

Árvores Binárias - Alocação Sequencial

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

```
1  typedef struct {
2      int info;
3      int left;
4      int right;
5      int father;
6  } NODE;
7  typedef struct{
8      int root;
9      int nodeFree;
10     NODE nodes[NUMNODES]; /*#define NUMNODES 100*/
11 }ARV_BIN_SEQ;
12 void maketree(ARV_BIN_SEQ *, int);
13 void setleft(ARV_BIN_SEQ *, int, int);
14 void setright(ARV_BIN_SEQ *, int, int);
15 int info(ARV_BIN_SEQ *, int);
16 int left(ARV_BIN_SEQ *, int);
17 int right(ARV_BIN_SEQ *, int);
18 int father(ARV_BIN_SEQ *, int);
19 int brother(ARV_BIN_SEQ *, int);
20 int isleft(ARV_BIN_SEQ *, int);
21 int isright(ARV_BIN_SEQ *, int);
```

```
1 int brother(ARV_BIN_SEQ *t, int p) {
2     if (father(t, p) != -1) /*Se não for a raiz*/
3         if (isleft(t, p))
4             return right(t, father(t, p));
5         else
6             return t->nodes[t->nodes[p].father].left;
7     return -1;
8 }
```

Árvores Binárias - Alocação Sequencial

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

```
1  typedef struct {
2      int info;
3      int left;
4      int right;
5      int father;
6  } NODE;
7  typedef struct{
8      int root;
9      int nodeFree;
10     NODE nodes[NUMNODES]; /*#define NUMNODES 100*/
11 }ARV_BIN_SEQ;
12 void maketree(ARV_BIN_SEQ *, int);
13 void setleft(ARV_BIN_SEQ *, int, int);
14 void setright(ARV_BIN_SEQ *, int, int);
15 int info(ARV_BIN_SEQ *, int);
16 int left(ARV_BIN_SEQ *, int);
17 int right(ARV_BIN_SEQ *, int);
18 int father(ARV_BIN_SEQ *, int);
19 int brother(ARV_BIN_SEQ *, int);
20 int isleft(ARV_BIN_SEQ *, int);
21 int isright(ARV_BIN_SEQ *, int);
```

```
1  int isleft(ARV_BIN_SEQ *t, int p) {
2      int q = father(t, p);
3      if (q == -1) /*Se for a raiz*/
4          return (0);
5      if (left(t,q) == p)
6          return (1);
7      return (0);
8  }
```

Árvores Binárias - Alocação Sequencial

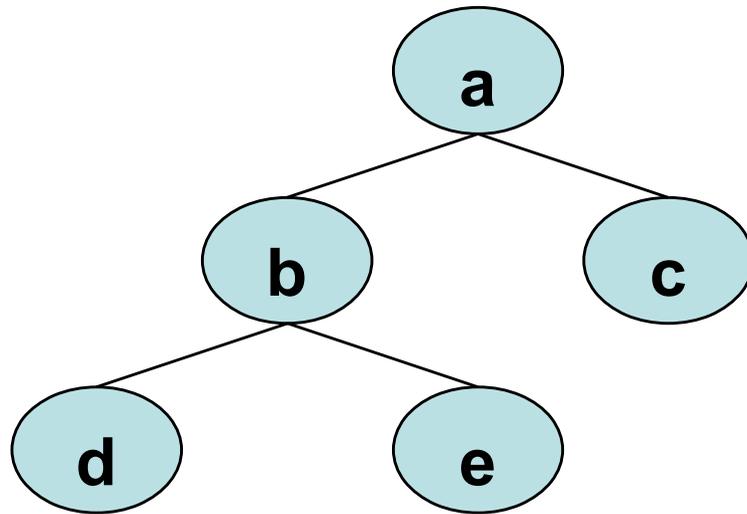
Com base no que foi visto implemente as operações que compõem o TAD ARV_BIN_SEQ.

```
1  typedef struct {
2      int info;
3      int left;
4      int right;
5      int father;
6  } NODE;
7  typedef struct{
8      int root;
9      int nodeFree;
10     NODE nodes[NUMNODES]; /*#define NUMNODES 100*/
11 }ARV_BIN_SEQ;
12 void maketree(ARV_BIN_SEQ *, int);
13 void setleft(ARV_BIN_SEQ *, int, int);
14 void setright(ARV_BIN_SEQ *, int, int);
15 int info(ARV_BIN_SEQ *, int);
16 int left(ARV_BIN_SEQ *, int);
17 int right(ARV_BIN_SEQ *, int);
18 int father(ARV_BIN_SEQ *, int);
19 int brother(ARV_BIN_SEQ *, int);
20 int isleft(ARV_BIN_SEQ *, int);
21 int isright(ARV_BIN_SEQ *, int);
```

```
1  int isright(ARV_BIN_SEQ *t, int p) {
2      if (father(t, p) != -1)
3          return (!isleft(t,p));
4      return (0);
5  }
```

Árvores Binárias - Alocação Sequencial

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

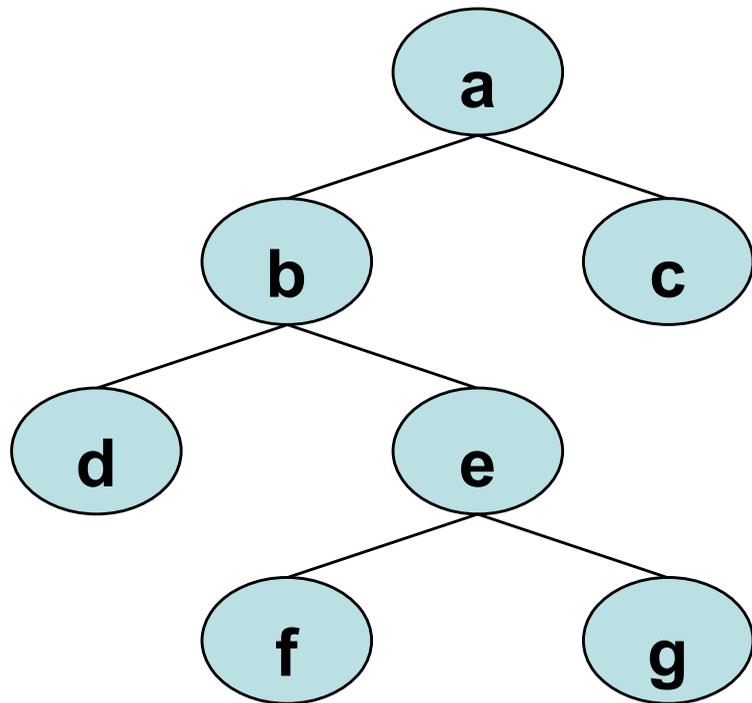


Esta Árvore poderia ser armazenada em um vetor da seguinte forma:



Árvores Binárias - Alocação Sequencial

Se a árvore anterior passasse a ter a seguinte configuração:



Como ficaria o armazenamento no vetor?
Assim?



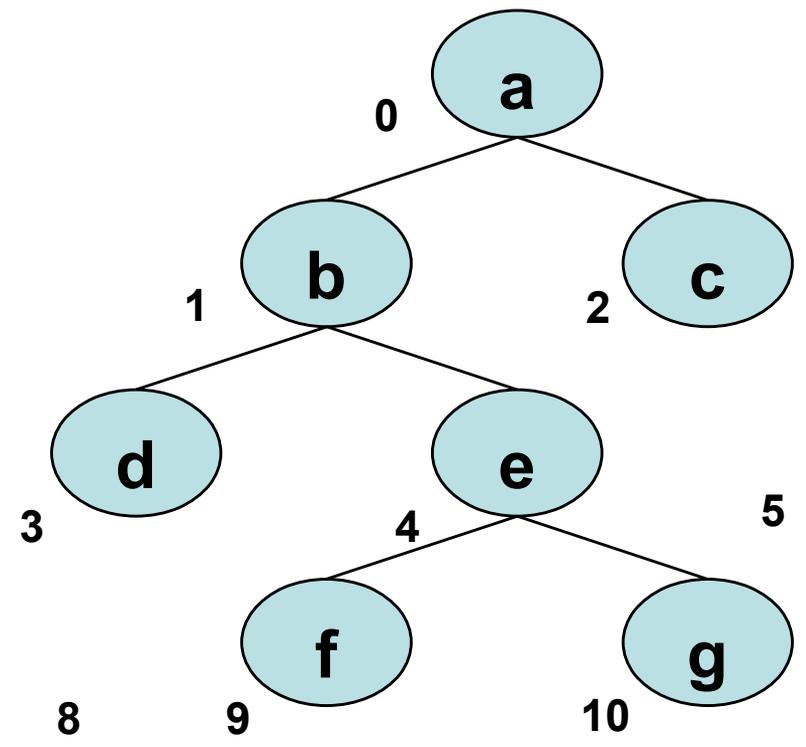
E se fosse assim:

a	b	c	d	e					f	g				
---	---	---	---	---	--	--	--	--	---	---	--	--	--	--

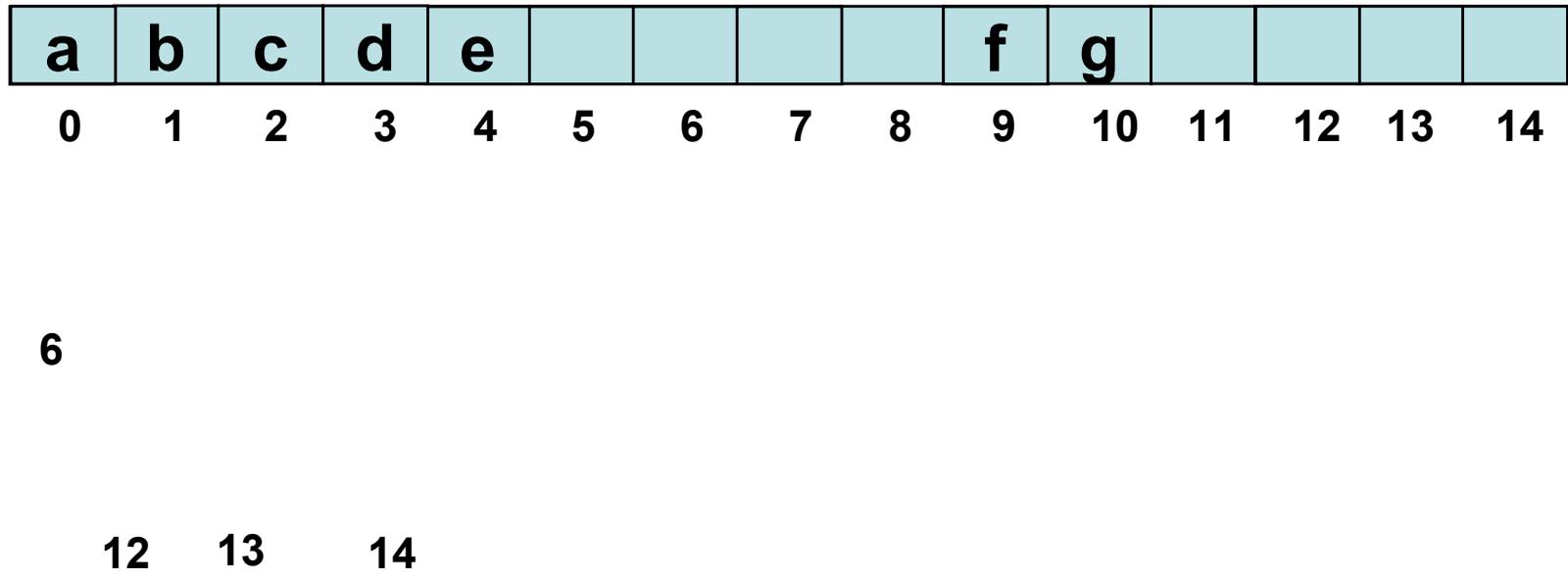
Vocês identificam alguma vantagem?

Árvores Binárias - Alocação Sequencial

Árvore:



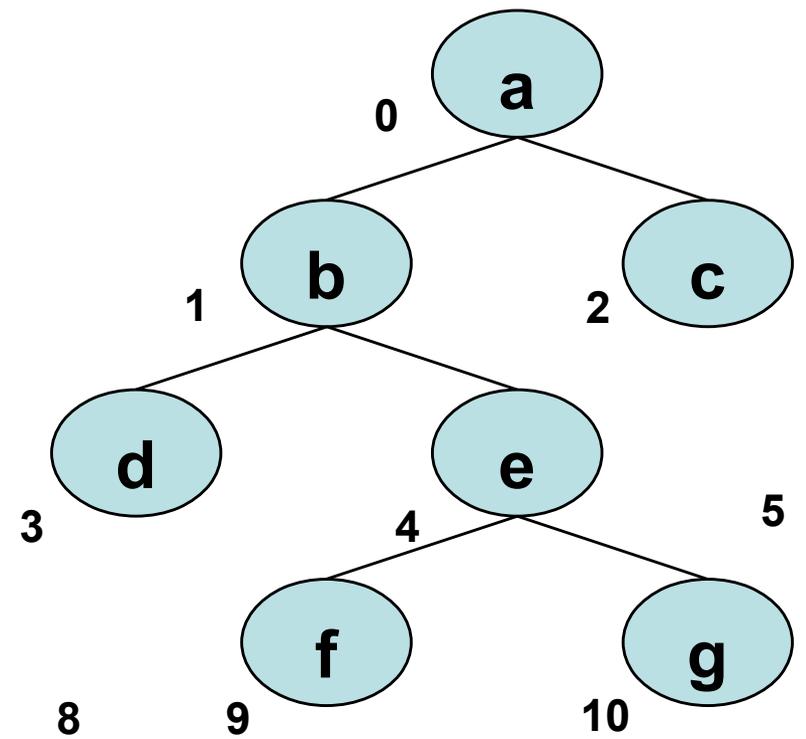
Representação:



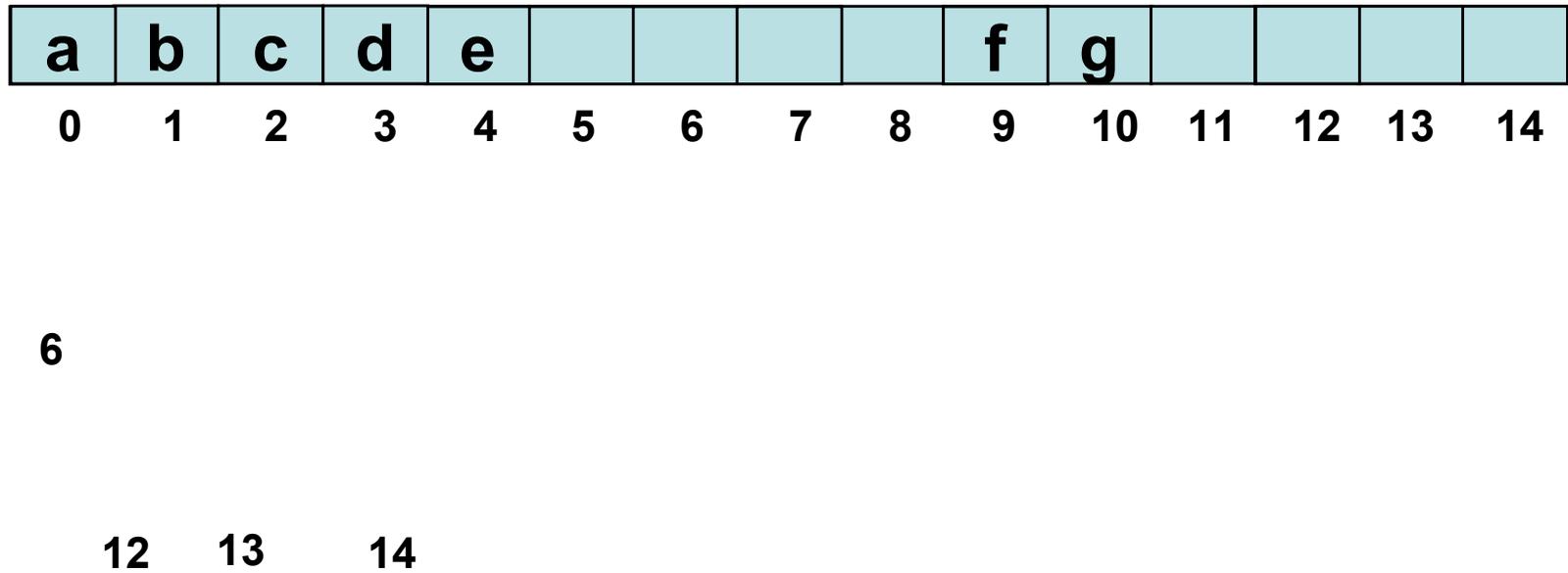
A raiz está aonde?
No índice 0 (zero).

Árvores Binárias - Alocação Sequencial

Árvore:



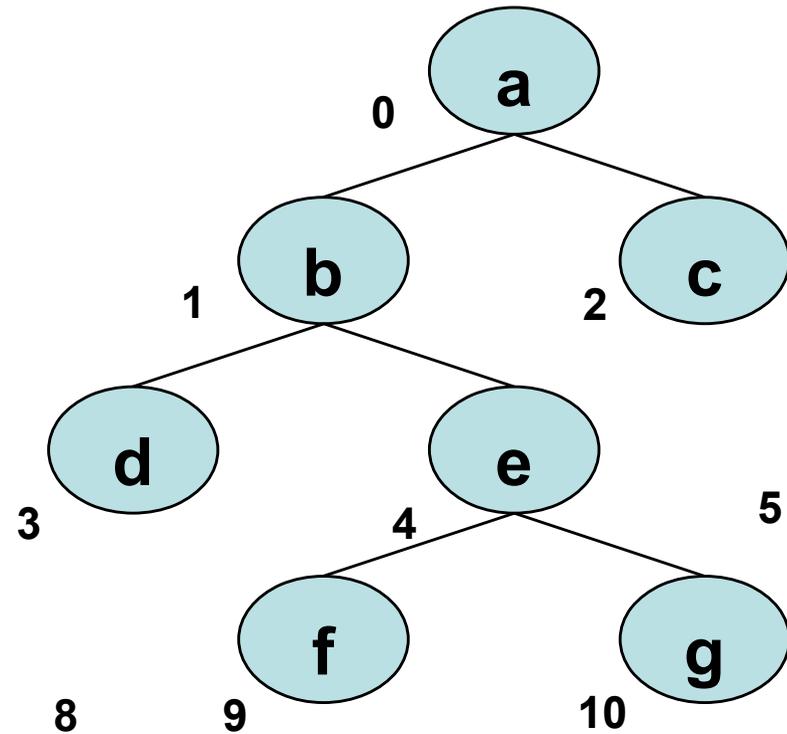
Representação:



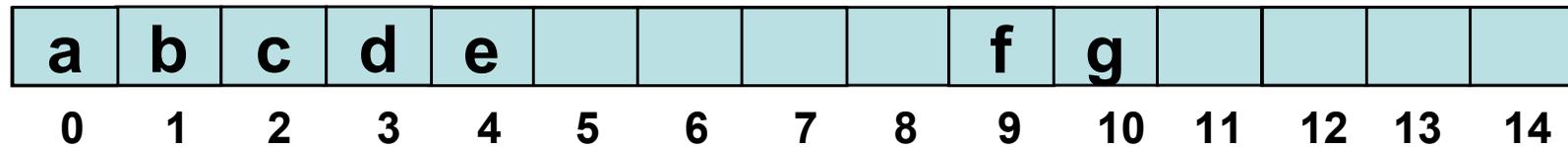
Como saber quem é o pai de um nó?
 $(p-1)/2$.
 p é o índice do vetor que contém o nó.

Árvores Binárias - Alocação Sequencial

Árvore:



Representação:

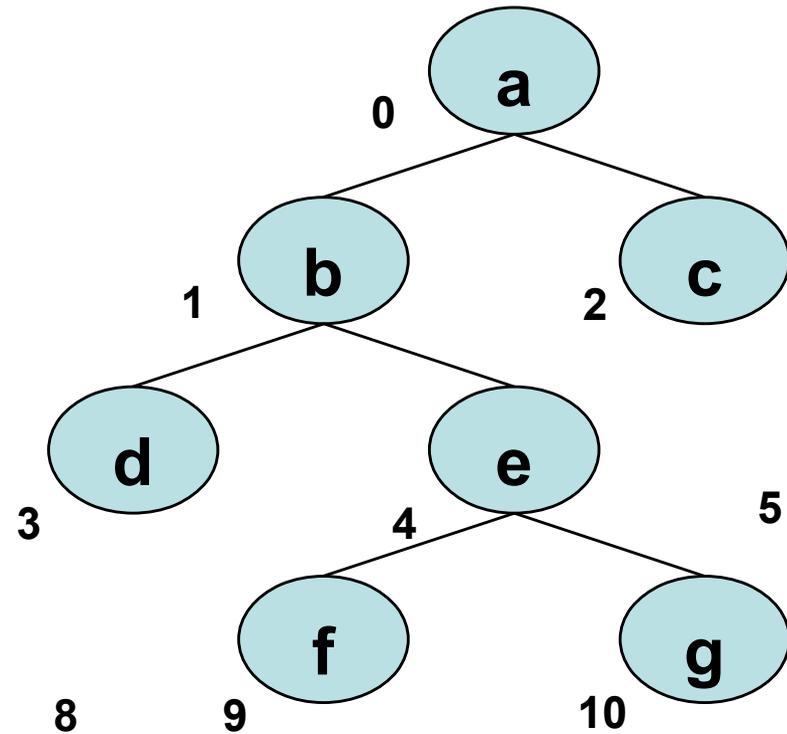


Como saber quem é o filho esquerdo de um nó?

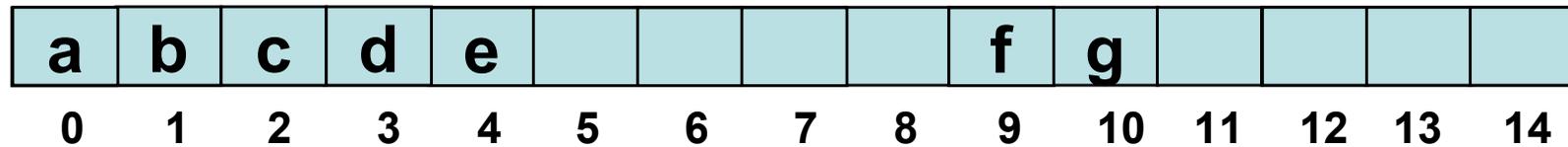
$$2^*p+1$$

Árvores Binárias - Alocação Sequencial

Árvore:



Representação:

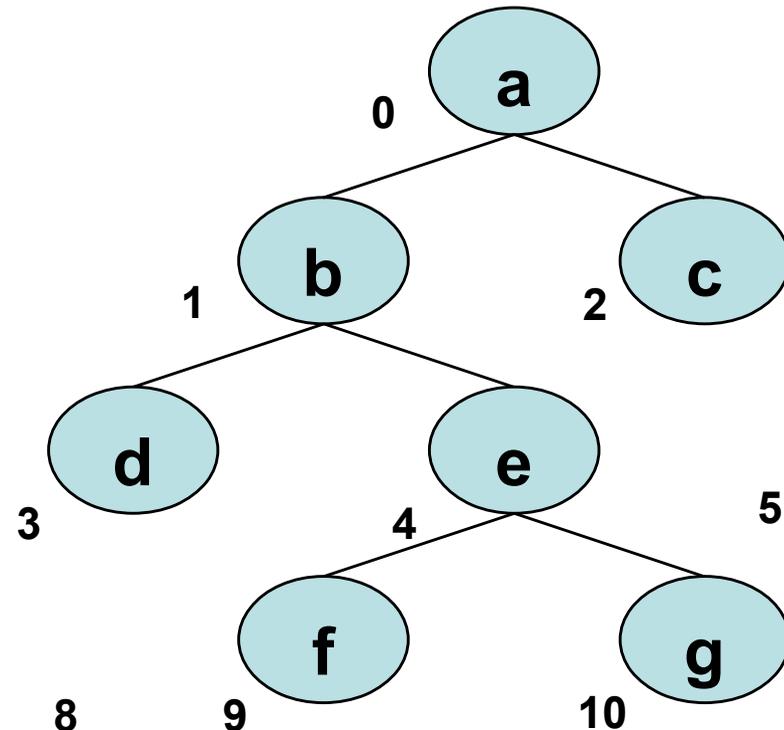


Como saber quem é o filho direito de um nó?

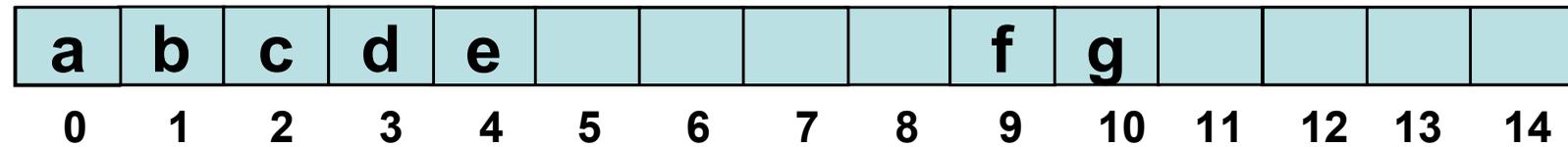
$$2 * p + 2$$

Árvores Binárias - Alocação Sequencial

Árvore:



Representação:



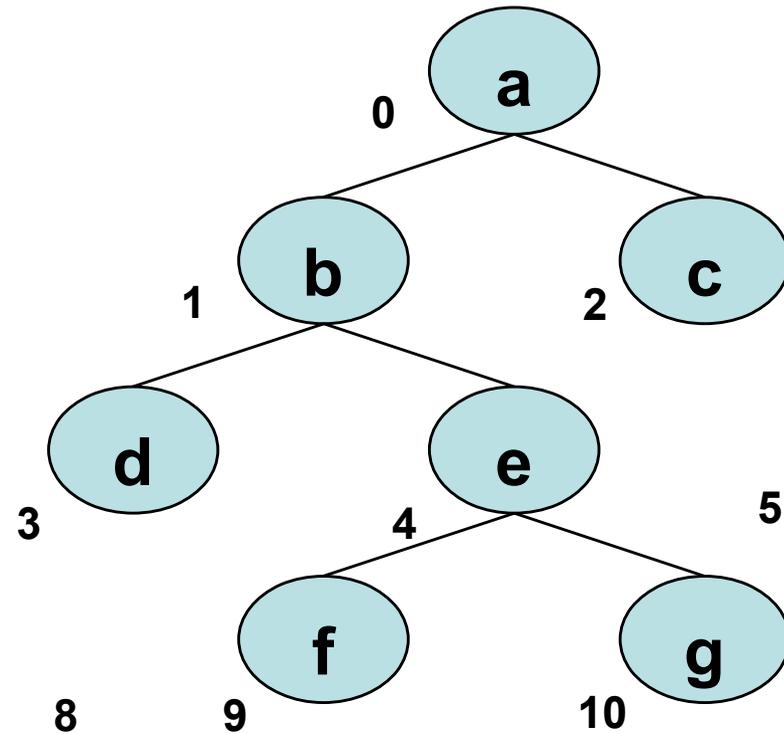
Como saber onde está o irmão direito de um nó?

Primeiro, devemos saber se ele é um filho esquerdo.

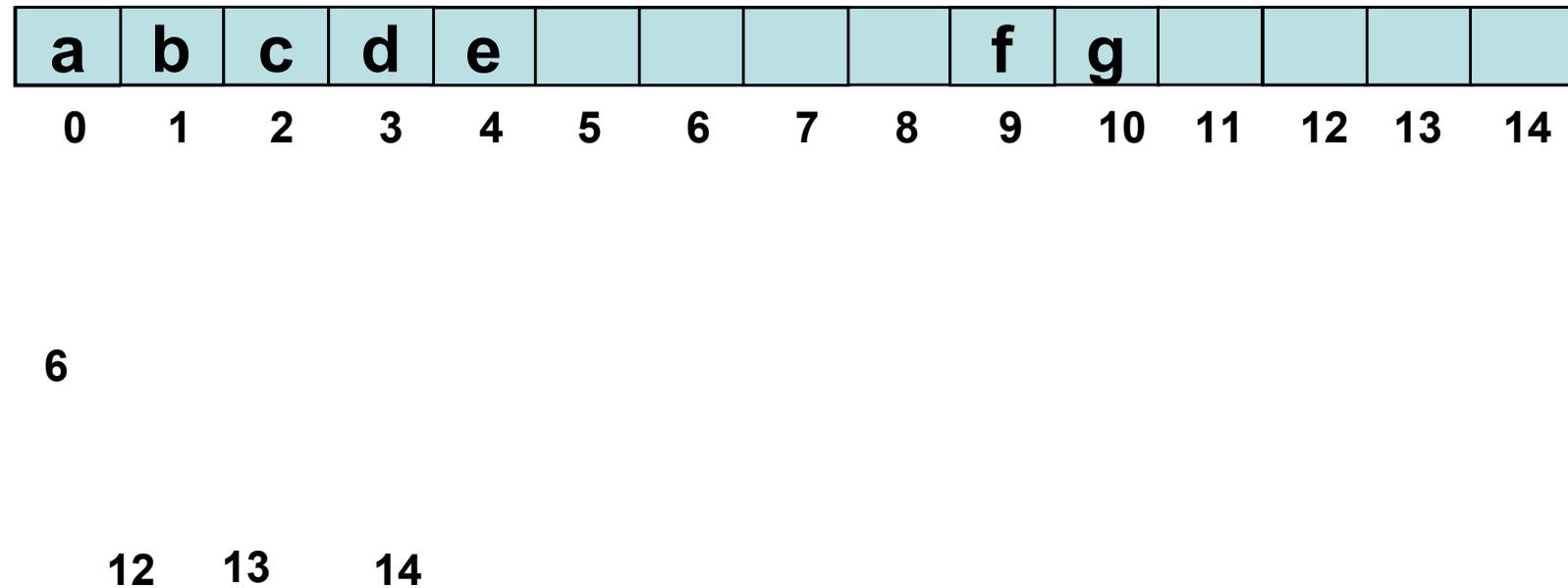
Como saber? É só verificar se p é ímpar.

Árvores Binárias - Alocação Sequencial

Árvore:



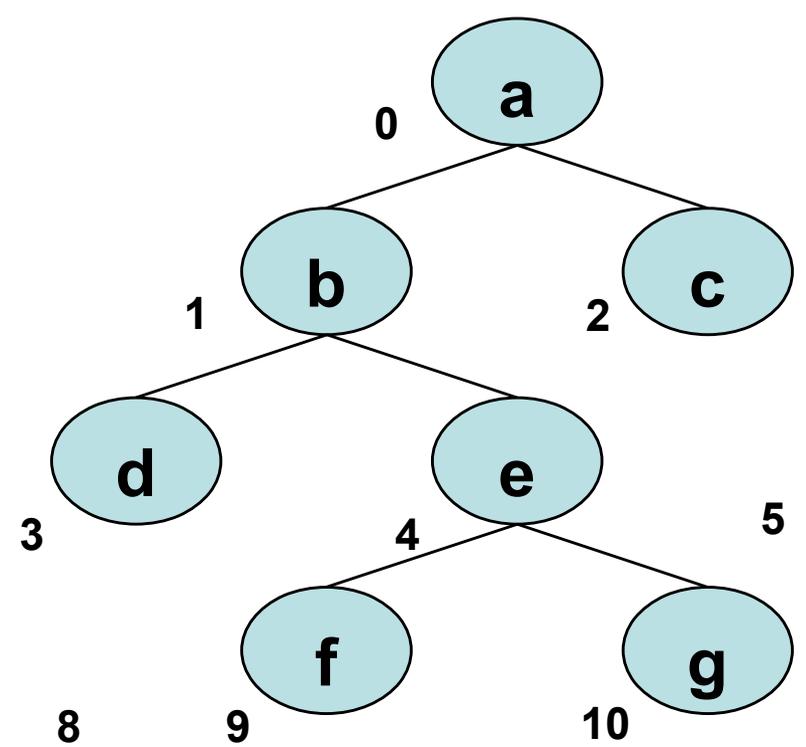
Representação:



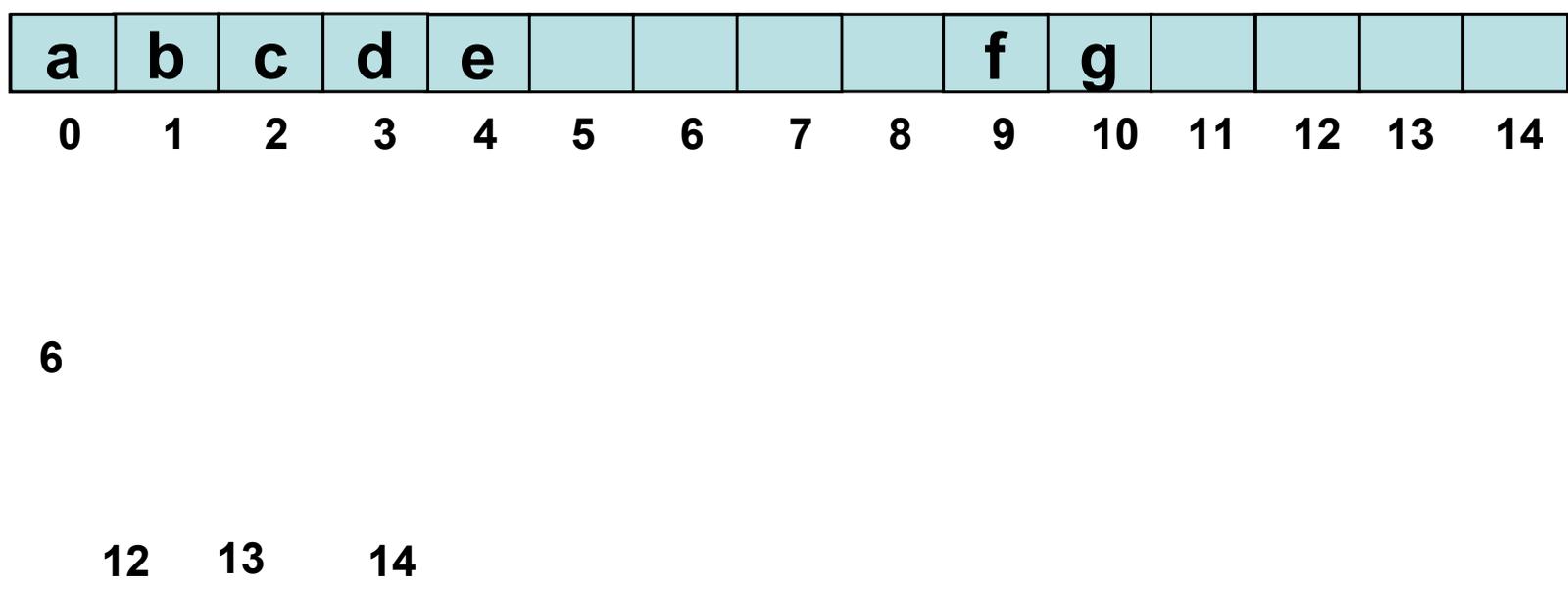
Se um nó for um filho a esquerda, para saber onde está o seu irmão direito basta somar uma unidade a p .

Árvores Binárias - Alocação Sequencial

Árvore:



Representação:

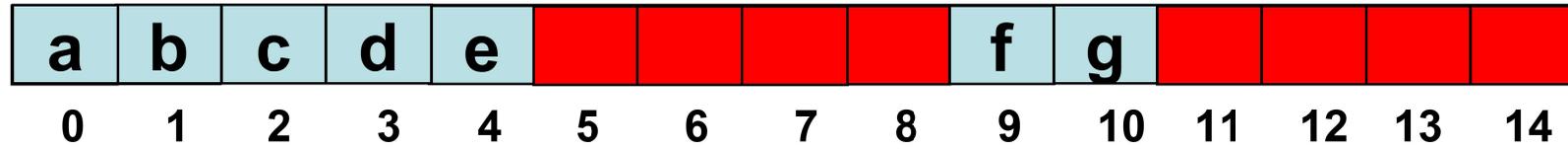


De forma análoga determinamos se um nó é um filho à direita e onde está o seu irmão esquerdo.

Árvores Binárias - Alocação Sequencial



Representação:



6

11 12 13 14

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 - Alocação Sequencial

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.

Árvore Binária

Alocação Encadeada

Exemplos de Aplicação

Árvores Binárias - Alocação Encadeada

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

Para tal, podemos definir a estrutura para os nós da seguinte forma:

```
1  typedef struct node
2  {
3      int info;
4      struct node *left;
5      struct node *right;
6      struct node *father;
7  } NODE;
```

```
1 typedef struct node
2 {
3     int info;
4     struct node *left;
5     struct node *right;
6     struct node *father;
7 } NODE;
8 typedef NODE * ARV_BIN_ENC;
9 void maketree(ARV_BIN_ENC *, int);
10 void setleft(ARV_BIN_ENC, int);
11 void setright(ARV_BIN_ENC, int);
12 int info(ARV_BIN_ENC);
13 ARV_BIN_ENC left(ARV_BIN_ENC);
14 ARV_BIN_ENC right(ARV_BIN_ENC);
15 ARV_BIN_ENC father(ARV_BIN_ENC);
16 ARV_BIN_ENC brother(ARV_BIN_ENC);
17 int isleft(ARV_BIN_ENC);
18 int isright(ARV_BIN_ENC);
```

Árvores Binárias - Alocação Encadeada

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

```
1  typedef struct node
2  {
3      int info;
4      struct node *left;
5      struct node *right;
6      struct node *father;
7  } NODE;
8  typedef NODE * ARV_BIN_ENC;
9  void maketree(ARV_BIN_ENC *, int);
10 void setleft(ARV_BIN_ENC, int);
11 void setright(ARV_BIN_ENC, int);
12 int info(ARV_BIN_ENC);
13 ARV_BIN_ENC left(ARV_BIN_ENC);
14 ARV_BIN_ENC right(ARV_BIN_ENC);
15 ARV_BIN_ENC father(ARV_BIN_ENC);
16 ARV_BIN_ENC brother(ARV_BIN_ENC);
17 int isleft(ARV_BIN_ENC);
18 int isright(ARV_BIN_ENC);
```

```
1 void maketree(ARV_BIN_ENC *t, int x)
2 {
3     *t = (ARV_BIN_ENC) malloc (sizeof (NODE));
4     if (!(*t))
5     {
6         printf("Erro! Nao existe memoria disponivel!");
7         exit (1);
8     }
9     (*t)->info = x;
10    (*t)->left = NULL;
11    (*t)->right = NULL;
12    (*t)->father = NULL;
13 }
```

Árvores Binárias - Alocação Encadeada

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

```
1  typedef struct node
2  {
3      int info;
4      struct node *left;
5      struct node *right;
6      struct node *father;
7  } NODE;
8  typedef NODE * ARV_BIN_ENC;
9  void maketree(ARV_BIN_ENC *, int);
10 void setleft(ARV_BIN_ENC, int);
11 void setright(ARV_BIN_ENC, int);
12 int info(ARV_BIN_ENC);
13 ARV_BIN_ENC left(ARV_BIN_ENC);
14 ARV_BIN_ENC right(ARV_BIN_ENC);
15 ARV_BIN_ENC father(ARV_BIN_ENC);
16 ARV_BIN_ENC brother(ARV_BIN_ENC);
17 int isleft(ARV_BIN_ENC);
18 int isright(ARV_BIN_ENC);
```

```
1 void setleft(ARV_BIN_ENC t, int x)
2 {
3     t->left = (ARV_BIN_ENC) malloc (sizeof (NODE));
4     if (!(t->left))
5     {
6         printf("Erro! Nao existe memoria disponivel!");
7         exit (1);
8     }
9     t->left->info = x;
10    t->left->left = NULL;
11    t->left->right = NULL;
12    t->left->father = t;
13 }
```

Árvores Binárias - Alocação Encadeada

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

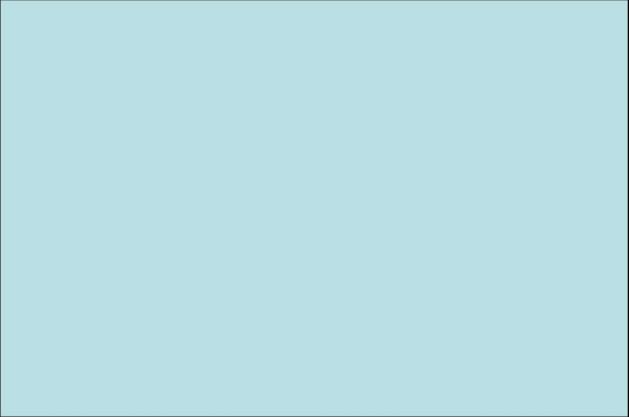
```
1  typedef struct node
2  {
3      int info;
4      struct node *left;
5      struct node *right;
6      struct node *father;
7  } NODE;
8  typedef NODE * ARV_BIN_ENC;
9  void maketree(ARV_BIN_ENC *, int);
10 void setleft(ARV_BIN_ENC, int);
11 void setright(ARV_BIN_ENC, int);
12 int info(ARV_BIN_ENC);
13 ARV_BIN_ENC left(ARV_BIN_ENC);
14 ARV_BIN_ENC right(ARV_BIN_ENC);
15 ARV_BIN_ENC father(ARV_BIN_ENC);
16 ARV_BIN_ENC brother(ARV_BIN_ENC);
17 int isleft(ARV_BIN_ENC);
18 int isright(ARV_BIN_ENC);
```

```
1 void setright(ARV_BIN_ENC t, int x)
2 {
3     t->right = (ARV_BIN_ENC) malloc (sizeof (NODE));
4     if (!(t->right))
5     {
6         printf("Erro! Nao existe memoria disponivel!");
7         exit (1);
8     }
9     t->right->info = x;
10    t->right->left = NULL;
11    t->right->right = NULL;
12    t->right->father = t;
13 }
```

Árvores Binárias - Alocação Encadeada

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

```
1  typedef struct node
2  {
3      int info;
4      struct node *left;
5      struct node *right;
6      struct node *father;
7  } NODE;
8  typedef NODE * ARV_BIN_ENC;
9  void maketree(ARV_BIN_ENC *, int);
10 void setleft(ARV_BIN_ENC, int);
11 void setright(ARV_BIN_ENC, int);
12 int info(ARV_BIN_ENC);
13 ARV_BIN_ENC left(ARV_BIN_ENC);
14 ARV_BIN_ENC right(ARV_BIN_ENC);
15 ARV_BIN_ENC father(ARV_BIN_ENC);
16 ARV_BIN_ENC brother(ARV_BIN_ENC);
17 int isleft(ARV_BIN_ENC);
18 int isright(ARV_BIN_ENC);
```



```
1  int info(ARV_BIN_ENC t)
2  {
3      return t->info; /*Se a arvore for vazia?*/
4  }
```

Árvores Binárias - Alocação Encadeada

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

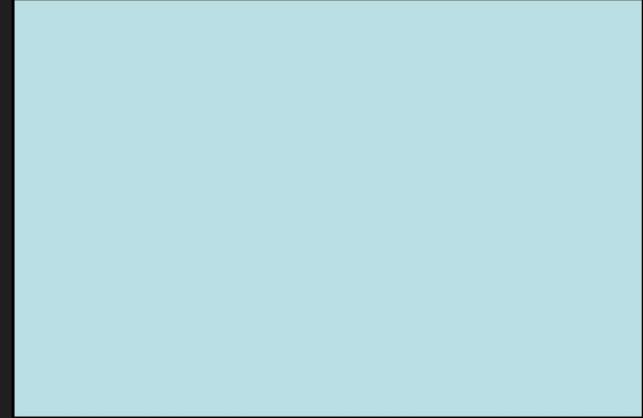
```
1  typedef struct node
2  {
3      int info;
4      struct node *left;
5      struct node *right;
6      struct node *father;
7  } NODE;
8  typedef NODE * ARV_BIN_ENC;
9  void maketree(ARV_BIN_ENC *, int);
10 void setleft(ARV_BIN_ENC, int);
11 void setright(ARV_BIN_ENC, int);
12 int info(ARV_BIN_ENC);
13 ARV_BIN_ENC left(ARV_BIN_ENC);
14 ARV_BIN_ENC right(ARV_BIN_ENC);
15 ARV_BIN_ENC father(ARV_BIN_ENC);
16 ARV_BIN_ENC brother(ARV_BIN_ENC);
17 int isleft(ARV_BIN_ENC);
18 int isright(ARV_BIN_ENC);
```

```
1  ARV_BIN_ENC left(ARV_BIN_ENC t)
2  {
3      return t->left;
4  }
```

Árvores Binárias - Alocação Encadeada

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

```
1  typedef struct node
2  {
3      int info;
4      struct node *left;
5      struct node *right;
6      struct node *father;
7  } NODE;
8  typedef NODE * ARV_BIN_ENC;
9  void maketree(ARV_BIN_ENC *, int);
10 void setleft(ARV_BIN_ENC, int);
11 void setright(ARV_BIN_ENC, int);
12 int info(ARV_BIN_ENC);
13 ARV_BIN_ENC left(ARV_BIN_ENC);
14 ARV_BIN_ENC right(ARV_BIN_ENC);
15 ARV_BIN_ENC father(ARV_BIN_ENC);
16 ARV_BIN_ENC brother(ARV_BIN_ENC);
17 int isleft(ARV_BIN_ENC);
18 int isright(ARV_BIN_ENC);
```



```
1  ARV_BIN_ENC right(ARV_BIN_ENC t)
2  {
3      return t->right;
4  }
```

Árvores Binárias - Alocação Encadeada

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

```
1  typedef struct node
2  {
3      int info;
4      struct node *left;
5      struct node *right;
6      struct node *father;
7  } NODE;
8  typedef NODE * ARV_BIN_ENC;
9  void maketree(ARV_BIN_ENC *, int);
10 void setleft(ARV_BIN_ENC, int);
11 void setright(ARV_BIN_ENC, int);
12 int info(ARV_BIN_ENC);
13 ARV_BIN_ENC left(ARV_BIN_ENC);
14 ARV_BIN_ENC right(ARV_BIN_ENC);
15 ARV_BIN_ENC father(ARV_BIN_ENC);
16 ARV_BIN_ENC brother(ARV_BIN_ENC);
17 int isleft(ARV_BIN_ENC);
18 int isright(ARV_BIN_ENC);
```



```
1  ARV_BIN_ENC father(ARV_BIN_ENC t)
2  {
3      return t->father;
4  }
```

Árvores Binárias - Alocação Encadeada

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

```
1  typedef struct node
2  {
3      int info;
4      struct node *left;
5      struct node *right;
6      struct node *father;
7  } NODE;
8  typedef NODE * ARV_BIN_ENC;
9  void maketree(ARV_BIN_ENC *, int);
10 void setleft(ARV_BIN_ENC, int);
11 void setright(ARV_BIN_ENC, int);
12 int info(ARV_BIN_ENC);
13 ARV_BIN_ENC left(ARV_BIN_ENC);
14 ARV_BIN_ENC right(ARV_BIN_ENC);
15 ARV_BIN_ENC father(ARV_BIN_ENC);
16 ARV_BIN_ENC brother(ARV_BIN_ENC);
17 int isleft(ARV_BIN_ENC);
18 int isright(ARV_BIN_ENC);
```

```
1  ARV_BIN_ENC brother(ARV_BIN_ENC t)
2  {
3      if (t->father)
4          if (isleft(t))
5              return (t->father)->right;
6          else
7              return (t->father)->left;
8      return NULL;
9  }
```

Árvores Binárias - Alocação Encadeada

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

```
1  typedef struct node
2  {
3      int info;
4      struct node *left;
5      struct node *right;
6      struct node *father;
7  } NODE;
8  typedef NODE * ARV_BIN_ENC;
9  void maketree(ARV_BIN_ENC *, int);
10 void setleft(ARV_BIN_ENC, int);
11 void setright(ARV_BIN_ENC, int);
12 int info(ARV_BIN_ENC);
13 ARV_BIN_ENC left(ARV_BIN_ENC);
14 ARV_BIN_ENC right(ARV_BIN_ENC);
15 ARV_BIN_ENC father(ARV_BIN_ENC);
16 ARV_BIN_ENC brother(ARV_BIN_ENC);
17 int isleft(ARV_BIN_ENC);
18 int isright(ARV_BIN_ENC);
```

```
1  int isleft(ARV_BIN_ENC t)
2  {
3      NODE *q = t->father;
4      if (!q)
5          return (0);
6      if (q->left == t)
7          return (1);
8      return (0);
9  }
```

Árvores Binárias - Alocação Encadeada

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

```
1  typedef struct node
2  {
3      int info;
4      struct node *left;
5      struct node *right;
6      struct node *father;
7  } NODE;
8  typedef NODE * ARV_BIN_ENC;
9  void maketree(ARV_BIN_ENC *, int);
10 void setleft(ARV_BIN_ENC, int);
11 void setright(ARV_BIN_ENC, int);
12 int info(ARV_BIN_ENC);
13 ARV_BIN_ENC left(ARV_BIN_ENC);
14 ARV_BIN_ENC right(ARV_BIN_ENC);
15 ARV_BIN_ENC father(ARV_BIN_ENC);
16 ARV_BIN_ENC brother(ARV_BIN_ENC);
17 int isleft(ARV_BIN_ENC);
18 int isright(ARV_BIN_ENC);
```

```
1  int isright(ARV_BIN_ENC t)
2  {
3      if (t->father)
4          return (!isleft(t));
5      return (0);
6  }
```

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.

Árvores Binárias – Aplicações

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.

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.

Árvores Binárias – Aplicações

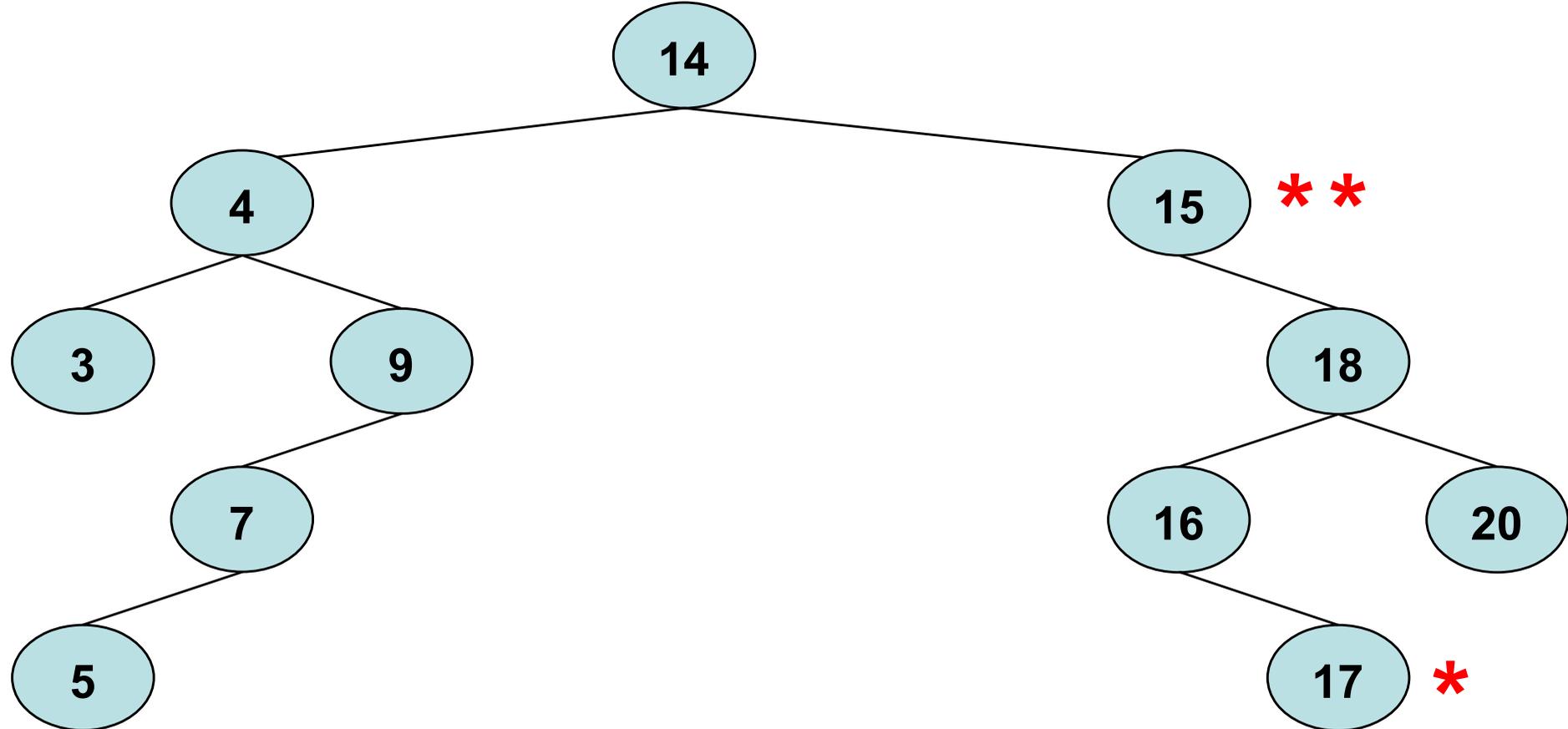
Se a subárvore estiver vazia, o número não será repetido e será colocado num novo nó nesta posição na árvore.

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

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.

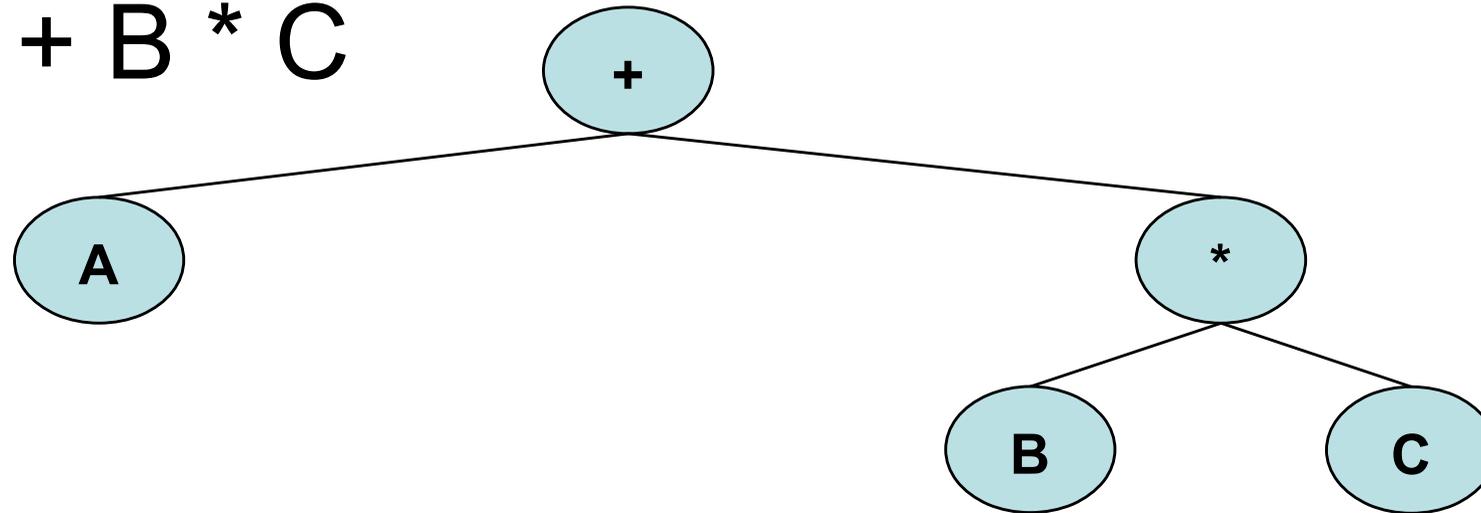
Onde a raiz da árvore estritamente binária conterá um operador que deve ser aplicado aos resultados das avaliações das expressões representadas pelas subárvores esquerda e direita.

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

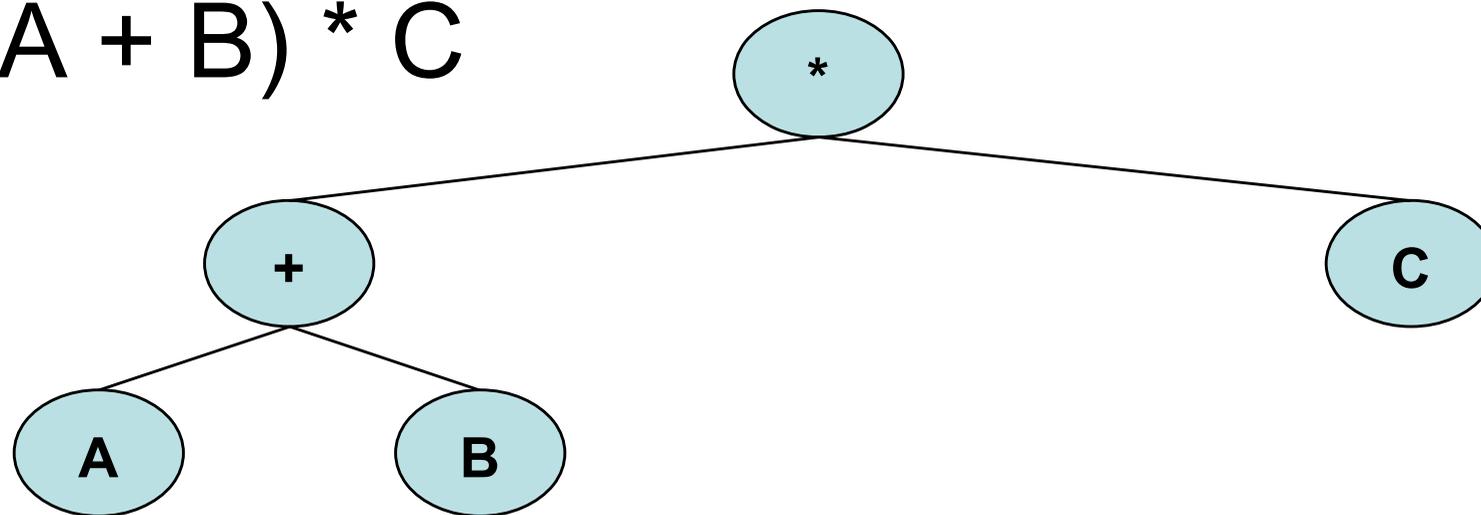
Árvores Binárias – Aplicações

Vejam os alguns exemplos:

$$A + B * C$$

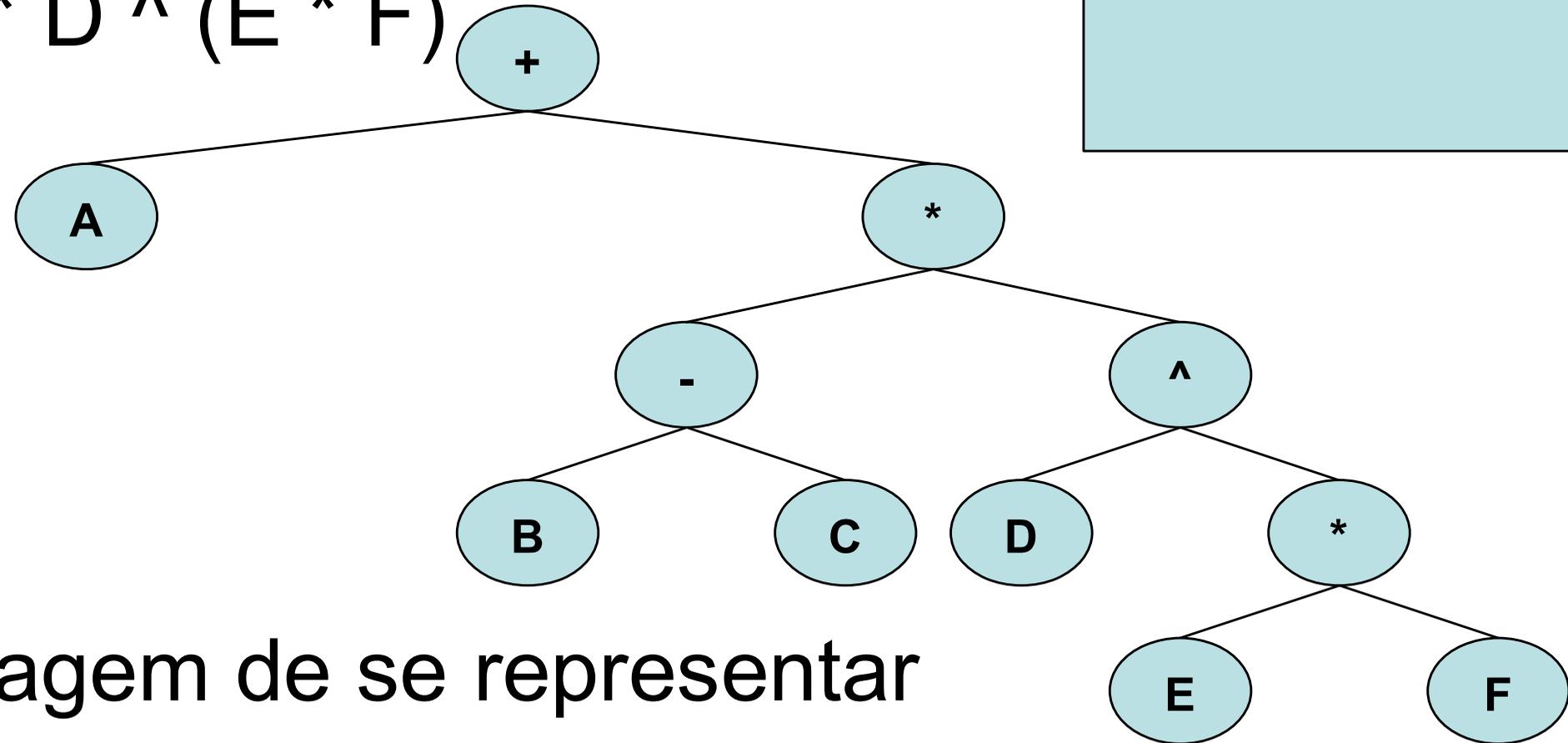


$$(A + B) * C$$



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

Definição, Percurso

Inserção e Remoção de um Nó

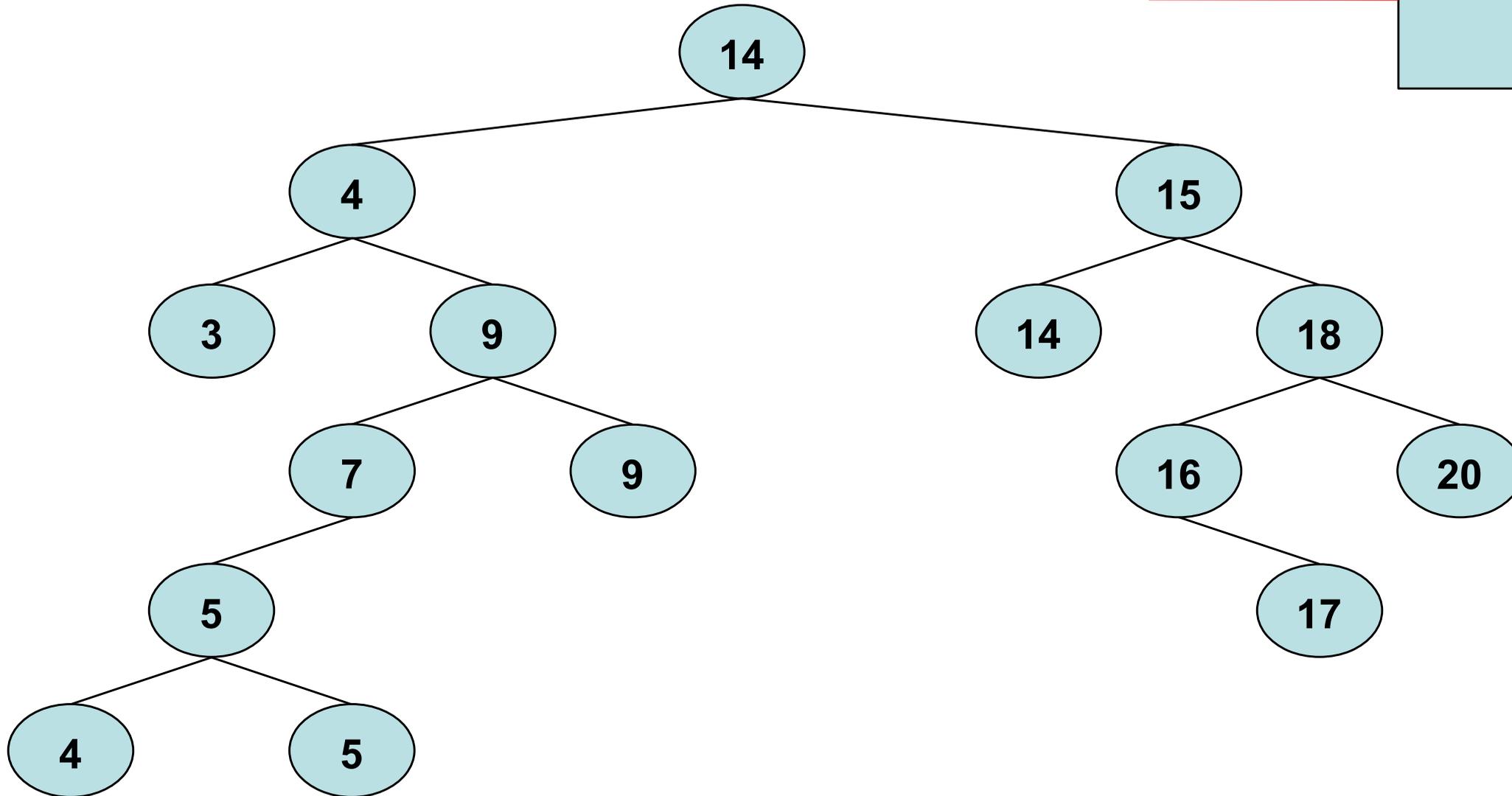
Árvore Binária de Busca

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 .

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.

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 como...

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.

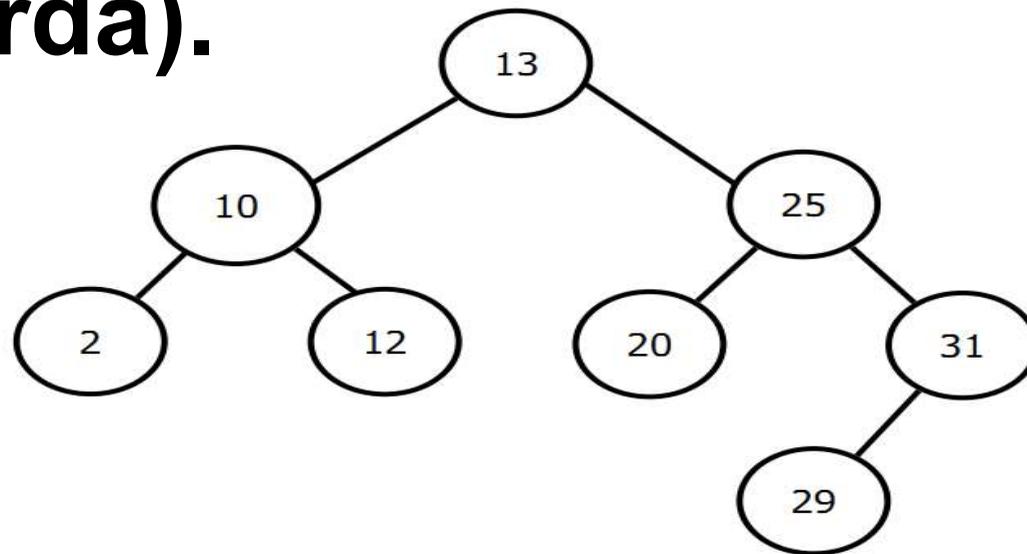
Diante de tamanha quantidade de opções, iremos restringir este universo a duas classes de percursos.

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**

Árvore Binária de Busca – Modos de Travessia

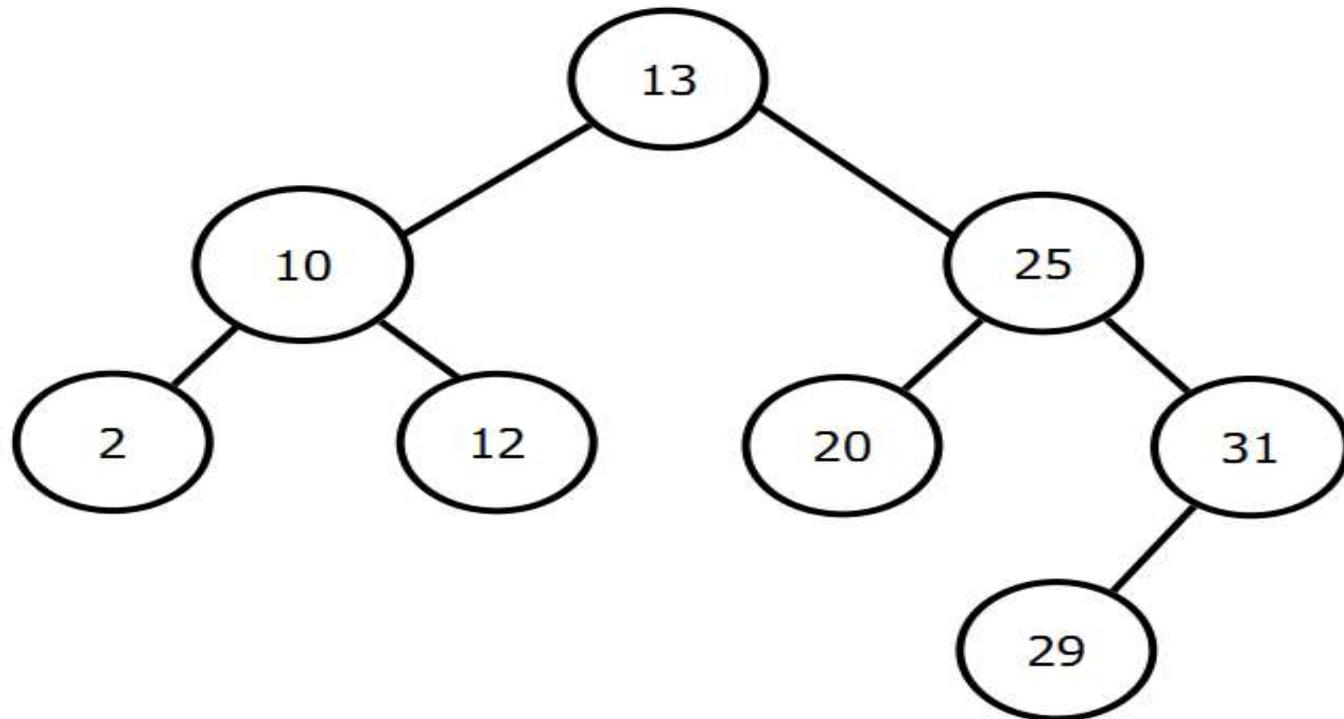
Exercício:

Considerando a existência do TAD `ARV_BIN_ENC` (implementado anteriormente), implemente o percurso em largura.

```
1 void percursoEmLargura(ARV_BIN_ENC arvore)
2 {
3     FILA_ENC fila; /*FILA_ENC armazenada ARV_BIN_ENCs*/
4     cria_fila(&fila);
5     if (arvore)
6         ins_fila (fila, arvore);
7     while (!eh_vazia_fila(fila))
8     {
9         printf("%d ", info(cons_fila(fila))); /*Visita ao nó*/
10        if (left(cons_fila(fila)))
11            ins_fila (fila, left(cons_fila(fila)));
12        if (right(cons_fila(fila)))
13            ins_fila (fila, right(cons_fila (fila)));
14        ret_fila(fila);
15    }
16 }
```

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:

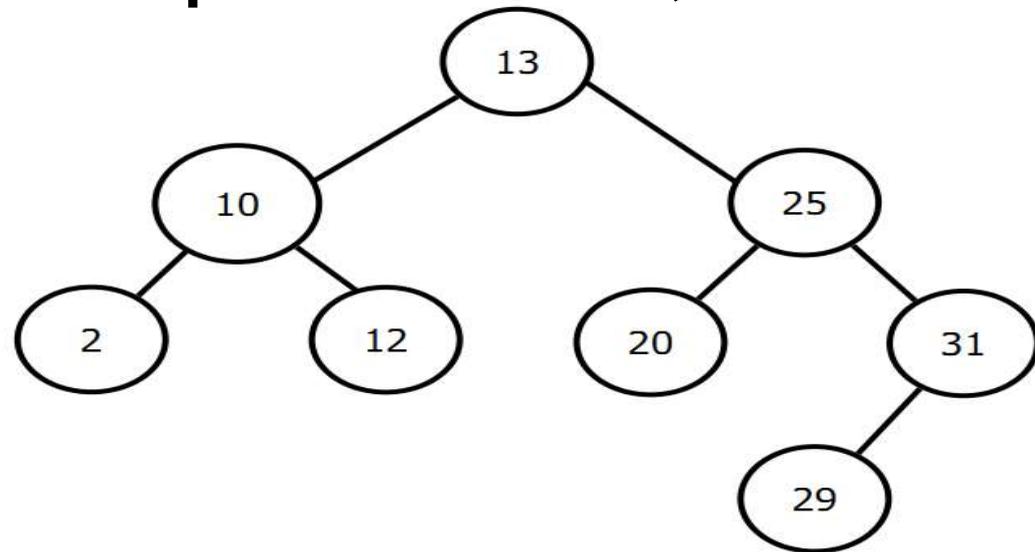
VLR – cruzamento de árvore em pré-ordem (as vezes denominado em profundidade)

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

LRV – cruzamento de árvore em pós-ordem

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 explícita (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.