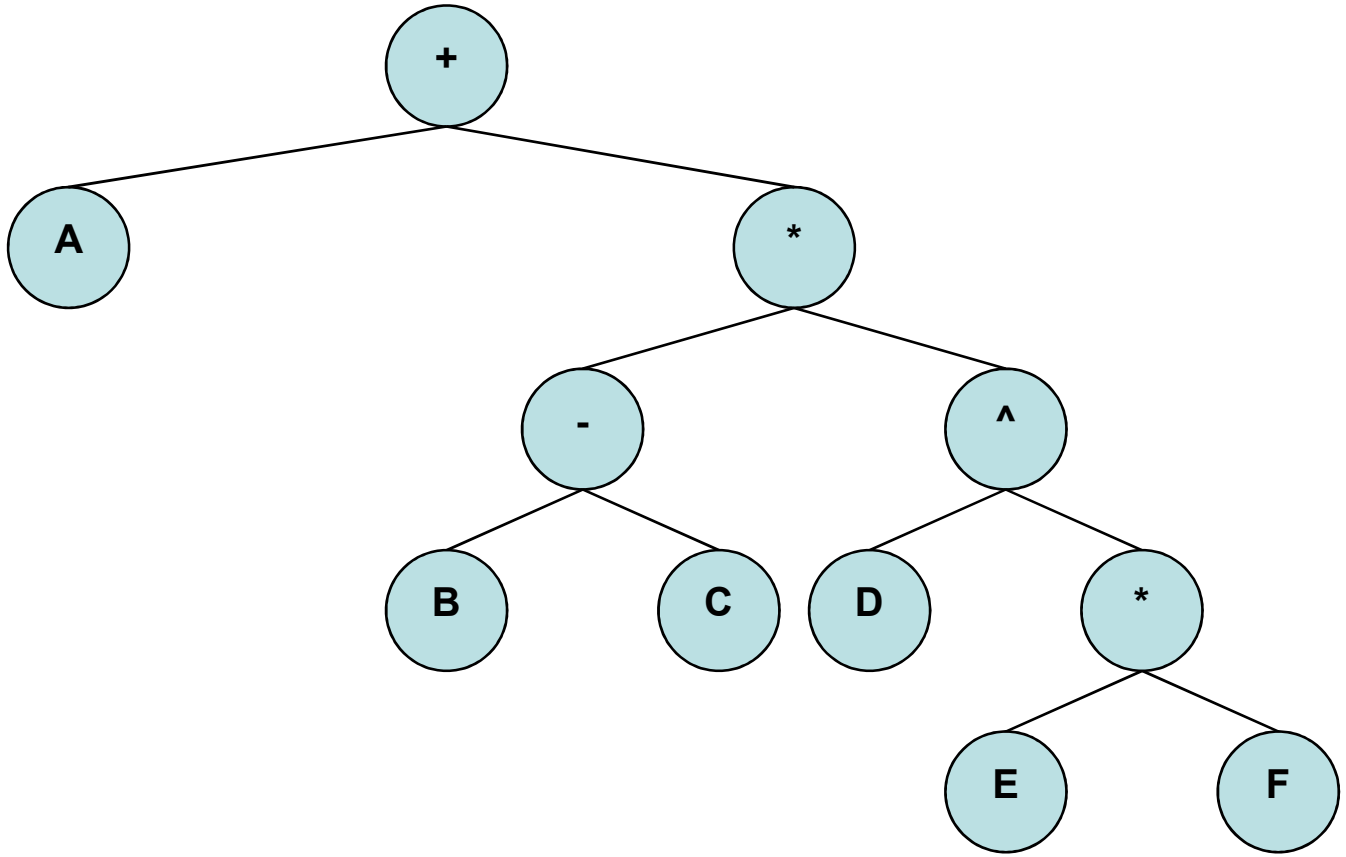


(A + B) * C



Árvores Binárias - Aplicações

A + (B - C) * D ^ (E * F)



Qual a vantagem de se representar uma expressão assim?

Veremos em breve, quando estudarmos formas de percurso em árvores.

Árvore binária de busca

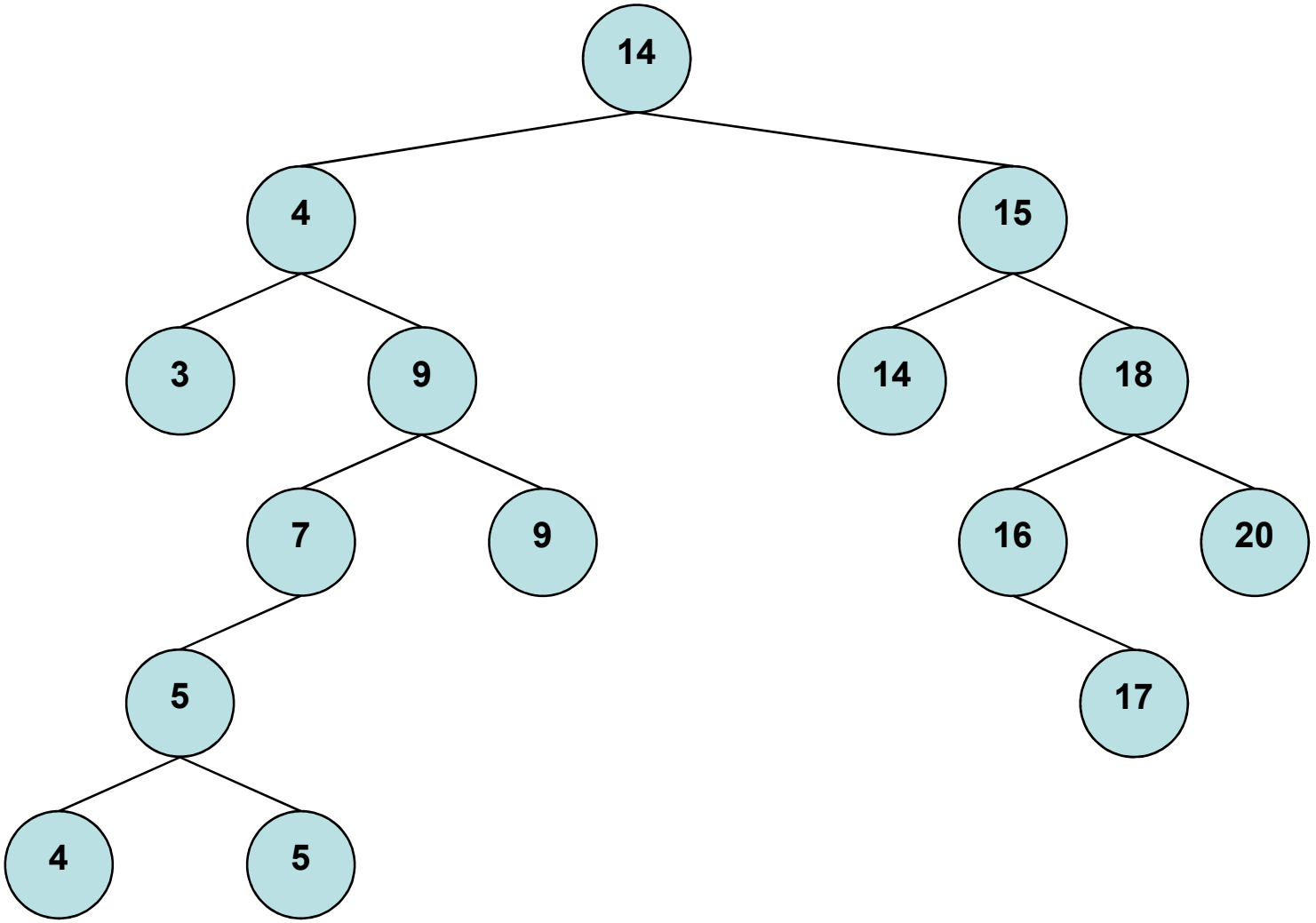
Veremos agora a definição de uma árvore binária de busca.

Também chamadas de árvore binária ordenada, é uma árvore binária com a seguinte propriedade: para cada nó n da árvore, todos os valores armazenados em sua subárvore à esquerda (a árvore cuja raiz é o filho à esquerda) são menores que o valor v armazenado em n, e todos os valores armazenados na subárvore à direita são maiores ou iguais a v.

UNIVASF

Árvore binária de busca

Exemplo de uma árvore binária de busca.



Qual seria sua aplicação?

Árvore binária de busca

Como é feita a localização de um elemento em uma árvore binária de busca?

Para cada nó, partindo da raiz, compare a chave a ser localizada com o valor armazenado no nó correntemente apontado.

Se a chave for menor que o valor, vá para a subárvore à esquerda e tente novamente.

Se for maior que o valor, tente a subárvore à direita.

Se for o mesmo, a busca poderá parar*.

A busca também é abortada se não há meios de continuar, indicando que a chave não está na árvore.

***(isto depende da árvore possuir ou não elementos/nós com o mesmo valor)**

UNIVASF

Árvore binária de busca – Modos de travessia

O percurso em uma árvore pressupõe visitar cada nó da árvore exatamente uma vez. Este processo pode ser interpretado com...

colocar todos os nós em uma linha ou a linearização da árvore.

Em uma árvore com n nós existem quantos percursos diferentes?

n! percursos diferentes.

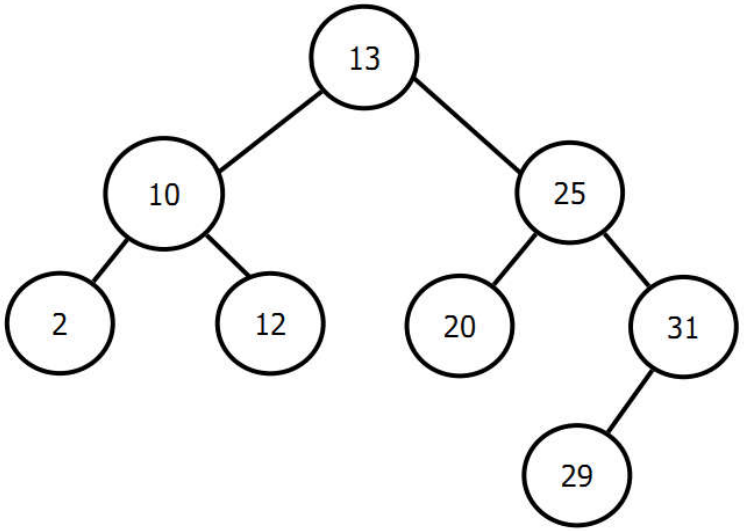
Diante de tamanha quantidade de opções, iremos restringir este universo a duas classes de percursos.

Percurso em largura e em profundidade.

Árvore binária de busca – Modos de travessia

Em que consiste o percurso em largura?

Consiste em visitar cada nó começando no nível mais baixo (nível da raiz) e movendo-se para as folhas nível a nível, visitando todos os nós em cada nível da esquerda para a direita (ou da direita para a esquerda).



Sequência de nós visitados
13, 10, 25, 2, 12, 20, 31, 29

Exercício: considerando a existência do TAD ARV_BIN_ENC (implementado anteriormente), implemente o percurso em largura.

Árvore binária de busca – Modos de travessia

A implementação deste tipo de percurso é direta quando uma fila é utilizada.

Considere um percurso em largura de cima para baixo da esquerda para a direita.

Depois que um nó é visitado, seus filhos, se houver algum, são colocados no final da fila e o nó no início da fila é visitado.

Considere que para um nó no nível ***n***, seus filhos estão no nível ***n+1***, colocando-se esses filhos no final da fila, eles serão visitados depois que todos os nós do nível ***n*** forem visitados.

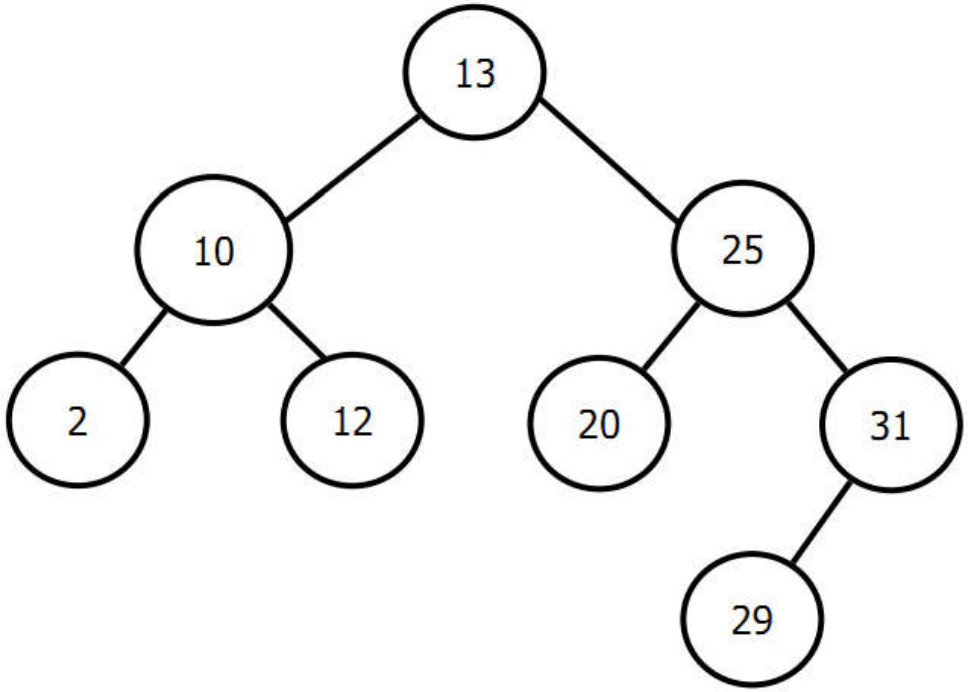
Assim, a restrição de que todos os nós no nível ***n*** precisam ser visitados antes de visitar quaisquer nós do nível ***n+1*** será satisfeita.

UNIVASF

```
void precursoEmLargura(ARV_BIN_ENC arvore)
{
  FILA_ENC fila; /*FILA_ENC armazena ARV_BIN_ENCs*/
  cria_fila(&fila);
  if (arvore)
    ins_fila (fila, arvore);
  while (!eh_vazia_fila(fila))
  {
    printf("%d ", info(cons_fila(fila))); /*Visita ao nó*/
    if (left(cons_fila(fila)))
      ins_fila (fila, left(cons_fila(fila)));
    if (right(cons_fila(fila)))
      ins_fila (fila, right(cons_fila (fila)));
    ret_fila(fila);
  }
}
```

Árvore binária de busca – Modos de travessia

Com base no que vimos sobre o percurso em largura, como você definiria o percurso em profundidade?



Árvore binária de busca – Modos de travessia

O percurso em profundidade consiste das seguintes operações básicas:

V – Visitar o nó;

L – percorrer a subárvore esquerda;

R – percorrer a subárvore direita.

Existem 3! modos de organizar tais operação, porém este conjunto é reduzido a três possibilidades que seriam:

VLR – cruzamento de árvore em pré-ordem (as vezes denominado em profundidade)

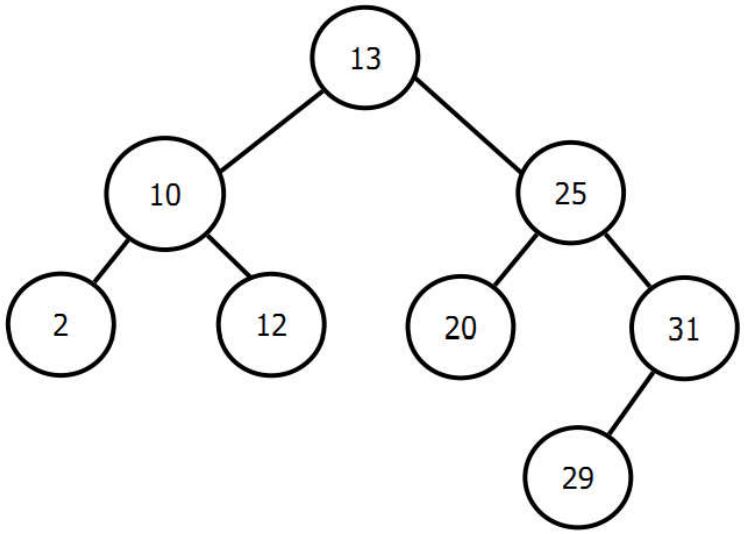
LVR – cruzamento de árvore em in-ordem (conhecido também como ordem simétrica)

LRV – cruzamento de árvore em pós-ordem

UNIVASF

Árvore binária de busca – Modos de travessia

Como ficaria o percurso da árvore abaixo em pré-ordem, in-ordem e pós-ordem?



Pré-ordem
13, 10, 2, 12, 25, 20, 31, 29

In-ordem
2, 10, 12, 13, 20, 25, 29, 31

Pós-ordem
2, 12, 10, 20, 29, 31, 25, 13

Estes percursos podem ser implementados com recursividade (pilha implícita), com uma pilha explicita implementada pelo programador ou utilizando o algoritmo idealizado por Joseph M. Morris.

Árvore binária de busca – Modos de travessia

Com base no que foi visto implemente, utilizando recursividade, as operações de percurso de uma árvore em pré-ordem, in-ordem e pós-ordem.